Assignment 1
COMP SCI 7209 - Introduction to Statistical Machine Learning
Support Vector Machine implementation

Julian Cabezas Pena
Student ID: a1785086
University of Adelaide, SA 5005 Australia
julian.cabezaspena@student.adelaide.edu.au

1. Introduction

In the field of machine learning and pattern recognition, classification problems are one of the most common tasks. In these kind of problems, an algorithm takes a vector X as input, and assigns it to one or many K discrete classes [1]. Considering that the labelled data take values in a discrete set, the input space can be divided in regions that can be labelled according to the classification. Boundaries of these regions can be smooth or rough, depending on the function applied by the algorithm [2]. When lineal functions are used to estimate these decision boundaries or surfaces, these algorithms can be labelled as part of the "linear models for classification" family, while data sets whose classes can be separated by lineal functions, are often called to be lineally separable [1].

One of the most common algorithms used in classification problems is the Support Vector Machine (SVM). This supervised classification algorithm was developed by Vapnik [3], and uses an hyperplane that separates a high dimensional space defined by the input variables into discrete classes. This hyperplane is defined to have the largest possible distances to the closest point of either class [2], thus, maximising the margin (M) between two classes. This algorithm can define an hyperplane even when classes overlap (non linearly separable), maximizing the margin but allowing some points to be in the wrong side of the boundary.

The objective of this project is to implement the Support Vector Machine using using convex optimization to solve the primal and dual forms of the algorithm. The implemented methods will be compared with the implementation of the Support Vector Machine Classifier implemented in the commonly used Scikit-learn package, using a simulated dataset.

2. Methods

2.1. Maximal margin classifier

The margin can be interpreted as the distance of a hyperplane and the closest data points, in the field of machine learning, a hyperplane that can separate two or more classes, maximising the margin (or the distance to the closest points) is called a maximal margin classier (Figure 1). This concept is used in many classification algorithms, that seek to maximise this distance. The advantage of this maximal margin classification is that it can produce a great level of generalisation, easily transferring the learning in the training data to the testing data.

Another big advantage of the maximum margin concept is that it can be easily generalisable to multiple dimensions, allowing to generate multidimensional hyperplanes, the maximum margin classifier can also be applied to transformed data, allowing us to work with kernels [1].

2.2. Hard Margin SVM

When implementing the SVM, we can assume we count with n points of data corresponding to pairs of m independent variables x and dependent variable y, as $(x_1, y_1), (x_2, y_2), (x_3, y_3)...(x_n, y_n)$, were x belong to the real realm and y is a vector of true labels coded like $-1, 1$ (binary class).

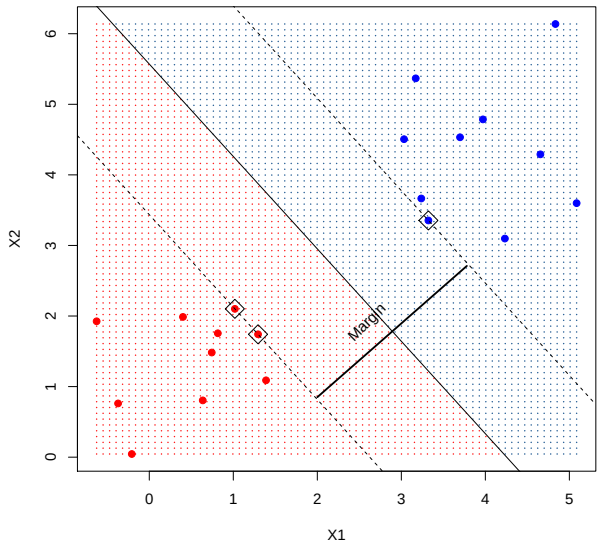

Figure 1. Margin concept - Hard margin SVM (Support vector are inside squares)

The objective of the SVM is to find an hyperplane defined by the coefficients w and the intercept b:

$$f(x) = w^T x + b \tag{1}$$

That should give $y_i f(x_i) > 0 \; \forall \; i$, meaning all the samples are well classified, using the sign function as classification rule:

$$C(x) = sign(w^T x_i + b) \tag{2}$$

This constrains are only able to be followed if the dataset is linearly separable, meaning a hyperplane that completely separates the two classes can be found. Hence, the optimization problem is to find the biggest margin M between classes:

$$max_{w,b} \, M \tag{3}$$

$$subject \;\; to \;\; y_1(w^T x_i + b) \geq M \tag{4}$$

2.2.1 Primal form

As the margin can be understood as the distance of the closest points to the hyperplane $(1/||w||)$ towards both sides [2] (with $||w||$ being the L2 norm), it is possible to conveniently redefine this equation as a minimization problem, as follows:

$$min_{w,b} \frac{1}{2}||w||^2 \tag{5}$$

$$subject \;\; to \;\; y_i(w^T x_i + b) \geq 1 \tag{6}$$

This formulation is called the **primal** form of the problem, and it is considered a convex optimization problem. The Lagrange form (where α are the lagrange multipliers) of this equation can be also written as [1]:

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1) \tag{7}$$

2.2.2 Dual Form

If we consider the partial derivatives of w and b in the primal form and set them to be equal to zero, we can get the following conditions:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{8}$$

$$0 = \sum_{i=1}^{n} \alpha_i y_i \tag{9}$$

If we eliminate the b and w terms of the primal form equations it is possible to obtain the Lagrangian function [1]:

$$L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{10}$$

For computation purposes, it is convenient to express that equation as:

$$min_{(\alpha)} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{n} \alpha_i \tag{11}$$

with constrains:

$$\alpha_i \geq 0 \;\; \forall i \tag{12}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{13}$$

2.3. Soft Margin SVM

In real-world data, the data is often not linearly separable, so it is convenient to "allow" the classifier to consider that some points will not be classified correctly, for that, the slack variables $\xi = (\xi_1, \xi_2, \xi_3 ... \xi_n)$ are considered, such that we continue to maximize the margin, but we permit some flexibility, that allows some points to be misclassified, as shown in Figure 2:

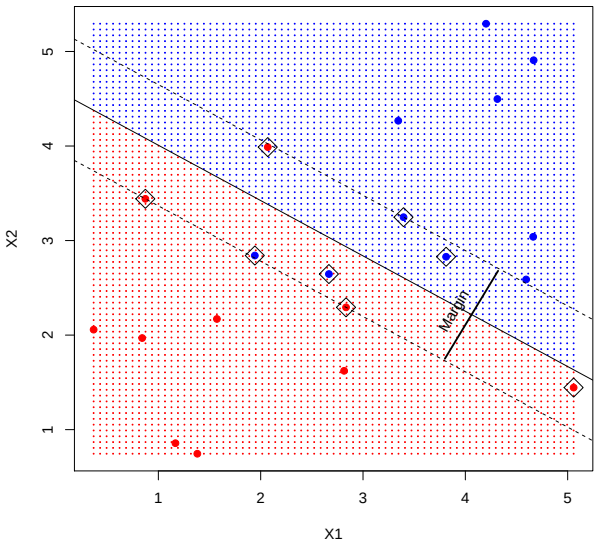

Figure 2. Margin concept - Hard margin SVM (Support vector are inside squares)

$$y_i(w^T x_i + b) \geq M - \xi_i \tag{14}$$

2.3.1 Primal form

By including this new term, it is possible to write the optimization function as:

$$min \frac{1}{2}||w||^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \tag{15}$$

with constrains:

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i \tag{16}$$

$$\xi_i \geq 0 \quad \forall i \tag{17}$$

The term C (cost) here is a regularization parameter, that determines the weights of the misclassified samples, with small cost more samples are allowed to be inside de margin boundaries, while a big C will be penalizing those samples harder, this would mean that an infinite C would create a hard margin classifier.

This equation can be also expressed using the Lagrange form

$$L(w, b, \alpha) = \frac{1}{2}||w||^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$
$$\sum_{i=1}^{n}\alpha_i(y_i(w^T x_i + b) - (1 - \xi_i)) - \sum_{i=1}^{n}\mu_i\xi_i \tag{18}$$

2.3.2 Dual form

In order to get a more convenient optimizing function, it is possible to set the partial derivatives os w, b and ξ to zero [2], getting:

$$w = \sum_{i=1}^{n}\alpha_i y_i x_i \tag{19}$$

$$0 = \sum_{i=1}^{n}\alpha_i y_i \tag{20}$$

$$\alpha_i = C - \mu_i \quad \forall i \tag{21}$$

Obtaining the Lagrange function:

$$L(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^T x_j \tag{22}$$

In order to have a minimizable, it is possible to write the optimization function as:

$$min_{(\alpha)} \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^T x_j - \sum_{i=1}^{n}\alpha_i \tag{23}$$

with constrains (where μ is a constant)

$$0 \leq \alpha_i \leq \frac{C}{n} \quad \forall i \tag{24}$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0 \tag{25}$$

2.3.3 The concept of support vectors

When optimizing these Lagrange functions, typically many of the α values are equal to zero [2], as these points are far away from the separating hyperplane. Thus, the hyperplane is defined by a subset of the the data points, that are called "support vectors", that are shown in Figures 1 and 2.

As the support vectors are close to the hyperplane (Figures 1 and 2), if one of them is removed, the position of the hyperplane would change, thus, if a data point that is not a support vector is removed, the hyperplane would not change. The amount of support vectors that are included in soft margin SVM is influenced by the cost (C) parameter, if the C value is bigger, the misclassification would be penalized hardly in the model, making the margin smaller and considering fewer support vectors.

2.4. Third Party Implementation: *Scikit-learn*

In order to compare the implemented method with more mainstream application of the Support Vector Machine Classifier, a third party library was used to solve the classification problem. In this case, the Scikit-learn library was chosen due to its wide popularity in the machine learning community.

Scikit-learn is a Python library that integrates various tools for statistics and machine learning applications, that include classification, regression, metrics and feature selection, among many others. This library is distributed under a BSD licence and includes compiled code, that makes it very efficient. The library is built using other popular numerical Python libraries, such as *Numpy* and *Scipy* [4].

According to the library documentation [1], the support vector machine algorithms that are implemented in this package can be used for classification, regression and outlier detection. Among the advantages that are listed for this method are its effectiveness in high dimensional spaces, memory efficiency and its capability to use different kernel functions.

The mathematical formulation of the SVM in the scikit-learn package [2] follows a similar approach to what was previously explained, with the minimization function in the primal form being:

$$min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i \tag{26}$$

with constrains:

$$y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i \tag{27}$$

$$\xi_i \geq 0 \quad \forall i \tag{28}$$

As the SVM implementation of this library can work with several kernel functions. The constraint includes the ϕ, transformation, that can take the form of a linear, radial, polynomial or sigmoid function. As the focus of this project is to compare this implementation with the primal and dual forms explained above, the function to be used is the linear function, meaning that $w^T \phi(x_i)$ is equal to $w^T x_i$.

Additionally, as in the minimization function $w^T w$ can be interpreted as $||w||^2$, it is possible to rewrite the equation as

$$min \frac{1}{2} ||w||^2 + \frac{C}{\sum}^{n}_{i=1} \xi_i \tag{29}$$

with constrains:

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i \tag{30}$$

$$\xi_i \geq 0 \quad \forall i \tag{31}$$

Thus, the only difference between the implementation described in section 2.2.1 and the one implemented in Scikit-learn is the form of the regularization parameter C, that in this case is not divided by the number of samples (C/n).

The optimization problem in this library is solved using the dual form of the problem, that also has the same difference regarding the cost parameter. Given that fact, to compare the two methods, the cost parameter was divided by the number of samples (C/n).

2.5. Train and testing

2.5.1 Databases

The database in which the implemented methods will be tested corresponds to two datasets. In one hand, the train dataset counts with the 8500 records of data and

[1] https://scikit-learn.org/stable/modules/svm.html

[2] https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation

200 features, with a target variable that contains two classes, that were encoded as $-1, 1$. Additionally, the test dataset has 1500 samples, and the same amount of features (200). The database is not linearly separable, thus the soft margin implementation of the SVN (both in its primal and dual forms) were used to train the model in the train dataset and get predictions in the test data.

2.5.2 Cross validation - C Parameter tuning

As the C (cost) parameter can have a critical importance on the performance of the SVN model, a 5-fold cross validation on the train data was used to select the best C value in $C = (1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100)$. To perform this, the train database was divided into 5 folds (subsets of the database) with equal number of samples, for each value of C that was tested, 4 of these folds were used to train the model, and then this model was used to predict the target feature in the remaining fold, to then calculate the accuracy as:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{32}$$

Where TP is the number of true positive, TP: true negative, FP: false positives and FN: false negatives.

This process was repeated 5 times for each C value, and then the accuracy was averaged. The best C parameter was selected using the maximum value of mean accuracy.

Giving the above mentioned differences in the Scikit-learn C parameter, the cost values that were tested in this case were divided by the number of samples in the model (C/n) for comparison purposes.

2.5.3 Model testing

Once the C parameter was selected for the primal, dual and Scikit-learn implementation of the SVN, the best C parameter was used to train the model using the complete train dataset, then, this model was used to predict the target variable in the test dataset, measuring the archived accuracy.

2.6. Code

The code to reproduce this project was attached to this report. The primal and dual implementations of the models were programmed in Python 3.7 using the CVXOPT tool for convex optimization, and followed the API directions of this tool for the solving of the problem [3]:

$$min \frac{1}{2} x^T P x + q^T \tag{33}$$

subject to

$$Gx \leq b \tag{34}$$

$$Ax = b \tag{35}$$

Where x corresponds to the values to be optimized, that in the primal form are the vector $[w, b, \xi]$ and in the dual form $[\alpha]$. The matrices P,q,G,b,A and b were constructed to match the dual and primal minimization problems.

As the cross validation to select the optimal C values can take several hours (depending of the computing power of the equipment), the code of the primal and dual implementations (support_vector_machine_primal_dual.py) was separated into several steps, and as the steps 1 and 2 can take a great amount of time, the software stores the intermediate results into .csv files, that are placed into the "./Cross_Validation" folder. These results are read if a .csv file is detected in this folder, skipping the cross validation process (steps 3 and 4):

- Step 0: Data Input and Preprocessing
- Step 1: Cross validation for the tuning of the C-parameter in the primal form of the SVM
- Step 2: Cross validation for the tuning of the C-parameter in the dual form of the SVM
- Step 3: Training and testing of the final primal form SVM model
- Step 4: Training and testing of the final dual form SVM model

[3] https://cvxopt.org/userguide/coneprog.html

On the other hand, the results of the implementation of the SVM using the third party option (Scikit-learn)was placed in a separate Python code (support_vector_machine_sklearn.py), and also stores the results of the cross validation.

The instructions to run the attached codes can be found in Appendix 1, or in the accompanying README.txt file.

3. Experimental results

3.1. C parameter tuning

The results of the cross-validation process for finding the optimal C value gave the results shown in Figure 3. It is important to note that the cost value in the scikit-learn was defined as C/n to make this parameter equivalent to the primal and dual form implementations. As it is possible to notice, the cost values that were tested (from 1 to 100) do not greatly modify the performance of the model, ranging from %96.84 when $C = 90$ for the Scikit-learn and primal form implementation, and an accuracy of %97.21 in the best case for all three models when $C = 10$

The behaviour of the three models is very similar, with the primal form and Scikit-learn implementation being almost identical. In all the tested SVM models, the maximum accuracy is archived with the same C value (10, or equivalently in Scikit-learn, 1/n), that what then used to train the final models

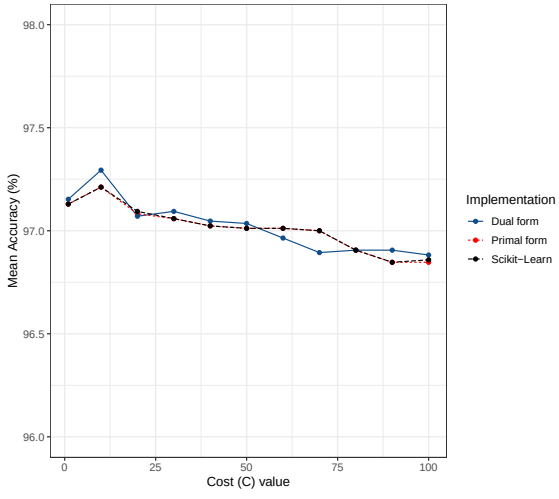

Figure 3. Cost parameter tuning using 5 fold cross validation

3.2. Model training

Using the optimized C parameter, the final models for the implemented SVM forms were trained. The distribution of the weight (w) values (Figure 4) shows that 6 out of the 200 variables present larger absolute values, being more influential in the final outcome, as the scaling of the features is relatively similar.

The weights of the three models are similar, with much more noticeable differences in the intercept (b), were the dual form gives $b = 1.1086$, and the primal form and the Scikit-learn implementations yielding $b = 1.15601$ (presenting differences after the fourth decimal).

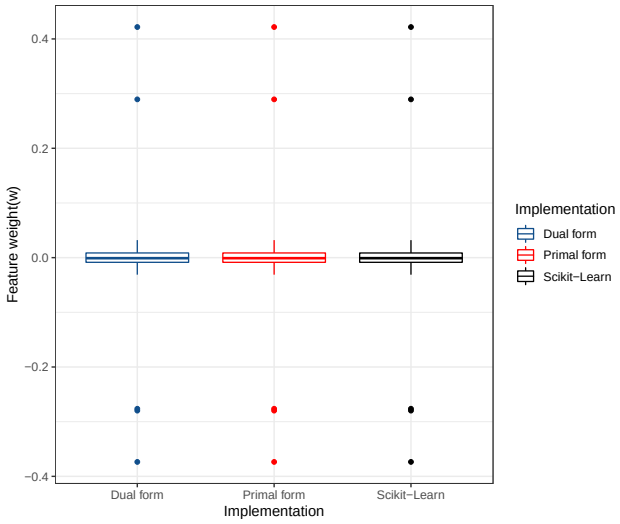

Figure 4. Weights (w) distribution in the models

3.3. Training and testing performance

The results in the training and testing dataset show that the theme models did not show significant overfitting, with the accuracy values in the test data being only slightly smaller than the training accuracy (Table 1), showing the importance of using cross-validation to tune the hyperparameters of the model.

The three models show a test accuracy above 97%, showing a high predicting performance of the SVM implementations for this database, showing the high performance the SVM can archive, even in non-separable cases.

SVM Implementation	Best C value	Train accuracy	Test accuracy
Primal form	10	0.9764	0.9740
Dual form	10	0.9774	0.9747
Scikit-learn	10/8500	0.9764	0.9740

Table 1. SVM implementations train and test accuracy results

4. Conclusion

This project showed that with enough understanding of the principles and processes behind a maximum margin classifier such as SVM, it is possible to implement an effective classification algorithm using convex optimization libraries. The implemented methods produced almost equal results when compared with third party implementations, showing that it is possible to program complex algorithms that can produce accurate results when the mathematics behind it are well understood.

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. 2009.

[3] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer, 1995.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

Appendix 1: Instructions on how to run the attached code:

This code was tested under a Linux 64 bit OS (Ubuntu 18.04 LTS), using Python 3.7.7

In order to use this code:

1. Install Miniconda or Anaconda 2. Add conda forge to your list of channels

In the terminal run:

```
conda config --add channels conda-forge
```

3. Create a environment using the requirements.yml file included in this .zip:

Open a terminal in the folder were the requirements.yml file is (Assign1-code) and run:

```
conda env create -f requirements.yml --name svm-env
```

4. Make sure the folder structure of the project is as follows

```
Assign1-code
 Input_Data
 Cross_Validation
 Results
 support_vector_machine_primal_dual.py
 support_vector_machine_sklearn.py
 ...
```

If there are csv files in the Cross_Validation folder the code will read them to avoid the delay of the cross validation and go straight to fitting the models

5. Run the code in the conda environment: Open a terminal in the Assign1-code folder and run

```
conda activate svm-env
python support_vector_machine_primal_dual.py
```

or run the support_vector_machine_primal_dual.py code in your IDE of preference, (VS Code with the Python extension is recommended), using the root folder of the directory (Assign1-code) as working directory to make the relative paths work.

6. For comparison, run the code of the SVM implementation in Scikit-learn

```
python support_vector_machine_sklearn.py
```

Note: Alternatively, for 2 and 3 you can build your own environment following the package version contained in requirements.yml file