COMP9417 Project: Customer Transaction Prediction

Changxing Sun z5236644
Zhuolin Li z5272697

1 Introduction

There are all kinds of customers in this world, each with their own preferences. Determining whether a customer will buy the product or repay the loan has been a difficult problem for financial institutions to solve. With the increasing popularity of machine learning, is it possible to use machine learning to predict whether a customer will make a transaction or not? More backgrounds are available in the Santander Customer Transaction Prediction on Kaggle.

In this challenge problem, we are given an anonymized dataset containing numeric feature variables and a binary target column. The task of the project is to predict the value of the target column in the test set. We intend to build four models: Logistic Regression, Random Forest, Gaussian Naive Bayes and Support Vector Machines to accomplish this task. The final prediction will be evaluated on area under the Receiver Operating Characteristic (ROC) curve between the predicted probability and the observed target on Kaggle.

2 Implementation

In this section, we will mainly talk about data preprocessing, evaluation metrics and K Fold cross validation we use.

2.1 Data Analysis

Before building machine learning models, we need to first view and preprocess the datasets. The components of the training set are 200000 rows data, each row with an ID_code, 200 features and a target value. The target value is 0 or 1, representing two classes. The testing set is also composed of 200000 rows data, each row with an ID_code and 200 features. The features are named with var_0 to var_199. All the features are numerical and all the types are float64. There are no missing values (empty, null or NA) in both datasets.

We will train several machine learning models using the training set, then select the model that gives best performance, and use that model to predict the target value of testing set.

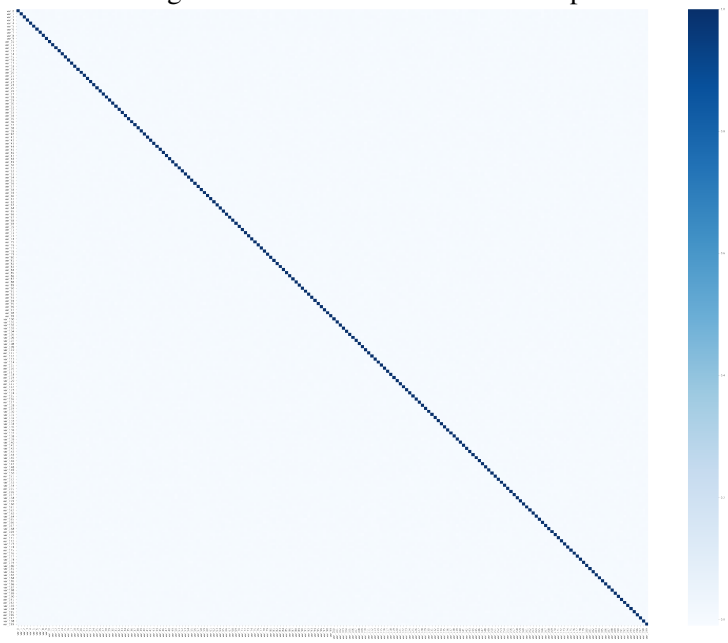
Because the names of all the features are not provided and there are no missing values in both datasets, we do not have much to do with cleaning the dataset. And we also cannot take the risk of dropping any of the features now.

Since all features are numerical values, so for each feature, we plot the probability density distribution for each of the two classes to get an intuitive comparison. (Plots are in appendix). From the plots, we can see the difference of probability density between two classes are very small for most features. But for some features, the difference is very large, such as feature var_81, var_80, var_139.... So, we make

the reasonable guess that these features will have greater impact on the models than other features. And we will verify the guess in the end.

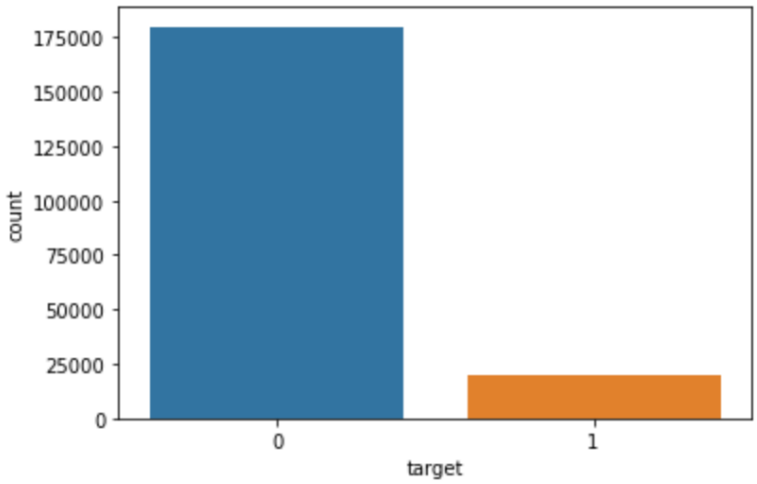
Next, we try to observe the correlations between all features. From the heatmap of all the correlations, it is obvious that there is very small (close to 0) correlation between any two features. So, we decide not to drop any features.

Figure 1: Feature correlation heatmap



Moreover, the two classes are very unevenly distributed in training set, with far more class 0 than class 1.

Figure 2: Classes distribution



2.2 Metrics

We will use 5 metrics to evaluate the final model: ROC AUC, accuracy, precision, recall and f1-score, of which ROC AUC is the Kaggle's evaluation metric. The definitions are as follows. (TP: True Postive, TN: True Negative, FP: False Positive, FN: False Negative)

- Accuracy means the probability that we predict correctly.

$$\text{accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

- Precision means the probability that we predict correctly among those predicted positive observations.

$$\text{precision} = \frac{TP}{TP+FP}$$

- Recall means the probability that we predict correctly among those actually positive observations.

$$\text{recall} = \frac{TP}{TP+FN}$$

- F1-score is the weighted average of precision and recall.

$$\text{f1-score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- ROC AUC is the area under the Receiver Operating Characteristic (ROC) curve, which is a graphic plot with the horizontal axis false positive rate (FPR) and the vertical axis true positive rate (TPR). True positive rate is also known as recall defined before. The range of the area is [0,1], and the closer to 1, the better the performance of the model.

$$\text{TPR} = \frac{TP}{TP+FN} \qquad \text{FPR} = \frac{FP}{FP+TN}$$

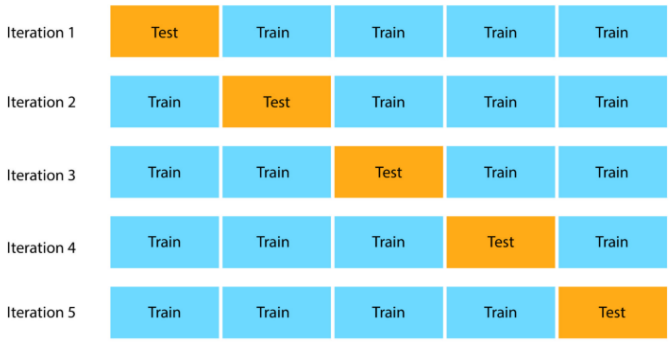
2.3 Cross Validation

Cross Validation is a model evaluation method that minimize the sampling bias of machining learning models. And cross-validation can avoid overfitting by checking how accurate the model is on multiple different subsets of data and ensure that the model will have good performance on future data.

We will use K-Fold Cross-Validation for each training model and the process is as follows:

1. Divide the original training sample into K subsamples of equal size.
2. For each unique subsample
   (a) Take the subsample as testing set.
   (b) Take the remaining K-1 folds as training set.
   (c) Run the model and record the score.
3. Average the K scores to get a single estimation.

Figure 3: Example of how Cross Validation works[1]



Larger K can make the size of training fold closer to the dataset, hence reduce the bias with true error, but larger k will also induce higher variance and increase computing burden as the iteration increases and size of training fold decrease.[2] Based on several values of K we experimented with and consider the training sample size, we choose K=5 which gives a good balance between score variance and computing time.

3 Modelling

3.1 Logistic Regression

3.1.1 Discription

Logistic regression models the probabilities for classification problems and it's an extension of the linear regression model. So instead of fitting a straight line or hyperplane as in linear regression model, the logistic regression uses the logistic function to squeeze the output of a linear equation between 0 and 1.[3] We will use the LogisticRegression classifier in scikit-learn to find the optimized feature vector for this project.

3.1.2 Parameters Choosing

We consider the following main parameters and let other parameters be default:

choosing Solver:

There are various optimization methods provided by scikit-learn. For standard **Gradient Descent** (GD), we go over the entire trading set on each weight vector update, this can result in high computation costs and only gives a linear convergence rate. The **Stochastic Gradient Descent** (SGD), on the other hand, updates the weight vector on each observation instead of evaluating the full training set, hence reduce the computation burden and get result faster than GD. But the convergence rate of SGD, which is O(1/N), is still not fast enough for large size of training sample. The **Stochastic Average Gradient** (SAG) method, introduced by Schmidt, Le Roux and Bach, reduce the convergence rate to O(1/sqrt(N)) by integrating a memory of prior gradient values.[4] **SAGA** (Defazio, Bach, Lacoste-Julien, 2014) is another gradient method inspired by SAG, also provided fast convergence but moreover support L1 regularization.[5] The Newton's Method uses a better quadratic function minimization because it uses both first and second partial derivatives, and so can be thought as a twisted GD with the Hessian (the Hessian is a square matrix of second-order partial derivatives), but it's computationally expensive because

of the Hessian Matrix.[6] The **Limited-memory BFGS** (L-BFGS) improve this situation by using an estimate of the inverse Hessian matrix and storing only a few vectors that represent the approximation implicitly.[7] So L-BFGS can provide good results for datasets that are not too large.

Therefore, we choose L-BFGS solver in our model considering the training sample size and result accuracy.

choosing regularization:

The hyperparameter **L1** and **L2** specified the norm in penalization. With larger penalty, the method will make more effort to satisfy the norm constraint, hence the coefficients are more likely to shrink to 0, which can help avoid overfitting. L1 tends to shrink coefficients to zero whereas L2 is more likely to shrink coefficients evenly, hence L1 is useful for feature selection because we can drop variables associated with small coefficients, and L2 is more useful for preserving collinear features. [8] Both L1 and L2 can be used in this project, but we will choose L2 because the L-BFGS solver only support L2 penalty in scikit-learn. We will tune the **C** regularization parameter in the model and the strength of the regularization is inversely proportional to **C**.

choosing C:

We first test the value of C in the range of [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000] using 5-fold cross validation for each value to find a roughly level of penalty. The result is shown in Table 1.

Table 1: The roc_auc_score of different choices of C under 5 fold CV

C	0.0001	0.001	0.01	0.1	1	10	100	1000
mean	0.840740	0.851064	0.854398	0.854106	0.854313	0.854306	0.854302	0.854286
std	0.000320	0.000872	0.001596	0.001657	0.002004	0.002024	0.002044	0.002037

We find C = 0.01 gives best average score around 0.854398. Therefore, we choose C=0.01 in our logistic regression model.

choosing other parameters:

We will let the **class_weight** in "balanced" mode because the distribution of class size in the sample is very uneven, and the "balanced" mode can automatically adjust weights inversely proportional to class frequencies in the input data.

3.1.3 Result

The metrics of our final Logistic Regression model is summarised in Table 2.

Table 2: The metrics of our final Logistic Regression model

roc_auc_score	accuracy	precision	recall	f1
0.854398	0.77746	0.77746	0.77746	0.77746

3.2 Random Forest

3.2.1 Discription

Random Forest is a widely-used ensemble learning algorithm used for classification based on Decision Tree. The algorithm constructs multiple number of decision trees by using feature randomly bagging and combine them to a forest. The final predicted class of the random forest is chosen by max voting, i.e. by most of decision trees.[9] We will use the RandomForestClassifier in sklearn package for this project.

3.2.2 Parameters Choosing

We need to choose following parameters: n_estimators, criterion, max_depth and min_samples_leaf.

choosing n_estimators:

n_estimators indicates how many trees there are in the random forest. A larger n_parameter will always provide better performance while it will decrease the speed. This is because the decision trees in the random forest are trained separately from each other, which doesn't lead to risk of overfitting with more decision trees.[10] Here we choose the largest parameter under a consideration of a balance of performance and speed, in this case, 10000.

choosing criterion:

There two options for criterion: gini and entropy. These two criterion are used when a decision tree is choosing a feature to split the nodes.

- Gini represents Gini Impurity. It is computed under the following formula.

$$GI = 1 - \sum_i p_i^2, pi : \text{the probability of class i}$$

- Entropy represents information gain, It is computed under the following formula:

$$IG = -\sum_i p_i \cdot \log pi, pi : \text{the probability of class i}$$

They both have the minimum value 0 when p_i equals to 0 or 1 and they both have the maximum value when p_i equals to 0.5. The difference is that the maximum value of Gini Impurity is 0.5 while the maximum value of Information Gain is 1. Both criterion have a similar property. However, entropy computation is much slower than gini because it computes the logarithm function. Hence, for time efficiency we choose gini as criterion.

choosing max_depth:

Parameter max_depth indicates the maximum decision tree depth we allow in the random forest. The larger the value we set, the deeper our random forest will be and the more information the model can process. As the value increases, the performance of the training set will be better. However, with a high max_depth, the model may overfit the training data and give a bad performance on the testing set. Hence, we use 5 fold Cross Validation to choose the best parameter among [5, 10, 15, 20, 30, 50, None]. Fix the other parameters and the result is shown in Table 3.

Table 3: The roc_auc_score of different choices of max_depth under 5 fold CV

max_depth	5	10	15	20	30	50	None
mean	0.872259	0.878903	0.879682	0.879967	0.879844	0.879844	0.879844
std	0.006983	0.006675	0.006685	0.007409	0.007258	0.007258	0.007258

Note that when we take the max_depth as 30, 50 or None, the outputs are the same value. This is because no trees will arrive at a depth more than 30. The nodes will not expand when there are less than min_samples_leaf samples in all leaves. Finally, we choose the max_depth = 20 that gives us the best roc_auc_score.

choosing min_samples_leaf:

Parameter min_samples_leaf is a stopping criterion that stops splitting when a leaf node's number of samples reaches the value. If the leaf node is small, then it will be more sensitive to noise in training data and more likely to capture them.[11] Again, we use 5 fold Cross Validation to test the values in the range of [20,50,100,200,300,500,1000]. The result is shown in Table 4.

Table 4: The roc_auc_score of different choices of min_samples_leaf under 5 fold CV

min_samples_leaf	20	50	100	200	300	500	1000
mean	0.889839	0.891111	0.890750	0.890156	0.889459	0.889048	0.885400
std	0.005459	0.004879	0.005571	0.005421	0.005253	0.005595	0.005534

We find when min_samples_leaf = 50, the score is the highest (0.891111).

3.2.3 Result

The metrics of our final Random Forest model is summarised in Table 5.

Table 5: The metrics of our final Random Forest model

roc_auc_score	accuracy	precision	recall	f1
0.891111	0.89985	0.89985	0.89985	0.89985

3.3 Gaussian Naive Bayes

3.3.1 Discription

Naive Bayes classifier is a simplified version of generative probabilistic model for classification based on the Bayes theorem. Gaussian Naive Bayes is a special type of Naive Bayes, which is often used for features with continuous values. Gaussian Naive Bayes has two basic assumptions:[12]

- All the features are mutually independent given the class value.
- The feature values in each class are distributed according to Gaussian or Normal Distribution.

We have already checked the features correlations and density in Section 2 and make sure two assumptions hold in our dataset. Hence, we can directly apply Gaussian Naive Bayes. We will use the GaussianNB in sklearn package for this project.

3.3.2 Parameters Choosing

Since Naive Bayes has very strong assumptions and a very complete model, it usually generalizes well and there are almost no hyperparameters to tune. Here we just choose the best var_smoothing value.

choosing var_smoothing:

Parameter var_smoothing is a value we add manually to the features' distributions variance to make the calculation stable. [13] We will test in the range of $[10^{-10}, 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}]$ using 5 fold cross validation. We find the var_smoothing = 10^{-9} (default value) gives the best ROC AUC score of 0.888408. The result is shown in Table 6.

Table 6: The roc_auc_score of different choices of var_smoothing under 5 fold CV

var_smoothing	10^{-10}	10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}
mean	0.888408	0.888408	0.888404	0.888365	0.888305	0.002743
std	0.002713	0.002714	0.002718	0.002742	0.002759	0.888153

3.3.3 Result

The metrics of our final Gaussian Naive Bayes model is summarised in Table 7.

Table 7: The metrics of our final Gaussian Naive Bayes model

roc_auc_score	accuracy	precision	recall	f1
0.888408	0.921515	0.921515	0.921515	0.921515

3.4 Support Vector Machines

3.4.1 Discription

Support Vector Machines (SVM) can efficiently perform a non-linear classification using kernel trick, implicitly mapping inputs into high-dimensional feature spaces. To be more specific, for many classification problems, the data set are not linearly separable in the original space, so SVM will map the original finite-dimensional space into a much higher-dimensional space where the separation is much easier. And to keep the computational load reasonable, the mappings used by SVM are designed to ensure that dot products of pairs of new input data vectors may be computed easily in terms of the old data vectors in the original space, by defining suitable kernel functions.[14] The optimization goal of SVM is to give the hyperplane that can not only classify the data but will also give the largest separation margin between the two class.

There are many different kernel functions for different types of problem, and we will use the linear kernel for linear SVM in this project. And to reduce computation burden more, we choose the SGDClassifier in scikit-learn, which use regularized linear classifiers with stochastic gradient descent (SGD) training.

3.4.2 Parameters Choosing

We will tune the following main parameters and let other parameters be default: loss function, penalty strength, learning rate and class weight. And we use 5-fold cross validation for each value of each parameter.

choosing Class Weight:

We will let the class_weight in "balanced" mode because of the uneven distribution of target class.

choosing Loss function, Penalty:

For loss function parameters, we consider the 'hinge' and 'squared_hinge', both use linear SVM, and the difference is that 'squared_high' is quadratically penalized. We test the following four combinations of loss parameter and penalty (L1 or L2, the same effect as discussed in logistic regression model), using the same list of regularization strength alpha, where larger alpha means stronger regularization. The result of four combinations is shown in Table 8.

Table 8: The roc_auc_score of different choices of alpha of 4 combinations under 5 fold CV

combination	alpha	0.0001	0.001	0.01	0.1	1	10	100
hinge + L2	mean	0.825143	0.830063	0.842274	0.842135	0.830078	0.799648	0.734618
hinge + L2	std	0.006654	0.004926	0.001358	0.002067	0.000461	0.002048	0.004443
squared_hinge + L2	mean	0.825928	0.810930	0.756356	0.742152	0.724586	0.697529	0.642921
squared_hinge + L2	std	0.003318	0.006263	0.004847	0.005533	0.002777	0.011122	0.015894
hinge + L1	mean	0.810563	0.806682	0.792145	0.703988	0.5	0.498698	0.5
hinge + L1	std	0.013418	0.001290	0.005673	0.029293	0.0	0.002604	0.0
squared_hinge + L1	mean	0.819395	0.809181	0.806022	0.791341	0.777197	0.796325	0.5
squared_hinge + L1	std	0.011799	0.002109	0.002695	0.011802	0.013383	0.005900	0.0

In summary, the combination of 'hinge' and 'L2' gives the best overall performance.

choosing Learning Rate:

The model will update the weighted vector with a decreasing strengthen because we want the hyperplane change more at the beginning than at the end in order to improve the convergence. We will choose the learning rate schedule between the 'optimal' and 'adaptive'. Since the default schedule is 'optimal', which is used for all trainings above, so now we only test 'adaptive' schedule. Moreover, we narrow down the alpha list to be centered at 0.01 based on above result.

Adaptive:

We test different values of ata0(the initial learning rate), and because it is initial, so we only test small values from [0.1, 0.01, 0.001, 0.0001, 0.00001]. The result is shown in Table 9.

Table 9: The roc_auc_score of different choices of alpha of different ata0 under 5 fold CV

ata0	alpha	0.001	0.005	0.01	0.05	0.1
0.1	mean	0.836745	0.827015	0.815659	0.768701	747034
0.1	std	0.003088	0.005605	0.011380	0.007727	0.008774
0.01	mean	0.839210	0.850498	0.851552	0.848589	0.844669
0.01	std	0.003947	0.002168	0.001212	0.000665	0.001625
0.001	mean	0.841149	0.848957	0.850212	0.847691	0.845074
0.001	std	0.003044	0.001932	0.001727	0.000739	0.000528
0.0001	mean	0.847781	0.849478	0.849819	0.847259	0.844135
0.0001	std	0.001755	0.001758	0.001176	0.000805	0.000639
0.00001	mean	0.847252	0.847409	0.847261	0.845892	0.843668
0.00001	std	0.001201	0.001154	0.001137	0.000747	0.000994

In conclusion, the combination of 'adaptive' learning rate schedule, 'hinge' loss function,'L2' penalty, alpha=0.01, ata0=0.01, gives best average score around 85.2

3.4.3 Result

The metrics of our final SVM model is summarised in Table 10.

Table 10: The metrics of our final SVM model

roc_auc_score	accuracy	precision	recall	f1
0.851552	0.76371	0.76371	0.76371	0.76371

4 Summaries

Table 11: The final metrics of our 4 machine learning models

metric	LogisticRegression	Random Forest	GaussianNB	SVM
roc_auc_score	0.854398	0.891111	0.888408	0.851552
accuracy	0.77746	0.89985	0.921515	0.76371
precision	0.77746	0.89985	0.921515	0.76371
recall	0.77746	0.89985	0.921515	0.76371
f1	0.77746	0.89985	0.921515	0.76371

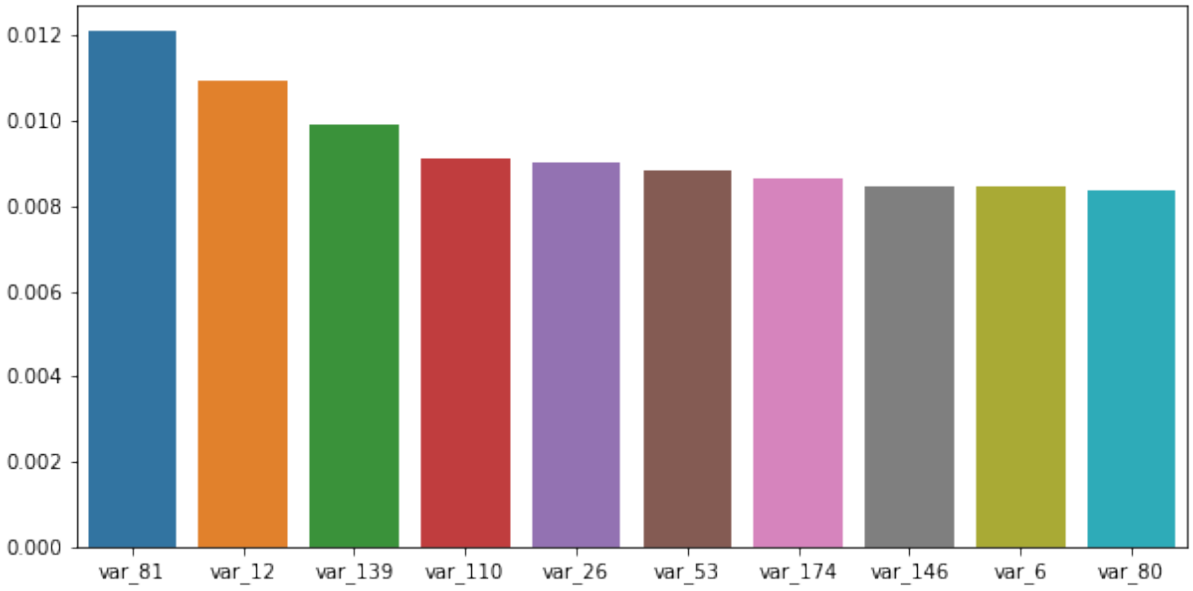
Among all the four machine learning methods, the Random Forest gives the best score of 89%. However, Random Forest is also the most sensitive to hyperparameter tuning of all models, and it is also the model that needs the greatest number of hyperparameters to test. At first its accuracy is only around 70%, and finally we improve its score to almost 90% through many attempts and research.

The second highest score of 88% Is given by the Naive Bayes model. The bayes classifier in scikit-learn not only gives good performance, but it is also a relatively mature model that trains data relatively quickly and is less sensitive to parameter tuning.

The Logistic Regression classifier and linear SVM optimized by gradient method are both more sensitive to penalty strength tuning and they all give the best average AUC around 85%.

For the top ten features with the highest weights in the Random Forest model, we can see that the density plots (Figure 5 in Appendix) of class 1 and class 0 all have great difference in these features. This verifies our guess in the Data Analysis part that features with greater class density difference will have bigger influence in the model.

Figure 4: Top ten features with the highest weights



Although the highest AUC we obtained is 89%, which is almost the best score on Kaggle (the highest score on Kaggle is now around 92%), it is still not good enough for applications. We think the most difficulty in improving the score is that we lack information about all the features' meaning. So, it is hard to create new useful features or make any features transformation. We believe better accuracy can be obtained if more information about features is provided.

References

[1] What is Cross Validation in Machine learning? Types of Cross Validation, mygreatlearning.com

[2] Choice of K in K-fold cross-validation, Stack Exchange

[3] Interpretable Machine Learning, christophm.github.io

[4] Minimizing Finite Sums with the Stochastic Average Gradient (inria.fr)

[5] The solver stuff, medium.com

[6] Logistic regression python solvers' definitions - Stack Overflow

[7] Limited-memory BFGS, Wikipedia

[8] The difference between L1 and L2 regularization (explained.ai)

[9] Random Forest, Wikipedia

[10] How to tune hyperparameters in a random forest, Stack Exchange

[11] Tuning random forest model, Analyticsvidhya blog

[12] Naive Bayes, Wikipedia

[13] sklearn.naive_bayes.GaussianNB, scikit-learn 0.24.2

[14] Support-vector machine, Wikipedia

Appendix

Figure5: Probability Density Distribution Plot for all 200 features

Test C for Logistic Regression Model

	C=0.0001	C=0.001	C=0.01	C=0.1	C=1	C=10	C=100	C=1000
score_0	0.840390	0.850363	0.853815	0.853316	0.853695	0.853728	0.853706	0.853697
score_1	0.841248	0.851481	0.854373	0.854040	0.854178	0.854107	0.854119	0.854045
score_2	0.840843	0.852187	0.856720	0.856770	0.857292	0.857342	0.857354	0.857340
score_3	0.840400	0.849773	0.851879	0.851724	0.851159	0.851135	0.851090	0.851107
score_4	0.840818	0.851518	0.855205	0.854679	0.855240	0.855218	0.855241	0.855243
mean	0.840740	0.851064	0.854398	0.854106	0.854313	0.854306	0.854302	0.854286
std	0.000320	0.000872	0.001596	0.001657	0.002004	0.002024	0.002044	0.002037

Test max_depth for Random Forest Model

	max_depth=5	max_depth=10	max_depth=15	max_depth=20	max_depth=30	max_depth=50	max_depth=100	max_depth=None
score_0	0.871307	0.875561	0.877784	0.877696	0.877919	0.877919	0.877919	0.877919
score_1	0.882846	0.888850	0.888977	0.889141	0.888587	0.888587	0.888587	0.888587
score_2	0.871391	0.879961	0.882422	0.884174	0.883673	0.883673	0.883673	0.883673
score_3	0.861106	0.868661	0.868509	0.867142	0.867039	0.867039	0.867039	0.867039
score_4	0.874645	0.881482	0.880721	0.881683	0.882000	0.882000	0.882000	0.882000
mean	0.872259	0.878903	0.879682	0.879967	0.879844	0.879844	0.879844	0.879844
std	0.006983	0.006675	0.006685	0.007409	0.007258	0.007258	0.007258	0.007258

Test min_samples_leaf for Random Forest Model

	min_samples_leaf=20	min_samples_leaf=50	min_samples_leaf=100	min_samples_leaf=200	min_samples_leaf=300	min_samples_leaf=500	min_samples_leaf=1000
score_0	0.889586	0.890508	0.889779	0.889466	0.888623	0.887810	0.883140
score_1	0.896117	0.895588	0.896182	0.895628	0.894448	0.894365	0.891982
score_2	0.888380	0.889774	0.888576	0.887395	0.887515	0.887219	0.883758
score_3	0.894519	0.896683	0.897268	0.896454	0.895640	0.895696	0.891076
score_4	0.880592	0.883000	0.881945	0.881834	0.881068	0.880150	0.877045
mean	0.889839	0.891111	0.890750	0.890156	0.889459	0.889048	0.885400
std	0.005459	0.004879	0.005571	0.005421	0.005253	0.005595	0.005534

Test var_smoothing for Navie Bayes

	var_smoothing=1e-10	var_smoothing=1e-09	var_smoothing=1e-08	var_smoothing=1e-07	var_smoothing=1e-06	var_smoothing=1e-05
score_0	0.886705	0.886703	0.886693	0.886625	0.886558	0.886457
score_1	0.884776	0.884774	0.884754	0.884638	0.884516	0.884336
score_2	0.888192	0.888194	0.888210	0.888255	0.888236	0.888006
score_3	0.889562	0.889562	0.889567	0.889556	0.889540	0.889547
score_4	0.892805	0.892805	0.892798	0.892751	0.892676	0.892420
mean	0.888408	0.888408	0.888404	0.888365	0.888305	0.888153
std	0.002713	0.002714	0.002718	0.002742	0.002759	0.002743

Test alpha for SGDClassifier with ‘hinge’ and ‘L2’

	alpha=0.0001	alpha=0.001	alpha=0.01	alpha=0.1	alpha=1	alpha=10	alpha=100	alpha=1000	alpha=10000
score_0	0.820410	0.834788	0.841144	0.842040	0.830338	0.801805	0.731696	0.730097	0.729761
score_1	0.833488	0.833413	0.842931	0.845226	0.829611	0.797472	0.728200	0.728847	0.728324
score_2	0.832611	0.828661	0.841585	0.842817	0.829463	0.799961	0.734848	0.735145	0.734926
score_3	0.817138	0.821091	0.841072	0.841805	0.830316	0.801892	0.737240	0.737684	0.737732
score_4	0.822066	0.832363	0.844638	0.838786	0.830662	0.797109	0.741104	0.742936	0.742923
mean	0.825143	0.830063	0.842274	0.842135	0.830078	0.799648	0.734618	0.734942	0.734733
std	0.006654	0.004926	0.001358	0.002067	0.000461	0.002048	0.004443	0.005140	0.005327

Test alpha for SGDClassifier with ‘squared_hinge’ and ‘L2’

	alpha=0.0001	alpha=0.001	alpha=0.01	alpha=0.1	alpha=1	alpha=10	alpha=100	alpha=1000	alpha=10000
score_0	0.825545	0.815018	0.757117	0.746556	0.724468	0.682722	0.628185	0.733811	0.667684
score_1	0.829734	0.808059	0.748930	0.731679	0.719524	0.699133	0.636585	0.730157	0.660020
score_2	0.820137	0.819625	0.763709	0.742176	0.725009	0.686967	0.631468	0.703865	0.664993
score_3	0.825679	0.801156	0.757939	0.746863	0.726124	0.710583	0.672331	0.643548	0.574396
score_4	0.828543	0.810791	0.754087	0.743486	0.727805	0.708237	0.646039	0.740097	0.577659
mean	0.825928	0.810930	0.756356	0.742152	0.724586	0.697529	0.642921	0.710296	0.628950
std	0.003318	0.006263	0.004847	0.005533	0.002777	0.011122	0.015894	0.035587	0.043293

Test alpha for SGDClassifier with ‘hinge’ and ‘L1’

	alpha=0.0001	alpha=0.001	alpha=0.01	alpha=0.1	alpha=1	alpha=10	alpha=100	alpha=1000	alpha=10000
score_0	0.809932	0.805834	0.796025	0.656409	0.5	0.500000	0.5	0.5	0.5
score_1	0.795083	0.804832	0.790993	0.711124	0.5	0.500000	0.5	0.5	0.5
score_2	0.832619	0.807038	0.793602	0.701542	0.5	0.493491	0.5	0.5	0.5
score_3	0.798756	0.808663	0.781877	0.702522	0.5	0.500000	0.5	0.5	0.5
score_4	0.816424	0.807042	0.798227	0.748345	0.5	0.500000	0.5	0.5	0.5
mean	0.810563	0.806682	0.792145	0.703988	0.5	0.498698	0.5	0.5	0.5
std	0.013418	0.001290	0.005673	0.029293	0.0	0.002604	0.0	0.0	0.0

Test alpha for SGDClassifier with ‘squared_hinge’ and ‘L1’

	alpha=0.0001	alpha=0.001	alpha=0.01	alpha=0.1	alpha=1	alpha=10	alpha=100	alpha=1000	alpha=10000
score_0	0.828306	0.806687	0.807527	0.774745	0.754319	0.792155	0.5	0.5	0.5
score_1	0.814092	0.806617	0.803544	0.784053	0.779012	0.799997	0.5	0.5	0.5
score_2	0.830736	0.811554	0.809746	0.800384	0.794983	0.799696	0.5	0.5	0.5
score_3	0.798727	0.810195	0.802381	0.808142	0.783692	0.788439	0.5	0.5	0.5
score_4	0.825112	0.810853	0.806913	0.789381	0.773977	0.801337	0.5	0.5	0.5
mean	0.819395	0.809181	0.806022	0.791341	0.777197	0.796325	0.5	0.5	0.5
std	0.011799	0.002109	0.002695	0.011802	0.013383	0.005090	0.0	0.0	0.0

Test ata0 and alpha for SGDClasifier with ‘hinge’ and ‘L2’

ata0 = 0.1

	alpha=0.001	alpha=0.005	alpha=0.01	alpha=0.05	alpha=0.1
score_0	0.833462	0.833127	0.795480	0.762032	0.742813
score_1	0.834991	0.824358	0.813550	0.761907	0.750013
score_2	0.840630	0.817454	0.827252	0.765865	0.733077
score_3	0.834316	0.831209	0.816301	0.771060	0.759531
score_4	0.840328	0.828928	0.825714	0.782641	0.749736
mean	0.836745	0.827015	0.815659	0.768701	0.747034
std	0.003088	0.005605	0.011380	0.007727	0.008774

ata0 = 0.01

	alpha=0.001	alpha=0.005	alpha=0.01	alpha=0.05	alpha=0.1
score_0	0.835792	0.850311	0.850435	0.847721	0.843355
score_1	0.837974	0.849575	0.850996	0.848704	0.842166
score_2	0.846101	0.854652	0.853455	0.849702	0.846390
score_3	0.835380	0.848374	0.850403	0.848674	0.845875
score_4	0.840801	0.849580	0.852472	0.848145	0.845558
mean	0.839210	0.850498	0.851552	0.848589	0.844669
std	0.003947	0.002168	0.001212	0.000665	0.001625

ata0 = 0.001

	alpha=0.001	alpha=0.005	alpha=0.01	alpha=0.05	alpha=0.1
score_0	0.842013	0.847219	0.848475	0.847140	0.844889
score_1	0.841660	0.849404	0.851443	0.848750	0.845932
score_2	0.841646	0.851932	0.852443	0.848325	0.845400
score_3	0.835559	0.846508	0.847951	0.846764	0.844478
score_4	0.844867	0.849723	0.850745	0.847477	0.844671
mean	0.841149	0.848957	0.850212	0.847691	0.845074
std	0.003044	0.001932	0.001727	0.000739	0.000528

ata0 = 0.0001

	alpha=0.001	alpha=0.005	alpha=0.01	alpha=0.05	alpha=0.1
score_0	0.846552	0.848501	0.848796	0.846710	0.843620
score_1	0.845542	0.850142	0.850711	0.848720	0.845349
score_2	0.850372	0.852481	0.851376	0.847358	0.843808
score_3	0.847275	0.847294	0.848201	0.846378	0.843699
score_4	0.849163	0.848969	0.850009	0.847127	0.844199
mean	0.847781	0.849478	0.849819	0.847259	0.844135
std	0.001755	0.001758	0.001176	0.000805	0.000639

ata0 = 0.00001

	alpha=0.001	alpha=0.005	alpha=0.01	alpha=0.05	alpha=0.1
score_0	0.846751	0.847612	0.845529	0.844846	0.843182
score_1	0.849094	0.848818	0.848340	0.847090	0.845474
score_2	0.848165	0.847288	0.848677	0.846004	0.843956
score_3	0.845813	0.845341	0.846901	0.845436	0.842676
score_4	0.846438	0.847987	0.846856	0.846086	0.843054
mean	0.847252	0.847409	0.847261	0.845892	0.843668
std	0.001201	0.001154	0.001137	0.000747	0.000994

Top 100 weighted features in final Random Forest Model