ECE - 579A

Data Analysis Report
On
Fashion MNIST Dataset

Submitted by:
Divyansh Bhardwaj (V00949736)
Master of Engineering in Applied Data Science (MADS)
University of Victoria

Submitted to:
Dr Nishant Mehta
University of Victoria

INTRODUCTION

This is an analysis report based on the performance of different Machine Learning Models implemented in Python. The problem given to us was a classification problem, based on whether a particular image in the dataset belongs to which one of the 10 classes which are there in the dataset. Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We were advised to extract out two classes 'sandles' and 'sneakers' (classes 5 and 7) from the dataset which constituted 12000 images in the training dataset and 2000 for the testing dataset.

The dataset has 784 features and a target variable (binary classification) which takes the value either 5 or 7 (5 indicating the sandals and 7 indicating the sneakers). Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The data has been loaded using one of the functions provided on the GitHub page of the MNIST dataset. The data has been stored first into the pandas Dataframes and then have been divided into the testing and training dataset. The features have been normalized by dividing the values by 255 as each pixel has a value between 0 and 255. After all this processing, the data is finally converted into numpy arrays so that it can be used in a better way with sklearn library.

Analysis on the Fashion MNIST Dataset

The analysis on the Fashion MNIST Dataset has been carry forwarded by implementing different machine learning classification models namely Logistic Regression, Support Vector Machine (Soft margin version of the SVM) and further tuning their hyperparameters using cross validation techniques to get a better accuracy or lower error for the same. Since implementing the models were taking a lot of time, I had to make the dataset a bit smaller in order to get the results on time. I personally feel that we need a good GPU for doing the processing of a visual dataset. But since most of the machines that we have at our homes does not have a good GPU. Hence it takes a lot of time to train the models and plotting the graphs or tuning the hyperparameters were taking forever to get implemented. Hence the instead of using all of the dataset of 12000 images, I used 6000 images for training dataset (3000 for each class) and used 2000 for test dataset. The running time was still on a higher side but was implemented by having a reasonable patience in me.

Now let's have a look at the different machine learning models that have been implemented in this assignment and see how the results came out to be for each model.

1.) Logistic Regression Model:

The Logistic regression model was implemented on the dataset having 6000 training samples and 2000 samples for testing the model. First the model was trained by using the default parameters of the model. The error for the model and hence the accuracy of the model was calculated to see how the model is performing for the data. Below is shown the accuracy result for the trained model with the default hyperparameters.

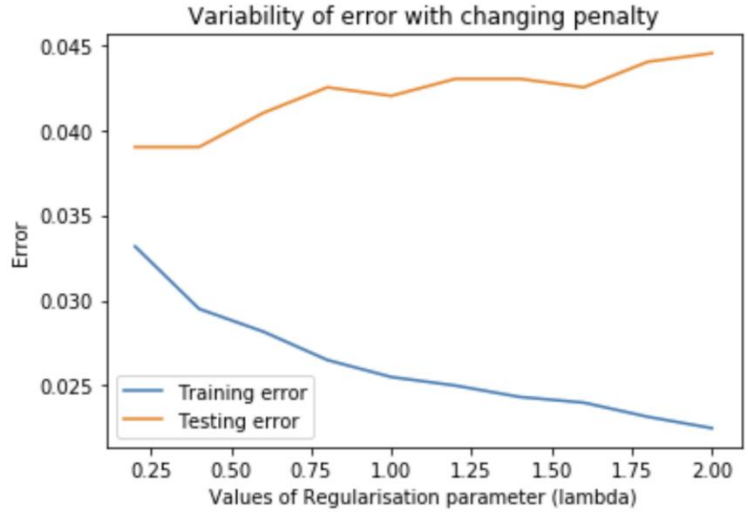
```
In [73]: print('The accuracy score of the Logistic regression model:')
         lr_pred = LR_model_train(x_train, y_train, x_test)
         print(accuracy_score(y_test, lr_pred) * 100)

         The accuracy score of the Logistic regression model:
         95.8
```

As can be seen from the above image, the accuracy score for the logistic regression model came out to be 95.8%. The accuracy is quite on a higher side, but this accuracy might still have a scope to be increased as the regularisation parameter has not been tuned yet.

After this, now it was time to manually tune the logistic regression model by taking 10 values for C and computing error for the same. To perfectly depict these scenarios, I like to plot a graph for the same. So taking the values of regularisation parameter on the x-axis and error on y axis, I plotted a graph which has been shown below:

```
Out[17]: Text(0.5, 1.0, 'Variability of error with changing penalty')
```




```
In [74]: print('The best accuracy score came out to be:')
         print((1 - min(error_test))*100)

         The best accuracy score came out to be:
         96.1
```

As we can see from the above graph, the Training and testing errors have been plotted against the values of regularisation parameter ranging from 0.1 to 1.0 (10 values). We can clearly see from the above graph that the value for the test error was at its minimum already and started increasing as the value for the regularisation parameter has been increased. This clearly depicts how the overfitting has been avoided and how important it is to select the correct and concise value for regularisation parameter so that we can have a better model. To check what was the minimum error or in other words what is the best accuracy achieved by this model after tuning it with

different values of C came out to be 96.1% which is a good increase from the previous value of accuracy. Here we can appreciate the increase in the accuracy though it may seem to be a slight increase because already the model has been performing quite well with default values of the regularisation parameter.
For getting the proper tuned value for the regularisation parameter, we will be performing one of the cross validation techniques popularly known as k-fold cross validation. For now let's go ahead with the next machine learning model (SVM) and see how that is performing for the same dataset.

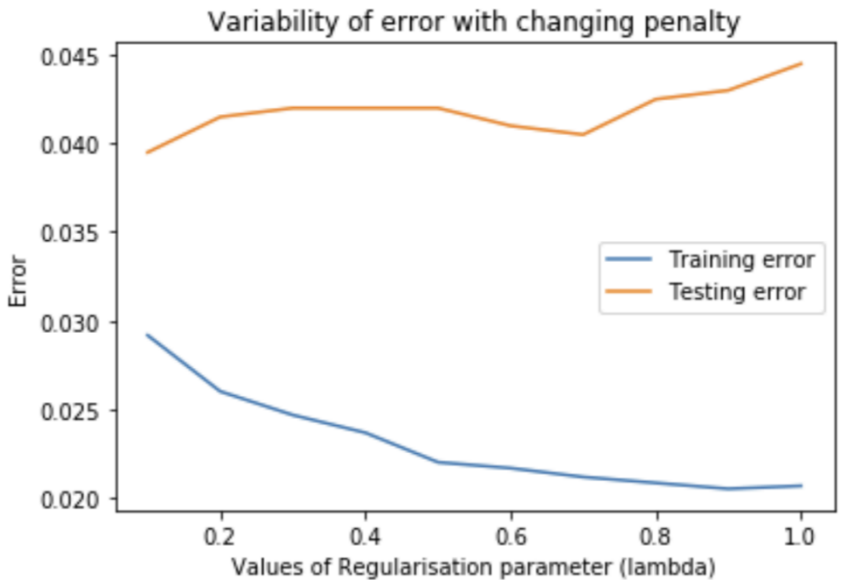
2.) Support Vector Machine (SVM with Linear Kernel):

After implementing the logistic regression, I implemented SVM classifier for the same dataset by using the Linear kernel in the model. Just like we did with the logistic regression model, we first trained the model with the default parameters for the model. We will also be applying the k-fold cross validation technique for tuning the hyperparameters but first let's have a look at how the model is performing with the default hyperparameters. Below is shown the accuracy calculated for the model trained with default values of parameters:

[98]:
```
print('The accuracy score of the SVM model:')
svm_pred = SVM_model_train(x_train, y_train, x_test)
print(accuracy_score(y_test, svm_pred) * 100)
```

```
The accuracy score of the SVM model:
95.55
```

The accuracy of linear SVM model came out to be 95.55 % which is very slightly lower than what we got for logistic regression. In other words we can say that both the models equivalently good on this dataset. But since i used all the default parameters for training the model, I was curious what effect or what change could be there when we combat the overfitting and to what extent the accuracy could be increased. Hence, I plotted the graph for training and testing error for the values of regularisation parameter C. Below is the graph shown for the same:



From the above graph, there is a particularly important point that needs a proper mentioning here. As it was discussed in the class as well, the testing error, if we look closely, is increasing but it dipped again. There can be some scenarios where the testing error might increase a bit but then have a dip as well which could have a test error minimum than the previous dip. So even if the test error starts increasing with the decrease in training error, we should always check for the next dip if it is there. Moreover, this again depicts the overfitting situation quite well as the value of test error was at it minimum but starts increasing overall even with the decrease in training error.

The minimum error or the best accuracy we got with the model trained with the default parameters was calculated and is shown below:

```
In [100]: print('The best accuracy achieved with the value of C is:')
          print((1 - min(error_svm_test)) * 100)

          The best accuracy achieved with the value of C is:
          96.05
```

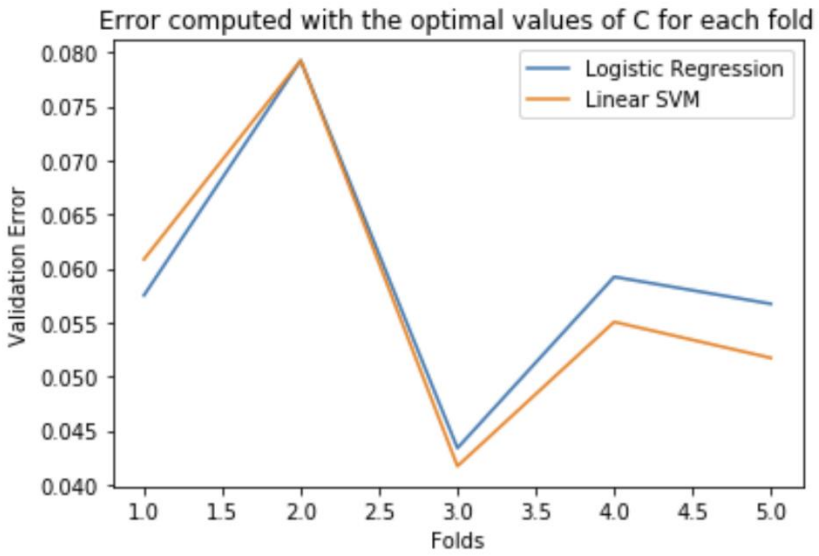
The best accuracy our model achieved with the SVM model came out to be 96.05%. Again we will be performing the hyperparameter tuning with SVM model afterwards in the coming pages of report. We will be doing the k-fold cross validation with the linear kernel SVM and with the Gaussian kernel SVM. We will also be tuning the hyperparameters 'γ' and regularisation parameter 'C'. This will be done the next section.
Now, Since we have been talking a lot about cross validation, let's have a look at the k-fold cross validation technique and the results produced by it.

3.) K-Fold Cross Validation:

While training the machine learning models, there are many hyperparameters that we have to tune so that we get the best results for that dataset and also problems like overfitting can be avoided. For tuning them we use one of the cross-validation technique known as k-fold cross validation. For this assignment, I have chosen k to be 5 and have applied to logistic regression model and many versions of SVM. These are discussed in the sub headings.

- **Applying K-Fold Cross validation on Logistic Regression model and Linear SVM model**

  First, we applied the k-fold cross validation for both Logistic regression model and Linear SVM model. The value of K was set to be 5. So, for every fold i computed the optimal regularisation parameter for each of the model and the test error (or validation error as it was done on training set) has been computed. Since the models were taking a lot of time to get trained on multiple folds hence the value of k was set to 5 (as the value was suggested to be between 5 and 10). To depict the validation error in a better way, I plotted a graph for the 5 folds for Logistic regression and Linear SVM. Below is the graph shown for both the models:



From the above figure, we can notice that the errors for both the models are behaving very closely for every fold. Particularly, for the third fold, the error has dropped for both the models and Linear SVM has been performing relatively better after the first fold, but the difference is ver minute. Hence, we can say that Linear SVM model and Logistic Regression model both are performing equivalently in terms of performance after the cross validation. After performing the cross validation we trained both the models with the optimal parameters. Below is shown the accuracy after training with the optimal parameters.

In [95]:
```
y_predlr = LR_model_train(x_train, y_train, x_test, c_logr)
err_lr = accuracy_score(y_predlr, y_test)*100
print(err_lr)
```

```
96.15
```

Hence, the accuracy for the Logistic model trained with the optimal parameter came out to be 96.15 percent whereas with the default parameters, it was 95.85%. Now let's check for the accuracy of the Linear SVM model.

```
y_predsvm = SVM_model_train(x_train, y_train, x_test, csvm[ind_svm])
err_svm = accuracy_score(y_predsvm, y_test)*100
print(err_svm)
```

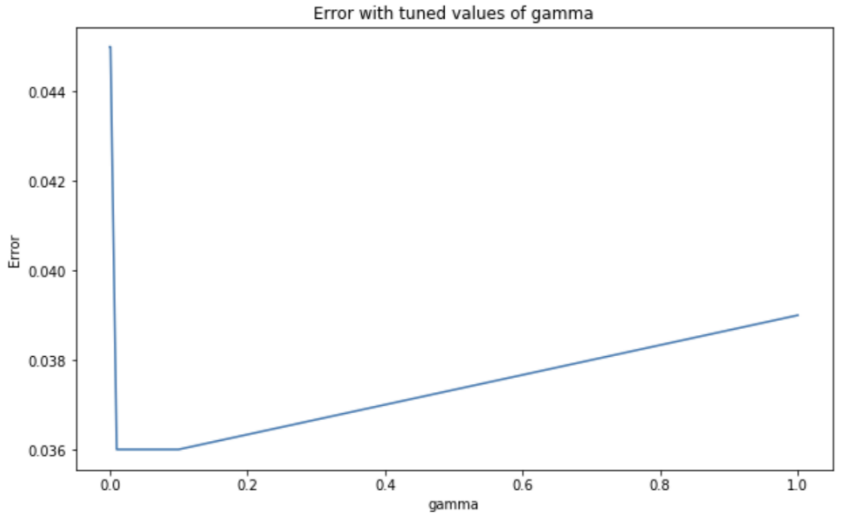
```
96.0
```

The accuracy for the Linear SVM model came out to be 96.0% where as earlier it was 95.55% which can be considered as a significant increase from the earlier model.
Now, comparing both the models, we can say that after tuning the hyper parameters the models have shown the accuracy of 96 and 96.15 %. The difference between the performances of the

models is negligibly small. Hence for the current dataset, the match is a tie between both the models.

- **Applying K-Fold cross validation on Non Linear (Gaussian) Kernel SVM**

  After applying the cross validation on Logistic and Linear SVM, we will see the results with the Non Linear Gaussian Kernel SVM. Now here what I did was, I kept the SVM kernel to be a gaussian kernel. With this setting, I tuned 2 parameters, that is gamma('γ') and regularisation parameter 'C' using the k-fold cross validation method. The value of k was set to be 5 again for the reasons stated above.
  I took the values of gamma to be in the range [0.0001, 0.001, 0.01, 0.1, 1.0]. For these 5 values of gamma, I tuned the Non Linear SVM model and computed optimal values of C. Below is shown the graph of validation or test error vs the gamma values. Because of the close proximity of the gamma values, the graph is not able to depict it visually, but the result will be clear from the graph.



The above graph clearly depicts that with the rising value of gamma the error first drops to a minimum but then it starts increasing gradually. By tuning the values of Gamma and regularisation parameter C, I then computed the accuracy for the Non Linear SVM model after training the model on the whole training dataset with the optimal hyperparameters. Below is shown the accuracy for the same.

```
In [68]: (1 - min(err_ind))*100
Out[68]: 96.39999999999999
```

The value of accuracy achieved by the non linear SVM model with the tuned hyperparameters came out to be the maximum of all the models that we have computed

so far, i.e. 96.40 % (approx.). So the Non Linear SVM with gaussian kernel and tuned hyperparameters performed the best of all the models.

Conclusion:

The conclusion that I have drawn after looking at all the models is that the lower bound and the upper bound of the accuracy came out to be 95.85% and 96.40%. The change in the accuracy after tuning the hyperparameters and using linear and non linear models is not a significant one. Hence, according to me, the data is linearly separable as the both linear and non linear svm gave the accuracy close to 96 % which is huge. Hence the classes sandals and sneakers are the classes which can be separated via a linear decision boundary.