UNIVERSITY OF BAHRAIN

College of Science

Department of
Postgraduate Programs

Effective Systems for Bot Detection and Real-Time Stock Market Predictions

Thesis submitted in partial fulfilment of the requirements for the degree of M.Sc. In Big Data Science and Analytics

Prepared by

Shaikha Reem Abdulla Salman Alkhalifa

University Number: 202012269

Supervised by

Prof. Mustafa Aytekin
Professor
University of Bahrain

Dr. Riadh bin Mohammad Ksantini
Associate Professor
University of Bahrain

Kingdom of Bahrain

June 2022

UNIVERSITY OF BAHRAIN

College of Science



Department of Postgraduate Programs

Effective Systems for Bot Detection and Real-Time Stock Market Predictions

Thesis submitted in partial fulfilment of the requirements for the degree of M.Sc. In Big Data Science and Analytics

Prepared by

Shaikha Reem Abdulla Salman Alkhalifa

University Number: 202012269

Supervised by

Prof. Mustafa Aytekin
Professor
University of Bahrain

Dr. Riadh bin Mohammad Ksantini
Associate Professor
University of Bahrain

Kingdom of Bahrain

June 2022

University of Bahrain
Deanship of Graduate Studies
And scientific Research

جامعة البحرين
عمادة الدراسات العليا والبحث العلمي

Thesis Discussion Committee Decision

Student Name: Shaikha Reem Abdulla Salman Alkhalifa	Student ID: 202012269
Department: Choose an item.	College: **Science**
Program: M.Sc. In Big Data Science and Analytics	
Thesis Title: **Effective Systems for Bot Detection and Real-Time Stock Market Predictions**	
Supervisor Name: **Dr. Riadh Ksantini and Pr Hatem Masri**	Student GPA: **3.33**

The Discussion Committee met on 25-May-22 In College of Science. Upon discussing the student whose details are stated above in public, the discussion committee recommends the following:

The student is awarded a grade A in the thesis subject.

☐ Without making any changes to the thesis.

☒ After making the modifications attached with this decision.

Discussion Committee Members Approval:

Name		Signature
1. Dr. Riadh Ksantini and Pr. Hatem Masri enter text.	(Advisor)	Click here to
2. Dr. Wael Elmedany	(Internal Examiner)	
3. Pr. Brij B. Gupta	(External Examiner)	

Approved at the University Council 6th meeting, 2013
In accordance with Decision No. 736/2013 dated 14/3/2013
Modified by University Council Decision No. 2043/2013 dated 20/11/2013.

Abstract

Social media has reshaped the way in which individuals communicate with disruptive negative and positive externalities, alongside being breeding grounds for bot activities. Social media, limited to Twitter in this research, is rarely monitored and has accelerated the dissemination of disinformation, most of which is attributed to bot activities. To date, bot detection techniques are challenged by their inability to keep abreast with bot progression, while its importance is increasingly being bought to the forefront of political discussions. Therefore, the research will introduce a local-global SVM (LG-SVM) model to increase flexibility in detection methods to keep abreast with the ever-evolving world of Twitter. The end-to-end framework has increased accuracy in detection by treating the data as unsupervised and introducing an autocoder to reduce dimensionality, which has solved for over-fitting. The multi-level framework starts with a convolutional layer that reduces dimensionality and learns key short-term characteristics, followed by a max pooling layer, two bi-LSTM models, a bottleneck layer, an upscaling layer, and a deconvolution layer. Dimension reduction oversaw the reduction of the dataset into 3 lengths: 70, 100, and 130, resulting in a 99% accuracy rate. Additionally, a multilayer perceptron model is built to predict the impact Bot Tweets have on stock market volatility with a 96.53% coefficient correlation.

Acknowledgements

This research is supported by the University of Bahrain as part of the Big Data Analytics Science and Analytics Masters program. This is a letter of acknowledgment to all support extended by Dr. Riadh bin Mohammad Ksantini and Professor Hatem Masri. Thanks also to Luis-Daniel for providing a comprehensive dataset.

Table of Contents

Defense Committee's Approval ..... A

Abstract ..... B

الملخص ..... - 142 -

Acknowledgements C

Table of Contents ..... D

List of Tables ..... H

List of Figures I

List of Abbreviations ..... A

Chapter One Introduction to the Study 2

1.1 Introduction 2

1.2 Research Problem and Question 2

1.3 Research Objectives 2

1.4 Research Significance 2

Chapter One Introduction to the Study 3

1.1 Introduction 3

1.2 Research Problem and Question 4

1.3 Research Objectives 5

1.4 Research Significance 6

Chapter Two Literature Review 8

2.1 Literature Review 8

2.1.1 Bot Detection Literature Review 8

2.1.2 Stock Market Literature Review 8

2.2 Theoretical Framework....8
2.2.1 Data Extraction....8
2.2.2 Fitting the data and validating the model....8
2.3 Research Hypothesis....8
2.2 Literature Review....9
2.2.1 Bot Detection Literature Review....9
2.2.2 Stock Market Literature Review....11
2.3 Theoretical Framework....13
2.3.1 Data extraction....14
2.3.2 Fitting the data and validating the model....14
2.4 Research Hypothesis....15
Chapter Three Methods and Procedures....- 16 -
3.1 Data Collection and Procedures....- 16 -
3.1.1Data Description and Visualization....- 16 -
3.2 Model....- 16 -
3.2.1 Convolutional and Max Pooling Layer....- 16 -
3.2.2 Bi-LSTM....- 16 -
3.2.3 LG-SVM....- 16 -
3.2.4 Multilayer Perceptron....- 16 -
3.2.5 Autoencoder....- 16 -
3.3 Statistical Approach....- 16 -
3.3.1 Convolutional and Max Pooling Layer....- 16 -
3.3.2 Bi-LSTM....- 16 -
3.3.3 LG-SVM....- 16 -

3.3.4 Autoencoder ..... - 16 -
3.3.5 Multilayer Perceptron ..... - 16 -
3.1 Data Collection and Procedures ..... - 17 -
3.1.1 Data Description and Visualization ..... - 18 -
3.2 Model ..... - 22 -
3.2.1 Convolutional and Max Pooling Layer ..... - 23 -
3.2.3 Bi-LSTM Architecture ..... - 23 -
3.2.4 LG-SVM ..... - 25 -
3.2.5 Multilayer Perceptron ..... - 25 -
3.2.6 Autoencoder ..... - 26 -
3.3 Statistical Approach ..... - 29 -
3.3.1 Convolution and Max Pooling Layer ..... - 29 -
3.3.2 Bi-LSTM ..... - 29 -
3.3.3 LG-SVM ..... - 30 -
3.3.4 Autoencoder ..... - 39 -
3.3.5 Multilayer Perceptron ..... - 39 -
Chapter Four Results, Discussion, Conclusion and Recommendations ..... - 43 -
4.1 Results ..... - 43 -
4.1.1 Raw Data ..... - 43 -
4.1.2 Data Transformation ..... - 43 -
4.1.3 Stock Market ..... - 43 -
4.2 Discussion ..... - 43 -
4.3 Conclusion and Recommendations ..... - 43 -

4.1 Results ..... - 44 -

4.1.1 Raw Data ..... - 44 -

4.1.2 Data transformation ..... - 46 -

4.1.3 Stock Market ..... - 48 -

4.2 Discussion ..... - 49 -

4.3 Conclusion and Recommendations ..... - 52 -

References ..... - 54 -

5.0 Appendix I: Code ..... - 61 -

5.0.1 Data exploration and Bi-LSTM (google Collaboratory). - 61 -

5.0.2 Classification M. (Matlab) ..... - 69 -

5.0.3 IDSVM (Matlab) ..... - 90 -

5.0.4 IKNSVM (Matlab) ..... - 91 -

5.0.5 IKN Train (Matlab) ..... - 95 -

5.0.6 Incremental Train (Matlab) ..... - 111 -

5.0.7 KNSVM (Matlab) ..... - 121 -

5.0.8 SVCM (Matlab) ..... - 124 -

List of Tables

Table Number	Table Title	Page	Chapter
Table I-4	Average AUC and Average Training Time (in seconds) of each classifier for the original data	45	4
Table II-4	Required training Times for the LG-SVM vs different classifiers (in seconds) (addition of 3741 samples in each iteration)	45	4
Table III-4	Average AUC and Average Training Time (in seconds) of each classifier	46	4
Table IV-4	Required training Times for the LG-SVM vs different classifiers (in seconds)	47	4
Table V-4	Average Performance Metric and Average Training Time (in seconds) of each classifier	49	4

List of Figures

Table Number	Table Title	Page	Chapter
Figure 1-3	User Registration	19	3
Figure 2-3	User Usage (left to right: spam bots, human users, traditional bots)	19	3
Figure 3-3	Top Used Words (left to right: spam bots, human users, traditional bots)	20	3
Figure 4-3	Frequency of Sentiment (left to right: spam bots, human users, traditional bots)	21	3
Figure 5-3	VIX Score	21	3
Figure 6-3	LG-SVM Model Architecture	23	3
Figure 7-3	Bi-LSTM Model Architecture	25	3
Figure 8-3	Architecture of the autoencoder	28	3
Figure 9-3	Technical makeup of the autoencoder	28	3

List of Abbreviations

Abbreviation	Word
LG-SVM	Local Global Support Vector Machine
SVM	Support Vector Machine
CBOE	Chicago Board Options Exchange
VIX Index	Volatility Index
Bi-LSTM	Bidirectional long-short term memory

Chapter One

Introduction to the Study

1.1 Introduction

1.2 Research Problem and Question

1.3 Research Objectives

1.4 Research Significance

Chapter One

Introduction to the Study

Chapter one will introduce readers to the subjects alongside explaining why it is imperative to deploy adaptive solution to the outlined ever-evolving challenge.

1.1 Introduction

Social media, which constitutes all electronic communication platforms, is a relatively new phenomenon that has reshaped the way in which individuals communicate, share content, conduct business, and obtain information with disruptive negative and positive externalities. Although primarily intended for human use, social media platforms have been known to be breeding grounds for robotic activities, referred to as 'bots' hereafter. The coexistence of humans and bots in a single platform that is rarely monitored has accelerate the dissemination of un-factual opinions in what is being referred to as the 'dissemination of disinformation' (Australian Government, 2021).

The dissemination of disinformation, or false information deliberately spread across a large audience with the intention to mislead (UW Bothell and Cascadia College: Campus Library, 2021), has real life consequences on society as a whole, such as the Jan 6, 2021, attack on the US's democracy that was only possible due to the movement's large reach over Facebook, which the House Energy & Commerce Committee (Hendrix, 2021) referred to as the "megaphone for the lies that fueled the insurrection," alongside a myriad of research attributing the promulgation of disinformation to the wide presence of bots, or automated accounts.

1.2 Research Problem and Question

Social media platforms have taken note of the presence of bots, along with action to ensure that their presence do not undermine democratic values. The latest change to Twitter's policy allows users to voluntarily label an account as a bot (Hutchinson, 2022). Altogether, policy changes and attempts to limit bots have fallen short of their objectives.

To date, bot detection techniques are challenged by their inability to keep abreast with bot progression as bots upgraded from their simplistic nature to resembling human activity, while its importance is increasingly being bought to the forefront of political discussions. During COVID-19, 43% of COVID-19 related tweets came from bots who have advanced several conspiracies, including QAnon, alongside retweeting partisan sites (McKenzie, et al., 2021).

Researchers are yet to come to consensus on the extent of bots' presence on social media with reports suggesting that nearly half of Twitter accounts are automated. These figures have been continuously refuted as bot detection methods' performances waned overtime and are challenged by the wide presence of users. For example, researchers may focus on a certain sub-section of Twitter during its model building period but use the same model on users that were not accounted for, such as South African women who respond to successful businesswomen with emojis (they were falsely classified as bots) (Roberts, 2020).

Nevertheless, researchers have pointed out that the extent of bots' presence is not as important as the extent of their influence. However, identifying their presence with precision remains a pre-request to measuring their impact on public discourse. Thus, Bot identification serves as the steppingstone to pinpointing the means in which they interact and their reach (Roberts, 2020).

Therefore, this research will increase bot detection flexibility in order for detection methods to keep abreast and evolve alongside bot activities. This will allow for enhanced and more flexible detection methods that to date remain limited. Their influence on human behavior, acknowledged to be far-reaching, will be limited to the influence they exert on investors' decision at the stock market due to the relative ease of measurement and the availability of data. The research will assume that investors are rational beings who undertake the best decision given the limited information presented with bots being one of many sources; thus, there presence influence decision-making processes. As such, the research will attempt to answer the following questions: Do flexible, more dynamic models perform better in bot detection and do they have a real-time impact on the stock market?

In keeps, this research will limit bot detection methods to the Twitter platform, while the influence bots exert limited to the volatility of the Chicago Board Options Exchange.

1.3 Research Objectives

As a response to growing public concerns, this thesis will address bot detection methods' inflexibility of detecting evolved bot activities by making bot detection activities more flexible and dynamic. Such efforts will overlook the introduction of an encoder-decoder architecture to condense sequential data while minimizing the loss of information and hold learn retention over time. The results will then be compared to a less flexible model to demonstrate the extent of its importance. The relatively inflexible methods are chosen to be support vector machine (SVM).

Bot's influence on human's behavior will then be measured by their influence on the stock market's volatility index through a multilayer

perceptron model under the assumption that humans are rational beings who undertake the best decision given all the information available; therefore, any behavioral changes is assumed to be attributed to changes in information driven by bots.

The thesis will center on Twitter, which is a free membership-based service for disseminating brief posts under 280 characters called "Tweets," due to over reliance on the platform in getting real-time information. This will help achieve the aim of the paper in the context of assessing real-time influence on market price volatility (Lutkevich & Wigmore).

1.4 Research Significance

The research will build on the social, scientific, political and economic need to continuously update bot detection methods to keep abreast with their progression and understand the full extent of their influence. Bots are continuously evolving, as such, detection methods must also evolve, requiring researchers to provide a continual string of solutions. This thesis's added value will include expanding detection method's flexibility to address changes in behavior across all time frames as the previous methods have failed to hold their ability as time progressed. Furthermore, the thesis will also attempt to measure bots' impact on human decision-making process, paying particular attention to financial decisions.

The contribution will centralize around extending the local global support vector model (LG-SVM) model to bot detection. The model expands detection flexibility by treating the data as unsupervised and introducing an autoencoder to reduce dimensionality in an end-to-end model framework. The framework will have multiple layers, the first of which is a convolutional layer that reduces dimensionality and learns key short-term characteristics, followed by a max pooling layer, two

Bidirectional long-short term memory (bi-LSTM) models, a bottleneck layer, an upscaling layer, and a deconvolution layer. This model solves for over-fitting and learns to evolve with time and proved to be more beneficial in real-world deployment (Madiraju, Sadat, Fisher, & Karimabadi, 2018). Additionally, the autoencoder will contribute by providing dimension reduction and learning in a smaller dimension without foregoing important information.

Chapter Two

Literature Review

2.1 Literature Review

2.1.1 Bot Detection Literature Review

2.1.2 Stock Market Literature Review

2.2 Theoretical Framework

2.2.1 Data Extraction

2.2.2 Fitting the data and validating the model

2.3 Research Hypothesis

Chapter two will build the theoretical framework for the study by exploring relevant publications and explore the dataset to provide a comprehensive understanding of the data, upon which hypothesis will be made.

2.2 Literature Review

2.2.1 Bot Detection Literature Review

Researchers first started looking into bot detection in 2010, using techniques that are now outdated by technological advances. At the start researchers detected bots by training models with a dataset that was manually labeled as bots or humans based on the researchers' opinion. This technique suffered from bias but was accepted at the time because bots were simple with their activities, merely being "fake followers" that rarely interacted with other users or spammed users with the identical messages. This theory was administered in a research assessment of bots of a particular category known as "fake followers" against a baseline of characteristics using multiple classifier models, including decision tree; however, it suffered from over-fitting and required trimming to be generalized, which makes it incompatible to use in a highly dynamic and evolving world (Cresci, Pietro, Petrocchi, Spognardi, & Tesconi, 2015).

To avoid over-fitting the data, researchers also explored minimizing features during the process of training models to classify whether a tweet is authored by a robot or a human or whether an account is human or bot. With a contextual LSTM model, which is used to incorporate the non-sequential metadata, it was found that over-sampling methods increased accuracy. Overall, metadata alone did not suffice; however, when combined with tweet content, the results were strong even though it may

be improved by increasing metadata utilization (Kudugunta & Ferrara, 2018).

Similarly, traditional machine learning bot detections measured equally incapable of accurately detecting newly emerged bots when compared to a more dynamic model known as the "Digital DNA model," which encodes behavior into a string known as the DNA, returning a higher accuracy rate. The higher accuracy was accredited to the model's ability to take into consideration different behaviors or inputs at once; for example, it considered comments, likes, shares, and mentions, allowing for an overall comprehensive analysis that identifies similar behaviors shared by bots such as retweeting political candidates tweet within minutes of a Tweet's publication. Despite the higher accuracy rate, this approach fell short of detecting bots that demonstrate a new set of behavior because it depends on a given baseline of characteristics (Cresci, Pietro, Petrocchi, Spognardi, & Tesconi, The paradigm-shift of social spambots, 2017).

Other baseline characteristic analysis include social network analysis of what is coined to be botnets, which is a group of bots working in coordination towards certain goals where a single bot master controls all bots. Bots are assessed by their social network or similarity in online activity using a B-corr model that classifies output based on the analysis of network traffic flow using IP addresses. The research faced similar shortcomings faced in the Digital DNA model (Hostiadi, Tohari, & Wibisono, 2020).

As a response to bot evolving along with the deployment of AI, researchers have also adapted AI technology into bot detection. Generative adversarial networks (GANs) are machine-learning analysis in which two or more training models are trained jointly in a stimulated environment. The stimulant network has a generator network that is responsible for creating the data, while the discriminator model classifies the date on the

spot. With time, the generator is trained to generate more complicated examples and the discriminator model learns to adapt to a highly evolving environment, which resulted in a high accuracy rate. However, researchers questioned whether the generated samples hold in comparison to a real dataset extracted from a live online network (Randhawa, Aslam, AlAuthman, Rafiq, & Comeau, 2021).

Most recently published papers have been updating older models in order to bypass their limitation and detect bots in a more flexible manner. An example of which includes upgrading the generative adversarial network (GAN), a semi-supervised method that that extracts behavioral patterns, to create a newer model, referred to as GANBOT, by incorporating a layer of LSTM between the generator and the classifier layer. The GANBOT had a better score when compared with traditional methods, LSTM, which the authors attributed to the method's ability to learn behavioral patterns (Shaghayegh, Salehi, & Reza, 2021).

2.2.2 Stock Market Literature Review

Similar to bot detection, stock market prediction has also undergone tech-advancement based evolution. At the onset, forecast traditionally centered around time-series methods such as Autoregressive Integrated Moving Average, which smooths the model using exponential mathematics (ARIMA) (Udom, 2014). ARIMA has traditionally outperformed moving average methods due to the nature of the cyclical business market; however, the market has lost its stability as they became increasingly integrated and susceptible to unprecedented political challenges (Zhou, 2020).

However, as markets evolved, technology did to, creating space for the implementation of machine learning algorithms, such as the Convolutional Neural Network (CNN), which has reduced dependence on

experts' expectations as the algorithm extracted features manually. CNN far outperformed traditional methods even though performance metrics was note published into the study (Zhou, 2020). This strong performance has extended into other machine learning algorithms when researchers compared the Long Short- term Memory (LSTM) model to the ARIMA model alongside comparing an LSTM model that incorporated the ARIMA shortcoming into what became known as the hybrid model to predict housing sales. The hybrid model outperformed other models in short-term predictions while the traditional LSTM yielded stronger results in long-term predictions as they learned characteristics not pinned by basic mathematical algorithm forecasts (A. Temür, 2019).

Moreover, technology created new high dimensional spaces, to analyze dynamic business behavior such as the plethora of promotions that eliminate cyclical patterns, in sales in which the Support Vector Machine (SVM) returned the highest accuracy in predictions. In a similar study, deep learning models were found to predict sales in a more accurate manner; however, such difference remain marginal when Decision Tree (DT), Random Forest (RF), Support Vector Regression (SVR), Artificial Neural Networks (ANN) and Linear Regression (LR) are compared (A. Loureiro, 2018).

Feature extraction remains a key determinant of the success of the models. The key difference between deep learning models and other traditional methods, is that deep learning conducts feature extraction, allowing the model to learn about trends that influence sales, allowing for a more precise prediction (A. Temür, 2019).

Nevertheless, Stock market prediction has been proven to be increasingly challenging as stocks respond to changes across the world with no limits or warning to what may influence it. Thus, researchers have adopted a more dynamic model with an incremental learning method,

which was proven to be more accurate in its short-term prediction. Researchers note that these methods is that they rely on an inversion of an information matrix, challenging their ability to adapt quickly (Basnat, Ahmed, El Ghazali, & Jain, 2014).

Furthermore, with the rise of technology, data scientists can incorporate Google search trends into their models to improve accuracy (T. Boone, 2015) as the more information the model had, the greater its accuracy. Likewise, bots presence on Twitter have redirected susceptible users and altered investment techniques as a an exploratory analysis using Tweet sentiment and text mining have proven to be correlated with stock price fluctuations in a study conducted by researchers in the US (Kretinin, Samuel, & Kashyap, 2018).

2.3 Theoretical Framework

This research's empirical analysis is shaped by and rests on findings from academic publications associated with bot detection and stock market prediction models. This thesis attempts to detect bot activities on Twitter utilizing an end-to-end LG-SVM model with the use of autoencoder to minimize Tweet features, while comparing findings to that of a standard model such as the SVM.

The model assumes Twitter to be highly dynamic environment that requires an equally flexible model to detect bot activities. The assumption is shared across researchers in the field, particularly since the results of dynamic models have proven to be stronger in yield.

The end-to-end model is assumed to learn and minimize its findings without foregoing vital information that help distinguish bot activities from that of human, particularly since they have been increasingly cohesive with

their activities as bots are being shaped by human activities through their learning algorithms.

Later, human and bot activities will be fed into a multilayer perceptron model to assess their level of influence on stock market volatility to measure their impact on human decision-making. This attempt is made under the assumption that humans are fully logical human beings and make the best decision given the information available; therefore, information shared by bots does influence human decision-making.

2.3.1 Data extraction

The data utilized in this thesis is extracted from a private database shared by researcher Louis Daniel (Daniel, Loyola-González, Monroy, & Medina-Pérez, 2021). The dataset then underwent cleaning with the goal of eliminating the traditional bot due to their diminishing activity on the platform; thus, building a model that reflects current activities as much as possible. Data cleaning also included removing punctuations, hashtags and user handles, alongside stemming the word to its bases in order to minimize the number of words used in the training to enhance accuracy. Stock market volatility data is extracted from Chicago Board Options Exchange, which publishes data in several different intervals, such as daily.

2.3.2 Fitting the data and validating the model

Bot detection modeling is undertaken with the simple SVM model and later compared to a more flexible LG-SVM model, and stock market volatility is assessed through the use of a back-propagation multilayer perceptron model. The model is fitted using tweets and whether they are a bot or not. Features were left to a minimum, limited to the tweet itself, to understand their activities rather than their popularity since the goal is assess outcomes based merely on content shared. Upon validation of a model, which will be discussed at a later stage, the model will deployed to

detect whether a Tweet is written by a human or a bot, and assess their impact on the stock market volatility with an incremental model. The model is validated using the area under the curve (AUC).

2.4 Research Hypothesis

At this point, the hypothesis being made in correspondence to bot detection is that the more dynamic model with an encoder will outperform the less flexible model due to their ability to minimize features without foregoing information learned. The thesis operated under the assumption that encoders are able to detect latent features in a highly dynamic environment which are usually overlooked in standard models that attempts to treats data across all timeframes as unified information; therefore, the dynamic model will outperform standard models. No attempts will be made to the extent of bots' presence on social media platforms as the thesis aims to explore the impact of applying more dynamic models rather than detecting their presence.

In terms of bots' influence on human behavior, the thesis hypothesizes that bots skew public discourse and vigilantly influence human behavior due to their ability to share or hide vital information. Thus bots' activities, or tweets, is expected to significantly correlated with stock market volatility rate since investment techniques are driven by expert decisions who are susceptible to the influence of bots behaviors. No attempts will be made to predict whether their influence is positive or negative since it is dependent on the tweet content, which is hard to predict without knowing what information is being shared.

Chapter Three

Methods and Procedures

3.1 Data Collection and Procedures

3.1.1Data Description and Visualization

3.2 Model

3.2.1 Convolutional and Max Pooling Layer

3.2.2 Bi-LSTM

3.2.3 LG-SVM

3.2.4 Multilayer Perceptron

3.2.5 Autoencoder

3.3 Statistical Approach

3.3.1 Convolutional and Max Pooling Layer

3.3.2 Bi-LSTM

3.3.3 LG-SVM

3.3.4 Autoencoder

3.3.5 Multilayer Perceptron

Chapter 3 will provide the blueprint of the thesis's analytical framework, alongside setting the groundwork for hypothesis testing and recommendations made at later chapters.

3.1 Data Collection and Procedures

To deliver on its goal of contributing to previous publications, the thesis obtained the dataset previously used and only manipulate the model while holding all else constant. This procedure will allow for a more precise comparison of the new models' impact particularly since Twitter has a vast landscape of data; therefore, extracting data from a different segment may lead to different outcomes not solely due to model changes.

The dataset was requested from Louis Daniel, who has granted access to their private dataset under the condition that it is not shared with the public and is properly sited in the paper. The data collected covers tweets from human users, social spam bots, and traditional spam bots, from 2009 to 2014. Traditional spam bots are automated accounts that were used in the early days of automation, primarily used to spam users with links, while the social spam bots are novel automated users that resemble human behavior.

The dataset is an unbalanced dataset with a large number of variables. In terms of data size, human tweets total to 574,948 spam bots 1,610,034, and traditional bots 1,048,575. The data cleaning process started with the dropping of all features with the exception of the Tweet itself and the date it was written since the aim of this thesis is to predict the author of the tweet using the tweet content only, regardless of the extra information the metadata may provide. Later, a column was added to each dataset with binary outcome that states 1 for bots and 0 for humans (traditional and spam bots were both summed into the bot category).

The next data cleaning process focused on minimizing noise within the Tweets themselves by reducing words to their original form. To do so, emojis, URLs and punctuations were dropped as they add little meaning to the dataset and skew findings. As such, "Stop words" or a list of commonly used words that add no significance to the meaning were also dropped. Stop words include: "is," "are," and "the" among others.

The text cleaning process also underwent further generalization, which include the conversion of uppercase letters to lowercase, stemming, the transformation of words to their derived stems, and lemmatization, which further reduces words to their roots.

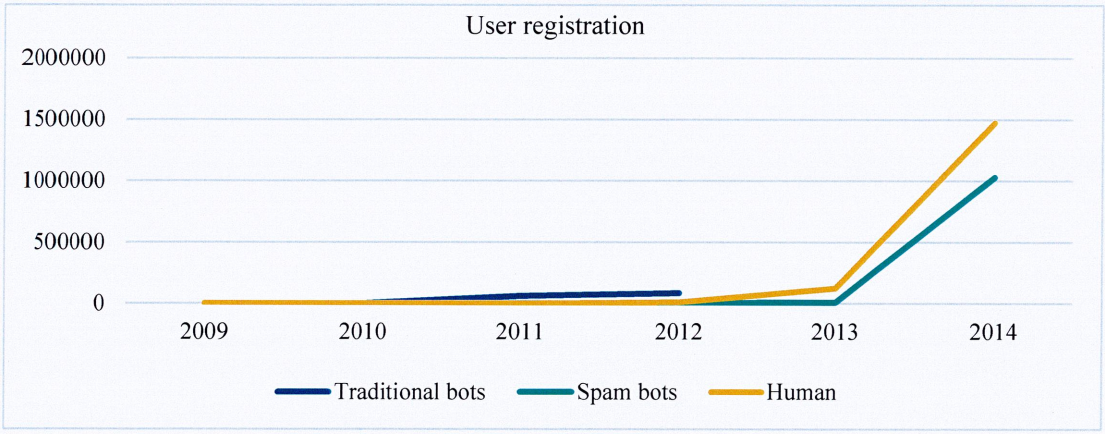
The volatility rate is extracted from the Chicago Board Options Exchange and is measured on a daily basis. The VIX Index is enlisted among the most popular volatility indexes, and is the average of a 30-day expected volatility in the US S&P 500. The volatility rate did not undergo any cleaning other than the upending of it to the master dataset, which includes human accounts and bots. A volatility score of 30 or greater is an indication of extreme volatility (Cboe Volatility Index).

3.1.1 Data Description and Visualization

The data cleaning and visualization process both took place over Google Colab. In line with the literature review, figure 1 reveals the trend in twitter.

Traditional bots have been decreasing in popularity as they have become relatively more recognizable by the social media platform, which, in return, increases the probability of them getting shutdown. Among human users, twitter has been growing in popularity along the years, which is expected because user increase supports further increase as they collectively enhance the content shared. Likewise, as demonstrated in Figure 1, spambot has seen a rise in popularity due to technological

advancement, and its relatively lower probability of getting detected due to



its resemblance of human behavior, leading to an increase in their presence.

Figure 1-3: User Registration

On a usage trend level, figure 2 affirms that traditional spambots have utilized twitter the least with a slightly higher usage towards the end of the day, while human users tend to use twitter on every other day with an uptake in usage towards the end of the day. Spambots tend to use twitter heavily with almost equal activation level throughout the day except for the morning hours.

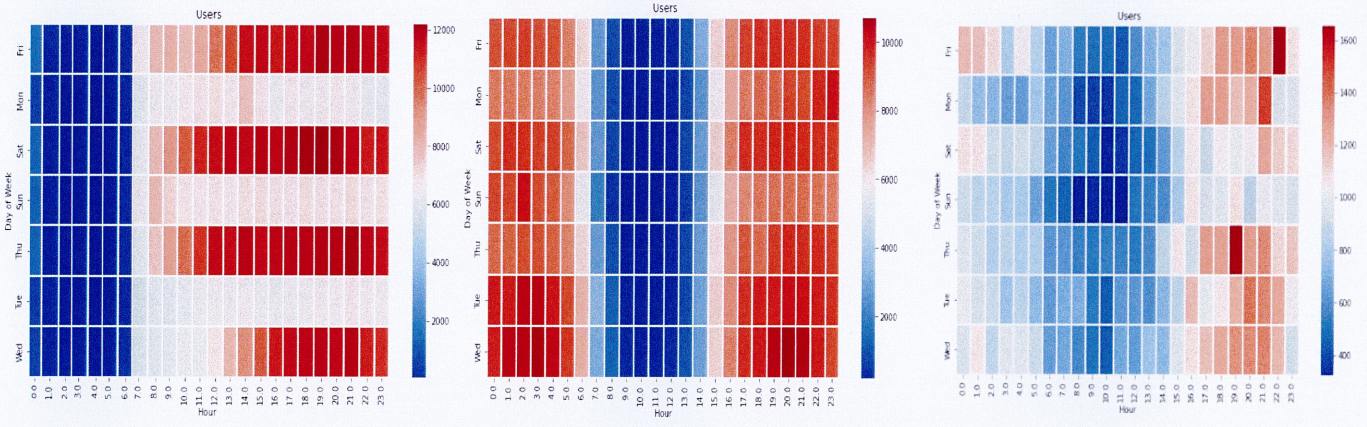


Figure 2-3: User Usage (left to right: spam bots, human users, traditional bots)

Using NLTK and frequency distribution sort, the top 25 words most used were extracted and outlined in figure 3. Genuine users and spam bots both used real words to communicate with users; however, genuine users

were more inclined to write in Italian since the dataset was extracted by Italian researchers.

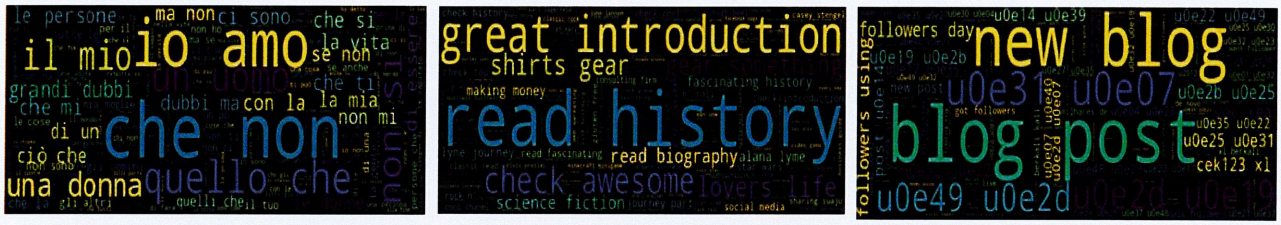


Figure 3-3: Top Used Words (left to right: spam bots, human users, traditional bots)

Traditional bots behaved as expected with their mandate of sharing URLs, with a majority sharing blog posts or advertising for followers' purchase.

Finally, the data underwent sentiment analysis, or opinion mining. Sentiment analysis analyzes the words users use to determine a public consensus and scale emotions from 0 to a 100, where a score of zero depicts extreme negativity and a score of a hundred depicts an extreme positive sentiment. It is worth noting that opinion mining does not recognize sarcasm or jokes as it takes words at face value.

Figure 4 outlines that human users tend to tweet in a neutral manner, expressing words that do not depict positive or negative emotions. This could be a result of the timeframe in which the tweets were collected. During the years of 2009 and 2014, political polarity has yet to take over the online world. Users at that time were mainly using the platform to share updates of their lives. Likewise, traditional bots followed the same trend of neutrality, in line with their activity mandate of spamming users with links and advertisements. Spam bots, all though created to follow genuine users'

behavior patterns, tend to overly use positive vocabularies, as well as using a higher relatively negative sentiment.

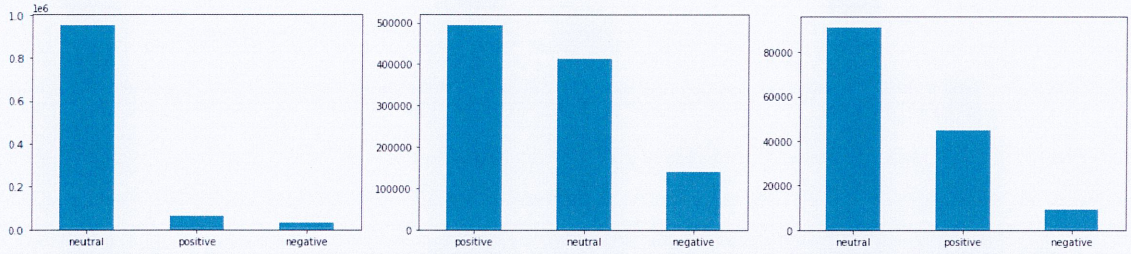


Figure 4-3: Frequency of Sentiment (left to right: spam bots, human users, traditional bots)

As a result of the exploration, it has been decided to drop the traditional bot dataset since their function has been outdated or easy to spot, placing them in a position in which they are particularly vulnerable to getting shutdown by Twitter. Therefore, there findings would not add much to the evolving platform in which bots that mirror human behavior is expected to takeover.

In terms of the VIX score, the volatility rate held a constant high rate with fluctuations between 145 and 61. The data shared in Figure 5 reflects the economic landscape of the time. 2009 marked the end of an economic downturn that initially started in 2006, but overlook many economic shocks driven by market bubbles (Kuepper, 2022).



Figure 5-3: VIX Score

3.2 Model

The flexible model developed to identify whether a Tweet is written by a human or a bot is a deep temporal end-to-end model that incorporate different models. At the first level, the data will be inputted into a 1-D convolutional layer followed by a max pooling lens, both of which are tasked with reducing the dimensionality of the Tweet by learning key word features and learn dominant short-term characteristics in the language used. The output is latter supplied to a bi-LSTM model that is also tasked with learning changes in key features across time in both forward and backward directions. As a result, the sequence is further shrunk to fit into a lower dimensional space. The two steps mentioned above will act as an encoder, which treats the data as an unsupervised dataset to learn the vital aspects of the dataset as it shifts the data from a higher dimension to a lower dimension. The encoder adds a dimension of flexibility as most models treat higher dimensional data as lower dimensional data during its learning process, leading to the loss of key information. During the model training phase, the data undergoes reconstruction through the use of an upsampling layer and passed through a deconvolutional layer to reverse the effect of data reduction, under the 'decoder' phase. Additionally, a bottleneck layer is introduced to vectorize the Tweets in order for it to be accepted by the model. The encoder and decoder both serve to minimize the Tweets while minimizing the loss of information.

After the model is trained using forward and backward propagation, the decoder is dropped, and the output is reprocessed and feed into bi-LSTM model. The LG-SVM model improves on the performance of the SVM model by introducing a DSVM layer that will allow for incremental layer, under which the dataset is divided into sub-groups that are trained individually against an existing optimal solution. The model will allow for

the computation of a scatter matrix during the input stage with the objective of identifying near-global characteristics. Thus, hyperplanes are built around local characteristics which is identified during the input space.

Finally, the output is assigned an output class under a cost function set by minimizing the mean square error (MSE). Figure 6 outlines the steps mentioned above

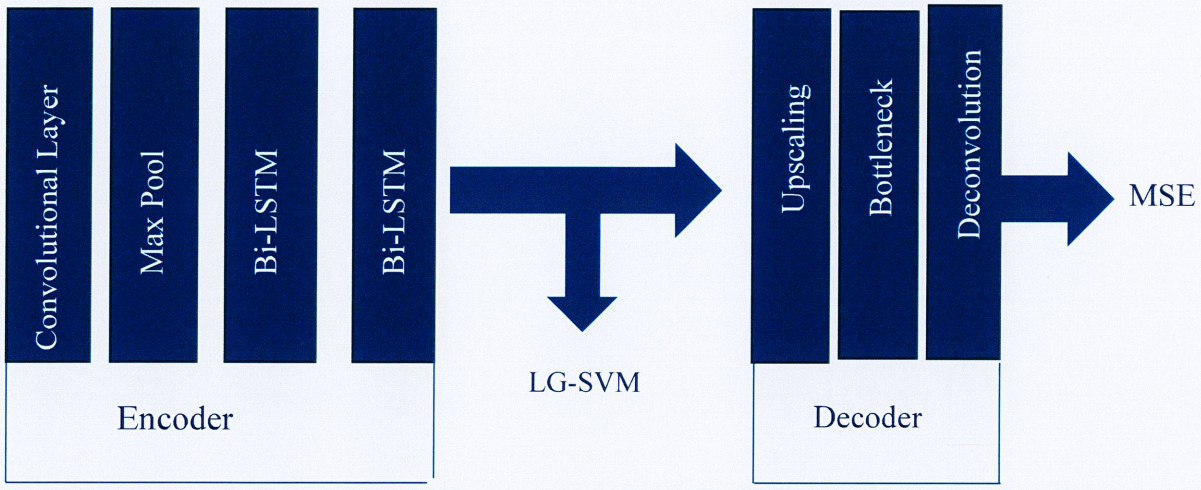


Figure 6-3: LG-SVM Model Architecture

3.2.1 Convolutional and Max Pooling Layer

The convolutional layer, as a tool, identifies and maps features by extracting features with a filter of a particular size to produce a feature map. The feature map is then feed to a max pooling layer to extract the largest features.

3.2.3 Bi-LSTM Architecture

The bi-LSTM is a combination of two LSTM models working side-by-side where one takes the input forward and the other takes the input backwards. The LSTM model has 4 main functions: the memory cell, the forget gate, the input gate, and the output gate. Altogether, these functions work to store essential information while learning to forego irrelevant information.

At the heart of the model, sits the cell state that transports information deemed important across the sequence chain, where information is added or removed from the cell state through gates. The gates are Artificial Neural Networks (ANN) that determine which information is relevant to add or remove during the training process, while the cell state serve as the memory of the network. The memory allows the model to remember information incorporated at early stages, creating a long-term memory.

Gates assess information using mathematical computation of a sigmoid activation function. Sigmoid function is a logistic function with a range limited to 0 and 1, where the output closer to 0 is dropped. The sigmoid function has an S-curve function, which allows for non-linear learning boundaries useful in reducing the sensitivity of predicting complex datasets such as text.

The output of the sigmoid function then determines the course of action the gates undertake. The closer the value is to 0, the more likely the gate will forget the information, while the closer it is to 1, the more likely it will retain the information. Output is then carried over the calculate its entry into the cell state, which is pointwise multiplied by the forget vector, during which state information may be removed from the cell state if they are multiplied by a near-zero value. Input that gets past through the input gate then undergo a pointwise addition to update the cell state with the new values deemed relevant. Finally, the modified cell state is passed through a tanh function, which is later multiplied with the sigmoid output to obtain the output probability.

To turn the LSTM model into a bi-LSTM model, additional hidden layers are added to the model that allow for the reading of inputs backwards. The new model is now able to read data in a feedforward style as well a

feeding data backwards, both of which are connected to a single output. The process is highlighted in Figure 7.

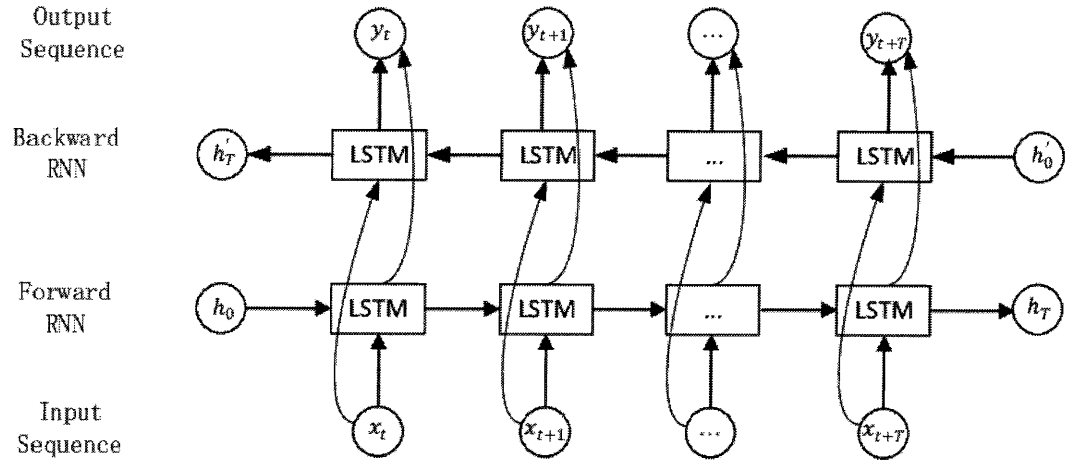


Figure 7-3: Bi-LSTM Model Architecture

3.2.4 LG-SVM

The SVM model is a model that separates the dataset in a hyperplane by drawing clear-cut boundaries. The model has an input layer that accepts the data, a hidden-lay with a black box learning process, and an output neuron that decides the class after considering a bias value. The SVM is incremental when it the data it accepts is divided into subgroups that are fed into the model at different timeframes, of which only the support vectors are preserved and added to the next subset. This is achieved in the LG-SVM model by adding a layer of will allow for character identification in the input process by clustering characteristics in local clusters that belong to a global support vector that identifies the key characteristics.

3.2.5 Multilayer Perceptron

To evaluate the impact human versus bot tweets and other preventive measures had on stock market volatility, it was assumed that Tweets are the single go-to source for investors to gain knowledge and that an impact is felt on time with no time-lags. Therefore, a multilayer perceptron model was conducted on Weka. The perceptron model was selected since the data

had a complex relationship that does not satisfy the requirements of other models, such as the linear regression's assumptions, despite the regression's popularity in the business world. Published works have proven the perceptron model to operate better with heteroskedasticity as it learns from hidden relationships and is a relatively dynamic model, which contributes towards this research's goal of incorporating more flexible models in predictions (Mahanta, 2017).

The multilayer perceptron is a back propagation machine learning algorithm that mirrors the way in which the human brain functions. Like the human brain, knowledge is learned through a learning process that allows for the ability to create reasonable outputs or generalizability. The learning process is stored in the weights where information is processed through a neurons and weights are assigned to the synapses to store the learning algorithm. The weights are later summed with the synapses before passing through an activation function that sets parameters to the output before assigning the class. However, neural networks are challenged by the requirement to decompose problems into relatively small tasks and treat each segment as independent subsets.

The perceptron model followed the same method utilized in the bot detection model with an autoencoder that reduced dimensionality from 130 to 100 and 70. A learning rate of 0.3 was set with 500 neurons since it is a large dataset.

3.2.6 Autoencoder

An autoencoder neural network subsists of an unsupervised learning algorithm that take on backpropagation computation to turn the raw values to equate to the input; thus, reducing the size of the input. The autoencoder is made up of three parts, the first of which is the encoder, followed by the code and a decoder. The encoder is tasked with supressing the input into a

latent space depiction to fit into a reduced dimension, while the code carries the compressed input to the decoder, during which vital patterns are learned. The decoder turns the compressed input back to its original dimensionality (Sayantini, 2020).

The autoencoder is characterized by four key parameters as follows: the code size, which signifies the number of nodes, the smaller the size the more the data is compressed. The research includes 3 sizes for comparison reasons: 70, 100 and 130. The Number of layers, which is set to ten. The number of nodes per layer refers to the nodes that decreases with each subsequent layer and returns to its normal levels at the decoder stage. The number of nodes per layer are also 130, 100 and 70 per experiment. The final parameter is the mean square error, which is used as a performance metric. Figure 8 outlines the architecture of the autoencoder, while figure 9 highlights the technical input of the autoencoder (Sayantini, 2020).

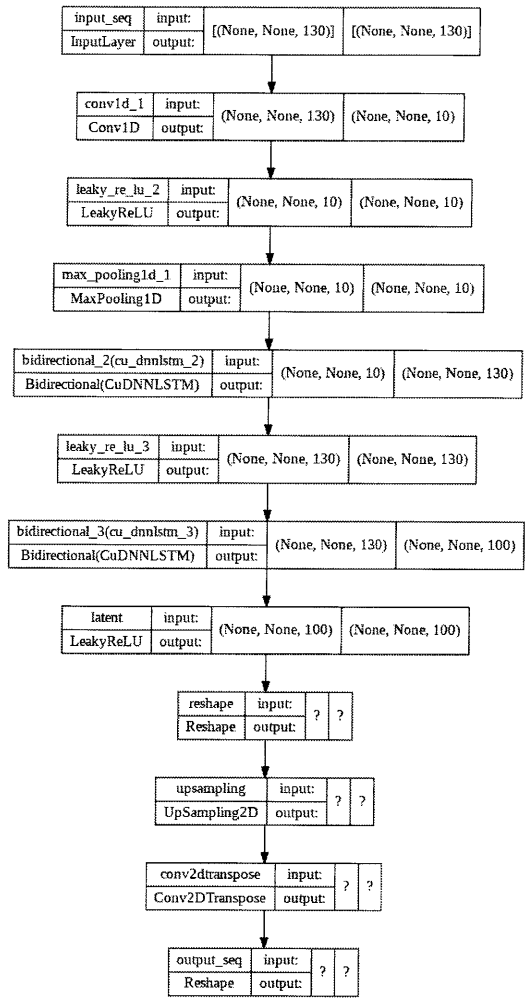


Figure 8-3: Architecture of the autoencoder

```
Model: "AE"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_seq (InputLayer)       [(None, None, 130)]       0

conv1d_1 (Conv1D)            (None, None, 10)          6510

leaky_re_lu_2 (LeakyReLU)    (None, None, 10)          0

max_pooling1d_1 (MaxPooling  (None, None, 10)          0
1D)

bidirectional_2 (Bidirectio  (None, None, 130)         147680
nal)

leaky_re_lu_3 (LeakyReLU)    (None, None, 130)         0

bidirectional_3 (Bidirectio  (None, None, 100)         185600
nal)

latent (LeakyReLU)           (None, None, 100)         0

reshape (Reshape)            multiple                  0

upsampling (UpSampling2D)    multiple                  0

conv2dtranspose (Conv2DTran  multiple                  65130
spose)

output_seq (Reshape)         multiple                  0

=================================================================
Total params: 404,920
Trainable params: 404,920
Non-trainable params: 0
```

Figure 9-3: Technical makeup of the autoencoder

3.3 Statistical Approach

3.3.1 Convolution and Max Pooling Layer

The output was classified by first passing the data through a one-dimension convolutional layer, where an input of length n denoted as x and a kernel of length k denoted by h is shifted by s under W strides, presented in formula (1).

$$\sum_{i0}^{P} x(n-i)h(i) \tag{1}$$

before passing the data through a max pooling layer, which maximizes the input function, is calculated using formula 7, where **I** refers to the input array, **F** the pooling window size, and **S** the stride

$$\left(\frac{(I-F)}{S}\right) + 1 \tag{2}$$

3.3.2 Bi-LSTM

In terms of the Bi-LSTM, the LSTM model compromises of three key units to learn, as follows: the input gate denoted as $\boldsymbol{i_t}$, the forget gate denoted as $\boldsymbol{f_t}$, and an output gate denoted as $\boldsymbol{o_t}$. Information inputted as $\boldsymbol{X_t}$, pass through each gate with the use of a point-wise multiplication and sigmoid function operations, formula 3, to give an output from the hidden gate as $\boldsymbol{h_{t-1}}$,

$$\sigma(\boldsymbol{x}) = \frac{1}{1+e^{-x}} \tag{3}$$

The forget gate, calculated in formula 4, gives an output between one and zero as the value closer to zero is discarded, while input gate, formula 5, decides what information to update with the aforementioned logic.

$$f_t = \sigma(W_f * [h_{t-1}] + b_f \tag{4}$$

$$I_t = \sigma(W_i * [h_{t-1}] + x_t \quad (5)$$

Finally, the modified cell state is passed through a tanh function, which is later multiplied with the sigmoid output to obtain the output probability. The tanh function is outlined in formula (6).

$$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \quad (6)$$

To turn the LSTM model into a bi-LSTM model, additional hidden layers are added to the model that allow for the reading of inputs backwards. The new model is now able to read data in a feedforward style as well as feeding data backwards, both of which are connected to a single output.

3.3.3 LG-SVM

The LG-SVM classifier is motivated by the combination of the merits of the discriminant-based classifiers and the Support Vector Machine (SVM) classifier. These two are combined by incorporating the between-class and within-class scatter matrices, calculated in the input space, into the SVM optimization problem. Calculating the scatter matrices in the input space will overcome the limitations in terms of time and memory storage, while considering the global information provided through discriminant-based classifiers. Then, combining these characteristics with local ones provided by SVM will improve the performance. Thus, the proposed approach enhances output by considering class distributions' near-global variation, as well as the utilization of the Kernel Nonparametric Discriminant Analysis (KNDA) classifier in the calculation of observations that are both part of a class and fall between classes, during the input phase.

The model's input phase is denoted by formula 7, where **N** is the total number of samples that are split into two different classes denoted as **N1 and N2**, where N1 refers to the total number of samples in class one (x_1)

and N2 in class 2 (x_2). The output of the model is outlined in formula 8, where the output is a number between 0 and 1 associated with the probability of falling a given class.

$$X = \{x_i\}_{i=1}^{N} \tag{7}$$

$$Y = \{y_i\}_{i=1}^{N} \tag{8}$$

Additionally, the observations that fall between classes are calculated with the nearest neighbor mean vectors as outlined in formula 9, where K refers to the number of neighbors considered, x_m the observations that belong to x_1 and $(x_m)nn(i)$ refer to the i-th nearest neighbor of x_2.

$$M_q^k(x_m) = \frac{1}{k}\sum_{i=1}^{K}(x_m)NN(i) \tag{9}$$

Therefore, building on the above, the scatter matrix corresponding to the observations that fall between classes is calculated during the input phase utilizing formula 10 with L1 and L2 formulated in formula 10 and 12, respectively.

$$S_b = \frac{1}{N_1+N_2}\left(\sum_{i=1}^{N_1}\Psi_i L1(i)^T L1(i) + \sum_{i=1}^{N_2}\Psi_i L2(i)^T L2(i)\right) \tag{10}$$

$$L1(i) = x_i - M_2^k(x_i) \tag{11}$$

$$L2(i) = x_i - M_1^k(x_i) \tag{12}$$

The model has put control factors in space to ensure its continuity as data noise increases. The weighted function, denoted as $\boldsymbol{\Psi}$, counters and minimizes the influence observations further away from the decision boundary have on the output by deaccelerating the rate in which values

recede to zero as they move further from the decision boundary through the introduction of parameter **Y**. This is presented in formula 13:

$$\Psi_i = \frac{\min\left(d(x_i, M_1^k(x_i))^{\Upsilon}, d(x_i, M_2^k(x_i))^{\Upsilon}\right)}{\left(d(x_i, M_1^k(x_i))^{\Upsilon} + d(x_i, M_2^k(x_i))^{\Upsilon}\right)} \quad (13)$$

On the other hand, the observations that fall a given category's scatter matrix as computed as outlined in formula 14 during the input phase. The formula considers I as the identity matrix, 1_{N1} and 1_{N2} as the umbrella matrix with all entries belonging to $\frac{1}{N_1}$ and $\frac{1}{N_2}$, respectively.

.

$$S_w = X_1(I - 1_{N_1})X_1^T + X_2(I - 1_{N2})X_2^T \quad (14)$$

The overall matrix, denoted as $\boldsymbol{\Theta}$, is outlined in formula 15, where η is a control parameter, calculated in formula 16, for the weight vector and M sets the upper limit boundary of the dataset as it equates to or is a lesser value than N.

$$\Theta = \eta S_w (S_{b+} \beta I)^{-1} S_w + I \quad (15)$$

$$n = 1 - \sum_{i=1}^{M} \frac{\Psi_i}{M} \quad (16)$$

The above symbol of $\boldsymbol{M}$ symbolizes the Mu, which reflects the 0.000001 preceded multiplier. The computation of the trade-off parameter $\boldsymbol{M}$, is computed by taking into account the effect of the near-global characteristics of the dataset, which, in return, reduces the calculation time, in which $\boldsymbol{M} \leq \boldsymbol{N}$, total samples available, and Ψ_i is the computational terms in accordance with equation 13, which penalizes observations based on their distance from the decision boundary as new observations are considered incrementally. Figure 9 outlines the above graphically (Soula, Tbarki, Ksantini, Said, & Lachiri, 2020).

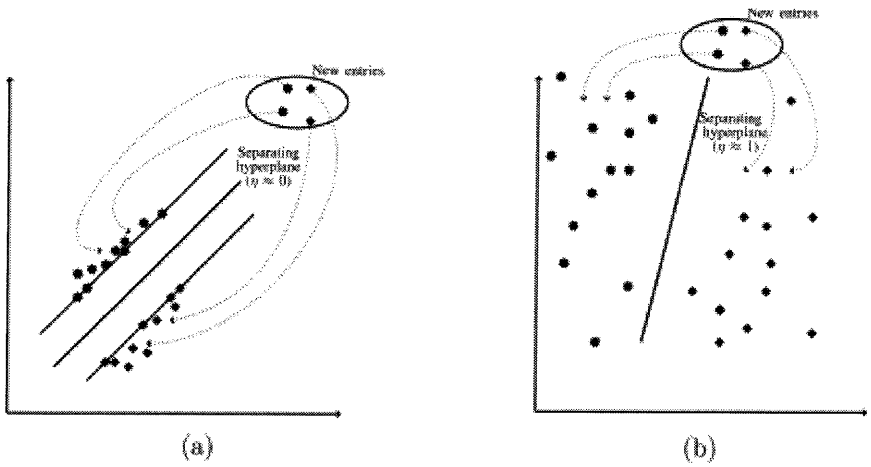


Figure 9, Chapter 3: 2. Hyperplane separation of (a) low data spread and (b) wide data spread

To further deliver on the SVM's objective, the formula undergoes minimization by incorporating Θ into the optimization phase as outlined in formula 17, and further reformulated into formula 18 through the insertion of a Karush-Kuhn-Tucker (KKT) condition. α_i in formula 18 is a number equal to or greater than zero and equal to or less than C For all i from 1 to N, and $\sum_{i=1}^{N} y_i \alpha_i = 0$. The new kernel matrix, K_{DSVM} refers to X_T Θ

$$\min_{w \neq 0, w_0} \left\{ \frac{1}{2} w^t \Theta w + c \sum_{i=1}^{N} \max(0, 1 - y_i(W^T x_i + w_0)) \right\} \quad (17)$$

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K_{DSVM}(x_i x_j) \quad (18)$$

The mathematical computation denoted to date is a linear function that needs to undergo further transformation to support dynamic learning, which is done by the introduction of a Singular Value Decomposition on Θ, as denoted in formula 19, where **U** refers to the initial matrix, and Σ the transverse matrix, under which Q = $\Sigma^{-\frac{1}{2}}$U is applied to the sample to calculate the new sample $xi' = Qx_i$ to obtain formula 20, which results in formula 21 if extended to the whole initial matrix

$$\Theta = U^T \sum U \quad (19)$$

$$x_{i'T} x_{ji} = x_i^T Q^T Q x_j = x_i^T \Theta^{-1} x_j \quad (20)$$

$$X' = [x'_1, \ldots, x'_N] = [Qx_1, \ldots, Qx_N] = QX \quad (21)$$

Thus, solutions are attained using standard linear SVM with samples xi' and $x'TX'$ as a kernel matrix, which if extended to the transformed xi' samples would result in a non-linear flexibility in $x'TX'$ and xj as outlined in function 22 with an RBF kernel in formula 23.

$$x'^T x' = X^T Q^T QX = X^T \Theta^{-1} X \quad (22)$$

$$K_{DSVM-RBF}(x_i, x_j) = e^{-\frac{(x_i - x_j)^T \Theta^{-1} (x_i - x_j)}{2\sigma^2}} \quad (23)$$

Next the model underwent further alteration to support incremental learning, in which each additional observation, X_c is added to the optimal solution. This is achieved by identifying the scatter matrix of the discriminant-based classifier during the input phase to detect near-global features, which is added to the information retained on local and global characteristics. Computing the feature identification during the input space allows for the reduction of costs associated with time. The initial data is then moved to a higher dimensional feature space with a singular value decomposition to allow the classifier to process big and complex datasets with a relatively high rate of accuracy. In this model, the Karush–Kuhn–Tucker conditions is of utmost importance for achieving optimality. By preserving the KKT conditions for all trained data, an adiabatic increment is maintained, to which formula 12 is reformulated to formula 24, where K is equal to $X^T \Theta^{-1} X$. It is worth noting that the model only accepts new observations if all of the KKT requirements are met, and its weight is set to 0 as is non-support vectors. However, weights and bias values are updated if the newly added observation is proven to be a support vector using formula 25 and will keep undergoing updates until an optimal solution is reached.

$$g_i = -1 + \sum_{j=1}^{N} K_{i,j} a_j + y_i w_0 = \begin{matrix} \geq 0, \text{if } \alpha i = 0 \\ 0, \text{if } 0 < \alpha i < C \\ \leq 0, \text{if } \alpha i = C \end{matrix} \quad (24)$$

$$\frac{\partial W}{\partial W_0} = \sum_{j=1}^{N} y_{i,} a_j = 0 \quad (25)$$

To reach such a goal, the dataset **X** is divided into subsets, while taking into consideration the following KKT conditions: The first set **S** will hold an unbounded support vector with weight values between 0 and C, the second set, **E**, is constrained by weights equal to C, the last set, **N**, comprises of a non-support vector points with weight values of 0, and P = E U N. The conditions may be expressed in a matrix form denoted in formula 26.

$$\begin{bmatrix} \triangle g_c \\ \triangle g_s \\ \triangle g_p \\ 0 \end{bmatrix} = \begin{bmatrix} y_c \, K_{cs} \\ y_s \, K_{ss} \\ y_p \, K_{ps} \\ 0 \, y_s^T \end{bmatrix} \begin{bmatrix} \triangle w_0 \\ \triangle \alpha_s \end{bmatrix} + \triangle \alpha_c \begin{bmatrix} K_{cc}^T \\ K_{cs}^T \\ K_{cp}^T \\ y_c \end{bmatrix} \quad (26)$$

As such in formula 26, any change to value $\triangle \alpha_c$ is captivated by the required modifications in $\triangle \alpha_s$ and $\triangle W_0$. Therefore, row two and four is further modified in formula 27 and 28, respectively, as $\triangle g_s = 0$.

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \, y_s^T \\ y_s K_{ss} \end{bmatrix} \begin{bmatrix} \triangle w_0 \\ \triangle \alpha_s \end{bmatrix} + \begin{bmatrix} y_c \\ K_{cs}^T \end{bmatrix} \quad (27)$$

$$\begin{bmatrix} \triangle w_0 \\ \triangle \alpha_s \end{bmatrix} = \beta \triangle \alpha_c \quad (28)$$

$$\beta = \begin{bmatrix} \beta_{w0} \\ \beta_m \end{bmatrix} = -Q \begin{bmatrix} y_c \\ K_{cs}^T \end{bmatrix}, \forall m \in s \quad (29)$$

$$Q = \begin{bmatrix} 0 \, y_s^T \\ y_s \, K_{ss} \end{bmatrix} -1 \quad (30)$$

$$\beta_{w0} = \frac{\triangle w_0}{\triangle a_c} \quad (31)$$

$$\beta_m = \begin{Bmatrix} \frac{\triangle a_m}{\triangle a_c}, \forall\, m \in S \\ 0, \quad \forall m \notin S \end{Bmatrix} \tag{32}$$

In the above formulas, **β** refers to the gradient of all optimal solutions, denoted as **αc** that is tasked with regulating model revisions. β_{w0} refers to the sensitivity of changes in the weight as corresponds to $\triangle \alpha_c$, while β_m refers to the sensitivity of $\triangle \alpha_s$ as corresponds to $\triangle \alpha_c$; thus, we can replace formula 26 with row 1 and 3 of formula 27 to obtain the formula 33.

$$\begin{bmatrix} \triangle g_c \\ \triangle g_p \end{bmatrix} = \gamma_m \triangle \alpha_c, \forall m \in P \cup c \tag{33}$$

$$\gamma_m = \begin{Bmatrix} K_{mc} + \sum_{i \in S} K_{mi}\, \beta_i, \\ 0, \end{Bmatrix} \begin{matrix} \forall m \notin S \\ \forall m \notin S \end{matrix} \tag{34}$$

A new update strategy is introduced to circumvent mathematical limits by referring to the largest possible change in α_s as a migrant point that will be moved to another subset. This strategy is made possible under four conditions, as follows:

- Weight α_i in reaches the lower bound 0 or the upper bound C, when ε is the small figure and $\triangle \alpha_c^s$ represents the greatest incremental jump as follows in formula 35

$$\triangle \alpha_c^s = \begin{cases} \frac{C - \alpha_i}{\beta_i}, if\ \beta_i > \varepsilon \\ \frac{\alpha_i}{\beta_i}, if\ \beta_i < \varepsilon \end{cases} \tag{35}$$

- A weight reaches the lower or upper boundary in formula 36:

$$\Delta \alpha_c^p = \frac{-g_i}{\Upsilon_i} \tag{36}$$

- A gradient denoted as x_c reaches zero in formula 37, or its weights equate to C as formula 38:

$$\Delta\alpha_c^g = \frac{-g_i}{\Upsilon_c} \tag{37}$$

$$\Delta\alpha_c^a = C - \alpha_c \tag{38}$$

Finally, we take the smallest of the four case values to compute the maximum feasible increment of α_c as outlined in formula 39, following which the inverse matrix Q is recalculated with every addition using the Woodbury formula, included in formula 40, while formula 41 is used to calculate datapoint exclusions from the matrix.

$$\Delta\alpha_c^{max} = \min(\Delta\alpha_c^s, \Delta\alpha_c^p, \Delta\alpha_c^g, \Delta\alpha_c^a) \tag{39}$$

$$Q = \begin{bmatrix} Q & 0 \\ 0 & 0 \end{bmatrix} + \frac{1}{\Upsilon_c}\begin{bmatrix} \beta^2 & \beta \\ \beta & 1 \end{bmatrix} \tag{40}$$

$$Q_{i,j} = Q_{i,j} - Q_{s,s}^{-1} Q_{i,s} Q_{i,j}, \quad \forall i,j \epsilon s;\ i,j \neq s \tag{41}$$

The LG-SVM is presented in Algorithm 1 as pseudo-code. As outlined below, the incorporation of the Woodbury formula has scaled the matrix θ with O(D2), which represents data points' total attributes and N the number of training points. Computational complexity equites to $O(N_s^2), with\ N_s \leq N$.

Algorithm one:

1: Inputs: $X[x_1, \ldots, X_n],\ Y = [y_1, \ldots, y_n]$

2: Loading: K: Number of nearest neighbours

3: Read a new input: x_c

4: Set a_c to zero

5: Compute θ using (9)

6: Compute the kernel matrix according to (16-17)

7: Compute g_c according to (18)

8: While $g_c < 0 \; and \; a_c < C \; do$

9: Compute β using (23)

10: Compute γ using (28)

11: Compute $\triangle \alpha_c^{max} using \; (33)$

12: $\alpha_c = \alpha_c + \triangle \alpha_c^{max}$

13: $\alpha_s = \beta_s \triangle \alpha_c^{max}$

14: $w_0 = \beta_{w0} \triangle \alpha_c^{max}$

15: $g_{c,p} = \gamma \triangle \alpha_c^{max}$

16: Let j be the index of point $\triangle \alpha_c^{max}$

17: if j $\in S \; then$

18: If $\alpha_j = 0$ then

19: Move $x_j \; to \; N$

20: else

21: Move $x_j \; to$ E

22: end if

23: else if j $\in P \; then$

24: Move $x_j \; to \; S$

25: else

26: j = c, terminate

27: end if

28: Update Q

29: end while

30: Return a_j, w_0, S, E, N

3.3.4 Autoencoder

In a simple autoencoder, the bottom and top layers have the same number of neurons, while the middle layer usually has a fewer number of neurons with a number of weights connecting the bottom and middle layer, denoted as W_1, and connecting the middle and top layer, denoted as W_2. Thus, the input X is fed through the bottom layer to the middle with the following function, 42, and output function of 43:

$$h = \sigma(W_x + b) \tag{42}$$

$$\tilde{X} = \sigma(W^T h + c) \tag{43}$$

The autoencoder works to minimize the deviation between $\tilde{X}$ and X by compressing the data in the middle layer, denoted as h, creating the following formulas, 44 and 45:

$$h = C(x) \tag{44}$$

$$\tilde{X} = \int \left(C\left(x^{(i)}\right)\right) \tag{45}$$

When combined, the encoder and decoder give formula 46, with error in formula 47 (Choi, 2017).

$$X^{(i)} = \int \left(C\left(x^{(i)}\right)\right) \tag{46}$$

$$E = \frac{1}{2}\sum_i \sum_k |\tilde{X}^i - x^i|^2 \tag{47}$$

3.3.5 Multilayer Perceptron

The multilayer perceptron is comprised of an input layer, hidden layer, and an output layer. The input layer accepts the data that will later undergo learning, while the output later is the post-processed data that is utilized to categorize the data. The hidden layer, which may be multiple, is where the learning takes place, during which key patterns are learned and passed on to the next layer. The hidden layer identifies vital information to

pass forward to achieve efficiency. The activation function learns non-linear relationship and pinpoints key information used during the categorization process. The activation function learns from sigmoid function, which assess information using mathematical computation of a sigmoid activation function. Sigmoid function is a logistic function outlined in formula 48 with a range limited to 0 and 1, where the output closer to 0 is dropped. The sigmoid function has an S-curve function, which allows for non-linear learning boundaries useful in reducing the sensitivity of predicting complex datasets such as text. The x value is shared in formula 49, where W refers to the weights.

$$S(x) = \frac{1}{1+e^{-x}} \tag{48}$$

$$x = W_1 \times X_1 + W_2 \times X_2 + W_3 \times X_3 \tag{49}$$

The above covers the feedforward part of the perceptron model, the model is later added a dimension of flexibility by a feed backward propagation, in which the perceptron model learns by calculating its error in order to adjust. During the backward phase, the perceptron model minimizes the error through a loss function depicted in formula 50, where the error turns to formula 51, in which O is the subtraction of the error.

$$Loss\ function = \frac{(Y-\hat{Y})^2}{2} \tag{50}$$

$$ERROR = \frac{(Y-O)^2}{2} \tag{51}$$

Later the coefficients or the input variables, W_{is} and the biases, B_{h0} and B_{hi} are revised utilizing the Gradient descent algorithm through the subsequent equation shared in formula 52, where W_{new} refers to the new weights, W_{old} the old weights, a the learning rate, and $\frac{d_e}{d_w}$ the derivative of the error.

$$W_{new} = W_{old} - (a \times \frac{d_e}{d_w}) \quad (52)$$

Later, the perceptron model determines the contribution of each weight and biases to the error term by defining the error's rate of change in relation to the weights and bias parameters through a chain rule in formula 53.

$$\frac{d_e}{d_{W_{ho}}} = \frac{d_e}{d_o} \times \frac{d_o}{d_{z_2}} \times \frac{d_{z_2}}{d_{W_{ho}}} \quad (53)$$

Following the computation of each partial derivatives influence, the partial influence is then taken in terms of what is shared in formula 54 and is calculated in pertinence to the output using formula 55 and Z_2in formula 56.

$$E = \frac{(Y-O)^2}{2} \quad (54)$$

$$\frac{d_e}{d_o} = \frac{2\times(Y-O)(-1)}{2} = (O - Y) \quad (55)$$

$$\frac{d_o}{d_{z_2}} = \sigma(Z_2) \times (1 - \sigma_{Z_2}) \times (1 - \sigma_{Z_2}) = O \times (1 - O) \quad (56)$$

The partial derivative of Z_2 with respect to W_{ho} shared in formula 57

$$\frac{d_e}{d_{W_{ho}}} = (O - Y) \times O \times (1 - O) \times h_1 \quad (57)$$

Likewise, the contribution of each partial parameters is calculated with the bias between hidden and the output layer or input layer demonstrated in formula 58 and 59. As a result, after plugging in the partial derivative value, the equation turns to formula 60

$$\frac{d_e}{db_{ho}} = \times \frac{d_o}{d_{z_2}} \times \frac{d_{z_2}}{db_{ho}} \quad (58)$$

$$\frac{d_e}{db_{ho}} = (O - Y) \times O \times (1 - O) \times 1 \quad (59)$$

$$\frac{d_e}{d_{W_{jh}}} = (O - Y) \times O \times (1 - O) \times W_{ho} \times h_1(1 - h_1) \times X \quad (60)$$

Finally, the parameters are updated using formula 61 through 64.

$$W_{ih} = W_{jh} - (a \times \frac{d_e}{dW_{jh}}) \quad (61)$$

$$b_{ih} = b_{jh} - (a \times \frac{d_e}{db_{jh}}) \quad (62)$$

$$W_{ho} = W_{ho} - (a \times \frac{d_e}{dW_{ho}}) \quad (63)$$

$$b_{ho} = b_{ho} - (a \times \frac{d_e}{db_{ho}}) \quad (64)$$

Chapter Four

Results, Discussion, Conclusion and Recommendations

4.1 Results

4.1.1 Raw Data

4.1.2 Data Transformation

4.1.3 Stock Market

4.2 Discussion

4.3 Conclusion and Recommendations

4.1Results

4.1.1 Raw Data

This section will overlook the model's evaluation in terms of prediction accuracy as well as the time it takes to build the model in a comprehensive manner. To achieve such comprehension, the LG-SVM model will be compared to other similar classifiers such as the batch kernel support vector machine (KSVM) (Mohammed, Noaman, & Azzai, 2022). The KSVM introduces a layer of kernel function to the SVM model to the sequence data, enabling the algorithm to function a high-dimensional spaces through the computation of inner products of all data pairs in the feature space rather data coordinates. The second model is the incremental kernel non-parametric support vector machine (IKSVM) (Maji, Malik, & Berg, 2008). The IKSVM and KSVM model is an SVM model where the learning occurs incrementally through the use of an algorithm that is flexible and updates along the process; however, the learning takes places after the batch is processed.

Additionally, the two above mentioned models are also compared to their updated version: the batch kernel non-parametric support vector machine (KNSVM). The KNSVM incorporates a nonparametric support vector classifier, coupling data distribution knowledge from the kernel support vector machine (IKSVM) (Salhi, Ksantini, & Zouari, 2022) and the kernel nonparametric discriminant (KND) (Li, Zhou, & Wei, 2017), in order to capitalize from near-global variations. When trained incrementally, the model is referred to as incremental batch kernel non-parametric support vector machine (IKNSVM)

The model was tested using the 10-fold cross validation process, under which the dataset was divided into 10 subgroups and, of which one was held out to test. The accuracy score is the average of all 10 accuracy scores.

The area under the curve (AUC) was also utilized. The AUC is a derivative of the receiver operating characteristic (ROC) curve, under which trade-offs between the true positive rate and the false positive rate is illustrate (Hajian-Tilaki, 2013). The AUC follows the same aforementioned 10-fold cross validation process. All data processed through the model were limited to vectors of 130 words.

Results related to model accuracy is shared in Table I, while training times are shared in Table II. The average score illustrated in Table I averaged at 79% with the LG-SVM recording the highest at 79.712% and the KSVM the lowest at 78.979%. This is a relatively high accuracy score compared to the regular Bi-LSTM that has recorded an average of 52% accuracy score. The training process in Table II oversaw the training of a total of 3741 sample in each round. Batch processing, column three and four, took a longer time to train since the model is only processed once all batches are completed.

Table I-4: Average AUC and Average Training Time (in seconds) of each classifier for the original data

Classifier	Average AUC	Average training time
LG-SVM	79.712	0.2533
KNSVM	79.431	338.6508
IKNSVM	79.455	0.5386
KSVM	78.979	4.7579
IKSVM	79.238	0.2444

Table II-4: Required training Times for the LG-SVM vs different classifiers (in seconds)

LG-SVM	IKNSVM	IKNSVM	KNSVM	KSVM
0.03066	1.5475	0.47015	238.5659	0.47015
0.024035	0.82895	0.81168	368.7075	4.9631
0.02433	0.53863	0.02094	706.2941	8.0256
0.030496	0.49988	0.05545	1073.1679	12.6043

0.030352	0.9582	0.01929	1412.6015	18.0922
0.03026	0.2965	0.02248	1749.4815	23.0433
0.028874	0.6043	0.0512	2010.3415	25.2483
0.022455	0.66271	0.0161	2472.4615	28.1733
0.029001	0.66944	0.02094	2810.0515	31.7876

4.1.2 Data transformation

In order to increase prediction accuracy, a new model with an autoencoder was used, which was lacking in the model mentioned in 4.1.1. The models with the autoencoder performed better since the autoencoder extracted sequential data used to enhance the learning process. Three autoencoder models were created with dimensional reduction of 100, 130, and 70, as outlined in Table III. The results demonstrated in Table III follow a U-shaped curve in which a dimensional reduction of 130 had a relatively high accuracy score, which took a dip at 100, only to increase again at 70.

Table III-4: Average AUC and Average Training Time (in seconds) of each classifier

	Classifier	Average AUC	Average training time
Dimensional reduction of 100	LG-SVM	99.75	2.4086
	KNSVM	99.68	639.966
	IKNSVM	99.73	6.692
	KSVM	99.31	50.0272
	IKSVM	99.39	1.644
Dimensional reduction of 70	LG-SVM	99.91	0.1797
	KNSVM	99.89	349.2133
	IKNSVM	99.904	6.151
	KSVM	99.859	0.49398
	IKSVM	99.89	0.029763

Dimensional reduction of 130	LG-SVM	99.86	0.2957
	KNSVM	99.80	387.8444
	IKNSVM	99.84	0.43334
	KSVM	99.76	2.6575
	IKSVM	99.83	0.2102

Table IV demonstrates the time it took to build the models. As expected, Batch processing, column three and four, took a longer time to train for the same aforementioned reason. The iteration of 70 took the fastest time to train, followed by 100 and 130.

Table IV-4: Required training Times for the LG-SVM vs different classifiers (in seconds)

LG-SVM	IKNSVM	IKSVM	KNSVM	KSVM
Dimensional reduction of 70				
0.20634	9.2123	0.059016	350.335	0.21143
0.24455	1.2679	0.021878	699.6986	0.47458
0.18545	5.4905	0.10818	1049.7615	0.63481
0.20002	2.4086	0.013309	1398.3695	0.91791
1.17704	8.5949	0.024096	1747.9796	1.3718
0.15755	3.3412	0.018002	2098.8932	1.7535
0.07111	9.1344	0.016572	2447.9545	2.0264
0.079729	4.4733	0.012405	2788.5784	2.4378
0.076965	6.0555	0.039655	3140.2585	3.0429
Dimensional reduction of 100				
0.03066	1.5475	0.47015	238.5659	0.47015
0.024035	0.82895	0.81168	368.7075	4.9631
0.02433	0.53863	0.02094	706.2941	8.0256
0.030496	0.49988	0.05545	1073.1679	12.6043
0.030352	0.9582	0.01929	1412.6015	18.0922
0.03026	0.2965	0.02248	1749.4815	23.0433

0.028874	0.6043	0.0512	2010.3415	25.2483
0.022455	0.66271	0.0161	2472.4615	28.1733
0.029001	0.66844	0.02094	2810.0515	31.7876
Dimensional reduction of 130				
0.3724	0.5256	0.1698	384.2309	1.3553
0.3275	0.6523	0.1303	745.1588	1.4737
0.0983	0.2364	0.0727	0.0014927	1.5802
0.3026	0.4521	0.2745	0.0018805	1.9422
0.3003	0.5852	0.1702	000.2000359	2.0633
0.1319	0.7854	0.4341	0.00281	2.2552
0.0977	0.3244	0.0708	0.0029933	2.3642
0.1689	0.2362	0.0926	0.0030728	2.4452
0.2144	0.5623	0.4770	0.0036806	2.7143

These results are significantly better than the standard SVM model, which recorded an average validation accuracy of 52.4%, and validation loss of 69.2%, utilizing the same 10-fold cross validation method.

4.1.3 Stock Market

The perceptron model has kept par with the bot detection results, in the sense that the data reduction to 70 has yielded the best performance with a correlation coefficient of 96.58%, meaning that there is a strong statistical correlation between the predicted and the actual values. This is also reflected by a low level of mean absolute error, which measures the average magnitude of the errors. The perceptron model recorded a mean absolute error of 8.9. Likewise, the relative absolute error of the mode, the deviation between prediction and actual output, was also low at 18.4% The errors, when normalized for the sample, also produced a low error rate, with a relative absolute error of 26.6%. The low error values recorded coupled with a high correlation coefficient is proof that the perceptron model is

capable of making prediction at a high accuracy level. Moreover, the stronger performance between the incorporation of an autoencoder and against the 130 dataset is reflected by the root relative squared error of 26.6% versus 81% this means that the model is capable closer to the actual value, since it is normalized for the actual output. The findings in Table V demonstrates the aforementioned statistical outcomes.

Table V-4: Average Performance Metric and Average Training Time (in seconds) of each classifier

	Correlation Coefficient	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Root Relative Squared Error	Time Taken to Build the Model in Seconds
70	0.9658	6.0042	8.9126	18.4108 %	26.6149 %	189.87
100	0.9538	7.2202	9.9978	22.9191 %	30.4901 %	284.3
130	0.4662	25.6227	33.996	33.996%	81.3292 %	462.68

4.2Discussion

Although recording almost similar accuracy scores, incremental learning methods are always preferred to batch processing in such dynamic environments. Batch learning models do not incorporate the latest information available as the model is shaped using N number of examples, only to drop old examples when its memory is depleted. As a result, the incremental model is preferred to the batch since it holds its accuracy by incorporating the latest example with no down time, which is an added value in dynamic environments (Read, Bifet, Pfahringer, & Holmes).

Despite the high accuracy rates, researchers have pointed out that there is no assurance for consistency in dynamic models, particularly when the models are continuously shaped by and measured against evolving data. Thus, observations that are either correctly or incorrectly classified by the current form of the model may be incorrectly or correctly classified by the newer format (Carbonara & Borrowman, 1998).

Moreover, the batch KSVM model has held the worst record when it comes to model accuracy, as expected. This is due to the fact that all other models, which it is compared to, are, arguably, subsequent models that has addressed the KSVM's shortcomings by computing the near-global variational in the input rather than the feature space.

The research has centered on analyzing the models' performance in terms of its accuracy; however, an in-depth comparison of the models' components would provide a stronger argument for the use of the LG-SVM. The in-depth comparison is not conducted at this time since it challenging to investigate all the vast parameters involved; however, a recommendation is being made for future researchers to explore.

Future researchers are also encouraged to investigate the models' performance in terms of false positive and false negative to compute the business cost of implementing the model. A false positive would lead to barring human users from Twitter, which is costly to a business that depends on user-generated content to expand.

The models used to build and assess outcomes suffer from limited data. The data used in this research is outdated since it is collected between 2009 and 2014. This is a relatively old dataset since, as chapter one and two demonstrated, Twitter is a highly dynamic environment with data evolving on a daily basis. Thus, a recommendation is also being made for data to be collected in real-time to ensure that the performance is held at a high level in current terms.

In terms of model building time, the incremental learning methods addresses batch processing's time inefficiencies. By incrementally incorporating new observations, rather than building a new model from scratch, allows for a smoother and more time efficient learning method. Additionally, subsequent incremental models' modification of the core model outlines the relationship between the current and past models; thus,

analyzing the changes gives researchers the opportunity to spot trends overtime (Carbonara & Borrowman, 1998).

The dynamic models have recorded far better accuracy score than the standard SVM model, as witnessed an estimate 20% increase in accuracy. This is in line with what was found in the literature review, which argued that standard models are unable to detect evolving bots due to its dependence on a set of baseline characteristics. The model used in this research is like the upgraded models that have addressed shortcomings in terms of its structure as well as its accuracy scores. The literature review attributed the enhanced accuracy to the models' ability to detect and learn characteristics as it goes rather than fixating them, which has been found to be true here. The LG-SVM model's autoencoder served as a filter that learned the characteristics, and, as thus, proved to increase accuracy.

Turning to the stock market analysis, results are in line with the literature review, which have argued that Twitter shapes public sentiment, which spills over into investment behavior. However, correlation does not indicate causation as the stock market may be influenced by a vast number of factors. The time factor was not incorporated in the perceptron model; therefore, time lag, which takes places between information dissemination over Twitter and human action, is not considered. Stock markets close over the weekend, but Twitter continues to operate continuously with no specific measure set for the time lag (Nisar & Yeung, 2018).

Additionally, not all accounts are found to be equal in terms of their impact on readers. It is found that The New York Times, Bloomberg, CNN and Investing.com have a high level of impact on the stock market, offering a very high correlation between sentiments and market volatility (Valle-Cruz, Sandoval-Almazán , López-Chau, & Fernandez-Cortez, 2021). Thus, it is important to look into the meta data to distinguish not only bots from humans, but also their level or influence.

It is worth noting that despite proving a possible relation, grounds for speculation expand far beyond Tweeter. Users who depend on social media for information are more than likely to visit various platforms, such as Instagram; thus, any future model must include other platforms.

The relation between Tweets and stock market volatility that is captured by the perceptron model might not reflect the reality of the situation since it did take into consideration other potential contributing factors. Therefore, future works should consider including other important economic factors such as the money supply and prices of other investment goods such as gold. Moreover, this study has not considered political factors that may impact the market. President Trump and Elon Musks are both widely known for their influential tweets that are known to influence the market; therefore, their Tweets must be controlled for in future circumstances.

4.3 Conclusion and Recommendations

This research has explored the benefits of applying a flexible and dynamic learning algorithm to bot detection techniques, alongside exploring the relation between Tweets authored by humans or bots on stock market volatility. The analysis has provided sufficient evidence supporting the notion that bot detection in a highly dynamic environment requires a model that is flexible enough to learn at a high pace and adapt efficiently as witnessed in the different accuracy score recorded between the LG-SVM and the SVM, a nearly 20% improvement. Additionally, the research has demonstrated that Twitter has influence over public consensus, which, in return, influences market volatility.

Contributions are as follows: The research has proven that flexibility must be considered in the learning algorithm to detect bots, particularly since bots are evolving alongside human interactions. Second, incremental learning algorithms are more time efficient in detecting bots and do create a trail that allows for analyzing trends. Third, the research provided proof of the autoencoder's advanced ability to learn non-linear patterns in large dataset and increase efficiency through dimension reduction, which also reduces noise by learning relevant information as witnessed by the higher scores in the performance metrics and the reduced learning time (Sayantini, 2020).

Moreover, the perceptron model has proven to accurately predict short-term stock market volatility resulting from Tweeter trends. The main benefit of applying such a model is its ability to learn non-linear relationships that were able to determine a clear and strong relationship between the two variables in comparison to a simple linear model, which has proven to be unviable as there is no linear relationship. The perceptron model is also proven to be the best fit for stock market prediction as it works well with highly volatile datasets, which the stock market volatility is.

This research can be extended by, particularly in capturing Tweets' influence on the stock market, by analyzing other financial indicator, global market trends, and deep diving into each accounts' reach to distinguish far-reaching accounts' influence from those with limited reach. Finally, data from other social media platforms such as Instagram should also be considered.

References

A. Loureiro, V. M. (2018). Exploring the Use of Deep Neural Networks for Sales Forecasting in Fashion Retail. *Decision Support Systems, 114*, 81-93.

A. Temür, M. A. (2019). Predicting Housing Sales in Turkey Using ARIMA, LSTIM and Hybrid Models. *Journal of Business Economics and Management , 20*(5), 920-938.

Australian Government. (2021, June 23). *Social media for business*. Retrieved Oct 18, 2021, from Australian Government: Business: https://business.gov.au/online/social-media-for-business

Bandyopadhyay, H. (2021, Nov 29). *Computer Vision: Everything You Need to Know*. Retrieved Jan 15, 2022, from V7labs: https://www.v7labs.com/blog/what-is-computer-vision

Basnat, A., Ahmed, A., El Ghazali, A., & Jain, V. (2014). Incremental Learning Approach for Enhancing the Performance of Multi-Layer Perceptron for Determining Stock Market Trends. *IJSBAR*, 15-23.

Brownlee, J. (2019, Jul 05). *A Gentle Introduction to Computer Vision*. Retrieved Jan 15, 2022, from Machine Learning Mastery: https://machinelearningmastery.com/what-is-computer-vision/

Brydon, A. (2019, Oct 30). *Why AI Needs Human Input (And Always Will)*. Retrieved Jan 15, 2022, from Forbes: https://www.forbes.com/sites/forbestechcouncil/2019/10/30/why-ai-needs-human-input-and-always-will

Carbonara, L., & Borrowman, A. (1998). A Comparison of Batch and Incremental Supervised Learning Algorithms. *Springer*, 264-272.

Cboe Volatility Index. (n.d.). *External.*

Choi, H. I. (2017). *Lecture 16: Autoencoders*. Retrieved from Seoul National University:

http://www.math.snu.ac.kr/~hichoi/machinelearning/lecturenotes/Autoencoder.pdf

Cresci, S., Pietro, R. D., Petrocchi, M., Spognardi, A., & Tesconi, M. (2015, September 14). Fame for sale: efficient detection of fake Twitter followers. *Decision Support Systems*, 56-71.

Cresci, S., Pietro, R. D., Petrocchi, M., Spognardi, A., & Tesconi, M. (2017). The paradigm-shift of social spambots. *Proceedings of the 26th International Conference on World Wide Web Companion*, 3-7.

Daniel, L., Loyola-González, O., Monroy, R., & Medina-Pérez, M. A. (2021). Bot Datasets on Twitter: Analysis and Challenges. *Applied Sciences*.

Demush, R. (2019, Feb 26). *A Brief History of Computer Vision (and Convolutional Neural Networks)*. Retrieved Jan 15, 2022, from Hackernoon: https://hackernoon.com/a-brief-history-of-computer-vision-and-convolutional-neural-networks-8fe8aacc79f3

Dickson, B. (2020, Jan 06). *What are convolutional neural networks (CNN)?* Retrieved from TechTalks: https://bdtechtalks.com/2020/01/06/convolutional-neural-networks-cnn-convnets

Gavali, P., & Banu, J. S. (2019). Chapter 6 - Deep Convolutional Neural Network for Image Classification on CUDA Platform. In *Deep Learning and Parallel Computing Environment for Bioengineering Systems* (pp. 99-122). Academic Press. doi:10.1016/B978-0-12-816718-2.00013-0

Hajian-Tilaki, K. (2013). Receiver Operating Characteristic (ROC) Curve Analysis for Medical Diagnostic Test Evaluation. *PubMed Central*, 627–635.

Hartley, R. (2004). *Multiple View Geometry in Computer Vision.* Cambridge University Press.

Hendrix, J. (2021, Oct 6). *How Facebook is Misleading the Public About Its Role in January 6*. Retrieved Oct 18, 2021, from Just Security: https://www.justsecurity.org/78494/how-facebook-is-misleading-the-public-about-its-role-in-january-6/

Hostiadi, D. P., Tohari, A., & Wibisono, W. (2020). B-Corr Model for Bot Group Activity Detection Based on Network Flows Traffic Analysis. *KSII Transactions on Internet and Information Systems (TIIS), 14*(10), 4176-4197.

Hutchinson, A. (2022, February 16). *Twitter Launches New Bot Labels to Identify Bot Accounts In-Stream*. Retrieved from Social Media Today: https://www.socialmediatoday.com/news/twitter-launches-new-bot-labels-to-identify-bot-accounts-in-stream/618971/

Jang, B., Kim, M., Harerimana, G., Kang, S.-u., & Kim, J. W. (2020, Aug 24). Bi-LSTM Model to Increase Accuracy in Text Classification: Combining Word2vec CNN and Attention Mechanism. *MDPI*, 1-14.

Korteling, J. E., Boer-Visschedijk, G. C., Blankendaal, R. A., Boonekamp, R. C., & Eikelboom, A. R. (2021). Human- versus Artificial Intelligence. *Frontiers in Artificial Intelligence, 4*. doi:10.3389/frai.2021.622364

Kretinin, A., Samuel, J., & Kashyap, R. (2018). When the Going Gets Tough, The Tweets Get Going! An Exploratory Analysis of Tweets Sentiments in the Stock Market. *American Journal of Management, 18*(5), 23-36.

Kudugunta, S., & Ferrara, E. (2018, October). Deep Neural Network for Bot Detection. *Information Sciences, 467*, 312-322.

Kuepper, J. (2022, March 1). *What Caused the Russian Financial Crisis of 2014 and 2015*. Retrieved from The Balance: https://www.thebalance.com/what-caused-the-russian-financial-crisis-of-2014-and-2015-1979012

Lewis, C., & Monett, D. (2018). Getting clarity by defining Artificial Intelligence - A Survey. In *Philosophy and Theory of Artificial Intelligence 2017* (pp. 212-214). doi:10.1007/978-3-319-96448-5_21

Lewis, T. (2014, Dec 04). *A Brief History of Artificial Intelligence*. Retrieved Dec 01, 2021, from Live Science: https://www.livescience.com/49007-history-of-artificial-intelligence.html

Li, Q., Zhou, S., & Wei, F. (2017). Local kernel nonparametric discriminant analysis for adaptive extraction of complex structures. *Open Physics*.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision. 2*, pp. 1150-1157. IEEE. doi:10.1109/ICCV.1999.790410

Lutkevich, B., & Wigmore, I. (n.d.). *Social Media*. Retrieved Oct 18, 2021, from Tech Target: https://whatis.techtarget.com/definition/social-media

Macêdo, D. L. (2017). *Enhancing Deep Learning Performance using Displaced Rectifier Linear Unit.* Federal University of Pernambuco. doi:10.13140/RG.2.2.23893.88807

Madiraju, N. S., Sadat, S. M., Fisher, D., & Karimabadi, H. (2018). Deep Temporal Clustering: Fully Unsupervised Learning of Time Domaine Features. *AvLab*.

Mahanta, J. (2017, July 10). *Introduction to Neural Networks, Advantages and Applications*. Retrieved from Towards Data Science: https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207

Maji, S., Malik, J., & Berg, A. (2008). Classification using intersection kernel support vector machines is efficient. *IEEE Conference on*

Computer Vision and Pattern Recognition (pp. 1-8). IEEE Conference on Computer Vision and Pattern Recognition.

McKenzie, H.-W., Curtis, B., Leggio, L., Devoto, A., Giorgi, S., David, E. H., . . . Rahman, M. (2021, May 20). Bots and Misinformation Spread on Social Media: Implications for COVID-19. *Journal of Medical Internet Research, 23*(5).

Mohammed, A., Noaman, M., & Azzai, H. (2022). A combining two ksvm classifiers based on True pixel values and Discrete wavelet transform for mri-based brain tumor detection and Classification. *Engineering and Technology Journal*, 322-333.

Nisar, T. M., & Yeung, M. (2018). Twitter as a tool for forecasting stock market movements: A short-window event study. *The Journal of Finance and Data Science, 4*(2), 101-119. doi:https://doi.org/10.1016/j.jfds.2017.11.002

Prince, S. J. (2012). *Computer Vision: Models, Learning, and Inference.* Cambridge University Press.

Randhawa, R. H., Aslam, N., AlAuthman, M., Rafiq, H., & Comeau, F. (2021, June 3). Security Hardening of Botnet Detectors Using Generative Adversarial Networks. *IEEE Access, 9*.

Read, J., Bifet, A., Pfahringer, B., & Holmes, G. (n.d.). *Batch-Incremental vs. Instance-Incremental Learning in Dynamic and Evolving Data.* New Zealand: The University of Waikato.

Rhodes, A. D., Quinn, M. H., & Mitchell, M. (2016). Fast On-Line Kernel Density Estimation for Active Object Localization. *Computer Vision and Pattern Recognition*, 1611.05369. Retrieved from https://arxiv.org/abs/1611.05369

Roberts, S. (2020, June 16). *Who's a Bot? Who's Not?* Retrieved from New York Times: https://www.nytimes.com/2020/06/16/science/social-media-bots-kazemi.html

Salhi, M., Ksantini, R., & Zouari, B. (2022). A Real-Time Image-Centric Transfer Function Design Based on Incremental Classification. *Journal of Real-Time Image Processing.*

Savage, N. (2020, Dec 09). *The race to the top among the world's leaders in artificial intelligence*. Retrieved Jan 15, 2022, from Nature: https://www.nature.com/articles/d41586-020-03409-8

Sayantini. (2020, May 14). *Autoencoders Tutorial : A Beginner's Guide to Autoencoders*. Retrieved from Edureka: https://www.edureka.co/blog/autoencoders-tutorial/

Soula, A., Tbarki, K., Ksantini, R., Said, S. B., & Lachiri, Z. (2020). A novel incremental Kernel Nonparametric SVM model (iKN-SVM) for data classification: An application to face detection. *Engineering Applications of Artificial Intelligence.*

Stevens, K. (2012). The Vision of David Marr. *Perception, 41*, 1061-1072. doi:10.1068/p7297

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications.* Springer.

T. Boone, R. G. (2015). Incorporating Google Trends Data Into Sales Forecasting. *The International Journal of Applied Forecasting*, 9-14.

Tian, Z., Shen, C., Chen, H., & He, T. (2019). FCOS: Fully Convolutional One-Stage Object Detection. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (pp. 9626-9635). IEEE. doi:10.1109/ICCV.2019.00972

Udom, P. (2014). A comparison study between time series model and ARIMA model for sales forecasting of distributor in plastic industry . *IOSR Journal of Engineering, 4*(2), 32-38.

UW Bothell and Cascadia College: Campus Library. (2021, Oct 5). *News: Fake News, Misinformation & Disinformation*. Retrieved Oct 18,

2021, from UW Bothell and Cascadia College:: https://guides.lib.uw.edu/c.php?g=345925&p=7772376

Valle-Cruz, D., Sandoval-Almazán , R., López-Chau, A., & Fernandez-Cortez, V. (2021). Does Twitter Affect Stock Market Decisions? Financial Sentiment Analysis During Pandemics: A Comparative Study of the H1N1 and the COVID-19 Periods. *Springer Link*, 372–387.

victoriaruss. (2013, Mar 13). *Feature Detectors/ Hubel & Wiesel Experiment/ Prosopagnosia*. Retrieved Jan 15, 2022, from goodpsychology: https://goodpsychology.wordpress.com/2013/03/13/235/

Wang, P. (2019). On Defining Artificial Intelligence. *Journal of Artificial General Intelligence, 10*(2), 1-37. doi:10.2478/jagi-2019-0002

Zhou, H. P. (2020). Study on convolutional neural network and its application in data mining and sales forecasting for E- commerce . *Electronic Commerce Research, 20*(2), 297- 320.

5.0 Appendix I: Code

5.0.1 Data exploration and Bi-LSTM (google Collaboratory)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files
uploaded = files.upload()
import io
df = pd.read_csv(io.BytesIO(uploaded['df.csv']))
def get_times(df):
  df['timestamp'] = pd.to_datetime(df['timestamp'])
  df['Year'] = df['timestamp'].apply(lambda x: x.year)
  df['Hour'] = df['timestamp'].apply(lambda x: x.hour)
  df['Day of Week'] = df['timestamp'].apply(lambda day:day.dayofwe
ek)
  dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
  df['Day of Week'] = df['Day of Week'].map(dmap)
  df['Date'] = df['timestamp'].apply(lambda time:time.date())
  return df
df = df.drop(labels='source', 'url', 'turncated', 'in_reply_to_sta
tus_id', 'in_reply_to_user_id', 'in_reply_to_screen_name', 'retwee
ted_status_id', 'geo', 'place', 'contributors', 'possibly_sensitiv
e', 'created_at', 'crawled_at', 'updated', axis=14)
df['Year'].value_counts()
tempdf = pd.concat([df['Hour'],df['Day of Week'],pd.DataFrame(np.z
eros((2839362,),dtype = int),columns=['BOT'])\
                    , pd.DataFrame(np.ones((2839362,),dtype = int),
columns=['e']), df['Date'],df['Year']],axis = 1)
# groupby  hour
g_byHour = tempdf[tempdf['BOT'] == 0].groupby(by = ['Day of Week',
 'Hour'])['e'].sum().unstack()
# Heatmap
fig, axs = plt.subplots(ncols=2,figsize=(24,7))
sns.heatmap(g_byHour,linewidths=1,cmap = 'coolwarm',ax = axs[0])
axs[0].title.set_text('Users')
# by date
fig, axs = plt.subplots(ncols=2,figsize = (20,7))
g_bydate = tempdf.groupby('Date')['e'].sum()
plt.tight_layout
# date-wise tweets posted by genuine and Traditional bots
g_bydate.plot()
# Feature co relation
df.drop(labels = ['Date', 'text'],axis = 1).corr(method='spearman'
)
```

```
plt.figure(figsize=(12,7))
sns.heatmap(df.drop(labels = ['Date', 'text'],axis = 1).corr(metho
d='spearman'), cmap='Blues', annot=True)
plt.tight_layout()
#plt.savefig(save_plot+'/trad_bots/1_Feature_corrG',dpi = 300)
# distplots
fig, axs = plt.subplots(ncols=2,figsize = (22,7))
sns.distplot(tempdf[tempdf['BOT'] == 0]['Hour'],ax = axs[0],color
= 'Green')
axs[0].title.set_text('Users')
sns.countplot(df['Day of Week'])
# date-wise tweets posted by genuine and spam bots
fig, axs = plt.subplots(ncols=2,figsize = (20,7))
g_bydate.plot(ax = axs[0])
axs[0].title.set_text('Genuine Users')
# Create a function to clean the tweets using library re
def cleanTweet(doc):
    doc = re.sub("["u"\U0001F600-\U0001F64F""]",'',doc)
    doc = re.sub("["u"\U0001F300-\U0001F5FF""]",'',doc)
    doc = re.sub("["u"\U0001F680-\U0001F6FF""]",'',doc)
    doc = re.sub("["u"\U0001F1E0-\U0001F1FF""]",'',doc)
    doc = re.sub('@[A-Za-z0-9_]+','',doc)
    doc = re.sub('RT[\s]+','',doc)
    doc = re.sub('#','',doc)
    doc = re.sub('https?:\/\/\S+','',doc)
    doc = re.sub('&amp;','and',doc)
    doc = re.sub(' +',' ',doc)
    doc = re.sub("[\(\[].*?[\)\]]","",doc)
    doc = re.sub("\n"," ", doc)
    doc = re.sub("^[^A-Za-z0-9]+","",doc)
    doc = doc.strip()
    doc = doc.lower()
    return doc
# Clean the tweets
df['text'] = df['text'].apply(cleanTweet)
# Create a document
all_words = ' '.join([tweet for tweet in df['text']])
# Create a WordCloud object
wordCloud = WordCloud(background_color='black',
                      width=800, height=300,
                      random_state=21, max_font_size=110,
                      max_words=200,
                      contour_color='black',
                      contour_width=1).generate(all_words)
# Plot WordCloud object
plt.figure(figsize=[8,8])
plt.imshow(wordCloud, interpolation="bilinear")
plt.axis('off')
```

```
plt.show()
#Libraries for Sentimental analysis
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from textblob import TextBlob
# Create polarity and subjectivity columns
df['sentiment'] = df['text'].apply(lambda x: TextBlob(x).sentiment
)
df['polarity'] = df['sentiment'].apply(lambda x: x[0])
df['subjectivity'] = df['sentiment'].apply(lambda x: x[1])
# Create a function for transforming polarity
def transform_sentiment(value):
    if value < 0:
        sentiment = "negative"
    elif value > 0:
        sentiment = "positive"
    else:
        sentiment = "neutral"
    return sentiment
# Insert Sentiment into data frame
df["sentiment"]=df["polarity"].apply(
    transform_sentiment)
# Show first six row
df.head(6)
# Proportion of sentiment
df["sentiment"].value_counts(normalize=True)*100
# Proportion of subjectivity
df["subjectivity"].value_counts(normalize=True)*100
# Barplot
df["sentiment"].value_counts().plot(
    kind='bar',alpha=0.75,rot=0,title="Frequency of Sentiment \n")
plt.show()
lengths = df["tweet"].apply(lambda x : len(x.split(" "))) # try me
an + 3*sigma

mean = lengths.mean()
std = lengths.std()
approx_nseq = mean + 3*std
print(f"Mean: {mean:.4f}\nStd: {std:.4f}\nMean+3*Std = {approx_nse
q:.4f}")
N_SEQ = 23
# Fraction of samples to be used for training set
N_TRAIN = 0.9
from keras.preprocessing.text import Tokenizer

def getTokenizer():
    tok = Tokenizer()
    # Create word_index and vocab
    tok.fit_on_texts(df_train["tweet"])
```

```
    return tok
tok = getTokenizer()
word_index = tok.word_index
N_vocab = len(tok.word_index) + 1 # This +1 is used later
print(N_vocab)
from sklearn.feature_extraction.text import CountVectorizer
bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_featur
es=1000, stop_words='english')
# bag-of-words feature matrix
bow = bow_vectorizer.fit_transform(df['tweet'])
from keras.preprocessing.sequence import pad_sequences

def getXY(tok):
    x_train = pad_sequences(tok.texts_to_sequences(df_train["tweet
"]), maxlen = N_SEQ, padding = "post")
    x_val   = pad_sequences(tok.texts_to_sequences(df_val["tweet"]
), maxlen = N_SEQ, padding = "post")
    y_train = df_train["label"].to_numpy(dtype = float).reshape(-
1, 1)
    y_val   = df_val["label"].to_numpy(dtype = float).reshape(-
1, 1)
    return x_train, x_val, y_train, y_val
from keras import Sequential, Model
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
from keras.layers import Layer, Input, Embedding, Conv1D, Bidirect
ional, LSTM, Dense, Dropout, BatchNormalization, GlobalMaxPooling1
D, Flatten
import tensorflow as tf # Only used for various tensor operations

# A more general and complete version of the layer defined in the
linked keras example

class MultiHeadSelfAttention(Layer):
    """ This uses Bahadanau attention """

    def __init__(self, num_heads = 8, weights_dim = 64):
        """ Constructor: Initializes parameters of the Attention l
ayer """

        # Initialize base class:
        super(MultiHeadSelfAttention, self).__init__()

        # Initialize parameters of the layer:
        self.num_heads = num_heads
        self.weights_dim = weights_dim

        if self.weights_dim % self.num_heads != 0:
```

```
            raise ValueError(f"Weights dimension = {weights_dim} s
hould be divisible by number of heads = {num_heads} to ensure prop
er division into sub-matrices")

        # We use this to divide the Q,K,V matrices into num_heads
submatrices, to compute multi-headed attention
        self.sub_matrix_dim = self.weights_dim // self.num_heads
                # Weight matrices for computing query, key and val
ue (Note that we haven't defined an activation function anywhere)
        # Important: In keras units contain the shape of the outpu
t
        self.W_q = Dense(units = weights_dim)
        self.W_k = Dense(units = weights_dim)
        self.W_v = Dense(units = weights_dim)


    def get_config(self):
        """ Required for saving/loading the model """
        config = super().get_config().copy()
        config.update({
            "num_heads" : self.num_heads,
            "weights_dim" : self.weights_dim
            # All args of __init__() must be included here
        })
        return config


    def build(self, input_shape):
        """ Initializes various weights dynamically based on input
_shape """
        input_dim = input_shape[-1]
        self.input_dim = input_dim
        # Weight matrix for combining the output from multiple hea
ds:
        # Takes in input of shape (batch_size, seq_len, weights_di
m) returns output of shape (batch_size, seq_len, input_dim)
        self.W_h = Dense(units = input_dim)


    def attention(self, query, key, value):
        """ The main logic """
        # Compute the raw score = QK^T
        score = tf.matmul(query, key, transpose_b=True)

        # Scale by dimension of K
        dim_key = tf.cast(tf.shape(key)[-
1], tf.float32) # == DIM_KEY
        scaled_score = score / tf.math.sqrt(dim_key)
```

```
                # Weights are the softmax of scaled scores
        weights = tf.nn.softmax(scaled_score, axis=-1)

        # The final output of the attention layer (weighted sum of
 hidden states)
        output = tf.matmul(weights, value)

        return output, weights


    def separate_heads(self, x, batch_size):
        """
            Splits the given x into num_heads submatrices and retu
rns the result as a concatenation of these sub-matrices
        """
        x = tf.reshape(x, (batch_size, -
1, self.num_heads, self.sub_matrix_dim))
        return tf.transpose(x, perm=[0, 2, 1, 3])


    def call(self, inputs):
        """ All computations take place here """

        batch_size = tf.shape(inputs)[0]

        # Compute Q = W_q*X
        query = self.W_q(inputs)  # (batch_size, seq_len, weights_
dim)

        # Compute K = W_k*X
        key = self.W_k(inputs)  # (batch_size, seq_len, weights_di
m)
                # Compute V = W_v*X
        value = self.W_v(inputs)  # (batch_size, seq_len, weights_
dim)

        # Split into n_heads submatrices
        query = self.separate_heads(query, batch_size)  # (batch_s
ize, num_heads, seq_len, sub_matrix_dim)
        key = self.separate_heads(key, batch_size)  # (batch_size,
 num_heads, seq_len, sub_matrix_dim)
        value = self.separate_heads(value, batch_size) # (batch_si
ze, num_heads, seq_len, sub_matrix_dim)

        # Compute attention (contains weights and attentions for a
ll heads):
        attention, weights = self.attention(query, key, value)
```

```
        attention = tf.transpose(attention, perm=[0, 2, 1, 3])  #
(batch_size, seq_len, num_heads, sub_matrix_dim)

        # Concatenate all attentions from different heads (squeeze
 the last dimension):
        concat_attention = tf.reshape(attention, (batch_size, -
1, self.weights_dim))  # (batch_size, seq_len, weights_dim)

        # Use a weighted average of the attentions from different
heads:
        output = self.W_h(concat_attention)  # (batch_size, seq_le
n, input_dim)

        return output


    def compute_output_shape(self, input_shape):
        print(input_shape)
        """ Specifies the output shape of the custom layer, withou
t this, the model doesn't work """
        return input_shape
def buildModel1():

    # The input as sequences:
    input_layer = Input(shape=(N_SEQ,))
    # Create the embedding layer
    embedding_layer = Embedding(
        N_vocab,
        N_EMB,
        weights = [embedding_matrix],
        input_length = N_SEQ,
        trainable = False # No need to train as our embeddings are
 already finetuned on Twitter data
    )
    # Create the embeddings
    embedded_sequences = embedding_layer(input_layer)

    # The core of the model:

    # Single layer BiLSTM architecture
    x = Bidirectional(
        LSTM(
            units = DIM_HIDDEN, # In Keras, "units" mean the dimen
sionality of the hidden states h_t output by the LSTM
            dropout = 0.1,
            # recurrent_dropout = 0.1, # Can't use the GPU if this
 is included
            return_sequences = True
```

```
        ),
        merge_mode = "concat" # Just like in Transformers, thus ou
tput h = [h_f; h_b] will have dimension 2*DIM_HIDDEN
    )(embedded_sequences)

    # Adding multiheaded self attention
    x = MultiHeadSelfAttention(N_HEADS, DIM_KEY)(x)

    outputs = Flatten()(x)

    model = Model(input_layer, outputs)
    return model
def buildModel2():
    input_layer = Input(shape = (2*N_SEQ*DIM_HIDDEN,))
    # Fully connected classifer
    x = Dense(1024, activation = "relu")(input_layer)
    x = Dropout(0.1)(x)
    x = Dense(256, activation = "relu")(x)
    x = Dropout(0.1)(x)
    x = Dense(32, activation = "relu")(x)
    outputs = Dense(1, activation = "sigmoid")(x)
    model = Model(input_layer, outputs)
    return model
def combineModels():
    model1 = buildModel1()
    model2 = buildModel2()

    input1 = Input(shape=(N_SEQ,))
    out1 = model1(input1)
    out2 = model2(out1)

    model = Model(input1, out2)
    return model
#model = buildModel()
model = combineModels()
model.compile(
    optimizer = Adam(learning_rate = LR),
    loss = "binary_crossentropy",
    metrics = ["accuracy"])
# Model constants and hyperparameters:
LR = 0.05
N_BATCH = 100
N_EPOCHS = 10
N_HEADS = 100
DIM_HIDDEN = 256 # Dimensionality of the hidden state h_t outputte
d by the LSTM
# Dimensionality of the weight matrices used while computing the a
ttention, note that we divide this into n_head parts later
```

```
DIM_KEY = N_HEADS*256
history = model.fit(
    x_train,
    y_train,
    batch_size = N_BATCH,
    epochs = N_EPOCHS,
    validation_data = (x_val, y_val))
```

5.0.2 Classification M. (Matlab)

```
6     classdef Classification  <
      classreg.learning.classif.ClassificationModel
7         %ConvTOIncremental Support Vector Machine model for
      classification.
8         %   ConvTOIncremental is an SVM model for classification
      with one or
9         %   two classes. This model can predict response for new
      data.
10        %
11        %   ConvTOIncremental properties:
12        %       PredictorNames         - Names of predictors used
      for this model.
13        %       ExpandedPredictorNames - Names of expanded
      predictors.
14        %       ResponseName           - Name of the response
      variable.
15        %       ClassNames             - Names of classes in Y.
16        %       Cost                   - Misclassification costs.
17        %       Prior                  - Prior class
      probabilities.
18        %       ScoreTransform         - Transformation applied
      to predicted classification scores.
19        %       Alpha                  - Coefficients obtained by
      solving the dual problem.
20        %       Beta                   - Coefficients for the
      primal linear problem.
21        %       Bias                   - Bias term.
22        %       KernelParameters       - Kernel parameters.
23        %       Mu                     - Predictor means.
24        %       Sigma                  - Predictor standard
      deviations.
25        %       SupportVectors         - Support vectors.
26        %       SupportVectorLabels    - Support vector labels
      (+1 and -1).
27        %
28        %   ConvTOIncremental methods:
29        %       compareHoldout         - Compare two models using
      test data.
30        %       discardSupportVectors  - Discard support vectors
      for linear SVM.
31        %       edge                   - Classification edge.
32        %       fitPosterior           - Find transformation from
      SVM scores to class posterior probabilities.
33        %       incrementalLearner     - Return an incremental
      binary classification linear model.
34        %       loss                   - Classification loss.
```

```
35      %       margin                  - Classification margins.
36      %       predict                 - Predicted response of
    this model.
37      %       update                  - Update model parameters
    for code generation.
38      %
39      %   See also ClassificationSVM.
40
41      %   Copyright 2013-2020 The MathWorks, Inc.
42
43
    properties(SetAccess=protected,GetAccess=public,Dependent=tru
    e)
44          %ALPHA Coefficients obtained by solving the dual
    problem.
45          %   The Alpha property is a vector with M positive
    elements for M support
46          %   vectors. The PREDICT method computes scores F for
    predictor matrix X
47          %   using
48          %
49          %   F =
    G(X,SupportVectors)*(SupportVectorLabels.*Alpha) + Bias
50          %
51          %   where G(X,SupportVectors) is an N-by-M matrix of
    kernel products for N
52          %   rows in X and M rows in SupportVectors.
53          %
54          %   See also
    classreg.learning.classif.ConvTOIncremental,
55          %   Bias, SupportVectors, SupportVectorLabels,
    predict.
56          Alpha;
57
58          %BETA Coefficients for the primal linear problem.
59          %   The Beta property is a vector of linear
    coefficients with P elements for P
60          %   predictors. If this SVM model is obtained using a
    kernel function other than
61          %   'linear', this property is empty. Otherwise
    scores F obtained by this model
62          %   for predictor matrix X can be computed using
63          %
64          %   F = (X/S)*Beta + Bias
65          %
66          %   where S is the kernel scale saved in the
    KernelParameters.Scale property.
67          %
68          %   When there are categorical predictors, the matrix
    X in this expression
69          %   includes dummy variables for those predictors,
    and the Beta vector is
70          %   expanded accordingly.
71          %
72          %   See also
    classreg.learning.classif.ConvTOIncremental, Alpha, Bias,
73          %   KernelParameters, SupportVectors,
    SupportVectorLabels, predict.
```

```
74          Beta;
75
76          %BIAS Bias term.
77          %   The Bias property is a scalar specifying the bias
    term in the SVM
78          %   model. The PREDICT method computes scores F for
    predictor matrix X
79          %   using
80          %
81          %   F =
    G(X,SupportVectors)*(SupportVectorLabels.*Alpha) + Bias
82          %
83          %   where G(X,SupportVectors) is an N-by-M matrix of
    kernel products for N
84          %   rows in X and M rows in SupportVectors.
85          %
86          %   See also
    classreg.learning.classif.ConvTOIncremental, Alpha,
87          %   Bias, SupportVectors, SupportVectorLabels,
    predict.
88          Bias;
89
90          %KERNELPARAMETERS Parameters of the kernel function.
91          %   The KernelParameters is a struct with two fields:
92          %       Function     - Name of the kernel function, a
    string.
93          %       Scale        - Scale factor used to divide
    predictor values.
94          %   The PREDICT method computes a kernel product
    between vectors x and z
95          %   using Function(x/Scale,z/Scale).
96          %
97          %   See also
    classreg.learning.classif.ConvTOIncremental, predict.
98          KernelParameters;
99
100         %MU Predictor means.
101         %   The Mu property is either empty or a vector with
    P elements, one for
102         %   each predictor. If training data were
    standardized, the Mu property is
103         %   filled with means of predictors used for
    training. Otherwise the Mu
104         %   property is empty. If the Mu property is not
    empty, the PREDICT method
105         %   centers predictor matrix X by subtracting the
    respective element of Mu
106         %   from every column.
107         %
108         %   When there are categorical predictors, Mu
    includes elements for the
109         %   dummy variables for those predictors. The
    corresponding entries in Mu
110         %   are 0, because dummy variables are not centered
    or scaled.
111         %
112         %   See also
    classreg.learning.classif.ConvTOIncremental, predict.
```

```
113         Mu;
114
115         %SIGMA Predictor standard deviations.
116         %   The Sigma property is either empty or a vector
    with P elements, one for
117         %   each predictor. If training data were
    standardized, the Sigma property
118         %   is filled with standard deviations of predictors
    used for training.
119         %   Otherwise the Sigma property is empty. If the
    Sigma property is not
120         %   empty, the PREDICT method scales predictor matrix
    X by dividing every
121         %   column by the respective element of Sigma (after
    centering).
122         %
123         %   When there are categorical predictors, Sigma
    includes elements for the
124         %   dummy variables for those predictors. The
    corresponding entries in Sigma
125         %   are 1, because dummy variables are not centered
    or scaled.
126         %
127         %   See also
    classreg.learning.classif.ConvTOIncremental, predict.
128         Sigma;
129
130         %SUPPORTVECTORS Support vectors.
131         %   The SupportVectors property is an M-by-P matrix
    for M support vectors
132         %   and P predictors. The PREDICT method computes
    scores F for predictor
133         %   matrix X using
134         %
135         %   F =
    G(X,SupportVectors)*(SupportVectorLabels.*Alpha) + Bias
136         %
137         %   where G(X,SupportVectors) is an N-by-M matrix of
    kernel products for N
138         %   rows in X and M rows in SupportVectors.
139         %
140         %   When there are categorical predictors,
    SupportVectors includes dummy
141         %   variables for those predictors.
142         %
143         %   See also
    classreg.learning.classif.ConvTOIncremental,
144         %   Alpha, Bias, SupportVectorLabels, predict.
145         SupportVectors;
146
147         %SUPPORTVECTORLABELS Support vector labels.
148         %   The SupportVectorLabels property is a vector with
    M elements for M
149         %   support vectors saved in SupportVectors. For two-
    class learning with
150         %   non-zero prior probabilities, an element of this
    vector is set to -1 if
```

```
151         %   the respective observation is of class
    ClassNames(1) and +1 if the
152         %   respective observation is of class ClassNames(2)
    in the training data.
153         %   For one-class learning, all elements of this
    vector are set to +1. The
154         %   PREDICT method computes scores F for predictor
    matrix X using
155         %
156         %   F =
    G(X,SupportVectors)*(SupportVectorLabels.*Alpha) + Bias
157         %
158         %   where G(X,SupportVectors) is an N-by-M matrix of
    kernel products for N
159         %   rows in X and M rows in SupportVectors.
160         %
161         %   See also
    classreg.learning.classif.ConvTOIncremental,
162         %   ClassNames, Alpha, Bias, SupportVectors, predict.
163         SupportVectorLabels;
164     end
165
166     methods
167         function a = get.Alpha(this)
168             a = this.Impl.Alpha;
169         end
170
171         function a = get.Bias(this)
172             a = this.Impl.Bias;
173         end
174
175         function b = get.Beta(this)
176             b = this.Impl.Beta;
177         end
178
179         function p = get.KernelParameters(this)
180             p.Function = this.Impl.KernelParameters.Function;
181             p.Scale    = this.Impl.KernelParameters.Scale;
182             % Offset has been incorporated in the Bias
    already.
183             % For the user, offset for prediction is zero.
184
185             if     strcmpi(p.Function,'polynomial')
186                 p.Order =
    this.Impl.KernelParameters.PolyOrder;
187             elseif strcmpi(p.Function,'sigmoid')
188                 p.Sigmoid =
    this.Impl.KernelParameters.Sigmoid;
189             end
190         end
191
192         function a = get.Mu(this)
193             a = this.Impl.Mu;
194         end
195
196         function a = get.Sigma(this)
197             a = this.Impl.Sigma;
198         end
```

```
199
200         function a = get.SupportVectors(this)
201             a = this.Impl.SupportVectors;
202         end
203
204         function a = get.SupportVectorLabels(this)
205             a = this.Impl.SupportVectorLabels;
206         end
207
208         function newobj = update(obj,varargin)
209         %UPDATE Update a ConvTOIncremental model for code
    generation
210         %   by using new parameters
211         %
212         %   UPDATEDMDL = UPDATE(MDL,PARAMS) returns an
    updated version of MDL
213         %   that contains new parameters in PARAMS.
214         %
215         %   Use the output of validatedUpdateInputs (<a
    href="matlab:doc
    classreg.learning.coder.config.svm.classificationsvmcoderconf
    igurer/validatedUpdateInputs">validatedUpdateInputs</a>) as
    the input
216         %   PARAMS, a structure with a field for each
    parameter to
217         %   update in the ConvTOIncremental model.
218         %
219         %   If you train an SVM classification model is
    trained with a linear kernel function
220         %   and discard support vectors using
    discardSupportVectors,
221         %   then PARAMS includes Beta, Bias, Cost, Mu, Prior,
222         %   Scale (kernel scale parameter in
    KernelParameters), and Sigma.
223         %   Otherwise, PARAMS includes Alpha, Bias, Cost, Mu,
    Prior,
224         %   Scale (kernel scale parameter in
    KernelParameters), Sigma, SupportVectorLabels, and
    SupportVectors.
225         %
226         %   If you train an SVM model for one-class
    classification, then
227         %   PARAMS cannot include Prior or Cost.
228         %
229         %   See also learnerCoderConfigurer,
    validatedUpdateInputs,
230         %
      classreg.learning.coder.config.svm.ClassificationSVMCoderCo
    nfigurer.
231
232             % suppress fitPosterior warning
233             warnState  = warning('query','all');
234
    warning('off','stats:classreg:loadCompactModel:SVMFitPosterio
    rReset');
235             cleanupObj = onCleanup(@() warning(warnState));
236
237             % list of parameters that can be updated
```

```
238             updateableParams =
    {'Alpha','SupportVectors','SupportVectorLabels','Beta',...
239                 'Scale','Bias','Prior','Cost','Mu','Sigma'};
240
241             upperbound = 2*numel(updateableParams)+1;
242             narginchk(2,upperbound);
243
244             if
    isa(obj,'classreg.learning.FullClassificationRegressionModel'
    )
245
    warning(message('stats:classreg:coderutils:update:NoncompactM
    odel'));
246             end
247             try
248                 %validate inputs and return a parameter
    struct with new values
249                 paramStr =
    classreg.learning.coderutils.svm.validateSVMParamStruct(obj,t
    rue,updateableParams,varargin{:});
250
251                 updatedModelStr = toStruct(obj);
252
253                 % assign values from updated struct
254                 updatedModelStr.Impl.Alpha = paramStr.Alpha;
255                 updatedModelStr.Impl.SupportVectors =
    paramStr.SupportVectors;
256                 updatedModelStr.Impl.SupportVectorLabels =
    paramStr.SupportVectorLabels;
257                 updatedModelStr.Impl.Beta = paramStr.Beta;
258                 updatedModelStr.Impl.Bias = paramStr.Bias;
259                 updatedModelStr.Impl.KernelParameters.Scale =
    paramStr.Scale;
260
261                 updatedModelStr =
    classreg.learning.coderutils.updatedClassificationStruct(upda
    tedModelStr,paramStr);
262
263                 updatedModelStr.Impl.Mu = paramStr.Mu;
264                 updatedModelStr.Impl.Sigma = paramStr.Sigma;
265
266                 %get new object
267                 newobj =
    classreg.learning.classif.ConvTOIncremental.fromStruct(update
    dModelStr);
268             catch ME
269                 throw(ME);
270             end
271         end
272     end
273
274     methods(Static,Hidden)
275         function obj = fromStruct(s)
276             % Make an SVM object from a codegen struct.
277
278             % check for 2016b compatibility
279             if isfield(s,'fitPosterior')
280                 if s.fitPosterior
```

```
281
    warning(message('stats:classreg:loadCompactModel:SVMFitPosterior
    iorReset'));
282                     s.ScoreTransform =
    'classreg.learning.transform.identity';
283                 else
284                     s.ScoreTransform = s.ScoreTransformFull;
285                 end
286             end
287
288             s =
    classreg.learning.coderutils.structToClassif(s);
289
290             % check for 2018b compatibility
291             if
    isfield(s.Impl.KernelParameters,'IsCustomKernel')
292                 if ~s.Impl.KernelParameters.IsCustomKernel
293                     if isfield(s,'UseEnum') && s.UseEnum
294                         s.Impl.KernelParameters.Function =
    char(s.Impl.KernelParameters.Function);
295                     end
296                 end
297                 s.Impl.KernelParameters =
    rmfield(s.Impl.KernelParameters,'IsCustomKernel');
298             end
299             impl =
    classreg.learning.impl.CompactSVMImpl.fromStruct(s.Impl);
300
301             % Remove codegen-only fields
302             if isfield(s.DataSummary,'NumExpandedPredictors')
303                 s.DataSummary =
    rmfield(s.DataSummary,'NumExpandedPredictors');
304             end
305
306             if isfield(s.DataSummary,'NumCategories')
307                 s.DataSummary =
    rmfield(s.DataSummary,'NumCategories');
308             end
309
310             % Make an object
311             obj =
    classreg.learning.classif.ConvTOIncremental(...
312
    s.DataSummary,s.ClassSummary,s.ScoreTransform,s.ScoreType,imp
    l);
313         end
314     end
315
316     methods(Hidden)
317         function s = toStruct(this,varargin)
318             % Convert to a struct for codegen.
319
320             warnState  = warning('query','all');
321             warning('off','MATLAB:structOnObject');
322             cleanupObj = onCleanup(@() warning(warnState));
323             p = inputParser;
```

```
324
    addParameter(p,'UseEnumerations',true,@(x)isscalar(x)&&islogi
    cal(x));
325             parse(p,varargin{:});
326             useEnum = p.Results.UseEnumerations;
327             % convert common properties to struct
328             fh = functions(this.PrivScoreTransform);
329             fitPosterior = false;
330             % if scoreTransform is anonymous, check if
    attained by
331             % fitPosterior/fitSVMPosterior
332             if strcmpi(fh.type,'anonymous')
333                 % attained by fitPosterior/fitSVMPosterior
334                 if
    contains(fh.file,fullfile('stats','classreg','fitSVMPosterior
    .m'))
335                     fitPosterior = true;
336                     tempFcnStr = erase(fh.function,'@(S)');
337                     fitFunctionName =
    extractBefore(tempFcnStr,'(');
338                     fitFunctionArguments =
    extractBetween(tempFcnStr,',',')');
339                 else
340
    error(message('stats:classreg:learning:coderutils:classifToSt
    ruct:AnonymousFunctionsNotSupported','Score Transform'));
341                 end
342             end
343
344             % test provided scoreTransform
345             try
346
    classreg.learning.internal.convertScoreTransform(this.PrivSco
    reTransform,'handle',numel(this.ClassSummary.ClassNames));
347             catch me
348                 rethrow(me);
349             end
350
351             % convert common properties to struct
352             s =
    classreg.learning.coderutils.classifToStruct(this);
353             s.fitPosterior = fitPosterior;
354
355             if s.fitPosterior
356                 s.ScoreTransformFull = char(fitFunctionName);
357                 s.ScoreTransform = char(fitFunctionName);
358                 arguments = char(fitFunctionArguments);
359                 s.ScoreTransformArguments = arguments;
360                 arguments = str2num(arguments); %#ok<ST2NM>
361                 s.ScoreTransformArgumentsNum = arguments;
362                 s.CustomScoreTransform = false;
363             else
364                 s.ScoreTransformFull = s.ScoreTransform;
365                 scoretransformfull =
    strsplit(s.ScoreTransform,'.');
366                 scoretransform = scoretransformfull{end};
367                 s.ScoreTransform = scoretransform;
368
```

```
369                 % decide whether scoreTransform is a user-
    defined function or
370                 % not
371                 transFcn = ['classreg.learning.transform.'
    s.ScoreTransform];
372                 transFcnCG =
    ['classreg.learning.coder.transform.' s.ScoreTransform];
373                 if isempty(which(transFcn)) ||
    isempty(which(transFcnCG))
374                     s.CustomScoreTransform = true;
375                 else
376                     s.CustomScoreTransform = false;
377                 end
378                 s.ScoreTransformArguments = '';
379                 s.ScoreTransformArgumentsNum = [];
380             end
381
382             if ~s.CustomScoreTransform &&
    isempty(s.ScoreTransformArguments)
383                 temp = s.ScoreTransform;
384                 temp = [upper(temp(1)) temp(2:end)];
385                 s.ScoreTransform =
    classreg.learning.coderutils.Transform.(temp);
386                 s.UseEnumTransform = true;
387             else
388                s.UseEnumTransform = false;
389             end
390
391             % save the path to the fromStruct method
392             s.FromStructFcn =
    'classreg.learning.classif.ConvTOIncremental.fromStruct';
393
394             % impl
395             impl = this.Impl;
396             if isa(impl,'classreg.learning.impl.SVMImpl')
397                 impl = compact(impl,true);
398             end
399             s.Impl = struct(impl);
400             s.UseEnum = useEnum;
401             if
    ~ismember(s.Impl.KernelParameters.Function,{'linear','gaussia
    n','polynomial','rbf'})
402                 s.Impl.KernelParameters.IsCustomKernel =
    true;
403                 s.Impl.KernelParameters.Function =
    s.Impl.KernelParameters.Function;
404             else
405                 s.Impl.KernelParameters.IsCustomKernel =
    false;
406                 if s.UseEnum
407                     s.Impl.KernelParameters.Function =
    classreg.learning.coderutils.svm.KernelFunction.(s.Impl.Kerne
    lParameters.Function);
408                 end
409             end
410
411             s.DataSummary.NumExpandedPredictors =
    numel(getExpandedPredictorNames(this));
```

```
412             s.DataSummary.NumCategories =
    cellfun(@numel,this.VariableRange);
413         end
414     end
415
416     methods
417         function [varargout] = predict(this,X,varargin)
418             %PREDICT Predict response of the SVM model.
419             %   [LABEL,SCORE]=PREDICT(MODEL,X) returns
    predicted class labels and
420             %   scores for SVM model MODEL and predictors X.
    X must be a table if SVM
421             %   was originally trained on a table, or a
    numeric matrix if SVM was
422             %   originally trained on a matrix. If X is a
    table, it must contain all
423             %   the predictors used for training this model.
    If X is a matrix, it must
424             %   have P columns, where P is the number of
    predictors used for training
425             %   this model. Classification labels LABEL have
    the same type as Y used
426             %   for training. Scores SCORE are an N-by-K
    numeric matrix for N
427             %   observations and K classes. The predicted
    label is assigned to the
428             %   class with the largest score.
429             %
430             %   See also
    classreg.learning.classif.ConvTOIncremental.
431
432             [varargout{1:nargout}] = ...
433
    predict@classreg.learning.classif.ClassificationModel(this,X,
    varargin{:});
434         end
435
436         function [varargout] = loss(this,X,varargin)
437             %LOSS Classification error.
438             %   L=LOSS(MODEL,X,Y) returns classification
    error for SVM model MODEL
439             %   computed using predictors X and true class
    labels Y. X must be a table
440             %   if MODEL was originally trained on a table,
    or a numeric matrix if
441             %   MODEL was originally trained on a matrix.  If
    X is a table, it must
442             %   contain all the predictors used for training
    this model. If X is a
443             %   matrix, it must have size N-by-P, where P is
    the number of predictors
444             %   used for training this model. Y must be of
    the same type as
445             %   MODEL.ClassNames and have N elements, where N
    is the number of rows in
446             %   X.
447             %
```

```
448             %
    L=LOSS(MODEL,X,Y,'PARAM1',val1,'PARAM2',val2,...) specifies
    optional
449             %   parameter name/value pairs:
450             %       'LossFun'          - Function handle for
    loss, or string
451             %                            representing a
    built-in loss function.
452             %                            Available loss
    functions for classification:
453             %                            'binodeviance',
    'classiferror', 'hinge',
454             %                            'exponential',
    'logit', and 'quadratic'. If
455             %                            you pass a function
    handle FUN, LOSS calls it
456             %                            as shown below:
457             %
    FUN(C,S,W,COST)
458             %                            where C is an N-by-K
    logical matrix for N rows
459             %                            in X and K classes
    in the ClassNames property,
460             %                            S is an N-by-K
    numeric matrix, W is a numeric
461             %                            vector with N
    elements, and COST is a K-by-K
462             %                            numeric matrix. C
    has one true per row for the
463             %                            true class. S is a
    matrix of predicted scores
464             %                            for classes with one
    row per observation,
465             %                            similar to SCORE
    output from PREDICT. W is a
466             %                            vector of
    observation weights. COST is a
467             %                            matrix of
    misclassification costs. Default:
468             %                            'classiferror'
469             %       'Weights'          -  Vector of
    observation weights. By default the
470             %                            weight of every
    observation is set to 1. The
471             %                            length of this
    vector must be equal to the
472             %                            number of rows in X.
    If X is a table, this
473             %                            may be the name of a
    variable in the table.
474             %   See also
    classreg.learning.classif.ConvTOIncremental,
475             %   ClassNames, predict.
476             [varargin{:}] =
    convertStringsToChars(varargin{:});
477             [varargout{1:nargout}] = ...
```

```
478
    loss@classreg.learning.classif.ClassificationModel(this,X,var
    argin{:});
479         end
480
481         function [obj,trans] = fitPosterior(obj,X,Y)
482             %FITPOSTERIOR Fit posterior probabilities
483             %   OBJ=FITPOSTERIOR(OBJ,X,Y) finds the optimal
    sigmoid transformation for
484             %   SVM model OBJ using predictors X and class
    labels Y. X must be a table
485             %   if OBJ was originally trained on a table, or
    a numeric matrix if OBJ
486             %   was originally trained on a matrix. If X has
    N rows, pass Y as a vector
487             %   with N elements of one of the following
    types: single, double, logical,
488             %   categorical, or cell array of strings. Y can
    be omitted if X is a table
489             %   that includes the response variable.
    FITPOSTERIOR returns an object of
490             %   type ConvTOIncremental and sets the
    ScoreTransform property of
491             %   this object to the optimal transformation.
    The PREDICT method for the
492             %   updated model then returns posterior
    probabilities for 2nd output.
493             %
494             %   [OBJ,TRANS]=FITPOSTERIOR(OBJ,X,Y) also
    returns TRANS, a struct with
495             %   parameters of the optimal transformation from
    score S to posterior
496             %   probability P for the positive class in
    OBJ.ClassNames(2). TRANS has
497             %   the following fields:
498             %       Type                                - String,
    one of: 'sigmoid', 'step'
499             %                                             or
    'constant'. If the two classes
500             %                                             overlap,
    FITPOSTERIOR sets Type to
501             %
    'sigmoid'. If the two classes are
502             %
    perfectly separated, FITPOSTERIOR
503             %                                             sets
    Type to 'step'. If one of the
504             %                                             two
    classes has zero probability,
505             %
    FITPOSTERIOR sets Type to
506             %
    'constant.
507             %          If Type is 'sigmoid', TRANS has
    additional fields:
508             %             Slope                    - Slope A
    of the sigmoid
```

```
509             %
    transformation
510             %
    P(S)=1/(1+exp(A*S+B))
511             %             Intercept                -
    Intercept B of the sigmoid
512             %
    transformation
513             %
    P(S)=1/(1+exp(A*S+B))
514             %          If Type is 'step', TRANS has
    additional fields:
515             %             PositiveClassProbability -
    Probability of the positive class
516             %                                        in the
    interval between LowerBound
517             %                                        and
    UpperBound
518             %             LowerBound               - Lower
    bound of the interval in
519             %                                        which
    the probability for the
520             %                                        positive
    class is set to
521             %
    PositiveClassProbability. Below
522             %                                        this
    bound, the probability for
523             %                                        the
    positive class is zero.
524             %             UpperBound               - Upper
    bound of the interval in
525             %                                        which
    the probability for the
526             %                                        positive
    class is set to
527             %
    PositiveClassProbability. Above
528             %                                        this
    bound, the probability for
529             %                                        the
    positive class is one.
530             %          If Type is 'constant', TRANS has
    additional fields:
531             %             PredictedClass           - Name of
    the predicted class, same
532             %                                        type as
    OBJ.ClassNames. The
533             %
    posterior probability is one for
534             %                                        this
    class.
535             %
536             %   See also
    classreg.learning.classif.ConvTOIncremental,
537             %   ClassNames, ScoreTransform.
538             Y = convertStringsToChars(Y);
```

```
539             % Handle input data such as "tall" requiring a
    special adapter
540             adapter =
    classreg.learning.internal.makeClassificationModelAdapter(obj
    ,X,Y);
541             if ~isempty(adapter)
542
    error(message('MATLAB:bigdata:array:FcnNotSupported','FITPOST
    ERIOR'))
543             end
544
545             [obj,trans] = fitSVMPosterior(obj,X,Y);
546         end
547
548         function this = discardSupportVectors(this)
549             %DISCARDSUPPORTVECTORS Discard support vectors
    for linear SVM.
550             %   OBJ=DISCARDSUPPORTVECTORS(OBJ) empties the
    Alpha, SupportVectors, and
551             %   SupportVectorLabels properties of a linear
    SVM model. After these properties
552             %   are emptied, the PREDICT method can compute
    SVM scores and labels using the
553             %   Beta coefficients. DISCARDSUPPORTVECTORS
    errors for a non-linear SVM model.
554             %
555             %   See also Alpha, Beta, SupportVectors,
    SupportVectorLabels, predict.
556
557             this.Impl = discardSupportVectors(this.Impl);
558         end
559
560
561     end
562
563     methods(Access=protected)
564         function cl = getContinuousLoss(this)
565             cl = [];
566             if
    isequal(this.PrivScoreTransform,@classreg.learning.transform.
    identity)
567                 cl = @classreg.learning.loss.hinge;
568             elseif strcmp(this.ScoreType,'probability')
569                 cl = @classreg.learning.loss.quadratic;
570             end
571         end
572
573         function this = Classification(...
574
    dataSummary,classSummary,scoreTransform,scoreType,impl)
575             this =
    this@classreg.learning.classif.ClassificationModel(...
576
    dataSummary,classSummary,scoreTransform,scoreType);
577             this.Impl = impl;
578             this.DefaultLoss =
    @classreg.learning.loss.classiferror;
```

```
579             this.LabelPredictor =
    @classreg.learning.classif.ClassificationModel.maxScore;
580             this.DefaultScoreType = 'inf';
581             this.CategoricalVariableCoding = 'dummy';
582         end
583
584         function S = score(this,X,varargin)
585             if any(this.CategoricalPredictors)
586                 if ~this.TableInput
587                     X =
    classreg.learning.internal.encodeCategorical(X,this.VariableR
    ange);
588                 end
589                 X =
    classreg.learning.internal.expandCategorical(X,...
590
    this.CategoricalPredictors,this.OrdinalPredictors,...
591                     this.VariableRange);
592             end
593
594             f = score(this.Impl,X,true,varargin{:});
595
596             classnames = this.ClassSummary.ClassNames;
597
598             % Initialize all scores to minus the score for
    the positive
599             % class and fill out the column for the positive
    class later.
600             % When two classes are passed to fitcsvm, but
    only one of them
601             % has non-zero probability, scores obtained by
    one-class
602             % learning can be used for two-class predictions.
603             S = repmat(-f,1,numel(classnames));
604
605             % For one-class learning, the obtained score is
    for the only
606             % class with non-zero probability. For two-class
    learning, the
607             % obtained score is for the 2nd class (+1) with
    non-zero
608             % probability.
609             [~,loc] =
    ismember(this.ClassSummary.NonzeroProbClasses,classnames);
610             if numel(loc)==1 % one-class learning
611                 S(:,loc) = f;
612             else             % binary SVM
613                 S(:,loc(2)) = f;
614             end
615         end
616
617         function s = propsForDisp(this,s)
618             s =
    propsForDisp@classreg.learning.classif.ClassificationModel(th
    is,s);
619             hasAlpha = ~isempty(this.Alpha);
620
621             if ~hasAlpha
```

```
622                 s.Beta                  = this.Beta;
623             end
624
625             if hasAlpha
626                 s.Alpha                 = this.Alpha;
627             end
628
629             s.Bias                      = this.Bias;
630             s.KernelParameters          =
    this.KernelParameters;
631
632             if ~isempty(this.Mu)
633                 s.Mu                    = this.Mu;
634             end
635             if ~isempty(this.Sigma)
636                 s.Sigma                 = this.Sigma;
637             end
638
639             if hasAlpha
640                 s.SupportVectors        = this.SupportVectors;
641             end
642
643             if hasAlpha &&
    numel(this.ClassSummary.NonzeroProbClasses)>1
644                 s.SupportVectorLabels =
    this.SupportVectorLabels;
645             end
646         end
647
648         function n = getExpandedPredictorNames(this)
649             n =
    classreg.learning.internal.expandPredictorNamesWithReference(
    ...
650
    this.PredictorNames,this.CategoricalPredictors,...
651
    this.OrdinalPredictors,this.VariableRange);
652         end
653     end
654
655     %-----------------------------------------------------------
    ---------------
656     methods(Hidden, Static)
657         function name = matlabCodegenRedirect(~)
658             name =
    'classreg.learning.coder.classif.ConvTOIncremental';
659         end
660                 function incrmdl =
    ConvTOIncremental(this,varargin)
661         %INCREMENTALLEARNER Convert classification SVM model
    to incremental learner.
662         %   INCREMENTALMODEL=INCREMENTALLEARNER(MODEL)
    returns an incremental binary
663         %   classification linear SVM model INCREMENTALMODEL
    initialized
664         %   using the parameters of MODEL.
665         %
```

```
666         %   INCREMENTALLEARNER uses these properties of MODEL
    to initialize
667         %   INCREMENTALMODEL:
668         %       'Beta'
669         %       'Bias'
670         %       'ClassNames'              - Only binary
    classification is
671         %                                   allowed.
672         %       'Mu'
673         %       'Prior'
674         %       'Sigma'
675         %
676         %
    INCREMENTALMODEL=INCREMENTALLEARNER(MODEL,'PARAM1',val1,'PARA
    M2',val2,...)
677         %   specifies optional parameter name/value pairs:
678         %       'BatchSize'               - Mini-batch size
    for the stochastic
679         %                                   solvers,
    specified as a positive
680         %                                   scalar.
681         %                                   Default is 10 for
    stochastic solvers.
682         %                                   'BatchSize' is
    not supported
683         %                                   for scale-
    invariant solver.
684         %       'EstimationPeriod'        - Nonnegative
    integer specifying the number of
685         %                                   observations that
    are used for
686         %                                   estimation of
    hyperparameters prior to
687         %                                   training. For a
    positive integer, if
688         %                                   model has unknown
    hyperparameters
689         %                                   that require
    estimation,
690         %                                   hyperparameters
    are estimated.
691         %                                   Default value is
    1000 when estimation is needed.
692         %                                   Otherwise,
    default value is 0.
693         %                                   Refer to
    documentation for more information.
694         %       'Lambda'                  - Ridge (L2)
    regularization strength specified as a
695         %                                   nonnegative
    scalar.
696         %                                   Default: 1e-5 for
    stochastic solvers.
697         %                                   'Lambda' is not
    supported
698         %                                   for scale-
    invariant solver.
```

```
699         %       'LearnRate'              - Learning rate for
    the stochastic
700         %                                  solvers,
    specified either as 'auto' or as a positive
701         %                                  scalar. If
    'Solver' is specified
702         %                                  as 'sgd' or
    'asgd', default is 'auto'.
703         %                                  'LearnRate' is
    not supported
704         %                                  for scale-
    invariant solver.
705         %                                  Refer to
    documentation for info
706         %                                  on 'auto'
    behavior.
707         %       'LearnRateSchedule'      - String, one of:
    'decaying' or 'constant'.
708         %                                  For stochastic
    solvers, if 'decaying',
709         %                                  learning rate
    decreases over
710         %                                  iterations. If
    'constant', learning
711         %                                  rate stays
    constant. Default:
712         %                                  'decaying' for
    stochastic solvers.
713         %
    'LearnRateSchedule' is not
714         %                                  supported for
    scale-invariant solver.
715         %       'Metrics'                - String vector
    representing built-in
716         %                                  metrics, cell
    array of function handles
717         %                                  or a struct of
    function handles specifying
718         %                                  metrics that are
    tracked during incremental
719         %                                  training.
    'Metrics' can be a
720         %                                  combination of
    strings, function
721         %                                  handles or struct
    of function handles
722         %                                  passed as a cell
    array.
723         %                                  Built-in metrics
    for classification:
724         %                                  'binodeviance',
    'classiferror', 'exponential',
725         %                                  'hinge', 'logit',
    and 'quadratic'.
726         %                                  If you pass a
    function handle FUN, it
727         %                                  is called as
    shown below:
```

```
728         %                                   FUN(C,S)
729         %                                   where C is an N-
    by-K logical matrix for N
730         %                                   elements in Y and
    K classes in the ClassNames
731         %                                   property, S is an
    N-by-K numeric matrix.
732         %                                   C has one true
    per row for the true class.
733         %                                   S is a matrix of
    predicted scores for classes
734         %                                   with one row per
    observation, similar to score
735         %                                   output from
    predict.
736         %                                   Metric name is
    extracted from provided function
737         %                                   handle if it's
    not an anonymous
738         %                                   function handle.
    If it is an anonymous
739         %                                   function handle,
    Metric is named
740         %                                   as 'CustomMetric'
    and it is postfixed
741         %                                   with an
    underscore and an integer
742         %                                   identifying its
    relative position in
743         %                                   the Metrics list.
744         %                                   If you pass a
    struct, field names will
745         %                                   be treated as the
    metric names and
746         %                                   values must be
    function handles.
747         %                                   Incremental
    models track two values for
748         %                                   each metric; a
    'Cumulative' value and a
749         %                                   'Window' value.
    'MetricsWindowSize' determines
750         %                                   the update
    frequency of
751         %                                   'Window' metrics.
752         %                                   Default:
    'classiferror'.
753         %       'MetricsWarmupPeriod'     - Nonnegative
    integer specifying the number of
754         %                                   observations that
    will be used for
755         %                                   training the
    model prior to metrics
756         %                                   update. Default
    is 0.
757         %       'MetricsWindowSize'       - Positive integer
    specifying the window
```

```
758         %                                       size that will be
    used for metrics
759         %                                       update. Default
    value is 200.
760         %       'Shuffle'                   - Logical value
    specifying
761         %                                       whether
    observations
762         %                                       are shuffled at
    each step.
763         %                                       Default: true.
764         %                                       'Shuffle' is not
    supported for
765         %                                       stochastic
    solvers.
766         %                                       Observations are
    always
767         %                                       shuffled for
    stochastic
768         %                                       solvers.
769         %       'Standardize'               - Either 'auto' or
    logical scalar.
770         %                                       If true,
    standardize training data
771         %                                       by centering and
    dividing columns by
772         %                                       their standard
    deviations.
773         %                                       Default is
    'auto'.
774         %                                       Refer to
    documentation for info
775         %                                       on 'auto'
    behavior.
776         %       'Solver'                    - String, one of:
    'sgd', 'asgd' or 'scale-invariant'.
777         %                                       If 'sgd',
    stochastic gradient descent
778         %                                       solver is used.
    If 'asgd', average
779         %                                       stochastic
    gradient descent solver is
780         %                                       used. If 'scale-
    invariant', adaptive
781         %                                       scale-invariant
    solver is used.
782         %                                       Default: 'scale-
    invariant'.
783         %
784         %   Example: Initialize an incremental linear model
    using a traditionally trained model
785         %   and continue incremental training.
786         %
787         %       ttModel = fitcsvm(Xoffline,Yoffline);
788         %       mdl = incrementalLearner(ttModel);
789         %       mdl =
    updateMetricsAndFit(mdl,Xnew,Ynew,'Weights',Wnew);
790         %
```

```
791          %   See also incrementalClassificationLinear.
792              assert(size(this.ClassNames,1) ==
     2,message('stats:incremental:Offline:OneClass'));
793              [args,additionalargs] =
     incremental.utils.offlineSVMParameters(this,true,varargin{:})
     ;
794              [args,additionalargs] =
     incremental.utils.offlineClassificationParameters(this,args,a
     dditionalargs);
795             mergedargs =
     incremental.utils.mergeOfflineParameters(args,additionalargs)
     ;
796              incrmdl = IDIncremental(mergedargs);
797          end
798      end
799
800  end
```

5.0.3 IDSVM (Matlab)

```
function [R1,a, b, g, S, E, N,Q] =
IDSVM(Model,x_new,y_new,p1,p2,k,C,sig,newmodel,teta)
%UNTITLED4 Summary of this function goes here
%  Detailed explanation goes here
global teta;
global sig;
R1=[];
%teta=calculate_teta_input_space(p1,p2,k);
disp('done');
if (newmodel==0)
 R1 =
fitcsvm(x_new,y_new,'KernelFunction','Nkernel','KernelScale',sig,'BoxCo
nstraint',C);
elseif (newmodel == 2)
 %R1 = IDIncremental();
 R1 = incrementalClassificationKernel();
 TX=DataTransformation(x_new,teta);
 R1 = updateMetricsAndFit(R1,TX,y_new);
```

```
elseif (newmodel == 3)
 R1 = Classification.ConvTOIncremental(Model);
 TX=DataTransformation(x_new,teta);
 R1 = updateMetrics(R1,TX,y_new);
 R1 = fit(R1,TX,y_new);
elseif (newmodel ==4)
[a, b, g, S, E, N,Q] = IncremenTR1(x_new,y_new,teta,sig, C);
end
```

5.0.4 IKNSVM (Matlab)

```
function [a,b,inds,inde] = IKNSVM(C,X,labels,X1,X2,k,sigma)

alpha=0;
n=size(X,1);
% a=gamma(i);
%     gamma1=2.^gamma(i);
%     sigma=1/(2*gamma1).^(1/2);
%       K = kerneldist(X,2.^gamma(i))+alpha*eye(n);
%       K = svmrbf(X,sigma)+alpha*eye(n);
%        [~,Skernel] = eig(K);
        K = kernel(X,X)+alpha*eye(n);
      K1 = K(1:size(X1,1),:);
      K2 = K(size(X1,1)+1:size(X,1),:);

%find K-NN nearest neighbors and compute means M1 and M2


[idx,~]= knnsearch(K2,K1,'K',k);
d12= zeros((size(K1,1)),1);
 M2 = zeros(size(K1));
 dd = zeros(size(K1));

for id=1:size(K1,1)
    M=0;
    for j=1:k
        M=K2(idx(id,j),:)+M;
%         d12(id)=d(id,j)+d12(id);
```

```
33      end
34      M2(id,:)=M/k;
35  %     d12(id)=d12(id)/k;
36
37
38  dd(id,:)=K1(id,:)-M2(id,:);
39  d12(id,:)=sqrt(dd(id,:)*dd(id,:)');
40  end
41
42  [idx,~]= knnsearch(K1,K2,'K',k);
43  d21=zeros((size(K2,1)),1);
44  M1 = zeros(size(K2));
45   M=0;
46  for id=1:size(K2,1)
47
48      for j=1:k
49          M=K1(idx(id,j),:)+M;
50  %         d21(id)=d(id,j)+d21(id);
51      end
52      M1(id,:)=M/k;
53  %     d21(id)=d21(id)/k;
54  dd(id,:)=K2(id,:)-M1(id,:);
55  d21(id,:)=sqrt(dd(id,:)*dd(id,:)');
56  end
57
58  %compute Psi for Class 1
59  [idx,~]= knnsearch(K1,K1,'K',k);
60  d11=zeros((size(K1,1)),1);
61  M=0;
62  M3 = zeros(size(K1));
63  Psi1=zeros(1,size(X1,1));
64  for id=1:size(K1,1)
65      for j=1:k
66  %         d11(id)=d(id,j)+d11(id);
67            M=K1(idx(id,j),:)+M;
68      end
69       M3(id,:)=M/k;
70  %     d11(id)=d11(id)/k;
71  dd(id,:)=K1(id,:)-M3(id,:);
72  d11(id,:)=sqrt(dd(id,:)*dd(id,:)');
73      Psi1(id)=(min(d11(id),d12(id)))/(d11(id)+d12(id));
74  end
```

```
75
76  %compute Psi for Class 2
77  [idx,~]= knnsearch(K2,K2,'K',k);
78  M=0;
79  d22=zeros((size(K2,1)),1);
80  M4 = zeros(size(K2));
81  Psi2 = zeros(1,size(X2,1));
82  for id=1:size(K2,1)
83      for j=1:k
84          M=K2(idx(id,j),:)+M;
85  %         d22(id)=d(id,j)+d22(id);
86      end
87      M4(id,:)=M/k;
88      dd(id,:)=K2(id,:)-M4(id,:);
89      d22(id,:)=sqrt(dd(id,:)*dd(id,:)');
90  %     d22(id)=d22(id)/k;
91      Psi2(id)=(min(d22(id),d21(id)))/(d22(id)+d21(id));
92  end
93
94  %compute Matrices L1 and L2
95  L1=K1.'-M2.';
96  L2=K2.'-M1.';
97
98  %compute between-class scatter matrices nabla1 and nabla2
99  nabla1 = zeros(size(L1,1));
100 for id=1:size(L1,2)
101     nabla1=Psi1(id).*(L1(:,id)*L1(:,id).')+nabla1;
102 end
103
104 nabla2 = zeros(size(L2,1));
105 for id=1:size(L2,2)
106     nabla2=Psi2(id).*(L2(:,id)*L2(:,id).')+nabla2;
107 end
108
109 %compute between-class scatter matrix nabla
110 nabla=(1/(size(L1,2)+size(L2,2)))*(nabla1+nabla2);
111
112 %compute within-class scatter matrix delta
113 onesN1 = ones(size(K1,1))/size(K1,1);
114 onesN2 = ones(size(K2,1))/size(K2,1);
115 delta = K1.'*(eye(size(K1,1))-onesN1)*K1...
116     +K2.'*(eye(size(K2,1))-onesN2)*K2;
```

```
117 [Vdelta,Sdelta]=eig(delta);
118 %compute téta
119 % beta = 10;
120 beta = 10^-3;
121 psi=[Psi1,Psi2];
122 pp=sum(psi./n);
123  mu1 =1-pp;
124 % mu1=1;
125 nablaupd = nabla+beta.*eye(size(X,1));
126 detnablaupd=det(nablaupd);
127 inverse1= pinv(nablaupd);
128 teta=mu1.*delta*inverse1*delta + eye(size(X,1));
129 %  teta = round(teta,1);
130 % %compute eigenvalues and eigenvectors of téta
131 [Vteta,Steta]=eig(teta);
132 % Vteta=real(Vteta);
133 % Steta=real(Steta);
134 %compute coefficients for weights and kernel update
135 tetapos=Vteta*(Steta^(1/2))*pinv(Vteta);
136 tetaneg=Vteta*(Steta^(-1/2))*pinv(Vteta);
137
138
139 % compute kernel marices for linear kernel
140 K1opt= tetaneg*K1.';
141 K2opt= tetaneg*K2.';
142 Koptim = horzcat(K1opt,K2opt);
143  Koptim=real(Koptim);
144 W=ones(size(X,1),1);
145 %        W=tetapos*ones(size(X,1),1);
146 % else W=Wupd;
147 % end
148 % Mdl1 = fitcsvm(Koptim,labels,'KernelFunction','linear',
    'BoxConstraint',...
149 %     C,'weights',W);
150 [a, b, g, inds, inde, indw,Qc] = IKNtrain(Koptim, labels,
    C,sigma,teta);
151 % Mdl1 =
    fitcsvm(Koptim,labels,'KernelFunction','linear','BoxConstrain
    t',C);
152 % [label,score,cost] = predict(Mdl1,K);
153 % [XX,YY,~,AUC3] = perfcurve(labels,score(:,1),-1);
154 end
```

5.0.5 IKN Train (Matlab)

```
function [atrain, btrain, g, inds, inde, indw,Qc] = IKNtrain(x, y,
C,sigma,teta);
% function [a, b, g, inds, inde, indw] = svcm_train(x, y, C);
%         support vector classification machine
%         incremental learning
%         uses "kernel.m"
%
%         x: independent variable, (L,N) with L: number of points;
N: dimension
%         y: dependent variable, (L,1) containing class labels (-1
or +1)
%         C: soft-margin regularization constant
%
%         a: alpha coefficients (to be multiplied by y)
%         b: offset coefficient
%         g: derivatives (adding one yields margins for each point)
%         inds: indices of support vectors
%         inde: indices of error vectors
%         indw: indices of wrongly classified leave-one-out vectors

%%%%%%%%%% version 1.11; last revised 06/07/2002; send comments to
gert@jhu.edu %%%%%%%%%%

%%% GLOBAL VARIABLES:
global doloo online query terse verbose debug memoryhog visualize
eps sigma
if isempty(doloo)
    doloo = 0;                          % perform loo at end of
training
end
if isempty(online)
    online = 0;                         % take data in the order it
is presented
end
if isempty(query)
    query = 0;                          % choose next point based on
margin distribution
end
if isempty(terse)
    terse = 0;                          % print out only final
results
end
if isempty(verbose)
    verbose = 0;                        % print out details of
intermediate results
end
if isempty(debug)
    debug = 0;                          % use only for debugging;
slows it down significantly
end
if isempty(memoryhog)
    memoryhog = 0;                      % use more memory; good only
if kernel dominates computation
```

```
end
if isempty(visualize)
    visualize = 1;                    % record and plot trajectory
of coeffs. a, g, and leave-one-out g
end
if isempty(eps)
    eps = 1e-6;                       % margin "margin"; makes Q
strictly positive definite
end

%%% END GLOBAL VARIABLES

[L,N] = size(x);
[Ly,Ny] = size(y);
if Ly~=L
    fprintf('svcm_train error: x and y different number of data
points (%g/%g)\n\n', L, Ly);
    return
elseif Ny~=1
    fprintf('svcm_train error: y not a single variable (%g)\n\n',
Ny);
    return
elseif any(y~=-1&y~=1)
    fprintf('svcm_train error: y takes values different from {-
1,+1}\n\n');
    return
end

eps2 = 2*eps/C;
tol = 1e-6;        % tolerance on derivatives at convergence, and
their recursive computation

% fprintf('Support vector soft-margin classifier with incremental
learning\n')
% fprintf('  %g training points\n', L)
% fprintf('  %g dimensions\n\n', N)

keepe = debug|memoryhog;         % store Qe for error vectors as
"kernel cache"
keepr = debug|memoryhog;         % store Qr for recycled support
vectors as "kernel cache"
if verbose
    terse = 0;
    if debug
        fprintf('debugging mode (slower, more memory
intensive)\n\n')
    elseif memoryhog
        fprintf('memoryhog active (fewer kernel evaluations, more
memory intensive)\n\n')
    end
    fprintf('kernel used:\n')
    help kernel
    fprintf('\n')
end

%atrain = zeros(L,1);                      % coefficients, sparse
%btrain = 0;                               % offset
W = 0;                                  % energy function
```

```
g = -(1+eps)*ones(L,1);                 % derivative of energy
function

atrain = zeros(L,1);
atrain(model.IsSupportVector)=model.Alpha;
btrain = model.Bias;
inds = find(model.IsSupportVector);
inde  = find(model.Alpha==1);



%inds = [];                              % indices of support vectors;
none initially
%inde = [];                              % indices of error vectors;
none initially
indo = (L:-1:1)';                       % indices of other vectors; all
initially
indr = [];                              % indices of "recycled" other
vectors; for memory "caching"
indl = [];                              % indices of leave-one-out
vectors (still to be) considered
indw = [];                              % indices of wrongly classified
leave-one-out vectors
ls = length(inds);                      % number of support vectors;
le = length(inde);                      % number of error vectors;
la = ls+le;                             % both
lo = length(indo);                      % number of other vectors;
lr = length(indr);                      % number of recycled vectors
lw = length(indw);                      % number or wrongly classified
leave-one-out vectors
processed = zeros(L,1);                 % keeps track of which points
have been processed
R = Inf;                                % inverse hessian (a(inds) and b
only)
Qs = y';                                % extended hessian; (a(inds)
plus b, and all vectors)
Qe = [];                                % same, for inde ("cache" for
Qs)
Qr = [];                                % same, for indr ("cache" for
Qs)
Qc = [];                                % same, for indc (used for gamma
and Qs)
if visualize                            % for visualization
%     figure(4)
%     hold off
%     clf
%     axis([-0.1*C, 1.1*C, -1.2, 0.2])
    gctraj = [];
%     figure(5)
%     hold off
%     clf
%     figure(6)
%     hold off
%     clf
end

iter = 0;                               % iteration count
memcount = 0;                           % memory usage
```

```
kernelcount = 0;                    % kernel computations, counted
one "row" (epoch) at a time
training = 1;                       % first do training recursion
...
leaveoneout = 0;                    % ... then do leave-one-out
sequence (with retraining)
indc = 0;                           % candidate vector
indco = 0;                          % leave-one-out vector
indso = 0;                          % a recycled support vector;
used as buffer
free = atrain(indo)>0|g(indo)<0;        % free, candidate support
or error vector
left = indo(free);                  % candidates left
continued = any(left);
while continued                      % check for remaining free
points or leave-one-outs to process

    % select candidate indc
    indc_prev = indc;
    if online & indc_prev>0
        if query
            processed(indc_prev) = 1;      % record last point in
the history log
        else
            processed(1:indc_prev) = 1;    % record last and all
preceding points
        end
    end
    if query
%       [gindc, indc] = max(g(left));      % closest to the margin
        [gindc, indc] = min(g(left));      % greedy; worst margin
        indc = left(indc);
    else
        indc = left(length(left));         % take top of the stack,
"last-in, first-out"
    end

    % get Qc, row of hessian corresponding to indc (needed for
gamma)
    if keepr & lr>0 & ...          % check for match among recycled
vectors
        any(find(indr==indc))
            ir = find(indr==indc);                % found, reuse
            Qc = Qr(ir,:);
            indr = indr([1:ir-1,ir+1:lr]);        % ... remove
from indr
            Qr = Qr([1:ir-1,ir+1:lr],:);          % ... and Qr
            lr = lr-1;
    elseif indc==indso             % support vector from previous
iteration, leftover in memory
        Qc = Qso;
    elseif ls>0 & ...              % check for match among support
vectors
        any(find(inds==indc))
            is = find(inds==indc);                % found, reuse
            Qc = Qs(is+1,:);
    elseif keepe & le>0 & ...      % check for match among stored
error vectors
```

```
        any(find(inde==indc))
            ie = find(inde==indc);                  % found, reuse
            Qc = Qe(ie,:);
    elseif indc~=indc_prev          % not (or no longer) available,
compute
        xc = x(indc,:);
        yc = y(indc);
        Qc = (yc*y').*(Ikernel(xc,x,teta));

        Qc(indc) = Qc(indc)+eps2;
        kernelcount = kernelcount+1;
    end

    % prepare to increment/decrement z = a(indc)' or y(indc)*b,
subject to constraints.
    % move z up when adding indc ((re-)training), down when
removing indc (leave-one-out or g>0)
    upc = ~leaveoneout & (g(indc)<=0);
    polc = 2*upc-1;                 % polarity of increment in z
    beta = -R*Qs(:,indc);           % change in [b;a(inds)] per
change in a(indc)
    if ls>0
        % move z = a(indc)'
        gamma = Qc'+Qs'*beta;       % change in g(:) per change in z
= a(indc)'
        z0 = atrain(indc);              % initial z value
        zlim = max(0,C*polc);       % constraint on a(indc)
    else % ls==0
        % move z = y(indc)*b and keep a(indc) constant; there is
no a(:) free to move in inds!
        gamma = y(indc)*Qs';        % change in g(:) per change in z
= y(indc)*b
        z0 = y(indc)*btrain;            % initial z value
        zlim = polc*Inf;            % no constraint on b
    end
    gammac = gamma(indc);
    if gammac<=-tol
        fprintf('\nsvcm_train error: gamma(indc) = %g <= 0 (Q not
positive definite)\n\n', gammac)
    elseif gammac==Inf
        fprintf('\nsvcm_train error: gamma(indc) = %g (Q rank
deficient)\n\n', gammac)
        return
    end

    % intrinsic limit: g(indc) = 0, where indc becomes support
vector
    if ~leaveoneout                 % only consider when training
indc, not when removing indc
        zlimc = z0-g(indc)'./gammac;
    else                            % leave-indc-out!
        zlimc = polc*Inf;
    end

    % support vector constraints: 0<=a(inds)<=C
    zlims = Inf*polc;                           % by default,
immaterial
    if ls>0
```

```
        is = find(inds==indc);
        if any(is)                                  % leave-indc-out,
remove from inds
            zlims = z0;                             % clamp z; no
change to variables
        else
            betaa = beta(2:ls+1);                   % beta terms
corresponding to a(inds)  (not b)
            void = (betaa==0);                      % void zero betaa
values ...
            if any(any(~void))
                warning off % suppress div. by 0
                zlims = z0+(C*(betaa*polc>0)-atrain(inds))./betaa;
                warning on
                zlims(void) = polc*Inf;             % ... which don't
enter the constraints
                [zmins, is] = min(zlims*polc,[],1);
                imin = find(zlims==zmins);
                if length(imin)>1
                    [gmax, imax] = max(abs(betaa(imin)),[],1);
                    is = imin(imax);
                end
                zlims = zmins*polc;                 % pick tightest
constraint
            end
        end
    end

    % error vector constraints: g(inde)<=0
    zlime = Inf*polc;                               % by default,
immaterial
    if le>0
        ie = find(inde==indc);
        if any(ie)                                  % leave-indc-out,
remove from inde
            zlime = z0;                             % clamp z; no
change to variables
        else
            gammae = gamma(inde);
            void = (gammae*polc<0)|(gammae==0);     % void g moving
down, or zero gamma...
            if any(any(~void))
                warning off % suppress div. by 0
                zlime = z0-g(inde)./gammae;
                warning on
                zlime(void) = polc*Inf;             % ... which don't
enter the constraints
                [zmine, ie] = min(zlime*polc,[],1);
                imin = find(zlime==zmine);
                if length(imin)>1
                    [gmax, imax] = max(abs(gammae(imin)),[],1);
                    ie = imin(imax);
                end
                zlime = zmine*polc;                 % pick tightest
constraint
            end
        end
    end
```

```
    % ordinary vector constraints: g(indo)>=0 (only for those that
already are)
    zlimo = Inf*polc;                              % by default,
immaterial
    if lo>0
        gammao = gamma(indo);
        void =
(indo==indc)|(g(indo)<0)|(gammao*polc>0)|(gammao==0);
                                                   % void c, g
negative, g moving up, or zero gamma...
        if online
            void = void|~processed(indo);          % ... or, if
online, points not seen previously,...
        end
        if any(any(~void))
            warning off % suppress div. by 0
            zlimo = z0-g(indo)./gammao;
            warning on
            zlimo(void) = polc*Inf;                % ... which don't
enter the constraints
            [zmino, io] = min(zlimo*polc,[],1);
            imin = find(zlimo==zmino);
            if length(imin)>1
                [gmax, imax] = max(abs(gammao(imin)),[],1);
                io = imin(imax);
            end
            zlimo = zmino*polc;                    % pick tightest
constraint
        end
    end

    % find constraint-satisfying z
    [z,flag] = min([zlim;zlimc;zlims;zlime;zlimo]*polc);
    z = z*polc;
    if (z-z0)*polc<0
        fprintf('\nsvcm_train error: z-z0 of wrong polarity (Q not
positive definite)\n\n')
        return
    end

    if verbose & ~leaveoneout & abs(z-z0)<eps
        fprintf('%g*', flag)                       % procrastinating
iteration!  no progress made
    end

    % update a, b, g and W from z-z0
    if ls>0                                        % z = a(indc)
        atrain(indc) = z;
        btrain = btrain+(z-z0)*beta(1);
        atrain(inds) = atrain(inds)+(z-z0)*beta(2:ls+1);
        W = W+(z-z0)*(g(indc)'+0.5*(z-z0)*gammac);
% energy
    else                                           % z = y(indc)*b
        btrain = y(indc)*z;
    end
    g = g+(z-z0)*gamma;                            % update g
    iter = iter+1;
```

```
    if visualize & ~leaveoneout
        atraj(1:L,iter)=atrain;              % record trajectory
of a(:) over time
        gtraj(1:L,iter)=g;
        ctraj(iter)=indc;
    end

    % bookkeeping: move elements across indc, inds, inde and indo,
and update R and Qs
    converged = (flag<3);                    % done with indc; no other
changes in inds/inde
    incl_inds = 0;
    if flag==1                               % a(indc) reaches the
limits 0 or C, stop moving
        if upc                                       % a(indc)=C, add
to inde
            inde = [inde; indc];
            le = le+1;
            if keepe
                Qe = [Qe; Qc];
            end
            atrain(indc) = C;                          % should be
OK, just to avoid round-off
        else % ~upc                                 % a(indc)=0,
indc stays in (or moves to) indo
            atrain(indc) = 0;                          % should be
OK, just to avoid round-off
        end
    elseif flag==2                           % add indc to support
vectors ...
        incl_inds = 1;
        indb = indc;                                 % ... store in
buffer indb for now
        Qb = Qc;
    elseif flag==3                           % one of support vectors
becomes error or other vector
        indso = inds(is);                            % outgoing inds
        Qso = Qs(is+1,:);                            % could be reused
later
        free_indc = (indc==indso);                   % leave-indc-out:
indc is part of inds
        if beta(is+1)*polc<0 | free_indc             % a(indso)=0 or
indso=indc, move to indo
            if ~free_indc
                if keepr
                    indr = [indr; indso];            % also recycle
into indr for later use
                    Qr = [Qr; Qs(is+1,:)];
                    lr = lr+1;
                end
                if atrain(indso)>C/2
                    fprintf('svcm_train error: a(indso)=%g; 0
anticipated\n', atrain(indso));
                end
                atrain(indso) = 0;                       % should be
OK, just to avoid round-off
            end
            g(indso) = 0;                            % same
```

```
            indo = [indo; indso];
            lo = lo+1;
        else % beta(is+1)*polc>0 & ~free_indc    % a(indso)=C, move
to inde
            if atrain(indso)<C/2
                fprintf('svcm_train error: a(indso)=%g; C
anticipated\n', atrain(indso));
            end
            if keepe
                Qe = [Qe; Qs(is+1,:)];          % save to memory
cache
            end
            atrain(indso) = C;                       % should be
OK, just to avoid round-off
            g(indso) = 0;                       % same
            inde = [inde; indso];
            le = le+1;
        end
        inds = inds([1:is-1,is+1:ls]);          % remove from inds
        stripped = [1:is,is+2:ls+1];            % also from Qs and
R ...
        Qs = Qs(stripped,:);
        ls = ls-1;
        if ls > 0
            if R(is+1,is+1)==0
                fprintf('\nsvcm_train error: divide by zero in R
contraction\n')
                R(is+1,is+1)=1e-8;
            end
            R = R(stripped,stripped)-
R(stripped,is+1)*R(is+1,stripped)/R(is+1,is+1);
        else % no support vectors left
            R = Inf;
        end
    elseif flag==4                          % one of error vectors
becomes support/other vector
        indeo = inde(ie);                           % outgoing inde
        if indc==indeo                              % leave-indc-out
            indo = [indo; indeo];                   % add inde(ie) to
other vectors
            lo = lo+1;
        else
            incl_inds = 1;                          % add inde(ie) to
support vectors ...
            indb = indeo;                           % ... store in
buffer indb for now
            if keepe
                Qb = Qe(ie,:);                      % recover from Qe
cache
            elseif indb==indso
                Qb = Qso;                           % recover from
previous outgoing support vector
            else                                    % not in memory
either way--- recompute

                Qb = (y(indb)*y').*(Ikernel(x(indb,:),x,teta));
                Qb(indb) = Qb(indb)+eps2;
                kernelcount = kernelcount+1;
```

```
            end
        end
        inde = inde([1:ie-1,ie+1:le]);          % remove from
inde
        if keepe
            Qe = Qe([1:ie-1,ie+1:le],:);        % remove from Qe
        end
        le = le-1;
    elseif flag==5                          % one of other vectors
becomes support vector
        indoo = indo(io);                       % outgoing indo
        incl_inds = 1;                          % add indo(io) to
support vectors ...
        indb = indoo;                           % ... store in
buffer indb for now
        if keepr & lr>0 & any(find(indr==indb))  % check for match
among recycled vectors
            ir = find(indr==indb);                      % found,
reuse
            Qb = Qr(ir,:);
            indr = indr([1:ir-1,ir+1:lr]);              % ...
remove from indr
            Qr = Qr([1:ir-1,ir+1:lr],:);                % ... and
Qr
            lr = lr-1;
        elseif indb==indso
            Qb = Qso;                           % recover from
previous outgoing support vector
        else                                    % not in memory
either way--- recompute
            Qb = (y(indb)*y').*(Ikernel(x(indb,:),x,teta));
            Qb(indb) = Qb(indb)+eps2;
            kernelcount = kernelcount+1;
        end
        indo = indo([1:io-1,io+1:lo]);          % remove from
indo
        lo = lo-1;
    end

    if incl_inds                            % move buffer indb into
support vectors inds
        inds = [inds; indb];                    % move to inds
...
        ls = ls+1;
        Qs = [Qs; Qb];                          % and also Qs and
R ...
        if ls==1                                % compute R
directly
            R = [-Qb(indb), y(indb); y(indb), 0];
        else                                    % compute R
recursively
            if flag==2                          % from indc; use
beta and gamma
                    pivot = gamma(indb);
            else % flag==4                      % from inde;
compute beta and pivot
                beta=-R*Qs(1:ls,indb);
                pivot = [beta',1]*Qs(:,indb);
```

```
            end
            if pivot<eps2        % should be eps2 when kernel is
singular (e.g., linear)
                fprintf('\nsvcm_train error: pivot = %g < %g in R
expansion\n\n', pivot, eps2)
                pivot = eps2;
                return ;
            end
            R =
[R,zeros(ls,1);zeros(1,ls+1)]+[beta;1]*[beta',1]/pivot;
        end
    end

    % minor correction in R to avoid numerical instability in
recursion when data is near-singular
    Qss = [[0;y(inds)],Qs(:,inds)];
    R = R+R'-R*Qss*R';

    % indc index adjustments (including leave-one-out)
    if converged & (upc|flag==2)                % indc is now part
of inds or inde
        i = find(indo==indc);
        indo = indo([1:i-1,i+1:lo]);  % remove indc from indo
        lo = lo-1;
    elseif keepr
        indr = [indr; indc];                    % recycle again
into indr for later use
        Qr = [Qr; Qc];
        lr = lr+1;
    end
    if leaveoneout
        indoc = indo(find(indo~=indc));         % indo other than
indc
        free = atrain(indoc)>0|g(indoc)<0;           % candidate
support/error vectors in indoc
        if visualize & ~any(free)
            gctraj = [gctraj,[atrain(indc);g(indc)]];
        end
        converged = converged & ~any(free);
        if converged                            % leave-indc-out
reached a(indc)=0, all others settled
            indl = indl(find(indl~=indc));      % remove indc from
indl
            if visualize
                figure(1)
                hold on
                if any(gctraj)

plot(gctraj(1,:),gctraj(2,:),'ok',gctraj(1,:),gctraj(2,:),'-k')
                end
                gctraj = [];                     % cleanup for
next curve
            end
            if g(indc)<-1                       % if leave-indc-
out generates an error ...
                indw = [indw; indc];            % ... store its
index
```

```
                lw = lw+1;                          % ... and
increment leave-one-out error count
            end
        end
    end

    % debugging mode: check for consistency; a, R, g and W
    if debug
        f = find(isnan(atrain));
        if any(f)
           fprintf('svcm_train error: a(%g) = %g\n', [f,
atrain(f)]')
        end
        f = find(isnan(g));
        if any(f)
           fprintf('svcm_train error: g(%g) = %g\n', [f, g(f)]')
        end
        f = find(atrain>C);
        if any(f)
           fprintf('svcm_train error: a(%g) = %g > C\n', [f,
atrain(f)]')
        end
        f = find(atrain<0);
        if any(f)
           fprintf('svcm_train error: a(%g) = %g < 0\n', [f,
atrain(f)]')
        end
        f = inde(find(atrain(inde)>C|atrain(inde)<C));
        if any(f)
                fprintf('svcm_train error: a(%g) = %g ~= C\n', [f,
atrain(f)]')
        end
        if abs(y'*atrain)>tol
                fprintf('svcm_train error: y''*a = %g ~= 0
(tol=%g)\n', y'*atrain, tol);
        end
        Rdiv = max(max(abs(Qss*R-diag(ones(ls+1,1)))));
        if Rdiv>tol
            if flag==2|flag==4    % support vector added
                fprintf('svcm_train error: divergence %g in R
expansion (tol=%g; pivot=%g)\n',...
                        Rdiv, tol, pivot);
            elseif flag==3        % support vector removed
                fprintf('svcm_train error: divergence %g in R
contraction (tol=%g)\n', Rdiv, tol);
            else                  % no support vector added or
removed; strange...
                fprintf('svcm_train error: divergence %g in R ?
(tol=%g)\n', Rdiv, tol);
            end
        end
        if keepe&keepr            % only check g when Qe and Qr
are readily available
            greal = Qs'*[btrain;atrain(inds)]-(1+eps);
            if le>0
                greal = greal+Qe'*atrain(inde);
            end
            if lr>0
```

```
                greal = greal+Qr'*atrain(indr);
            end
            if max(abs(greal-g))>tol
                fprintf('svcm_train error: tolerance (tol=%g)
exceeded in computation of g\n', tol);
            end
        end
        f = inds(find(abs(g(inds))>tol));
        if any(f)
            fprintf('svcm_train error: g(%g) = %g ~= 0
(tol=%g)\n', [f, g(f), f*0+tol]');
        end
        f = inde(find(g(inde)>tol));
        if any(f)
            fprintf('svcm_train error: g(%g) = %g > 0 (tol=%g)\n',
[f, g(f), f*0+tol]');
        end
        Wreal = 0.5*sum((g-btrain*y-1-eps).*atrain);        %
energy
        if abs(Wreal-W)>tol*abs(W)
            fprintf('svcm_train error: energy W = %g ~= %g
(tol=%g)\n', W, Wreal, tol);
        end
        inda = sort([indo;inds;inde]);
        if any(inda~=(1:L)')
            fprintf('svcm_train error: union [indo;inds;inde] does
not equate entire set\n');
        end
        if ls~=length(inds)
            fprintf('svcm_train error: miscount in number of
support vectors\n');
        end
        if le~=length(inde)
            fprintf('svcm_train error: miscount in number of error
vectors\n');
        end
        if lo~=length(indo)
            fprintf('svcm_train error: miscount in number of other
vectors\n');
        end
    end

    memcount = max(memcount,ls+keepe*le+keepr*lr+~(keepe&keepr));
% kernel storage

    if verbose
        fprintf('    c: %g (#%g)', y(indc)>0, indc)
        if leaveoneout
            fprintf(', margin: %6.3g, gamma: %6.3g', g(indc)+1,
gammac)
        end
        fprintf(' | s: ')
        fprintf('%g', y(inds)>0)
        if ls<6
            fprintf(' (')
            fprintf('#%g', inds)
            fprintf(')')
        end
```

```
        fprintf(' | e: 0:%g, 1:%g\n', sum(y(inde)<0),
sum(y(inde)>0))
    end

    if converged                          % indc finished; report
        if ~training & ~leaveoneout       % retraining or tracing
back; uninteresting
            % nothing
        elseif ~terse & ~leaveoneout
%             fprintf(' [%3g%%] iter. %3g: %3g support vectors;
%3g error vectors;  energy %g\n', ...
%                 100-round(100*length(left)/L), iter, ls, le, W)
        elseif ~terse % & leaveoneout
%             fprintf(' [%3g%%] iter. %3g: %3g leave-one-out
errors', 100-round(100*length(indl)/L), iter, lw)
%             fprintf(' (# %g: margin %g)\n', indc, g(indc)+1)
        elseif ~leaveoneout % & terse
            if ls+le>la
%                 fprintf('+')
            elseif ls+le<la
%                 fprintf('-')
            else
%                 fprintf('=')
            end
        else % leaveoneout & terse
            if g(indc)<-1
%                 fprintf('x')
            else
%                 fprintf('o')
            end
        end
    la = ls+le;
    end

    % prepare for next iteration, if any
    indoc = indo(find(indo~=indco));       % indo other than indco
(leave-one-out index, if active)
    free = atrain(indoc)>0|g(indoc)<0;          % candidate
support/error vectors in indoc
    if any(free)
        left = indoc(free);                % candidates left, keep
(re-)training
        leaveoneout = 0;                   % interrupt leave-one-
out, if active
    else % ~any(free)                      % done; finish up and
(re-)initiate leave-one-out
        if training                        % first time around
(not re-training)
            % print out results of svm training
            if terse
%                 fprintf('\n\n  %g support vectors; %g error
vectors; energy %g\n\n', ...
%                                     ls, le, W)
            else
%                 fprintf('\n%4g epoch kernel evaluations (%3g%%
of run-time)\n',...
% %                           kernelcount,
round(kernelcount/(ls+le)*100))
```

```
%                 fprintf(  '%4g epoch vectors in memory  (%3g%%
of data)\n\n',...
%                         memcount, round(memcount/(N+1)*100))
            end
            if visualize

                % plot a trajectory
%                 figure(5)
%                 h=image(atraj/C*length(gray));
%                 h2=get(h,'Parent');
%                 set(h2,'YDir','normal')     % 'image' normally
reverts the y axis
%                 colormap(1-gray)
%                 xlabel('Iteration')
%                 ylabel('Coefficients \alpha_{\it{i}}')
%                 print ('C:\Users\Derbe\Desktop\atraj.eps', '-
deps' )

                % plot g trajectory
%                 figure(6)
%                 gmax = max(max(abs(gtraj)));
%                 h=image((gtraj/gmax+1)/2*length(gray));
%                 h2=get(h,'Parent');
%                 set(h2,'YDir','normal')     % 'image' normally
reverts the y axis
%                 grey = gray;
%                 redblue = min(grey,1-grey);
%                 redblue(:,1) = redblue(:,1)+1-grey(:,1);
%                 redblue(:,2) = 2*redblue(:,2);
%                 redblue(:,3) = redblue(:,3)+grey(:,3);
%                 colormap(redblue)
%                 xlabel('Iteration')
%                 ylabel('Coefficients {\it{g}_{\it{i}}}')
%                 print ('C:\Users\Derbe\Desktop\gtraj.eps', '-
deps' )
%
%
%                 save C:\Users\Derbe\Desktop\traj
%                 save C:\Users\Derbe\Desktop\atraj
%                 save C:\Users\Derbe\Desktop\gtraj
%                 save C:\Users\Derbe\Desktop\ctraj
            end
            if debug                            % store final result
to compare with retraining later
                afinal = atrain;
                gfinal = g;
                Wfinal = W;
            end

            % initiate leave-one-out, if desired
            if doloo
              leaveoneout = 1;
              indl = [inds;inde];               % support and error
vectors (others already correct)
              indw = indl(g(indl)<-1);          % leave-one-out errors
so far (g<-1) ...
              lw = length(indw);                % ... their number
```

```
            indl = indl(g(indl)>=-1);      % remove errors so far
from leave-one-out stack
            indco = indl(length(indl));    % pick first leave-
one-out index; top of stack ...
            left = indco;                  % ... and let
indc=indco (untrain; upc=0)
          else
            continued = 0;
          end

          training = 0;                    % don't ever visit
again!
        elseif ~leaveoneout                % retrained or traced
back
          if debug&~any(find(indo==indco))
              % traced back; compare with previously trained
results
              if max(abs(atrain-afinal))>tol
                    fprintf('svcm_train error: final coeffs. a
exceed tolerance (tol=%g)\n', tol);
              end
              if max(abs(g-gfinal))>tol
                    fprintf('svcm_train error: final
derivatives g exceed tolerance (tol=%g)\n', tol);
              end
              if abs(W-Wfinal)>tol*abs(W)
                    fprintf('svcm_train error: final energy W
= %g ~= %g (tol=%g)\n', Wfinal, W, tol);
              end
          end
          if any(indl)
              indco = indl(length(indl)); % leave-one-out index;
top of stack
              left = indco;
              leaveoneout = 1;
          else % ~any(indl) % finished all leave-one-outs;
summarize, and done!
              continued = 0;
              ltw = sum(g<-1);             % number of training
errors
              if terse
%                   fprintf('\n\n  %g leave-one-out errors;',
lw)
%                   fprintf(' %g training errors\n\n', ltw)
              else % ~terse
%                   fprintf('\n%4g training points\n', L)
%                   fprintf(  '%4g support/error vectors
(%g/%g)\n', la, ls, le)
%                   fprintf(  '%4g leave-one-out errors
(%3.1f%%)\n', lw, lw/L*100)
%                   fprintf(  '%4g training errors
(%3.1f%%)\n', ltw, ltw/L*100)
%                   fprintf('\n%4g epoch kernel evaluations
(%3g%% of run-time)\n',...
%                       kernelcount,
round(kernelcount/(ls+le)*100))
%                   fprintf(  '%4g epoch vectors in memory
(%3g%% of data)\n\n',...
```

```
%                           memcount, round(memcount/(N+1)*100))
                end
                if visualize                    % plot g trajectory
%                     figure(1)
%                     hold off
%                     xlabel('\alpha_{\it{c}}')
%                     ylabel('\it{g_c}')
%                     axis([-0.1*C, 1.1*C, -1.2, 0.2])
%                     h=line([0,0,C,C],[-1,0,0,-1]);
%                     set(h,'Color',[0 0 0])
%                     set(h,'LineStyle',':')
%                     set(h,'LineWidth',[0.2])
%                     h=line([0,C],[-1,-1]);
%                     set(h,'Color',[0 0 0])
%                     set(h,'LineStyle','--')
%                     set(h,'LineWidth',[1.0])
%                      print ('C:\Users\Derbe\Desktop\gctraj.eps',
'-deps' )
                end
            end
            else % leaveoneout                  % leave-one-out
procedure
            if flag==1                          % finished leave-one-
out, now trace back
                % note: already decremented indl and updated
indw/lw above
                leaveoneout = 0;
                indco = 0;
            else                                % not done yet with
leave-one-out ...
                left = indco;                   % ... continue with
indc=indco
            end
        end
    end
end
```

5.0.6 Incremental Train (Matlab)

```
function [atrain, btrain, g, inds, inde, indw,Qc] = IncrementTR1(x,
y,teta,sigma, C);
global doloo online query terse verbose debug memoryhog visualize
eps sigma
if isempty(doloo)
    doloo = 0;                          % perform loo at end of
training
end
if isempty(online)
    online = 0;                         % take data in the order it
is presented
end
if isempty(query)
    query = 0;                          % choose next point based on
margin distribution
end
if isempty(terse)
    terse = 0;                          % print out only final
results
```

```
end
if isempty(verbose)
    verbose = 0;                        % print out details of
intermediate results
end
if isempty(debug)
    debug = 0;                          % use only for debugging;
slows it down significantly
end
if isempty(memoryhog)
    memoryhog = 0;                      % use more memory; good only
if kernel dominates computation
end
if isempty(visualize)
    visualize = 1;                      % record and plot trajectory
of coeffs. a, g, and leave-one-out g
end
if isempty(eps)
    eps = 1e-6;                         % margin "margin"; makes Q
strictly positive definite
end
[L,N] = size(x);
[Ly,Ny] = size(y);
if Ly~=L
    fprintf('x and y different number of data points (%g/%g)\n\n',
L, Ly);
    return
elseif Ny~=1
    fprintf(' y not a single variable (%g)\n\n', Ny);
    return
elseif any(y~=-1&y~=1)
    fprintf('y takes values different from {-1,+1}\n\n');
    return
end
keepe = debug|memoryhog;
keepr = debug|memoryhog;
eps2 = 2*eps/C;
tol = 1e-6;
atrain = zeros(L,1);                        % coefficients, sparse
btrain = 0;
W = 0;
g = -(1+eps)*ones(L,1);
inds = [];
inde = [];
indo = (L:-1:1)';
indr = [];
indl = [];
indw = [];
ls = length(inds);
le = length(inde);
la = ls+le;
lo = length(indo);
lr = length(indr);
lw = length(indw);
processed = zeros(L,1);
R = Inf;
Qs = y';
Qe = [];
```

```
Qr = [];
Qc = [];

iter = 0;
memcount = 0;
kernelcount = 0;
training = 1;
leaveoneout = 0;
indc = 0;
indco = 0;
indso = 0;
free = atrain(indo)>0|g(indo)<0;
left = indo(free);
continued = any(left);
while continued

    indc_prev = indc;
    if online & indc_prev>0
        if query
            processed(indc_prev) = 1;
        else
            processed(1:indc_prev) = 1;
        end
    end
    if query
%       [gindc, indc] = max(g(left));
        [gindc, indc] = min(g(left));
        indc = left(indc);
    else
        indc = left(length(left));
    end

    if keepr & lr>0 & ...
        any(find(indr==indc))
            ir = find(indr==indc);
            Qc = Qr(ir,:);
            indr = indr([1:ir-1,ir+1:lr]);
            Qr = Qr([1:ir-1,ir+1:lr],:);
            lr = lr-1;
    elseif indc==indso
        Qc = Qso;
    elseif ls>0 & ...
        any(find(inds==indc))
            is = find(inds==indc);
            Qc = Qs(is+1,:);
    elseif keepe & le>0 & ...
        any(find(inde==indc))
            ie = find(inde==indc);
            Qc = Qe(ie,:);
    elseif indc~=indc_prev
        xc = x(indc,:);
        yc = y(indc);
        Qc = (yc*y').*(Mykernel(xc,x,teta));

        Qc(indc) = Qc(indc)+eps2;
        kernelcount = kernelcount+1;
    end
```

```
upc = ~leaveoneout & (g(indc)<=0);
polc = 2*upc-1;
beta = -R*Qs(:,indc);
if ls>0
    % move z = a(indc)'
    gamma = Qc'+Qs'*beta;
    z0 = atrain(indc);
    zlim = max(0,C*polc);
else
    gamma = y(indc)*Qs';
    z0 = y(indc)*btrain;
    zlim = polc*Inf;
end
gammac = gamma(indc);


if ~leaveoneout
    zlimc = z0-g(indc)'./gammac;
else
    zlimc = polc*Inf;
end


zlims = Inf*polc;
if ls>0
    is = find(inds==indc);
    if any(is)
        zlims = z0;
    else
        betaa = beta(2:ls+1);
        void = (betaa==0);
        if any(any(~void))
            warning off
            zlims = z0+(C*(betaa*polc>0)-atrain(inds))./betaa;
            warning on
            zlims(void) = polc*Inf;
            [zmins, is] = min(zlims*polc,[],1);
            imin = find(zlims==zmins);
            if length(imin)>1
                [gmax, imax] = max(abs(betaa(imin)),[],1);
                is = imin(imax);
            end
            zlims = zmins*polc;
        end
    end
end


zlime = Inf*polc;
if le>0
    ie = find(inde==indc);
    if any(ie)
        zlime = z0;
        gammae = gamma(inde);
        void = (gammae*polc<0)|(gammae==0);
        if any(any(~void))
            warning off
```

```
                zlime = z0-g(inde)./gammae;
                warning on
                zlime(void) = polc*Inf;
                [zmine, ie] = min(zlime*polc,[],1);
                imin = find(zlime==zmine);
                if length(imin)>1
                    [gmax, imax] = max(abs(gammae(imin)),[],1);
                    ie = imin(imax);
                end
                zlime = zmine*polc;
            end
        end
    end


    zlimo = Inf*polc;
    if lo>0
        gammao = gamma(indo);
        void =
(indo==indc)|(g(indo)<0)|(gammao*polc>0)|(gammao==0);

        if online
            void = void|~processed(indo);
        end
        if any(any(~void))
            warning off
            zlimo = z0-g(indo)./gammao;
            warning on
            zlimo(void) = polc*Inf;
            [zmino, io] = min(zlimo*polc,[],1);
            imin = find(zlimo==zmino);
            if length(imin)>1
                [gmax, imax] = max(abs(gammao(imin)),[],1);
                io = imin(imax);
            end
            zlimo = zmino*polc;
        end
    end

    [z,flag] = min([zlim;zlimc;zlims;zlime;zlimo]*polc);
    z = z*polc;
    if (z-z0)*polc<0
        fprintf('\n Error: z-z0 of wrong polarity (Q not positive
definite)\n\n')
        return
    end

    if verbose & ~leaveoneout & abs(z-z0)<eps
        fprintf('%g*', flag)
    end
    if ls>0
        atrain(indc) = z;
        btrain = btrain+(z-z0)*beta(1);
        atrain(inds) = atrain(inds)+(z-z0)*beta(2:ls+1);
        W = W+(z-z0)*(g(indc)'+0.5*(z-z0)*gammac);
    else                                    % z = y(indc)*b
        btrain = y(indc)*z;
    end
```

```
    g = g+(z-z0)*gamma;
    iter = iter+1;


    converged = (flag<3);
    incl_inds = 0;
    if flag==1
        if upc
            inde = [inde; indc];
            le = le+1;
            if keepe
                Qe = [Qe; Qc];
            end
            atrain(indc) = C;
        else % ~upc
            atrain(indc) = 0;
        end
    elseif flag==2
        incl_inds = 1;
        indb = indc;
        Qb = Qc;
    elseif flag==3
        indso = inds(is);
        Qso = Qs(is+1,:);
        free_indc = (indc==indso);
        if beta(is+1)*polc<0 | free_indc
            if ~free_indc
                if keepr
                    indr = [indr; indso];
                    Qr = [Qr; Qs(is+1,:)];
                    lr = lr+1;
                end
                if atrain(indso)>C/2
                    fprintf('Error: a(indso)=%g; 0 anticipated\n',
atrain(indso));
                end
                atrain(indso) = 0;
            end
            g(indso) = 0;
            indo = [indo; indso];
            lo = lo+1;
        else
            if atrain(indso)<C/2
                fprintf('Error: a(indso)=%g; C anticipated\n',
atrain(indso));
            end
            if keepe
                Qe = [Qe; Qs(is+1,:)];
            end
            atrain(indso) = C;
            g(indso) = 0;
            inde = [inde; indso];
            le = le+1;
        end
        inds = inds([1:is-1,is+1:ls]);
        stripped = [1:is,is+2:ls+1];
        Qs = Qs(stripped,:);
        ls = ls-1;
```

```
        if ls > 0
            if R(is+1,is+1)==0
                R(is+1,is+1)=1e-8;
            end
            R = R(stripped,stripped)-
R(stripped,is+1)*R(is+1,stripped)/R(is+1,is+1);
        else
            R = Inf;
        end
    elseif flag==4
        indeo = inde(ie);
        if indc==indeo
            indo = [indo; indeo];
            lo = lo+1;
        else
            incl_inds = 1;
            indb = indeo;
            if keepe
                Qb = Qe(ie,:);
            elseif indb==indso
                Qb = Qso;
            else

                Qb = (y(indb)*y').*(Mykernel(x(indb,:),x,teta));
                Qb(indb) = Qb(indb)+eps2;
                kernelcount = kernelcount+1;
            end
        end
        inde = inde([1:ie-1,ie+1:le]);
        if keepe
            Qe = Qe([1:ie-1,ie+1:le],:);
        end
        le = le-1;
    elseif flag==5
        indoo = indo(io);
        incl_inds = 1;
        indb = indoo;
        if keepr & lr>0 & any(find(indr==indb))
            ir = find(indr==indb);
            Qb = Qr(ir,:);
            indr = indr([1:ir-1,ir+1:lr]);
            Qr = Qr([1:ir-1,ir+1:lr],:);
            lr = lr-1;
        elseif indb==indso
            Qb = Qso;
        else
            Qb = (y(indb)*y').*(Mykernel(x(indb,:),x,teta));
            Qb(indb) = Qb(indb)+eps2;
            kernelcount = kernelcount+1;
        end
        indo = indo([1:io-1,io+1:lo]);
        lo = lo-1;
    end

    if incl_inds
        inds = [inds; indb];
        ls = ls+1;
        Qs = [Qs; Qb];
```

```
        if ls==1
            R = [-Qb(indb), y(indb); y(indb), 0];
        else
            if flag==2
                    pivot = gamma(indb);
            else % flag==4
                beta=-R*Qs(1:ls,indb);
                pivot = [beta',1]*Qs(:,indb);
            end
            if pivot<eps2
                pivot = eps2;
                return ;
            end
            R =
[R,zeros(ls,1);zeros(1,ls+1)]+[beta;1]*[beta',1]/pivot;
        end
    end


    Qss = [[0;y(inds)],Qs(:,inds)];
    R = R+R'-R*Qss*R';


    if converged & (upc|flag==2)
        i = find(indo==indc);
        indo = indo([1:i-1,i+1:lo]);
        lo = lo-1;
    elseif keepr
        indr = [indr; indc];
        Qr = [Qr; Qc];
        lr = lr+1;
    end
    if leaveoneout
        indoc = indo(find(indo~=indc));
        free = atrain(indoc)>0|g(indoc)<0;
        if visualize & ~any(free)
            gctraj = [gctraj,[atrain(indc);g(indc)]];
        end
        converged = converged & ~any(free);
        if converged
            indl = indl(find(indl~=indc));
            if g(indc)<-1
                indw = [indw; indc];
                lw = lw+1;
            end
        end
    end

    if debug
        f = find(isnan(atrain));
        if any(f)
           fprintf('Error: a(%g) = %g\n', [f, atrain(f)]')
        end
        f = find(isnan(g));
        if any(f)
           fprintf('Error: g(%g) = %g\n', [f, g(f)]')
        end
        f = find(atrain>C);
```

```
        if any(f)
           fprintf('Error: a(%g) = %g > C\n', [f, atrain(f)]')
        end
        f = find(atrain<0);
        if any(f)
           fprintf('Error: a(%g) = %g < 0\n', [f, atrain(f)]')
        end
        f = inde(find(atrain(inde)>C|atrain(inde)<C));
        if any(f)
                fprintf('Error: a(%g) = %g ~= C\n', [f,
atrain(f)]')
        end
        if abs(y'*atrain)>tol
                fprintf('Error: y''*a = %g ~= 0 (tol=%g)\n',
y'*atrain, tol);
        end
        Rdiv = max(max(abs(Qss*R-diag(ones(ls+1,1)))));
        if keepe&keepr
            greal = Qs'*[btrain;atrain(inds)]-(1+eps);
            if le>0
                greal = greal+Qe'*atrain(inde);
            end
            if lr>0
                greal = greal+Qr'*atrain(indr);
            end
            if max(abs(greal-g))>tol
                fprintf('Error: tolerance (tol=%g) exceeded in
computation of g\n', tol);
            end
        end
        f = inds(find(abs(g(inds))>tol));
        if any(f)
            fprintf('Error: g(%g) = %g ~= 0 (tol=%g)\n', [f, g(f),
f*0+tol]');
        end
        f = inde(find(g(inde)>tol));
        if any(f)
            fprintf('Error: g(%g) = %g > 0 (tol=%g)\n', [f, g(f),
f*0+tol]');
        end
        Wreal = 0.5*sum((g-btrain*y-1-eps).*atrain);        %
energy
        if abs(Wreal-W)>tol*abs(W)
            fprintf('Error: energy W = %g ~= %g (tol=%g)\n', W,
Wreal, tol);
        end
        inda = sort([indo;inds;inde]);
        if any(inda~=(1:L)')
            fprintf('svcm_train error: union [indo;inds;inde] does
not equate entire set\n');
        end
        if ls~=length(inds)
            fprintf('svcm_train error: miscount in number of
support vectors\n');
        end
        if le~=length(inde)
            fprintf('Error: miscount in number of error
vectors\n');
```

```
        end
        if lo~=length(indo)
            fprintf('Error: miscount in number of other
vectors\n');
        end
    end

    memcount = max(memcount,ls+keepe*le+keepr*lr+~(keepe&keepr));

    if verbose
        fprintf('    c: %g (#%g)', y(indc)>0, indc)
        if leaveoneout
            fprintf(', margin: %6.3g, gamma: %6.3g', g(indc)+1,
gammac)
        end
        fprintf(' | s: ')
        fprintf('%g', y(inds)>0)
        if ls<6
            fprintf(' (')
            fprintf('#%g', inds)
            fprintf(')')
        end
        fprintf(' | e: 0:%g, 1:%g\n', sum(y(inde)<0),
sum(y(inde)>0))
    end

    if converged
    la = ls+le;
    end
    indoc = indo(find(indo~=indco));
    free = atrain(indoc)>0|g(indoc)<0;
    if any(free)
        left = indoc(free);
        leaveoneout = 0;
    else
        if training
            if debug
                afinal = atrain;
                gfinal = g;
                Wfinal = W;
            end
            if doloo
              leaveoneout = 1;
              indl = [inds;inde];
              indw = indl(g(indl)<-1);
              lw = length(indw);
              indl = indl(g(indl)>=-1);
              indco = indl(length(indl));
              left = indco;
            else
              continued = 0;
            end

            training = 0;
        elseif ~leaveoneout
            if any(indl)
                indco = indl(length(indl));
                left = indco;
```

```
                leaveoneout = 1;
            else
                continued = 0;
                ltw = sum(g<-1);

            end
            else
            if flag==1
                leaveoneout = 0;
                indco = 0;
            else
                left = indco;
            end
        end
    end
end
```

5.0.7 KNSVM (Matlab)

```
function [Mdl1] = KNSVM(X,X1,X2,labels,k,C,eta,sigma)
% global eta
alpha=0;
n=size(X,1);
% a=gamma(i);
%     gamma1=2.^gamma(i);
%     sigma=1/(2*gamma1).^(1/2);
%       K = kerneldist(X,2.^gamma(i))+alpha*eye(n);
%       K = svmrbf(X,sigma)+alpha*eye(n);
%        [~,Skernel] = eig(K);
        K = kernel(X,X)+alpha*eye(n);
      K1 = K(1:size(X1,1),:);
      K2 = K(size(X1,1)+1:size(X,1),:);

%find K-NN nearest neighbors and compute means M1 and M2


[idx,~]= knnsearch(K2,K1,'K',k);
d12= zeros((size(K1,1)),1);
 M2 = zeros(size(K1));
 dd = zeros(size(K1));

for id=1:size(K1,1)
    M=0;
    for j=1:k
        M=K2(idx(id,j),:)+M;
%         d12(id)=d(id,j)+d12(id);
    end
    M2(id,:)=M/k;
%     d12(id)=d12(id)/k;


dd(id,:)=K1(id,:)-M2(id,:);
d12(id,:)=sqrt(dd(id,:)*dd(id,:)');
end

[idx,~]= knnsearch(K1,K2,'K',k);
d21=zeros((size(K2,1)),1);
M1 = zeros(size(K2));
```

```
 M=0;
for id=1:size(K2,1)

    for j=1:k
        M=K1(idx(id,j),:)+M;
%         d21(id)=d(id,j)+d21(id);
    end
    M1(id,:)=M/k;
%     d21(id)=d21(id)/k;
dd(id,:)=K2(id,:)-M1(id,:);
d21(id,:)=sqrt(dd(id,:)*dd(id,:)');
end

%compute Psi for Class 1
[idx,~]= knnsearch(K1,K1,'K',k);
d11=zeros((size(K1,1)),1);
M=0;
M3 = zeros(size(K1));
Psi1=zeros(1,size(X1,1));
for id=1:size(K1,1)
    for j=1:k
%         d11(id)=d(id,j)+d11(id);
          M=K1(idx(id,j),:)+M;
    end
     M3(id,:)=M/k;
%     d11(id)=d11(id)/k;
dd(id,:)=K1(id,:)-M3(id,:);
d11(id,:)=sqrt(dd(id,:)*dd(id,:)');
    Psi1(id)=(min(d11(id),d12(id)))/(d11(id)+d12(id));
end

%compute Psi for Class 2
[idx,~]= knnsearch(K2,K2,'K',k);
M=0;
d22=zeros((size(K2,1)),1);
M4 = zeros(size(K2));
Psi2 = zeros(1,size(X2,1));
for id=1:size(K2,1)
    for j=1:k
         M=K2(idx(id,j),:)+M;
%         d22(id)=d(id,j)+d22(id);
    end
    M4(id,:)=M/k;
    dd(id,:)=K2(id,:)-M4(id,:);
    d22(id,:)=sqrt(dd(id,:)*dd(id,:)');
%     d22(id)=d22(id)/k;
    Psi2(id)=(min(d22(id),d21(id)))/(d22(id)+d21(id));
end

%compute Matrices L1 and L2
L1=K1.'-M2.';
L2=K2.'-M1.';

%compute between-class scatter matrices nabla1 and nabla2
nabla1 = zeros(size(L1,1));
for id=1:size(L1,2)
    nabla1=Psi1(id).*(L1(:,id)*L1(:,id).')+nabla1;
end
```

```
nabla2 = zeros(size(L2,1));
for id=1:size(L2,2)
    nabla2=Psi2(id).*(L2(:,id)*L2(:,id).')+nabla2;
end

%compute between-class scatter matrix nabla
nabla=(1/(size(L1,2)+size(L2,2)))*(nabla1+nabla2);

%compute within-class scatter matrix delta
onesN1 = ones(size(K1,1))/size(K1,1);
onesN2 = ones(size(K2,1))/size(K2,1);
delta = K1.'*(eye(size(K1,1))-onesN1)*K1...
    +K2.'*(eye(size(K2,1))-onesN2)*K2;
[Vdelta,Sdelta]=eig(delta);
%compute téta
%  beta = 10;
 beta = 10^-3;
% psi=[Psi1,Psi2];
% pp=sum(psi./n);
% mu1 =1-pp;
 mu1=eta;
nablaupd = nabla+beta.*eye(size(X,1));
detnablaupd=det(nablaupd);
res=isnan(nablaupd);
%  nnz(res)
 nablaupd(res==1)=0;
inverse1= pinv(nablaupd);
teta=mu1.*delta*inverse1*delta + eye(size(X,1));
  teta = round(teta,1);
  res1=isnan(teta);
%  nnz(res)
 teta(res1==1)=0;
% %compute eigenvalues and eigenvectors of téta
[Vteta,Steta]=eig(teta);
% Vteta=real(Vteta);
% Steta=real(Steta);
%compute coefficients for weights and kernel update
tetapos=Vteta*(Steta^(1/2))*pinv(Vteta);
tetaneg=Vteta*(Steta^(-1/2))*pinv(Vteta);


% compute kernel marices for linear kernel
K1opt= tetaneg*K1.';
K2opt= tetaneg*K2.';
Koptim = horzcat(K1opt,K2opt);
  Koptim=real(Koptim);
W=ones(size(X,1),1);
%        W=tetapos*ones(size(X,1),1);
% else W=Wupd;
% end
 Mdl1 =
fitcsvm(Koptim,labels,'KernelFunction','Rbf','KernelScale',sigma,'
BoxConstraint',C);
% ,'KernelFunction','Linear', 'BoxConstraint',...
%     C,'weights',W
```

```
%  Mdl1 =
fitcsvm(Koptim,labels,'KernelFunction','kernel','BoxConstraint',C)
;
% [label,score,cost] = predict(Mdl1,K);
% [XX,YY,~,AUC3] = perfcurve(labels,score(:,1),-1);
end
```

5.0.8 SVCM (Matlab)

```
function [atrain, btrain, g, inds, inde, indw,Qc] = svcm_train(x,
y, C,sigma);
% function [a, b, g, inds, inde, indw] = svcm_train(x, y, C);
%       support vector classification machine
%       incremental learning
%       uses "kernel.m"
%
%       x: independent variable, (L,N) with L: number of points;
N: dimension
%       y: dependent variable, (L,1) containing class labels (-1
or +1)
%       C: soft-margin regularization constant
%
%       a: alpha coefficients (to be multiplied by y)
%       b: offset coefficient
%       g: derivatives (adding one yields margins for each point)
%       inds: indices of support vectors
%       inde: indices of error vectors
%       indw: indices of wrongly classified leave-one-out vectors

%%%%%%%%%% version 1.11; last revised 06/07/2002; send comments to
gert@jhu.edu %%%%%%%%%%

%%% GLOBAL VARIABLES:
global doloo online query terse verbose debug memoryhog visualize
eps sigma
if isempty(doloo)
    doloo = 0;                          % perform loo at end of
training
end
if isempty(online)
    online = 0;                         % take data in the order it
is presented
end
if isempty(query)
    query = 0;                          % choose next point based on
margin distribution
end
if isempty(terse)
    terse = 0;                          % print out only final
results
end
if isempty(verbose)
    verbose = 0;                        % print out details of
intermediate results
end
if isempty(debug)
    debug = 0;                          % use only for debugging;
slows it down significantly
```

```
end
if isempty(memoryhog)
    memoryhog = 0;                      % use more memory; good only
if kernel dominates computation
end
if isempty(visualize)
    visualize = 1;                      % record and plot trajectory
of coeffs. a, g, and leave-one-out g
end
if isempty(eps)
    eps = 1e-6;                         % margin "margin"; makes Q
strictly positive definite
end

%%% END GLOBAL VARIABLES

[L,N] = size(x);
[Ly,Ny] = size(y);
if Ly~=L
    fprintf('svcm_train error: x and y different number of data
points (%g/%g)\n\n', L, Ly);
    return
elseif Ny~=1
    fprintf('svcm_train error: y not a single variable (%g)\n\n',
Ny);
    return
elseif any(y~=-1&y~=1)
    fprintf('svcm_train error: y takes values different from {-
1,+1}\n\n');
    return
end

eps2 = 2*eps/C;
tol = 1e-6;         % tolerance on derivatives at convergence, and
their recursive computation

% fprintf('Support vector soft-margin classifier with incremental
learning\n')
% fprintf('  %g training points\n', L)
% fprintf('  %g dimensions\n\n', N)

keepe = debug|memoryhog;            % store Qe for error vectors as
"kernel cache"
keepr = debug|memoryhog;            % store Qr for recycled support
vectors as "kernel cache"
if verbose
    terse = 0;
    if debug
        fprintf('debugging mode (slower, more memory
intensive)\n\n')
    elseif memoryhog
        fprintf('memoryhog active (fewer kernel evaluations, more
memory intensive)\n\n')
    end
    fprintf('kernel used:\n')
    help kernel
    fprintf('\n')
end
```

```
atrain = zeros(L,1);                    % coefficients, sparse
btrain = 0;                             % offset
W = 0;                                % energy function
g = -(1+eps)*ones(L,1);               % derivative of energy
function

inds = [];                          % indices of support vectors;
none initially
inde = [];                          % indices of error vectors; none
initially
indo = (L:-1:1)';                   % indices of other vectors; all
initially
indr = [];                          % indices of "recycled" other
vectors; for memory "caching"
indl = [];                          % indices of leave-one-out
vectors (still to be) considered
indw = [];                          % indices of wrongly classified
leave-one-out vectors
ls = length(inds);                  % number of support vectors;
le = length(inde);                  % number of error vectors;
la = ls+le;                         % both
lo = length(indo);                  % number of other vectors;
lr = length(indr);                  % number of recycled vectors
lw = length(indw);                  % number or wrongly classified
leave-one-out vectors
processed = zeros(L,1);             % keeps track of which points
have been processed
R = Inf;                            % inverse hessian (a(inds) and b
only)
Qs = y';                            % extended hessian; (a(inds)
plus b, and all vectors)
Qe = [];                            % same, for inde ("cache" for
Qs)
Qr = [];                            % same, for indr ("cache" for
Qs)
Qc = [];                            % same, for indc (used for gamma
and Qs)
if visualize                        % for visualization
%     figure(4)
%     hold off
%     clf
%     axis([-0.1*C, 1.1*C, -1.2, 0.2])
    gctraj = [];
%     figure(5)
%     hold off
%     clf
%     figure(6)
%     hold off
%     clf
end

iter = 0;                           % iteration count
memcount = 0;                       % memory usage
kernelcount = 0;                    % kernel computations, counted
one "row" (epoch) at a time
training = 1;                       % first do training recursion
...
```

```
leaveoneout = 0;                    % ... then do leave-one-out
sequence (with retraining)
indc = 0;                           % candidate vector
indco = 0;                          % leave-one-out vector
indso = 0;                          % a recycled support vector;
used as buffer
free = atrain(indo)>0|g(indo)<0;        % free, candidate support
or error vector
left = indo(free);                  % candidates left
continued = any(left);
while continued                      % check for remaining free
points or leave-one-outs to process

    % select candidate indc
    indc_prev = indc;
    if online & indc_prev>0
        if query
            processed(indc_prev) = 1;      % record last point in
the history log
        else
            processed(1:indc_prev) = 1;    % record last and all
preceding points
        end
    end
    if query
%       [gindc, indc] = max(g(left));      % closest to the margin
        [gindc, indc] = min(g(left));      % greedy; worst margin
        indc = left(indc);
    else
        indc = left(length(left));         % take top of the stack,
"last-in, first-out"
    end

    % get Qc, row of hessian corresponding to indc (needed for
gamma)
    if keepr & lr>0 & ...           % check for match among recycled
vectors
        any(find(indr==indc))
            ir = find(indr==indc);                % found, reuse
            Qc = Qr(ir,:);
            indr = indr([1:ir-1,ir+1:lr]);        % ... remove
from indr
            Qr = Qr([1:ir-1,ir+1:lr],:);          % ... and Qr
            lr = lr-1;
    elseif indc==indso              % support vector from previous
iteration, leftover in memory
        Qc = Qso;
    elseif ls>0 & ...               % check for match among support
vectors
        any(find(inds==indc))
            is = find(inds==indc);                % found, reuse
            Qc = Qs(is+1,:);
    elseif keepe & le>0 & ...       % check for match among stored
error vectors
        any(find(inde==indc))
            ie = find(inde==indc);                % found, reuse
            Qc = Qe(ie,:);
```

```
    elseif indc~=indc_prev          % not (or no longer) available,
compute
        xc = x(indc,:);
        yc = y(indc);
        Qc = (yc*y').*(kernel(xc,x));

        Qc(indc) = Qc(indc)+eps2;
        kernelcount = kernelcount+1;
    end

    % prepare to increment/decrement z = a(indc)' or y(indc)*b,
subject to constraints.
    % move z up when adding indc ((re-)training), down when
removing indc (leave-one-out or g>0)
    upc = ~leaveoneout & (g(indc)<=0);
    polc = 2*upc-1;                 % polarity of increment in z
    beta = -R*Qs(:,indc);           % change in [b;a(inds)] per
change in a(indc)
    if ls>0
        % move z = a(indc)'
        gamma = Qc'+Qs'*beta;       % change in g(:) per change in z
= a(indc)'
        z0 = atrain(indc);              % initial z value
        zlim = max(0,C*polc);       % constraint on a(indc)
    else % ls==0
        % move z = y(indc)*b and keep a(indc) constant; there is
no a(:) free to move in inds!
        gamma = y(indc)*Qs';        % change in g(:) per change in z
= y(indc)*b
        z0 = y(indc)*btrain;            % initial z value
        zlim = polc*Inf;            % no constraint on b
    end
    gammac = gamma(indc);
    if gammac<=-tol
        fprintf('\nsvcm_train error: gamma(indc) = %g <= 0 (Q not
positive definite)\n\n', gammac)
    elseif gammac==Inf
        fprintf('\nsvcm_train error: gamma(indc) = %g (Q rank
deficient)\n\n', gammac)
        return
    end

    % intrinsic limit: g(indc) = 0, where indc becomes support
vector
    if ~leaveoneout                 % only consider when training
indc, not when removing indc
        zlimc = z0-g(indc)'./gammac;
    else                            % leave-indc-out!
        zlimc = polc*Inf;
    end

    % support vector constraints: 0<=a(inds)<=C
    zlims = Inf*polc;                               % by default,
immaterial
    if ls>0
        is = find(inds==indc);
        if any(is)                                  % leave-indc-out,
remove from inds
```

```
            zlims = z0;                                  % clamp z; no
change to variables
        else
            betaa = beta(2:ls+1);                        % beta terms
corresponding to a(inds)  (not b)
            void = (betaa==0);                           % void zero betaa
values ...
            if any(any(~void))
                warning off % suppress div. by 0
                zlims = z0+(C*(betaa*polc>0)-atrain(inds))./betaa;
                warning on
                zlims(void) = polc*Inf;                  % ... which don't
enter the constraints
                [zmins, is] = min(zlims*polc,[],1);
                imin = find(zlims==zmins);
                if length(imin)>1
                    [gmax, imax] = max(abs(betaa(imin)),[],1);
                    is = imin(imax);
                end
                zlims = zmins*polc;                      % pick tightest
constraint
            end
        end
    end

    % error vector constraints: g(inde)<=0
    zlime = Inf*polc;                                    % by default,
immaterial
    if le>0
        ie = find(inde==indc);
        if any(ie)                                       % leave-indc-out,
remove from inde
            zlime = z0;                                  % clamp z; no
change to variables
        else
            gammae = gamma(inde);
            void = (gammae*polc<0)|(gammae==0);          % void g moving
down, or zero gamma...
            if any(any(~void))
                warning off % suppress div. by 0
                zlime = z0-g(inde)./gammae;
                warning on
                zlime(void) = polc*Inf;                  % ... which don't
enter the constraints
                [zmine, ie] = min(zlime*polc,[],1);
                imin = find(zlime==zmine);
                if length(imin)>1
                    [gmax, imax] = max(abs(gammae(imin)),[],1);
                    ie = imin(imax);
                end
                zlime = zmine*polc;                      % pick tightest
constraint
            end
        end
    end

    % ordinary vector constraints: g(indo)>=0 (only for those that
already are)
```

```
    zlimo = Inf*polc;                                % by default,
immaterial
    if lo>0
        gammao = gamma(indo);
        void =
(indo==indc)|(g(indo)<0)|(gammao*polc>0)|(gammao==0);
                                                     % void c, g
negative, g moving up, or zero gamma...
        if online
            void = void|~processed(indo);            % ... or, if
online, points not seen previously,...
        end
        if any(any(~void))
            warning off % suppress div. by 0
            zlimo = z0-g(indo)./gammao;
            warning on
            zlimo(void) = polc*Inf;                  % ... which don't
enter the constraints
            [zmino, io] = min(zlimo*polc,[],1);
            imin = find(zlimo==zmino);
            if length(imin)>1
                [gmax, imax] = max(abs(gammao(imin)),[],1);
                io = imin(imax);
            end
            zlimo = zmino*polc;                      % pick tightest
constraint
        end
    end

    % find constraint-satisfying z
    [z,flag] = min([zlim;zlimc;zlims;zlime;zlimo]*polc);
    z = z*polc;
    if (z-z0)*polc<0
        fprintf('\nsvcm_train error: z-z0 of wrong polarity (Q not
positive definite)\n\n')
        return
    end

    if verbose & ~leaveoneout & abs(z-z0)<eps
        fprintf('%g*', flag)                 % procrastinating
iteration!  no progress made
    end

    % update a, b, g and W from z-z0
    if ls>0                                  % z = a(indc)
        atrain(indc) = z;
        btrain = btrain+(z-z0)*beta(1);
        atrain(inds) = atrain(inds)+(z-z0)*beta(2:ls+1);
        W = W+(z-z0)*(g(indc)'+0.5*(z-z0)*gammac);
% energy
    else                                     % z = y(indc)*b
        btrain = y(indc)*z;
    end
    g = g+(z-z0)*gamma;                      % update g
    iter = iter+1;
    if visualize & ~leaveoneout
        atraj(1:L,iter)=atrain;                  % record trajectory
of a(:) over time
```

```
        gtraj(1:L,iter)=g;
        ctraj(iter)=indc;
    end

    % bookkeeping: move elements across indc, inds, inde and indo,
and update R and Qs
    converged = (flag<3);                % done with indc; no other
changes in inds/inde
    incl_inds = 0;
    if flag==1                           % a(indc) reaches the
limits 0 or C, stop moving
        if upc                                    % a(indc)=C, add
to inde
            inde = [inde; indc];
            le = le+1;
            if keepe
                Qe = [Qe; Qc];
            end
            atrain(indc) = C;                       % should be
OK, just to avoid round-off
        else % ~upc                              % a(indc)=0,
indc stays in (or moves to) indo
            atrain(indc) = 0;                       % should be
OK, just to avoid round-off
        end
    elseif flag==2                       % add indc to support
vectors ...
        incl_inds = 1;
        indb = indc;                              % ... store in
buffer indb for now
        Qb = Qc;
    elseif flag==3                       % one of support vectors
becomes error or other vector
        indso = inds(is);                         % outgoing inds
        Qso = Qs(is+1,:);                         % could be reused
later
        free_indc = (indc==indso);                % leave-indc-out:
indc is part of inds
        if beta(is+1)*polc<0 | free_indc          % a(indso)=0 or
indso=indc, move to indo
            if ~free_indc
                if keepr
                    indr = [indr; indso];         % also recycle
into indr for later use
                    Qr = [Qr; Qs(is+1,:)];
                    lr = lr+1;
                end
                if atrain(indso)>C/2
                    fprintf('svcm_train error: a(indso)=%g; 0
anticipated\n', atrain(indso));
                end
                atrain(indso) = 0;                  % should be
OK, just to avoid round-off
            end
            g(indso) = 0;                         % same
            indo = [indo; indso];
            lo = lo+1;
```

```
        else % beta(is+1)*polc>0 & ~free_indc    % a(indso)=C, move
to inde
            if atrain(indso)<C/2
                fprintf('svcm_train error: a(indso)=%g; C
anticipated\n', atrain(indso));
            end
            if keepe
                Qe = [Qe; Qs(is+1,:)];          % save to memory
cache
            end
            atrain(indso) = C;                       % should be
OK, just to avoid round-off
            g(indso) = 0;                       % same
            inde = [inde; indso];
            le = le+1;
        end
        inds = inds([1:is-1,is+1:ls]);          % remove from inds
        stripped = [1:is,is+2:ls+1];            % also from Qs and
R ...
        Qs = Qs(stripped,:);
        ls = ls-1;
        if ls > 0
            if R(is+1,is+1)==0
                fprintf('\nsvcm_train error: divide by zero in R
contraction\n')
                R(is+1,is+1)=1e-8;
            end
            R = R(stripped,stripped)-
R(stripped,is+1)*R(is+1,stripped)/R(is+1,is+1);
        else % no support vectors left
            R = Inf;
        end
    elseif flag==4                          % one of error vectors
becomes support/other vector
        indeo = inde(ie);                           % outgoing inde
        if indc==indeo                              % leave-indc-out
            indo = [indo; indeo];                   % add inde(ie) to
other vectors
            lo = lo+1;
        else
            incl_inds = 1;                          % add inde(ie) to
support vectors ...
            indb = indeo;                           % ... store in
buffer indb for now
            if keepe
                Qb = Qe(ie,:);                      % recover from Qe
cache
            elseif indb==indso
                Qb = Qso;                           % recover from
previous outgoing support vector
            else                                    % not in memory
either way--- recompute

                Qb = (y(indb)*y').*(kernel(x(indb,:),x));
                Qb(indb) = Qb(indb)+eps2;
                kernelcount = kernelcount+1;
            end
        end
```

```
        inde = inde([1:ie-1,ie+1:le]);            % remove from
inde
        if keepe
            Qe = Qe([1:ie-1,ie+1:le],:);          % remove from Qe
        end
        le = le-1;
    elseif flag==5                          % one of other vectors
becomes support vector
        indoo = indo(io);                           % outgoing indo
        incl_inds = 1;                              % add indo(io) to
support vectors ...
        indb = indoo;                               % ... store in
buffer indb for now
        if keepr & lr>0 & any(find(indr==indb))   % check for match
among recycled vectors
            ir = find(indr==indb);                          % found,
reuse
            Qb = Qr(ir,:);
            indr = indr([1:ir-1,ir+1:lr]);                % ...
remove from indr
            Qr = Qr([1:ir-1,ir+1:lr],:);                  % ... and
Qr
            lr = lr-1;
        elseif indb==indso
            Qb = Qso;                               % recover from
previous outgoing support vector
        else                                        % not in memory
either way--- recompute
            Qb = (y(indb)*y').*(kernel(x(indb,:),x));
            Qb(indb) = Qb(indb)+eps2;
            kernelcount = kernelcount+1;
        end
        indo = indo([1:io-1,io+1:lo]);            % remove from
indo
        lo = lo-1;
    end

    if incl_inds                           % move buffer indb into
support vectors inds
        inds = [inds; indb];                        % move to inds
...
        ls = ls+1;
        Qs = [Qs; Qb];                              % and also Qs and
R ...
        if ls==1                                    % compute R
directly
            R = [-Qb(indb), y(indb); y(indb), 0];
        else                                        % compute R
recursively
            if flag==2                              % from indc; use
beta and gamma
                    pivot = gamma(indb);
            else % flag==4                          % from inde;
compute beta and pivot
                beta=-R*Qs(1:ls,indb);
                pivot = [beta',1]*Qs(:,indb);
            end
```

```
            end
        end
    end

    % debugging mode: check for consistency; a, R, g and W
    if debug
        f = find(isnan(atrain));
        if any(f)
           fprintf('svcm_train error: a(%g) = %g\n', [f,
atrain(f)]')
        end
        f = find(isnan(g));
        if any(f)
           fprintf('svcm_train error: g(%g) = %g\n', [f, g(f)]')
        end
        f = find(atrain>C);
        if any(f)
           fprintf('svcm_train error: a(%g) = %g > C\n', [f,
atrain(f)]')
        end
        f = find(atrain<0);
        if any(f)
           fprintf('svcm_train error: a(%g) = %g < 0\n', [f,
atrain(f)]')
        end
        f = inde(find(atrain(inde)>C|atrain(inde)<C));
        if any(f)
              fprintf('svcm_train error: a(%g) = %g ~= C\n', [f,
atrain(f)]')
        end
        if abs(y'*atrain)>tol
              fprintf('svcm_train error: y''*a = %g ~= 0
(tol=%g)\n', y'*atrain, tol);
        end
        Rdiv = max(max(abs(Qss*R-diag(ones(ls+1,1)))));
        if Rdiv>tol
            if flag==2|flag==4    % support vector added
                fprintf('svcm_train error: divergence %g in R
expansion (tol=%g; pivot=%g)\n',...
                        Rdiv, tol, pivot);
            elseif flag==3        % support vector removed
                fprintf('svcm_train error: divergence %g in R
contraction (tol=%g)\n', Rdiv, tol);
            else                  % no support vector added or
removed; strange...
                fprintf('svcm_train error: divergence %g in R ?
(tol=%g)\n', Rdiv, tol);
            end
        end
        if keepe&keepr            % only check g when Qe and Qr
are readily available
            greal = Qs'*[btrain;atrain(inds)]-(1+eps);
            if le>0
                greal = greal+Qe'*atrain(inde);
            end
            if lr>0
                greal = greal+Qr'*atrain(indr);
            end
```

```
            if max(abs(greal-g))>tol
                fprintf('svcm_train error: tolerance (tol=%g)
 exceeded in computation of g\n', tol);
            end
        end
        f = inds(find(abs(g(inds))>tol));
        if any(f)
            fprintf('svcm_train error: g(%g) = %g ~= 0
 (tol=%g)\n', [f, g(f), f*0+tol]');
        end
        f = inde(find(g(inde)>tol));
        if any(f)
            fprintf('svcm_train error: g(%g) = %g > 0 (tol=%g)\n',
 [f, g(f), f*0+tol]');
        end
        Wreal = 0.5*sum((g-btrain*y-1-eps).*atrain);        %
 energy
        if abs(Wreal-W)>tol*abs(W)
            fprintf('svcm_train error: energy W = %g ~= %g
 (tol=%g)\n', W, Wreal, tol);
        end
        inda = sort([indo;inds;inde]);
        if any(inda~=(1:L)')
            fprintf('svcm_train error: union [indo;inds;inde] does
 not equate entire set\n');
        end
        if ls~=length(inds)
            fprintf('svcm_train error: miscount in number of
 support vectors\n');
        end
        if le~=length(inde)
            fprintf('svcm_train error: miscount in number of error
 vectors\n');
        end
        if lo~=length(indo)
            fprintf('svcm_train error: miscount in number of other
 vectors\n');
        end
    end

    memcount = max(memcount,ls+keepe*le+keepr*lr+~(keepe&keepr));
 % kernel storage

    if verbose
        fprintf('    c: %g (#%g)', y(indc)>0, indc)
        if leaveoneout
            fprintf(', margin: %6.3g, gamma: %6.3g', g(indc)+1,
 gammac)
        end
        fprintf(' | s: ')
        fprintf('%g', y(inds)>0)
        if ls<6
            fprintf(' (')
            fprintf('#%g', inds)
            fprintf(')')
        end
        fprintf(' | e: 0:%g, 1:%g\n', sum(y(inde)<0),
 sum(y(inde)>0))
```

```
    end

    if converged                          % indc finished; report
        if ~training & ~leaveoneout       % retraining or tracing
back; uninteresting
            % nothing
        elseif ~terse & ~leaveoneout
%             fprintf(' [%3g%%] iter. %3g: %3g support vectors;
%3g error vectors;  energy %g\n', ...
%                 100-round(100*length(left)/L), iter, ls, le, W)
        elseif ~terse % & leaveoneout
%             fprintf(' [%3g%%] iter. %3g: %3g leave-one-out
errors', 100-round(100*length(indl)/L), iter, lw)
%             fprintf(' (# %g: margin %g)\n', indc, g(indc)+1)
        elseif ~leaveoneout % & terse
            if ls+le>la
%                 fprintf('+')
            elseif ls+le<la
%                 fprintf('-')
            else
%                 fprintf('=')
            end
        else % leaveoneout & terse
            if g(indc)<-1
%                 fprintf('x')
            else
%                 fprintf('o')
            end
        end
    la = ls+le;
    end

    % prepare for next iteration, if any
    indoc = indo(find(indo~=indco));       % indo other than indco
(leave-one-out index, if active)
    free = atrain(indoc)>0|g(indoc)<0;          % candidate
support/error vectors in indoc
    if any(free)
        left = indoc(free);                % candidates left, keep
(re-)training
        leaveoneout = 0;                   % interrupt leave-one-
out, if active
    else % ~any(free)                      % done; finish up and
(re-)initiate leave-one-out
        if training                        % first time around
(not re-training)
            % print out results of svm training
            if terse
%                 fprintf('\n\n  %g support vectors; %g error
vectors; energy %g\n\n', ...
%                                   ls, le, W)
            else
%                 fprintf('\n%4g epoch kernel evaluations (%3g%%
of run-time)\n',...
% %                           kernelcount,
round(kernelcount/(ls+le)*100))
%                 fprintf(  '%4g epoch vectors in memory  (%3g%%
of data)\n\n',...
```

```
%                           memcount, round(memcount/(N+1)*100))
            end
            if visualize

                % plot a trajectory
%                 figure(5)
%                 h=image(atraj/C*length(gray));
%                 h2=get(h,'Parent');
%                 set(h2,'YDir','normal')     % 'image' normally
reverts the y axis
%                 colormap(1-gray)
%                 xlabel('Iteration')
%                 ylabel('Coefficients \alpha_{\it{i}}')
%                 print ('C:\Users\Derbe\Desktop\atraj.eps', '-
deps' )

                % plot g trajectory
%                 figure(6)
%                 gmax = max(max(abs(gtraj)));
%                 h=image((gtraj/gmax+1)/2*length(gray));
%                 h2=get(h,'Parent');
%                 set(h2,'YDir','normal')     % 'image' normally
reverts the y axis
%                 grey = gray;
%                 redblue = min(grey,1-grey);
%                 redblue(:,1) = redblue(:,1)+1-grey(:,1);
%                 redblue(:,2) = 2*redblue(:,2);
%                 redblue(:,3) = redblue(:,3)+grey(:,3);
%                 colormap(redblue)
%                 xlabel('Iteration')
%                 ylabel('Coefficients {\it{g}_{\it{i}}}')
%                 print ('C:\Users\Derbe\Desktop\gtraj.eps', '-
deps' )
%
%
%                 save C:\Users\Derbe\Desktop\traj
%                 save C:\Users\Derbe\Desktop\atraj
%                 save C:\Users\Derbe\Desktop\gtraj
%                 save C:\Users\Derbe\Desktop\ctraj
            end
            if debug                          % store final result
to compare with retraining later
                afinal = atrain;
                gfinal = g;
                Wfinal = W;
            end

            % initiate leave-one-out, if desired
            if doloo
              leaveoneout = 1;
              indl = [inds;inde];             % support and error
vectors (others already correct)
              indw = indl(g(indl)<-1);        % leave-one-out errors
so far (g<-1) ...
              lw = length(indw);              % ... their number
              indl = indl(g(indl)>=-1);       % remove errors so far
from leave-one-out stack
```

```
              indco = indl(length(indl));    % pick first leave-
one-out index; top of stack ...
              left = indco;                  % ... and let
indc=indco (untrain; upc=0)
            else
              continued = 0;
            end

            training = 0;                    % don't ever visit
again!
        elseif ~leaveoneout                  % retrained or traced
back
            if debug&~any(find(indo==indco))
                % traced back; compare with previously trained
results
                if max(abs(atrain-afinal))>tol
                        fprintf('svcm_train error: final coeffs. a
exceed tolerance (tol=%g)\n', tol);
                end
                if max(abs(g-gfinal))>tol
                        fprintf('svcm_train error: final
derivatives g exceed tolerance (tol=%g)\n', tol);
                end
                if abs(W-Wfinal)>tol*abs(W)
                        fprintf('svcm_train error: final energy W
= %g ~= %g (tol=%g)\n', Wfinal, W, tol);
                end
            end
            if any(indl)
                indco = indl(length(indl)); % leave-one-out index;
top of stack
                left = indco;
                leaveoneout = 1;
            else % ~any(indl) % finished all leave-one-outs;
summarize, and done!
                continued = 0;
                ltw = sum(g<-1);             % number of training
errors
                if terse
%                       fprintf('\n\n  %g leave-one-out errors;',
lw)
%                       fprintf(' %g training errors\n\n', ltw)
                else % ~terse
%                       fprintf('\n%4g training points\n', L)
%                       fprintf(  '%4g support/error vectors
(%g/%g)\n', la, ls, le)
%                       fprintf(  '%4g leave-one-out errors
(%3.1f%%)\n', lw, lw/L*100)
%                       fprintf(  '%4g training errors
(%3.1f%%)\n', ltw, ltw/L*100)
%                       fprintf('\n%4g epoch kernel evaluations
(%3g%% of run-time)\n',...
%                         kernelcount,
round(kernelcount/(ls+le)*100))
%                       fprintf(  '%4g epoch vectors in memory
(%3g%% of data)\n\n',...
%                         memcount, round(memcount/(N+1)*100))
                end
```

```
                if visualize                    % plot g trajectory
%                     figure(1)
%                     hold off
%                     xlabel('\alpha_{\it{c}}')
%                     ylabel('\it{g_c}')
%                     axis([-0.1*C, 1.1*C, -1.2, 0.2])
%                     h=line([0,0,C,C],[-1,0,0,-1]);
%                     set(h,'Color',[0 0 0])
%                     set(h,'LineStyle',':')
%                     set(h,'LineWidth',[0.2])
%                     h=line([0,C],[-1,-1]);
%                     set(h,'Color',[0 0 0])
%                     set(h,'LineStyle','--')
%                     set(h,'LineWidth',[1.0])
%                      print ('C:\Users\Derbe\Desktop\gctraj.eps',
'-deps' )
                end
            end
            else % leaveoneout                  % leave-one-out
procedure
            if flag==1                          % finished leave-one-
out, now trace back
                % note: already decremented indl and updated
indw/lw above
                leaveoneout = 0;
                indco = 0;
            else                                % not done yet with
leave-one-out ...
                left = indco;                   % ... continue with
indc=indco
            end
        end
    end
end
```

الملخص

أدت تكنلوجيا المعلومات والاتصالات الجديدة، بما فيها ذلك الوسائط الاجتماعية، إلى إعادة تشكيل طريقة نشر المعلومات بأثارها الإيجابية والسلبية. هذه الوسائط الاجتماعية لا تخضع للإشراف واصبحت مرتعاً لنشاطات الروبوتات وقد أدى ذلك إلى تسريع نشر المعلومات المضللة وذلك لسبب عدم قدرت تكنلوجية المراقبة على اكتشاف الروبوتات بشكل دقيق نظراً لتطورها الدائم. لذلك سيختبر البحث قدرت شعاع الدعم الآلي المحلي العالمي كشف الروبوتات بشكل مرن. الإطار الفني يتكون من عدت مراحل من أولها طبقة الشبكات العصبونية الالتفافية, ويتبها طل من طبقة التجميع وغيرها. تم تقليل حد البيانات إلى ثلاثة أحجام: 70و 100و 130. بالإضافة إلى ذلك ، تم تصميم الشبكات العصبونية للتنبؤ بتأثير تغريدات بوت على تقلبات سوق الأسهم مع ارتباط معامل 96.53٪ الاصطناعية

جامعة البحرين

كلية العلوم

قسم برامج الدراسات العليا

أنظمة فعالة لاكتشاف الروبوتات و توقعات سوق الأسهم في الوقت الفعلي

مقدمة استيفاءً لمتطلبات الماجستير في علوم وتحليلات البيانات الضخمة

إعـــداد

الشيخة ريم بنت عبدالله آل خليفة

الرقم الجامعي: 202012269

إشـــراف

د. رياض بن محمد قسنطيني

أستاذ مشارك

جامعة البحرين

بروفسور هاتم مصري

بروفسور

جامعة البحرين

مـملكة البـــحرين

يونيو 2023م