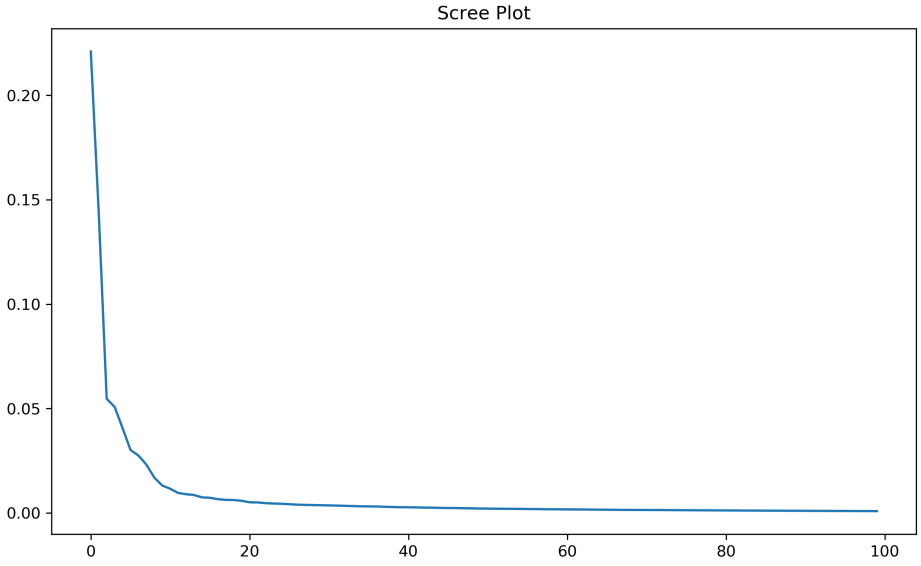
The source code is at the end of this document

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