# COMP5318

## Assignment 1

ARYAN BHATIA: 490352056
MUDIT MALHOTRA: 490583269

## TABLE OF CONTENTS

## INTRODUCTION

The following document is a report summarising the methods and experiments conducted by our team for COMP5318: Machine Learning and Data Mining Assignment 1. This assignment aimed to teach students how to clean and pre-process a dataset and then apply machine learning algorithms to train a model that can classify future unseen data. The dataset provided consisted of labelled 28X28 grayscale images of clothing items. Our goal was to build multiple classifiers that can categorise new unlabelled images. The categories of clothing items are:

- 0: T-Shirt/Top
- 1: Trouser
- 2: Pullover
- 3: Dress
- 4: Coat
- 5: Sandal
- 6: Shirt
- 7: Sneaker
- 8: Bag
- 9: Ankle boot

The importance of this assignment was that it allowed students to work on a real-world project from scratch. It gives them an idea of what it is like to be a data scientist and how they tackle new problems. The study is crucial to understand the performance of each type of classifier taught in the course and to understand the impact of data pre-processing and parameter tuning for enhancing the performance of the classifiers. Furthermore, this study provided the experience to produce tables and graphs to compare the results of different classifiers and have a visual understanding of the algorithms' performance.

## METHODS

### CLASSIFIERS USED

The following section summarises the 4 classifiers and 1 ensemble method we considered during our search for the best classifier for this dataset:

**KNN**

KNN or K-Nearest-Neighbours algorithm classifies a new unlabelled data point by considering the classes for "similar" data points in the data set. Similar points or "neighbours" are found using distance metrics like Manhattan Distance or Euclidian Distance (Appendix B: Figure 1). The algorithm calculates the distance between the unlabelled data point and all other data points using their attributes and one of the distance metrics. It then determines the label for the new point by looking at the labels of the k points nearest to it.

The following is a summary of the relevant parameters of the sklearn KNeighboursClassifier along with their default values [6]:

- *n_neighbours (default=5)*: The number of neighbours to be used for classification
- *p (default*=2, Euclidian)

**Logistic Regression**
Logistic Regression helps in binary classification i.e., classifying a data point into one of 2 types (0/1, true/false, yes/no). In the case of this assignment, there are more than 2 classes available, so the algorithm follows a One vs All approach for categorisation. In the One vs All approach, we work with each class by treating that class as 1 and all other classes as 0 [8].

The following is a summary of the relevant parameters of the sklearn LogisticRegression along with their default values [7]:

- C (default=1.0)
- solver (default=ibfgs)
- *max_iter (default=100):* This value decides the maximum number of iterations the solver takes to converge. At default values, our code threw an error because of the large size of the dataset. Therefore, 100 iterations are not enough to determine convergence. Hence, we set this value to 5000 every time we used the LogisticRegression class.

**Naïve Bayes**
When the algorithm comes across a new data point, it calculates the probability of the occurrence of each class based on the new data provided. The class with the maximum probability wins and the new data point is labelled as that. The formula to calculate this probability (Appendix B: Figure 2) is based on the Bayes probability theorem and two assumptions:

- attributes are conditionally independent
- attributes are equally importance

In real world scenarios, both these conditions are barely fulfilled, which is why the algorithm is called "naïve".

The following is a summary of the relevant parameters of the sklearn GaussianNB along with their default values [9]:

- var_smoothing (default=1e-9)

**Support Vector Machine**
SVM is another example of a binary classifier. Given a dataset, it tries to find the ideal hyperplane that can act as a decision boundary. Its goal is to separate the dataset into two classes with the maximum

margin in between. Data points falling on either side of the hyperplane can be classified into one of the two categories. The data points of each class on the respective boundary of this hyperplane are called support vectors as they help define the margin (Appendix B: Figure 3).

The following is a summary of the relevant parameters of the sklearn SVC along with their default values [10]:

- C (default=1.0)
- gamma (default=scale)
- kernel (default=rbf)

**Bagging (SVM)**
Bagging is the ensemble method we used to increase the accuracy. It divides the training set into multiple subsets called "bags" and trains multiple base estimators on each of the bag. It then takes the average of all these estimators in order to classify data points. Since SVM was our best model so far, we decided to choose it as the base estimator to perform bagging.

## PRE-PROCESSING TECHNIQUES

**Normalisation**
An extremely useful pre-processing step which transforms the attributes to a [0, 1] range. Especially crucial in the case of distance-based algorithms like KNN. The best way to explain the concept of normalisation is through an example.

Example: Predict if a customer is likely to buy a house based on 2 attributes age and house_price. Now age is recorded in year while house_price in a currency like US$, AU$. Therefore, if any distance metric is applied on two datapoints then the distance will be predominantly determined by the house_price attribute and the age attribute will end up playing no role in deciding the nearest data points.

A = [20, 500000]

B = [40, 600000]

Manhattan(A, B) = |20 – 40| + |600000 – 500000| = 100020

To solve this issue, for each value x of a particular attribute, we replace it with the scaled x' value. This x' is calculated using the following formula:

$$x' = x - \frac{\min(x)}{\max(x) - \min(x)}$$

For the purpose of this assignment, we used the MinMaxScaler class of the sklearn.preprocessing library to normalise the dataset to transform every value into the [0, 1] range.

**Dimensionality Reduction (PCA)**

Dimensional reduction is the process of reducing the number of features in a dataset. This is especially important pre-processing step for large datasets as it can help significantly when it comes to reducing computation time. PCA or Principal Component Analysis is a dimensional reduction method that constructs a new projection of the dataset having new features. These new features are built by combining multiple original features of a dataset and help catch the essence of it.

## EXPERIMENTS RESULT AND DISCUSSION

The data has been split into training and testing set using *train_test_split* in the sklearn library with 70% of the data in training set and 30% of the data in testing set. The training set has been used to build the models and the testing set has been used to measure the performance of the models.

The performance of each classifier has been evaluated on different stages of building the models. These stages are:

- Model creation without data pre-processing and using default parameters for the classifiers
- Model creation with data pre-processing and using default parameters for the classifiers
- Model creation with data pre-processing and using tuned hyperparameters for the classifiers

The performance is measured by testing the accuracy on training set, accuracy on testing set, testing set precision, testing set recall, testing set F-score, and computation time (training + inference) at each stage of building the model to evaluate the impact of data pre-processing and parameter tuning in each classifier.

**K – Nearest Neighbours**

| | Default Parameters + No Pre-Processing | Default Parameters + Pre-Processing | Tuned Parameters + Pre-Processing |
|---|---|---|---|
| Accuracy Training-Set (%) | 88.7 | 89.5 | 89.8 |
| Accuracy Testing-Set (%) | 84.1 | 85 | 85.1 |
| Precision (%) | 84.4 | 85.1 | 85.2 |
| Recall (%) | 84.1 | 85 | 85.1 |

| | | | |
|---|---|---|---|
| **F-Score (%)** | 84.1 | 85 | 85.1 |
| **Computation Time (seconds)** | 25.5 | 15.4 | 56.9 |

- By creating a model without implementing pre-processing techniques and performing hyperparameter tuning, an accuracy of 84.1% is achieved by KNN classifier on the testing set.
- After normalising and standardising the data and performing dimensionality reduction, a boost in accuracy is seen for both training and testing sets. Therefore, it can be inferred that data pre-processing can enhance the accuracy of a KNN model. Furthermore, dimensionality reduction using PCA has reduced the computation time for creating the model by 1 second.
- Following is the table for parameters and their corresponding values used for performing hyperparameter tuning.

| **Parameter** | **Value Range used for Parameter Training** |
|---|---|
| **n_neighbour** | 1, 3, 5, 11, 15 |
| **P** | 1, 2 |

Grid search has been performed in combination with 5 cross-fold validation to find the best value of the parameters using the *GridSearchCV* function in sklearn library. This helps in reducing overfitting/underfitting if it exists in the model. The best combination of parameters found from hyperparameter tuning are:

- n_neighbours = 5
- p = 1 (Manhattan distance)

Hyper-parameter tuning did not have a significant impact on enhancing the performance of the model since the model performed the best with the default value for n_neighbour = 5. However, the computational time increased significantly due to the use of Manhattan distance instead of Euclidean distance as it takes more time to compute *[1]*. Furthermore, it contributed towards minor performance improvement in the model as it works well for higher dimension data and reduces the chances of misclassification as it emphasises less on the outliers *[2]*.

Overall prediction performance of the algorithm is good since KNN is well known for multi-class classification. This is because it works on the assumption that similar data points are close to each other and classifies a new data point according to its distance to the specified number of nearest neighbours *[3]*.

**Logistic Regression**

| | Default Parameters + No Pre-Processing | Default Parameters + Pre-Processing | Tuned Parameters + Pre-Processing |
|---|---|---|---|
| **Accuracy Training-Set (%)** | 92.1 | 85.4 | 85.4 |
| **Accuracy Testing-Set (%)** | 79.8 | 84.3 | 84.3 |
| **Precision (%)** | 79.6 | 84.2 | 84.2 |
| **Recall (%)** | 79.9 | 84.3 | 84.3 |
| **F-Score (%)** | 79.7 | 84.2 | 84.2 |
| **Computation Time (seconds)** | 220 | 9.59 | 9.59 |

- By creating a model without implementing pre-processing techniques and performing hyperparameter tuning, an accuracy of 79.8% is achieved by Logistic Regression classifier on the testing set.
- Data pre-processing had a significant impact on the model performance as the accuracy on test set increased by 4.5%. Furthermore, computation time to create the model also decreased significantly from around 3.5 minutes to just 8.4 seconds. This is due to the use of PCA dimensionality reduction because of which the number of features reduced from 784 to 84 principal components.
- Following is the table for parameters and their corresponding values used for performing hyper-parameter tuning:

| **Parameter** | **Value Range used for Parameter Training** |
|---|---|
| **C** | 100, 10, 1.0, 0.1, 0.01 |
| **solver** | lbfgs, liblinear |

Grid search has been performed in combination with 5 cross-fold validation to find the best value of the parameters using the *GridSearchCV* function in sklearn library. The best combination of parameters found from hyperparameter tuning are:

- C = 1
- solver = lbfgs

Hyper-parameter tuning didn't have an impact on the performance of the classifier since the default values of the parameters lead to best performance of the classifier for the given data. The default value of C = 1 is the best parameter which means increasing or reducing the regularisation has a negative impact on the performance of the model for the given dataset.

Overall prediction performance of the classifier is good since the different image classes are linearly separable and can be distinguished from one another using logistic regression [4].

**Gaussian Naïve Bayes**

| | Default Parameters + No Pre-Processing | Default Parameters + Pre-Processing | Tuned Parameters + Pre-Processing |
|---|---|---|---|
| **Accuracy Training-Set (%)** | 60.5 | 77.3 | 77.4 |
| **Accuracy Testing-Set (%)** | 60.1 | 77.1 | 77.2 |
| **Precision (%)** | 65.2 | 77.8 | 78 |
| **Recall (%)** | 60 | 77.1 | 77.2 |
| **F-Score (%)** | 57.4 | 77.2 | 77.4 |
| **Computation Time (seconds)** | 2.88 | 0.178 | 0.170 |

- By creating a model without implementing pre-processing techniques and performing hyperparameter tuning, an accuracy of 60.1% is achieved by Naïve Bayes classifier on the testing set.
- Data pre-processing had a significant impact on the model performance as the accuracy on test set increased by 17%. Furthermore, computation time to create the model also decreased significantly from around 0.7 to just 0.2 seconds. This is due to the use of PCA dimensionality

reduction because of which the number of features reduced from 784 to 84 principal components.

- Following is the table for parameters and their corresponding values used for performing hyperparameter tuning:

| **Parameter** | **Value Range used for Parameter Training** |
|---|---|
| **var_smoothing** | 100 numbers spaced evenly on log scale between 0 and -9 |

Grid search has been performed in combination with 5 cross-fold validation to find the best value of the parameter using the *GridSearchCV* function in sklearn library. The best parameter value found from hyperparameter tuning is:

  - Var_smoothing = 0.000285

Hyperparameter tuning for Naïve Bayes did not improve the accuracy significantly but helped in reducing the computational time.

Performance of this classifier is not good compared to other classifiers since it is “naïve” in nature as explained in the *Methods section* of the report.

**Support Vector Machine + Ensemble (Bagging)**

| | Default Parameters + No Pre-Processing | Default Parameters + Pre-Processing | Tuned Parameters + Pre-Processing | Tuned Parameters + Pre-Processing + Bagging |
|---|---|---|---|---|
| Accuracy Training-Set (%) | 90.4 | 89.8 | 93.4 | 93.2 |
| Accuracy Testing-Set (%) | 87.4 | 87.8 | 88.7 | 88.7 |
| Precision (%) | 87.3 | 87.7 | 88.6 | 88.6 |
| Recall (%) | 87.4 | 87.8 | 88.7 | 88.7 |

|  |  |  |  |  |
|---|---|---|---|---|
| **F-Score (%)** | 87.3 | 87.7 | 88.6 | 88.6 |
| **Computation Time (seconds)** | 355 | 70 | 61 | 122 |

- By creating a model without implementing pre-processing techniques and performing hyperparameter tuning, an accuracy of 87.4% is achieved by SVM classifier on the testing set.
- Data pre-processing reduced the computation time by almost three times for creating the model while maintaining its accuracy. This is due to the use of PCA dimensionality reduction because of which the number of features reduced from 784 to 84 principal components.
- Hyperparameter tuning was done using Grid Search and following is the table for parameters and their corresponding values used for performing the grid search:

| **Parameter** | **Value Range used for Parameter Training** |
|---|---|
| **C** | 100, 10, 1.0, 0.1, 0.01 |
| **Gamma** | 1, 0.1, 0.01, 0.001, 0.0001 |
| **Kernel** | 'linear', 'poly', 'rbf', 'sigmoid' |

Grid search has been performed in combination with 5 cross-fold validation to find the best combination of values of the parameter using the *GridSearchCV* function in sklearn library. The best parameter value found from hyper-parameter tuning is:

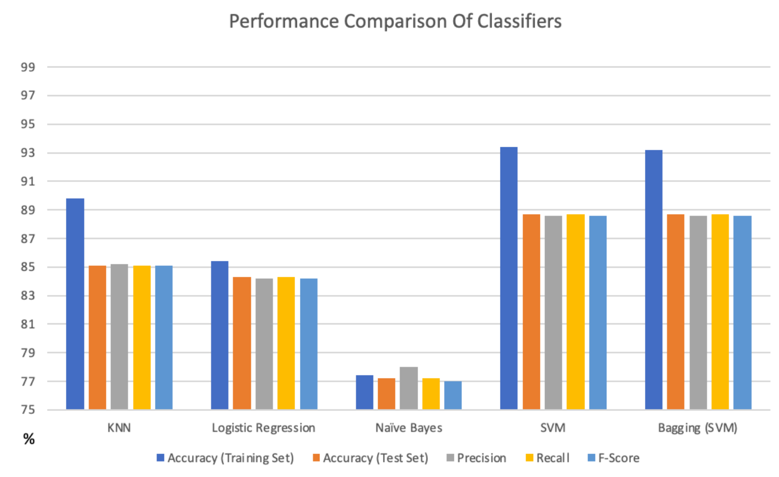
- C = 10
- Gamma = 0.01
- Kernel = rbf

SVM classifier is highly sensitive towards parameter tuning and therefore tuning the parameters lead to enhanced performance of the model (increase in accuracy by 1%) and reduced computational time (reduced by almost 3 seconds).

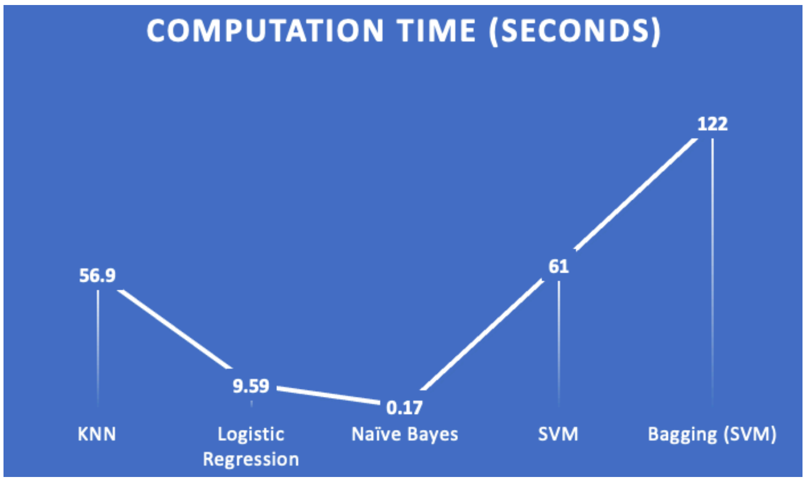
The model performs really well as SVM is a supervised learning algorithm and is known to work well for classification problems [5]. Increasing the C value led to a better performance on the test data set because the hyper plain has a smaller margin between the classes and is classifying the training points with higher accuracy. This combined with a low 0.01 value of gamma ensured that each training example has significant influence on the model classification performance as it reduces overfitting and under fitting and generalises the results of the model more accurately [5].

Bagging was performed on the classifier after hyper parameter tuning to see if the performance can further be enhanced. However, the performance remained the same.

The following plots summaries and compares the performance of the classifiers with each other



## CONCLUSION



After analysing the performance of each classifier, we can see that the SVM classifier can predict the class of the new data more accurately than others with an acceptable amount of computational time. To further enhance the performance bagging was performed on the SVM classifier but as unsuccessful. Therefore, bagging although a useful ensemble for enhancing the performance of a classifier does not always give the desired results. Furthermore, for each classifier it can be seen that the computation time (training and inference) can be reduced significantly by performing data pre-processing (normalisation and PCA dimensionality reduction). Performance of each model can be improved by adding more data for training and carrying out sophisticated hyper-parameter tuning by choosing a large range of values of parameters which will ensure maximum reduction in underfitting/overfitting when training the models.

## REFERENCES

**Note**: Our team frequently referred to the lecture slides and tutorial sheets while writing the code/report for this assignment. Other than the lecture slides, we also referred to some external resources and documentation to better understand the concepts involved and the coding practices followed in the industry:

[1] S. Karki, "Comparison of A*, Euclidean and Manhattan distance using Influence Map in Ms. Pac-Man", Diva-portal.org, 2016. [Online]. Available: http://www.diva-portal.org/smash/get/diva2:918778/FULLTEXT02.pdf.

[2] K. Gohrani, "Different Types of Distance Metrics used in Machine Learning", Medium, 2019. [Online]. Available: https://medium.com/@kunal_gohrani/different-types-of-distance-metrics-used-in-machine-learning-e9928c5e26c7#:~:text=This%20means%20that%20the%20L1,the%20'curse%20of%20dimensionality'.

[3] "k-nearest neighbors algorithm - Wikipedia", En.wikipedia.org. [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[4] S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: a methodology review", Journal of Biomedical Informatics, vol. 35, no. 5-6, pp. 352-359, 2002. Available: 10.1016/s1532-0464(03)00034-0

[5]"Support-vector machine - Wikipedia", En.wikipedia.org, 2022. [Online]. Available: https://en.wikipedia.org/wiki/Support-vector_machine.

[6] s. docs, "sklearn.neighbors.KNeighborsClassifier", *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. [Accessed: 09-Apr- 2022].

[7] s. docs, "sklearn.linear_model.LogisticRegression", *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html. [Accessed: 09-Apr- 2022].

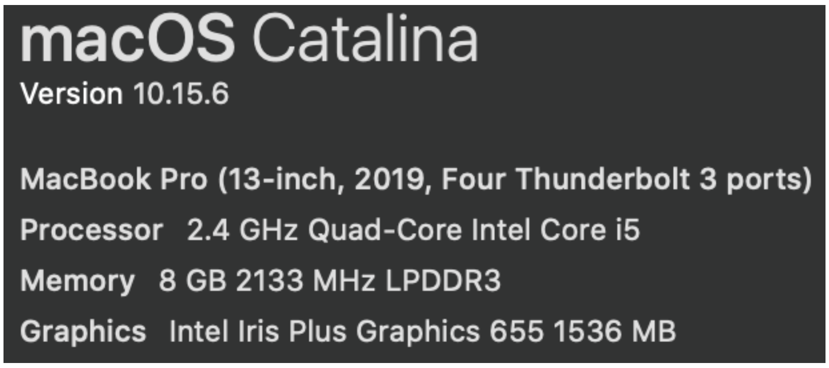
[8] R. Sucky, "Multiclass Classification Algorithm from Scratch with a Project in Python: Step by Step Guide", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/multiclass-classification-algorithm-from-scratch-with-a-project-in-python-step-by-step-guide-485a83c79992. [Accessed: 09-Apr- 2022].

[9] s. docs, "sklearn.naive_bayes.GaussianNB", *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html. [Accessed: 09- Apr- 2022].

[10] s. docs, "sklearn.svm.SVC", *scikit-learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. [Accessed: 09- Apr- 2022].

# APPENDIX: A

## HARDWARE AND SOFTWARE SPEC USED FOR EVALUATION



macOS Catalina
Version 10.15.6

MacBook Pro (13-inch, 2019, Four Thunderbolt 3 ports)
Processor 2.4 GHz Quad-Core Intel Core i5
Memory 8 GB 2133 MHz LPDDR3
Graphics Intel Iris Plus Graphics 655 1536 MB

The usual run time of our code on the above-mentioned specifications is between **5:30 to 6 minutes.** This includes time for data importing, pre-processing, implementing each algorithm.

## RUNNING THE CODE

Some code snippets take a long time to run and are not required to be run every time. For example, the GridSearchCV for parameter tuning takes between 10 minutes to over 2.5 hours depending on the classifying algorithm. Therefore, such time-consuming snippets have been converted to markdown and stored in a **static** folder so that the code runs in under 10 minutes and can also be run by simply clicking **Run All** in jupyter notebooks.

**Note:** The static folder should always be present in the same directory as the 490352056_490583269_code.ipynb file.

For exporting the code, we have created an **export_predictions(filename, classifier)** helper function that takes a filename and classifier as arguments. It calculates the predictions of the classifier on the blind data and stores these in the file passed as the argument inside the output folder. The options for the classifier argument are:

- knn
- logreg (LogisticRegression)
- nb (Naïve Bayes)
- svm
- bclf (BaggingClassifier using SVM)

## APPENDIX: B

- Euclidean distance (L2 norm) – most frequently used

$$D(A,B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2}$$

D(A,B) = sqrt $((1-1)^2+(3-6)^2+(5-9)^2)=5$

- Manhattan distance (L1 norm)

$$D(A,B) = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$

D(A,B)=|1-1|+|3-6|+|5-9|=7

Figure 1: Distance Measures

$$P(yes \mid E) = \frac{P(E_1 \mid yes)P(E_2 \mid yes)P(E_3 \mid yes)P(E_4 \mid yes)P(yes)}{P(E)}$$

$$P(no \mid E) = \frac{P(E_1 \mid no)P(E_2 \mid no)P(E_3 \mid no)P(E_4 \mid no)P(no)}{P(E)}$$

Figure 2: Naive Bayes Formula

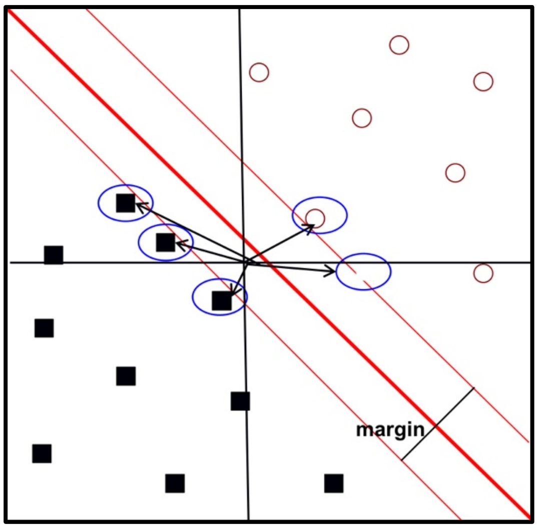


Figure 3: SVM Hyperplane