University of Piraeus

Department of Digital Systems

M.Sc. Digital Systems and Services

Big Data and Analytics

Thesis by Kalderemidis Alexandros St.

Experimental Evaluation of Online Classification Models

Supervisor: Telelis Orestis

2018-2019

Abstract

In this thesis we compare experimentally several online machine learning methods, for classification. The term "online" refers to the fact that the classifier examines the training examples one by one and not as a whole dataset, and additionally the classifier's decision is irrevocable. We evaluate their performance on three different datasets. Specifically, the online learning methods used are Perceptron, Support Vector Machines (SVMs), Winnow, Logistic Regression (LR), along with kernel methods for Perceptron and SVM. The datasets used cover real-life application fields. The first dataset is used to predict spam e-mails, the second for cancellation of shipment and the third for constant surveillance of people suffering epilepsy. After transforming the datasets to the desired form, we implement the said methods and algorithms from scratch using Python, evaluate the models created and present our results. Our findings are summarized as follows. First, the kernel methods do not outperform the non-kernel ones, due to the lack of quantity of training examples. Second, the Winnow performance was poor, probably due to its lack of negative correlation. Finally, we find that the deviation between the models we created, and the corresponding static models created by Python's sk-learn, was insubstantial, therefore proving our models' performance was satisfactory.

Περίληψη

Σε αυτή τη διατριβή συγκρίνουμε πειραματικά διάφορες μεθόδους άμεσης μηχανικής μάθησης (online machine learning) για κατηγοριοποίηση. Ο όρος "άμεση" αναφέρεται στο γεγονός ότι ο κατηγοριοποιητής εξετάζει τα παραδείγματα ένα-ένα και όχι σαν ολόκληρο σετ δεδομένων, συν το ότι η απόφαση του κατηγοριοποιητή είναι μη ανακλήσιμη. Αξιολογούμε την απόδοσή τους σε τρία διαφορετικά σύνολα δεδομένων. Συγκεκριμένα, οι ηλεκτρονικές μέθοδοι μάθησης που χρησιμοποιούνται είναι το Perceptron, οι μηχανές διανύσματος στήριξης (SVM), ο Winnow, λογιστική παλινδρόμηση (LR), καθώς και μέθοδοι πυρήνα για Perceptron και SVM. Τα σύνολα δεδομένων που χρησιμοποιούνται καλύπτουν πολλά πεδία εφαρμογής στη ζωή. Το πρώτο σύνολο δεδομένων χρησιμοποιείται για την πρόβλεψη μηνυμάτων ηλεκτρονικού ταχυδρομείου ανεπιθύμητης αλληλογραφίας, το δεύτερο για την ακύρωση της αποστολής παραγγελίας και το τρίτο για τη συνεχή παρακολούθηση των ατόμων που πάσχουν από επιληψία. Μετά τη μετατροπή των συνόλων *δεδομένων στην επιθυμητή μορφή, υλοποιούμε τις εν λόγω μεθόδους και αλγόριθμους από το μηδέν χρησιμοποιώντας Python, αξιολογούμε τα μοντέλα που δημιουργήσαμε και παρουσιάζουμε τα αποτελέσματά μας. Τα ευρήματά μας συνοψίζονται ως εξής. Πρώτον, οι μέθοδοι πυρήνα δεν ξεπερνούν σε απόδοση τις γραμμικές μεθόδους, λόγω έλλειψης ποσότητας παραδειγμάτων εκπαίδευσης. Δεύτερον, η απόδοση του Winnow ήταν φτωχή, πιθανώς λόγω της έλλειψης αρνητικής συσχέτισης. Τέλος, εξακριβώνουμε πως η απόκλιση μεταξύ των μοντέλων που δημιουργήσαμε, με τα αντίστοιχα στατικά μοντέλα που δημιούργησε η βιβλιοθήκη sk-learn της Python, ήταν ελάχιστη, συνεπώς αποδεικνύεται πως η επίδοση των μοντέλων μας ήταν ικανοποιητική.*

Contents

1. Introduction

This research focuses on the use of online machine learning algorithms in order to provide predictive models for real world problems. We compare the performance of six different online learning algorithms (Perceptron, SVM, Logistic Regression, Winnow, RBF-Perceptron, RBF-SVM) and try to minimize their error rate by tuning their hyper-parameters. The datasets used are Spam, where we predict if a new mail is spam, Online Retail, where we predict if an incoming order is going to be cancelled, and Epileptic Seizure, where we monitor people suffering from epilepsy and we monitor if they are currently having an epileptic seizure episode. Our results show that the kernel methods are not necessarily more reliant with respect to non-kernel methods with regards to performance, the Winnow performance was poor because of its inability for negative correlation and that the deviation between our models and sk-learn's models was insubstantial.

In the first chapter we discuss machine learning in general, meaning how it works, what are the linear methods, how we measure the performance of an algorithm and how to overcome the obstacle of having to deal with non-linear decision boundaries. In the second chapter we focus on online machine learning, that is the changes that have to take place in order to turn a method from offline to online, and how to evaluate their performance. In the third chapter we provide more information on the specifics of the datasets and our methodology, along with providing our results. In the forth chapter we discuss the conclusions of our research and provide possible improvements or changes concerning our research. Finally, in the fifth chapter you can find the citation used in order to actuate this research.

1.1 Different Types of Machine learning

The machine learning training process is as follows. First, we create a *weight vector* which we initialize with values equal to zero or a small arbitrary number. Then, we separate the *features* or *attributes* of the dataset from what we want to predict (called *class* or *label*). The algorithm then takes a new incoming feature vector from the dataset, uses the weight vector and predicts the label. If necessary, which most commonly means if the prediction was wrong, the weight vector is updated using an update rule. This process is continued until the whole dataset has been parsed, or another stop condition is met.

The most common and well-known type of machine learning is the ***supervised*** learning. In supervised learning, when we train our model, we already know the labels of the data and train the model as such. For example, let's say we are working for a marketing agency and we want to run specific ads for our clients. In this scenario we collect information for our client and based on his behavior and preferences, we build a model feeding it input-output pairs, or in other words, we have the knowledge of all labels of our data. Supervised learning algorithms include K-nearest neighbors, decision trees, neural networks, naive Bayes, logistic regression, support vector machine and many more.

The next type of machine learning is ***unsupervised*** learning. In unsupervised machine learning, we have no prior knowledge of our labeled data. Or in other words, we don't know the labels of our data. Using this type of machine learning, we try to find the commonalities in data. Let's assume that a new microbe is discovered and we want to learn if it dangerous, but obviously, we cannot test it on humans or animals. A way to approach this problem, would be to train a model of some microbes that are known to be dangerous and other harmless, and if the new microbe belongs to the dangerous group, we can decide (with a certain error margin) that it could be dangerous, or vice versa. Unsupervised learning methods include Clustering, dimensionality reduction, principal component analysis, anomaly detection and others.

Between supervised and unsupervised, there is ***semi-supervised*** learning. In semi-supervised learning, we have *some* knowledge of the labeled data, meaning that we have some information which is limited, and we want to maximize the use of it. Big companies like YouTube, Amazon and Netflix use semi-supervised learning to improve customer experience. Let's say we just subscribed on Netflix and have only watched one movie, the Lord of the Rings. Other people have also watched this movie, and *based on them* (rather than us), they also liked the Harry Potter movie, so the Netflix algorithm will suggest Harry Potter to us. Using this method, Netflix doesn't just know that we will probably like Harry Potter, but rather it knows the probability of us liking every single movie in their database! semi-supervised learning methods include generative models, low-density separation, graph-based methods, heuristic approaches and latent factor models.

Another type of machine learning is ***reinforcement*** learning. In reinforcement learning, rather than data, the machine is given an environment, in which it should maximize its reward. A very common use of reinforcement learning is in several games, such as the game of chess. The machine starts out as "unintelligent", meaning it only knows how to play the game, but not *well*. The *human agent*, the person training the machine, provides some knowledge of the game, either through actual games of good players, or through general chess knowledge (castle as soon as possible, don't move the pawns in front of your king, take control of the center etc.). The machine slowly learns what moves to prioritize and avoid, but always seeks its reward mechanism, for example, victory

1.2 Linear Methods

When dealing with classification problems, many well-known algorithms fall under the category of linear methods, due to their efficiency and accuracy. Linear methods take their name from the fact that they use a line to separate the labels of the data, using the equation:

$$\boldsymbol{w} * \boldsymbol{x} + b = 0$$

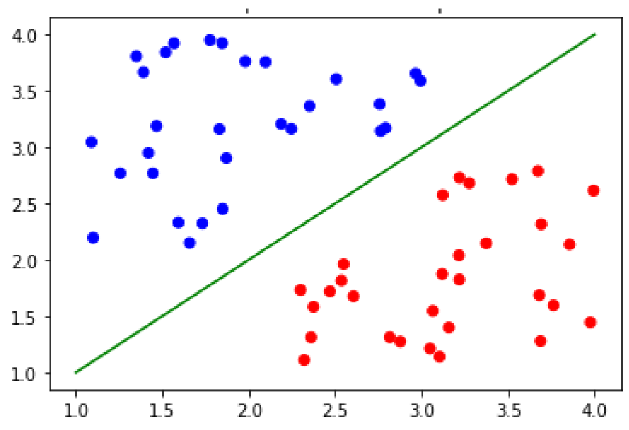

***Figure 1**: Example of linear method separation of data*

where y is the label, ***w*** is the weight vector which indicates the importance of a given attribute, ***x*** is the attribute vector and *b* is a constant. Depending on which algorithm we use, the goal of a linear method is to optimize this line according to a criterion measuring the accuracy of the dataset.

1.3 Perceptron

The Perceptron, as proposed by Rosenblatt (1958) is another well-known machine learning algorithm that many scientists of the field have studied. Since that time, further research has evolved this algorithm, each towards a certain improvement. In [20], the researchers manage to develop distributed training strategies, and thereby reducing the training time, when computer clusters are available. Another research [22] focuses on improving the accuracy by bounding the number of mistakes, using eigenvalues and a parameter controlling the sensitivity of the algorithm. Using this technique the experimental results showed a twenty to thirty percent increase in accuracy compared to the Perceptron. Further research with the goal of increasing the accuracy has been done in [17] with the introduction of the boosting algorithms, whose idea is to combine many weak learners in order to create a strong and accurate classifier.

The perceptron is a popular machine learning algorithm proposed by (Rosenblatt, 1958; Collins, 2002). It is one of the most famous and oldest machine learning methods that is used for both classification and regression.

Consider a dataset with target values *y*(where *y*∈{-1, 1}), and a set of attributes ***x***. This algorithm maintains a weight vector ***w*** with initial values equal to zero or a random number between minus one and one**.** In each iteration *t*, it receives a new data point with attributes $\boldsymbol{x_t}$**,** and using the ***w*** predicts the class of the new data point, using the $\boldsymbol{sign}(\boldsymbol{w} * \boldsymbol{x})$ function, where:

$$\boldsymbol{w} * \boldsymbol{x} = \sum_{i=1}^{m} w_i x_i$$

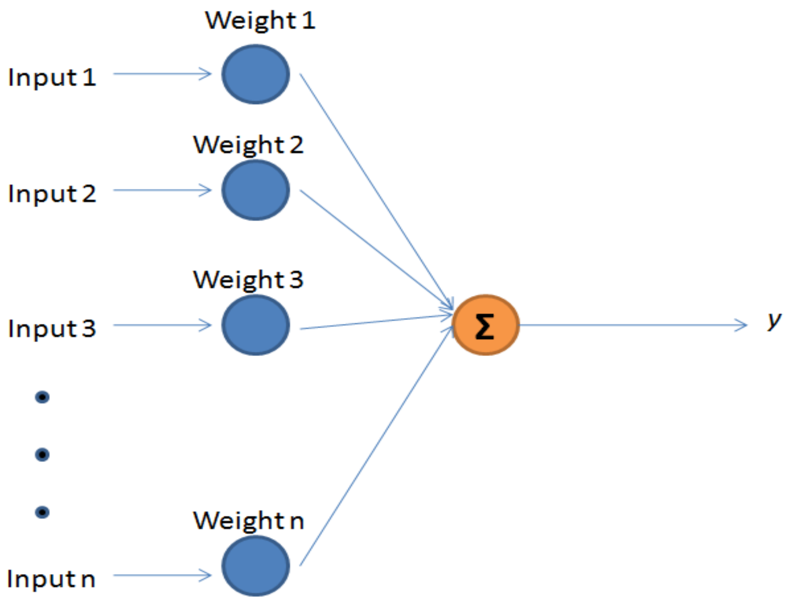

Figure 2*: The Perceptron algorithm*

The model then checks if the prediction was correct or not. In case the model misclassified the data point, the weight vector ***w*** is updated using the update rule:

$$\boldsymbol{w_j} := \boldsymbol{w_j} + n * y_t * \boldsymbol{x_t}$$

where *n* is the *learning rate,* a constant used to control the step size of the convergence of a model. The algorithm stops when it performs a certain number of iterations. For more information, please consult [32].

1.4 Support Vector Machines

Online support vector machines have been thoroughly studied by many machine learning experts throughout the years. It was first proposed by [6], who also had the revolutionary idea of creating a prediction model whose training time was independent of the size of the dataset. In [24], using a new design of storage and numerical operations, managed to train SVM models five to twenty times faster than the previously known models. Another breakthrough regarding SVMs was the creation of LIBSVM [15], a library available to the public which is evolving until today and provides many options to the user such as classification, regression, probability estimates, feature selection and more with excellent accuracy and training times. Finally, in [26], we come across a radical idea of suppressing the effect of outliers and therefore decreasing training time and improving robustness without sacrificing generalization or performance, along with gaining the ability of better scaling of the data.

The Support Vector Machine (SVM) is a machine learning method used for classification and regression. SVMs are one of the modern ML methods and have gained popularity due to their strong theoretical background and exceptional results.

In order to separate the data points of a given dataset with two classes {-1,1}, there are infinite decision boundaries. The way SVMs work, they try to find the optimal line (or hyperplane) to divide the data points. The optimal boundary is the line that has equal distance from the closest points of each class.

Consider a dataset with binary target values *y* (where $y \in \{-1, 1\}$), and a set of attributes ***x***. SVMs also keep a weight vector in memory, with initial values equal to either zero or some small random number $w_0 \in [0,1]$. Each iteration t, the SVM performs a prediction using the **sign(**$\boldsymbol{w}*\boldsymbol{x}_t$**)** function (where $\boldsymbol{x}_t$ the attributes of the new data point). Then, the algorithm verifies if its prediction was correct or not and updates the weight vector ***w*** accordingly.

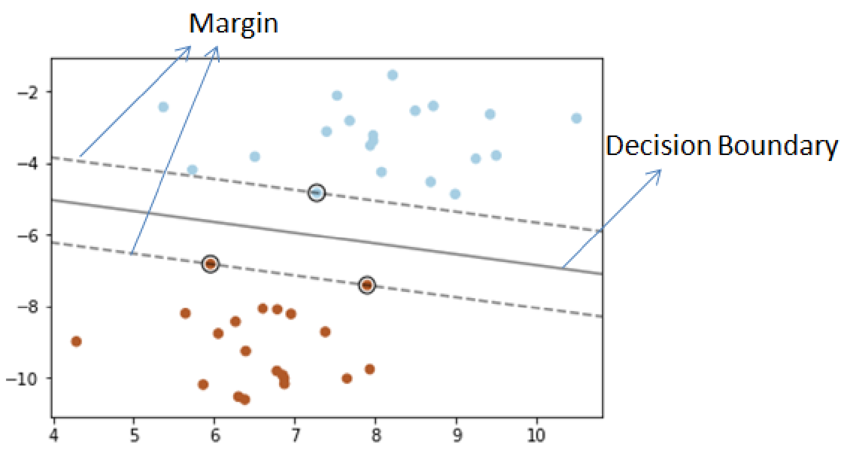

***Figure 3**: SVM boundaries*

In a problem with two classes, we assume that the data points are separable with a straight line $\boldsymbol{w} * \boldsymbol{x} + b$, which indicates the existence of ***w*** and b such that for each training example $i = 1,2, \dots, m$ the sign of $\boldsymbol{w} * \boldsymbol{x_i} + b$ is the same as the sign of y_i. Therefore, we solve the following equation to find ***w*** and b:

$$w, b = arg \min_{w,b} \frac{1}{2} \left| |w| \right|^2$$

$$so\ that \quad y_i(w * x_i - b) \geq 1, with\ 1 \leq i \leq m$$

This mathematical optimization can be solved by either Convex Optimization, Gradient Descent[6] and Sequential Minimal Optimization[15].

For more information on the topic, please visit [33].

1.5 Logistic Regression

Logistic regression is a well known machine learning method that has been studied extendedly by many scientists of the field. In [14], Francis Bach proves that by having an adaptive step-size (instead of a constant one) the convergence rate improves, therefore reducing the training time. In [29], a new kind of logistic regression which combines a forgetting factor and posterior predictive distribution is proposed, for the purpose of overcoming confidentiality breaches by only temporarily storing the data. Finally, in [21] the researchers study the use of logistic regression on encrypted data along with reducing the computational cost and application of parallelization techniques.

Logistic regression, despite its misleading name, is a probabilistic binary classifier. It is a machine learning algorithm and belongs to the family of linear methods. Logistic regression uses the sigmoid function to calculate the probability of the label of an example be "0" or "1"like so:

$$p(y = 1\,|\boldsymbol{x}) = h(\boldsymbol{x}) = \frac{1}{1 + \exp(\,-\boldsymbol{w} * \boldsymbol{x})}$$

$$p(y = 0\,|\boldsymbol{x}) = \ 1 - p(y = 1\,|\boldsymbol{x}) = 1 - h(\boldsymbol{x})$$

where *y* is the label, ***w*** is the weight vector, ***x*** is the feature vector and *h,* the hypothesis. Note that the sigmoid function is $(\alpha) = \frac{1}{1+\exp(-\alpha)}$. For every training example, the logistic regression model utilizes the sigmoid function, using the current weights and attributes, and predicts if the label of the example is 0 or 1. In order to train a logistic regression model, we have to optimize the weights by calculating the following:

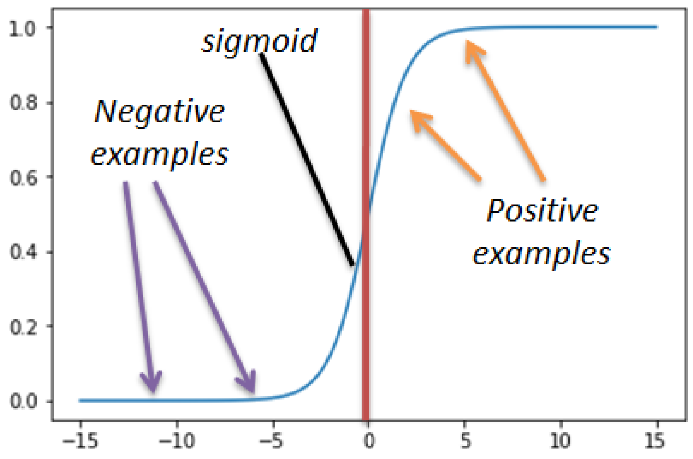

***Figure 5**: Logistic Function*

$$\dot{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}}\{\frac{1}{m} * \sum_{i=1}^{m} \log\left(1 \ + exp\left(-y_i(\boldsymbol{w} * \boldsymbol{x})\right)\right) \}$$

where w^T the transposed weight vector. For more information, consult [34].

1.6 Winnow

The Winnow algorithm is an online learning algorithm that has been researched thoroughly. One of the most important research is [28], which proposes a regularized Winnow, that is the winnow algorithm but with a modification so that the weights do not exceed a certain threshold. Using this modification, both accuracy and speed are increased based on the results. Another very interesting paper [13] explains how it utilizes the Modified Balanced Winnow algorithm in order to detect various types of computers attacks in real time. Finally, Winnow has been used in various research for text analysis purposes, for example in [27,28], due to the algorithm's natural robustness to irrelevant features.

The Winnow algorithm is a machine learning algorithm created by Littlestone [8] and is used for classification. It is similar to the Perceptron with the exception that the weight update is multiplicative, instead of additive. The advantage of winnow is that it tends to perform better when the dataset has many irrelevant attributes and few relevant. There are different versions of this algorithm, we are going to examine the most basic, the "Winnow 1".

The algorithm initializes a weight vector ***w*** with initial values equal to one and takes as input boolean features ***x.*** The algorithm multiplies the weight vector with the input vector, makes a prediction and proceeds to update the weight vector if necessary.

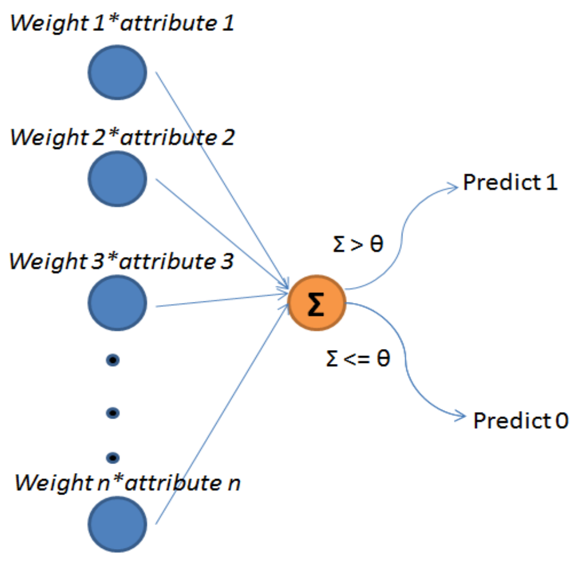

***Figure 6**: The Winnow1 Algorithm*

Specifically, a prediction is performed using this pattern:

$$if \sum_{i=1}^{m} w_i * x_i > \Theta \text{ ,} predict\ 1$$

$$if \sum_{i=1}^{m} w_i * x_i \leq \Theta \text{ ,} predict\ 0$$

where ϑ is the threshold, a constant. The algorithm then checks if the prediction was correct or not and acts accordingly. There are three different responses. If the prediction was correct, the algorithm does nothing. If the prediction was incorrect and the real value was 0, for every attribute that was 1, the corresponding weight is set to 0, or mathematically $\forall\, \boldsymbol{x_i} = 1 => \boldsymbol{w_i} = 0$. Finally, if the prediction was incorrect and the real value was 1, the corresponding weight is multiplied by *n* (called promotion step), or $\forall\, \boldsymbol{x_i} = 1 => \boldsymbol{w_i} = n * \boldsymbol{w_i}$.

For further information, please visit [1].

1.7 Gradient Descent

When solving an optimization problem, there are many possible solutions. Each solution has a certain error rate. Gradient descent is an optimization algorithm used extensively in machine learning for the purpose of minimizing a cost function$J(w)$, which represents a classification error, therefore reducing the error rate to the minimum possible.

Gradient descent assigns random initial weights ***w*** to the cost function. It then parses the whole dataset and calculates the cost (or loss) for using those weights. After that, it uses the negative derivative (or slope) to redistribute the weights. In order to control the rate of change regarding the weights, gradient descent also utilizes a constant *n* called learning rate.

Mathematically, gradient descent is described as:

$$w_j := w_j - n * \frac{\partial J(w)}{\partial w_j}$$

This process continues until a certain condition is met, for example a small error improvement from one iteration to the next. When using gradient descent, we also have to be careful when tuning the learning rate. If the learning rate is too small, the convergence will need many iterations over the dataset, therefore increasing the time needed. If it is large, we increase the chance of deviating from the minimum (overshooting).

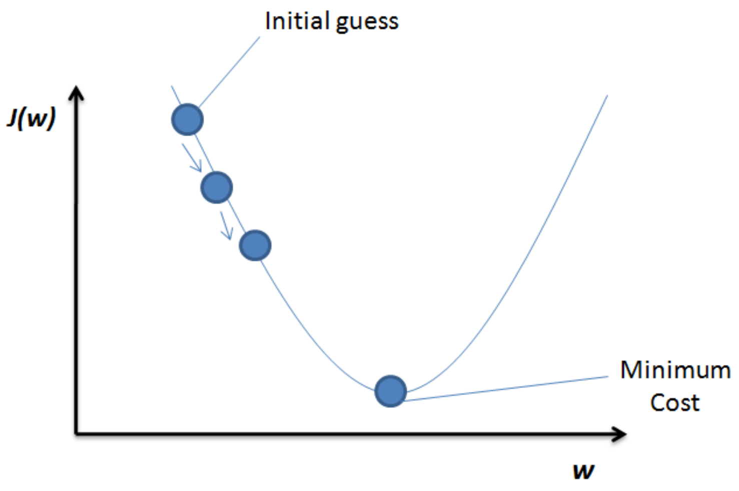

Figure 4*: Gradient descent*

Gradient Descent is mostly used for offline machine learning algorithms. For online learning algorithms, we will use *Stochastic* Gradient Descend.

1.8 Kernel Methods

Kernel SVMs are widely used and studied due to their phenomenal accuracy and ability to interpret non-linear results. One of the studies [25] focuses on automatically adapting the step size using the Stochastic Meta Descend algorithm, which increases the speed needed to train a predictive model by needing fewer iterations. In [19], the researchers propose a new kind of SVM that is scalable on limited resources, which they call LLSVM (Low-rank Linearized SVM), by transforming a non-linear SVM to a linear one via an empirical kernel map computed from efficient kernel matrices. Finally, in [16], a state-of-the-art SVM is proposed called Divide and Conquer SVM (DC-SVM), which partitions the problem of training a model into subproblems by clustering the data, and achieving excellent accuracy and speed between seven and one hundred times faster than the LIBSVM library.

Kernel Perceptron is an infamous online learning algorithm that has been used and studied for academic and business purposes due to its state-of-the-art performance. In [18], the researchers come up with the Projectron, which instead of discarding previous online hypotheses, it projects them onto a new space, which proves to improve the algorithm's performance. In [23], emphasis is given in reducing the memory usage, which is done by restricting the number of examples the algorithm stores, and therefore "forgetting" the rest, while maintaining a relative mistake bound. Finally, in [30] the Tighter Perceptron is introduced, which supresses the number of misclassified training points keeping only those who help increase the accuracy, therefore drastically increasing the model's performance.

When using the linear methods, we make the assumption that the dataset is linearly separable. However, for most of the real world datasets this is not true. In general, kernel methods solve this problem by taking the data from the original space (or *input space*) and projecting it to higher dimensions (or *feature space*) until the data is linearly separable.

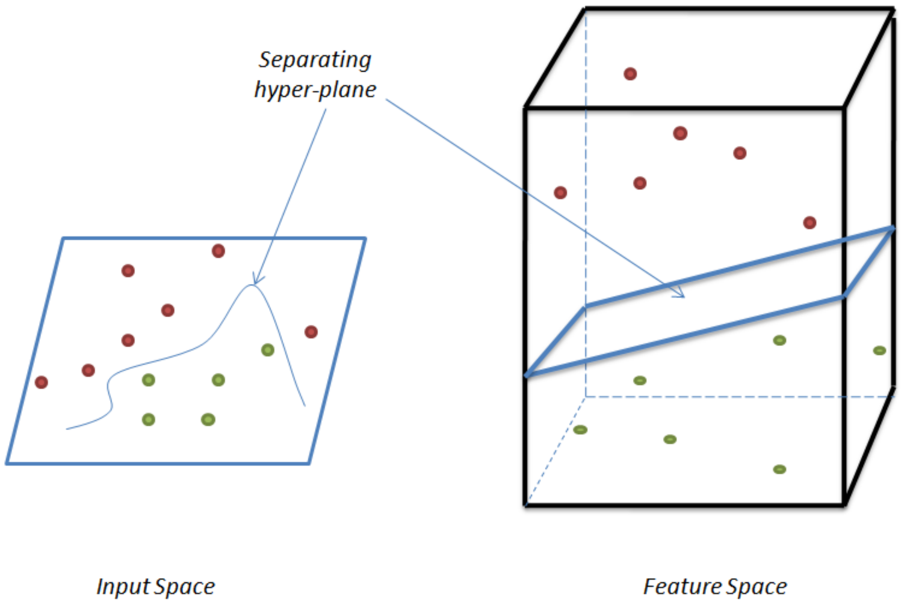

***Figure 7**: Kernel method projecting a 2D dataset in 3D*

Mathematically, a kernel function has the following form:

$$k(x, x') = \varphi^T(x) * \varphi(x') = \langle \varphi(x), \varphi(x') \rangle$$

which is the inner product of the vectors *φ(x)*and *φ(x')* in the feature space. There are many different types of kernel methods, as presented below:

Radial Basis Function (RBF): $k(x, y) = \exp(-\frac{|x-y|^2}{2\sigma^2})$

Sigmoid: $k(x, y) = \tanh(a * x^T + \theta)$

Polynomial:$k(x, y) = [x^T y + c]^d$

with each method having its pros and cons. In specific, the most common type of pros and cons is the tradeoff between accuracy of prediction and explanation of results. The more you increase accuracy, the less likely it is one is able to interpret the results, like which parameter is important, and vice versa. In our research, we will use the radial basis function kernel, combining it with the Perceptron and the Support Vector Machine. For more information on the topic, please consult [35].

2 Online Machine Learning

In this section we discuss online learning and other specifics used in our research. We present the theory behind the algorithms used and how we evaluated the models we constructed.

2.1 Online Machine Learning Overview

In machine learning, or *offline* machine learning, in order to create a model which predicts we view a dataset as a batch, take all the training examples to train the network and act accordingly. In online machine learning, or online learning for short, the data becomes available to the model in a sequential order.

Specifically, an online learning algorithm operates as follows. First, it initializes a weight vector ***w*** (with zeros or random numbers between negative one and one). Then, in time *t* (which represents the natural number of the training example), a new feature vector arrives which the algorithm uses along with the weight vector and predicts the class of the training example. If necessary, which regularly means if the prediction was wrong, the weight vector is updated using its corresponding update rule. The algorithm then continues to the next example, until every training example has been parsed. This is the most common online machine learning procedure, but there are other similar procedures, like random parsing instead of serial.

The typical learning process for an online model is as follows. After stating what we want to predict and find the relevant data, we distinguish the attributes and labels, and pick an online learning algorithm. In the beginning, our model is "dumb" in the sense that it only knows the first (of many) training example and has been trained with only this. Then, the model is given the second example and the algorithm adjusts in order to explain both data points. In this way, the model is eventually fed all the information. As time progresses and the more training examples are fed to the model, it keeps getting "smarter", so as to adapt for the purpose of explaining all the examples.

But why would someone choose to use online learning? There is a plethora of reasons. First and foremost, in online learning the algorithms have *adaptability,* meaning that if there is a change in the trend, the algorithm gradually understands it and proceeds with that "in mind". After that, there are many algorithms that *bound the number of possible mistakes* an online model can make, assuring that there is quality in the type of predictions that the model makes. Finally, in online learning, it is much *easier and cheaper* to deal with large quantities of data. In offline learning, one would typically need to train a model doing hundreds or even thousands of iterations over the same dataset, whereas in online learning, the data is only parsed once. For all these reasons, online learning is considered a staple in machine learning theory.

2.2 Evaluation of Online Machine Learning Models

In offline machine learning, there are various metrics we use in order to determine the quality of a given model. Although, for offline models we deal with the data as a batch. In online machine learning we deal with, and care for, the *progression* of the data. So the conventional offline metrics cannot be used for online machine learning.

In online machine learning, we assess the value of a model using *regret,* and measuring the cumulative loss over a sequence of examples. We assume that the data were generated by an unknown, yet fixed, hypothesis *h* such that $y_t = h(x_t)$ for $t \epsilon T$ and the entire sequence has no loss. Should this be the case, it is preferred that the loss of our model be independent of *T*. But in reality, there is no such hypothesis, so rather than searching for the ideal hypothesis, we calculate the hypothesis that has the least cumulative loss of *all* hypotheses. Therefore, for every hypothesis $h \epsilon H$ we define as regret, the excess loss for not consistently predicting with the hypothesis ,

$$R(h,T) = \sum_{t=1}^{T} l(h_t, (x_t, y_t)) - \sum_{t=1}^{T} l(h, (x_t, y_t))$$

or in other words, for every hypothesis we calculate the loss of the current hypothesis compared to the "perfect" hypothesis.

In our research, we define as the "perfect" hypothesis two different types of hypotheses. The first one, is an offline model created with the use of Python's *sklearn* library, which we arbitrarily call *relative regret*. For the second one, we assume an actual perfect hypothesis with zero errors over *T.*

A typical and desirable regret plot has increased error values for small *T* and as time progresses the error rate, $\frac{R(h,T)}{T}$, drops drastically approaching zero, simulating the transition from a bad learner to a good one. An example of a regret plot is as shown below at *Figure 8.*

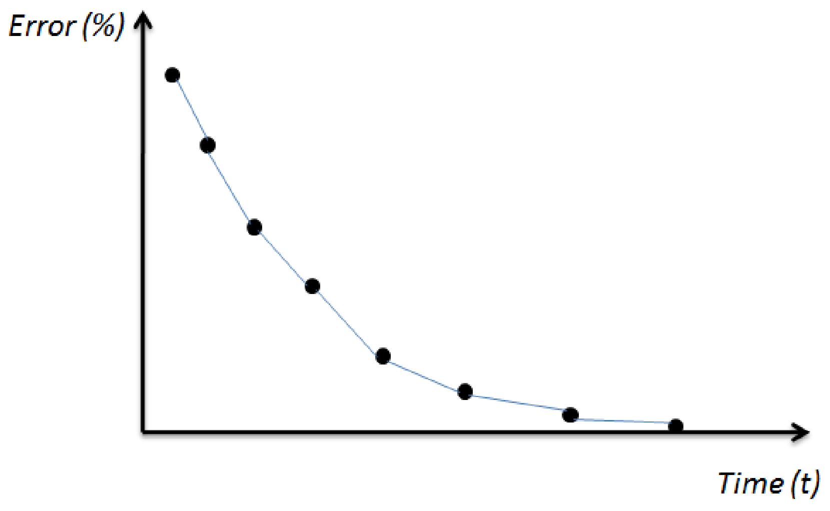

Figure 8: *Regret Plot*

For more information on regret, we recommend [17].

2.3 Online Perceptron

This method maintains a weight vector $\boldsymbol{w}$. Each iteration t, it receives a new data point with attributes $\boldsymbol{x_t}$, and using the $\boldsymbol{w_t}$predicts the class of the new data point, using the $\boldsymbol{sign}(\boldsymbol{w_t} * \boldsymbol{x_t})$ function. In case that the classifier missclassified the data point, $\boldsymbol{w}$ is updated by $\boldsymbol{w_t} = \boldsymbol{w_t} + n * y_{t*}\boldsymbol{x}_t$, where n is the learning rate (*n*>0), and y_t the real label of the data point. If the prediction was correct, the weights stay the same.

```
• create a weight vector w with initial values equal zero
• set learning rate n
• for every new incoming feature set x at time t:
    ➢ $y_{predicted} = sign\ (\boldsymbol{w} * \boldsymbol{x})$
    ➢ get $y_{real}$
    ➢ if $y_{real}\ != \ y_{predicted}$
        o update weight vector by:

          $\boldsymbol{w_j} \ := \ \boldsymbol{w_j} + n * \boldsymbol{x_j} * y_{real}$
```

Chart 2: *Online Perceptron pseudocode*

The algorithm is given a set of *m* examples like so:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), .., (x^{(m)}, y^{(m)})\}$$

and sees the examples one by one. In particular, it first sees $x^{(1)}$, and we ask it to predict the class of $y^{(1)}$**.** The perceptron makes its prediction and the true label is revealed and it uses this information to induce learning by readjusting the weights if need be. Then, we give the algorithm the second example $(x^{(2)})$, we ask it to predict $y^{(2)}$, we show the true label, and again it performs learning. Likeso, we feed it every training example [36].

2.4 Online SVM

As in offline SVMs, online SVMs are also a linear method which uses a line (or generally a hyper-plane for higher dimensions) to separate the two classes of the data.

Online SVMs also keep a weight vector in memory, with initial values equal to either zero or some small random number $w_0 \epsilon$ [0,1]. Each iteration t, the SVM performs a prediction using the $\boldsymbol{sign}(\boldsymbol{w_t} * \boldsymbol{x_t})$function (where $\boldsymbol{x_t}$ the attributes of the new data point). Then, depending on the outcome of a certain equality (as explained below), the model induces learning as follows. If $y_t * (\boldsymbol{w_t} * \boldsymbol{x_t}) < 1$ (where y_t the real label of the new data point) , we update the weight vector by $n * (\boldsymbol{w_t} - C * y_t * \boldsymbol{x_t})$ **,** where *n* the learning rate and *C* the regularization parameter. In this case, we have misclassified the instance so the weights are updated by a larger amount. If $y_t * (\boldsymbol{w_t} * \boldsymbol{x_t}) > 1$, we update the weight vector by $w_t - n * w_t$. In this case we have correctly classified the instance but we want to bring the decision boundary closer to the correctly classified instance. If $y_t * (\boldsymbol{w_t} * \boldsymbol{x_t}) = 1$ the weights remain the same [36].

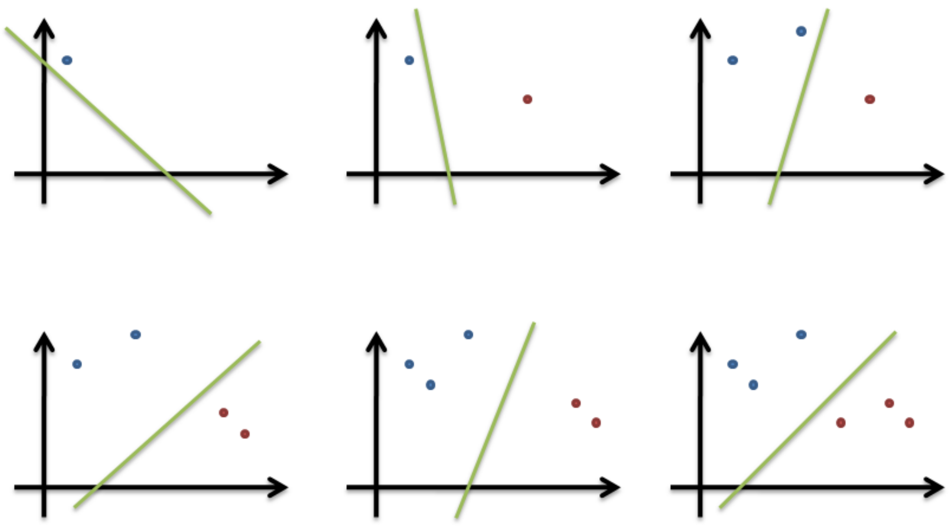

Figure 9: *Example of changing decision boundary*

```
• create a weight vector $\mathbf{w}$ with initial values equal zero
• set learning rate $\boldsymbol{n}$
• set $\boldsymbol{C}$ variable
• for every new incoming feature set $\boldsymbol{x}$ at time $\boldsymbol{t}$:
    ➢ $y_{predicted} = sign\ (\mathbf{w} * \boldsymbol{x})$
    ➢ get $y_{real}$
    ➢ set a check variable, where $check = y_{real} * (\mathbf{w} * \boldsymbol{x})$
    ➢ if $check < 1$:
        o update weight vector by:

            $\mathbf{w}_j := \mathbf{w}_j - n * \mathbf{w}_j + n * C * \boldsymbol{x}_j * y_{real}$

    ➢ if $check > 1$:
        o update weight vector by

            $\mathbf{w}_j := \mathbf{w}_j - n * \mathbf{w}_j$
```

Chart 1: *Online SVM pseudode*

2.5 Online Winnow

The winnow algorithm maintains a non-negative weight vector ***w,*** with initial values equal to the uniform weight vector with values summing to one. As an input, we also choose a learning rate *n.* At each iteration, we predict the label of the incoming data point using the $\boldsymbol{sign}(\boldsymbol{w}_t * \boldsymbol{x}_t)$decision function (where $\boldsymbol{x}_t$ is the attribute values of the new data point). If we misclassified the data point, we first calculate a normalization factor *Z,* where:

$$Z = \sum_{i=1}^{N} \boldsymbol{w}_t * \exp(n * y_t * \boldsymbol{x}_t)$$

and *N* the number of features. After obtaining zeta, we need to update weights using the rule:

$$\boldsymbol{w}_t := \frac{\boldsymbol{w}_t * \exp(n * y_t * \boldsymbol{x}_t)}{Z}$$

It is worth noting that since the initial weights are positive and the updating rule is multiplicative and positive, the weights are always positive. In this way, we manage to easily distinguish the *relevant* features as they will grow faster than in a Perceptron. Therefore, it is theoretically proven that Winnow makes fewer mistakes than the Perceptron if the number of relevant features is much smaller than the total number of features.

For more information, please consult [1].

```
• create a weight vector $\boldsymbol{w}$ with initial values equal zero
• set learning rate $\boldsymbol{n}$
• for every new incoming feature set $\boldsymbol{x}$ at time $\boldsymbol{t}$:
    ➢ $y_{predicted} = sign(\boldsymbol{w} * \boldsymbol{x})$
    ➢ get $y_{real}$
    ➢ calculate zeta, where
          $z := \sum_{i=1}^{m} \boldsymbol{w}_i * e^{n * y_{real} * \boldsymbol{x}_i}$
    ➢ if $y_{real} \, != y_{predicted}$
          o update weight vector by
               $\boldsymbol{w}_j := \frac{w_j * e^{n * y_{real} * x_j}}{zeta}$
```

Chart 3: *Online Winnow pseudocode*

2.6 Online Logistic Regression

Online logistic regression is an online learning algorithm that uses probabilities to determine if a data point should be positive or negative. It keeps in memory a weight vector ***w***and an attribute vector ***x.*** The values of the target label$y$ ϵ $\{0{,}1\}$, instead of {-1,1}.

Online logistic regression first creates a hypothesis,

$$h = \frac{1}{1 + \exp(\boldsymbol{w_t} * \boldsymbol{x_t})}$$

using the logistic function. It then predicts the class label of the incoming training example using the decision function:

$$y = 1, if\ h > \theta$$

$$y = 0, if\ h \leq \theta$$

where ϑ is the threshold, a constant (typically about 0.5). Then, the real value is revealed to the model, triggering an update to the weight vector if the prediction was incorrect, using the following update rule:

$$\boldsymbol{w_t} := \boldsymbol{w_t} - n * (\theta - y_t) * \boldsymbol{x_t}$$

```
• create a weight vector $\mathbf{w}$ with initial values equal zero
• set threshold
• set learning rate $\boldsymbol{n}$
• for every new incoming feature set $\boldsymbol{x}$ at time $\boldsymbol{t}$:
    ➢ $hypothesis = 1/(1 + e^{\mathbf{w} * \boldsymbol{x}})$
    ➢ if $hypothesis > threshold$, then $y_{predicted} = 1$
    ➢ else $y_{predicted} = 0$
    ➢ get $y_{real}$
    ➢ if $y_{predicted} \mathrel{!}= y_{real}$
        o update weight vector by:

          $\mathbf{w}_j := \mathbf{w}_j - n * (hypothesis - y_{real}) * \boldsymbol{x}_j$
```

Chart 4: *Online Logistic Regression pseudocode*

2.7 Incremental Machine Learning with Kernels

In online machine learning, when the algorithm sees a training example, it immediately discards it for the next (example). But in order to use kernel methods, and particularly dual methods, we need to have a set of support vectors which we will use in order to further train the algorithm. Therefore, we need to keep in memory a certain number of training examples. The solution to this problem is to use incremental learning, which remembers a specific amount of examples that it uses in order to train and predict. According to each application, the amount of examples kept in memory may differ, but the logic that we store some examples and their features remains.

2.7.1 Incremental Perceptron with Radial Basis Function

Kernel methods in incremental learning are also used to find a decision boundary that is non-linear. The kernel Perceptron algorithm in this case maintains a vector $\boldsymbol{\alpha}$ rather than the weight vector, also known as the *Lagrange variable*. This vector represents the coefficients assigned at each data point $\boldsymbol{x}_t$, $t \in [1, T]$. The initial values of this vector are typically zero. For the purpose of predicting the label of our training example, we use the $sign(\boldsymbol{w} * \boldsymbol{x}_t)$ function as well, but in this case:

```
• create a Lagrange variable a with initial values equal zero
• set gamma variable g
• for every new incoming feature set x at time t:
    ➢ calculate the sum:
          $\sum_{t=1}^{T} \boldsymbol{a_t} * y_t * e^{-(\boldsymbol{x_t} * \boldsymbol{x_s})^2 * g}$
    ➢ $y_{predicted} = sign(sum)$
    ➢ get $y_{real}$
    ➢ if $y_{predicted}$ != y_real
        ○ $a_t := a_t + 1$
```

Chart 5: *Incremental RBF-Perceptron pseudocode*

$$\boldsymbol{w} = \sum_{t=1}^{T} a_t * y_t * K(\boldsymbol{x_t}, \boldsymbol{x_s}) = \sum_{t=1}^{T} a_t * y_t * \exp(-\frac{(\boldsymbol{x_t} * \boldsymbol{x_s})^2}{\boldsymbol{2\sigma^2}})$$

Each time the model makes an incorrect prediction, a_t is incremented by one for the particular data point and $\boldsymbol{x_s}$represents (the attributes of) those data points that were previously misclassified. For example, if we are about to predict the 10th data point and example 3 and 7 were misclassified, those examples are going to have an *a* equal to one, and we will use their attributes in order to predict the 10th data point.

An update for x_t, is almost identical to augmenting the weight vector ***w*** with $y_t * x_t$, showing that this algorithm matches the standard Perceptron algorithm.

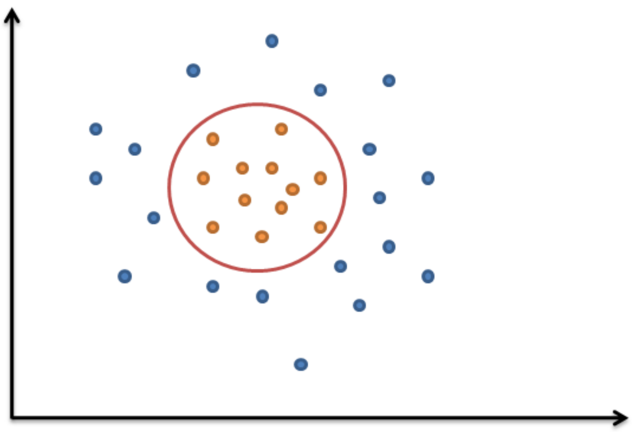

Figure 10: *Example of non-linear boundary*

2.7.2 Incremental Support Vector Machine with Radial Basis Function

The attribute vector ***x*** are mapped into higher dimensions to find a non-linear decision boundary as $\varphi(x)$, with *C*>0 being the penalty parameter, and solve the following dual problem:

$$\min_{a} \ F(a) \ = \frac{1}{2} a^T Q a - e^T a$$

$$subject\ to\ \ 0 \leq a_i \leq C, i = 1, \ldots, l,$$

$$y^T a = 0$$

where *e* is a vector of ones, and *Q* is a positive semi-definite matrix. The (i, j) $-$th element of *Q* equals $Q_{ij} = y_i * y_j * K(x_i x_j)$ and $K(x_i, x_j) = \exp(\frac{-||x_i - x_j||^2}{2\sigma^2})$since we chose the RBF kernel. Finally,

$$w = \sum_{i=1}^{l} a_i y_i \varphi(x_i)$$

and,

$$sign\left(\boldsymbol{w^T} \varphi(x)\right) = sign(\sum_{i=1}^{l} a_i y_i \exp(\frac{-\left|\left|\boldsymbol{x_i} - \boldsymbol{x_j}\right|\right|^2}{2\sigma^2}))$$

is the decision function used to classify an incoming data point. For more information on the topic, we recommend [5].

```
• create a Lagrange variable $\boldsymbol{a}$ with initial values equal zero
• set gamma variable $\boldsymbol{g}$
• set C variable
• for every new incoming feature set $\boldsymbol{x}$ at time $\boldsymbol{t}$:
    ➢ calculate the sum:

      $\sum_{t=1}^{T} \boldsymbol{a_t} * y_t * e^{-(\boldsymbol{x_t} * \boldsymbol{x_s})^2 * g}$

    ➢ y_predicted = sign(C * sum / t)
    ➢ set check variable, where check = $(y_{real} * sum * C)/t$
    ➢ if check < 1
        ○ $a_t := a_t + 1$
```

Chart 6*: Incremental RBF-SVM pseudocode*

3 Experimental evaluation

In this section, we present the methodology and results of our research. We evaluate our models . First, we present the regret for linear methods, then regret for kernel methods, and after that we present the relative regret for linear methods and kernel methods. The results were satisfying for the most part, with the exception of winnow.

3.1 Datasets

In this section we discuss the datasets used for this research. The datasets were chosen due to their size (many data points), they have two classes, and because of their importance on real world applications. Finally, the source for these datasets is the UCI open source machine learning repository.

3.1.1 Spam Mail

The objective for this dataset is to predict whether the mails are spam or not. This is a usually studied subject since billions of e-mails are transferred daily for several reasons such as prevent schemes and unwanted advertisements. The target value for this class is "0" or "1", referring to whether the mail is spam or not. There are in total 57 attributes concerning the word frequency, the number of capital letters, the longest sequence of uninterrupted capital letters and other, and 4600 instances. For more information on this dataset, please visit [4].

3.1.2 Online retail

Our objective for this dataset is to predict whether a specific order will be cancelled. Online shops are increasing by the day. For a company to maximize its profits one, it must minimize the loss, and predicting order cancellations plays a big part in such a process. This dataset contains 541099 instances and 7 attributes concerning the time of placement of order, the country, customer identification, the type and quantity of product and other. This dataset is a time-series, ordered by the time of transaction, and since we use the nominal feature of date as a feature in our dataset, it means that we can't shuffle this dataset because it will affect the model's correlation with the original ordering. Also, for this dataset there are only 6000 instances of cancellation of order and the rest are not. This does not necessarily pose a problem, because we can predict that the order will be cancelled (even if it doesn't), and act accordingly, for example increase the shipping cost or apply a no-refund policy. For more information on this dataset, please visit [3].

3.1.3 Epileptic Seizure

With the progress of technology there has been an uprising in protection and prevention of spiraling negative incidents for susceptible social groups. In this dataset there are over 11500 instances, and it has 178 attributes. The attributes keep track of several medical data. This data is a succession of several time-series (each of which is for a person), ordered by time, for several individuals and depicts the general behavior of a patient for nearly 23 seconds. Our goal is to predict if a person is currently having an epileptic seizure. For more information on this dataset, please visit [2].

3.2 Dataset Handling

In machine learning, our data is almost never in the desired form. A necessary step before dealing with machine learning algorithms, is to shape the data according to the needs of each algorithm. In our case, we use various linear algorithms, so our main objective is to regularize the data in the range {-1,1}. Technically, this is not possible because we do not have the whole dataset, but we can use it if we know the upper and lower boundaries of our features. In this section we discuss the changes made in each dataset to achieve this.

For *Spam* and *Epileptic Seizure* datasets, the methodology was relatively easy. After reading the data, we separate the attributes and label in two different variables, and scale each attribute to the range {-1,1}. As for the labels, in the *spam* dataset there are only two values "0" and "1", so we make it "-1" and "1". For the *epileptic seizure* there are five classes, but only one of them is described as "having a seizure" and the rest are not, so we set the value of having a seizure as "1" and not having a seizure as "-1" (which is a common treatment for this dataset according to its description [2]).

For *Online Retail* dataset, there were various non-numerical values. In the dataset description, it mentions that cancelled orders are mentioned with a 'C' before the number of items bought (e.g. C32 means a person ordered 32 items and cancelled the order). Therefore, we isolate the 'InvoiceNo' column, and if the string begins with a "C" we mark it as -1 and if not, we mark it as 1. As for the attributes, we take all the numerical and leave the nominal, and then scale it to {-1,1}.

3.3 Experimental Methodology

For this research, the experimental process that took place was the same for all ML methods. Even though six different online machine learning methods were implemented, they all had a similar structure for efficiency and consistency purposes. First of all, we normalized the data, which cannot technically be used in online learning, but is possible if we know the upper and lower bounds of our features. Next, we shuffled the datasets 10 times to remove any bias based on order, but we didn't shuffle for the online retail dataset, because it was ordered by time which was used as a feature. After that, we had a section in which we tuned the hyperparameters of each method (for example in SVM, we had the hyperparameters ***C*** and ***n***). Next, we created a variable that would save the predictions our model would make. Now, according to the ML method used, we had a section implementing each method and saving the predictions of the model. An important side note is that the dataset was accessed *just once* and in a *serial* way (starting from the first data point all the way to the last). Finally, we split each dataset in 10 chunks (where a chunk equals 1/10 of the length of the dataset) and counted the errors in each chunk (for example if a dataset had one thousand data points, we split it into ten one-hundred chunks) to help us measure both regret and relative regret. In order to measure relative regret we created another model using the *sklearn* library from python and would save the errors of this model on the same chunks, and compare it to our model's errors. Noted, the procedure of chunking might not be the most "correct" (because the model changes with each data point and, technically, should be treated as such), but it is the most computationally efficient since our resources are finite and it does capture the progression of the model. In order to measure relative regret, we trained an offline model using a 50/50 split for training/testing, and we chose the 50/50 split because we wanted to have the least amount of training examples given as training and the maximum amount of examples to predict.

3.4 Parameterization of ML Methods

On Table 1 we provide the parameters used for each method on each dataset. Several tests took place in order to maximize the efficiency of each method. According to the parameter many ranges and different combinations of parameters were used (for example in SVM which has more than one parameters) in order to achieve a desirable regret with downwards trend and low overall error rate.

Dataset	Method	Parameters
Epileptic Seizure	Perceptron	n=0.05

	SVM	C=20
		n=0.05

	Logistic Regression	n=0.5
		threshold=0.5

	Winnow	-
	RBF-Perceptron	gamma=10^{18}

	RBF-SVM	C=1
		gamma=$2*10^{-5}$

Spam	Perceptron	n=1

	SVM	C=1000
		n=0.0002

	Logistic Regression	n=0.5
		threshold=0.5

	Winnow	n=300
	RBF-Perceptron	gamma=10^{5}

	RBF-SVM	C=10
		gamma=10^{8}

Online retail	Perceptron	n=0.05

	SVM	C=20
		n=0.05

	Logistic Regression	n=0.2
		threshold=0.5

	Winnow	-
	RBF-Perceptron	gamma=$2 * 10^{-5}$

	RBF-SVM	C=10^{7}
		gamma=$2 * 10^{-5}$

Table 1: *Parameter Tuning*

3.5 Comparative Results

In this section we compare the results for each method. We explore the performance of each method on each dataset and the overall performance, using both regret and relative regret as metrics.

Dataset	Method	Error
Epileptic Seizure	Logistic Regression	0.1784
	Perceptron	0.1823
	SVM	0.1757
	Winnow	-
	RBF-Perceptron	0.1812
	RBF-SVM	0.1694
Spam	Logistic Regression	0.1204
	Perceptron	0.1586
	SVM	0.1539
	Winnow	0.1586
	RBF-Perceptron	0.3244
	RBF-SVM	0.3850

***Table 2**: Offline error*

Dataset	Method	Ending regret
Epileptic Seizure	Logistic Regression	0.2001
	Perceptron	0.228
	SVM	0.1966
	Winnow	-
	RBF-Perceptron	0.2
	RBF-SVM	0.1802
Spam	Logistic Regression	0.1937
	Perceptron	0.1993
	SVM	0.1848
	Winnow	0.2848
	RBF-Perceptron	0.3609
	RBF-SVM	0.4098
Online retail	Logistic Regression	0.0219
	Perceptron	0.0428
	SVM	0.0329
	Winnow	-
	RBF-Perceptron	0.0428
	RBF-SVM	0.0176

***Table 3**: Ending Regret*

On Tables 2 and 3 we also present information about the accuracy of the offline models and the ending regret of our models. On an offline model, we train it doing a 50/50 split for training/testing, and use that model to predict all the training examples, count the errors the same way we counted the online errors, and get the relative regret by subtracting the offline from online errors.

3.6 Comparative Results - Regret

In this section we compare the results for each method and each dataset, using the regret metric.

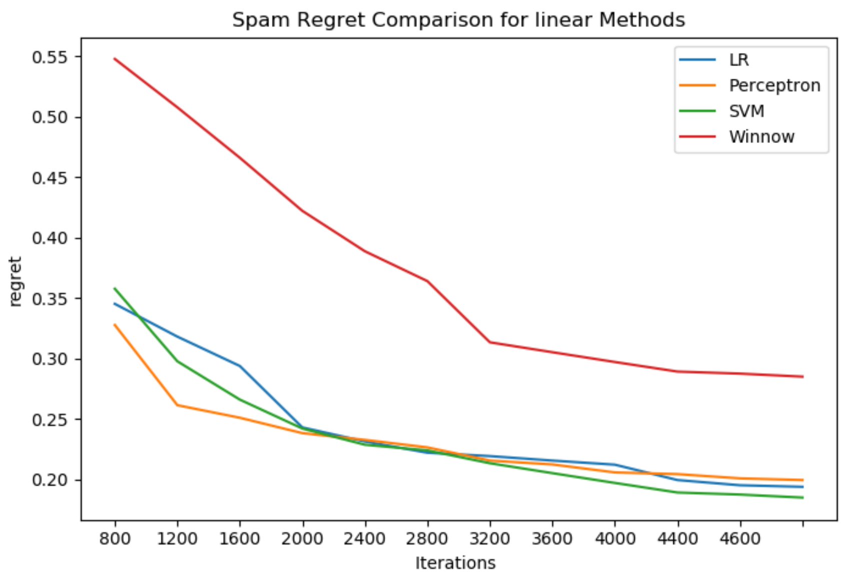

Figure 1: *Regret comparison for Spam linear methods*

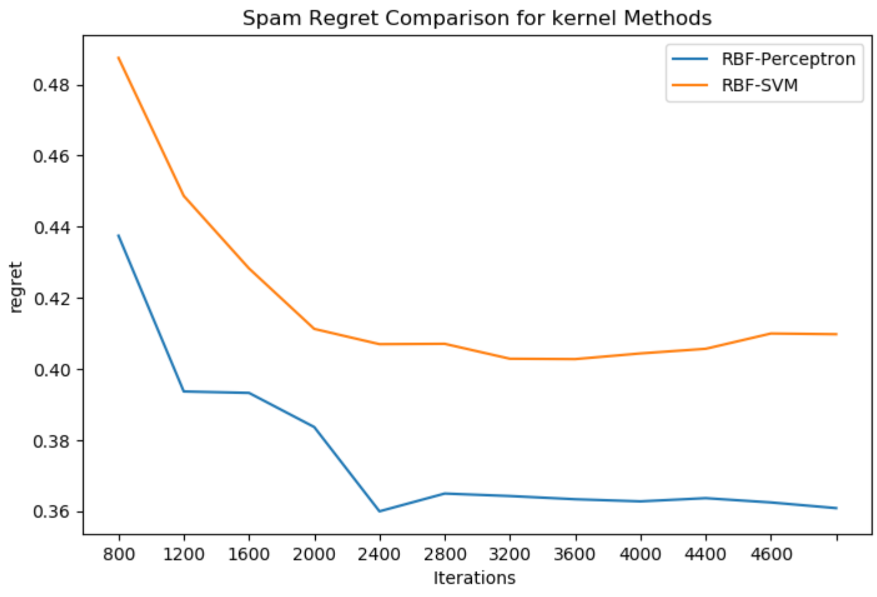

Figure 2: *Regret comparison for Spam kernel methods*

For the spam dataset, we see that all methods act as expected, by having a good downwards trend. SVM, Logistic Regression and Perceptron had the best performance with the lowest error rate and Winnow had a very good decrease in error. The kernel methods (Figure 2) did not perform quite well, due to the lack of many training examples, which is necessary when dealing with higher dimensions.

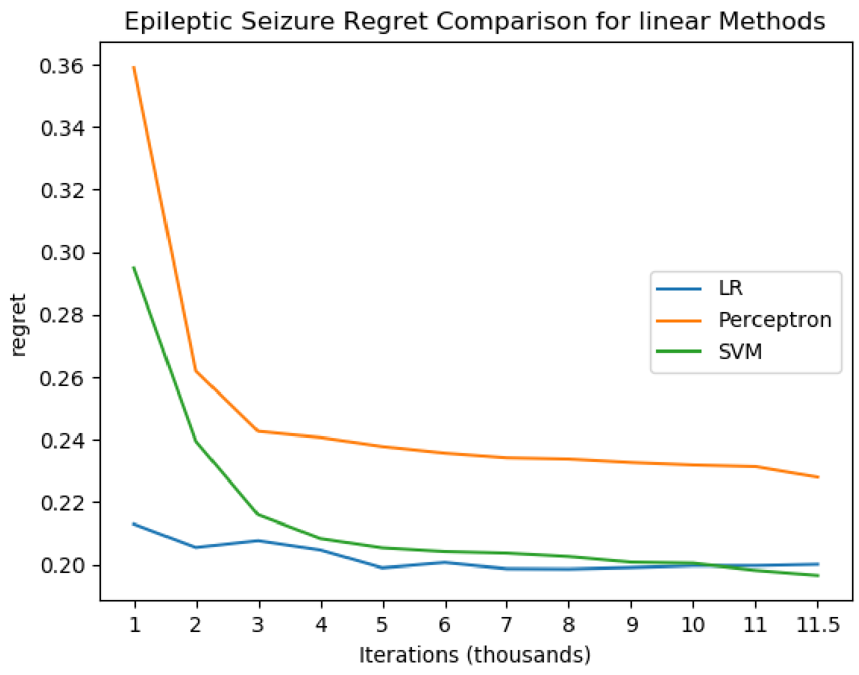

***Figure 3**: Regret comparison for Epileptic Seizure linear methods*

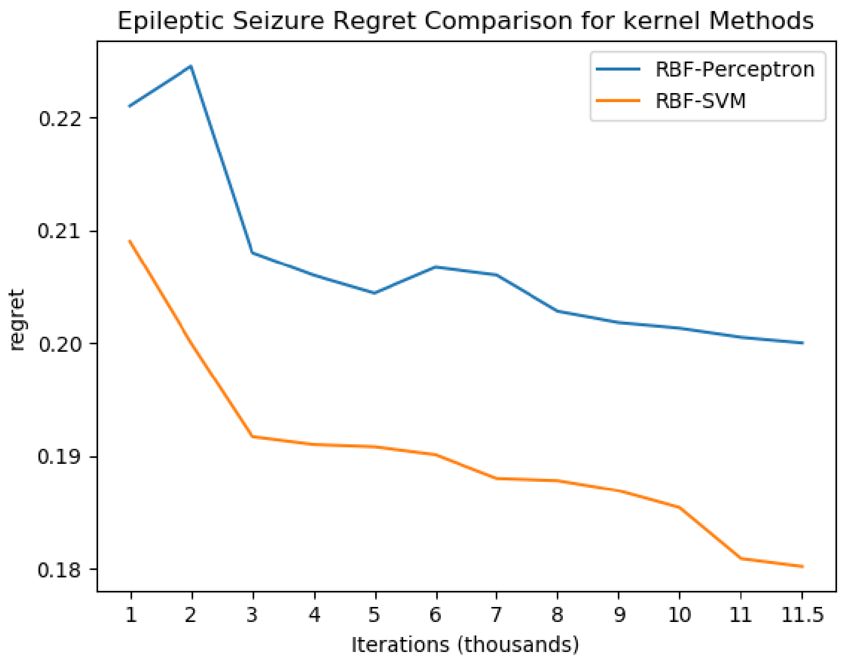

***Figure 4**: Regret comparison for Epileptic Seizure kernel methods*

Regret for Epileptic Seizure dataset, based on Figures 3 and 4, was relatively good, since all methods (except Winnow which is once again not depicted because of its high error rate) had a downwards trend. The best performance was from kernel SVM even if it did not have a good downwards trend.

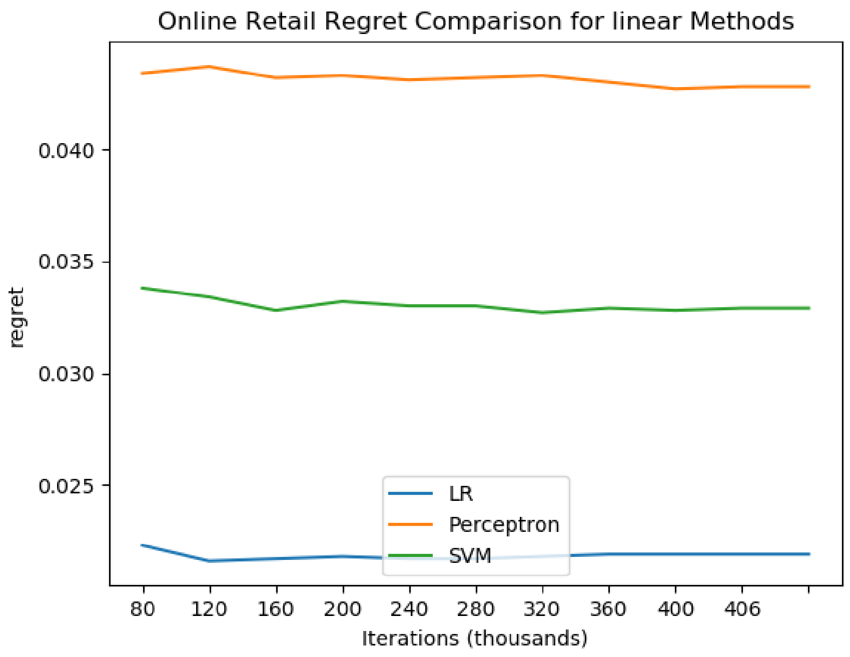

***Figure 5**: Regret comparison for Online Retail linear methods*

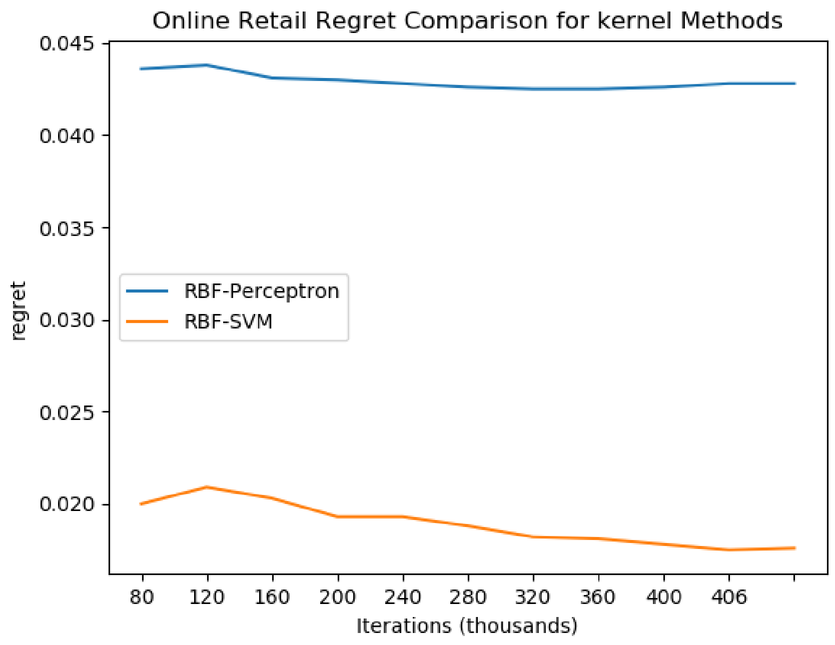

***Figure 6**: Regret comparison for Online Retail kernel methods*

The online retail dataset methods had low overall error rates but no downwards trend (Figures 5 and 6). This leads us to the conclusion that learning was not induced, other than in the beginning (since there was no downwards trend). One reason that learning was not induced is because the class had several zeros (the order will not be cancelled) and very few ones (the order will be cancelled), so the models learn to prioritize the zeros. This, however, is a minor problem, since with our models we can gain useful information by predicting that an order might be cancelled (even if in the end it doesn't) and act accordingly, for example enforce a no-refund policy. Lastly, in this figure Winnow was not depicted due to its very high regret of over fifty percent.

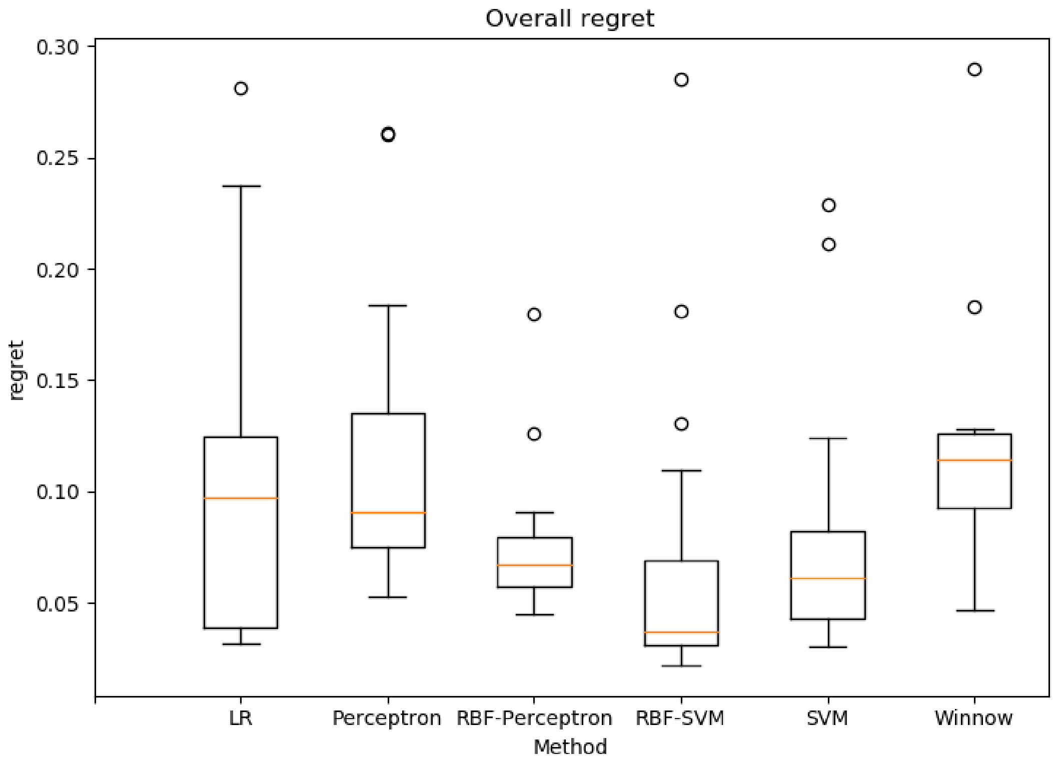

***Figure 7**: Overall regret across all methods/datasets*

From *Figure 7* we understand that SVM and Logistic Regression had the best performance throughout all the datasets, with small median and overall values. The kernel methods did not perform as well due to the lack of quantity of training examples. Winnow had the worst overall performance with extremely high regret values. Finally, the boxplots were created by collecting the values of all methods across all datasets and representing them on this graph (Figure 7).

3.7 Comprative results - Relative Regret

Here we present our results concerning relative regret on each dataset for each method.

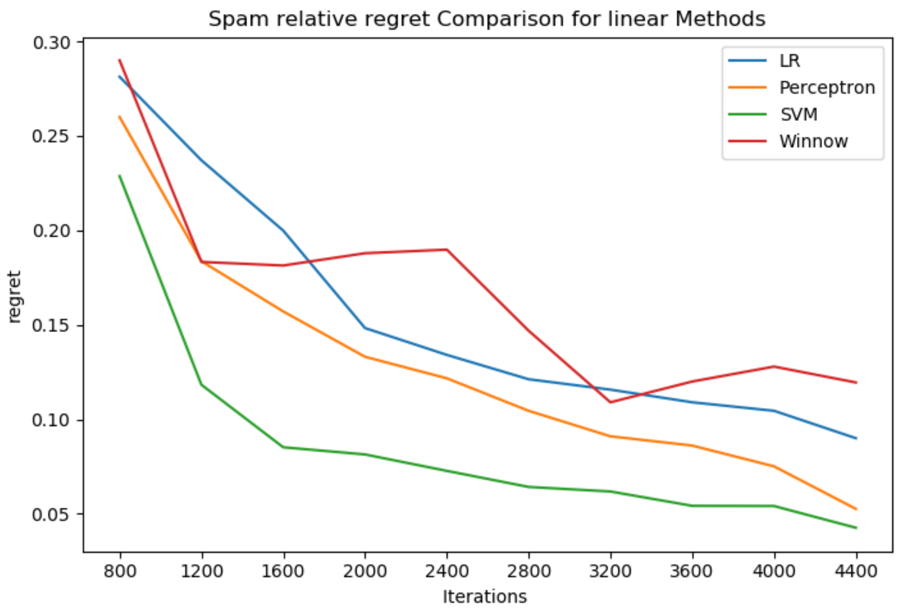

Figure 8*: Relative Regret Comparison for Spam linear methods*

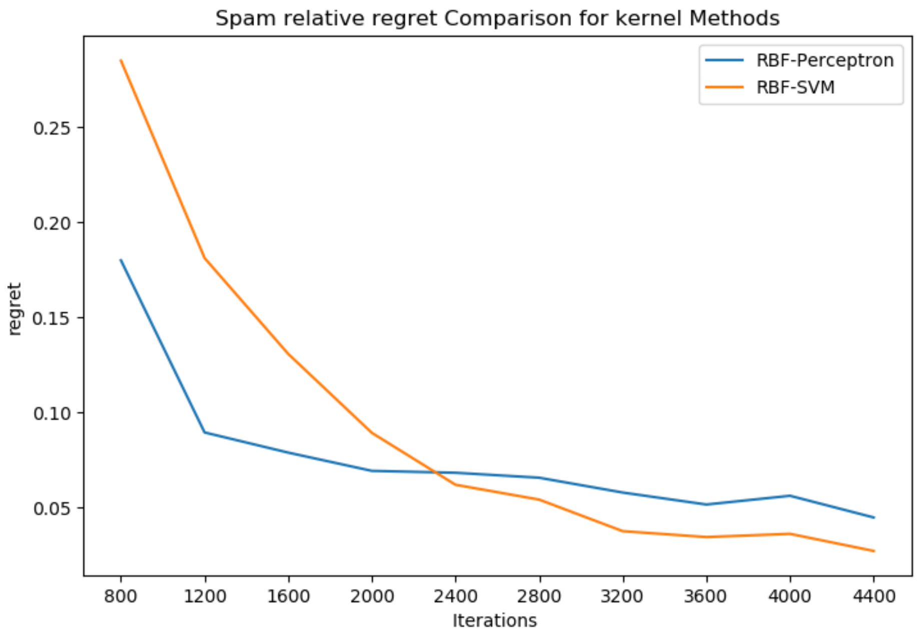

Figure 9*: Relative Regret Comparison for Spam linear methods*

Relative regret for the spam dataset (figures 8 and 9) was optimal. All methods, linear and kernel, converged towards their offline model counterpart with a very good rate. Winnow seemed to have some fluctuations but overall the trend is obviously downwards.

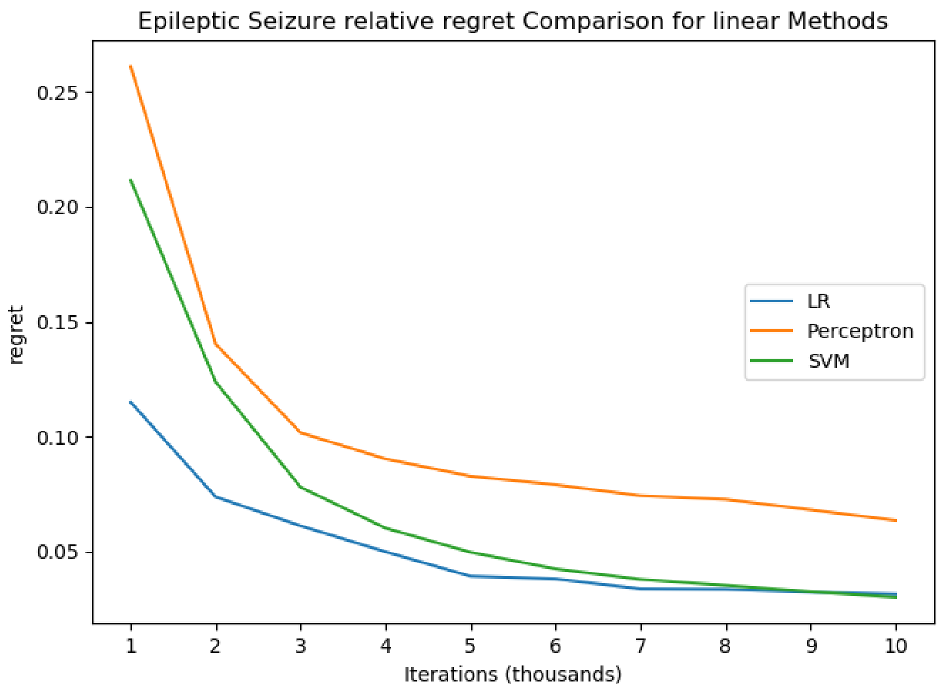

***Figure 10**: Relative regret Comparison for Epileptic Seizure linear methods*

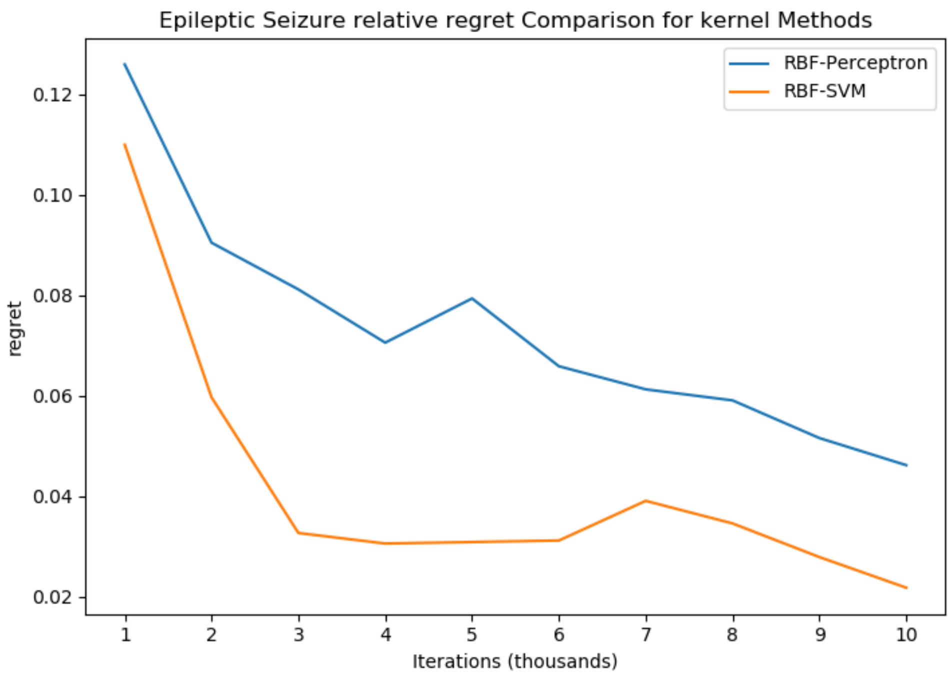

***Figure 11**: Relative regret Comparison for Epileptic Seizure kernel methods*

An overall downwards trend for relative regret (Figures 10 and 11) for the Epileptic Seizure dataset shows that most of our models were trained well. All methods, kernel and linear, seem to be converging towards the offline model.

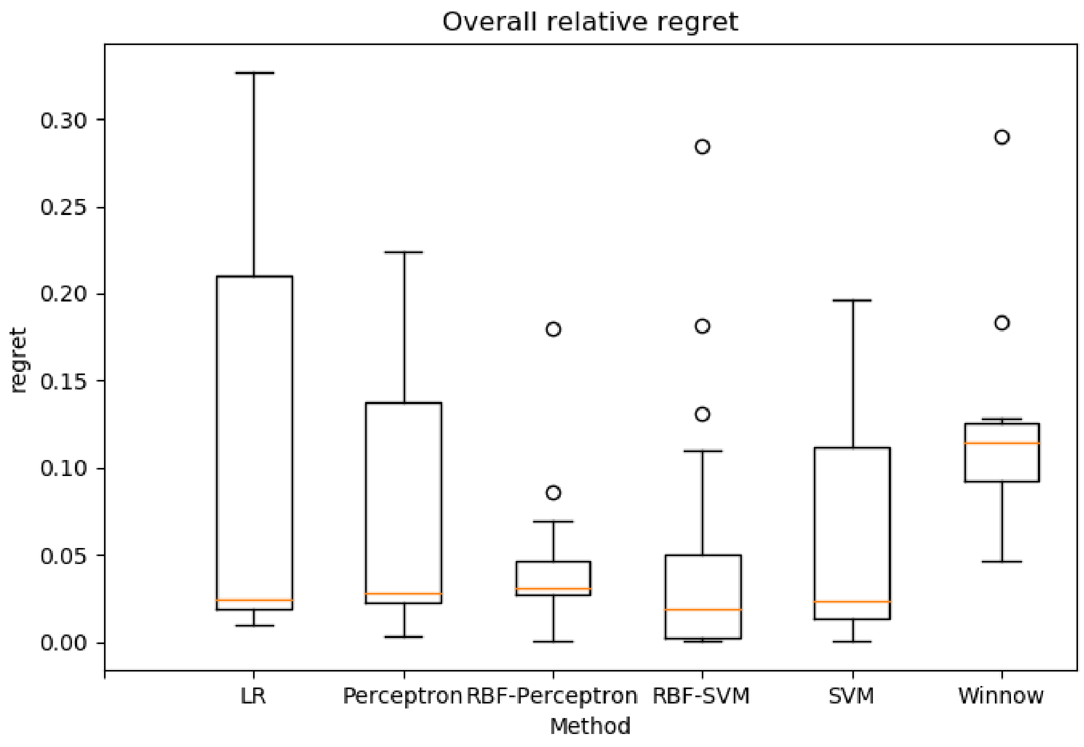

***Figure 12**: Overall relative regret across all methods/datasets*

Figure 12 shows that all methods were close their offline counterparts. The kernel methods and winnow had some excessive outlier values. Overall, the kernel methods seem to have the performance closest to assembling an offline model.

**: Figures 7 and 12 use a boxplot, which represents the distribution of quantitative data. Inside the box there are 50% of the values, the orange line represents the median, outside the box is 49.3% of values and 0.7% is outliers (dots above box).*

4 Conclusion

In this research, we managed to provide solutions to real-world problems using linear and non-linear online machine learning algorithms, as well as provide relatively good results.. By comparing the performance of the methods throughout all datasets, it became clear that kernel methods do not necessarily perform better than linear ones, if there is a lack of volume on the training examples, which therefore means that the volume of training examples is an important factor to implementing and increasing the quality of kernel methods for incremental learing. In contrast, SVM and logistic regression had the best performance from the linear methods, with a small median error rate, but they also had some higher error rates throughout. Perceptron had a relatively higher median error rate, and Winnow was disappointing throughout, with extremely high error rates and general results worse than even a coin flip (except for spam dataset), due to Winnow's lack of hyperparameters and inability of negative correlation. Furthermore, we did not measure the relative regret of online retail dataset because of the lack of balance in negative and positives examples (and therefore high accuracy), which would not give us enough information with regards to our models' quality. Finally, it is worth noticing that during the initial tests the algorithms were programmed using various for loops, which when changed to vector multiplication/manipulation, the training time became one to five thousand times quicker.

For further extension of our research, we propose the comparison of these methods with meta-algorithms like Weighted Majority and Exponential Weighted Average, algorithms which train several unintelligent models, and use a voting system to decide which of these models to trust during each iteration. Also, a useful metric as an indicator for model performance is training time, which was not measured in this research. Finally, with the exception of the Online Retail dataset which had hundreds of thousands of entries, our datasets were relatively small, and in a way hindering the evolution or development of our models. Therefore, larger datasets would account for better and more meaningful performance.

5 Bibliography

[1] N Littlestone (1988), Learning Quickly when irrelevant Attributes Abound: A New-Linear-Threshold Algorithm, Journal of Machine Learning Research 4 (2), 285-318

[2]Repository, U. M. *Epileptic Seizure recognition Data Set*. Retrieved from https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition (2018)

[3] Repository, U. M. *Online Retail Data Set*. Retrieved from https://archive.ics.uci.edu/ml/datasets/Online+Retail?fbclid=IwAR0Kth0FchTGO08w_yRi6O02_YVheg-celQvdP--xrpsC1XBZVbirb9JAqs (2018)

[4] Repository, U. M. *Spambase Data Set*. Retrieved from https://archive.ics.uci.edu/ml/datasets/Spambase?fbclid=IwAR2QfISaliFLChbr_aC8HXNdsV0Ch3ShrZN4YK3u1gZiPgX8aITWyFdb5Gs (2018)

[5] S. Keerthi, C. Lin (2003), Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel, Centre for Studies on Inclusive Education 7 (15), 1667-1689

[6] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter (2011), Pegasos: Primal Estimated sub-Gradient Solver for SVM, Mathematical Programming 3 (127), 3-30

[7] D. Hush, C Scovel (2007), Stability of Unstable Learning Algorithms, Machine Learning 3 (167), 197-206

[8] Y. Wang, I. Witen (2002), Modeling for Optimal Probability Prediction, International Conference of Machine Learning, 650-657

[9] R. Webber (2013), The evolution of direct, data and digital marketing, Journal of Direct Data and Digital Marketing Practice 4 (14), 291-309

[10] A. Singh, G Rumantir, A. South, B. Bethwaite , Clustering Experiments on Big Transaction Data for Market Segmentation, Proceedings of the 2014 International Conference on Big Data Science and Computing, Article no. 16, 2014

[11] Z. You, Y. Si, D. Zhang, X. Zeng, S. Leung, T. Li (2015), A decision-making framework for precision marketing, Expert Systems with applications: An International Journal 7 (42), 3357-3367

[12] Andrzejak RG, Lehnertz K, Rieke C, Mormann F, David P, Elger CE (2001) Indications of nonlinear deterministic and finite dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state, Phys. Rev. E, 64, 061907

[13] Amey Kulkarni, Y. P. (2016). Adaptive Real-time Trojan Detection Framework through Machine Learning, Proceedings of the IEEE International Sympozium on Hardware Oriented Security and Trust (HOST), 120-123

[14] F. Bach (2014), Adaptivity of averaged stochastic gradient descent to local strong convexity for logistic regression, Journal of Machine Learning Research 1 (15), 595-627

[15] C. Chang, C. Lin (2011), LIBSVM: A Library for Support Vector Machines, ACM Transactions on Intelligent Systems and Technology 3 (2), Article no. 27

[16] Cho-Jui Hsieh, S. S. (2014). A Divide-and-Conquer Solver for Kernel Support Vector Machines, Proceedings of the International Conference on Machine Learning (ICML), 566-574

[17] M. Collins (2002), Ranking Algorithms for named-entity extraction: boosting the voted Perceptron, Proceedings on the 40th Annual Meeting on Association for Computational Linguistics, 486-496

[18] F. Orabona, J. Keshet, B. Caputo (2008), The Projectron: a bounded kernel-based Perceptron, Proceedings of the 25th Internation Conference of Machine Learning, 720-727

[19] K. Zhang, L. Lan, Z. Wan, F. Moerchen (2012), Scaling up Kernel SVM on Limited Resources: A Low-Rank Linearization Approach, Proceedings of the fifteenth International Conference on Artificial Intelligence and Statistics (22), 1425-1434

[20] L. Huang, S. Fayong, Y. Guo (2012), Structured Perceptron with inexact search, Proceedings of the 2012 Conference of the North American Chapter of the association for Computational Linguistics: Human Language technologies, 142-151

[21] M. Kim, Y. Song, S. Wang, Y. Xia, X. Jiang (2018), Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation, JMLR Med Inform 2 (6), e19

[22] N. Cesa-Bianchi, A. Conconi, C. Gentile (2005), A Second-Order Perceptron Algorithm, Society for Industrial and Applied Mathematics 3 (34), 640-668

[23] O. Dekel, S. Shwartz, Y. Singer (2005), The Forgetron: a Kernel Based Perceptron on a fixed Budget, Proceedings of the 18th International Conference on Neural Information Processing Systems, 259-266

[24] P. Laskov, C. Gehl, s Kruger, K. Muller (2006), Incremental Support Vector Learning: Analysis, Implementation and Applications, Journal of Machine Learning Research (7), 1909-1936

[25] S. Vishwanathan, N. Schraudolph, A. Smola (2006), Step size Adaptation in reproducing Kernel Hilbert Space, Journal of Machine Learning Research (7), 1107-1133

[26] Seyda Ertekin, L. B. (2008). Ignorance is Bliss: Non-Convex Online Support Vector Machines, IEEE Transactions on Pattern analysis and Machine Learning Intelligence 2 (33), 368-381

[27] Y. Lin, H. Lei, J. Wu, X. Li (2015), An Empirical Study on Sentiment classication of Chinese Documents Review using Word Embedding, 29th Pacific Asia Conference on Language, Information and Computation, 258-266

[28] T. Zhang, F. Damerau, D. Johnson (2002), Text Chunking based on a Generalization of Winnow, Journal of Machine Learning Research 2, 615-637

[29] T. McCormick (2012), Dynamic Logistic Regression and Dynamic Model averaging for Binary Classification, Biometrics, 23-30

[30] Z. Wang, S. Vucetic (2009), Tighter Perceptron with Improved Dual use of Cached Data for Model representation and Validation, International Joint conference on Neural Networks, 3297-3302

[31] Christos Dimitrakakis and SamyBengioy. Online Policy Adaptation for Ensemble Classifiers, Proceedings of the 12th EuropianSympozium on Artificial Neural Networks (ESANN) 2004

[32] M. Minsky, S. A. Papert (1987), Perceptrons: An Introduction to Computational Geometry, Expanded Edition, MIT Press

[33] I. Steinwart, A. Christmann (2008), Support Vector Machines (Information Science and Statistics), Springer

[34] D. G. Kleinbaum, M. Klein (2010), Logistic regression: A Self-learning Text (Statistics for Biology and Health), Springer, 3rd edition

[35] B. Schlkopf, A J. Smola (2001), Learning with kernels: Support Vector Machines, Regularization, Optimization and Beyond (Adaptive Computation and Machine Learning), MIT Press

[36] M. Mohri, A. Rostamizadeh, A. Talwalkar (2012), Foundations of Machine Learning, MIT Press