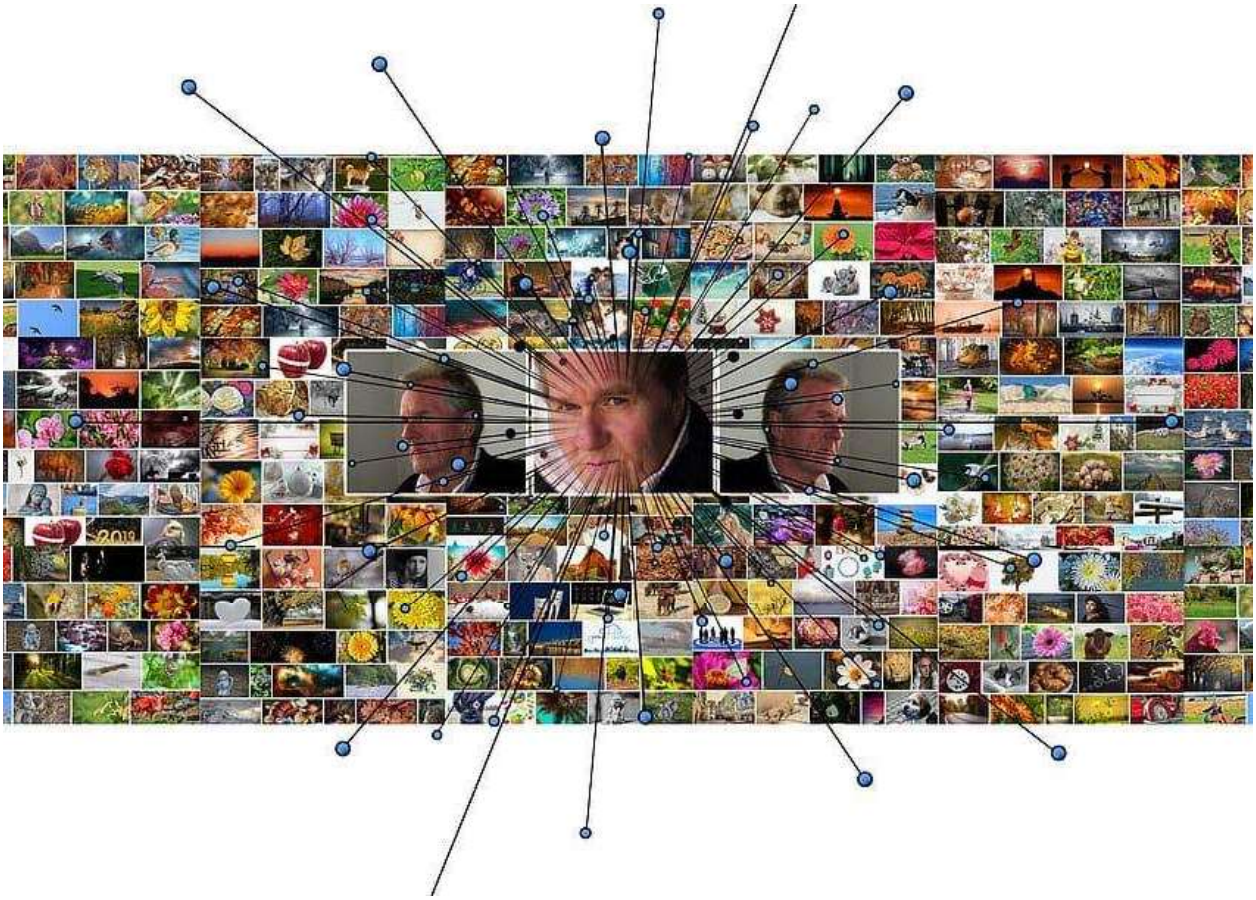
STAT 542

FINAL PROJECT

FASHION MNIST- IMAGE CLASSIFICATION PROBLEM



Group Members

Chirag Ranjan (cranjan2@illinois.edu)

Debapratim Ghosh (dg19@illinois.edu)

Shantanu Solanki (solanki7@illinois.edu)

STAT 542 Project Final Report

Project Description and Summary

Fashion MNIST dataset is a classic dataset used to analyze machine learning models on image recognition. Our objective is to implement several supervised and unsupervised machine learning models on dataset and attain better accuracy in classifying the images.

We started with some unsupervised learning models to cluster the data, using two clustering models, namely Hierarchical Agglomerative Clustering with Ward's Linkage and Gaussian Mixture Model. The objective was to find clusters that could classify the images. There were 7-8 clusters obtained in the two models. While some clusters had one dominant label, other clusters also had those labels, eg. bags dominated cluster 2 but also had sizeable representation in cluster 1. Though there was no dominant label in most of the clusters using both the methods, HAC performed marginally better than GMM on the basis of recovering the labels and cohesiveness of the observations. We also did Principal Component Analysis on the dataset and found that, though the dataset has 784 variables (pixel values) for each observation, 90% of the variation is explained by just 85 principal components. This implies some of the models, like KNN, which face severe problems in case of high-dimensional data, will perform reasonably well, since the data resides in a smaller dimensional subspace of the full column space.

Since our dataset has multiple classes, we started with implementing two multi-class classification models, namely Gaussian Naive Bayes and Random Forest Classifier and tune the parameters to give us best results. Additionally, we also used KNN, with tuned number of nearest neighbors. Then, we extended a binary classifier, i.e. SVM, to perform multi-class classification by using one versus one comparison of labels to reduce it to multiple binary problem. From the performance metrics, we observed that SVM performs better than the previous two models in terms of accuracy.

An important part of our work was to stack the individual models into a meta-model to achieve better performance. This was done by using the output labels from four learning models, namely, K-nearest neighbors, Random Forest Classifier, Support Vector Classifier and Light Gradient Boost, and training a meta Random Forest Classifier on these output labels as features. Having just 4 features (4 model outputs), this step is comparatively faster. This step was also tried with SVM in the stacked model. However, Random Forest performs marginally better than SVM. The overall accuracy of our stacked model comes to around 91.5%. Thus, we conclude that stacking different models can substantially improve the performance.

Literature Review

There have been extensive analyses of the Fashion MNIST dataset from the point of view of image recognition. In one of the papers, Greeshma, K. V., and K. Sreekumar[1] have used Histogram of Oriented Gradient (HOG) feature descriptor and multiclass SVM to classify the images. HOG is a simple and effective feature descriptor which describes the shape and appearance of the images. It divides the image into small cells like 4-by-4 and computes the edge directions. In this method the X and Y gradients are calculated and plotted with respect to the different cells. The principle behind the method is that the we get sharp gradients, with the values rising/falling drastically, around the edges of the pictures. The choice of the size of the cells determines the dimensionality of the data. Smaller the cell size, more features we get. For improving the accuracy the histograms can be normalized. With these features in hand, the analysis done in the paper attains an accuracy of 86.53% using SVM. In our project, we have taken on a different path, to use the original features which are much less in number than that obtained by HOG, but use stacking to improve the accuracy.

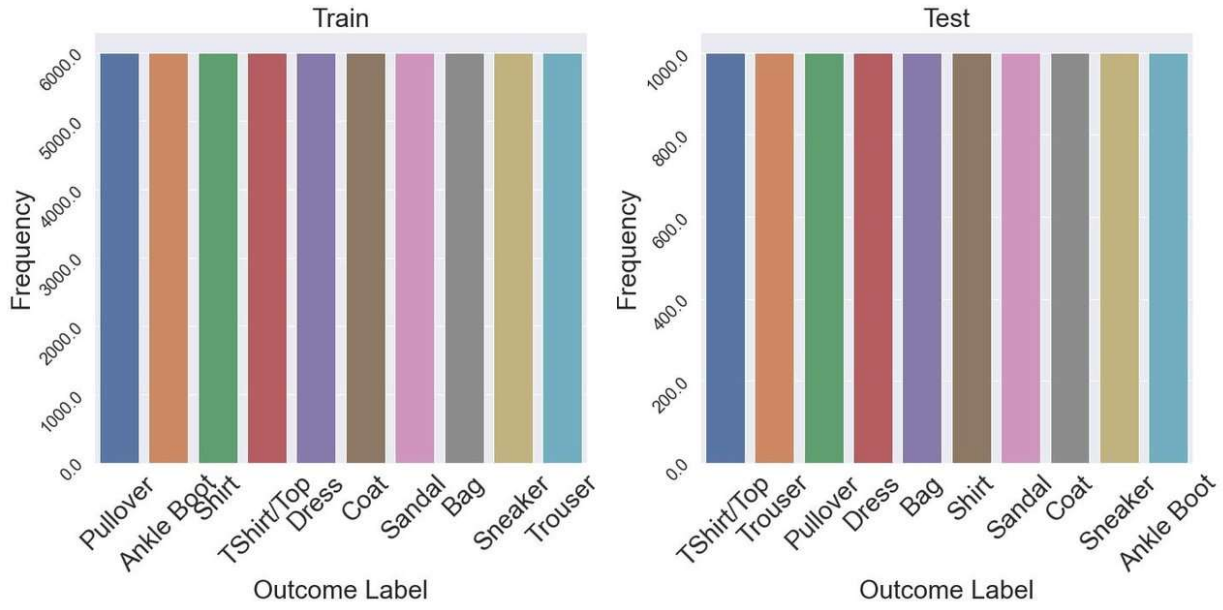
Another example we looked for inspiration came through github publication : 2020-08-19-solving-fashion-mnist-using-ensemble-learning, Thiago Melo[2]. Here, a case is made for ensemble learning as it is shown that standalone models do not have the capacity to handle the complexity of this dataset. For Ensemble Learning to work, there needs to be bias-variance

tradeoff. But, standalone models already have a high bias reducing the effectiveness. In order to solve this we either need to build on the models capacity or make the classification task easier. Different approaches are applied here : Edge Detection Transformers to make the problem simpler, although this doesn't work for cases which are difficult to classify with human intervention. Another approach is to simplify the model pixel-wise : for this PCA is used. Preserving 95% variance of the original data gives a reduction to 187px from 785 px which greatly simplifies the classification process, and the model attains accuracy ~ 90%.

Summary Statistics, data processing and unsupervised learning

FREQUENCY TABLE OF TARGET VARIABLE

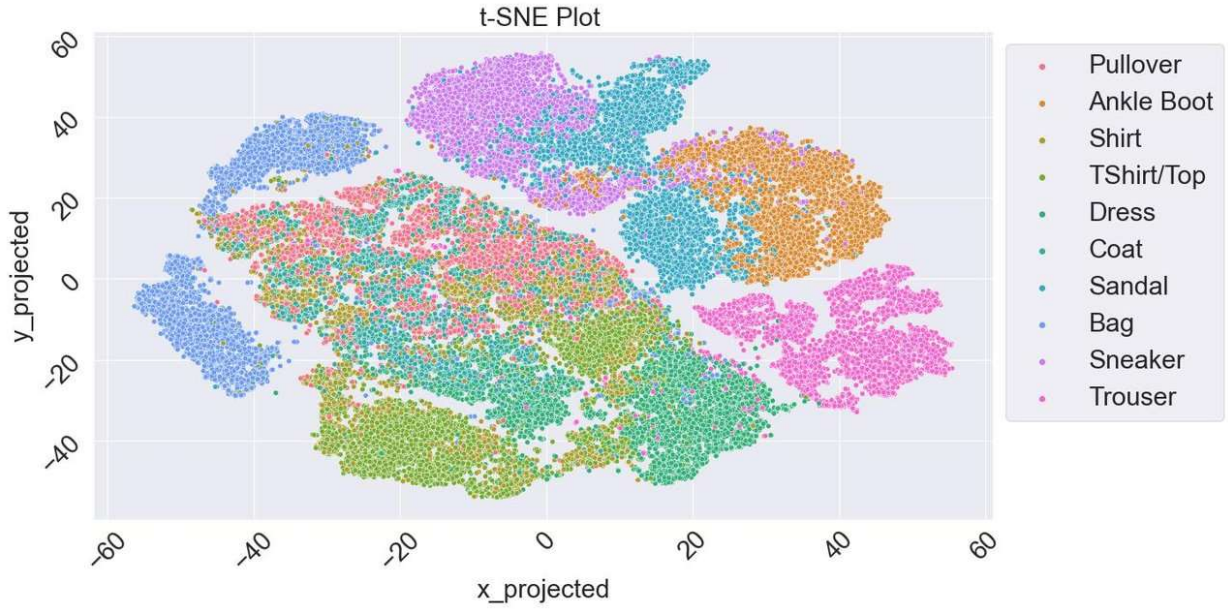
We plotted the frequency of the target variable for both the training and testing datasets . The figures are shown below



We can observe that both the Training and Testing datasets are well balanced with an equal number of images in each class label. The training dataset has 60000 images in each class (0-9) while the Test dataset has 10000 images in each class.

Since we have confirmed that the dataset is well balanced, we would like to utilise a few clustering techniques to understand how each of the features (pixels) are related to each class. Because we are dealing with a high dimensional dataset, we need to somehow understand the under-lying clustering structure in 2D . This would enable us to decide on an appropriate clustering algorithm.

To reduce the number of dimensions to 2 D while preserving the clustering structure, we would use t-Distributed Stochastic Neighbor Embedding **(t-SNE) plots.** Below is the t-SNE Plot for the training dataset colored by observed class labels



We can observe the following things :

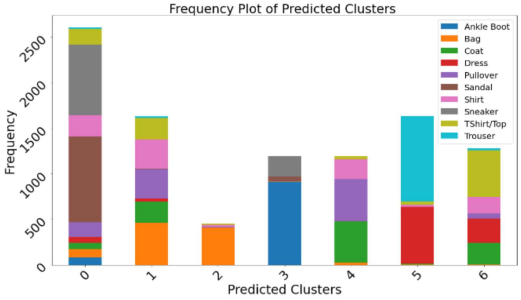
1. There are roughly 6-7 major clusters in the dataset which are highly overlapping
2. We cannot say clearly how many clusters are there as the clusters are not well separated.
3. Except for trousers and bag, almost all the clusters are merging into one another
4. T-shirt/ Top is a fashion item that might have sub-classes

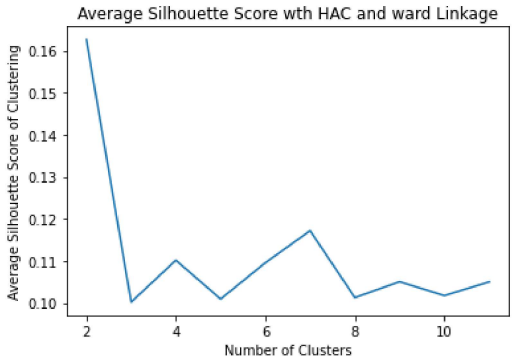
Because of the overlapping and nested nature of the underlying clustering structure of the training dataset, we would like to use Hierarchical Clustering with Ward Linkage (to identify the nested nature) and Gaussian Mixture Models (to obtain a matrix of cluster membership scores instead of **hard** partitioning)

Note- The unsupervised learning analysis was done on the test dataset as performing the same analysis on the training data was proving to be computationally expensive. This was done under the assumption that both the datasets are well balanced among the classes and the underlying distributions (or data generating processes) of the classes are identical for both datasets.

HIERARCHICAL AGGLOMERATIVE CLUSTERING

Silhouette Score is a metric that lets us know how *cohesive* and *well-separated* the clusters are after a clustering algorithm is executed. We have plotted the average silhouette score for HAC with Ward's linkage below (Plot on the left). Using this analysis, we decided to go ahead with 7 clusters for our dataset. The frequency plot of the 7 clusters is on the right.





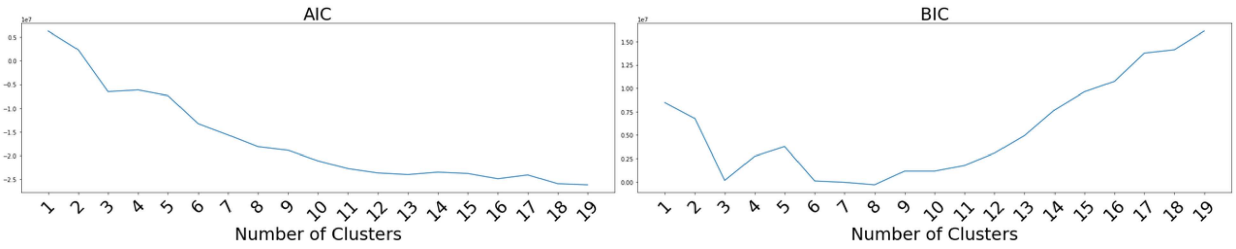
We can observe the following things :

1. The clusters are not one to one related to each fashion item label. **So the clustering algorithm was not able to identify the original class labels. This is reflected by a poor adjusted rand score of 0.314.**
2. While Cluster 2 is mostly made of Bag items there is a 50-50 split between the Trousers and Dresses in Cluster 5.
3. In cluster 3, the dominating fashion item label is Ankle Boot. The remaining clusters has an uneven distribution of the the other class labels.

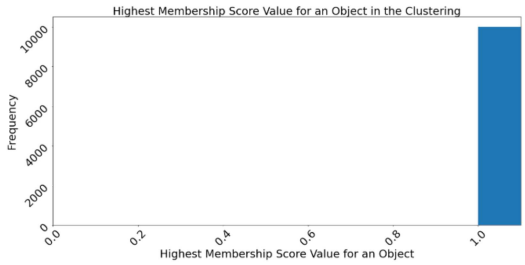
We now move forward to Gaussian Mixture Models to get a “fuzzy” partitioning of our dataset.

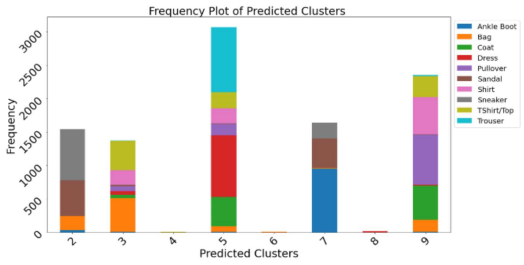
GAUSSIAN MIXTURE MODELS

Since Gaussian Mixture Models is a statistically rigorous model, we expect it to perform better than the Hierarchical Agglomerative clustering algorithm. We will assess whether this is a sensible mixture model to fit to this dataset . Let us decide the ideal number of clusters from AIC and BIC scores for different cluster numbers.



We can observe that the AIC keeps on decreasing as the number of clusters increases and the BIC has a minima at number of clusters =8 . We would be moving forward with 8 Clusters for the GMM algorithm and then assessing the clustering performance.





From the plot on the left, we can observe that the GMM clustering with 8 clusters is a hard clustering ie. there is no partial probability of belonging to different cluster.
From the plot on the right we can see that GMM wasn't able to identify the original fashion labels in the dataset. This is evident from an adjusted rand score of 0.29 which is poorer than the HAC algorithm . Moreover, the clusters 4,6 and 8 contain very less images indicating that the clustering is highly unbalanced.
Additionally, it seems that in cluster 7, Ankle Boot is the dominant fashion item label while the remaining clusters don't have a dominant fashion item label as such.

UNSUPERVISED LEARNING MODELS COMPARISON

	A	B	C
1	**Model Name**	**Adjusted Rand Score**	**Average Silhouette Score**
2	HAC with Ward's Linkage	0.314	0.117
3	GMM	0.2919	0.015

1. As compared to GMM, HAC with Ward's linkage was able to recover the original class labels better as its adjusted rand score is higher
2. As compared to GMM, HAC with Ward's linkage has much more cohesive and well separated clustering as the average silhouette score for HAC is much higher than GMM

Therefore, in case of Fashion MNIST, HAC with Ward's Linkage is a much better Clustering Algorithm.

Multi-Class Classification Model

We will begin the exercise of classifying the images by using Multi-Class Classifiers i.e. Classifiers that can inherent classify more than two classes. We will explore the performance of two such algorithms :

1. Gaussian Naive Bayes
2. Random Forest Algorithm

Next, we will look at extending a more popular binary classifier (**Support Vector Machines**) to a multi-class classification problem.

Before performing the classification exercises on the Fashion-MNIST images, we need to define a few performance metrics in order to evaluate each model and compare them against other models
The evaluation metrics we will be using are (Applied to a multi-class setting) :

1. **Accuracy** - It defines how accurate your model is. Accuracy Score = $\frac{TP+TN}{FP+FNTP+TN}$, where TP = True Positives , FP= False Positives and so on.
2. **Macro Averaged Precision-** calculate precision for all classes individually and then average them
3. **Micro Averaged Precision -** calculate class wise true positive and false positive and then use that to calculate overall precision
4. **Macro Averaged Recall** - calculate recall for all classes individually and then average them
5. **Micro Averaged Recall**- calculate class wise true positive and false negative and then use that to calculate overall recall
6. **Macro Averaged F1 Score** : calculate f1 score of every class and then average them
7. **Micro Averaged F1 Score** : calculate macro-averaged precision score and macro-averaged recall score and then take there harmonic mean
8. **Multi-class AUC Score -** The AUC Score of the predictions for each class is calculated by splitting the predictions into multiple class wise predictions and comparing them with the rest of the classes. This is similar to the one-vs-rest setting used to extend binary classifiers to multi-class problems.

GAUSSIAN NAIVE BAYES

We will begin with a simpler model ; Gaussian Naive Bayes Classifier which is able to classify multiple classes.

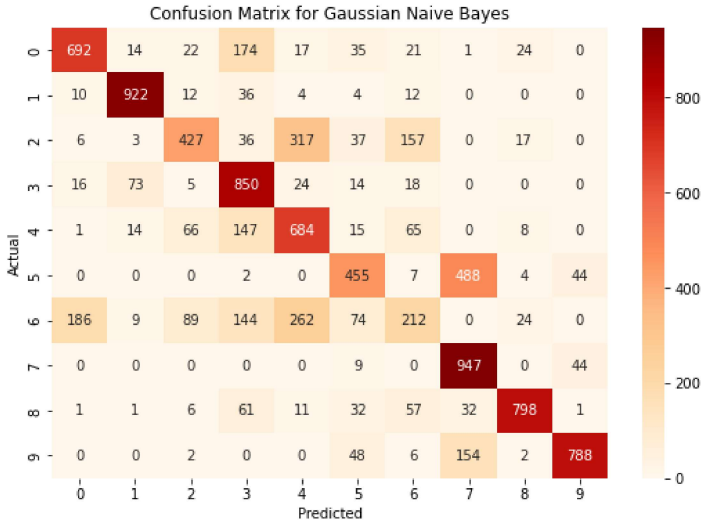
Hyper-Parameter Tuning
Since Naive bayes “Naively” multiplies all the feature likelihoods together, if any of the terms of the product is zero, it would void all other evidence and the probability of the class is going to be zero. In order to avoid zero probabilities , we add some noise in the model defined as α . We need to tune this α to get the performance out of the model. This hyperparameter tuning is done through **a 10-fold cross-validation.**

Using the var_smoothing parameter in the GridSearchCV function in python, we get the optimum α of **0.0811.**

Model Fitting Results

	A	B	C	D	E	F	G	H	I
1	**Model**	**Training Accuracy**	**Testing Accuracy**	**Macro-Averaged Precision Score**	**Micro-Averaged Precision Score**	**Macro-Averaged Recall Score**	**Micro-Averaged Recall Score**	**Macro-Averaged F1 Score**	**Micro-Averaged F1 Score**
2	Gaussian Naive Bayes	0.676	0.6775	0.6836	0.6775	0.6775	0.6775	0.6651	0.6775

Confusion Matrix



RANDOM FOREST

We will move on to Random-Forest ; which is an ensemble modelling technique consisting of multiple weak learners usually decision trees. Before we proceed any further, we need to tune certain hyperparameters to obtain the least cross-validation error.

Hyper-Parameter Tuning

There are many hyper-parameters to tune for the Random-Forest Algorithm. In the interest of computational complexity, we have decided to tune the following parameters

n_estimators : the number of trees in the forest (Search among a list of 100,200,300 and 400)
max_features : The number of features to consider when looking for the best split
"auto" **-** sqrt(number of features)
"log2"- log2(number of features)
max_depth : The maximum depth of the tree. (Search among a list of 5,10,15 and 20)

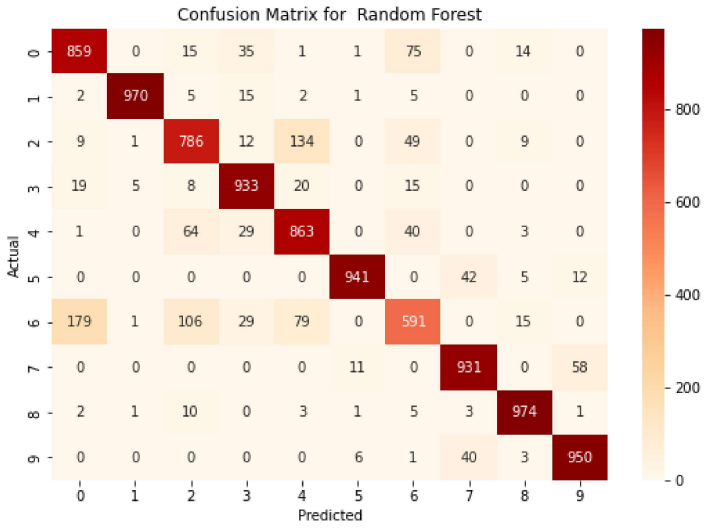
After performing **5-fold Cross Validation** over 3 iteration and using random search to find the best set of parameters, we arrived at the below set which gives the least CV error
n_estimators: 400
max_features: log2
max_depth : 15

Now we fit the RandomForest model using the above HyperParameters

Model Fitting Results

	A	B	C	D	E	F	G	H	I
1	**Model**	**Training Accuracy**	**Testing Accuracy**	**Macro-Averaged Precision Score**	**Micro-Averaged Precision Score**	**Macro-Averaged Recall Score**	**Micro-Averaged Recall Score**	**Macro-Averaged F1 Score**	**Micro-Averaged F1 Score**
2	Random Forest	0.9646	0.8798	0.8789	0.8798	0.8798	0.8798	0.8779	0.8798

Confusion Matrix



SUPPORT VECTOR MACHINES (ONE VS REST)

Extension from the Binary Classifier

Since Support Vector Classifier is a binary classifier (ie. it will be able to classify only 2 classes at a time) , we need to extend it in someway to a multiclass setting. There are two popular ways of performing this extension

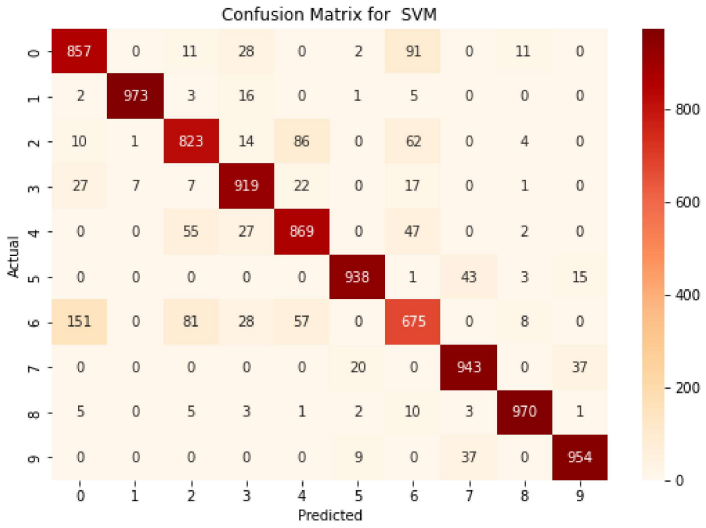
- One vs One - This approach splits the multi-class dataset into each class and then performs binary classifications against **every other class** . In our case, since there are 10 classes, there would be 2(10)(10–1) = 45 binary classifiers
- One vs Rest : Similar to One vs One, this divides the multi-class problem into multiple binary classification problems. Unlike One vs One, this approach builds a model for each class against **all other classes bundled together.** Hence there would be only 10 binary classifiers in the fashion-mnist dataset as there are only 10 classes.

It is evident one-vs-rest classification method would be less computationally expensive as lesser number of models would have to be built. We would try out One vs Rest and compute the model fitting results in terms of performance metrics.

Model Fitting Results

	A	B	C	D	E	F	G	H	I
1	**Model**	**Training Accuracy**	**Testing Accuracy**	**Macro-Averaged Precision Score**	**Micro-Averaged Precision Score**	**Macro-Averaged Recall Score**	**Micro-Averaged Recall Score**	**Macro-Averaged F1 Score**	**Micro-Averaged F1 Score**
2	SVM (One vs Rest)	0.911	0.8921	0.8915	0.8921	0.8921	0.8921	0.8915	0.8921

Confusion Matrix



MODEL COMPARISON

Below is the summary of the model performance metrics from different models we have analysed so far

Model Wise Performance Metrics

	A	B	C	D	E	F	G	H	I
1	**Model**	**Training Accuracy**	**Testing Accuracy**	**Macro-Averaged Precision Score**	**Micro-Averaged Precision Score**	**Macro-Averaged Recall Score**	**Micro-Averaged Recall Score**	**Macro-Averaged F1 Score**	**Micro-Averaged F1 Score**
2	Gaussian Naive Bayes	0.676	0.6775	0.6836	0.6775	0.6775	0.6775	0.6651	0.6775
3	Random Forest	0.9646	0.8798	0.8789	0.8798	0.8798	0.8798	0.8779	0.8798
4	SVM (One vs Rest)	0.911	0.8921	0.8915	0.8921	0.8921	0.8921	0.8915	0.8921

Class-Wise AUC Scores for each model

	A	B	C	D
1	**Classes**	**Gaussian Naive Bayes**	**Random Forest**	**SVM (One vs Rest)**
2	0 (TShirt/Top)	0.8337	0.9177	0.91766
3	1 (Trouser)	0.954	0.984	0.986
4	2 (Pullover)	0.702	0.881	0.902
5	3 (Dress)	0.891	0.959	0.953
6	4 (Coat)	0.806	0.918	0.925
7	5 (Sandal)	0.712	0.969	0.967
8	6 (Shirt)	0.586	0.784	0.824
9	7 (Sneaker)	0.936	0.96	0.966
10	8 (Bag)	0.894	0.984	0.983
11	9 (Ankle Boot)	0.889	0.971	0.974

From the confusion matrices and the performance metrics above, we can make the following observations :

- Almost all the models have trouble correctly classifying the Class 6 (Shirt). This is evident from the corresponding confusion matrices and the AUC scores listed above.
- The SVM Extension to a multi-class setting yields the highest test accuracy of 89.21%. The precision, recall and F1 Score is also high for SVM. This implies that among all the 3 models considered, SVM has the highest predictive power for the positive class label (in this case that particular class)
- SVM is also the only model which has an AUC of over 80% for the problematic class (Shirt)

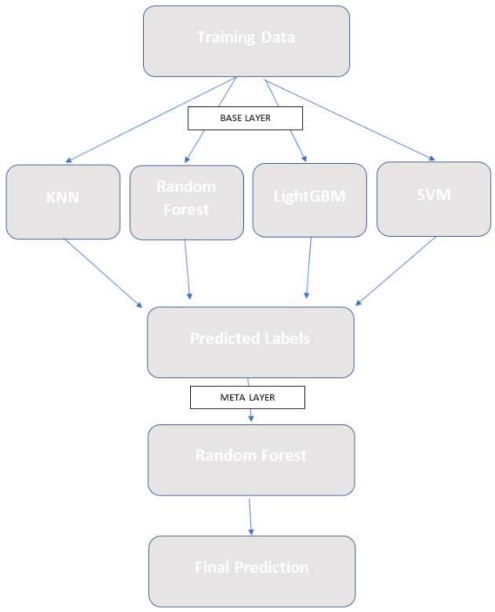
Considering the above observations, we can conclude that the **SVM extension to multiclass setting (One vs Rest) is a better model than Random Forest and Gaussian Naive Bayes.**

Ensemble Model and Feature Engineering

Base Layer: We fit the following four models (because of good accuracies) on our data in the first layer of our analysis.

1. KNN
2. Random Forest
3. Light GBM
4. SVM

Meta Layer: Having tuned the models in the base layer, we have optimal single models that we can stack in our meta model (figure below) to train on the basis of the predictions of the four models. We can use the outputs from the base layer to train the meta layer. The outputs can be either the predicted labels or the predicted probabilities for different classes. In the former case we get **NX4** matrix while in the latter, an **NX10** matrix of features. Also, we can combine the original features along with the predicted labels to train the meta model. This will also increase the computational cost and we should use it only if it enhances the performance.



Testing the three methods, we find that implementing stacking using predicted labels gives better results than using predicted

probabilities. Therefore, our final stacked meta model is based on predicted labels. Also, adding the original features/transformed features in the meta model does not contribute much with respect to accuracy. Thus, we have a total of 4 features in our meta model, which also reduces the computational cost in this layer. Moreover, testing different methods to be used in the meta model, we observed that Random Forest performs marginally better than SVM. Thus, our final meta-model is Random Forest Classifier.

COMPUTATIONAL COMPLEXITY

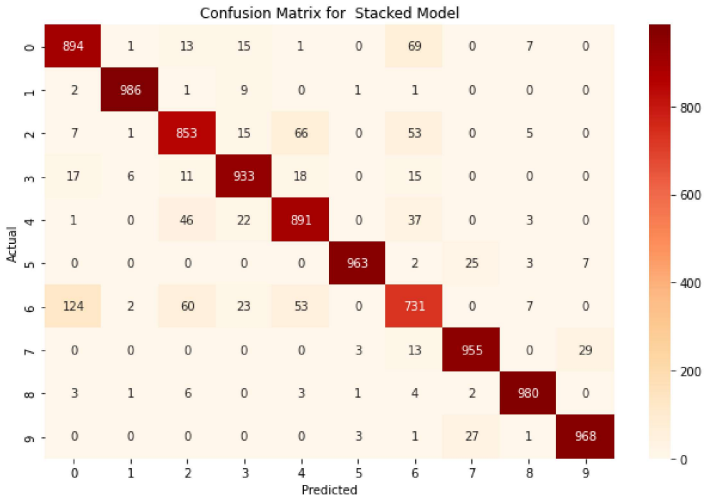
For the the base layer we have 4 models :

Model	Testing
K-Nearest Neighbors	O(n)
Random Forest	O(np)
LightGBM	O(np)
SVM	O(np)

MODEL FITTING RESULTS

	A	B	C	D	E	F	G	H	I	J
1	**Layer**	**Model**	**Training Accuracy**	**Testing Accuracy**	**Macro-Averaged Precision Score**	**Micro-Averaged Precision Score**	**Macro-Averaged Recall Score**	**Micro-Averaged Recall Score**	**Macro-Averaged F1 Score**	**Micro-Averaged F1 Score**
2	Base Layer	KNN	0.8987	0.8671	0.8675	0.8671	0.8671	0.8671	0.8655	0.8671
3		Random Forest	0.9662	0.8586	0.8573	0.8586	0.8586	0.8586	0.8564	0.8586
4		LightGBM	1	0.9168	0.9163	0.9168	0.9168	0.9168	0.9163	0.9168
5		SVM	0.911	0.8921	0.8915	0.8921	0.8921	0.8921	0.8915	0.8921
6	Meta Layer	Random Forest	1	0.9154	0.915	0.9154	0.9154	0.9154	0.915	0.9154

Confusion Matrix



	A	B
1	**Classes**	**Stackd Model**
2	0 (TShirt/Top)	0.9384
3	1 (Trouser)	0.9923
4	2 (Pullover)	0.9188
5	3 (Dress)	0.9619
6	4 (Coat)	0.9376
7	5 (Sandal)	0.981
8	6 (Shirt)	0.8546
9	7 (Sneaker)	0.9745
10	8 (Bag)	0.988
11	9 (Ankle Boot)	0.982

AUC For Stacked Model

Observations on the Stacked Model

- The Stacked model outperforms the individual models in the base layer in terms of all performance metrics. It has the highest accuracy **91.54%** and a significantly higher precision and recall metrics as compared to the base layer models
- The problematic class of shirt has a AUC score of 0.8546 which is so far the highest we have seen across all the models considered for this dataset. This shows that the stacked model is even able to correctly classify classes which are difficult to classify by other models.
- Essentially, the better performance of the stacked model can be attributed to the feature engineering performed in the inner layer in terms of transforming our training and test dataset into a set of predicted labels. Hence, the stacked model leverages the predictive power of individual models considered and “enhances” the predictions using the meta layer.

References

[1] Greeshma, K. V., and K. Sreekumar. "Fashion-MNIST classification based on HOG feature descriptor using SVM." *International Journal of Innovative Technology and Exploring Engineering* 8.5 (2019): 960-962.

[2] Solving fashion mnist using ensemble learning https://thiagolcmelo.github.io/2020-08-19-solving-fashion-mnist-using-ensemble-learning/