Assignment 1 – Report

My Understanding

Support Vector Machines or SVMs are a supervised machine learning algorithm. SVMs are mostly used in the classification of datasets. This report will be dealing with the case of binary class linear SVMs, where there will be only two classifications that the data points can fall under.

The principle idea of SVMs is to draw a linear decision boundary between the two separate classes. When new data is placed into the model, it is classified on its location relative to the decision boundary.

The dataset used in SVM consists of n samples. Each of the n samples are made up of a couple containing $\boldsymbol{x_i}$ and y_i. Where $\boldsymbol{x_i}$ is a p-dimensional vector (p being the number of features measured for each sample) and y_i is the class indicator, which will have a value of either -1 or 1. The dataset can therefore be expressed via:

$$\mathcal{D} = \{ (\boldsymbol{x_i}, y_i) \mid \boldsymbol{x_i} \in \mathbb{R}^p, y_i \in \{-1, 1\} \}_{i=1}^{n}$$

The data set is divided by a hyperplane, which separates the two classes from one another. Any points that lie on the hyperplane will satisfy:

$$\boldsymbol{w^T x} + b = 0$$

Where $\boldsymbol{w}$ is a normal to the hyperplane and b is a bias value.

The margin of the decision boundary can be found to be $d_+ + d_-$, where d_+ is the minimum distance between a point classified as +1 and the hyperplane and where d_- is the minimum distance between the hyperplane and a point classified as -1.

The hyperplane on the edge of the margins, on both the positive and negative classification sides have been normalised to be the following:

$$\text{For the +ve side: } \boldsymbol{w}^T \boldsymbol{x_i} + b \geq 1 \quad \text{for } y_i = 1$$
$$\text{For the -ve side: } \boldsymbol{w}^T \boldsymbol{x_i} + b \leq -1 \quad \text{for } y_i = -1$$

These two equations can be combined into one inequality:

$$y_i[\boldsymbol{w}^T \boldsymbol{x}_i + b] - 1 \geq 0 \quad \forall i$$

In the case where the data is linearly separable (that is, a line can be drawn that classifies the data with 100% accuracy) the maximum sized margin possible will be ideal.

The margin, which can be found to be $\frac{2}{||w||}$ can be maximised by minimising $||w||$.

Support vectors are training points that sit on either one of the two outer hyperplanes at the edges of the margin. These vectors define the position of the hyperplanes and if moved, will affect the position of the margins, thus giving them the name support vectors.

Data points can be identified as support vectors as they will be the only vectors with alpha values greater than zero.

The primary objective of the optimisation in the SVM is to maximise the width of the margin. This serves the purpose of decreasing the possibility of an incorrect classification. This principle is rather intuitive as a larger margin would give a larger 'buffer' between the two classes and thus decrease the chances of any classification errors from occurring.

For this reason, the margin is optimised using gradient descent, to be as large as possible.

A linearly non-separable case contains some data points from either class that is mixed among one another. In this case it is not possible to separate the classes with 100% accuracy using only a straight line. A soft margin can be utilised in this situation to reduce the amount of misclassified data points.

The hard margin options for primal and dual forms are used mostly in linearly separable cases. This is because there are solutions possible that do not need to account for incorrect classifications during training. Whereas the soft margin allows for linearly inseparable cases to be classified with some level of error allowance.

The primal form of the hard margin is derived from the idea that we want to maximise the margin width, which is $\frac{2}{||w||}$. In order to do so, we must minimise $\left||w|\right|$. However, the constraints shown in the previous section still stand.

So, the primal form of hard margin SVM looks like:

$$\min_{w} \frac{1}{2}\left||w|\right|^2 \; s.t. \;\; y_i[\boldsymbol{w}^T\boldsymbol{x}_i + b] \geq 1 \;\; \forall i$$

The dual form of the hard margin can then be found by taking the Lagrangian formulation of the problem above. This will introduce the Lagrangian multipliers, one for each inequality restraint.

The Lagrangian will look like:

$$\frac{1}{2}\left||\boldsymbol{w}|\right|^2 - \sum_{i=1}^{l} \alpha_i y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i \; + b\right) + \; \sum_{i=1}^{l} \alpha_i$$

And then by substituting in the equality constraints of

$$\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x_i} \text{ and } \sum_i \alpha_i y_i = 0.$$

We find the dual form:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i}^T \boldsymbol{x_j} \quad s.t. \;\; \sum_i \alpha_i y_i = 0 \; and \; \alpha_i \geq 0 \; \forall i$$

In the case of linear inseparability, where there is no single hyperplane that can separate the data points with 100% accuracy, both the primal and the dual forms are slightly modified to produce the soft margin.

To adapt the primal form of the optimization problem to inseparable classes, slack variables, ξ_i are added. The slack variables will add a penalty to the function for each datapoint that is misclassified. The slack variables are added to the function in the form of the sum of all the slack values and that total is multiplied by the regularisation parameter, C.

Giving the minimisation problem of:

- The Primal form:

$$\min_{\boldsymbol{w},b,\xi} \frac{1}{2}\left|\left|\boldsymbol{w}\right|\right|^2 + C \cdot \frac{1}{n}\sum_{i=1}^{n} \xi_i$$

$$\boldsymbol{s.t.}\ \ y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 - \xi_i$$

$$\xi \geq 0, \qquad i = 1, \dots, n$$

- And the Dual form:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum\nolimits_{i,j} \alpha_i\alpha_j y_i y_j \boldsymbol{x_i^T x_j}$$

$$\boldsymbol{s.t.}\ \ \textstyle\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \qquad i = 1, \dots, n$$

The larger the regularisation parameter is, the larger the penalty for errors will be.

Experiments

Built in SVM Classifier

First the built-in SVM within MATLAB was run. Cross validation was used to help build an accurate model that would fit a whole range of data. This model was trained using the data found in 'train.mat' file and partitioned using 'cvpartition', the built-in MATLAB cross partitioning function.

By using a K-Fold of 4, four different models, each with a different w vector and bias, b were found. To test each model, they were given the data from 'test.mat' and the predicted classification from each observation was compared to the actual classification given in the class table.

The total correct predictions and the resulting accuracy are also displayed in the table below.

Model	Bias Value	Correct Predictions*	Accuracy
1st Fold	2.4544	1,444	96.27%
2nd Fold	2.4500	1,453	96.87%
3rd Fold	2.1812	1,455	97.00%
4th Fold	2.3476	1,452	96.80%

*Out of 1500

Looking at the table, we can see that the built-in SVM classifier scored an average accuracy of 96.74%.

When the same function was used to train a model on all the training data within 'train.mat', the model scores an accuracy of 96.8%.

For the next parts of the report, the models will be tested by using a 4-fold cross validation on the data found in 'train.mat'. This serves the purpose of not only ensuring that the model works with a multitude of different inputs but also speeding up the runtime of the functions.

Finding Optimal Hyperparameters

In this section, the optimal values for the hyperparameter, C will be found. This will be done by trialling a selection of different values ranging from 0 to 100. Cross-validation will be utilized to ensure that the results are relevant to several sets training data and not just a single specific set. K-Fold cross validation will be used on the set of 8,500 observations from the file 'train.mat' and 4-fold has been chosen to speed up processing time and allow for a larger testing set.

This process will be repeated for both the Primal Form function and the Dual Form function, before both sets of data will be average and the resulting optimal hyperparameter will be used in the same tests as MATLABs built-in function was in the previous section.

Primal Form Function

Hyper-parameter C Value	Fold 1		Fold 2		Fold 3		Fold 4	
	Correct Predictions	**Accuracy**	**Correct Predictions**	**Accuracy**	**Correct Predictions**	**Accuracy**	**Correct Predictions**	**Accuracy**
0	1,043	49.08%	1,056	49.69%	1,071	50.40%	1,079	50.78%
0.1	2,115	99.53%	2,116	99.58%	2,119	99.72%	2,118	99.67%
0.5	2,109	99.25%	2,104	99.01%	2,110	99.29%	2,109	99.25%
1	2,108	99.20%	2,102	98.91%	2,108	99.20%	2,102	98.92%
5	2,104	99.01%	2,103	98.96%	2,103	98.96%	2,100	98.82%
10	2,098	98.73%	2,100	98.82%	2,101	98.87%	2,102	98.92%
25	2,097	98.68%	2,100	98.82%	2,098	98.73%	2,098	98.73%
50	2,097	98.68%	2,095	98.59%	2,094	98.54%	2,093	98.49%
75	2,099	98.78%	2,092	98.45%	2,092	98.45%	2,093	98.49%
100	2,098	98.73%	2,092	98.45%	2,090	98.35%	2,093	98.49%

Dual Form Function

Hyper-parameter C Value	Fold 1		Fold 2		Fold 3		Fold 4	
	Correct Predictions	**Accuracy**	**Correct Predictions**	**Accuracy**	**Correct Predictions**	**Accuracy**	**Correct Predictions**	**Accuracy**
0	1,043	49.08%	1,056	49.69%	1,053	49.56%	1,048	49.32%
0.1	2,110	99.29%	2,112	99.39%	2,113	99.44%	2,111	99.34%
0.5	2,109	99.25%	2,104	99.01%	2,109	99.25%	2,109	99.25%
1	2,108	99.20%	2,105	99.06%	2,110	99.29%	2,103	98.96%
5	2,105	99.06%	2,104	99.01%	2,104	99.01%	2,100	98.82%
10	2,100	98.82%	2,102	98.92%	2,101	98.87%	2,102	98.92%
25	2,098	98.73%	2,101	98.87%	2,100	98.82%	2,099	98.78%
50	2,100	98.82%	2,096	98.64%	2,094	98.54%	2,096	98.64%
75	2,100	98.82%	2,096	98.64%	2,094	98.54%	2,096	98.64%
100	2,097	98.68%	2,094	98.54%	2,092	98.45%	2,094	98.54%

Below is a table showing the average accuracies for each of the trialled C values for both primal form and dual form of the SVM model. The best of the results has been boldened to stand out.

Hyper-parameter C Value	Primal Form	Dual Form
0	49.99%	49.41%
0.1	**99.63%**	**99.37%**
0.5	99.20%	99.19%
1	99.06%	99.13%
5	98.94%	98.98%
10	98.84%	98.88%
25	98.74%	98.80%
50	98.58%	98.66%
75	98.54%	98.66%
100	98.51%	98.55%

As shown above for both forms of SVM training, a hyperparameter of 0.1 proved to provide the best accuracies, both scoring above 99%. Next these values will be used to train two new models using all 8500 observations and then be tested using the set of testing data found in 'test.mat' and compared to the initial model made using the built-in function.

Model	Bias Value	Correct Predictions*	Accuracy
Average Built-In	2.4544	1,444	96.74%
Primal ($C = 0.1$)	0.3848	1,496	99.73%
Dual ($C = 0.1$)	0.3119	1,490	99.53%

*Out of 1500

As can be seen in the table above, both models produced by the functions I created appear to outperform the model that was built by MATLABS SVM function. This could be caused by the method of generating the model used by MATLAB (SMO) being different to that of mine (L1 Quadratic programming).

Another factor that may affect the results having such a difference is that the built-in model was created using a very minimal number of input parameters. By adding some specifications regarding the hyperparameter optimization and others, the 96.74% result could be increased.

Overall, I am quite pleased with the results of my SVM models. I would like to see how it performs with new sets of data and see how it fares with less features.