Machine Learning Engineer Nanodegree

Capstone Project

Santander Customer Satisfaction Kaggle Competition

Scott Yao

July 31st, 2017

I. Definition

Project Overview

This project is based on the Santander Customer Satisfaction Kaggle Competition.

The competition will provide train and test dataset containing a large number of numeric variables. The feature names has been anonymized. The "TARGET" column is the variable to predict. It equals one for unsatisfied customers and 0 for satisfied customers. Therefore, this problem falls into a typical supervised learning classification category.

By finishing this project, I'll provide a fine tuned classification model to predict the customer satisfaction label with outperforming metrics based on input features.

Since this project ended a year ago, there have been multiple Kagglers posting their solutions. Some good examples here:

- Feature exploration: shows strong data analysis and visualization skills
- PCA example: a good visualization on PCA
- XGBoost with 36 features: an end-to-end solution with ~0.83 score, also used as benchmark

Problem Statement

The problem is to predict a probability of customer satisfaction classification based on anonymous numerical features. The train dataset has ~700 features and ~75k samples

which will require more data analysis and exploration work. Data preprocessing and feature selection will help increase the model performance. This is a typical supervised learning classification problem, so comparing multiple model's performance with cross validation will be useful. I've selected 14 classification models and listed them in the Algorithms and Techniques section.

After selected the best performing model by cross validation, I applied GridSearchCV to do parameter tuning trying to get a better score.

Metrics

To be consistent with the competition, the results will be evaluated on area under the ROC curve between the predicted probability and the observed target. Roc_auc_score is also a popular metrics for measuring binary classification problems.

More details on AUC from wikipedia:

- In statistics, a receiver operating characteristic curve, i.e. ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings. TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate.
- AUC formula:
  - $$A = \int_{\infty}^{-\infty} \mathrm{TPR}(T)\left(-\mathrm{FPR}'(T)\right)\,dT = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(T' > T) f_1(T') f_0(T)\,dT'\,dT = P(X_1 > X_0)$$
  - where X1 is the score for a positive instance and X0 is the score for a negative instance.

II. Analysis

Data Exploration

For this project, data is provided by Santandar as the Kaggle competition. The train dataset has 76720 rows of data with 370 features and 1 labels, while test dataset has 75818 rows.

Features have been anonymized and the label is “TARGET”. Data samples can be found in the capstone-analysis.ipynb. All features are numeric type, TARGET is 0/1. 0 means the customer is satisfied while 1 is disappointed.

I found one abnormalities for a feature named “var3”, the following is the top 10 value distribution:

Value	Count
2	74165
8	138
-999999	116
9	110
3	108
1	105
13	98
7	97
4	86
12	85

-999999 is probably a missing value has been transformed to the extreme negative, so I changed it to the most common 2 to reduce outlier impact. Following is the distribution after change:

Value	Count
2	74281
8	138
9	110
3	108
1	105
13	98
7	97

4	86
12	85
6	82

The dataset also have 34 invariance columns. I did a scan to find out the columns with 0 standard deviation:

['ind_var2_0', 'ind_var2', 'ind_var27_0', 'ind_var28_0', 'ind_var28', 'ind_var27', 'ind_var41', 'ind_var46_0', 'ind_var46', 'num_var27_0', 'num_var28_0', 'num_var28', 'num_var27', 'num_var41', 'num_var46_0', 'num_var46', 'saldo_var28', 'saldo_var27', 'saldo_var41', 'saldo_var46', 'imp_amort_var18_hace3', 'imp_amort_var34_hace3', 'imp_reemb_var13_hace3', 'imp_reemb_var33_hace3', 'imp_trasp_var17_out_hace3', 'imp_trasp_var33_out_hace3', 'num_var2_0_ult1', 'num_var2_ult1', 'num_reemb_var13_hace3', 'num_reemb_var33_hace3', 'num_trasp_var17_out_hace3', 'num_trasp_var33_out_hace3', 'saldo_var2_ult1', 'saldo_medio_var13_medio_hace3']

Since the invariance won't provide any useful information to predict target, it's better to drop them to save execution time and memory.

In addition, 29 columns have the same value with the other:

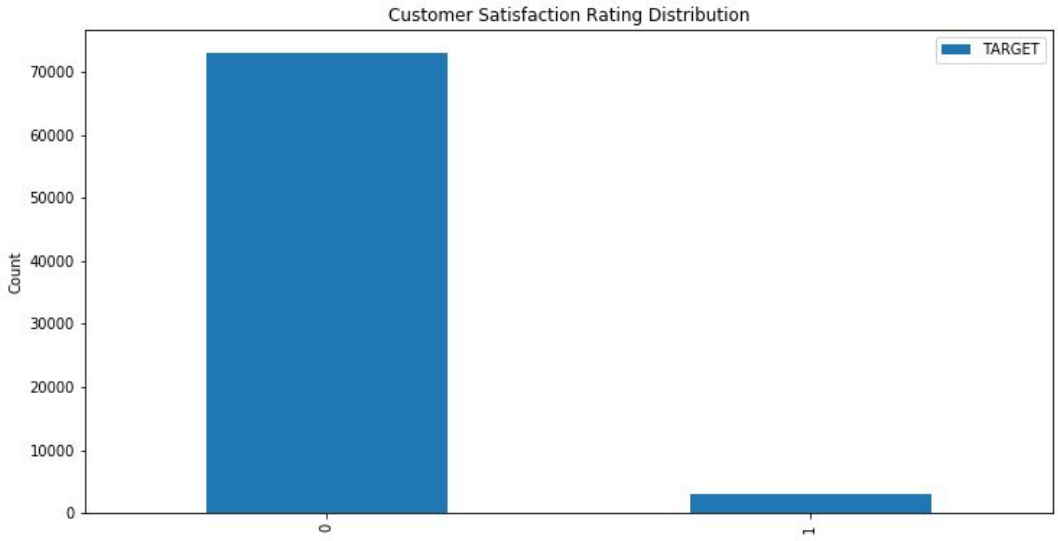
['ind_var29_0', 'ind_var29', 'ind_var13_medio', 'ind_var18', 'ind_var26', 'ind_var25', 'ind_var32', 'ind_var34', 'ind_var37', 'ind_var39', 'num_var29_0', 'num_var29', 'num_var13_medio', 'num_var18', 'num_var26', 'num_var25', 'num_var32', 'num_var34', 'num_var37', 'num_var39', 'saldo_var29', 'saldo_medio_var13_medio_ult1', 'delta_num_reemb_var13_1y3', 'delta_num_reemb_var17_1y3', 'delta_num_reemb_var33_1y3', 'delta_num_trasp_var17_in_1y3', 'delta_num_trasp_var17_out_1y3', 'delta_num_trasp_var33_in_1y3', 'delta_num_trasp_var33_out_1y3']

After dropping the invariance and duplicate, the train dataset has 306 features left. Increasing the size of feature set by ~17%.

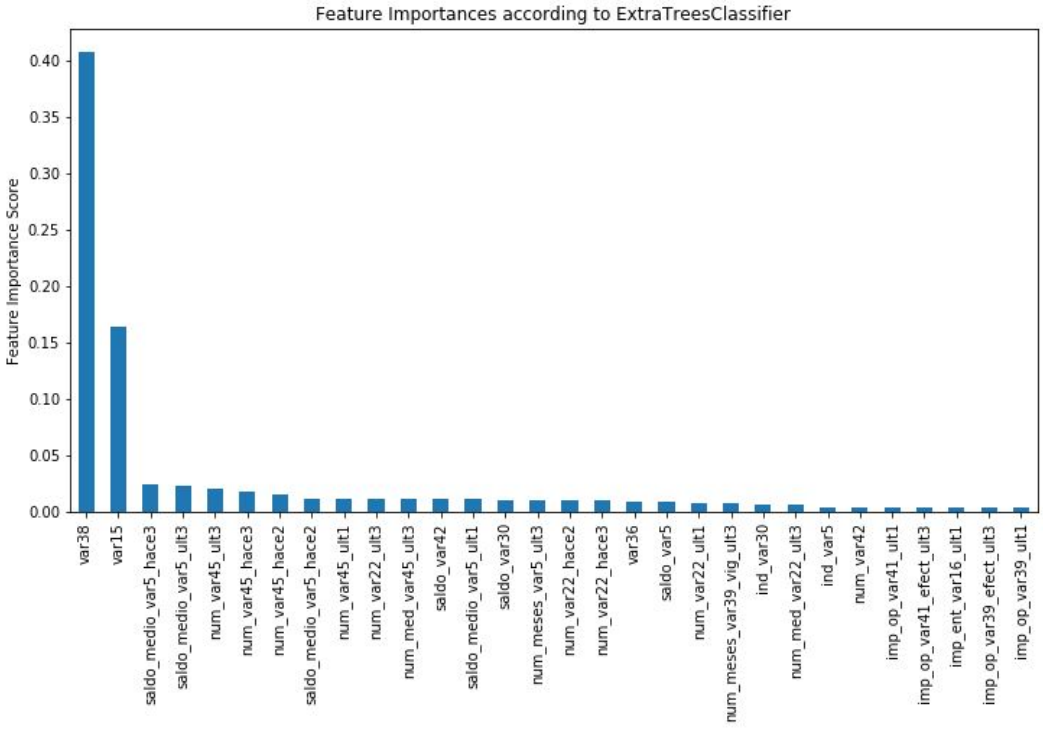
I also apply statistic analyze to the dataset by utilizing pandas dataframe describe function, due to the page size, please refer to the project notebook.

Exploratory Visualization

This section is based on train dataset only. I found out ~96% of customers are satisfied with the banking product:



After the previous section, we still have 306 features left. This is not a small amount to me, so I'm pretty interested in the relevance of each feature to the target label. Therefore, I used the ExtraTreesClassifier from sklearn to apply the feature selection method. And the result is pretty inspiring:



Var38 and var15 have the most feature importance score, while others are pretty close. The sklearn SelectFromModel selected total 37 features, that's ~89% feature set decrease compared to the original data.

Algorithms and Techniques

For this project, I'll use pandas and sklearn to perform the analyze and prediction. Algorithms and techniques listed below with related topic:

- Data cleansing: Statistical analyze to detect outliers.
- Feature selection: Tree based feature selection
- Data visualization: matplotlib
- Model selection: Cross validation with Stratified K-Folds cross-validator
- Parameter tuning: GridSearch

The following models has been chosen with default settings:

- Decision Tree
    - tree.DecisionTreeClassifier()
- Ensemble Learning

    - ensemble.ExtraTreesClassifier()
    - ensemble.RandomForestClassifier()
    - ensemble.AdaBoostClassifier()
    - ensemble.BaggingClassifier()
    - ensemble.GradientBoostingClassifier()
- Linear Model
    - linear_model.RidgeClassifier()
    - linear_model.PassiveAggressiveClassifier()
    - linear_model.SGDClassifier()
- Naive Bayes
    - naive_bayes.GaussianNB()
- Instance Based Learning
    - neighbors.KNeighborsClassifier()
- Neural Network
    - neural_network.MLPClassifier()
- Support Vector Machine
    - svm.SVC() by default using rbf kernal
    - svm.LinearSVC()

Benchmark

Since this project is a real world competition, I think it will be meaningful if I can compare with other players solution and use them as benchmark. I found one here is using XGBClassifier from xgboost, producing a ~0.83 score with 36 features.

Reasons why I chose that model as a benchmark:

- The implementation is simple and straightforward.
- It also did similar data cleansing and feature selection but got fewer result
- ~0.83 score is really good since the top player got ~0.829 in the competition
- It only chose XGBClassifier with the default parameter settings, I'd like to try multiple models and fine tuning the parameter to get a close or better result

III. Methodology

Data Preprocessing

Based on the data exploration section, I've performed data cleansing to make the modeling perform better:

- Drop "ID" column in both dataset, since "ID" normally has no relation with the target
- Replace "-999999" value in Var3 with the most common "2"
- Removed 34 columns with zero standard deviation, since invariance won't provide any useful information in prediction
- Removed 29 duplicate columns have the same values as others
- Using sklearn ExtraTreesClassifier and SelectFromModel to get total 37 features

Implementation

In order to have a good fitting model, I decide to try multiple classification models and use cross validation to select the best performing one.

```
# model candidates
models = {}

models['tree'] = tree.DecisionTreeClassifier()
models['extra_tree'] = ensemble.ExtraTreesClassifier()
models['forest'] = ensemble.RandomForestClassifier()
models['ada_boost'] = ensemble.AdaBoostClassifier()
models['bagging'] = ensemble.BaggingClassifier()
models['grad_boost'] = ensemble.GradientBoostingClassifier()
models['ridge'] = linear_model.RidgeClassifier()
models['passive'] = linear_model.PassiveAggressiveClassifier()
models['sgd'] = linear_model.SGDClassifier()
models['gaussian'] = naive_bayes.GaussianNB()
models['kneighbors'] = neighbors.KNeighborsClassifier()
models['mlp'] = neural_network.MLPClassifier()
##Too slow
#models['svc'] = svm.SVC()
models['linearSVC'] = svm.LinearSVC()
```

I tried to use more different classification model as the nanodegree course covered. For each model, I applied cross-validation with StratifiedKFold. This cross-validation object is a variation of KFold that returns stratified folds. The folds are made by preserving the percentage of samples for each class. Since the satisfaction rate is quite skewed with

~96%, it will be more accurate if each folds can preserving the sample distribution percentage.

KFold is also better than one 80-20 split since it will apply average on multiple folds tests, which avoids one single bad split's huge impact on model evaluation.

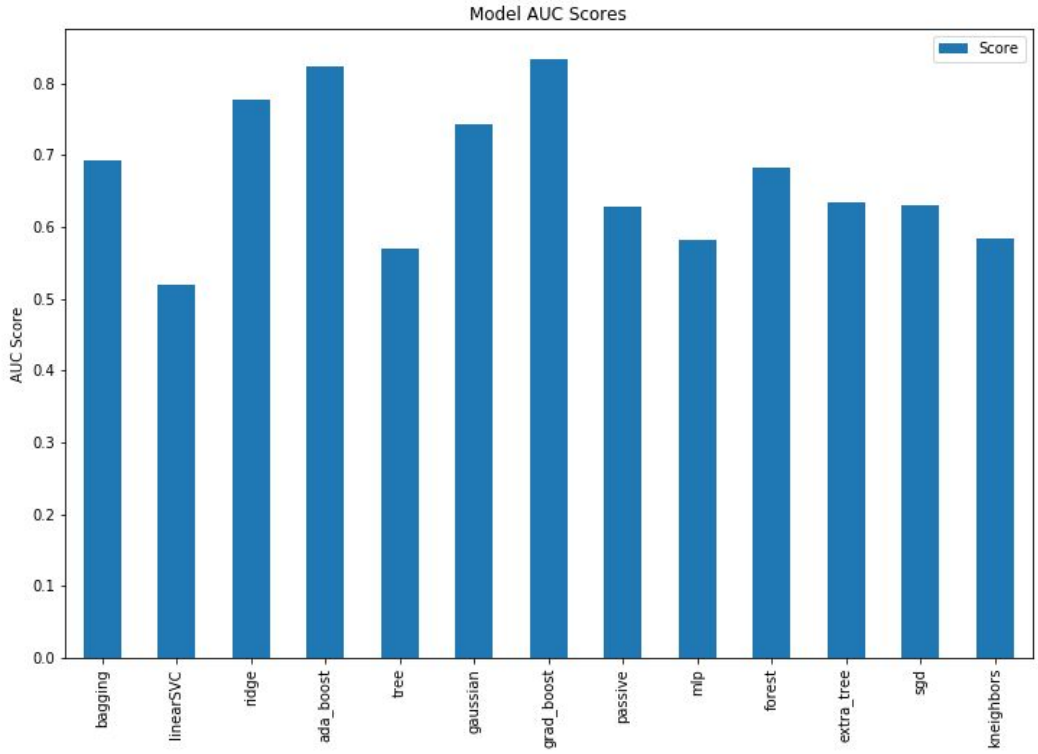
```
# KFold
skf = model_selection.StratifiedKFold(shuffle=True, random_state=123)
score_metric = 'roc_auc'
scores = {}
compute_time = {}

# Scoring function by cross_val_score
def score_model(model):
    return model_selection.cross_val_score(model, X, y, cv=skf, scoring=score_metric)
```

I also used the cross_val_score to compare each model's scoring. By averaging KFolds I can get a generalized roc_auc score:

```
# Evaluate each model on score avg and stddev and record time spent
for k, v in models.items():
    start_time = time.time()
    score = score_model(v)
    scores[k] = np.mean(score)
    scores_std[k] = np.std(score)
    compute_time[k] = time.time() - start_time
```

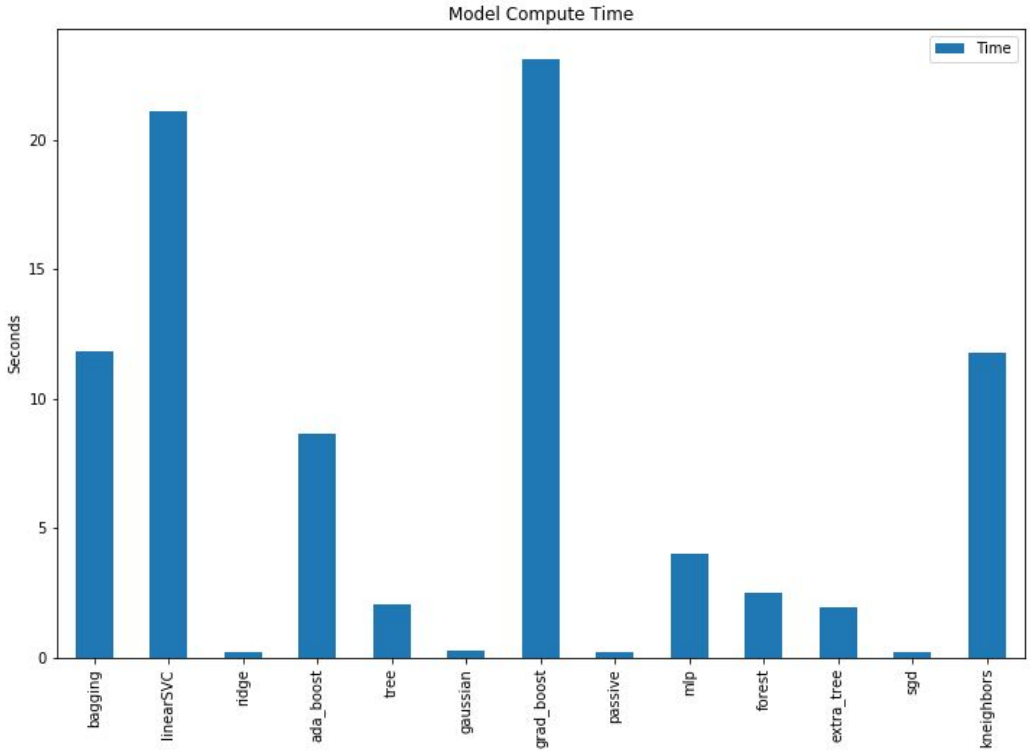
After cross validation on the model selected as candidate in the previous section, I evaluated them by the roc_auc score. The roc_auc is a popular evaluation metrics for binary classification task since it will take into consideration of both precision and recall scores. For detail please refer here.

The following graph shows the roc_auc score for each classification model with default settings on a default StrtifiedKFold cross-validation:



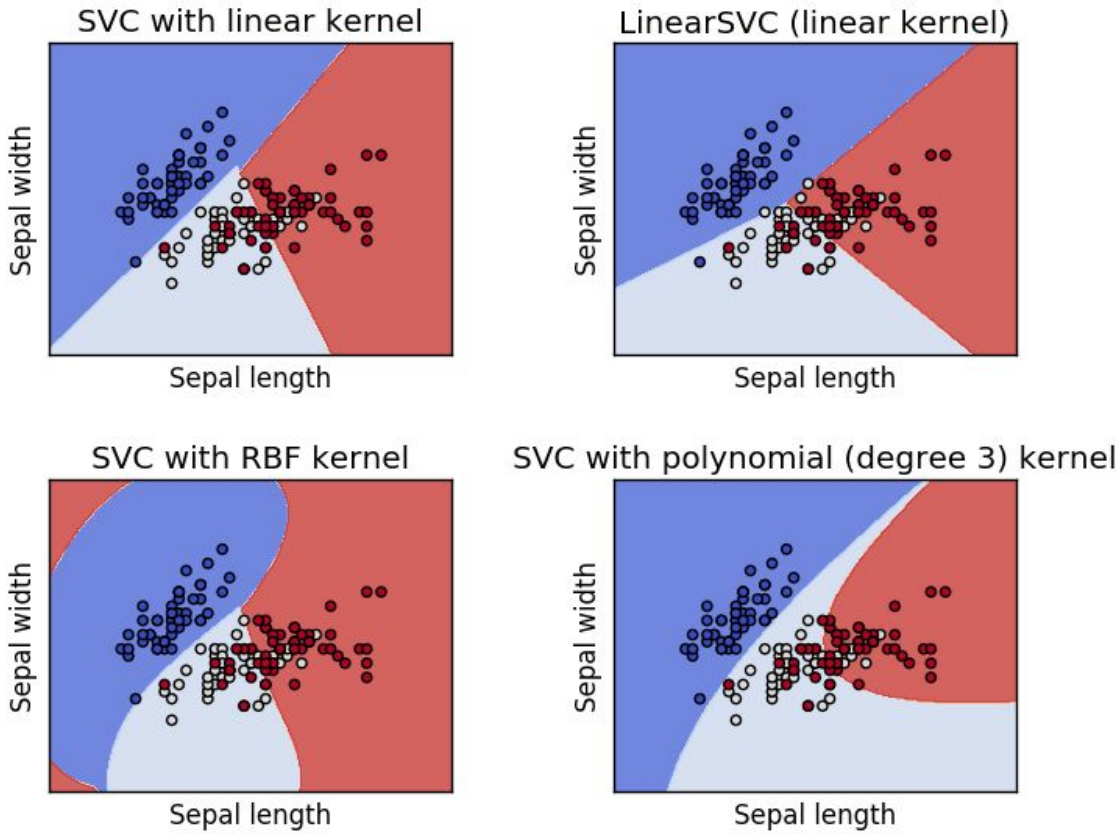
Two ensemble learning classification models (GradientBoostingClassifier and AdaBoostClassifier) outperforms other models with over 0.8 score. While LinearSVC has the lowest score ~0.52.

I also recorded the model computing time to compare the efficiency of each candidate:



Surprisingly SVC runs super slow compared to other candidates. After refer here I summarized the following reasons:

- The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples.
- LinearSVC would be a good replacement for SVC with faster computing time.



The above graph shows the difference between each SVC model applied on iris dataset. Ref here.

In the end, I chose GradientBoostingClassifier to deep dive on parameter tuning since it has the best score ~0.83, even runs the slowest 23 seconds.

Refinement

There's a great study about the parameter tuning for GradientBoostingClassifier, in this project, I'll focus on the following parameters:

- **learning_rate**
  - This determines the impact of each tree on the final outcome. GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates.

    - Lower values are generally preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize well.
    - Lower values would require higher number of trees to model all the relations and will be computationally expensive.
- **n_estimators**
    - The number of sequential trees to be modeled
    - Though GBM is fairly robust at higher number of trees but it can still overfit at a point. Hence, this should be tuned using CV for a particular learning rate.

I used GridSearchCV on the above 2 parameters to do tuning:

- 'learning_rate':np.arange(0.01, 0.2, 0.05)
- 'n_estimators':np.arange(100, 200, 20)}

Find the best parameter and score:

- Best Parameters: {'n_estimators': 120, 'learning_rate': 0.11}
- Best Scores: 0.833990675676

The best score is a little bit below the default parameter settings, I think that's because the parameter chosen are also close to the default. Considering the leaderboard scores, mine is pretty close now. I think it reached the limit within error range.

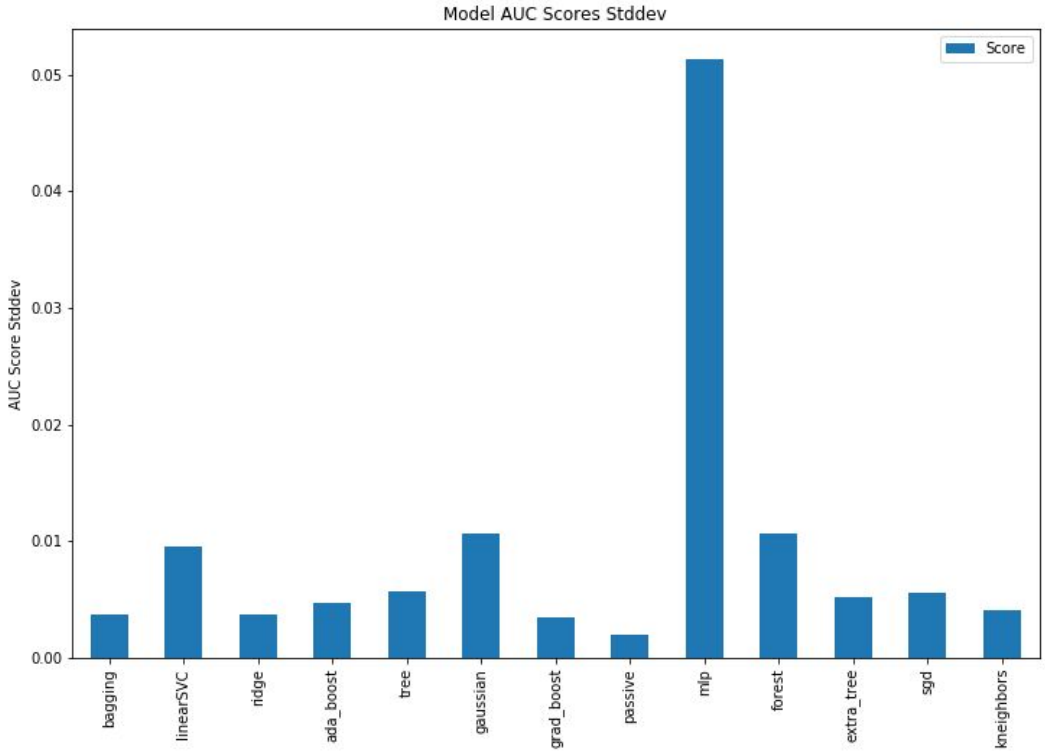
IV. Results

Model Evaluation and Validation

The final model is evaluated on the test data set provided by the Kaggle competition, with TARGET column removed. Kaggle will run the test result against 50% of the data, and have a competition score with the other 50%.

The final score with test data set validated from Kaggle is ~0.8340. That's less than 0.5% compared to the top leaders. I'm satisfied with the model performance.

For robustness of the model, I've calculated the standard deviation for each KFold test:



GradientBoostingClassifier is pretty stabled with 0.003513, while MLPClassifier has very high stddev 0.051356 with very low score 0.604032.

Considering the randomized train/test dataset, I think it's a robust, reliable and result satisfied model.

Justification

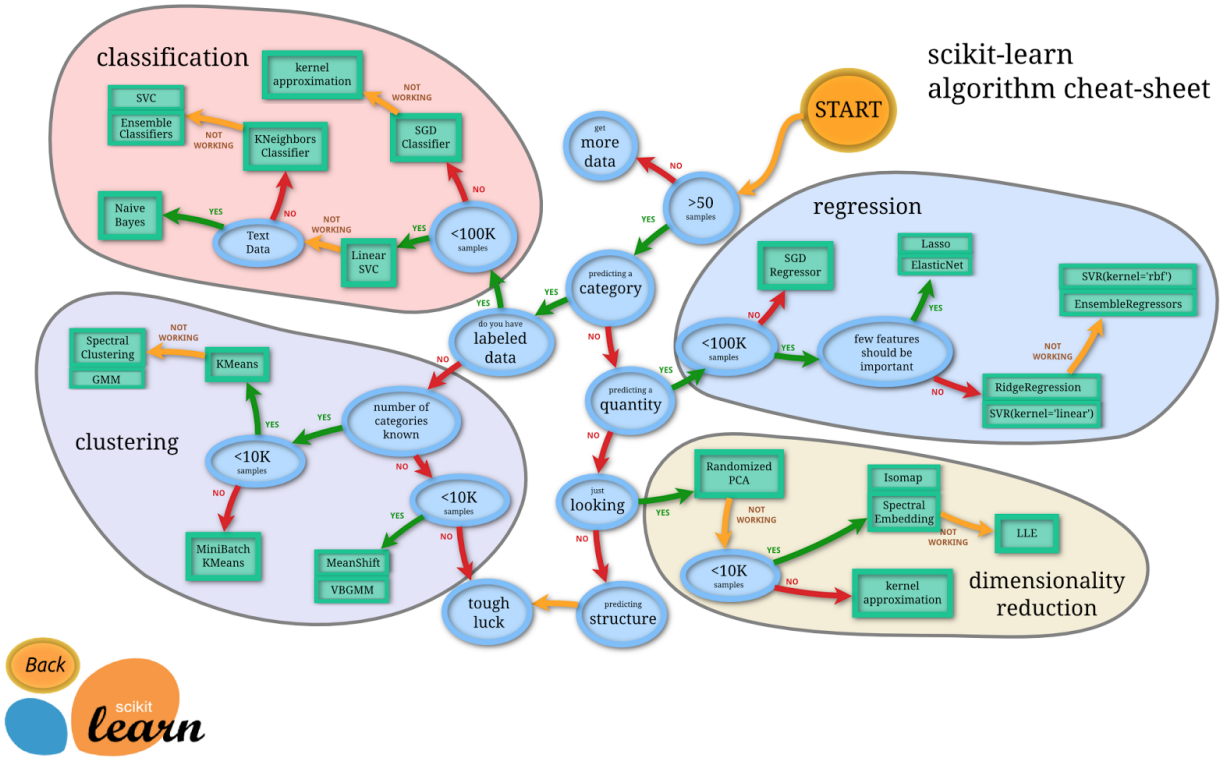
Comparing to the benchmark model using XGBClassifier from xgboost library, my final model performs as good as the benchmark result ~0.83. XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

That also proves the similar score between GradientBoostingClassifier and XGBClassifier because the algorithm and ideas behind these two classifier are pretty close.

V. Conclusion

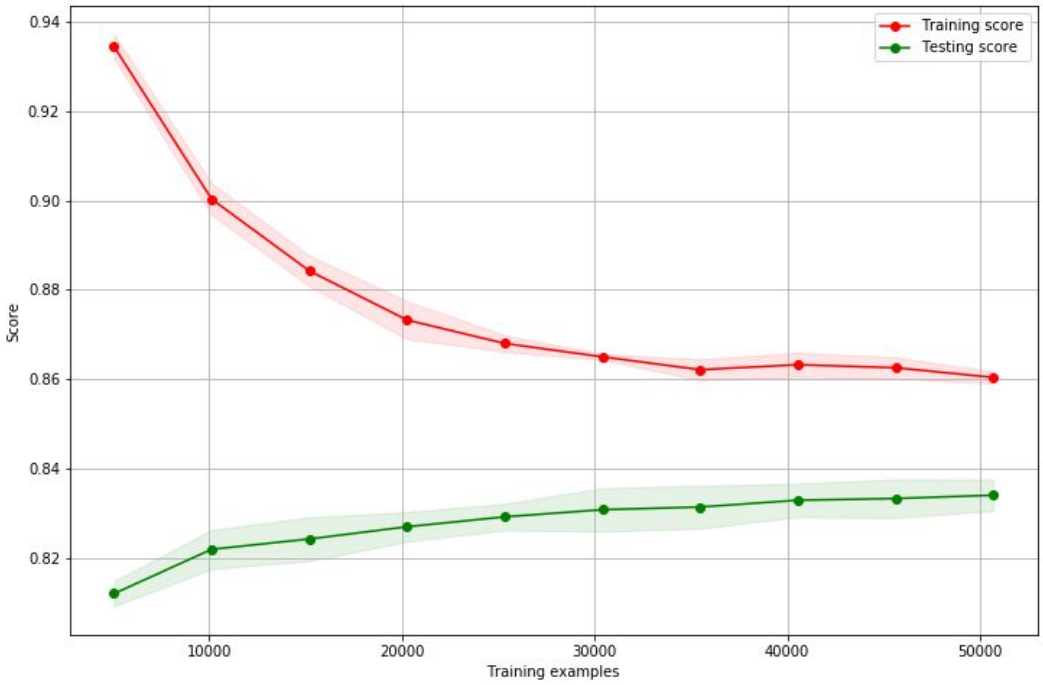
Free-Form Visualization

SKLearn provided a cheat sheet for how to choose estimator:



For this project, I've almost covered every models listed in the classification category. And the final winner is EnsembleClassifier.

I also did some exploration on the learning rate for GradientBoostingClassifier with default settings by utilizing the learning_curve from sklearn:



From the above graph, there's still a ~0.03 gap between the training score and testing score. However, starting from 40K samples, the testing score is pretty stable.

Reflection

The project is pretty interesting, it covered most parts of supervised learning topics:

- Data Exploration: Statistical analysis
- Data Cleansing and Pre-processing: Remove redundant, irrelevant and outlier data
- Data Visualization: Matplotlib
- Feature Selection: Tree based feature selection
- Model Evaluation: roc_auc score, learning rate
- Model Selection: Cross validation with multiple category classifiers
- Model Tuning: GridSearchCV with model parameters

The easy part for this project is data is provided by the competition and well-defined. The goal is clear to have a better score on desired metrics. So that's the guidance I

appreciate mostly. However, it still have lot of unknown parts. I think data exploring and parameter tuning are most interesting and challenging.

The data set originally has 700+ features, that makes manually analysis extremely difficult. By applying several steps of preprocessing, I reduced the feature size to 37.

Improvement

I'm pretty satisfied with the final score I got from the GradientBoostingClassifier. But some areas can still improve in my opinion:

- Parameter Tuning: There's no big difference between tuned model and default one. It might because the default already performs better and close to the limit, or I can try more options.
- Data Visualization: I only used basic bar graph to represent the distribution of data. I'd like to explore more advanced graphs to provide useful information. A good example here.