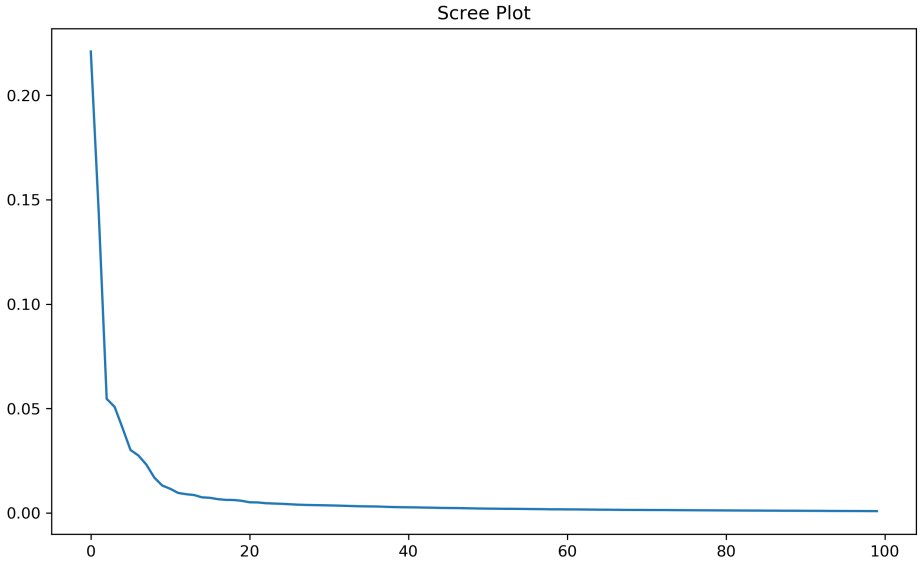
The source code is at the end of this document

1 Classification: Feature Extraction + Classical Methods

1.1 Explanation of Design and Implementation Choices of your Model

The models I used on Kaggle are XGBoost, SVM, Random Forest and Hist Gradient Boosting. Below is the procedure to show why I chose them.

Dataset was given with labels, so naturally supervised learning methods were considered. There are 784 dimensions in the dataset, therefore, PCA should be applied on it to extract the main features and to decrease the dimension. First I used standard scalar to centralize data and then drew the scree plot with top 100 principle components.



More than 99 percent of variance can be explained by the top 20 principle components. I took n_components = 30 for a better result.

The following classification methods with default parameters were applied on the PCA transformed data. (Except for MultinomialNB since Naive Bayes based on applying Bayes' theorem with strong independence assumptions between the features, and preprocessing will seriously impact the result.) Using 20 percent data as test set, their corresponding accuracy are shown below.

Classifier	**Accuracy**
MultiNomialNB	47.18%
Random Forest	86.95%
KNN	85.86%
SVM	86.13%
Logistic Regression	67.78%
LDA	63.68%
AdaBoost	61.03%
XGBoost	87.14%
Histogram-based Gradient Boosting	87.38%

I did not try Gradient Boosting because both XGBoost and Hist Gradient Boosting are the "upgraded" version of it. E.g. In the Hist Gradient Boosting's User Guide

> These histogram-based estimators can be **orders of magnitude faster** than GradientBoostingClassifier and GradientBoostingRegressor when the number of samples is larger than tens of thousands of samples.

Random Forest, KNN, SVM, XGBoost and Hist Gradient Boosting were chosen for further parameter tuning. KNN's accuracy was always 1-2 percent lower than the others so I eliminated it as well. The working mechanism for the rest four classifiers are as follows:

Random Forest: uses a large number of uncorrelated individual decision trees, and each tree will give a class prediction. The class with the most votes will be the model's final prediction.

SVM: applies a kernel function on the data and finds hyperplanes to separate two classes, and picks the one that maximizes the margin. For multiclass problems like this one, the problem will be treated as multiple binary classifications(One-vs-All or One-vs-One).

Extreme Gradient Boosting: just like gradient boosting, xgboost builds multiple trees, and each tree is based on minimizing the loss of the previous tree. And the final prediction is based on the sum of the learning rate times the prediction of each tree. However, xgboost uses a different loss function and introduces other parameters to make the modeling more robust and faster.

Histogram-based Gradient Boosting: a new experimental implementation of gradient boosting trees in sklearn 0.21, no blog or paper currently explains its detailed implementation. According to documentation, it is inspired by LightGBM.

I chose the above algorithms because their performance is relatively good. And except for SVM, the other 3 are ensemble methods, which provide an extra degree of freedom in the classical bias/variance tradeoff and they are also unlikely to overfit. In addition, they are easy to run on multicores to save running time.

Apart from PCA alone as the data preprocessing method, I also tried histogram of oriented gradients(HOG) or HOG combined with PCA as data preprocessing according to this paper. HOG counts occurrences of gradient orientation in each localized area of an image, so that classification methods such as SVM can use these info to make prediction. However, the accuracy is only about 62%.

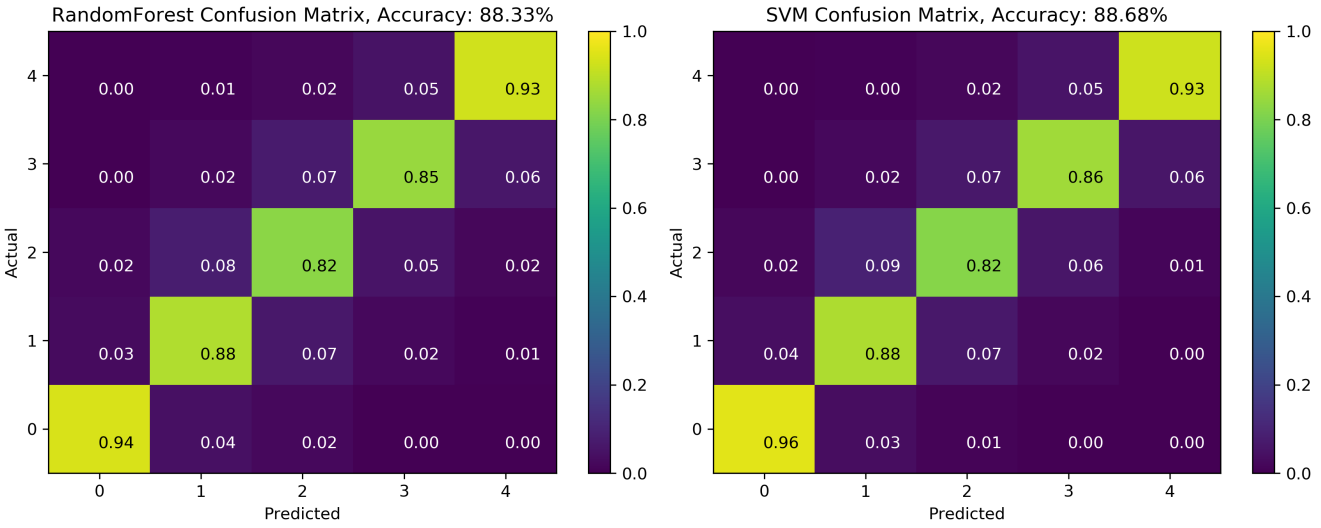
1.2 Implementation of your Design Choices

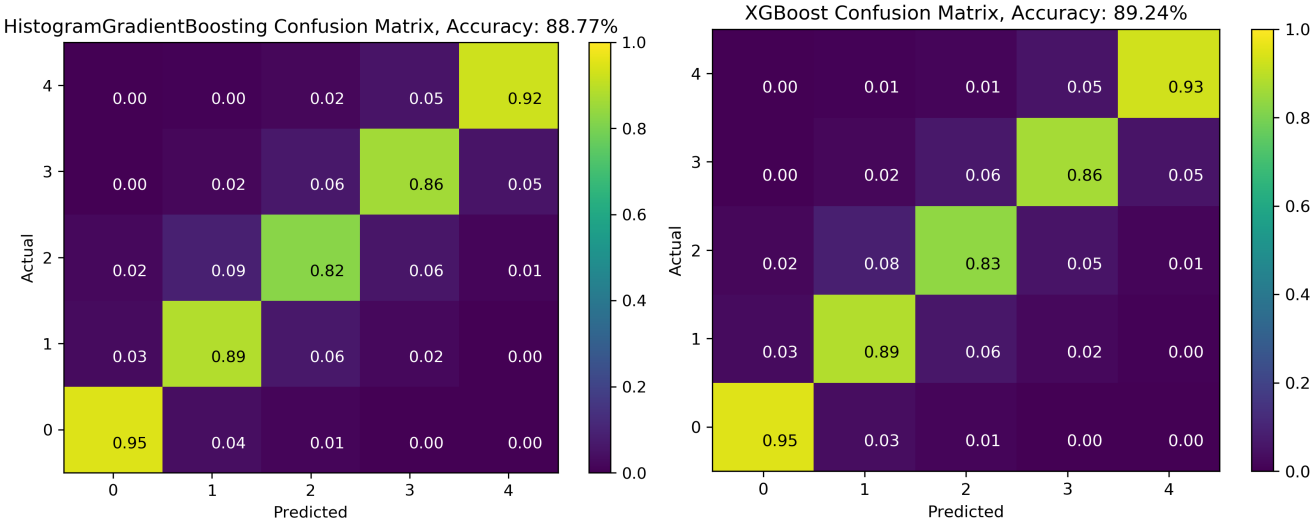
1.3 Kaggle Competition Score

1.4 Results Analysis

1.4.1 Runtime performance for training and testing

Using PCA n_components = 30, 80 percent labeled data for training and 20 percent for testing, confusion matrices are plotted as below.





The accuracy given by Kaggle is:

Classifier	**Accuracy**
Random Forest	88.14%
SVM	88.78%
XGBoost	89.38%
Histogram-based Gradient Boosting	89.20%

1.4.2 Comparison of the different algorithms and parameters

From the confusion matrices we can see that for all four algorithms, it is easier to classify class 0 and 4 than other 3 classes. The accuracy to classify 0 and 4 is around 93 percent while it is only 82 percent for class 2. We can also see that the similarity between adjacent classes is relatively greater, for most misclassification rate is contributed by the grids next to the minor diagonals on the matrices.

In terms of fitting time, to achieve the same(88%) accuracy, random forest, hist gradient boosting and xgboost use about 70 seconds and SVM uses about 80 seconds. However, the comparison

is not that meaningful because SVM is not an ensemble method, and all other three algorithms can run with multicores. The time will vary a lot if those algorithms are tested on a different computer.

For random forest, the accuracy will not increase any more when the number of estimators increased to a certain amount, no matter what the other parameters are. However, the accuracy of Hist gradient boosting and xgboost will increase gradually when the learning rate is decreased and n_estimator is increased.

The specific parameters I tried for each algorithm will be discussed in the following sub-subsection.

1.4.3 Explanation of your model

2 Classification: Convolutional Neural Networks

2.1 Design and Implementation Choices of your Model

The following CNN models are tested.

CNN1 is a basic CNN with one convolutional layer; CNN2 is similar to the network in the link given in the assignment; CNN3 is CNN2 with one more convolutional layer and max pooling layer; CNN4 is another VGG like network given in this link. In addition, all models in this blog are evaluated, but they have very similar performance with the previous ones so I will not show their structure below. Meanwhile, VGG19 model with imagenet pretrained weight and transfer learning of InceptionV3 were tested. Finally, I tried training a Resnet 50 and an Inception model from scratch, and the tutorial can be found here(Coursera CNN week 2 programming assignments) and

here respectively.