Q1- Classification: Feature Extraction + Classical Methods

Objective :

In this part, we have experimented with the Fashion-MNIST dataset using various Machine Learning algorithms/models. The objective is to identify (predict) different fashion products from the given images using various best possible **Machine Learning Models** and compare their results (performance measures/scores) to arrive at the best ML model. We have also performed **Feature extraction** and **Dimensionality reduction** in these experiments.

Acknowledgments:

We have referred various Kaggle notebooks to understand how to approach a machine learning problem and to understand how other people have approached this dataset in the past. These are the few resources we used to refer,

- https://www.kaggle.com/fuzzywizard/fashion-mnist-cnn-keras-accuracy-93
- https://github.com/zalandoresearch/fashion-mnist
- https://www.analyticsvidhya.com/blog/2020/01/build-your-first-machine-learning-pipeline-using-scikit-learn/
- https://www.kaggle.com/zalando-research/fashionmnist

Dataset

Fashion MNIST Training dataset consists of 60,000 images and each image has 784 features (i.e. 28×28 pixels). Each pixel is a value from 0 to 255, describing the pixel intensity. 0 for white and 255 for black.

1.1: Explanation of Design and Implementation Choices of your Model

From our previous assignment learnings, we formulated an approach to experiment on a machine learning problem as below:

Step 1: Exploratory data analysis:

- Summary statistics (seeing central tendency - Mean, STD, range of values in each feature)
- Clean the data if necessary (Not required here as there are no missing values or improper data elements)

The 'target' dataset has 5 class labels, the classes are distributed as,

Class 4 - **12067** , Class 2 - **11994**, Class 1 - **11986**, Class 3 - **11978**, Class 0 - **11975**

What kind of problem are we solving?

We are here to solve the '***Supervised Multi-class Classification***' problem. As we have to classify between 5 labels of clothing and the labels are explicitly given to us, it is, therefore 'supervised'.

- Normalization of the data (Feature Scaling)

This data has grayscale images of 28*28 pixels so the pixel can take any value between 0 to 255 and hence we try to normalize or rescale the data to unit dimension so that the value lies between 0-1 which allows the gradient descent to converge faster thereby reducing the training time.

Output of this step:

- **Dataset being preprocessed and normalized.**

Step 2: Baseline Experiment (Training with a sample of the data)

From this experiment, we can find the best hyperparameters of each model that contributes to the best accuracies. We took only 30000 data points(sample) to decrease the training time.

The chosen algorithms:

- KNN
- SVM
- Random forest classifier

- Gradient Boosting classifier
- XGBoost classifiers

How do the above algorithms work?

KNN: It is a lazy algorithm. When we get training examples, they are not processed to learn a model. Instead, the training examples are just stored and at the time of classification, the algorithm uses the stored instances from the memory to classify the new test example. It doesn't come with an apriori model. Instead, it uses the instance space of the training points and checks the closest possible instances of the point(Neighbors) to classify the new test point.

SVM: These are supervised machine learning algorithms that are used for both Classification and Regression Problems. This works by computing a hyper-plane in the N-dimensional space, where N is the total number of features. The hyperplane is chosen such that the margin is maximized between the support vectors.

Random Forests: It is a combination of several decision trees, which are prediction models using a binary set of rules to calculate a target variable. A random forest arrives at its target value based on all its decision trees. It is assumed to be more accurate than a single decision tree.

Gradient Boosting: The idea of Gradient boosting is that a weak learner can be modified to become better. This is done in a stage-wise fashion by optimizing the arbitrarily differentiable loss function.

XGBoost: It is an efficient open-source implementation of the Gradient boosted tree algorithm. It minimizes a regularized objective function that combines a convex loss function based on the predicted and target outputs and a penalty term. The training is done iteratively by adding new trees which predict the errors and residues of prior trees and are then combined with prior trees.

Why did we choose these algorithms?

Two important parameters to consider while choosing our algorithm are Variance and Bias. These are 2 main factors that decide the performance of our algorithm. There is one factor called irreducible error which is due to the framing of the problem and the mapping of the input variables to the output variables. This cannot be reduced, which leaves us with the other two factors: Bias and Variance.

Bias: These can be considered assumptions made by our model to learn the target function easily. If we have a high bias, our algorithm is learned faster, but the prediction performance takes a hit. Low Bias suggests lesser assumptions, while a higher bias suggests more assumptions

are made to learn our algorithm easily. Some low bias algorithms include KNN, SVM, Decision Trees, Random Forests, etc., while high bias algorithms include Linear Regression, LDA, etc

Variance: It can be thought of as a measure of the change in the prediction of our model if we trained it using a different data set. If we have a high variance, then the algorithm is affected by the specifics of the data set. If we have low Variance, then small changes to the estimate function, with changes in data. If we have a high variance, then there will be large changes to the estimate function, with changes in data. Some low variance algorithms include Linear Regression, LDA while the high variance algorithms include KNN, SVM, Decision Trees, Random Forests, etc.

We have to make sure our algorithm has a low variance and low bias. But it can be seen from the examples that Linear machine learning algorithms have high bias but a low variance, while nonlinear algorithms have a high variance but a low bias. We can observe that Increasing the variance decreases the bias while increasing the bias decreases the variance. So there is a trade-off between variance and bias and the aim of our algorithm should be such that there can be a balance achieved between bias and the variance by tuning the hyperparameters. Examples include,

- In KNN which is a high variance, low bias algorithm, we can achieve a balance by changing the value of k, the number of neighbors. Increasing the value of k attributes to the prediction performance and increases the bias.
- In SVM which is again a high variance, low bias algorithm, we can achieve a balance by changing the value of C. Increasing the value of C, the parameter which influences the number of violations of the margins on the training data increases the bias.
- Similarly in Random Forests, we can achieve this balance by tuning hyperparameters like max_features and min_samples_leaf.

The point to be noted here is that we can tune the parameters accordingly to our need to achieve a balance in the variance and bias. This is one of the main mathematical reasons for choosing each of the above algorithms.

To account for any non-linearity in our data, we need to do manual transformations in algorithms like Gradient boosting, XGBoosting, KNN, and Random Forests, but in SVM this is taken care of by the kernel functions.

Output of this step:

- **We train 30000 samples of the data using KNN, SVM, Gradient Boosting, Random Forest and XGBoost with various parameters specific to the model.**
- **We find the parameters that give the best accuracy for each model in specific.**

Step 3: Cross-validation - Accuracy, precision, and recall of each model

This part of the approach helps us to validate our choice of the best model. With the help of evaluation metrics like,

- Mean accuracy
- Mean Precision
- Mean Recall
- Mean f1_score
- Confusion matrix

From step 2, we know the best parameters that give best accuracy for each of the models. Now, we use those parameters and do cross validation with the whole **dataset** unlike we used only

Output of this step:

- **From Cross validation of each model, we plot the barplot of time taken by each model.**
- **We plot the barplot of *accuracy, precision, recall and f1-score* of each model.**
- **Using the barplot, we found *SVM ( )* to perform better compared to other models. And, also we considered *Random forest ( ) and XGBoost ( )* as it is almost near to *SVM.***
- **We then take the same parameters for both *Random forest and XGBoost* use cross validation to see which tops the performance. Eventually, we got *XGBoost* to win the duel.**
- **END - We chose SVM and XGBoost to proceed to Fine tuning.**

Step 4: Fine-tuning the best models chosen (SVM and XGBoost)

This part of the approach includes Feature Extraction and Feature reduction methods.

Feature Extraction: *Pearson Correlation*

Dimension Reduction: *PCA, LDA, t-SNE, Isomap, LLE, Kernel PCA*

We visualize the embeddings of each of the reduction models. Based on the relative separability of all classes, we choose one of the methods.

Output of this step:

- **We found PCA to be best among other reduction methods, as it showed better separability between classes using components scatter plot.**
- **Also, most of the nonlinear methods like, Kernel Pca, Isomap crashes due to high RAM usage.**
- **PCA components were found using screeplot. We found 30 components that retain most of the information.**
- **Before even performing PCA, we used Pearson correlation to eliminate highly correlated features with threshold greater than 0.95. That eliminated 9 features out of 784 features.**
- **Used Grid Search to find the best parameter for XGBoost.**
- **SVM using PCA+Pearson Correlation and XGBoost using PCA+Pearson Correlation was done with Cross validation.**

Step 5: Evaluate the fine-tuned models with the test dataset

- **Cross validation is done for SVM and SGBoost for their best parameters.**
- **Finally, we found for a specific parameter of each model, both give almost the same validation accuracy.**
- **But, kaggle test accuracy we got SVM as higher than XGBoost.**
- **Refer to Result and analysis for the values.**

1.2 Implementation of your Design Choices

SVM using Feature selection and Reduction

```
#Train and validation data splitting
X_train,X_val,y_train,y_val=trainValSplit(X_train_pca)

C_val= [10,13,15,17,20,22,25,30]
col=['C_value','accuracy','percision','recall','f1_score']
val = np.zeros((len(C_val),5))
evaluation=pd.DataFrame(val, columns=col)
for i,c in enumerate(C_val):
    svm = SVC(C=c,kernel='rbf',random_state=42)
    svm_scores = cross_val_score(svm,X_train_pca, y, cv=10, scoring="accuracy")
    svm_accuracy = svm_scores.mean()
    y_pred = cross_val_predict(svm, X_train_pca, y, cv=10)
    svm_precision = precision_score(y, y_pred, average='weighted')
    svm_recall = recall_score(y, y_pred, average='weighted')
    svm_f1_score = f1_score(y, y_pred, average='weighted')
    print("C Value:{}, Accuracy:{}".format(c,svm_accuracy))
    evaluation.at[i,'C_value'] = c
    evaluation.at[i,'accuracy'] = svm_accuracy
    evaluation.at[i,'percision'] = svm_precision
    evaluation.at[i,'recall'] = svm_recall
    evaluation.at[i,'f1_score'] = svm_f1_score
```

2.5) XGBoost classifier

parameters:

- n_estimators =[250,300,350,400,450,500,600,700,800,900,1000]
- Depth = 10
- random state = 42

Normalization: None (Scaling is not necessary for random forests,Reason in report, refer:)

```
#Train and validation data splitting
X_train,X_val,y_train,y_val=trainValSplit(X)

tree_list= [250,300,350,400,450,500,600,700,800,900,1000]
depth=[10]
col=['n_estimators','Depth','accuracy','percision','recall','f1_score']
val = np.zeros((11,6))
evaluation=pd.DataFrame(val, columns=col)
count=0
for t in tree_list:
  for n in depth:
    xgb_clf = XGBClassifier(n_estimators=t, max_depth=n, random_state=42)
    xgb_clf.fit(X_train, y_train)
    print(xgb_clf)
    y_pred = xgb_clf.predict(X_val)
    accscore = accuracy_score(y_val, y_pred)
    precision= precision_score(y_val,y_pred,average='weighted')
    recall = recall_score(y_val, y_pred, average='weighted')
    f1_scr = f1_score(y_val, y_pred, average='weighted')
    evaluation.at[count,'n_estimators'] = t
    evaluation.at[count,'Depth']=n
    evaluation.at[count,'accuracy'] = accscore
    evaluation.at[count,'percision'] = precision
    evaluation.at[count,'recall'] = recall
    evaluation.at[count,'f1_score'] = f1_scr
```

Using Grid Search to find the best parameter for XGBoost

```
# GridSearchCV

from sklearn.model_selection import GridSearchCV

param_grid = [
    # try (1x3)=3 combinations of hyperparameters
    {'n_estimators': [400], 'max_depth': [50,60,70]},

]

xgb_clf_grid_search = XGBClassifier(random_state=42)
# train across 3 folds, that's a total of 3x3=9 rounds of training
grid_search = GridSearchCV(xgb_clf_grid_search, param_grid, cv=3,
                           scoring='neg_mean_squared_error')
grid_search.fit(X_train_pca, y)
```

3.1) KNN

Best parameters:

Weight : Manhattan distance
K Value : 3
Accuracy of 30000 : 82.58
samples

```
knn = KNeighborsClassifier(n_neighbors=3, weights='distance',p=2)
knn_scores = cross_val_score(knn,X_scaled, y, cv=3, scoring="accuracy")
knn_accuracy = knn_scores.mean()
```

time: 1h 8min 39s

```
y_pred = cross_val_predict(knn, X_scaled, y, cv=3)
y_actu = pd.Series(y, name='Actual')
y_pred = pd.Series(y_pred, name='Predicted')
df_confusion = pd.crosstab(y_actu, y_pred)
knn_precision = precision_score(y, y_pred, average='weighted')
knn_recall = recall_score(y, y_pred, average='weighted')
knn_f1_score = f1_score(y, y_pred, average='weighted')

print(df_confusion)
display_scores(knn_scores)
print("KNN CV Accuracy: ", knn_accuracy)
print("KNN CV Precision: ", knn_precision)
print("KNN CV Recall: ", knn_recall)
print("KNN CV F1 Score: ", knn_f1_score)
```

2.3)Random forest classifier

parameters:

- n_estimators = [50,100,200,250,300,350,400,450,500,550,600]
- Depth = [20,30,40,50,None]
- random state = 42

Normalization: None (Scaling is not necessary for random forests,Reason in report, refer:)

```
#Train and validation data splitting
X_train,X_val,y_train,y_val=trainValSplit(X)

tree_list= [50,100,200,250,300,350,400,450,500,550,600]
depth=[20,30,40,50,None]
col=['n_estimators','Depth','accuracy','percision','recall','f1_score']
val = np.zeros((10,6))
evaluation=pd.DataFrame(val, columns=col)
count=0
for t in tree_list:
  for n in depth:
    rnd_clf = RandomForestClassifier(n_estimators=t, max_depth=n, random_state=42)
    rnd_clf.fit(X_train, y_train)
    y_pred = rnd_clf.predict(X_val)
    accscore = accuracy_score(y_val, y_pred)
    precision= precision_score(y_val,y_pred,average='weighted')
    recall = recall_score(y_val, y_pred, average='weighted')
    f1_scr = f1_score(y_val, y_pred, average='weighted')
    print("n_estimators:{}, Depth:{}, Accuracy:{}".format(t,n,accscore))
    evaluation.at[count,'n_estimators'] = t
    evaluation.at[count,'Depth']=n
    evaluation.at[count,'accuracy'] = accscore
    evaluation.at[count,'percision'] = precision
    evaluation.at[count,'recall'] = recall
    evaluation.at[count,'f1_score'] = f1_scr
    count+=1
```

```
pca_std = PCA(n_components=30).fit(X_train_scaled)
X_train_pca= pca_std.transform(X_train_scaled)
X_test_pca = pca_std.transform(X_test_scaled)
```

```
time: 3.45 s
```

```
ex_variance=np.var(X_train_pca,axis=0)
ex_variance_ratio = ex_variance/np.sum(ex_variance)
print (ex_variance_ratio)
print(sum(ex_variance_ratio))
```

```
[0.35414892 0.21564557 0.07346638 0.06043703 0.04686828 0.04235068
 0.02865591 0.02316527 0.01638992 0.01592145 0.01206165 0.01115966
 0.00933643 0.0080316  0.00742094 0.00715734 0.00675958 0.00643031
 0.0056193  0.00554971 0.0052796  0.00494827 0.00468615 0.00453947
 0.0043911  0.00426979 0.00406915 0.00387541 0.00376289 0.0036023 ]
1.000000066589564
time: 12 ms
```

```
pal={0:"#9b59b6",1:"#3498db",2:"#e74c3c",3:"#2ecc71",4:"#34495e"}
fig=plt.figure(figsize=(25,35))
for i in range(len(X_train_pca[:29])):
    x=X_train_pca[:,i]
    y1=X_train_pca[:,(i+1)]
    plt.title("PC"+str(i)+" Vs "+"PC"+str(i+1))
    plt.xlabel("PC"+str(i))
    plt.ylabel("PC"+str(i+1))
    plt.subplot(6,5,i+1)
    sns.scatterplot(x,y1,hue=y,palette=pal)
plt.show()
```

1.3 Kaggle Competition Score

The Highest score in Kaggle for question 1 - ***90% Test Accuracy***. And, the corresponding model and parameters used are,

- Classification ML model : *SVM (C=20, Kernel='rbf')*
- Feature Extraction method : *Pearson Correlation (Threshold = 0.95)*
- Feature Reduction method : *PCA (n_components=30)*

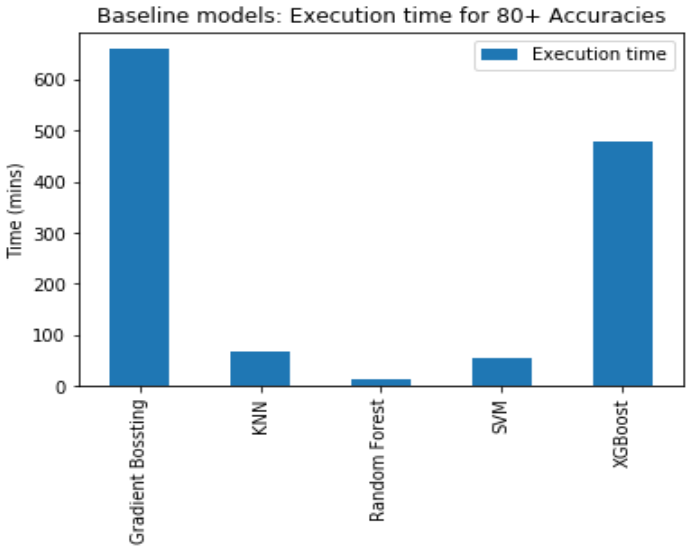
Though it is a good score for a classification model, some of our peers were able to achieve more than 90% accuracy using classical machine learning methods. The following could be the reason for their accuracies,

1. We can use XGBoost with proper parameter selection (In fact high estimator and depth) to train the dataset. Though it could give us better accuracy, the training time is very huge. The interpretability also can be complicated.
2. Any other proper feature extraction method could have been used to remove redundant features, which can result in better accuracies. As, the redundant features are mere noise, removing those will benefit the training.

1.4 Results Analysis

1) Runtime performance of training and testing :

Baseline models:



- **Gradient boosting** algorithm took ***more time*** than any other algorithm.
- **Random forest** was ***faster*** than any other algorithm.

2) Comparison of the different algorithms and parameters:

Baseline models: (without any feature extraction or reduction)

Algorithm	Parameter	Time (mins)	Accuracy	Precision	Recall	f1_score
KNN	K = 3, Weight= manhattan	68	0.8285	0.8276	0.8285	0.8277
SVM	**C = 15** **Kernel - 'rbf'**	**55**	**0.8774**	**0.8771**	**0.8774**	**0.8772**
Random forest	n_est = 250 Depth = 20	12	0.8436	0.8442	0.8436	0.8439
Gradient Boosting	n_est = 700	684	0.8297	0.8271	0.8297	0.8272
XGBoosting	n_est = 900 Depth =10	524	0.8497	0.8471	0.8471	0.8472

We found **SVM (0.8774)** to perform better compared to other models. And, also we considered **Random forest (0.8436)** and **XGBoost (0.8497)** as it is almost near to SVM.

Comparison of Random forest and XGBoost using same parameters (n_est = 20 , depth = 10)

Algorithm	Parameter	Time (mins)	Accuracy	Precision	Recall	f1_score
Random forest	n_est = 20 Depth = 10	0.12	0.7948	0.7965	0.7948	0.7954
XGBoosting	**n_est = 20** **Depth =10**	**13.24**	**0.8129**	**0.8139**	**0.8129**	**0.8133**

Therefore, we take SVM and XGBoost for further fine tuning.

Fine tuned models

Algorithm	Parameter	Time (mins)	Validation accuracy	Test Accuracy	Precision	Recall	f1_score
SVM	**C = 30** **Kernel - 'rbf'**	**16.9**	**0.8945**	**0.89980**	**0.8884**	**0.8887**	**0.8885**
XGBoosting	n_est = 400 Depth =70	28.3	0.8921	0.886	0.8892	0.8915	0.8912

3) Explanation of your model (algo, design choice, no of parameters)

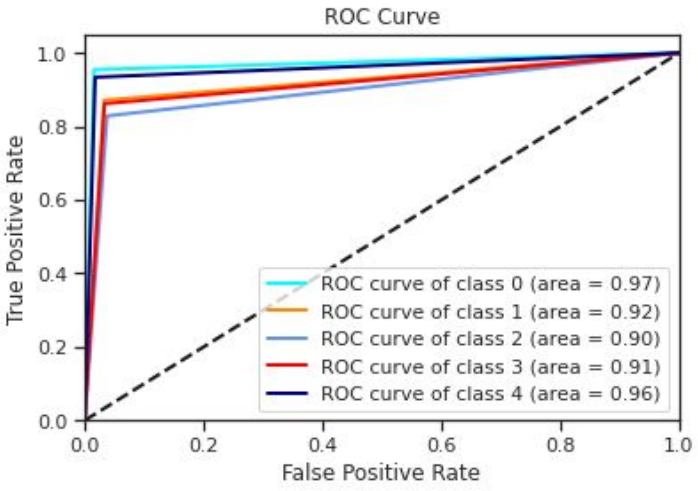
After Fine-tuning the model that gives the best test accuracy is SVM. We used cross validation to find the best parameters that perform well.

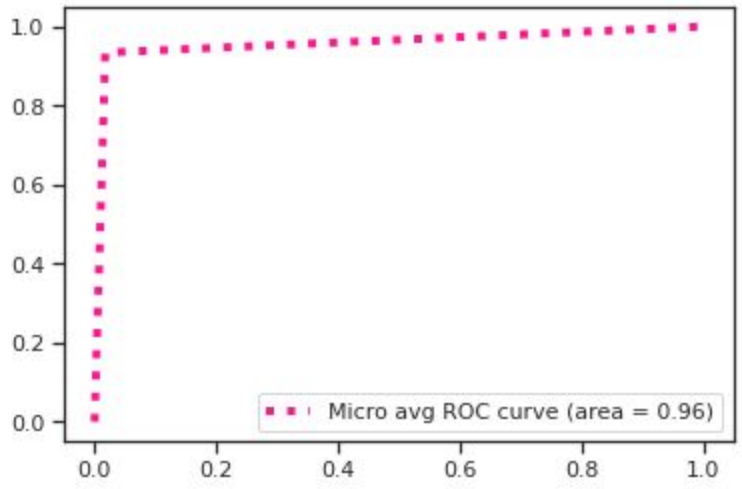
- C value - 30
- Kernel - 'rbf'
- PCA reduction - 30 components
- Pearson correlation - threshold = 0.95

These parameters resulted in best performance for us.

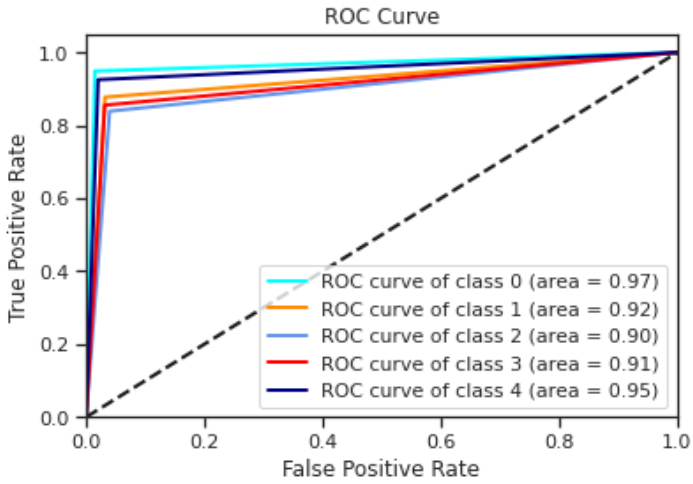
4) Use a ROC curve used for some method in your initial or results analysis such as exploring the impact on the accuracy of some parameter

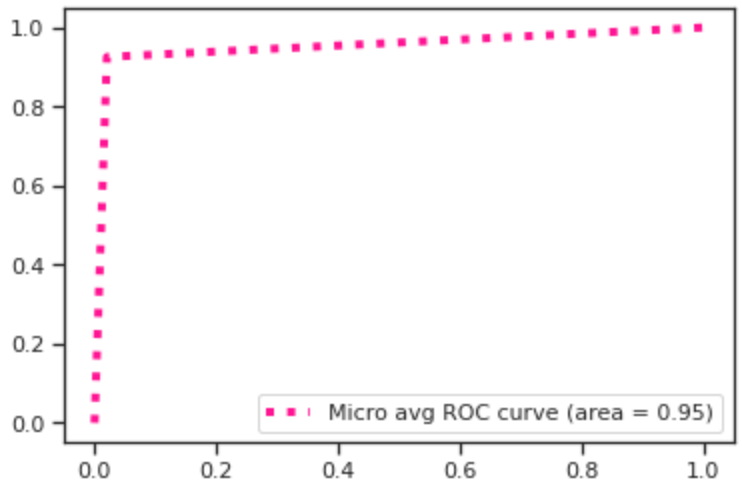
SVM :




1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Micro avg ROC curve (area = 0.96)

XGBoost:


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.97)
ROC curve of class 1 (area = 0.92)
ROC curve of class 2 (area = 0.90)
ROC curve of class 3 (area = 0.91)
ROC curve of class 4 (area = 0.95)



- The Class 0,1,2,3 all have the same area for both SVM and XGBoost.
- The Class 4 have more area for SVM than XGBoost
- The Micro average ROC area for SVM is 0.96 unlike XGBoost it is 0.95.

5) Evaluate your code with other metrics on the training data (by using some of it as test data) and argue for the benefit of your approach.

Algorithm	Parameter	Time (mins)	Accuracy	Precision	Recall	f1_score
KNN	K = 3, Weight= manhattan	68	0.8285	0.8276	0.8285	0.8277
SVM	**C = 15** **Kernel - 'rbf'**	**55**	**0.8774**	**0.8771**	**0.8774**	**0.8772**
Random forest	n_est = 250 Depth = 20	12	0.8436	0.8442	0.8436	0.8439
Gradient Boosting	n_est = 700	684	0.8297	0.8271	0.8297	0.8272

XGBoosting	n_est = 900 Depth =10	524	0.8497	0.8471	0.8471	0.8472

We have used confusion matrix, precision, recall and f1-score as evaluation metrics apart from accuracy.

Confusion Matrix: It can be thought of like a summary of the prediction results of a classification problem. The amount of correct and incorrect predictions is listed with the count values and they are broken down according to the class.

	Positive	Negative
Positive	True Positive(TP)	False Positive(FP)
Negative	False Negative(FN)	True Negative(TN)

Accuracy: From the above confusion matrix, accuracy is defined as the ratio of the number of true positives and true negatives to that of the sum of all true positives and negatives and False positives and negatives.

$$\text{Accuracy} = (TN+TP)/(TN+TP+FN+FP)$$

Precision: Precision can be thought of as, the number of true positive outcomes from all the outcomes which are predicted as positive. It is defined as the ratio of the no.of outcomes which are True Positive to that of the sum of the True Positive and False Positive.

$$\text{Precision} = (TP)/(TP+FP)$$

Recall: It is defined as the ratio of the number of correctly classified positive examples to that of the total number of Positive examples.

$$\text{Recall} = (TP)/(TP+FN)$$

F1_score or F-Measure: It conveys the balance between precision and recall. It makes use of the Harmonic mean instead of the arithmetic mean.

$$\text{F-Measure} = 2 * (\text{Precision}*\text{Recall})/(\text{Precision}+\text{Recall})$$