MACHINE LEARNING

Assignment 2015-2016

JANUARY 13, 2016

NIGEL GRECH

1. Introduction

Support Vector Machines (SVM) are often referred to as one of the most important machine learning algorithms that have been created in recent history. The algorithm was invented during the 1960 by Vladimir Vapnik and Alexey Chervonenkis to perform however it wasn't until the 1990s before it began being used. The algorithm originally designed to perform binary classification of linearly separable patterns now has been extended to work with nonlinearly separable patterns, multiclass problems and even regression.

This document is divided into two main sections the first being a background on SVMs and the second a report on experiments carried out on SVMs

2. Background

2.1. Basic Idea of SVM

The principal idea behind SVMs, when considering patters which are linearly separable, is finding a hyperplane which separates the patters and which maximizes the margin of the hyperplane. Hence an SVM which finds a larger margin between patterns will be able to classify those patterns to a higher degree of accuracy. [1]

When considering non-linear separable patterns the problem gets tricky since a hyperplane cannot be placed between the patters to separate them. This is solved by mapping the patterns, usually into a higher dimension, to do this a mapping function known as a kernel function is used and will transform the patterns such that they will become linearly separable. [1] The problem here lies with the selection of the correct kernel function, the function can be selected based on knowledge of the patterns or in the worst case by a trial and elimination process.

2.2. Linear SVM

This section details the inner working of SVMs and some of the mathematics and theories involved in their development and function, for this purpose we will consider the simplest case for their use; a linearly separable classification problem were each of the examples are either positive or negative examples, as can be seen in figure 1. The following explanation is based off the works of B. Boser, C. Cortes, I. Guyon, and V. Vapkin [2], [3] and [4], and the survey and summarisations in the following works [5], [6], [7], [8] and [9].

The main goal of SVM is that of finding the optimal hyperplane which will separate the two classes with a maximal margin as can be seen in both figures 1 and 2. This largest possible hyper plane is often referred to as the widest possible street with the margin lines (H1 and H2 in figure 2) on either side of the median referred to as the gutters. The points which lie on the margins are called support vectors (the encircled points lying on the margin lines in both figures).

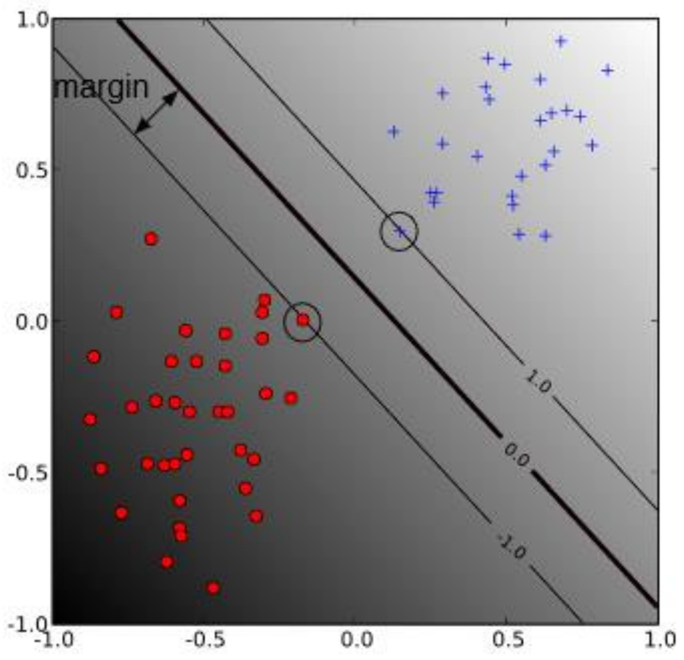

Figure 1 Example of Hyperplane separating two classes taken from: [8]

If we assume this hyperplane which we can be represented by the equation below, where b is some constant and $\boldsymbol{w}$ is a vector perpendicular to the hyperplane.

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

We can then create the following constraints which all the examples must satisfy

- For positive cases $\boldsymbol{w} \cdot \boldsymbol{x} + b \ \geq \ +1$
- For negative cases $\boldsymbol{w} \cdot \boldsymbol{x} + b \ \leq \ -1$

Introducing $\mathcal{Y}_i$ such that it is equal to +1 for positive examples and -1 for negative examples we can generalize the above two constraints into a single inequality

$$\mathcal{Y}_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) - 1 \ \geq 0 \ \forall_i$$

From the above inequality we can also conclude that those examples lying exactly on the margin lines (the support vectors) will be equal to 0. However at this stage $\boldsymbol{w}$ and the constant *b* are still unknown.

The hyperplane that the SVM uses to separate the examples and build the decision function which will classify new points can be found by maximizing the width of the hyperplane. Taking the difference between two support vectors, one positive example and one negative, and calculating the dot product and a unit vector perpendicular to the hyperplane we can calculate the width as a scalar product.

$$width = (\boldsymbol{x}_{+} - \boldsymbol{x}_{-}) \cdot \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

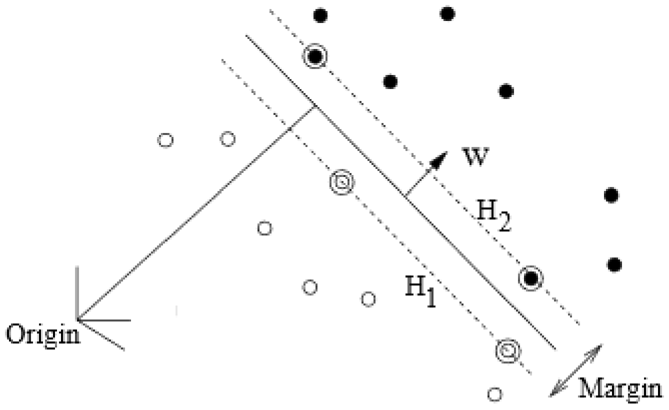

Figure 2 Linearly separating hyperplane with ***w*** *vector taken from: [7]*

Adding the above inequality to the expression for width we arrive at the conclusion that width is equal to $\frac{2}{\|W\|}$. Thus by maximizing this we will be able to find the optimal hyperplane. However for the sake of mathematical convince we choose to instead minimize ${}^{1}/_{2}\,\|\boldsymbol{w}\|^2$, this will become apparent in the following steps.

The minimization with respect to the constraints specified above is a Quadratic Programming problem and hence it is at this point that we introduce Lagrange Multipliers, in this case a positive multiplier α_i to the constraint $\mathcal{Y}_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) - 1 \ \geq 0$ resulting in the following.

$$L = \ {}^{1}/_{2}\,\|\boldsymbol{w}\|^2 - \sum \alpha_i[\mathcal{Y}_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) - 1\,]$$

The next step will be to differentiate the above equation with respect to **w** and b, and setting them to 0 to find the minimum.

$$\frac{\delta L}{\delta \boldsymbol{w}} = \ \boldsymbol{w} - \sum \alpha_i[\mathcal{Y}_i \boldsymbol{x_i}\,]$$

$$\frac{\delta L}{\delta b} = \ - \sum \alpha_i \mathcal{Y}_i \boldsymbol{x_i}$$

From the above two differentials we can conclude firstly a new expression for **w,** when $\frac{\delta L}{\delta \boldsymbol{w}}$ is equal to 0 (the support vector case) as follows.

$$\boldsymbol{w} = \sum \alpha_i [\mathcal{Y}_i \boldsymbol{x_i}\,]$$

Secondly we can conclude

$$\sum \alpha_i \mathcal{Y}_i \boldsymbol{x_i} = 0$$

Now taking the new expression for **w** and plugging it back into the Lagrange expression we arrive at the following.

$$L = \ ^1\!/_2 \left(\sum \alpha_i [\mathcal{Y}_i \boldsymbol{x_i}\,]\right)\left(\sum \alpha_j [\mathcal{Y}_j \boldsymbol{x_j}\,]\right) - \sum \alpha_i \mathcal{Y}_i \boldsymbol{x_i} \cdot \sum \alpha_j \mathcal{Y}_j \boldsymbol{x}_j - \sum \alpha_i \mathcal{Y}_i b + \sum \alpha_i$$

Simplifying this we get the following

$$L = \sum \alpha_i - \,^1\!/_2 \sum_i \sum_j \alpha_i \alpha_j \mathcal{Y}_i \mathcal{Y}_j \boldsymbol{x_i} \cdot \boldsymbol{x_j}$$

Hence from this equation we have found the dependencies with respect to the sample vectors, and we can see that the equation depends only on the dot product of pairs of samples. Now if we take this information and use it to update the decision rule we specified at the start we will arrive at the following inequality, the new decision function.

$$\sum \alpha_i \, \boldsymbol{x_i} \cdot u + b \ \geq 0 \ then\ positive\ example$$

And as above we see that the decision rule is also dependent of the dot product of the sample vectors and the unknown vector.

2.3. Nonlinear SVM and Kernel Functions

SVMs are very powerful and are commonly used in many modern applications, however most of the data which is processed by SVMs is nonlinear which proves a big problem since the methods described above will not be able to fit the decision boundary correctly. This said there is a simple and elegant solution to this problem which can be summed up in layman's terms as a change or perspective. By this we mean the process of using some function to map the input data from a nonlinear space into a new higher dimensional feature space where the patterns are linearly separable [10]. This process is visually represented by figure 3.

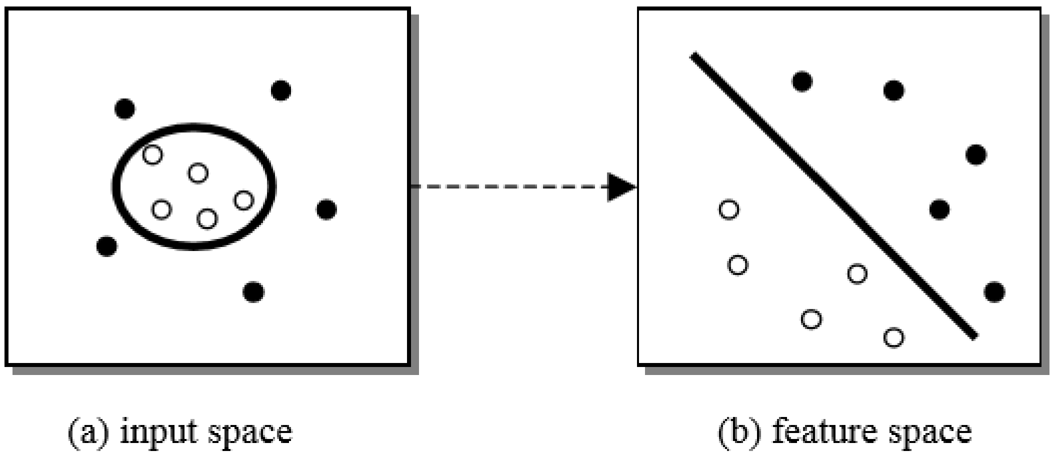

Figure 3 Transformation of feature space, taken from: [9]

Referring to the transformation function as $\phi(\boldsymbol{x}_i)$ and given, as was shown above, that the maximisation function is dependent on the dot product of the vectors then what really needs to be known is the dot product of the transformed vectors $\phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j)$ [7] [10]. Given the function

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j)$$

It becomes clear that the transformation $\phi(\boldsymbol{x}_i)$ is not needed, all is needed is the function *K* also known as a kernel function which will provide us with the dot product of the two vectors in the higher dimensional space.

There are many types of kernels which can be used the some of the more popular ones are listed below (these kernel examples have been taken from [7], [9] and [10]).

- Polynomial kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^T \cdot \boldsymbol{x}_j + 1)^d$
- Gaussian kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$
- Multi-layer perceptron $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\beta_0 \boldsymbol{x}_i^T \cdot \boldsymbol{x}_j + \beta_1)$

In figure 4 we can see the effects of the polynomial kernel and variations of the degree of the polynomial used. In the first case a linear kernel was used and as can be seen from the graph the decision boundary drawn is not a good fit, positive examples can be seen on the wrong side, such a decision boundary is likely to produce bad results when classifying unseen examples. The second

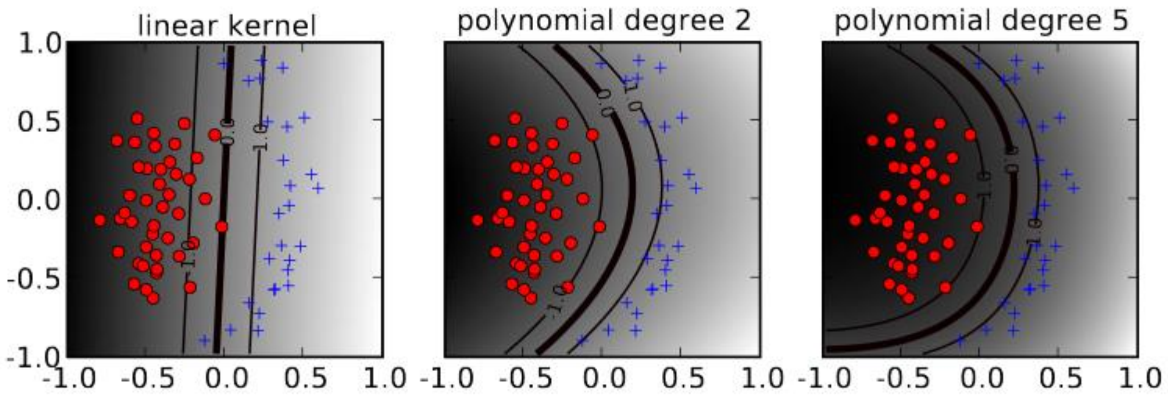

Figure 4 Effects of the degree of a polynomial kernel, taken from: [8]

case a polynomial kernel is used with a degree of 2, here we see that the decision boundary has a much better fit with no errors however there are still some positive examples which are close to the median line. In the final case we see a polynomial kernal with degree 5 this is very similar to that of degree 2 however the line is more curved and most of the positive exampes are close to their margin. [8]

In figure 5 observe the effects of training the SVM with the use of a Gaussiean kernel with different paremeters. In the first case we se a very small gamma (0.1), this causes any given point to have a non-zero kernel value in relation to any other training example as a result the whole set of training vectors will have an affect on the decision boundary causing it to be fery smooth, note that this is a case of under fitting. In the second pannel where gamma is equal to 1 in this case the result is that the decision boundary is curved tighter arround the examples which results in a better fit of the data and will result in better classificatons. The bottom two images are both cases when gamma is set to a large value, this has the affect of placing the decision boundary around those areas where the examples are located closely together, this is the case of overfitting. [8]

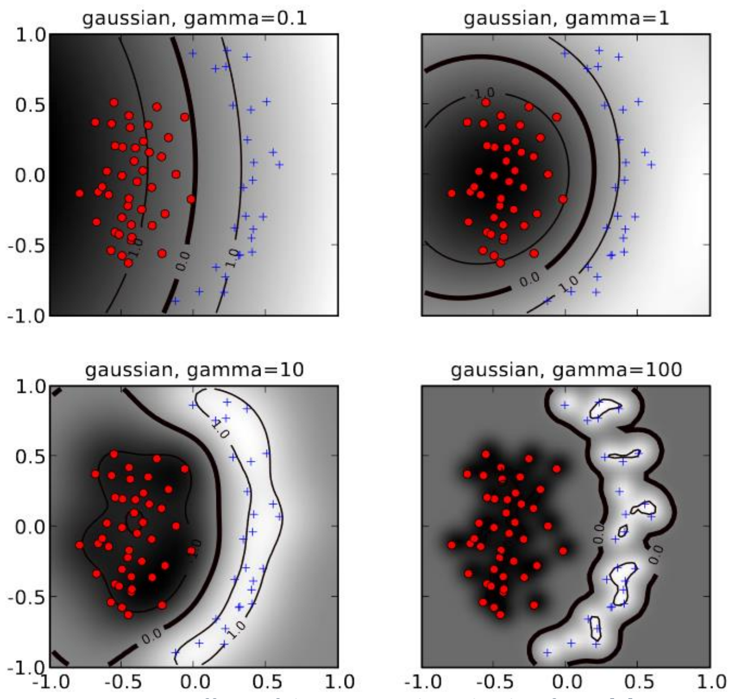

Figure 5 Effects of the Gaussian kernel, taken from: [8]

2.4. Multi-Class Classification

So far we have discussed cases for binary patterns only however there are a lot of cases which include multiple classes which we shall refer to as q-class problems. For such problems there are two methods to tackle multi-class problems such as these. [9]

The first method is one to others, in this approach for n classes n binary SVM will be built in each case the labels of one class chosen to be used and the rest used to represent the other class hence the problem is now a two class problem. [9]

The second technique is that of pairwise SVM. For this approach, given n classes, n^2 models will be trained. These models are then arranged in a tree, each node in the tree being a binary SVM which classifies two of the possible classes. There exist two approaches to constructing the trees, the first a top-down approach and the second a bottom-up approach. An example of a top-down tree can be seen in figure 6.

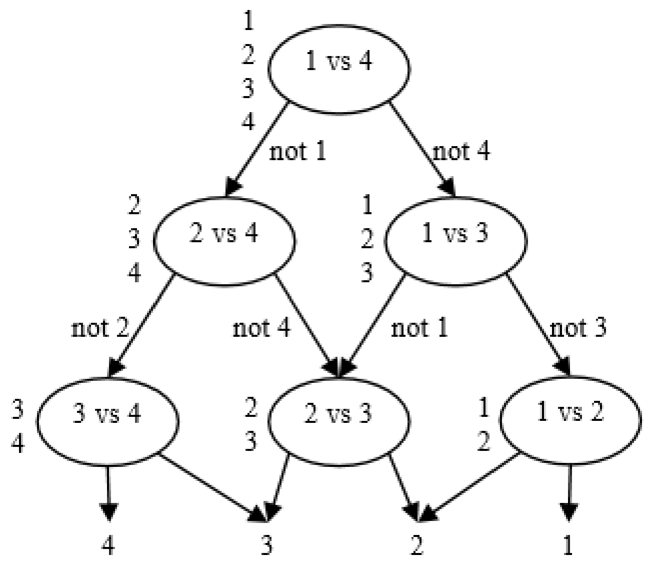

Figure 6 top-down tree structure, taken from: [9]

In this case the tree is a Directed Acyclic Graph, at each step the list of possible classes (seen on the left of each node) is reduced by one until the bottom of the tree is reached and a final classification is made. An example of the bottom-up approach can be seen in figure 7 below, as can be seen it is a binary tree, a data point to be classified will be compared with each of the bottom two pairs then the winner is then tested in an upper level, this continues until the top is reached. [9]

Note that the effort in regards to training pairwise trees is less preferable than training the one to others model, this is due to the fact that the former requires n^2 models to be trained and the latter only n models. In regards to making a classification at runtime both strategies have the same performance since they both need to n-1 SVMs.

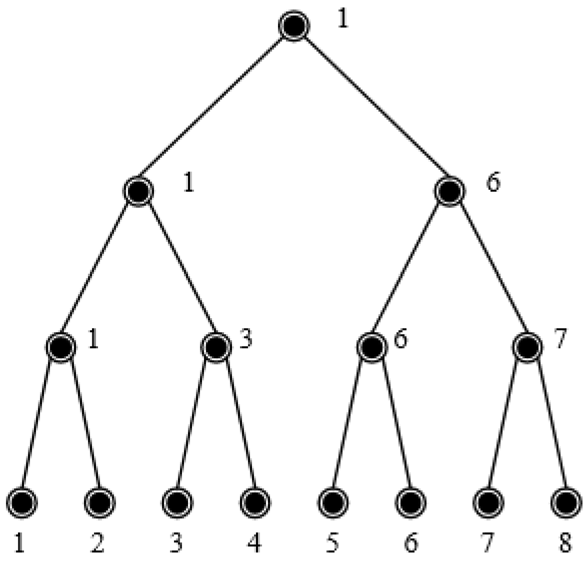

Figure 7 Example of bottom-up tree, taken from: [9]

2.5. Limitations of SVMs

Perhaps the biggest problem in regards to SVMs is that of choosing a kernel function, since there is no data driven approach to selection, it must be made based of experience and trial and error.

Due to the complexity of the algorithm it will require large amounts of memory and time to process large scale tasks.

Finally there is also the case of multiclass classifiers which in the best case require to train the same amount of models as there are classes.

3. Experiments

To investigate SVMs three data sets were chosen, described below. For the experiments we have chosen to investigate polynomial kernels and Gaussian kernels, experimental setup, implementation details and conclusions are discussed below.

3.1. Data sets

For these experiments we have chosen to use three data sets from LIBSVM [11]. The data sets chosen were the following:

- Diabetes
- Mushroom
- Iris

The first was chosen to represent a rather small learning problem with 2 classes, 768 examples and 8 features. [12]

The second was chosen because it represented a more complex binary learning problem with 8124 examples and 112 features. [12]

The final data set was chosen to represent a multi-class problem containing 3 classes, 150 examples and 4 features. Possibly one of the better known datasets in pattern recognition note that two of the classes are nonlinearly separable. [12]

3.2. Experimental methodology

The goal was given a data set to see whether a polynomial or Gaussian kernel would perform better. Hence for each data set the following models were trained:

- SVM with linear kernel
- SVM with polynomial kernel of degree 2
- SVM with polynomial kernel of degree 5
- SVM with Gaussian kernel with gamma set to auto (1/number of samples)
- SVM with Gaussian kernel with gamma set to 1
- SVM with Gaussian kernel with gamma set to 10

Before training the models the data will be split into two sets one for training and one for testing (60% and 40% of the data respectively). Once the data has been prepared and the models trained, they will be evaluated by calculating the accuracy, precision and F1 scores of the models.

3.3. Implementation Details

It was chosen to implement these experiments using Python and the SciKit Learn library, since it easy to use and only required a short script to automate the experiments.

3.4. Results

Results for Diabetes data set:

The data in this case was known to be linearly separable, hence we expected the linear kernel to perform the best. The results obtained which can be seen in the table below confirm this assumption obtaining a perfect F1 score of 1.0 along with the Gaussian kernel when gamma was set to 1. Note that during testing an error kept occurring which caused the values for the polynomial kernel set to degree 5 not being calculated, it is still unclear why this error is occurring and why it is only occurring in this case. Finally we also note that the Gaussian kernel when set to 10 performed the worst obtaining precision and F scores of 0, this clearly indicates that the training data is being over fitted with such a relatively high gamma value.

Kernel	Accuracy	Precision	F1 score
linear kernel	1.0	1.0	1.0
polynomial degree kernel= 2	0.984	0.993	0.983
polynomial degree kernel= 5	0.517	Na	Na
Gaussian kernel gamma = auto	0.996	1.0	0.996
Gaussian kernel gamma = 1	1.0	1.0	1.0
Gaussian kernel gamma = 10	0.517	0.0	0.0

Results for Mushroom Data set:

The patter in this case was not assumed to be linearly separable therefore we were not anticipating the linear kernel to produce the optimal result rather it to be provided by the polynomial or the Gaussian kernel. The results below corroborate this intuition, with the two best scores being produced by the degree 2 polynomial kernel and then followed by Gaussian kernel with automatic gamma.

We note that there is a drop-off in scores between the degree 2 and degree 5 kernels indicating that the optimal degree may lie somewhere between those two values. Also note that when gamma was set to 10, with respect to the Gaussian kernel we did not obtain radically different result as we did when it was set to 1, indicating that even though it was given a relatively large gamma it did not result in over fitting of the training examples.

Kernel	Accuracy	Precision	F1 score
linear kernel	0.772	0.799	0.845
polynomial degree kernel= 2	0.779	0.800	0.850
polynomial degree kernel= 5	0.691	0.691	0.817
Gaussian kernel gamma = auto	0.769	0.795	0.843
Gaussian kernel gamma = 1	0.772	0.820	0.839
Gaussian kernel gamma = 10	0.740	0.787	0.819

Result for Iris Data set:

Knowing that two on the classes in this data set were not linearly separable (described such in [12]) we expect polynomial or Gaussian kernels to produce the best result. Contrary to this intuition, as seen by the results below, we see that the linear kernel performed the best, this could possibly be due to the way that the library built the multiclass classifier. Also note that the Gaussian kernel with gamma set to 10 performed the second best overall, this was also counterintuitive as such a high gamma compared to the number of training examples would have caused overfitting of the data.

Kernel	Accuracy	Precision	F1 score
linear kernel	0.9666	0.9666	0.964
polynomial degree kernel= 2	0.833	0.872	0.801
polynomial degree kernel= 5	0.533	0.464	0.447
Gaussian kernel gamma = auto	0.933	0.930	0.930
Gaussian kernel gamma = 1	0.933	0.930	0.930
Gaussian kernel gamma = 10	0.95	0.94	0.947

4. Conclusion

In concluding we have discussed the theory behind SVMs and the mathematics that drive them. We have also experimented with two types of kernels over three different datasets, from these experiments we have concluded that kernel parameters can result in rather different level of accuracy in the classifications made. Even though there is no current data driven techniques or methods for kernel selection experimentation with different kernels can easily be automated by using python scripts. The running of such experiments may lead us to isolate one or two kernels which perform the best and possibly even a rough estimate of how to set their parameters. In following such experiments to improve the mode it would be best to run iterative tests on the model each time retraining it and incrementing the parameter by some arbitrary step with the aim to find the optimal setting for some parameter.

References

[1] A. Pradhan, “SUPPORT VECTOR MACHINE-A Survey,” *International Journal of Emerging Technology and Advanced Engineering,* vol. 2, no. 8, pp. 82-85, 2012.

[2] B. Boser, I. Guyon and V. Vapnik, “A training algorithm for optimal margin classifier,” in *In Proceedings of Fifth Annual Workshop on Computational Learning Theory*, New York, 1992.

[3] C. C. a. V. Vapnik, “Support vector networks,” *In Proceedings of Machine Learning,* vol. 20, pp. 273-297, 1995.

[4] V. Vapnik, The nature of statistical learning theory, Springer, 1995.

[5] MIT OpenCourseWare, “Learning: Support Vector Machines,” 10 January 2010. [Online]. Available: https://www.youtube.com/watch?v=_PwhiWxHK8o. [Accessed 12 2015].

[6] R. Brewick, “An Idiot's Guide to Support vector machines,” 2003. [Online]. Available: http://www.svms.org/tutorials/Berwick2003.pdf. [Accessed 20 12 2015].

[7] C. Burges, “A Tutorial on Support Vector Machines for Pattern Recognition,” *Data Min. Knowl. Discov.,* vol. 2, no. June 1998, pp. 121-167, 1998.

[8] A. Ben-hur and J. Weston, “A User’s Guide to Support Vector Machines,” in *Data Mining Techniques for the Life Sciences*, Humana Press, 2009, pp. 223-239.

[9] B. Hyeran and L. Seong-Whan, “Applications of Support Vector Machines for Pattern Recognition: A Survey,” in *Lecture Notes in Computer Science Volume 2388*, Berlin, Springer, 2002, pp. 213-236.

[10] C. Campbell, “Kernel Methods: A Survey of Current Techniques,” *Neurocomputing,* vol. 48, pp. 63--84, 2000.

[11] R.-E. Fan, “LIBSVM Data: Classification, Regression, and Multi-label,” National Taiwan University, [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/. [Accessed 12 2015].

[12] M. Lichman, “{UCI} Machine Learning Repository,” University of California, Irvine, School of Information and Computer Sciences, 2013. [Online]. Available: http://archive.ics.uci.edu/ml. [Accessed 12 2015].

[13] Wikipedia, “Support vector machine,” 9 1 2016. [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine. [Accessed 9 1 2016].