1 Introduction [20 points]

- **Group members**

Haotian Sheng	Kaggle ID: yooo
Yu Wu	Kaggle ID: dd
Sha Sha	Kaggle ID: Ss1996

- **Team name**
  dd

- **Performance score**
  2008 private: AUC: 0.79692 rank: 15
  2012 private: AUC: 0.78368 rank: 24
  Score: 18 [(0.79692 0.5) / (0.80685 - 0.5) 70% + (0.78368 0.5) / (0.79115 - 0.5) 30%] = 17.45

- **GitHub link**
  https://github.com/wuyudd/CS155_MiniProject_1

- **Division of labor**
  Haotian Sheng: tried the random forest model and Xgboost model, proposed K-fold cross validation method and handling unrelated features.
  Yu Wu: tried the linear models and Xgboost model, proposed handling same value features and almost the same value features and one-hot encoding.
  Sha Sha: tried the Xgboost model and optimized the parameters for Xgboost model, proposed handling no-response features and unbalanced data set.

2 Overview [20 points]

- **Models tried**
  - **Linear Models:** We have experimented with various linear models like Support-Vector Machines, Linear Regression, Logistic Regression and etc. The models behave similarly and Logistic Regression performs a little bit better in terms of AUC score.
  - **Random forest:** Random forest is an ensemble learning method by constructing a multitude of decision trees at training time and outputting the class or mean prediction of the individual trees. By sampling both data and features, Random forest helps reduce variance.
  - **Gradient Boosting:** Gradient Boosting is also an ensemble technique which produces a prediction model which ensembles weak prediction models, for here is the decision tree.
  - **Xgboost:** Xgboost is one of the fastest implementations of gradient boosted trees. It provides a parallel tree boosting in a fast and accurate way.
- **Techniques tried**
  - **Manual feature removal:** Manually remove obviously unrelated features and features that (almost) every sample has the same value.
  - **One-hot encoding:** Remove meaningless and incorrect ordering information and convert to one-hot vectors for better performance in prediction.
  - **Handling no-responses:** Replace the value -1 in the data set which represents no response with **NaN** to mark the missing data.
  - **Data balancing:** Deal with the unbalanced data set.
  - **K-fold Cross Validation:** Split the training data into k folds, use every fold as validation data set in turn to test the model trained from other data.
- **Out of ordinary**
  - **Handle no-response data:** Weakens the influence of no-response values on our trained model.
  - **Xgboost:** Provides a parallel tree boosting speeding up the training and flexible parameters to tune so as to adjust for different data with better prediction performance.
- **Timeline**
  Day1 - Day2: Train different models on the raw data.
  Day3 - Day4: Data processing.
  Day5 - Day6: Optimization and adjust the parameter of selected model.

3 Approach [20 points]

- **Data processing and manipulation**
  - **Manual feature removal**
    After observing the training data, we tried to deal with the following three kinds of features:
    1. Unrelated features: The id" column in the data is meaningless since it is randomly given to distinguish the interviewees, thus the value can confuse the model. Therefore, we dropped this feature and the validation error obviously decreases.
    2. Same value features: Some features have same value for each sample such as "HRMONTH" and "HRYEAR4". Therefore, they may contribute little in distinguishing data and thus we dropped such features. According to the experiments, the improvement is trivial.
    3. Almost the same value features: "Almost same value" means most data have same value of the feature, which may also have little impact on the final model. We tried two ways: 1) drop the feature directly; 2) reconstruct the feature by simplifying the values into two categories with main and the others. The results turned out the performance didn't improve, so we didn't do such processing in the final model.
  - **One-hot encoding**
    One-hot encoding is used to remove meaningless and incorrect ordering information. The values of some features only represent specific categories but can be misinterpreted as ordering. Thus we applied one-hot encoding on such features.

    We trained our model on the modified data set but the performance is not as good as we expected when applying Xgboost. It may because one-hot encoding takes a compact variable and turns it into a whole bunch of sparse variables. But a Tree tries to increase information gain on the data it's splitting at any given level. Therefore, the potential result of one-hot encoding is that a tree algorithm will down play and even ignore the sparse one-hot encoded features. According to the analysis and experimental results, we decided not to apply one-hot encoding on the features.
  - **Handling non-responses**
    The value -1 meaning that the interviewee chooses not to answer should be treated as missing data, and the value may confuse the model. Since Xgboost could deal with missing data, we replace all the -1 with **NaN** and the validation error decreases.
  - **Data balancing**
    We found the ratio of 1 and 0 close to $\frac{1}{3}$, which indicates an unbalanced data set. Therefore, we gave more penalty to the mis-classified positive data points by setting the scale_pos_weight parameter but the accuracy did not improve, so we didn't apply it.
- **Details of modeling techniques**
  - **Logistic Regression**
    Logistic regression(LR) is used on when the dependent variable is dichotomous (binary). One

key part in LR is to set regularization to avoid over-fitting. We studied the regularization parameter of this model:

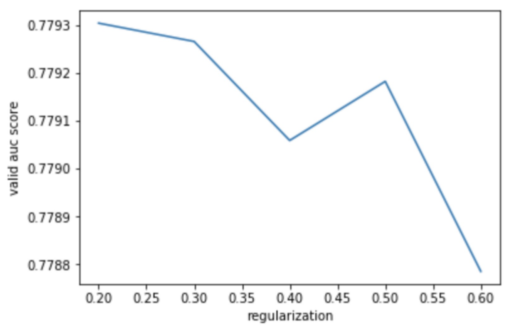

Figure 1: AUC v.s. Regularization Parameter

From the plot we can see the AUC achieves maximum around 0.78 when the regularization is around $0.2 \sim 0.3$. Considering model complexity and over-fitting, we chose $C = 0.3$. Other linear models like SVM also perform not well on the given data set as it contains much invalid information and is complex and unbalanced. Therefore, we consider more sophisticated ensemble models.

– **Random Forest**

The Random Forest (RF) is a classic ensemble learning method. Based on bagging and extended to feature sampling, RF reduces the variance by generating Bootstrap S from S and training based on the previous prediction. Given these features, taking the average prediction at the end, the RF performs fairly but not satisfyingly well.

When training the model, one critical parameter is the number of estimators. Rendering a more complicated model by increasing number of trees in the forest puts us at the risk of over-fitting while a small number of estimators may introduce an under-fitting model. We ran RF several times with different number of estimates ranging from 500 to 1000 with max_depth of 11 and 15:

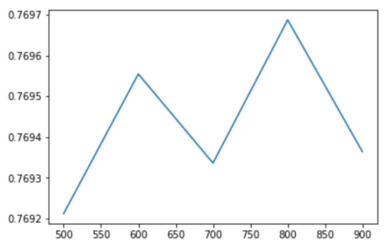

Figure 2: AUC v.s. N (max_depth = 11)

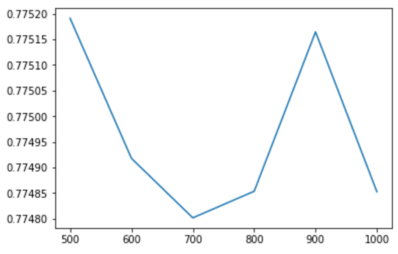

Figure 3: AUC v.s. N (max_depth = 15)

As we can see in the plot that max_depth of 15 performs better overall with higher accuracy on the validation set. However, due to the randomness presented in the curve, there is not enough information on how to choose the number of trees. In order to get a better insight on the influence of n_estimates we enabled the OOB feature, which uses out-of-bag samples to estimate the generalization accuracy during the training. This time we ran the model with three types of max_features(the number of features to consider when looking for the best split), log2, none(number of features) and sqrt.

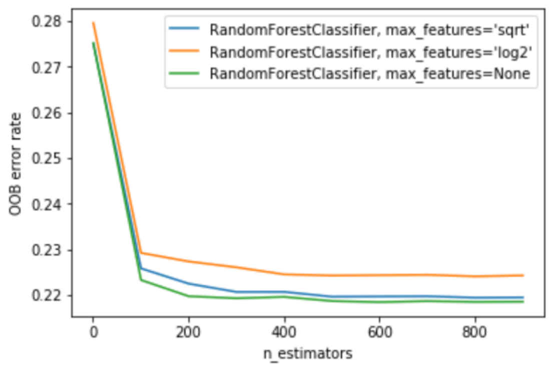

Figure 4: AUC v.s. OOB error rate

As we can see from the plot that as n_estimators increases, the error rate in the cross validation reduces at first and tends to be stable after 800. At the meantime, the max_features of n_features gives the lowest error due to more features are considered when looking for the best split. As we reached to a sophisticated random forest model, the test accuracy we got from Kaggle is 78.4% which is not bad but not satisfying enough so that we move on to the next ensemble model.

– **Xgboost**
  Selected model, see details in Model Selection section.

4 Model Selection [20 points]

- **Scoring**
  This competition is to predict whether people vote or not, which is a binary output classification. Therefore, we use classification error as the optimization objective. We used K-fold Cross Validation Method to test the model we trained in order to avoid over-fitting. Every time we separated the data set into 3 folds, and then train on 2 of them. The last fold is used as validation data. Repeat the process for 3 times on different 3 folds. We choose model according to both its in-sample and validation AUC. The model with high average AUC and low variance usually performs better on test data set. Among the models we tried, Xgboost performs the best. Therefore, we selected it as our final model.

- **Validation and test**
  After trying Gradient Boosting, we found it a promising model. In order to further improve the performance, we tried Xgboost, which is more powerful because of its clever penalization of trees, a proportional shrinking of leaf nodes, extra randomization parameter and some others. In addition, Xgboost model could deal with the missing data, which is quite suitable for our training data. We selected Xgboost as our final model.
  As mentioned before, we used validation method to evaluate our model. For the boosting algorithm, the number of estimators is crucial. Here we studied the impact of the number of estimators.

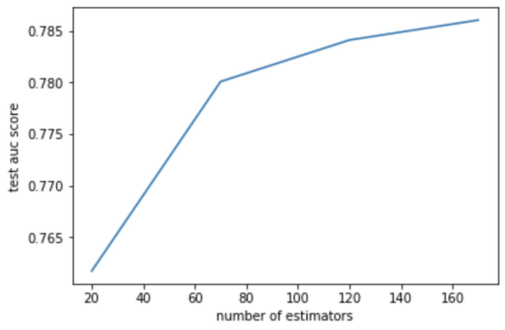

Figure 5: AUC v.s. N

From the plot, we can see that as the number of estimators increases, the AUC increases as well. At first, it increases very quickly, but when the number of estimators becomes larger, it increases much more slowly and may begin to decrease. This is intuitive because as the number of estimators increases, the model trained could fit more details of the training data. However, once it is too large, there may be over-fitting. Therefore, the number of estimators could be set around 150, where the AUC is good enough and over-fitting is little.
Since our Xgboost is in the form of an ensemble of decision trees, the maximum tree depth may affect the model a lot. Also, in order to avoid over-fitting, we also need to add some regularization to the model. Therefore, when training the model, we mainly tuned the such parameters to achieve a better performance. We tried different combinations of the parameters and test each model by validation

Table 1: Parameter tuning

n	gamma	max_depth	child_weight	subsample	colsample	valid auc
150	0.3	8	4	0.8	0.8	0.7901
150	0.3	9	4	0.8	0.8	0.7894
200	0.3	8	4	0.75	0.75	0.7895
150	0.3	8	4	0.75	0.75	0.7898
150	0.35	8	4	0.75	0.75	0.7897
170	0.35	8	4	0.75	0.75	0.7898
130	0.3	8	4	0.75	0.75	0.7896

AUC, the result is show in the table, where n, max_depth, child_weight, subsample, colsample, valid AUC respectively represent the number of estimators, the maximum depth of the tree, minimum sum of instance weight needed in child, subsample ratio of the training instances, subsample ratio of columns when constructing each tree, and the average validation AUC.

We choose several parameters which has higher validation auc to submit to Kaggle, and found the one with highest validation auc also performs the best.

5 Conclusion [20 points]

- **Insight**
  - **Top features**

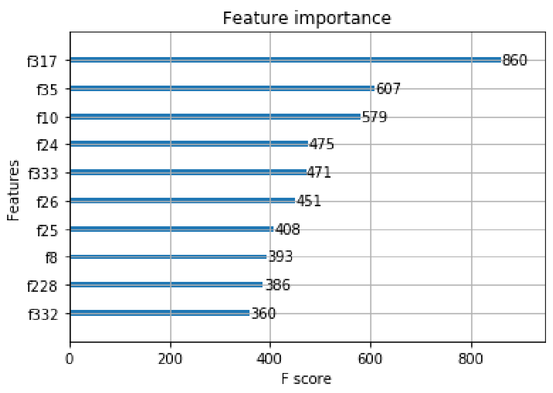

Figure 6: Top 10 features score

Table 2: Top 10 features

PXSCHLVL	ALLOCATION FLAG
PEPARENT	LINE NUMBER OF PARENT
HUTYPB	TYPE B NON-INTERVIEW REASON
GEREG	REGION
PEIO1ICD	INDUSTRY CODE FOR PRIMARY JOB
GESTFIPS	FEDERAL INFORMATION PROCESSING STANDARDS (FIPS) STATE CODE
GESTCEN	CENSUS STATE CODE
HEPHONEO	IS A TELEPHONE INTERVIEW ACCEPTABLE
PWFMWGT	FAMILY WEIGHT
PWCMPWGT	COMPOSITED FINAL WEIGHT

We can see from the top features that whether the people vote or not is closely related to the the region they settle in, the family background and the job industry they are in.

  - **AUC**
The implicit goal of AUC is to deal with situations where we have a very skewed sample distribution, and don't want to over-fit to a single class. In this competition, from the training data set, we know the sample distribution is unbalanced. We do not want the model to over-fit to the main class. If we use accuracy as metric, we may get a pretty accuracy by just saying all the data is in the main class since the sample distribution is skewed. Therefore, using AUC is a better metric for this project.

  - **Parallelizable methods**
Although Xgboost has to run multiple trees to add to the model sequentially, the parallelism happens during the construction of each tree, at a very low level. Each independent branches of

the tree are trained separately. Xgboost could parallelize the split finding for different features at the same level. Before building trees, it could sort the instance of node by the feature value globally and save it, which also made it possible for parallelism.

- **Something new**
  We learned the beauty lying in this handful model, Xgboost. While this tool did not appear in the lecture or homework, we indeed learned its underlying mechanism in the class, decision trees and ensemble methods. The new thing here is how to employ these concepts we already known into the field. The Xgboost is our bridge from theories to practice. We learned how to make trade off between the complexity and prevention of over-fitting. We also gained many first-hand experience in visualizing the performance of models.

- **Overall**
  Overall, we found the Kaggle project interesting and challenging as we learned all aspects of performing a learning task such as data processing, model selection and parameter tuning. As we moved along the way, we started to really first find the problem to need to solve and how would we solve it. Rather than given instructions like we were during the homework, we now need to closely examine and decide each step we make and the intuition during each step.

  In the data processing state, we learned that a bad processing action is worse than no action at all. When we one-hot encoded some sparse categorical features, it really expands the set and make features sparser, which is not a game thing for models basing on decision trees.

  In the parameter tuning stage, we learned that complexity does now always render better performance on the validation set due to the over-fitting issue. It is rather preferable for us to bring down the complexity of the model, sacrifice the training accuracy for better testing performance.

- **Challenges**
  As we found the course of pursuing higher test accuracy inspiring and interesting, we indeed encountered many challenges in the process. One challenge we met is the pre-processing of the data. As We faced a highly complicated and unsymmetrical data set, it is crucial that we pass a well processed training set to our models in order to get more juice out of them. We tried different kinds of mechanisms to trim the set, such as abandoning features that have same value in it, encoding the categorical features, mapping all the non-responsive values to ground and so on. We tried many combinations of these procedures and could not find the best way to engineer our training set. Even in some situations, we might need to remove some data processing procedures to improve the performance of the model. We think it might take more knowledge and experience for us to gain better intuitions on data processing.

  Another challenge we faced in the course is the selection of parameters of our model. For example, when we adjusted Xgboost, we have so many parameter choices, the depth and regularization parameters. We could definitely run some grids to gain better insight on the parameters; however, because of the elongated running time and complicated combination of parameters, it was hard for us to get a correct direction at first. We guess as we gain more experience in playing with these models, we gain better insight in parameter tweaking.