



THE UNIVERSITY OF  
SYDNEY

# COMP 5318 Assignment 1 Group-91

Tutors: Fengxiang He, Shaojun Zhang

Group Members: Sushil Kulkarni (470452709), Disha Mendapara  
(490494736), Anjali Upadhyay (490451474)

Report on comparison of classification methods  
using FashionMnist dataset

**School of Computer Science**  
Faculty of Engineering & IT

**ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT**

Unit of Study: COMP 5318 Machine learning and Data mining

Assignment name: Assignment1

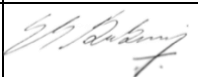
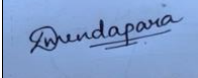
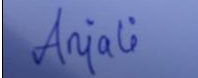
**Tutors: Fengxiang He, Shaojun Zhang**

**DECLARATION**

We the undersigned declare that we have read and understood the [University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy](#), and, except where specifically acknowledged, the work contained in this assignment/project is our own work, and has not been copied from other sources or been previously submitted for award or assessment.

We understand that failure to comply with the *Academic Dishonesty and Plagiarism in Coursework Policy* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

We realise that we may be asked to identify those portions of the work contributed by each of us and required to demonstrate our individual knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Project team members				
Student ID	Student ID in Words	Participated	Agree to share	Signature
470452709	Four seven zero four five two seven zero nine	Yes/No	Yes/No	
490494736	Four nine zero four nine four seven three six	Yes/No	Yes/No	
490451474	Four nine zero four five one four seven four	Yes/No	Yes/No	

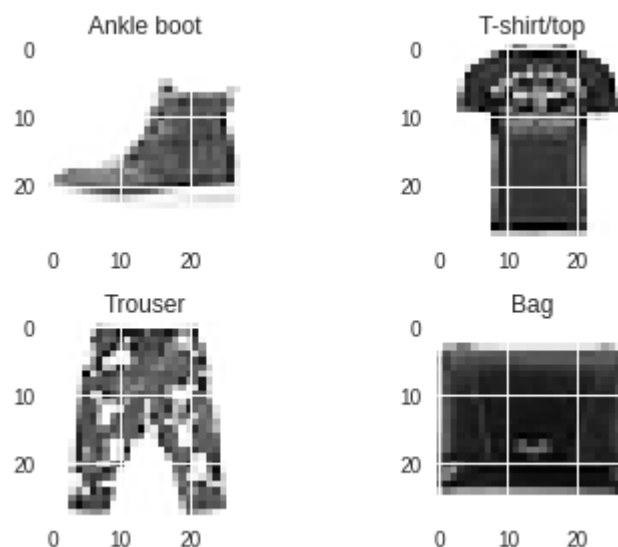
1. Abstract:	4
2. Introduction:	4
3. Preliminary data analysis:	5
Preprocessing of Input Data:	5
4. Methods:	7
KNN:	7
Logistics regression:	8
Naive Bayes:	10
Helper functions:	10
5. Experiments and results:	11
Hardware & Software:	11
Performance metrics:	11
Extensive Analysis:	11
KNN:	11
Logistic Regression:	13
Naive Bayes:	14
6. Personal reflection:	15
7. Conclusion:	16
8. References:	16

## 1. Abstract:

We performed classification on Fashion MNIST dataset using 3 techniques taught in class i.e. KNN, Naive Bayes & Logistic regression without using any advanced libraries such as sklearn. We have implemented PCA on the given dataset to reduce the size by 48.9%. KNN provided the highest accuracy followed by Logistics and NB. However, performance wise NB was the fastest, closely followed by Logistic, whereas KNN was a very distant third. Logistic Regression was found to be balanced in terms of both accuracy and speed.

## 2. Introduction:

The aim of the study to apply the Machine learning techniques taught in class to the given dataset. The given dataset (training\_data) has 30,000 rows and 28 by 28 columns. Each row indicates an image belonging to a class. There are 10 possible classes that each image belongs to. There are 5000 images reserved for testing.



Sample images from dataset

Images from dataset can be classified in the following 10 classes: 0: 'T-shirt/top', 1: 'Trouser', 2: 'Pullover', 3: 'Dress', 4: 'Coat', 5: 'Sandal', 6: 'Shirt', 7: 'Sneaker', 8: 'Bag', 9: 'Ankle boot'

The aim of this study is to apply the best techniques on the given dataset to achieve the most from the goals listed below:

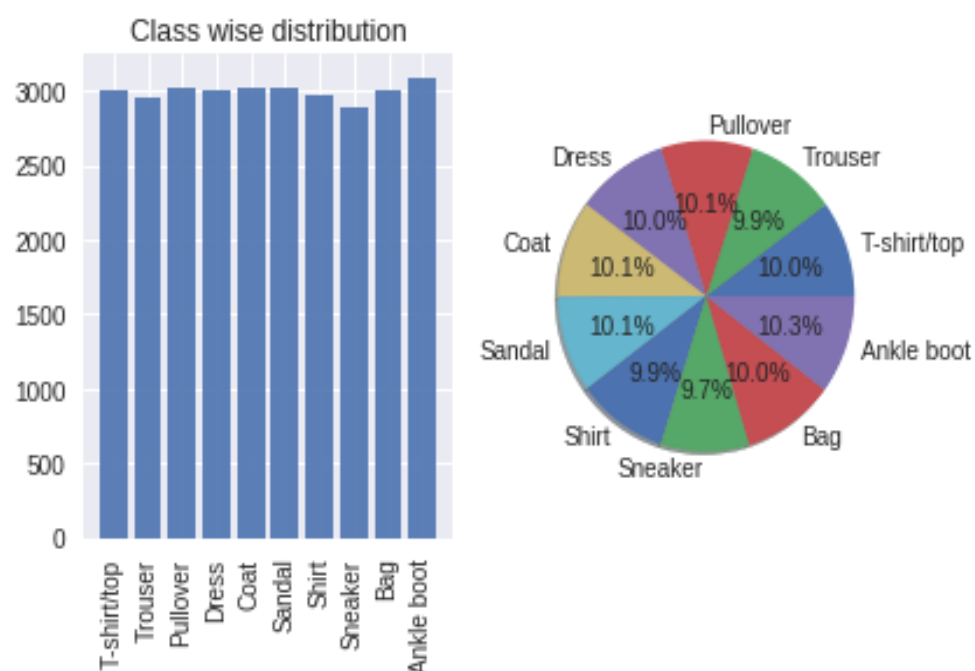
1. To simplify the given dataset by reducing the dimensions wherever possible thereby optimize the computing time and size of the dataset
2. Develop a classifier on our own to classify a given image into its category. No standard library such as "sklearn" has been used. All classifier are written from scratch.
3. To apply a classifier to the dataset which gives us high accuracy by predicting the

image to its corresponding class

4. Compare our custom classifier results and measure accuracy and performance using classification report measuring precision, recall, f1 score, etc.
5. Provide personal reflection on our findings and suggest areas of improvement wherever deemed fit.

### 3. Preliminary data analysis:

We loaded the dataset using “h5py” library and converted the data to “NumPy” thereafter. For the preliminary analysis, we created a histogram of the labels provided into classes that they belong to see the data distribution. We found that data distribution is uniform in this case and there is no class imbalance problem with the training dataset.



### Preprocessing of Input Data:

As we have flattened the data to get rid of the 3-dimension resulting in very high number of dimensions or attributes ( $28 \times 28 = 784$ ). There are 2 problems by working on the data with such huge dimensions.

1. Processing time is generally slow for any kind of algorithm as the data size is huge and attributes of the data define the classification process, so the more attributes the data have the more time it will take to process the entire information
2. When we fit the model using the data with many attributes the chances of overfitting the data increase, thus making prediction our unseen data ineffective.

Considering above scenarios, we prefer to apply PCA on the data provided.

But, before applying PCA we need to center the data. So, we have calculated the mean and subtracted the data from the mean and divided it by the standard deviation. Once the data is centered we can apply for PCA.

There were two approaches thought for applying PCA.

1. Apply PCA using eigen decomposition of the covariance matrix
2. Use SVD.

We have used both the techniques as an experiment to understand both the methods and the differences between them.

Firstly, we have applied PCA using Eigen decomposition. We found the covariance of the dataset and applied eigen decomposition on covariance which is ideally represented as

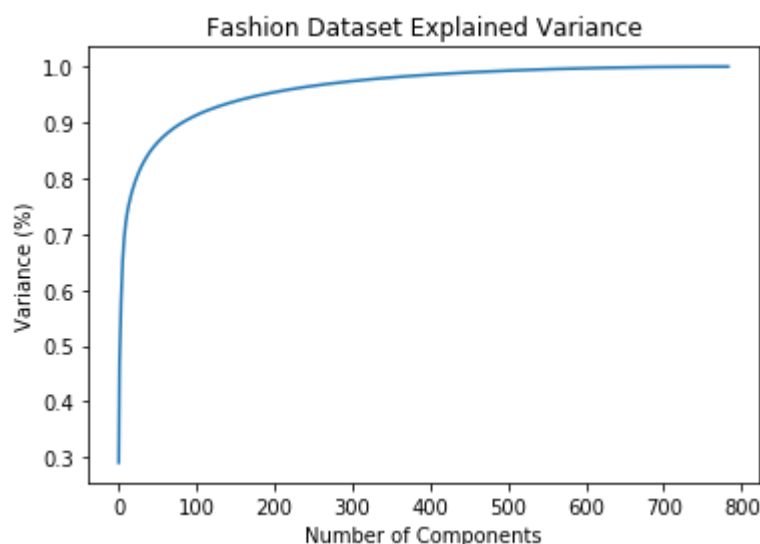
$$\text{np.linalg.eig}(X^T X)$$

This will provide the eigenvalues and eigenvectors, then sort the data in descending order.

After that important decision it to decide the no of principal components to be used for final reduced dataset.

For finding the no. of principal components we have used the predefined PCA sklearn functionality to determine the no. of principal components to be used.

**Note:** sklearn library has been used only to find optimum variance value. No other library or function is used to calculate PCA.



The above graph shows that after 400 components the data almost flattens and the variance is more than 95%, hence we concluded that using 400 Principal component axes should be a good estimate for the original data and we will not suffer from much data loss while modelling the data.

So once we figured out no. of principal component to be taken we can run through a loop to construct a 784\*400 Matrix and then we take a dot product with the original matrix sized (27000\*784) the resulting matrix data\_tr\_PCA is of size 27000\*400.

One of the other way to run the PCA is by using SVD. There are multiple advantages of using SVD, firstly we don't lose any data as we are not performing any  $(X^T X)$  covariance and secondly it is faster than eigen decomposition as we do not calculate the entire data and then filter out the required principal components required.

```
U, s, V = np.linalg.svd(data_tr, full_matrices=False)
```

There V is same as loadings which we create in PCA.

## 4. Methods:

### KNN:

KNN is considered as one of the most used classifier algorithms. It is used to predict the class of a data point using the data stored in different classes. It is mainly used as a classifier algorithm when the sample data is less than 100k.

The k-NN prediction of the query instance is mainly dependent on simple majority of all the nearest neighbors in the sample data.

#### Breaking down the KNN algorithm used:

1. We have loaded all the training data(data\_trn) and training labels(lbl\_trn) in the **KNearestNeighborClassifier()** through **fit** function.
2. We then use the **\_\_init\_\_()** function to initialize a value of neighbors
3. In this step we do multiple things:  
**calculating\_distance:**

We calculate the distance between the test data and each of the training data. Here we are using Euclidean or Manhattan distance based on our choice of two events but the default distance calculation method set is Euclidean. We use Manhattan only when the different dimensions are not comparable.

For calculating Manhattan Distance between vector x and y, the formula is (Considers the smallest absolute value as the nearest neighbour):

$$d(p, q) = \sum_{i=1}^n |p_i - q_i|$$

For calculating Euclidean Distance between vector x and y, the formula is:

$$d(p, q) = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

## Computing\_weights :

In this method we check distances of all the samples and assign a value to it as 1 if it is 0. One can choose any of the following ways to predict the class of the query instance. We have used two weights one is 'Uniform' and 'distance', the explanation of it is given as:

### ❖ Uniform-

If only uniform weights are taken into consideration, it will produce the class with more frequency as the nearest neighbour. Which in real scenario might be far away from the data that is to be tested. For an instance our `lbl_tst` has data as `[1,2,1,2,1,1,1,1,0]` it will consider 1 as the nearest neighbour as more data with value as 1 is present, whereas 0 or 2 might be more closer and correct. So to rectify this we have used distance as a weight.

### ❖ Distance-

In a scenario when the distance is 0, which means either the two points are very close to each other or have the exact same features. So when distance is calculated, it will give value as 0, which might happen sometimes when we pass the same data in the fit and predict functions or when we have a big sample data. Hence, when one distance is zero we forget about its weighted distance and assign its distance value as 1

## Predicting\_one\_instance:

In this function the samples are sorted by a pair of distance and its target class with reference to first element. Separated as Weight by class to predict the class of the data (multiclass classification). Sum is used to count how many members belong to each class then takes the majority with `max()` (looks at first element by default and second element as a predicted class).

## Predicting\_multiple\_instances:

In this function `predicting_one_instance` is called several times to predict its instance and then it is saved to the list.

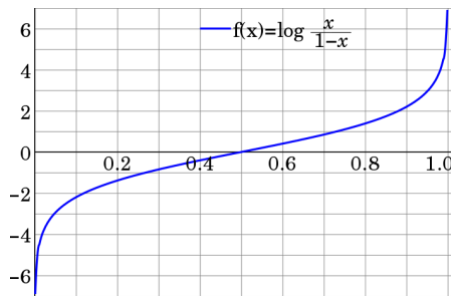
## Logistics regression:

Logistic regression is also one of the most popular Machine learning techniques used for binary classification. However, the same can be extended to multiclass problems as well. Its fundamental concepts lay a foundation to further complex deep learning algorithms. Logistics regression work by mapping one dependent binary variable in target to independent variables in training.

Logistic Regression especially works where the target variable is categorical in nature. It uses a log of odds to derive the target variable/class. For predicting binary event Logistic Regression make use of the 'logit' function in statistics.

The logit function or the log-odds is the logarithm of the odds  $p/(1 - p)$  where  $p$  is the probability. The Logit function maps out the probability values from -1 to +1. In the picture below, it can be seen the curve show inverse of sigmoidal "logistic" function or logistic transform used in statistics.





From the curve above one can see that if the curve goes to positive infinity,  $y$  predicted will become 1, and if the curve goes to negative infinity,  $y$  predicted will become 0, threshold being set to 0.5. So anything above (0.5 inclusive) that is set to be 1 and vice versa.

Logistic regression can be classified as follows:

**Binary Logistic Regression:** The target variable has only two possible outcomes such as pass or fail, guilty or not guilty, etc.

**Multinomial Logistic Regression:** The target variable has three or more classes such as in our case classes of object such as shirt, trouser, boots, etc.

In the Fashion MNIST dataset provided each image corresponds to one class out of 10 classes provided whereas logistic regression can only be used for binary classification. So be able to achieve multiclass classification, we use One vs Rest (OvR) method.

Out of 10 target classes we have, (0: 'T-shirt/top', 1: 'Trouser', 2: 'Pullover', 3: 'Dress', 4: 'Coat', 5: 'Sandal', 6: 'Shirt', 7: 'Sneaker', 8: 'Bag', 9: 'Ankle boot'). First, we compare 'T-shirt' and 'Trouser' as two class, and run a logistic regression. We store the resulting probability. Next, we take 'Trouser' and 'Pullover' and run the same. Thus, we get 10 independent logistic regressions.

So, when we predict, we will run all 10 classifiers, each one gives the probability of the class associated with it. The class which has the highest probability that class is returned as predicted class.

Logistic Regression function implements **fit**, **sigmoid**, **predict**, and **score** methods. Fit method takes training data and label to find the optimize values for learning rate, no. of iterations while minimizing the error. Sigmoid maps predicted values to probabilities as shown in the formula below.

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

Score and Predict methods take validation and test data and run through the fitted object to get predicted classes for unseen data.

During our initial research on Logistic regression we found that instead of sigmoid function we can use SoftMax function to solve multi-class classification tasks. SoftMax has the following components: a **score function** that maps the raw data to class scores, and a **loss function** that quantifies the agreement between the predicted scores and the ground truth labels. **cost function** to measure our errors with expected outcomes. And NN style **backpropagation function** to calculate the partial derivatives based on the loss function. It is done with respect to any weights/bias added in the network

## Naive Bayes:

Naive Bayes is one of the simplest yet powerful supervised way to classify the data. NB works on the Bayes theorem of probability.

The formula is:

$$P(A|B) = P(A) P(B|A)/P(B)$$

$P(A|B)$ -Probability of A occurrence if the B event has already occurred.

$P(A)$  and  $P(B)$  are the probabilities of the individual events

.

Naive Bayes works very well for the multi class classification, which is our present data. We have 10 individual classes to be classified. It works on the principal of the probability. We are trying to get the probability of the new data with respect to the existing data and finding out where the new data will fit in. The Naive Bayes algorithm assumes that there is a linear independence between variables.

We have 3 flavours of Naive Bayes

- 1) Multinomial NB- We use this for the text classification, counting the occurrence of the word in the document.
- 2) Gaussian NB- Assumes we have normal distribution of the data
- 3) Bernoulli NB: We use this for binary data, when our data just have binary values

We are using Gaussian NB, as the data is continuous.

To implement the functionality in the Python we have to take mean and the covariance of the matrix.

Before that we have to separate the data with respect classes so that it is easier to calculate mean and variance of the data.

Calculating the mean and variance of the data group by all the classes. So, for 10 classes we will have  $10 \times (768,768)$  values. We will calculate mean and variance for all the variables, and along with this we will calculate the probability of the all the individual classes i.e.  $\text{count}(c)/\text{total entries}$ .

```
self.prob_density=np.bincount(label)/len(label)
```

This data will represent our model to train the new data. Once we have new data we calculate probability density function for the newly arrived data.

We can calculate this based on the mean, variance calculated initially.  
`multivariate_normal.logpdf(data, mean, variance)`.

We can calculate this for all of the test and validation data.

## Helper functions:

Since we were not allowed to use advanced libraries like sklearn we had to implement our own helper functions to calculate metrics, formatting/printing classification report, etc.

We implemented the following 4 functions to the above purpose:

1. Precision\_recall\_fscore\_support\_Custom(): to calculate precision, recall, fscore using actual vs predicted classes.
2. Classification\_report\_Custom(): print the results into a formatted classification report
3. Inter\_class\_precision\_Custom(): plot class wise precision using actual vs predicted classes
4. LabelEncoder\_Custom(): sort and number the labels for the classes provided e.g. shirt, boot, etc. for printing and formatting

## 5. Experiments and results:

### Hardware & Software:

We have used Google Colaboratory online notebook for the purpose of testing and benchmarking all 3 algorithms created by the team members. Since google Colab provides uniform hardware/software configuration it was easy to compare the results and benchmark them.

Software: Python 3.0 notebook

Hardware: Google colab GPU with 12 GB RAM

### Performance metrics:

	Logistic Regression (PCA using 400 dims)	KNN (PCA using 400 dims)	Naive Bayes (PCA using 400 dims)
Overall Accuracy	75.05%	83%	70.50%
Time Taken	22.75 secs	5761.202 secs	0.13 secs

### Extensive Analysis:

After writing the code and performing validation for all 3 classifiers we applied extensive analysis, regularization and optimization techniques to get the best accuracy and performance over the test dataset provided. The results of the same are explained below:

#### KNN:

We began training k-NN with PCA as 40 and vary the results based on the different neighbour values.

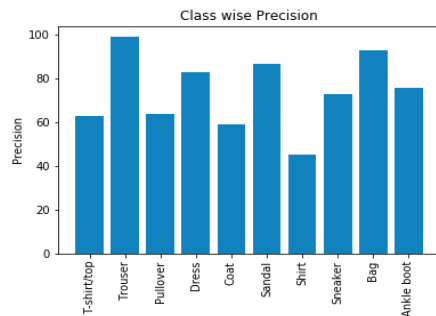
1. Using PCA as 40, weights= uniform and neighbors as 10, 1 and 6 respectively from left to right.

```

Val Accuracy: 0.756
Training Time 776.164711secs
Test Accuracy: 0.732
Training Time 1291.279115secs

```

	precision	recall	f1-score
T-shirt/top	0.63	0.77	0.69
Trouser	0.99	0.92	0.95
Pullover	0.64	0.64	0.64
Dress	0.83	0.77	0.80
Coat	0.59	0.64	0.61
Sandal	0.87	0.61	0.72
Shirt	0.45	0.42	0.43
Sneaker	0.73	0.87	0.80
Bag	0.93	0.80	0.86
Ankle boot	0.76	0.92	0.83
avg / total	0.74	0.74	0.73

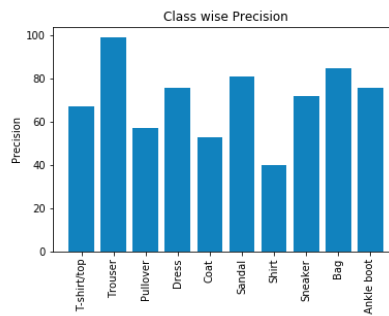


```

Val Accuracy: 0.7226666666666667
Training Time 769.068762secs
Test Accuracy: 0.698
Training Time 1282.542170secs

```

	precision	recall	f1-score
T-shirt/top	0.67	0.71	0.69
Trouser	0.99	0.95	0.97
Pullover	0.57	0.59	0.58
Dress	0.76	0.73	0.75
Coat	0.53	0.52	0.52
Sandal	0.81	0.61	0.70
Shirt	0.40	0.44	0.42
Sneaker	0.72	0.80	0.76
Bag	0.85	0.80	0.82
Ankle boot	0.76	0.87	0.81
avg / total	0.71	0.70	0.70

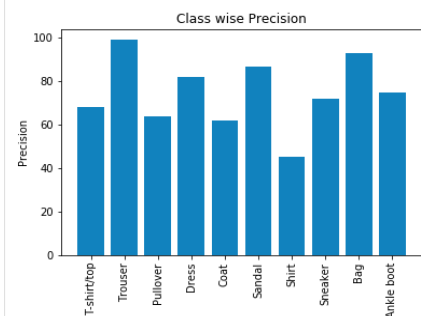


```

Val Accuracy: 0.7523333333333333
Training Time 767.018528secs
Test Accuracy: 0.735
Training Time 1279.425710secs

```

	precision	recall	f1-score
T-shirt/top	0.68	0.76	0.72
Trouser	0.99	0.94	0.96
Pullover	0.64	0.58	0.61
Dress	0.82	0.76	0.79
Coat	0.62	0.63	0.62
Sandal	0.87	0.60	0.71
Shirt	0.45	0.51	0.47
Sneaker	0.72	0.85	0.78
Bag	0.93	0.83	0.88
Ankle boot	0.75	0.94	0.83
avg / total	0.75	0.74	0.74



The above results shows that there is a huge difference between the accuracy between choosing 1 neighbour vs 6 neighbour, though there is a slight change in the number after increasing the no. of neighbours.

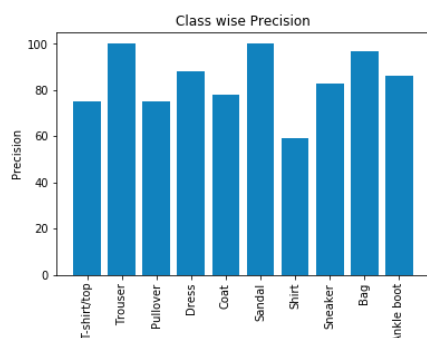
## 2. Using PCA =400, weights=uniform and neighbors as 6

```

Val Accuracy: 0.8456666666666667
Training Time 3457.908513secs
Test Accuracy: 0.8385
Training Time 5761.206120secs

```

	precision	recall	f1-score
T-shirt/top	0.75	0.83	0.79
Trouser	1.00	0.97	0.98
Pullover	0.75	0.79	0.77
Dress	0.88	0.88	0.88
Coat	0.78	0.77	0.77
Sandal	1.00	0.74	0.85
Shirt	0.59	0.54	0.56
Sneaker	0.83	0.97	0.90
Bag	0.97	0.96	0.97
Ankle boot	0.86	0.96	0.91
avg / total	0.84	0.84	0.84



The time taken to run for PCA=400 is way more but we have achieved better accuracy.

## Logistic Regression:

We began training Logistic regression with 50 iterations taking it to 500. However we found the accuracy degrades after 100 iterations hence was found 100 to be optimal in this case.

Results with iterations 50, 100 and 150

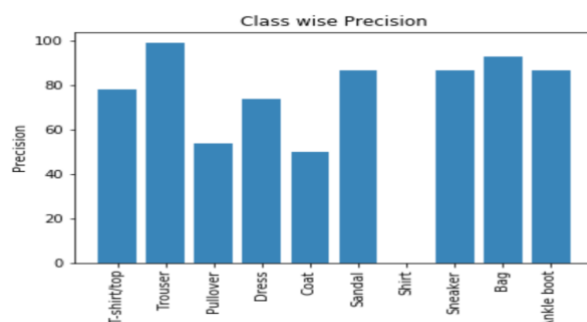
0.6311851851851852 Training Time 12.012983secs Val Accuracy: 0.616 Test Accuracy: 0.622	0.7536296296296296 Training Time 22.120760secs Val Accuracy: 0.7466666666666667 Test Accuracy: 0.7505	0.6956666666666667 Training Time 33.254585secs Val Accuracy: 0.6933333333333334 Test Accuracy: 0.6855
precision recall f1-score	precision recall f1-score	precision recall f1-score
T-shirt/top 0.82 0.55 0.66	T-shirt/top 0.78 0.69 0.73	T-shirt/top 0.72 0.73 0.72
Trouser 0.99 0.92 0.95	Trouser 0.99 0.95 0.97	Trouser 0.94 0.95 0.95
Pullover 0.00 0.00 0.00	Pullover 0.54 0.76 0.63	Pullover 0.42 0.94 0.58
Dress 0.62 0.87 0.73	Dress 0.74 0.84 0.79	Dress 0.68 0.93 0.79
Coat 0.34 0.96 0.50	Coat 0.50 0.84 0.62	Coat 0.65 0.24 0.35
Sandal 0.63 0.72 0.67	Sandal 0.87 0.80 0.83	Sandal 0.99 0.41 0.58
Shirt 0.00 0.00 0.00	Shirt 0.00 0.00 0.00	Shirt 0.00 0.00 0.00
Sneaker 0.65 0.96 0.78	Sneaker 0.87 0.88 0.88	Sneaker 0.62 0.96 0.75
Bag 0.84 0.79 0.81	Bag 0.93 0.83 0.88	Bag 0.87 0.91 0.89
Ankle boot 0.93 0.44 0.60	Ankle boot 0.87 0.91 0.89	Ankle boot 0.83 0.82 0.83
avg / total 0.58 0.62 0.57	avg / total 0.71 0.75 0.72	avg / total 0.67 0.69 0.64

Similarly, the learning rate starting at 0.1 taking it to 0.9. We found that steadily increasing the learning rate give us better accuracy with each run. Highest accuracy was achieved with learning rate of 0.9.

Results with ETA 0.1, 0.5 and 0.9

0.7127407407407408 Training Time 22.828824secs Val Accuracy: 0.7026666666666667 Test Accuracy: 0.7045	0.742 Training Time 22.735827secs Val Accuracy: 0.7293333333333333 Test Accuracy: 0.731	0.7536296296296296 Training Time 22.120760secs Val Accuracy: 0.7466666666666667 Test Accuracy: 0.7505
precision recall f1-score	precision recall f1-score	precision recall f1-score
T-shirt/top 0.44 0.95 0.60	T-shirt/top 0.59 0.87 0.70	T-shirt/top 0.78 0.69 0.73
Trouser 0.98 0.87 0.93	Trouser 0.87 0.96 0.91	Trouser 0.99 0.95 0.97
Pullover 0.73 0.57 0.64	Pullover 0.51 0.80 0.62	Pullover 0.54 0.76 0.63
Dress 0.93 0.41 0.57	Dress 0.93 0.48 0.63	Dress 0.74 0.84 0.79
Coat 0.44 0.91 0.60	Coat 0.56 0.67 0.61	Coat 0.50 0.84 0.62
Sandal 0.87 0.79 0.83	Sandal 0.83 0.80 0.82	Sandal 0.87 0.80 0.83
Shirt 0.00 0.00 0.00	Shirt 1.00 0.03 0.05	Shirt 0.00 0.00 0.00
Sneaker 0.87 0.91 0.89	Sneaker 0.83 0.89 0.86	Sneaker 0.87 0.88 0.88
Bag 0.96 0.75 0.84	Bag 0.90 0.91 0.90	Bag 0.93 0.83 0.88
Ankle boot 0.87 0.91 0.89	Ankle boot 0.86 0.93 0.89	Ankle boot 0.87 0.91 0.89
avg / total 0.71 0.71 0.68	avg / total 0.79 0.73 0.70	avg / total 0.71 0.75 0.72

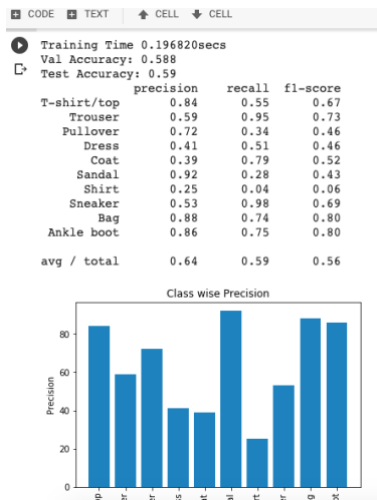
Overall it can be stated that Logistic regression the training time is found to be modest and provides the best overall accuracy of 75.05% using PCA.



0.7536296296296296 Training Time 22.214900secs Val Accuracy: 0.7466666666666667 Test Accuracy: 0.7505	precision recall f1-score
T-shirt/top 0.78 0.69 0.73	
Trouser 0.99 0.95 0.97	
Pullover 0.54 0.76 0.63	
Dress 0.74 0.84 0.79	
Coat 0.50 0.84 0.62	
Sandal 0.87 0.80 0.83	
Shirt 0.00 0.00 0.00	
Sneaker 0.87 0.88 0.88	
Bag 0.93 0.83 0.88	
Ankle boot 0.87 0.91 0.89	
avg / total 0.71 0.75 0.72	

## Naive Bayes:

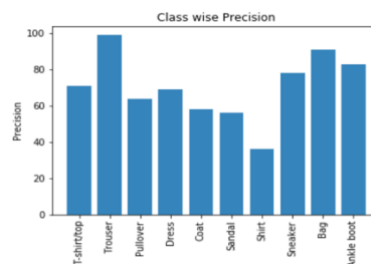
As there are no hyper parameters defined for the NB algorithm. Normalization of the data i.e. subtracting the mean and then taking the PCA can improve accuracy by a substantial amount, prior to applying the PCA below was the accuracy of the NB.



**Before PCA**

Training Time 0.135796secs  
Val Accuracy: 0.704  
Test Accuracy: 0.705

	precision	recall	f1-score
T-shirt/top	0.71	0.70	0.70
Trouser	0.99	0.90	0.94
Pullover	0.64	0.50	0.56
Dress	0.69	0.83	0.75
Coat	0.58	0.63	0.60
Sandal	0.56	0.73	0.63
Shirt	0.36	0.24	0.29
Sneaker	0.78	0.84	0.81
Bag	0.91	0.80	0.85
Ankle boot	0.83	0.90	0.86
avg / total	0.70	0.71	0.70



**After PCA**

The accuracy of NB is raised by 10% after reducing the dimensionality of the data and normalizing it.

The precision and recall is really less for the skirt is really low and for trouser it is really high.

## 6. Personal reflection:

We initially as a team had lot of discussion around the choice of 3 algorithms. We decided to use Naive Bayes and KNN and between Neural Network and Logistic regression we choose LR basically as it forms the stepping stone in understanding the Neural Networks.

The implementation of all the algorithms is very different and it gave all of us to get a flavour of how the different algorithms run and their mode of implementation.

We started with Naive Bayes and figured out this is the fastest algorithm to work on the dataset. NB can be preferred way to show an accuracy in a very short time. Suppose we need to present a report within an hour about the classification dataset which we have just received and the team wants to know about the basic understanding on the data, NB can be proved really helpful however since it assumes independence among the variables it can't be widely used as most of the data in the real world has some inter dependence between variables. Suppose, we have test data for which the class is not present in the training data the probability of this 'unknown' class turns out to be 0.

So, we started our assignment by creating a Naive Bayes algorithm and measuring the first set of accuracy which we can achieve on this dataset. We hoped that using other algorithms should give us better than that what we have already achieved with NB.

The next algorithm we started working was Logistic regression, we used one-vs-all multiclass classification to implement LR. After running the algorithm for multiple iterations, we figured out that we can increase the efficiency of the algorithm a lot by increasing the learning rate and tuning the efficiency via no. of iterations.

After implementing one-vs-all multiclass classification we also figured out we can also implement LR using cross entropy loss and SoftMax regression.

KNN, an easy to understand and implement algorithm, though because of the basic nature of the algorithm it proved to be very time-consuming algorithm while working on the bigger dataset like Fashion MNIST.

Therefore, PCA proved to be most applicable for KNN as the processing time might vary a lot depending upon the no. of variables we have. The most important parameter for the KNN is the no of neighbours on which we compute the distances. It is also known as lazy loading as the algorithm doesn't really do any 'fitting' on the training data.

**One common observation which we have seen from the dataset provided is that for all the algorithms the precision and recall is extremely high for Trouser and Bag Class whereas both of them are extremely low of Shirt Class.**

## 7. Conclusion:

As it is usually said in Machine learning terms, there is no free lunch provided, so we need to try multiple algorithms and compare their results to choose the one which fits best for the data provided.

As the conclusion, though we can achieve high accuracy using KNN but Logistic regression is the ideal algorithm for the data set provided as the processing time is modest and we can achieve high accuracy using LR.

## 8. References:

1. <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering>
2. <https://www.ijcai.org/proceedings/2017/0368.pdf>
3. [https://rstudio-pubs-static.s3.amazonaws.com/337306\\_79a7966fad184532ab3ad66b322fe96e.html](https://rstudio-pubs-static.s3.amazonaws.com/337306_79a7966fad184532ab3ad66b322fe96e.html)
4. [https://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)
5. <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>
6. <https://blog.usejournal.com/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7>
7. <https://datascience.stackexchange.com/questions/20075/when-would-one-use-manhattan-distance-as-opposite-to-euclidean-distance>
8. <https://elitedatascience.com/machine-learning-algorithms>
9. <https://sebastianraschka.com/faq/docs/naive-bayes-vs-logistic-regression.html>
10. <https://www.syntelli.com/demystifying-naive-bayes-classifier>
11. <https://en.wikipedia.org/wiki/Logit>