Assignment 1 - Group: 26

Tutors: The Canh Dinh, Fangzhou Shi

Group members : Gautam Radhakrishnan Ajit (grad0149), Lavanshu Agrawal (lagr7305), Renil Austin Mendez (rmen0735)

Abstract

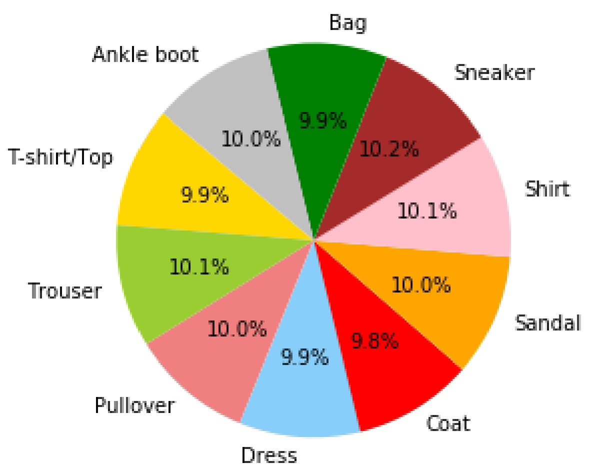
The purpose of this assignment is to evaluate and reflect upon the performances of classifier models - random forest classifier, decision tree, logistic regression and kNN - on the Fashion MNIST dataset. Their accuracies and running times will be compared side by side to realize the strengths and shortcomings of each classifier and then nominate the “best” option for further evaluation.

Introduction

Dataset:

Fashion-MNIST dataset is provided by Zalando Research which contains a training set of 60,000 examples and a test set of 10,000 examples. As a part of this assignment, the dataset has been subsetted to consist of a training set of 30,000 examples and a test set of 5,000 examples, with 2,000 of those having labels for estimating accuracy, and the remaining for predicting and evaluating purposes. The datasets consist of 28x28 pixel (a total of 784 features) grayscale images, each of which is to be categorised into one of the ten classes. The test and training datasets have a total of 785 columns which correspond to 784 (28*28) columns for the pixels and a column (Column 1) for the labels of the categories to which that particular item belongs to. The pixel columns consist of integer values from 0 to 255 depicting the lightness or darkness of that particular pixel, with the higher numbers representing darkness. (1)

The labels associated with the dataset into which each item is to be classified are:
T-shirt/Top (0), Trouser (1), Pullover (2), Dress (3), Coat (4), Sandal (5), Shirt (6), Sneaker (7), Bag (8), Ankle boot (9)



Pre-processing

Since the dataset is too large, some pre-processing methods have been considered, including singular value decomposition to run PCA (using libraries). The latter was thought to be useful for training the kNN classifier, as the number of components could be reduced to 200 from 784, but the computation time wasn't good enough to try to implement it from scratch. (2)

Logistic regression

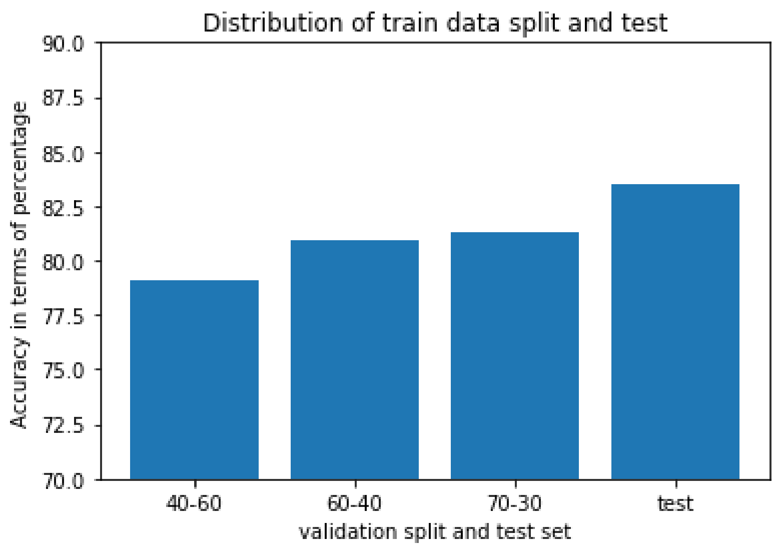
Multi-class logistic regression yielded reasonably good predictions in terms of both accuracy and running time, with each of the run models (differently tuned parameters) getting trained in under ten minutes. We started with logistic regression because of its high efficiency in fitting all the training data, especially for a large data set such as this one, compared to other classifiers like kNN and SVMs. It also makes use of a convenient **softmax** function which calculates the probability of an input vector corresponding to a particular class (exponential). (3)

Softmax function and gradient descent

For each of the categories, there is a corresponding weight matrix which determines the probability of an image belonging to that particular class given the input. With the use of a learning rate, the corresponding weights are updated, relative to the number of iterations performed. If the change in the normalized weights is less than a threshold (**'tol'**), the loop is broken, and we proceed with predicting the categories using pred function (**np.argmax**).

Bootstrapping

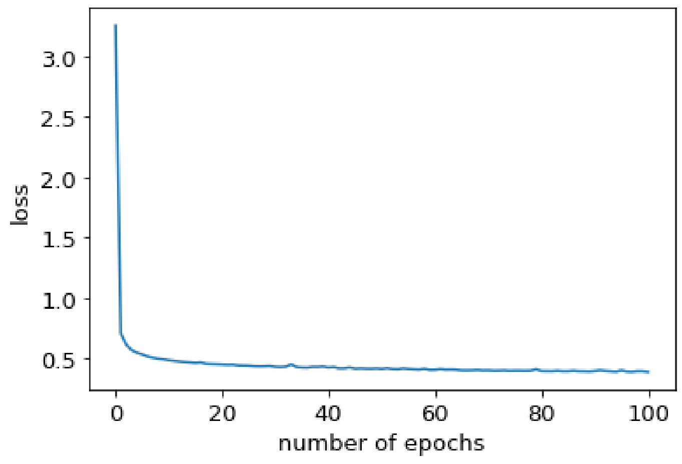
By decreasing the batch size, we increase the number of batches. This will decrease the extent to which overlapping of data points occur (as the number of batches increases, the algorithm has a lot more options to assign batches where there are less rows in common). This would update the weights more effectively, making use of a larger training dataset, and boost our accuracy. (4)



Furthermore, we split the training data into a training set and a validation set to test the accuracy of the corresponding splits before running the classifier on the test set. This is shown in the bar graph shown above.

Hyper-parameter tuning

The manner in which the weights are randomly initialised has a say in the final accuracy estimated on the test data. Often, the trade-off between the number of iterations performed, the learning rate and initial weight values determine the accuracy and performance of this classifier.



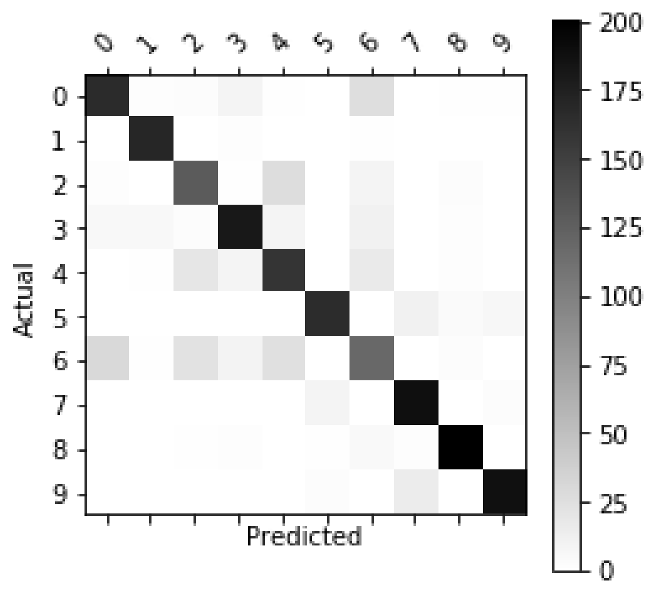
As expected, as we increase the number of epochs increases, the loss decreases.

Following is a table which depicts how changes in parameters affected our accuracy and performances on the test set:

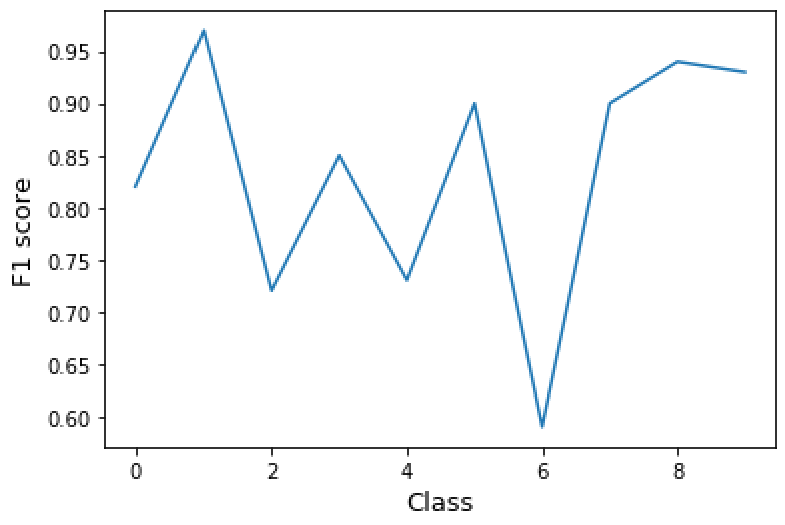
Sl No.	Batch size	Number of epochs	Learning rate	Accuracy %	Performance (in seconds)
1	100	100	0.05	83.6	26
2	100	100	0.025	83.39	26
3	100	100	0.01	83.2	27
4	1000	100	0.05	80.05	12
5	1000	100	0.025	80.25	12
6	1000	100	0.01	68.15	6
7	100	1000	0.05	83.5	51
8	100	1000	0.025	83.15	18
9	100	1000	0.01	81.39	10
10	100	2000	0.05	83.45	42
11	100	10000	0.05	83.8	43
12	100	10000	0.07	82.4	110
13	100	20000	0.07	82.45	121

Based on the observations contained in the first nine rows of the table, it can be seen that as the accuracy increases with a decrease in batch size (for the reasons stated earlier). Learning rate has an adverse effect on the performance. Lower learning rates seem to produce lower accuracies, given that the number of iterations performed is kept fixed.

Performance evaluation for logistic regression



The above chart shows the distribution of the predicted results with the actual results. The darker squares along the diagonal indicate that the classifier has more or less accurately predicted the actual classes, relatively less for classes 2 and 7. (6)



The F1 score is found to be higher for classes like 1 and 7. This could be due to an imbalanced representation of classes such as 6 in the first 2,000 test dataset labels, or due to randomising in the training set, the algorithm inadvertently misrepresented such classes.

Optimal parameters for logistic regression classifier

Batch size: 100

Number of epochs: 100

Learning rate: 0.05

Accuracy: 83.6%

Run time: 26 seconds

Random forest classifier

Random forest consist of large number of individual decision trees. The prediction of these decision trees are grouped together and the class with the most votes becomes the model’s prediction.

Considering the computational time and the ability to predict, Random forest was a better choice than decision tree under the tree classifier category. Decision tree has to feed all the 30000 training observation at once, making the computational time of finding the best split using overall entropy quite expensive. Whereas in random forest, different types of trees are fed with limited number of training data picked up randomly using bootstrap, which helps in finding the best split faster than decision tree. And when comes to prediction, its accurate to take the mode of prediction from different uncorrelated trees rather than an individual tree. (5)

Methodology used in the model

Bootstrap: It is a technique used involves random sampling of dataset with replacement. These are used for ensemble methods like in random forest which will reduce the variance of their prediction, thus increasing the prediction power.

Random subspace method: This is used to randomly pick given number of features from the training data for each decision tree.

Decision Tree – Entropy: The trees are formed by checking the purity of split of attributes. Entropy is used to measure the uncertainty of class for a subset of observations, which bring out the best split. The formula used for entropy is: ***-sum(p * log(p))*** where p refers to the probability of each class of that subset of data.

Classifying labels: After randomising each tree data, all individual decision trees will predict the class of the test data. The mode of the prediction of all the trees is taken to find the final prediction of the model.

Hyper-parameters used to the model

The parameters involved in the model for tuning are:

- *Number of Trees*: The number of decision trees included in the random forest model.

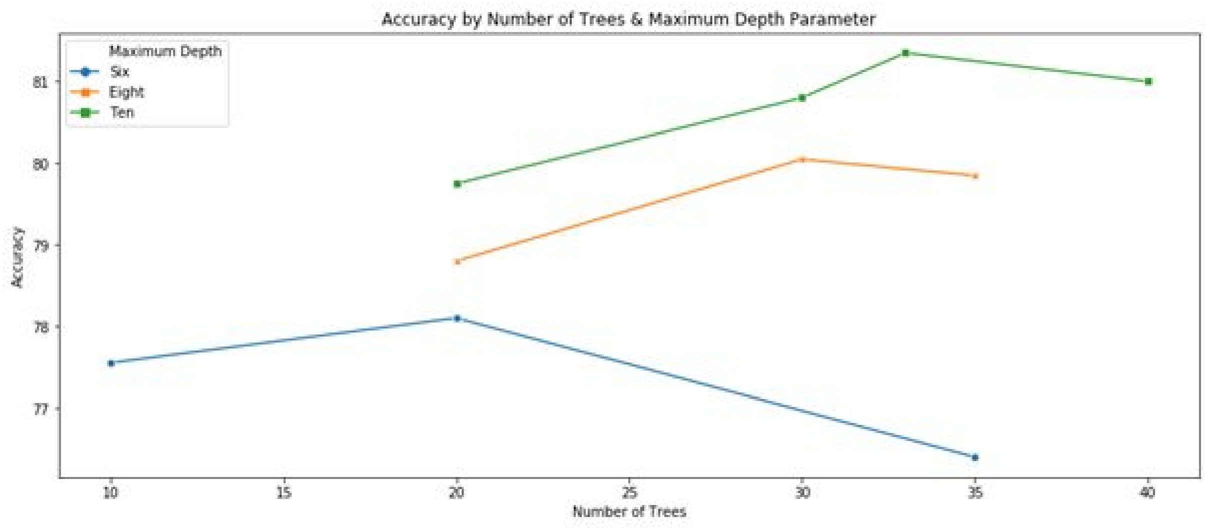
- *Size of Bootstrap*: The number of sample train data each tree should contain.
- *Number of Features*: The number of random features that each tree should try on.
- *Maximum Depth*: Represents the depth of each tree in the model. The deeper the tree, the more split and information it can capture about the data.

Comparison of random forest model with different hyper-parameter tuning

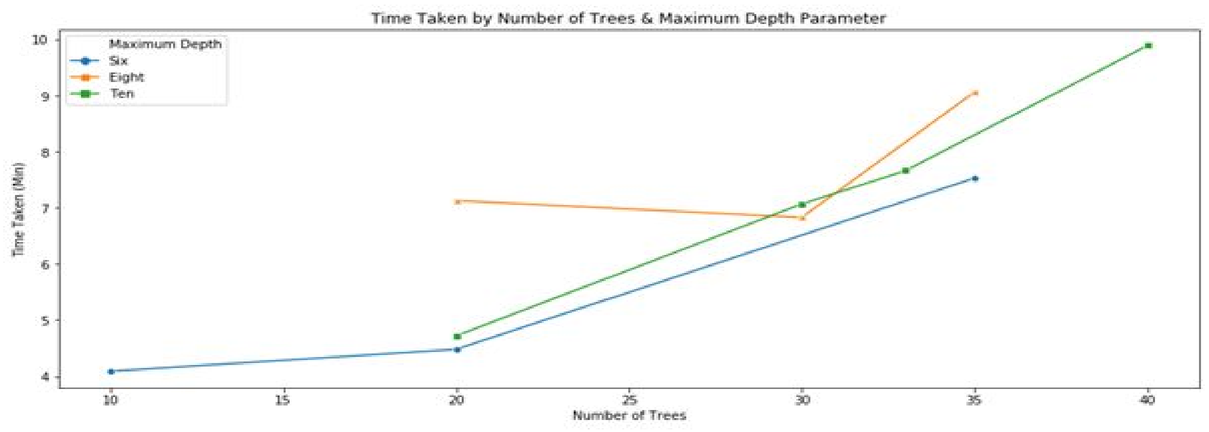
No	Number of Trees	Maximum Depth	Number of Features	Bootstrap Size	Time Taken (Min)	Accuracy Percentage (%)
1	10	6	28 (√total features)	800	4.09	77.55
2	20	6	28 (√total features)	800	4.48	78.1
3	35	6	28 (√total features)	800	7.53	76.4
4	20	8	28 (√total features)	800	7.13	78.8
5	30	8	28 (√total features)	800	8.47	80.05
6	35	8	28 (√total features)	800	9.06	79.85
7	20	10	28 (√total features)	800	4.72	79.75
8	30	10	28 (√total features)	800	7.07	80.8
9	33	10	28 (√total features)	800	7.66	81.35

10	40	10	28 (√total features)	800	9.89	81
11	30	12	28 (√total features)	800	12.3	81
12	20	6	28 (√total features)	1200	8.86	77.5
13	20	6	28 (√total features)	400	2.52	76.6
14	20	6	18	800	2.98	76.4
15	20	6	40	800	7.01	77.6

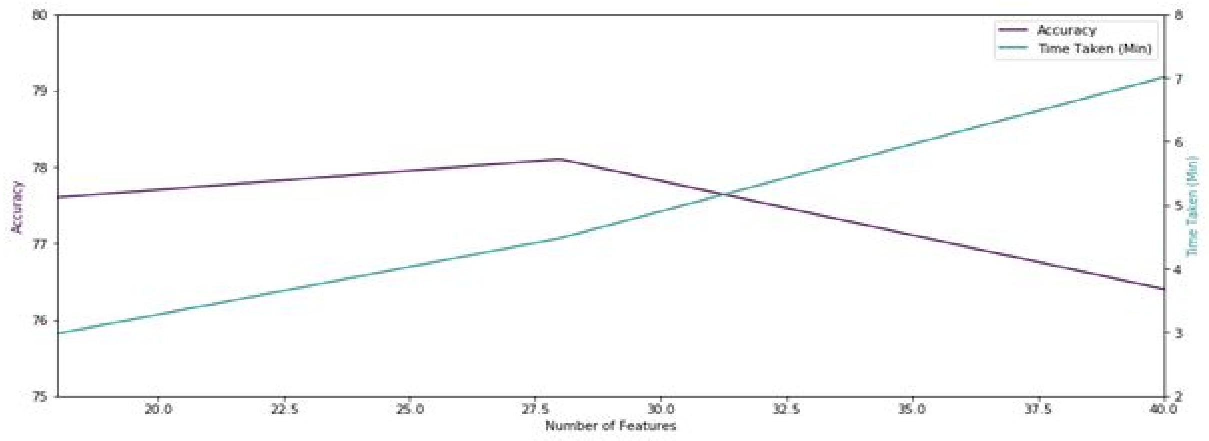
Performance evaluation by accuracy and run time



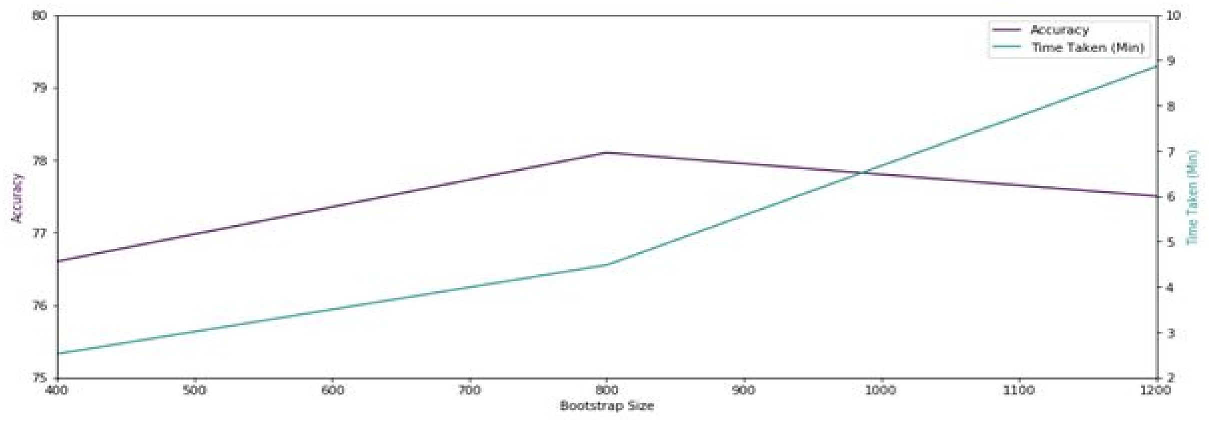
By looking at the accuracy patterns, it’s clear that increase in number of trees and maximum depth tends to make the model perform well. At some point, we can observe that the increase in number of trees starts to reduce the prediction by a slight margin, thus choosing the number of trees before that point to our model.



As the number of trees and maximum depth increases, the computational time taken for the model is really expensive.



By looking at the change in number of features taken by each tree, the performance time increases as the features increases. The optimal number of features to be taken was found to be 28 which is the square root of the total number of features (√784), which gives decent accuracy and computational time for the model.



As the bootstrap size increases, the computational time increases because each tree has to train more samples individually, making the model slow in performance. The optimal bootstrap size was found to be 800 because it gave the ability to the model in giving more accuracy with 20 to 35 trees within 10 minutes.

Optimal parameters for random forest classifier

By doing comparison analysis of all the models with different parameter tuning, the most optimal and accurate random forest model was found to be with parameters:

Number of Trees: 33

Maximum Depth: 10

Number of Features: 28 (√total features)

Bootstrap Size: 800

The performance of the model:

Accuracy: 81.35%

Time Taken: 7.66 Minutes

Model comparison and conclusion

Both the random forest classifier and logistic regression perform reasonably well with the entire dataset, although the computation time for the former would be higher to attain the same level of accuracy as that of the logistic regression. For logistic regression,Percentage difference in accuracy between validation set (70%-30%) and test set is only 2.24 percentage points - this could mean that underfitting/overfitting is less likely to occur

At a later stage, kNN was implemented, although it failed to train the data in under ten minutes, confirming the hypothesis that k nearest neighbour doesn’t work well with high dimensionality data.

Dimensionality reduction techniques like PCA and t-SNE could help to boost the run time performances of more complex algorithms like XGboost and support vector machines.