KTH Computer Science
and Communication

A Survey & Implementation of Financial Alarm Classification

Support Vector Machines

JENS WIRÉN
FARHAD KIMANOS

Batchelor's Thesis at CSC
Supervisor: Hedvig Kjellström

Abstract

The goal of this thesis is to find, implement and evaluate a suitable machine learning algorithm to classify and predict true and false alerts using labelled data. Alerts are triggered in the Scila Surveillance software when certain parameters are exceeded in a trade, such as a to big volume over a to small time-span.

Financial market operators are nowadays required by law to perform market surveillance and due to the huge amounts of data accumulated, machine learning techniques in general and supervised learning in particular comes as a natural choice.

This thesis starts with a survey of existing algorithms and their performance as well as related work. The technique of Support Vector Machines (SVM) is the most used and overall best performing algorithm, why it is chosen to be further tested. Next is a thorough derivation of the SVM classifier starting with convex optimisation theory and how SVM are mathematically constructed.

When implementing SVM both grid-search and cross-validation are utilized. The classifier is threaded as much as possible to allow parallelisation which drastically reduced computational time. The characteristics of a good classifier is not trivial and several accuracy-measures are implemented and tested showing that balanced accuracy and a combined analyses of positive and negative recall are the most useful.

The provided dataset is huge and a few specific alerts are chosen for the proof-of-concept implementation. These are in turn separated into subsets based on alert-specific subcategories. Several tests are then conducted using a lightly modified Java version of the open-source package libsvm.

Results show that it is easy to achieve either a high positive and low negative recall or vice versa but to find parameters where both are high is very difficult. For this thesis the choice of a moderately high recall is likely the most useful one.

SVM is definitely an interesting approach and perhaps other techniques such as neural networks or incorporating time-series evaluation might yield even better results but further investigations is needed.

Referat

En undersökning & implementation för klassificering av finansiella larm

Målet med detta examensarbete är att finna, implementera och utvärdera en lämplig maskininlärnings-algoritm för att klassificera och förutsäga sanna och falska larm med märkt data. Larm utlöses i Scila Surveillance programvaran när vissa parametrar överskrids i en handel, till exempel en för stor volym under en för liten tidsrymd.

Finansmarknadens aktörer är numera skyldiga enligt lag att utföra marknadsövervakning och på grund av de enorma mängder insamlade data kommer maskininlärning i allmänhet och övervakad inlärning i synnerhet som ett naturligt val.

Denna avhandling börjar med en kartläggning av befintliga algoritmer och deras prestanda samt tidigare studier. Support Vector Machines (SVM) är den mest använda och allmänt bäst presterande algoritmen, varför denna väljs att testas ytterligare. Sedan följer en grundlig härledning av SVM-klassificeraren, vilken börjar med konvex optimeringsteori och hur SVM är matematiskt konstruerade.

Vid genomförandet av SVM utnyttjas både rutnätssökning och korsvalidering. Klassificeraren är trådad så mycket som möjligt för att tillåta parallellisering som drastiskt sänker beräkningstiden. Vilka egenskaper som är bra hos en klassificerare är inte trivialt. Efter att flera noggrannhetsmått har implementerats och testats visar det sig att balanserad noggrannhet och en kombinerad analys av positiv och negativ sensitivitet är de mest användbara måtten.

Det tillhandahållna datasetet är enormt och några specifika larm väljs för en proof-of-concept implementation. Dessa är i sin tur uppdeladad i undergrupper baserade på larm-specifika underkategorier. Flera tester genomförs sedan med hjälp av en lätt modifierad Java version av open-source paketet libsvm.

Resultaten visar att det är lätt att uppnå antingen en hög positiv och låg negativ sensitivitet eller vice versa men att hitta parametrar där båda är höga är mycket svårt. När det gäller målet med denna avhandling är sannolikt valet av en måttligt hög sensitivitet den mest användbara.

SVM är definitivt en intressant metod och det är möjligt att andra algoritmer så som neurala nätverk eller tidsserieranalys kan ge ännu bättre resultat men ytterligare studier behövs.

Contents

Chapter 1

Preliminaries

1.1 Introduction

Machine Learning is a field in computer science that has been growing exponentially during the recent years. In general it can be defined as a branch of artificial intelligence focused on the construction and learning of computational systems. The average person associates this technique with companies like Google and Facebook where massive information databases are constantly searched for useful patterns and user behaviour analysed for marketing reasons. But the fact is that machine learning has shown to be extremely useful in a wide variety of different fields, e.g. chemin-formatics, DNA sequences classification, medical diagnosis, stock market analysis and credit card fraud detection.

1.1.1 Supervised Learning

Apart from massive data mining and pattern recognition where human capacity is simply insufficient machine learning can be used to drastically reduce human work-load and automate processes where a more dynamic decision making is required. The most apparent example of such implementations are email spam filters, that in general are constituted by decision making algorithms that are partly based on the history of decisions made by the user. The system tries to learn from and mimic the behaviour of the human user by applying a *learning model* designed for the specific purpose.

A complex process can be viewed as a mathematical function that maps input to output data, e.g. the human brain processing email content. Such a complex unknown function is referred to as the *target function* and can in general not be defined explicitly. An alternative strategy, that is rather intuitive, is to use known examples of the function behaviour to try to predict its future behaviour. This approach is know as the *learning methodology* and in the case where the examples are represented as a set of input-output pairs it is called *supervise learning* [1]. This approach is similar to the way children learn, which is based on generalizing from examples rather than applying given definitions. After the "training" the resulting system, which sometimes is referred to as the *decision function*, can make decisions that follows in the footprint of the target function.

1.1.2 Financial Surveillance

Financial market operators are required by law to perform market surveillance in order to detect illegal market activities, e.g. money laundering or insider fraud. Furthermore it is in the market operators' interests to keep their market attractive to honest customers. However due to the financial development during the last decades in foremost the western countries but also many development countries overall activity in financial markets has drastically increased. This has of course increased the need for more efficient surveillance systems which need to be automated as far as possible. Why this is another rapidly growing area for implementation of machine learning techniques.

Scila AB

This project will be carried out on the request of *Scila AB*, which is a company within the field of financial surveillance development. They are based in Stockholm but act globally, with customers in Europe, Asia and the MENA region. This company was founded in 2008 and has been tremendously successful in a very short time with a verity of target customers, e.g. trading venues, investment banks, brokers and regulators.

Their product is integrated into the customer's system where it looks for suspicious anomalies. When such is detected a human operator is alerted, which initiates an investigation that in turn leads to a classification of the alarm as valid or false. Among the pool of alarms raised during an average business day there are very few that are in fact valid. However since this product is very general the criteria of a valid alarm varies among different customers, hence a strict inflexible filter suited for one customer could render devastating misses of valid alarms for an other. There is an obvious need for a flexible dynamic filter that can adjust the criteria for filtering to the specific needs of each customer. Thus it is likely that machine learning is a cost-efficient complement to traditional methods.

1.1.3 Goals

The task appointed to us is to implement a learning model that can learn from the previous alerts already classified by human operators to try to classify future alerts.

1.2 Related Work

The problem at hand is a classic one, well suited for a number of different machine learning algorithms. Several of the most common approaches will be briefly described and evaluated in the following sections.

1.2.1 The Naïve Bayes Classifier

Probably the most common and well known approach is the naïve Bayes classifier[2]. We want to determine which class, C_i, an object belongs to with regard to the parameter X_j. This is the *conditional* probability that given the value X_j, the object belongs to the class C_i. This is denoted $P(C_i|X_j)$. By examining old data we can easily get the *conditional* probability that if the object belongs to the class C_i the value is X_i. We write this as $P(X_i|C_i)$. Now the *joint* probability of C_i and X_j is:

$$P(C_i, X_j) = P(X_j|C_i)P(C_i) \tag{1.1}$$

or equivalently:

$$P(C_i, X_j) = P(C_i|X_j)P(X_j) \tag{1.2}$$

Since the equations are equal we can rearrange the terms and arrive at:

$$P(C_i|X_j) = \frac{P(X_j|C_i)P(C_i)}{P(X_j)} \tag{1.3}$$

which is *Bayes rule* and the foundation of the naïve Bayes classifier.

Now, if every observation X_k belongs to a class C_i we can compute:

$$P(X_k) = \sum_i P(X_k|C_i)P(C_i) \tag{1.4}$$

which we already know from previous data. Bayes rule allows us to calculate posterior probabilities from known prior probabilities which allows us to classify new data.

If the data contains several features i.e. $P(\bar{X}_j|C_i) = P(X_j^1, X_j^2, ...X_j^n|C_i)$. This can be simplified by assuming that the elements of the feature vector is conditionally independent of each other.

The classification rule is then simply: given an feature vector $\bar{X}_j$, calculate the conditional probabilities for each class and choose the one with the highest probability:

$$P(C_i)\prod_k P(X_j^k = a_k|C_7 i) \tag{1.5}$$

However the naïve Bayes classifier has some drawbacks[3]. The first and most obvious is the fact that the derivation of Bayes rule only holds for *conditionally independent* variables. This is often a very unrealistic assumption in real-life applications. Another serious flaw is that the denominator in Bayes rule normalizes so the probability of getting any class with given X_k to unity. This means that if the training set is biased and not really representative of the real data the probabilities will be biased and decrease performance.

1.2.2 Neural Networks

A natural approach to create a learning classifier is to attempt to mimic our own learning process by simulating a simple brain. Very roughly the brain consist of billions of neurons. A neuron acts like a relay. From a nerve or another neuron ions are emitted and if the concentration reaches a certain limit, i.e. the electric potential is above or below a certain threshold, the neuron fires and releases ions to the next neuron[2].

This can be simulated by using a step function. We give an input and if it's above a certain level, most commonly zero, it sends a new value which is predetermined to be positive or negative. However a single neuron isn't going to learn much. But if we combine several in a single layer, as shown in Figure 1.1, we start to get some results.

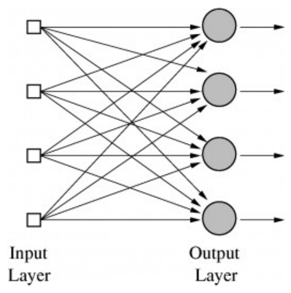


Figure 1.1. Single Layer Neural Network

However the network still isn't learning. By adding weights to each neuron we can start to correct the network after each trail. We use an input for which we know the desired output and we simply check which neurons fired and which didn't. Now we update the weights to minimise the error and this way the network begins to adapt.

What this method actually does is attempting to separate the classes with a straight line in 2D, a plane in 3D and a hyperplane in higher dimensions. To solve problems that aren't linear you need more neurons in a single layer or more layers. This is called a multi layer neural network and is shown in Figure 1.2.

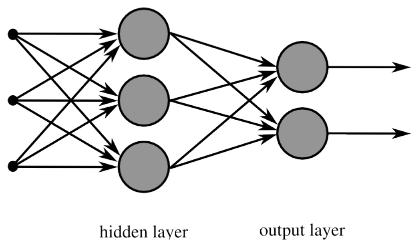


Figure 1.2. Multi layer neural network

The second layer is called a 'hidden' layer, since it cannot be directly observed.

Given an input it response with a correct of incorrect output. If it is incorrect there is no way of knowing which neuron or neurons in what layer that screwed up. However this can be solved by replacing the step function from the neuron with the sigmoid function. This function is very similar to the step function but is has a continuous derivative. This means that we can use the derivative, and if there's many hidden layers the chain rule, to minimise the error in the hidden layers as well.

A downside with neural networks is the fact that it's hard to understand them. If you would 'open' them and check the values of every neuron it would only seem to be random numbers. The complexity of even a small network makes it a 'black box'. You feed the network with an input, you get an output that's probably right but you have no idea what happened in between.

Also if you want to solve complex problems you need large networks and this means you're going to suffer from the curse of dimensionality. As your number of neurons increase the training time increases exponentially which leads to limitations and the need for simplification or massive computations.

1.2.3 Artificial Immune Systems

Artificial immune systems share some similarities with neural networks. You define a number of detectors with individual weights similar to anti-bodies in our own immune system. These are specialized and searches for certain patterns and if a detector successfully finds or classifies an input it's rewarded by incrementing it's weight and opposite by decrementing it if it wrongfully classifies an input.

Just as the naïve Bayse classifier this method is vulnerable to 'unbalanced' data and can become biased[5].

1.2.4 Support Vector Machines

Support Vector Machines is another mathematical theory founded approach to learning and classification. Just as the neural networks this method attempts to separate classes with a function. Every input you want to classify consists of a number of values. These can be seen as base vectors in an n-dimensional room. For the sake of simplicity, regard an example of two classes in a 2-dimensional vector space and assume that the classes are linear. In this case there exists a line that can separate the two classes and only the points from the different classes which are nearest this line is relevant. We only need to compare a new input with these points. Figure 1.3 displays a maximum margin with the associated support vectors.

As seen in the figure the choice of the separating line isn't the only one. However it is the one that maximises the distance between the two classes. But what if our classes aren't linear and no straight line exists between them? Using an appropriate kernel function we can map our input onto a more favourable vector room. Figure 1.4 shows the mapping of data from an linear un-separable space onto a separable one.

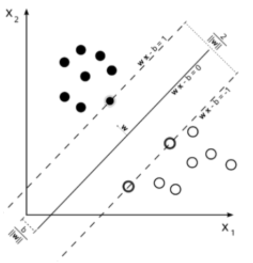

x_2
x_1
$\mathbf{w}\cdot\mathbf{x}-b=1$
$\mathbf{w}\cdot\mathbf{x}-b=0$
$\mathbf{w}\cdot\mathbf{x}-b=-1$
$\frac{2}{\|\mathbf{w}\|}$
$\frac{b}{\|\mathbf{w}\|}$
$\mathbf{w}$

Figure 1.3. Support vector with maximum margin

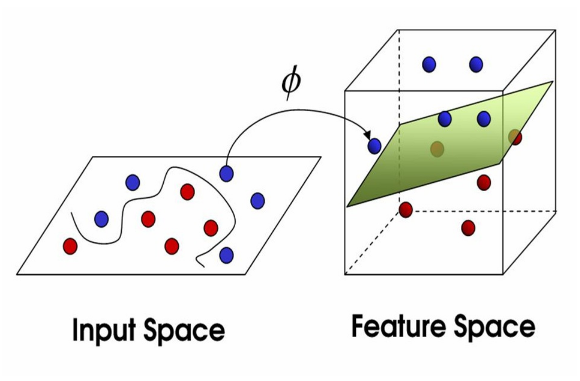


Figure 1.4. Changing the input space to a more favourable feature space

The SVM is built to separate between two classes but if there are methods to generalize this. The most common are to train an SVM to differentiate between one and all others or two simply train one for each pair of classes. The first method is known as one vs all and the other as all vs all. When you want to classify an input you simply let all your SVMs evaluate it and choose the one with best margin[1].

A complete derivation of the SVM will be given in Section 1.3.

1.2.5 Examples & Evaluation

One of the most if not the most common classification problem today is sorting out e-mail spam messages. This resemble our problem at hand quite a bit with the exception that it's often binary, it's either spam or legit.

Of all the different classifiers mentioned above the one who is most commonly used is the SVM. It has many pros and few cons compared to the other. The naïve Bayes classifier requires balanced data and assumes that the different features are conditionally independent. Neural networks are 'black boxes' with little insight and suffers from the curse of dimensionality when the problems becomes complex. The artificial immune systems can't simply keep up with the adaptiveness or the effectiveness of the SVM. In spam filtering the SVM is very versatile and effective[5] and in text classification it outperforms the other algorithms with margin[6]. SVMs also handles the problem of overfitting[4] with several techniques and the central kernel trick reduces calculation costs significantly. By choosing an appropriate kernel incorrect classifications as well as computational cost can be further decreased[7]. Even though the SVM is created to separate two classes the one vs all or all vs all makes it one of the best, if not the best, multi-class classifiers with several different methods to further boost the effectiveness[8]. The algorithm of choice for the classification problem at hand is the SVM which will be described in detail in the next section.

1.3 Theory

1.3.1 Convex optimization & Lagrange multipliers

To fully understand how SVM work we first have to equip ourselves with a few mathematical tools. The first of these is the theory of convex optimization.

We begin by defining the *primal optimization problems* as follows:

Given functions f, g_i, $i = 1, ..., k$ and h_i, $i = 1, ..., m$ defined on a domain $\Omega \subseteq \mathbb{R}^n$

$$\begin{aligned} &\textit{minimize} && f(\bar{w}), && \bar{w} \in \Omega \\ &\textit{subject to} && g_i(\bar{w}) \leq 0, && i = 1, ..., k \\ & && h_i(\bar{w}) = 0, && i = 1, ..., m \end{aligned} \tag{1.6}$$

The set where $f(\bar{w})$ is defined and the constraints are met is called the *feasible region.* A solution to the optimization problem is a point $\bar{w}^*$ in this region where there exists no other point $\bar{w}$ so that $f(\bar{w}) < f(\bar{w}^*)$. This is in other words a global minimum. But if we find a solution, how can we be sure that $f(\bar{w}) < f(\bar{w}^*)$ is actually true and we haven't just found a local minimum. If the set Ω is convex, i.e. the Hessian of $f(\bar{w})$ is strictly positive, we are assured that $\bar{w}$ is a global minimum[1].

When we are faced with a primal optimization problem on a convex set as described above we can use the method of Lagrange multipliers. For an optimization problem with objective function $f(\bar{w})$ and only equality constraints $h_i(\bar{w}) = 0$, $i = 1, ..., m$ we define the *Lagrangian function* as:

$$L(\bar{w}, \bar{\beta}) = f(\bar{w}) + \Sigma_{i=1}^{m} \beta_i h_i(\bar{w}) \tag{1.7}$$

where the coefficients β_i are called the *Lagrange multipliers.*

From this theory one can derive that in order for a point $\bar{w}^*$ to be a minimum of $f(\bar{w})$ subject to $h_i(\bar{w}) = 0$, $i = 1, ..., m$ with $f, h_i \in C^1$ the following conditions must be true:

$$\begin{aligned} \frac{\partial L(\bar{w}^*, \bar{\beta}^*)}{\partial \bar{w}} &= 0 \\ \frac{\partial L(\bar{w}^*, \bar{\beta}^*)}{\partial \bar{\beta}} &= 0 \end{aligned} \tag{1.8}$$

for some values of β^*. These conditions also implies that $L(\bar{w}, \bar{\beta}^*)$ is a convex function of $\bar{w}$.

However this problem only included the equality constrains, not the inequality constrains which we want to include. We need a new definition:

Given an optimization problem with domain $\Omega \subseteq \mathbb{R}^n$

$$\begin{aligned} &\textit{minimize} && f(\bar{w}), && \bar{w} \in \Omega \\ &\textit{subject to} && g_i(\bar{w}) \leq 0, && i = 1, ..., k \\ & && h_i(\bar{w}) = 0, && i = 1, ..., m \end{aligned} \tag{1.9}$$

we define the *generalised Lagrangian function* as:

$$\begin{aligned} L(\bar{w}, \bar{\alpha}, \bar{\beta}) &= f(\bar{w}) + \Sigma_{i=1}^{k} \alpha_i g_i(\bar{w}) + \Sigma_{i=1}^{m} \beta_i h_i(\bar{w}) \\ &= f(\bar{w}) + \bar{\alpha}\bar{g}(\bar{w}) + \bar{\beta}\bar{h}(\bar{w}) \end{aligned} \tag{1.10}$$

We can use this to define the *Lagrangian dual problem*:

$$\begin{aligned} &\textit{maximize} && \Theta(\bar{\alpha}, \bar{\beta}) \\ &\textit{subject to} && \bar{\alpha} \geq \bar{0} \end{aligned} \tag{1.11}$$

where $\Theta(\bar{\alpha}, \bar{\beta}) = inf_{\bar{w} \in \Omega} L(\bar{w}, \bar{\alpha}, \bar{\beta})$.

The dual problem is upper bounded by the primal problem and when the primal problem is convex the solutions to the dual and primal problem is the same[1]. This means that we can find a solution that satisfies both the dual and the primal problem, we have found the global minimum. This is a consequence of the strong duality theorem which states:

Given an optimization problem with convex domain $\Omega \subseteq \mathbb{R}^n$,

$$\begin{aligned} &\textit{minimize} && f(\bar{w}), && \bar{w} \in \Omega \\ &\textit{subject to} && g_i(\bar{w}) \leq 0, && i = 1, ..., k \\ & && h_i(\bar{w}) = 0, && i = 1, ..., m \end{aligned} \tag{1.12}$$

where g_i and h_i are affine functions, that is

$$\bar{h}(\bar{w}) = \bar{A}\bar{w} - \bar{b} \tag{1.13}$$

for some matrix $\bar{A}$ and vector $\bar{b}$, the duality gap is zero[1].

We are now fully equipped to derive the theory of Support Vector Machines.

1.3.2 Support Vector Machines

As described in Section 1.2.4 SVM:s attempts to classify data by separating it with a hyperplane based on the training data.

The most simple version of an SVM is a linear hard-margin classifier. This means that it can only separate linear data and it will attempt to classify the data *exactly*. While this might seem very good one has to remember that we are training the classifier using *one* set of data and we want it to be able to classify *any* data. A hard-margin classifier will most likely be *over-fitted* and generalize poorly when being presented new data. This is especially true when the training data is noisy. Fortunately both these problems can be countered. We begin by defining the margin M as seen in Figure 1.5

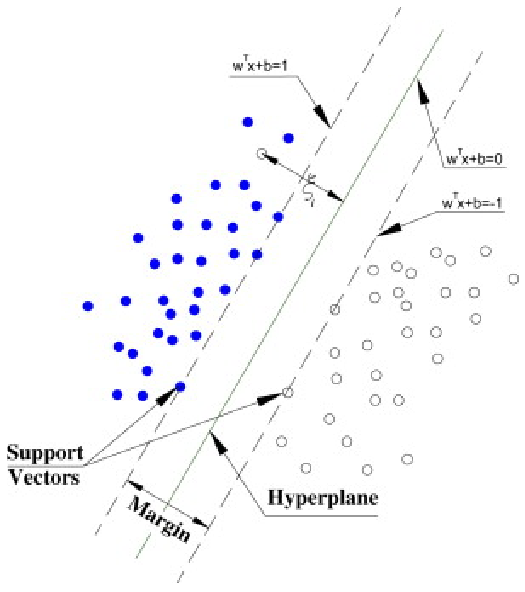


Figure 1.5. A linear soft-margin classifier

The line separating the two classes can be written as:

$$y = \bar{w} \cdot \bar{x} + b \tag{1.14}$$

where $\bar{x}$ is the input vector and $\bar{w}$ is the weight vector which determines how important an input is. From this we can express the margin M as:

$$M = \frac{1}{2\sqrt{\bar{w} \cdot \bar{w}}} \tag{1.15}$$

In order to find the line which maximize the margin and gives us the most accurate classification we need to minimize the norm of $\bar{w}$, i.e. minimize $\sqrt{\bar{w} \cdot \bar{w}}$. Furthermore we define the values of our two target classes to be ± 1 and the target function t_i to be the class which an input $\bar{x}_i$ belongs to. To ensure that an input is in the correct class we also get the constraint:

$$t_i(\bar{w} \cdot \bar{x} + b) \geq 1 \tag{1.16}$$

To handle the problem of with hard-margin and over-fitting we introduce another feature, slack variables denoted ϵ_i. These will allow the classifier to accept a certain level of error and thus make the separating hyperplane smoother which will allow it to better generalize. We are now ready to formulate our primal optimization problem:

$$\begin{aligned} &minimize \quad \bar{w}\cdot\bar{w} + C\Sigma_{i=1}^{l}\epsilon_i^2 \\ &subject\ to \quad y_i(\bar{w}\cdot\bar{x}_i + b) \geq 1-\epsilon_i,\ i=1,...,l \end{aligned} \tag{1.17}$$

However in our optimization problem we now have two variables, $\|\bar{w}\|$ and ϵ_i. How this should be solved is up to the application and the data to be classified. Choosing a big C will mean very strict boundaries and a small C might allow to many miss-classifications. This is a trade off between the two. To find the optimal C-value for the problem at hand a common technique is cross-validation which will be described in the next chapter.

We can now use the theory of Lagrangian multipliers to formulate the dual optimization problem:

$$\begin{aligned} maximize \quad & L(\bar{w}) = max\Sigma_{i=1}^{l}\alpha_i - \frac{1}{2}\Sigma_{i=1}^{l}\Sigma_{j=1}^{l}\alpha_i\alpha_j t_i t_j \bar{x}_i\cdot\bar{x}_j \\ subject\ to \quad & 0 \leq \alpha \leq C \\ and \quad & \Sigma_{i=1}^{l}\alpha_i\bar{x}_k = 0 \end{aligned} \tag{1.18}$$

However one last problem is that the classifier is still linear but if take another look at eq. 1.18 we notice that the input vector $\bar{x}$ only exists in an inner product with another input vector. There is no reason why we cant modify this to another function of $\bar{x}$, more accurately a kernel function. This changes the normal euclidean dot product an inner product which transforms and projects our data onto a more favourable vector space. However projecting your data from a linear vector space onto a higher dimensional or even infinite dimensional space means that the computation of the inner product is way more expensive then the simple dot product.

What saves us is that we actually never need to do any computations in the much more complex space, a good choice of kernel can always be reduced to a function containing the euclidean dot product and doing the computations in a smaller subspace. But which kernel should we choose?

There is plenty of inner product to choose from but we need one that is positive definite which implies that it is convex. To name a few one can use a polynomial kernel, a sigmoid kernel but the most common and well used is the Gaussian or Radial Base Functions expansion kernel:

$$\bar{K}(\bar{x}_i,\bar{x}_j) = exp(-\gamma(\bar{x}_i-\bar{x}_j)^2) \tag{1.19}$$

where $\gamma = \dfrac{1}{2\sigma^2}$ and thus is always positive. The RBF kernel is often the best because it doesn't have computational problem as many of the other kernels do have. The sigmoid kernel can have unusable parameters and the polynomial kernel might go to zero or diverge to infinity, both which are bad when trying to solve a problem numerically[9].

Now our SVM is ready for training and then finding the optimal parameters for the kernel, γ, and for the strictness of the classifier, C, to achieve optimal performance. This will be covered in the next chapter.

Chapter 2

Implementation & Testing

2.1 Method

The task to find optimal parameters for our SVM is not a trivial one. It depends entirely on our dataset and if we're unlucky it might not even be possible. There are a few different ways to do this but one in particular has shown to be quite effective[9] and simple why it now is widely used.

2.1.1 Grid-search

As naïve as it might sound, the very crude approach of simply testing a huge number of values in a systematic fashion is quite effective. The idea is to search through the space spanned by the parameters γ and C in rather large steps. Normally you start with values in powers of 2 from about -25 to 25 for both parameters. A finer search can then be conducted around the best regions if the results were inadequate.

An example of a more sophisticated method is the Gradients accent. Here one looks at the direction in the grid where the accuracy increases and move in that direction. The downside of this process is that it is very iterative and very hard to parallelize compared to a grid-search where every point is independent and can thus be easily calculated separately [9].

Another advantage of the grid-search approach is that it is very easy to implement. Once you set up the training of your SVM all you have do is repeat this for every point and evaluate it using Cross-validation.

2.1.2 Cross-validation

A problem with most classifiers is over-fitting. This is when you train your classifier "to hard" on your training data, often achieving great results during training, and when tested on new data it performs poorly. By over-fitting the classifier to the training data it has lost the ability to generalize to new data which is unsatisfactory. This can be prevented or at least reduced using cross-validation, CV[9].

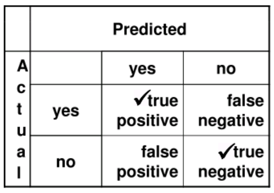
In n-fold CV you divide your training data into n different folds. Sequentially one fold is tested using the classifier trained on the remaining $n-1$ folds. The accuracy is then taken as the average of the n test. By doing this we can train and test every point in the grid-search on n "different" data sets. If a certain value of γ and C generalizes poorly the average accuracy will be poor as well and we can continue our search for better parameters.

A normal number of folds is 10 but for large datasets this might be incredibly time consuming and computational heavy and one might want to reduce the number of folds.

2.1.3 Accuracy Measurements

Up until now we have mentioned accuracy and results but no motivation of which accuracy to use or what a good result is. It's easy to simply state: "The number of correct classified items over all items must be the best accuracy measurement!". However giving this a second thought proves this to be slightly naïve, unfortunately. Our main goal in this project is to sort out the few number of real alerts, positives, from a huge number of false alerts, negatives. The classifier can easily achieve a great accuracy by simply classifying everything as false alerts, but in our case this is entirely useless.

A more common way to visualize performance in supervised learning is through a *confusion matrix*. For our binary case this is a two-by-two matrix with classified positives and classified negatives on one side and the real positives and real negatives on the other side. Figure 2.1 shows a standard confusion matrix.



		Predicted	
		yes	no
Actual	yes	✓true positive	false negative
	no	false positive	✓true negative

Figure 2.1. The layout of a confusion matrix in binary classification

The diagonal contains the true positives, TP, which is the correct classified positives, the true negatives, TN, which is the correct classified negatives. The other two entries is the false positives, FP, which is negatives miss-classified as positives and the false negatives, FN, which is positives miss-classified as negatives. This is called a confusion matrix since FN and FP is the items the classifier has confused with the wrong class. If both FN and FP is zero we have achieved a perfect classifier.

From this we can construct smarter measurements than our first naïve accuracy. We define the *precision* as:

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

and the *positive and negative recall* as:

$$Positive\ Recall = \frac{TP}{TP + FN} \qquad Negative\ Recall = \frac{TN}{TN + FP} \tag{2.2}$$

The precision is the ratio of correct classified positives over all classified positives. A high precision indicates few miss-classified negatives, while low values means that we have classified way to many as positives.

The recalls on the other hand is a measurements of how many points out of the set of one single class we have "found". It is noteworthy that if everything is classify as positive for instance the positive recall will peak since we haven't missed any real positives. The precision on the other hand would plummet.

These two measurements complement each other, why a good classifier renders high values in both. However finding a balance between them is difficult, thus three further accuracy measurements will be used.

F-score is the harmonic mean of the precision and positive recall.

$$F - Score = \frac{2 \times Precision \times Positive\ Recall}{Precision + Positive\ Recall} \tag{2.3}$$

Another used accuracy measurement is the *Matthews correlation coefficient* (MCC), that is generally considered a balanced measure which can be used even if the classes are unbalanced.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.4}$$

The final measurement is the *balanced accuracy* (BAC) that simply is the mean of positive and negative recall.

$$BAC = \frac{Positive\ Recall + Negative\ Recall}{2} \tag{2.5}$$

2.1.4 Implementation & Testing

For our task we have chosen to use the open-source libsvm project which is pretty much the standard SVM library for research these days. The original project is written in C and we have used a version ported to Java for convenience with the Scila software.

The first task was data processing and extracting the features to use for classification. Both the number of features and the size of the values differed greatly between different alerts and we choose to focus on a few of the most frequently occurring. Because of this we choose to train a different SVM for each alert type since they were far to different to compare.

The next step was scaling the data which is usually necessary to get decent results. Otherwise a feature with huge values might completely dominate the optimization process. If you rescale your data to the interval $[-1, 1]$ you can avoid these and also speed up the computations due to lower values.

Once this was done we set up a grid-search by creating a thread for each point which in turn handled cross-validation. This allowed us to parallelize the process and significantly reduce the computation time.

However to get both a high precision and recall is not trivial. Since the dataset contained only about a few percent of positives while the rest was negative, our classifiers became biased towards the negatives. It seldom classified any items as positives for any point in the grid-search. The good news is that libsvm has an options for dealing with unbalanced data sets. One can tweak the *weights* or *costs* of miss-classifying a positive or negative item. We increased the cost for miss-classifying the positives with a ratio of the number of positives over negatives in the actual dataset, which improved the results.

2.1.5 Data Labeling

Since the data provider was not able to label the data in beforehand some work needed to be done on labeling the data to be able to train and validate the models. The body of a single alarm in the dataset contains apart from a set of trigger parameters (later used as features) also an identification number. This number can be used to find related entries or so called "issues" in an separate dataset also containing information and notes about the manual handling of the related alerts. These notes were used as far as possible to designate the specific labeling of the issues and thus all the alarms.

Chapter 3

Results

3.1 General statistics

Some overview statistics of the provided dataset is presented in tables 3.1 and 3.2. The negative-positive-ratios are later used for further analysis.

Type	Count	Pos. %	Neg. %	NP-ratio	Undefined %
Volume Change	391 220	1,4	87,3	62,4	4,2
Trade To Trade	70 000	1,3	87,3	67,2	11,4
M. M. Obligation	56 485	20,0	60,0	3,0	20,0
Ramping	52 606	4,5	75,2	16,7	20,3
Unlikely Order	37 427	2,2	96,9	44,0	0,9
All Alarms	**828 272**	**3,9**	**88,1**	**22,6**	**7,9**

Table 3.1. Statistics for the five most frequent alarm types in the provided dataset. The percentage of the data made up by the positive and negative classes along with the negative-positive-ratio is presented. The "Undefined" percentage is the subset that could not be determined belonging to ether class and is thus considered unusable data.

Type	Count	Pos. %	Neg. %	NP-ratio	Undefined %
Trade to trade 2%	49 975	1,2	87,1	72,6	11,7
Ramping 0,3%	26 650	1,2	87,2	72,7	11,6
Trade to trade 5%	24 324	2,3	84,1	36,6	13,7
Ramping 5 Ticks	13 500	6,5	61,6	9,5	31,8
Ramping 3 Ticks	7 076	10,6	61,2	5,8	28,2
Ramping 7 Ticks	2 983	12,0	53,2	4,4	34,8

Table 3.2. Similar statistics as table 3.1 but for the subcategories of the "Ramping" and "Trade to trade" alarm types.

3.2 Spatial Disposition of Data

Figures 3.1 an3.2 shows the general spatial disposition of data for two different alarm types along three different features. The data is scaled to values between -1 and 1. Green and red points represent true (positive) and false (negative) alarms, respectively.

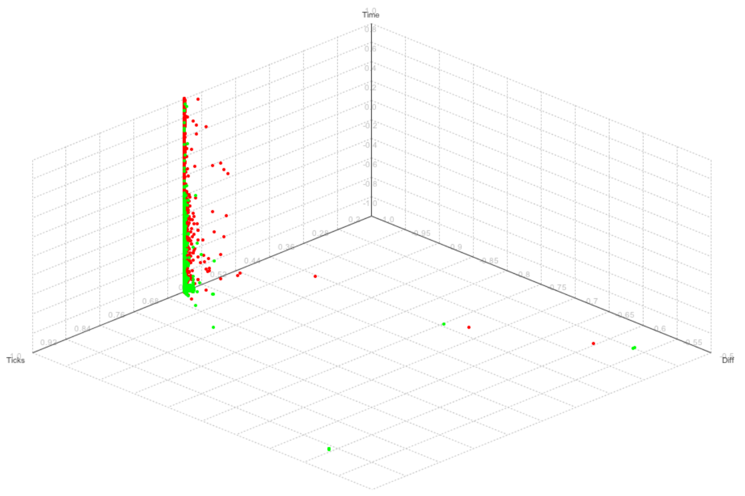


Figure 3.1. Plot of the dataset for the "Ramping Ticks" alarm type for three selected features: Ticks, time and diff.

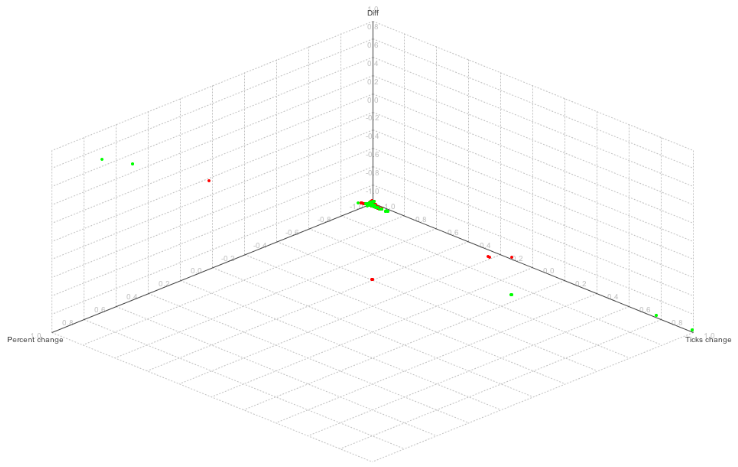


Figure 3.2. Plot of the dataset for the "Trade to Trade" alarm type for three selected features: Percent change, ticks change and diff. Both positive and negative extreme outliers are visible far from the main cluster at (-1,-1,-1), where the majority of points are clotted together.

3.3 Grid-search

In this section a heatmap representation will be used to visualize the results for the different accuracy measurements. A log-log scale will be used labeled with powers of 2. For most plots the global maximum will be of relevance, why these will be marked with a black square. In cases where the global maximum is shared by multiple points all of them will be marked in the same way.

3.3.1 Ramping 3 Ticks

The "Ramping 3 Ticks" alarm subtype was selected due to its convenient and well understood features. It was scaled and had its extreme outliers removed ahead of the grid-search and contains a total of 5081 datapoints after exclusion of the unclassified subset. Maximum execution time for each grid-search point is 40 min. "Ticks", "diff" and "time" are the names of the three selected features.

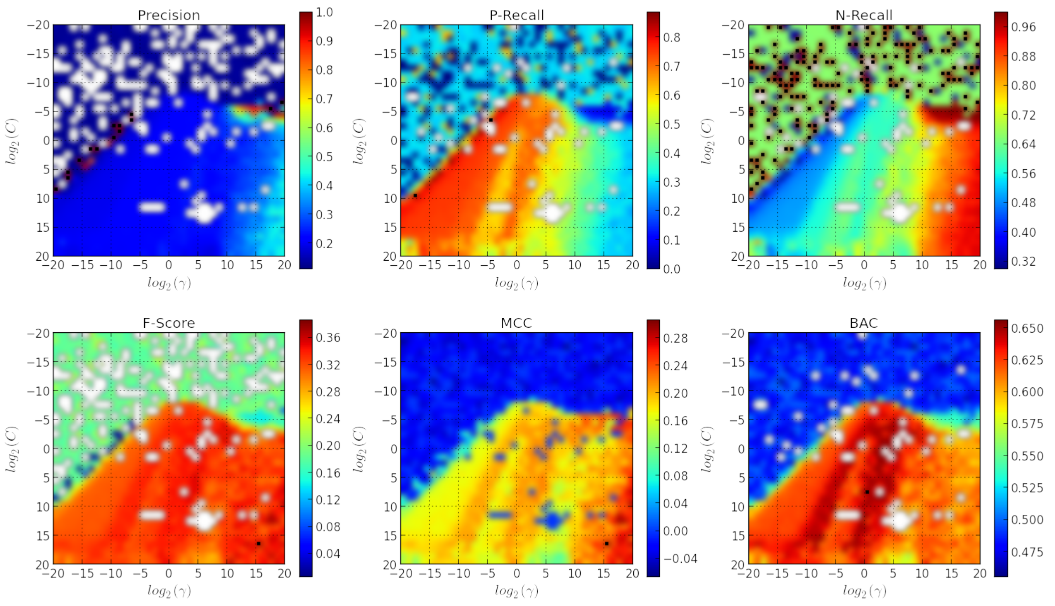


Figure 3.3. All the relevant accuracy measurements acquired from a C/γ-grid-search for the "Ramping Ticks 3" dataset using a weight of 5,8 for the positive class. The resolution is 40 by 40, thus a total of 1600 points. Global maximum for each individual subplot is marked with a black square.

	Precision	P-Recall	N-Recall	F-Score	MCC	BAC
F-Score	0.4569	0.3333	0.9333	0.3855	0.3053	0.6333
MCC	0.4569	0.3333	0.9333	0.3855	0.3053	0.6333
BAC	0.2293	0.7186	0.5934	0.3477	0.2201	0.6560

Table 3.3. Matrix of accuracy values at specific peak coordinated, using a positive class weight of 5,8. Columns show the specific accuracy values at the same C/γ-coordinates as the peak value of the accuracy measurements of the row. For instance the bottom most left cell shows the precision value at the peak point of the BAC-measurement. Precision, P-Recall and N-Recall rows excluded due to multiple peak points.

Weight	Precision	P-Recall	N-Recall	F-Score	MCC	BAC
5,4	0.2816	0.5656	0.7571	0.3760	0.2499	0.6614
5,6	0.2294	0.7213	0.5920	0.3481	0.2210	0.6567
5,8	0.2293	0.7186	0.5934	0.3477	0.2201	0.6560
6,0	0.2298	0.7213	0.5929	0.3485	0.2217	0.6571
6,2	0.2310	0.7077	0.6035	0.3484	0.2201	0.6556

Table 3.4. Values of different accuracy measurements (columns) for different positive class weights (rows) at the C/γ-coordinates of peak BAC-values.

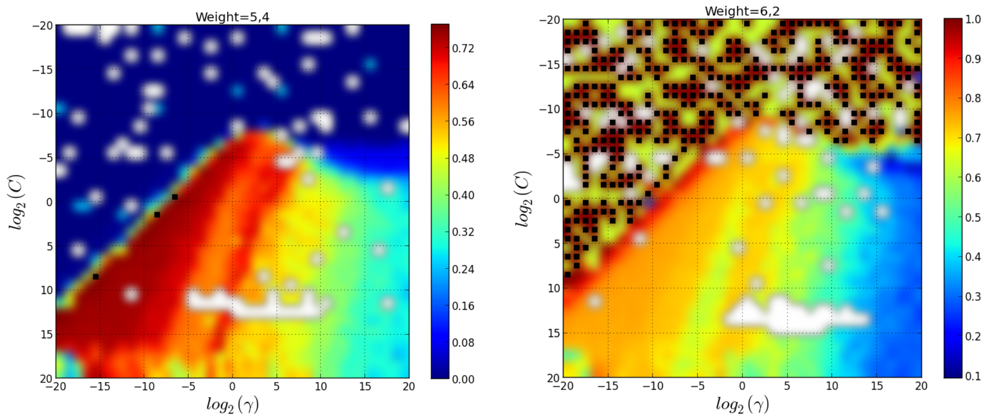


Figure 3.4. Positive recalls for a positive class weights of 5,4 and 6,2.

3.3.2 Ramping 0,3%

The "Ramping 0,3%" alarm subtype was selected due to similar reasons as "Ramping 3 Ticks". The dataset was also prepared in a similar fashion and had after exclusion of unclassified data a total of 23 559 datapoints. The dataset was then shrunk by random selection to half the size before training. Maximum execution time for each grid-search point is 40 min. "Percent", "diff" and "time" are the names of the three selected features.

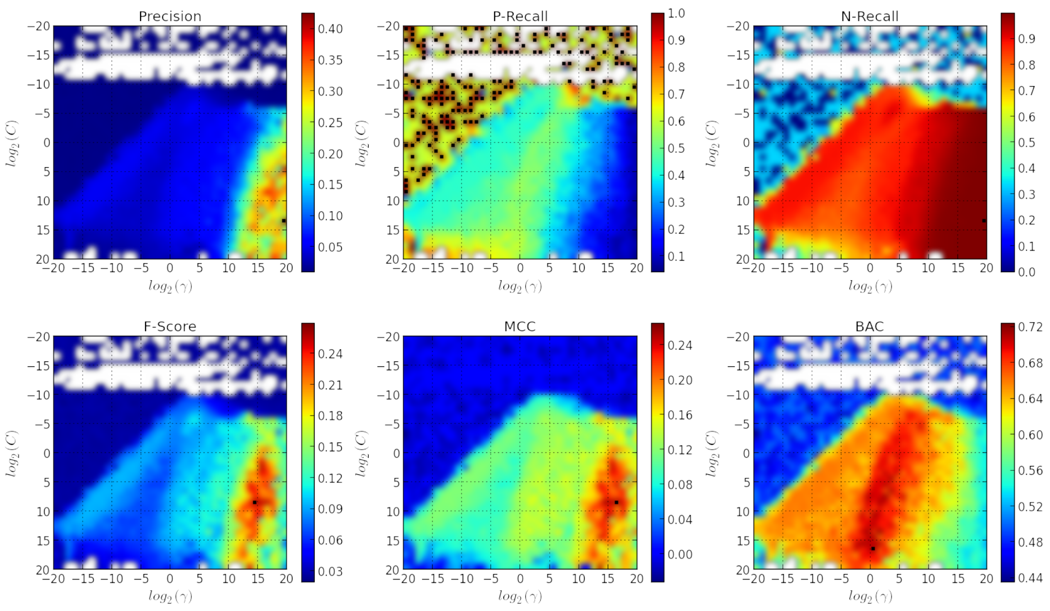


Figure 3.5. Relevant accuracy measurements from a C/γ-grid-search for the "Ramping 0,3%" dataset using a weight of 72,7 for the positive class. The resolution is 40 by 40, thus a total of 1600 points. Global maximum for each individual subplot is marked with a black square.

	Precision	P-Recall	N-Recall	F-Score	MCC	BAC
Precision	0.4250	0.1018	0.9980	0.1643	0.2028	0.5499
N-Recall	0.4250	0.1018	0.9980	0.1643	0.2028	0.5499
F-Score	0.2971	0.2455	0.9916	0.2689	0.2606	0.6186
MCC	0.3855	0.1916	0.9956	0.2560	0.2646	0.5936
BAC	0.0510	0.6108	0.8366	0.0942	0.1409	0.7237

Table 3.5. Matrix of accuracy values at specific peak coordinated, using a positive class weight of 5,8. Interpreted in the same fashion as table 3.3. P-recall excluded due to multiple peak points.

Weight	Precision	P-Recall	N-Recall	F-Score	MCC	BAC
71,9	0.0599	0.5689	0.8715	0.1084	0.1525	0.7202
72,1	0.0689	0.5749	0.8882	0.1230	0.1695	0.7315
72,3	0.0571	0.5928	0.8592	0.1042	0.1509	0.7260
72,5	0.0563	0.5928	0.8569	0.1028	0.1491	0.7249
72,7	0.0510	0.6108	0.8366	0.0942	0.1409	0.7237
72,9	0.0642	0.5689	0.8808	0.1154	0.1604	0.7248
73,1	0.0610	0.5629	0.8753	0.1101	0.1536	0.7191
73,3	0.0569	0.5988	0.8573	0.1040	0.1514	0.7281
73,5	0.0530	0.5928	0.8477	0.0973	0.1426	0.7202
73,7	0.0651	0.5629	0.8836	0.1166	0.1609	0.7232

Table 3.6. Values of different accuracy measurements (columns) for different positive class weights (rows) at the C/γ-coordinates of peak BAC-values.

3.4 Trade to Trade 2%

This dataset was after preparations consisting of 44 128 data points. The data was shrunk by random selection down to one forth the size, which is approximately the size of Ramping 0,3% used for training. Maximum execution time for each grid-search point is 60 min. "Percent change", "ticks change" and "diff" are the names of the three selected features.

	Precision	P-Recall	N-Recall	F-Score	MCC	BAC
Precision	0.3750	0.0207	0.9995	0.0392	0.0856	0.5101
P-Recall	0.0139	0.9103	0.1234	0.0273	0.0118	0.5169
F-Score	0.1250	0.0621	0.9941	0.0829	0.0795	0.5281
MCC	0.3750	0.0207	0.9995	0.0392	0.0856	0.5101
BAC	0.0236	0.4414	0.7527	0.0448	0.0514	0.5970

Table 3.7. Accuracy matrix of peak coordinated, using a positive class weight of 72,6 for Trade to Trade 2%. Interpreted in the same fashion as table 3.5. N-recall excluded due to multiple peak points..

Weight	Precision	P-Recall	N-Recall	F-Score	MCC	BAC
72,1	0.0220	0.4552	0.7257	0.0419	0.0464	0.5904
72,4	0.0317	0.3103	0.8719	0.0576	0.0621	0.5911
72,6	0.0236	0.4414	0.7527	0.0448	0.0514	0.5970
72,9	0.0260	0.3793	0.8075	0.0486	0.0541	0.5934
73,1	0.0233	0.4414	0.7498	0.0443	0.0505	0.5956

Table 3.8. Values of different accuracy measurements (columns) for different positive class weights (rows) at the C/γ-coordinates of peak BAC-values.

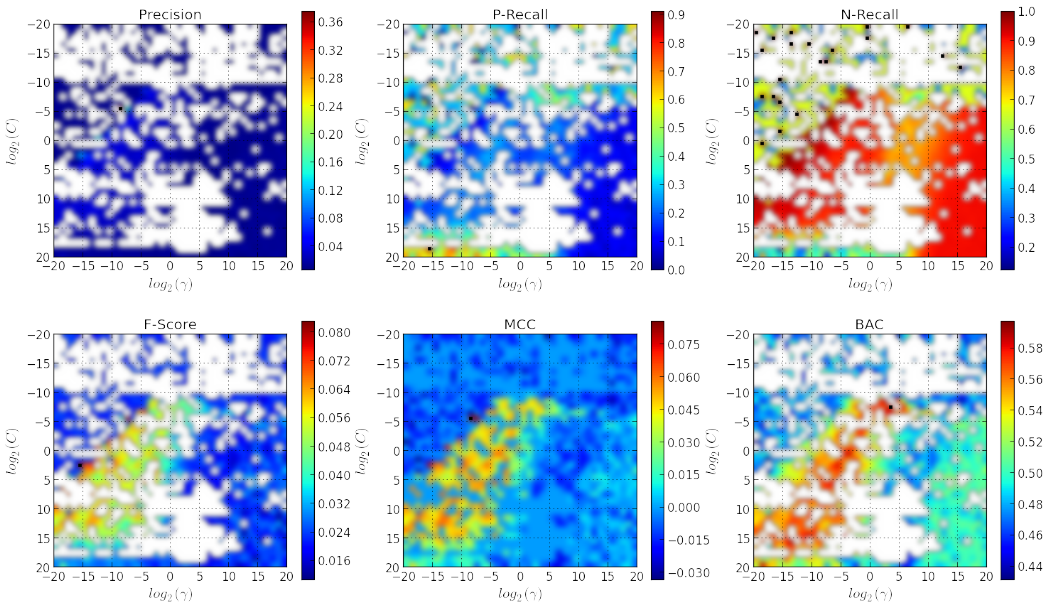


Figure 3.6. Accuracy measurements acquired from a grid-search for the "Trade to Trade 2%" dataset using a weight of 72,6 for the positive class. The resolution is 40 by 40, thus a total of 1600 points. Global maximum maximum marked in a similar fashion as other figures.

Chapter 4

Conclusions

4.1 Discussions

Unbalanced Data and the Use of Weights

Unbalanced data is a common problem in classification and usually leads to an over-estimation of the most occurring class, in our case the negative class. This distribution can be seen in Table 3.1. The negative-positive-ratio is quite large and as Table 3.2 shows the same applies for the subgroups of Ramping.

This comes from the fact that we use a soft-margin classifier and allow a certain amount of slack. The further into the wrong side a data point is the bigger the misclassification cost. But since the negative points are far more numerous it is still more cost-effective to simply classify the negatives correctly and neglect the positive points. As mentioned weights were introduced to counter this prblem.

This effect varies in magnitude and other regions of the the grid-search are more stable and changes more subtle. When increasing the weight more data points are classified as positive but the change is slower and without the huge jumps. This is likely due to the change in the maximum margin hyperplane. A larger margin makes the classifier more stable and would likely reduce the sudden changes caused by increasing weight.

4.1.1 Accuracy Measurements

To begin with we did not consider the selection of accuracy measurements as one of the challenges of this project. This was shown to be incorrect, mostly due to the unbalancedness of the dataset. We found that most available accuracies favored the correct classification of the majority class, if it was overwhelmingly more numerous.

Precision as for instance shown in figure 3.5 increase with increasing γ. This could in general be explained by the gauss bells in the RBF-kernel becoming narrower, which in turn makes the classified areas more precise in favor for the most numerous class. This does however not necessary imply better classification. Precision is a measurement of the ratio of true positives over all points classified as positive. This value increases of coarse with increasing true positives, but it also increases with decreasing false positives. In both figures 3.5 and 3.3 one sees that regions with high precision corresponds to regions with high negative recall. This fact emerges from the unbalancedness of the data, which specially in the case of Ramping 0,3% becomes quite substantial due to the large negative-positive-ratio. This renders this accuracy measurement rather unpractical on its own.

Positive and negative recall takes little effect of misclassification of the opposing class. Conclusions can mostly be drawn only about one of the classes, why considering a single plot of recall makes little sense.

However while the three less refined measurements used in this project (precision, positive and negative recall) have great limitations, they can be used to verify the quality of the three remaining more complex measurements. In tables 3.3 and 3.3 one can use for instance the positive and negative recall to draw qualitative conclusions about the different peak points.

F-score is in essence the harmonic mean of precision and recall and should at the first sight also be of interest. However data shows (table 3.3 and 3.3) that it suffers from the unbalancedness and thus favors the majority class although not nearly as much as precision. The same goes for MCC, which is developed with balance between classes in mind. Thus these two measurements are of interest if one are looking for best general classification of most data points of both classes.

It is however in our case of great importance to get an acceptable positive recall. The most intuitive interpretation of BAC is that it is equally affected by the accuracy of both classes. Since both recalls have the same significance on the final value, BAC could be imagined as a weighted accuracy of both classes. This fact is supported by the data, which renders BAC by far the accuracy measure of choice.

4.1.2 Ramping 3 Ticks

The theoretically best weight, i.e. the ratio between the occurrence of the negative and positive class, rendered a positive recall of 0.72 and an acceptable negative recall of 0.59 at maximum BAC (table 3.3). F-score and MCC peaks at the same grid-search point which corresponds to a positive and negative recall of 0.33 and 0.93.

The two alternatives pair of C and γ would result in classifiers of different character. If one imagine a dataset of 100 points with the same NP-ratio (i.e. 5.8), the BAC peak would in theory correctly classify 11 out of 15 positive points together with 35 misclassified negative points out of 85. The MCC/F-Score peak would, provided the same dataset, instead find 5 out of the 15 positives and present it together with 6 misclassified points out of 85 to the user.

Other weights were analyzed apart from the theoretically predicted one of 5,8. Accuracies at the BAC peaks are presented in table 3.4. The best values are acquired at weight 6,0 where also the best positive recall of 0.7213 is attained. This is however a minor improvement of only 0,4% and could be the effect of many factors including sheer luck.

Figure 3.4 shows the positive recalls for the weights 5,4 and 6,2. In combination with figure 3.3 one sees a dramatic effect taking place around the negative-positive-ratio. In regions for primarily small C the positive recall goes from zero to one for small variations of the weight. This is likely a result of the weights impact on the primal optimization problem. For weights up to the ratio the solution changes slightly, with little effect on the separating hyperplane. However at this threshold the sudden emergence of an entirely new more optimal solution strongly affects the shape of the hyperplane, which results in an entirely different classifier.

4.1.3 Ramping 0.3%

There is a region in figure 3.5 represented by white spots where the calculations were not completed within the time limit of 40 minutes. For some reason the optimization converged slower here. This does however not affect the peaks of the accuracy measurements since they are expected to occur in regions far from the canceled points.

For this alert type and the weight 72,7 for the positive class the F-Score and MCC peaks do not coincide, although they are close. The recalls are as expected even more in favor of the negative class, since the class ratio is much larger for this alert type.

The balanced accuracy peaks at a region of large C (65536) which is the equivalent of a large cost of misclassification and thus a stiffer classification boundary. This point would render a correct classification of almost one positive data point $(0, 83)$ while only 17 out of 99 negative points would be misclassified in a dataset of 100 points with the same class ratio.

Table 3.6 shows the BAC peak for different weights around the class ratio weight. One sees that the maximum peak BAC occurs at the weight 73.3, which is somewhat higher than the expected. This is once again a minor improvement and could be due to a different class ratio emerging after the randomly selected shrink of the dataset, as pointed in the results. This was however not investigated further.

4.1.4 Trade to Trade 2%

Figure 3.6 shows the grid-search results for the "Trade to Trade" subtype. One can see that even though this alarm type was shrunken to approximately the same size as the Ramping 0,3% dataset and given more computational time (60 minutes compared to 40 minutes) a lot of points where not completed. The obvious explanation is that this dataset was harder to separate. This could of course result in grid-search points with better accuracy being missed. But despite this fact the accuracy values presented in table 3.7 and 3.8 shows quite good results. A positive and negative recall of 0.44 and 0.75 was achieved, respectively, at peak BAC.

4.1.5 Computational time

One of the main drawbacks with SVM is the huge computational time. To begin with the main task is to solve a quadratic optimization problem where the complexity increases with the size of the dataset.

Moreover you have to find the optimal parameters which means you have to perform a grid-search. This is normally a 30-by-30 to 50-by-50 which corresponds to solving the problem between 900 and 2500 times.

Finally, to prevent over-fitting the standard tool is cross-validation which almost increase the complexity by a factor corresponding to the number of folds.

Unless you have a lot of computer power you are most likely forced to train on subsets, do a smaller grid-search and/or decrease the number of folds. These actions combined makes it more difficult to train a good classifier but it is unfortunately a price one might have to pay.

4.1.6 Data Handling

When starting this project we did not foresee the huge amounts of data handling we would have to do *before* even training a single classifier. The problems we encountered and how they were overcome or circumvented is discussed in the following section.

Labelling the Data

One of the uncertainties of this project from the start is that the data we had to work with was unlabelled as described in Section 2.1.5. Using this method we were able to label 92% of the data.

If the remaining 8% of the data is positive or negative is very hard to asses. This could have an impact on the results, for instance if many of them are positives. However, it is reasonably that this effect is rather limited, probably even negligible, due to the amount of "good" data already provided.

Subselection of Data

At first attempts were made to classify large parts of our dataset but the results where unsatisfactory. A decision was made to focus on smaller parts of the data, such as Ramping and TradeToTrade. The results were decent, but when examining the data closer we could see that what triggered an alert in one subgroup would not trigger in the others. This lead us to divide the data even further based on trigger parameter, which improved the results further.

Why did splitting the data into smaller groups effect the performance? Think of it this way: classifying if an apple is ripe or not based on its colour is almost impossible. Some apples are ripe when they are green, some when they are red and others anything in between. But if you focus on a particular sort almost every apple would have the same colour when ripe and the classification becomes much easier.

Effects of Outliers

As discussed in Section 3.2 the data contains some outliers. Several test were made both with and without the outliers but no improvement was made leading us to believe that the outliers were to few to significantly disturb the data.

4.2 Summary

First and foremost one must realize that classification through machine learning will always require an human operator to verify it's result when concerned with such important task as to detect money laundering and other illegal activities. We have been able to train an SVM that correctly classify 65-75% of the alerts and this can definitely be useful to ease the burden of operators.

To understand how to use the results obtained one must consider the versatility of the Scila software and how the different clients use it. Some customers want to trigger alerts constantly to observe the market while others only ever want alerts to trigger when something illegal is done. This means that operators for one client would like to focus on finding all positives, i.e. high positive recall. Operators who are flooded with false alerts however might instead go for a high negative recall to rule them out and focus on the remaining. Either way an SVM classifier would be useful for dividing a list of alerts into high and low priority lists. This way the operators can focus there attention where true alerts are most likely to occur.

However, to make this practically viable the process of training and updating classifiers must be more or less autonomous. This includes finding parameters and tweaking weights. Most effective is likely to train an SVM for each alert rule, such as Ramping 3 Ticks, and varying the weights around the NP-ratio as seen in Table [ref weight table]. The operators might then have to select parameters to use based on their preferences as mentioned above. When a significant amount of new data is collected it is possible to re-train the classifier in order to stay up to date with current alerts.

4.2.1 Future research

Even though quite impressive results were obtained using SVM, it would be interesting to try other methods such as neural network to find other patterns in the data.

The other possible approach is time-series evaluation. When using normal classifiers you group all your data and do not concern yourself with when the events took place. This might be especially interesting in this case as different alerts might trigger at different times of day or year in a regular pattern. A difficulty with this is to be able to handle when major events take place such as a large company bankrupting or a revolution in a country which influence trade in a region.

Unfortunately both of these approaches are outside the scope of this project and more research is needed.

Bibliography

[1] Nello Cristianini, John Shawe-Taylor *An Introduction to Support Vector Machines.* Cambridge University Press, 1nd Edition, 2000.

[2] S. Marsland, *Learning: An Algorithmic Perspective*, Taylor & Francis Group, First Edition 2009

[3] J. Rennie et al, *Tackling the Poor Assumptions of Naïve Bayes Text Classifiers*, ICML 2003, Washington DC

[4] Thiago S. Guzella, Walmir M. Caminhas, *A review of machine learning approaches to spam filtering*, Federal university of Minas Gerais, 2009

[5] Omar Saad et al, *A survey of machine learning techniques for spam filtering*, University of Helwan, 2012

[6] Thorsten Joachims, *Text categorization with Support Vector Machines: Learning with many relevant features*, Universität Dortmund

[7] Rong-En Fan et al, *Working Set Selection Using Second Order Information for Training Support Vector Machines*, University of Helwan, 2012

[8] Kai-Bo Duan1 o S. Sathiya Keerthi, *Which Is the Best Multiclass SVM Method? An Empirical Study* , Nanyang Technological University, 2005

[9] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin *A Practical Guide to Support Vector Classifcation* , National Taiwan University, 2010