SVM Report

Group X

Jay Mehta
Julian Henjes
Sam Adejumo
Tom Staplehurst
James Thackway
James Armer
Gabrielle Hornshaw

Contents

Introduction

Support vector machines are models used for classification and regression problems. In this report we cover how we trained and evaluated models for both problem types, and how their accuracies compare to our previous models: neural networks and decision trees. In addition to this, we perform a more in-depth statistical analysis of each of the models to give a more comprehensive comparison between them. We also explain how we used inner-fold cross-validation to tune our model hyperparameters and evaluate our SVMs accurately. Finally, we discuss how the hyperparameters affect the model on a low level.

Inner-fold Cross-Validation

We performed a grid search to test a range of hyperparameters. We used cross fold validation to measure the performance of the different parameters and then returned the best parameters to be used for the model.

Classification

Initial Results

To test which hyper parameters to use for classification we initially tried a large set of arbitrary parameter values. We have highlighted the best results found for each.

Linear Kernel

Table 1: Results of Hyperparameter tuning of linear kernel

BoxConstraint	**1**	**50**	**100**	**150**	**200**
Avg. Accuracy	***0.8014***	0.7732	0.7747	0.7847	0.7687

Polynomial Kernel

Table 2: Results of Hyperparameter tuning of polynomial kernel showing accuracies

	BoxConstraint					
PolynomialOrder		**1**	**50**	**100**	**150**	**200**
	2	0.70239	0.72349	***0.72509***	***0.72509***	***0.72509***
	3	0.6726	0.6726	0.6726	0.6726	0.6726
	4	0.51478	0.51478	0.51478	0.51478	0.51478

RBF Kernel

Table 3: Results of Hyperparameter tuning of RBF kernel showing accuracies

	BoxConstraint					
KernelScale		**1**	**50**	**100**	**150**	**200**
	1	0.4771	0.477	0.4767	0.4767	0.4767
	30	0.7805	0.82771	0.81931	0.8162	0.81711
	60	0.7953	0.8192	0.8268	0.8369	***0.8457***

Continuous Improvements

After we found that the RBF kernel gave us the highest accuracy for the given parameters, we worked on refining our search by testing granularity around those parameters to tune them more accurately. Table 4 shows the best final parameters we found in our final grid search after 2 more iterations of refinement.

Table 4: Further Hyperparameter tuning after a general grid search showing accuracies

	BoxConstraint			
KernelScale		**183**	**186**	**189**
	40	0.8479	0.8477	0.8475
	43	0.8537	0.854	0.8539
	46	0.8576	0.8578	0.8576
	49	0.8611	***0.8615***	0.861

This shows that we found our best results using the RBF kernel with KernelScale set as 49 and BoxConstraint as 186.

Regression

Similarly, we tuned hyperparameters for the Regression SVM using a similar methodology as above except using Root Mean Squared error as a loss function. A grid search was again used to find optimal values for each kernel. However with Regression, there was a further addition of Epsilon as a hyperparameter to tune, adding an extra level of dimensionality. Using 10,000 features and inner fold validation, the following hyperparameter values were found:

Table 5: Hyperparameter tuning results for the regression SVM problem. Invalid or inconsequential hyperparamters are omitted.

Hyperparameter	**Kernel**		
	Linear	Polynomial	RBF
BoxConstraint	1	1	1
Epsilon	0.15	0.35	0.05
KernelScale			4.0
PolynomialOrder		2	

BoxConstraint

A range of different values from 1 to 200 were used in the grid search, with 1 proving to best for all kernels. The general trend of the results is similar to that of classification shown above in Table 1, with BoxConstraint having a notable effect on both Linear and RBF kernels but having little no effect on the performance on the Polynomial kernel.

Table 6: The effect of changing BoxConstraint on the Linear kernel using a constant epsilon value.

BoxConstraint	1	25	50	75	100	150	200
RMSE	0.42	0.43	0.44	0.49	0.52	0.57	0.48

Epsilon

Epsilon is essential because controls the model's tolerance to errors while training, in part, this determines the number of support vectors used. If too low, there will too many support vectors resulting in the model becoming overfit, if too high, there may not be enough support vectors, overgeneralising the model making it underfit. The optimum values for each kernel can be seen above in table 5.

PolynomialOrder

The polynomial order determines the shape of the hyperplane used in the model for the polynomial kernel. The results we obtained clearly showed that a value of 2 would be best for this regression problem.

Table 7: The effect of changing PolynomialOrder on the Polynomial kernel using constant values for BoxConstraint and Epsilon.

PolynomialOrder	2	3	4
RMSE	0.65	2.39	11.54

Performance Comparison

Classification

After returning the best hyperparameters for each kernel, we then compared them to each other and to our Decision Tree. We trained the three SVMs on 30,000 rows of data for this analysis, an increase from the 10,000 used for the hyper-parameter tuning.

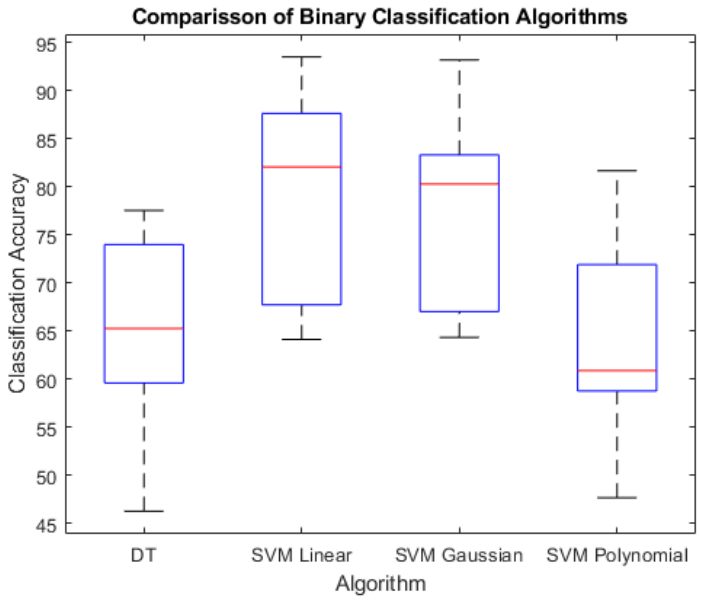

Figure 1: Comparison of different classification methods

Figure 1 shows the mean average accuracy of each type of algorithm in red. It also shows the upper and lower quartile accuracies, and the highest and lowest accuracies based on 10-fold cross validation.

Once we received the best hyperparameters to use we then tested these with significantly more data. We found that the SVM Linear outperformed the Gaussian classifier, which differs from the results obtained during hyperparameter tuning.

Support Vectors

Table 8: Number of support vectors used by each model.

Classifier	**Average No. Support Vectors**
SVM Linear	8493
SVM Gaussian	8370
SVM Polynomial	3010

The linear and Gaussian models have a very similar number of support vectors, whereas the polynomial model has significantly less. This suggests that a higher number of support vectors correlates to a higher accuracy as the linear and Gaussian SVMs produced better results than the polynomial. In terms of percentages of the data, around 28% were selected to be support vectors for Linear and Gaussian, whereas only around 10% were selected for Polynomial. Therefore, if we ran our models on more or less

data, we would expect to see around the same percentage of support vectors for the Linear and Gaussian SVMs to achieve a high accuracy.

F1-Score

From our decision tree report, we concluded that for a binary classification problem of identifying facial features, we favour neither precision or recall due to the cost of a false positive or false negative being fairly even. Therefore in our results we looked for a high F1-Score to go along with any high accuracies we attained, to show that the model was classifying the data set well.

Table 9: F1-Score comparison between our Decision Tree and the three different kernels

Model	**Average F1-Score**
Decision Tree	0.629
SVM Linear	0.745
SVM Gaussian	0.735
SVM Polynomial	0.611

The linear kernel gave us the highest accuracy and as we can see from the above table it also had the highest F1-Score, showing us that it classified the data the best. The Polynomial kernel had the lowest accuracy and the lowest F1-Score, showing us that it was the worst at classifying this data set.

T-Test Results

The null hypothesis is that both sets of distributions (set of 10-fold results) have the same mean for any given comparison. Having the same mean implies the models are equal and produce the same results.

```
2-sample t-test between Decision Tree and Polynomial SVM.
H = 0. P = 0.94283.

2-sample t-test between Linear SVM and Gaussian SVM.
H = 0. P = 0.84294.
```

Of the six comparisons between decision trees, linear SVM, Gaussian SVM, and polynomial SVM, only two of them accepted this hypothesis at the 5% significance level. The results show the models can be divided into two groups: the decision tree and polynomial SVM, and the Gaussian and Linear SVMs.

From the difference in means between the two groups we can draw the conclusion that using a Linear or Gaussian SVM would be best suited to this problem, but neither is a significantly better choice.

Regression

Using the same methodology as above, we then ran 10-fold validation using the hyperparameter results obtained for each kernel using a 30,000 rows of data. As this is a regression problem, we compared these results to those of our Artificial Neural Network we previously created.

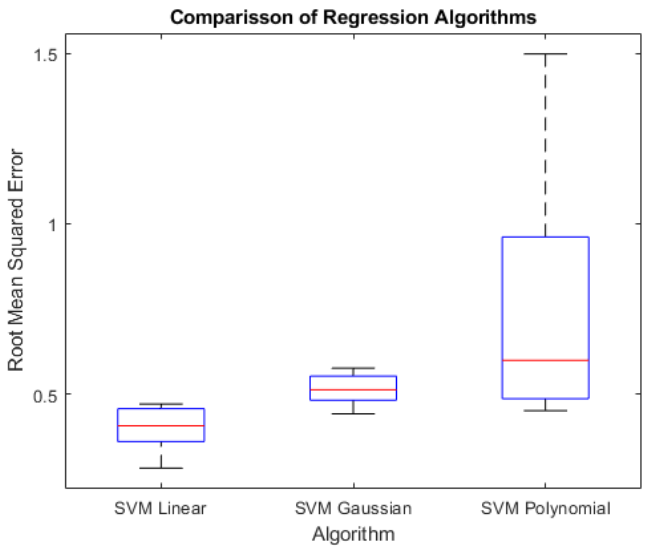

Figure 2: Comparison of different SVM models on regression problems.

Table 10: Results of the different SVMs and ANN

Model	**Average RMSE**
SVM Linear	0.5106
SMV Gaussian	0.45908
SVM Polynomial	0.7491
ANN	15.97378

The RMSE results we obtained from the ANN are significantly higher due to anomalies in the network, and therefore this may not give an accurate comparison for the three SVM models.

Support Vectors

Table 11: Number of support vectors used by each model.

Model	**Average No. Support Vectors**
SVM Linear	18155
SMV Gaussian	20038
SVM Polynomial	5386

Similarly to the classification results, the linear and Gaussian SVMs used about the same number of support vectors. The Polynomial SVM again used significantly less, and produced a significantly higher average RMSE, further proving that higher amounts of support vectors correspond to better results.

T-Test Results

All of the t-tests rejected the null hypothesis, which means none of the models perform similarly on regression problems. SVM linear produced the lowest RMSE and is the best choice.

Kernel Parameter of the Gaussian RBF Kernel

Sigma, in the case of a Gaussian distribution, represents the standard deviation and determines the width of the distribution. Put into the context of a Support Vector Machine, sigma affects the decision boundary qualitatively, and is an amplification factor of the distance between points. In Matlab the KernelScale parameter is directly correlated with the sigma of the SVM. When sigma is small the kernel becomes more of a local classifier and stricter on what is classified, which can often lead to overfitting due to less space for data points of the same class but having different values. Vice versa when sigma increases the decision boundary becomes less strict and more flexible on what it is able to classify. In contrast to the former situation, the decision boundary may avoid overfitting at the cost of misclassification.

Hard-Margin Support Vector Machines

To create a hard-margin SVM the value of C must be very high (as C tends towards infinity, the harder the margin becomes). This allows for less slack-variables to be greater than 1 (misclassified) as they are penalized with a greater cost. When finding a hard-margin of an SVM, we want to find a hyperplane with the largest margin which is able to separate all the observations correctly. To do this we need to satisfy the constraint of no misclassification errors while optimising the margin. When you train an SVM with a hard-margin on a data set that has overlapping features, it becomes impossible to find a hyperplane that does not misclassify the data. This is because the hard-margin doesn't allow for any data points on the incorrect side of the hyperplane, due to the cost of the incorrectly classified slack variables being very high.

Effects of Sub-Sampling the Data before a Split

Assuming the data is raw and not completely shuffled, then different sections may present differently. In terms of facial points this means there would be a huge range of coordinates resulting in a single classification, but the first 10,000 rows of data may only give faces in a single location (for example, if the images are frames of a video), while the whole dataset gives more variation**.** If the data is subsampled like this before training then it will be overfit to that subsample and be less able to handle outliers, or images with slightly different context. If the data is subsampled after splitting the whole dataset into the 10 folds then this effect is minimised, as the data being used will give a wider range of contexts and the model will be more robust to facial points in different areas of the image.