DS-GA 1001 Project Final Report

Name & NetID:
Li Lin Qin llq205 Xue Yang xy990 Yiran Xu yx1350 Han Zhao hz1411

Donorschoose.org is a public charity that connects the public to public schools in need. Public school teachers seek help by posting projects on the website such as requesting books, technology or other things for educational purposes. People can go on to the website and select a classroom project to support by donating money. After the required amount of donation is reached, Donorschoose then orders the requested goods to the teacher in need.

Business Understanding

We are picturing ourselves as data scientists hired by the website. Our goal is to predict whether a project will fail to achieve its donation goal and to rank projects in the website. With this information in hand, the website will be able to improve its user experience in several aspects and boost profits.

With every new project posted, the website may predict its probability of failure with the prediction model. If the project is predicted to fail, the website may suggest the person to write a longer essay, or to adjust to a lower budget goal. In order to boost its donations, the website may also send emails to potential donors. Another important implementation would be determining the ranking of the projects in the website. Projects with predicted higher failure rates should be placed on top of the page to increase their chances, while those with predicted low failure rates at the bottom. With features like these added in the website, the overall success rates of the projects should be increased, which would attract more recipients as well as donors to the website. Since 15% of the donations are dedicated to the website itself, this will also increase the total profits of Donorschoose.org.

Data Understanding and Preparation

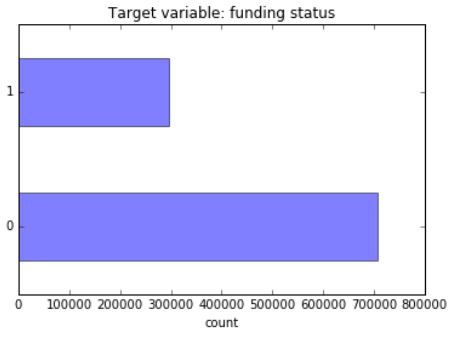
We obtained all the datasets directly from Donorschoose.org. The original uncleaned dataset has over one million observations with 45 variables. After cleaning, we have 1,004,604 observations and 32 features in total.

Some important features include: project categories, types of resource requested, poverty level of the school, and the grade level the project is targeting.

Since we are more interested in predicting a project's failure, we labeled the target variable "funding_status" with value 1 as failure and 0 as success. In the original dataset, the target variable contained four status: completed, reallocated, live, expired. We dropped the live projects, since they don't provide any results. We also changed the categories "reallocated" to "expired", as they are also counted as failures.

We then dropped features that clearly had nothing to do with our target variable, such as project ids, the date that a thank you card was sended to donors. To generate new and more useful features, we did some calculations on existed features. With a total of 50 states, the school states variable makes it really complicated in terms of utilizing one hot encoding. So we looked for State GDP per capita and compared each state's GDP value to the national average. We separated projects into three subgroups, whether their located state has a higher, lower, or about the same GDP compared to the the national average. We also added a new variable of time duration of projects using the datetime package. For numerical variables, we identified and replaced missing values with mean values. One hot encoding was used for all of the categorical features.

After that, we filtered features based on heuristics by picking the ones with high mutual information and correlation with the target variable. As you can see in Figure 1 in the Appendix, we dropped variables that have mutual info score close to zero. Three of our features are numerical so we also scaled to them to have mean 0 and variance 1. Finally, we split the original dataset into test and training subset where we gave 75% of the data to training.



Bar chart of Target variable: Funding status (1 indicates failure; 0 indicates success)

In summary, we have a total of 296,565 positive labels and 708,039 negative ones. Class probability estimation close to 1 means that the project has a high probability of failure. On the other hand, estimation close to 0 means that the project is very likely to be completed. Therefore in our case, the closer to 1 a probability estimation is, the higher rank a project should be placed in the webpage.

- **Essays as a feature**

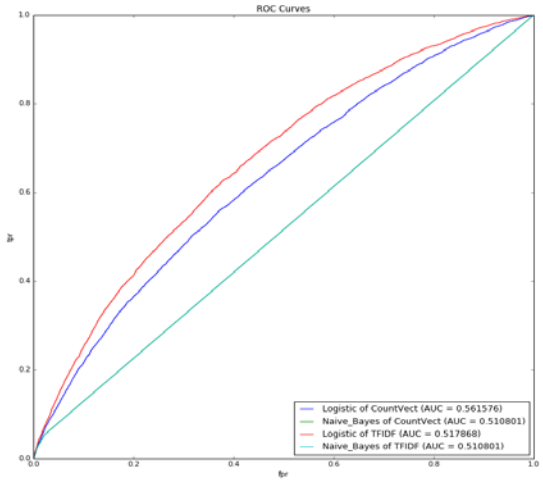
Since the provided dataset has a limited number of features, we also consider the descriptions of each donation project as a factor. There is a dataset of essays available in the website, so we merged it with the previous dataset by project id numbers.

At first, it is hard to observe any pattern, so we decided to cluster the essays by k-means, and later label each essay according to its located cluster. By using the NLTK package, we tokenized each essay and stemmed each token. A pandas DataFrame was then created with the stemmed vocabulary as the index and the tokenized words as the column. Next step, we converted the essay list into a TF-IDF matrix by tuning several parameters. Using the matrix, we incorporated k-means to better understand the hidden structure within the essays. We first chose number of clusters to be 17 (number of primary subject areas in the dataset) and then gradually increase the number. Each observation is assigned to a cluster to minimize the within cluster sum of squares. Next, the mean of the clustered observations is calculated and used as the new cluster centroid. Then, observations are reassigned to clusters and centroids recalculated in an iterative process until the algorithm reaches convergence. But the results are not as expected. After printing out the top 20 words in each cluster, we observed that they are almost identical words without distinct meanings, some examples including “learn”, “classroom”, “teach”, “work”, “books”, “love”, “materials”. This makes us reluctant to label the essays by clusters since we cannot really tell the difference between them.

The lecture of spam filtering made us realized that we should treat the essays as a separate dataset and classify them solely based on the target variable. We used methods similar to the homework, the binary count vectorizer and the TF-IDF vectorizer to turn the large amount of text into useful matrices. With these new features,

we split the data into training and testing sets and apply two models to each matrix, i.e. the Logistic Regression and the Naïve Bayes. At the first try, we used the default settings, along with original single words as well as bigrams, and an “english” stop word list. The AUC scores for all four curves turned out to be in the range of about 0.489 to 0.501, even worse than a random classifier. We then adjust different parameters of each model, including the ngram_range, max_df, min_df. And the best results, as you can see in the following graph, was generated by the Logistic Regression model of count vectorizer. But out of the four curves, even the highest AUC score is just about 0.56. Therefore regrettably, this method of feature extraction is not effective enough. We must look for another way to make use of the text data.

We printed out several misclassified essays by the logistic model. A few of the ones should have been successful but instead classified as failures have something in common: the essays are too short. This means that the model has somehow perceived that longer essays should have a higher success rate. Since none of the above models provide a satisfying output, we settled for counting the number of words in the essays. The essays are now divided into three categories: one with more than 350 words, one with less than 200 words, and one in between.



The model with highest AUC score: CountVectorizer*(stop_words = "english", ngram_range=(1, 2), max_df = 0.8, min_df = 0.2, binary = True)*

Modeling & Evaluation

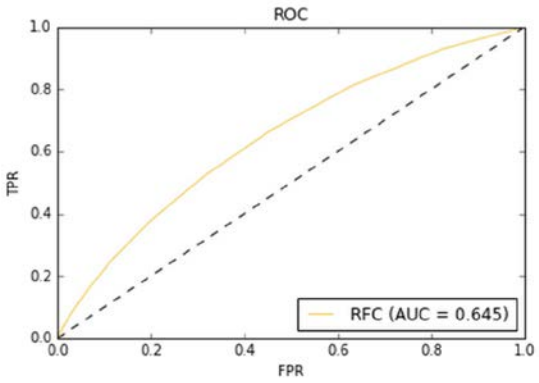
Since our business and data mining problem is to predict failure rate and rank projects, we use Area under the Receiver Operator Curve(AUC) as our evaluation metric.

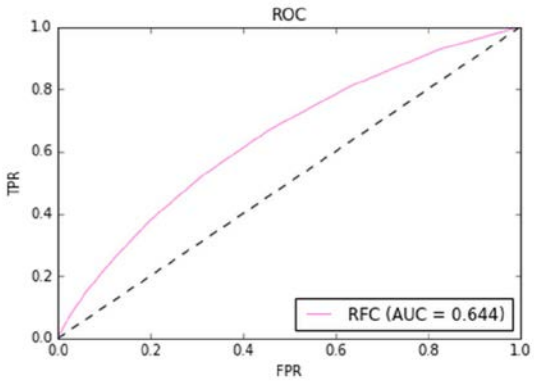
1. Random Forest

It is excelled in accuracy and runs efficiently on large data sets. It can handle thousands of input variables without variable deletion. As the forest building progresses, it generates an internal unbiased estimate of the generalization error. Random forest does not overfit, can run as many trees as you want, but storage requirements grow as the number of trees increases.

1.1 Baseline Model

We first ran two Random Forest models with default settings, one on the original data, the other on the scaled data. There are not much differences between the two as we can see below.

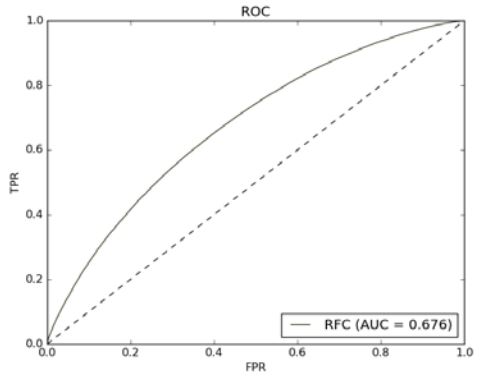




RandomForestClassifier(*n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_split=1e-07, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None*)

1.2 Grid search

To improve our baseline model, a grid search is necessary. We usually want to overfit the individual trees, and use some hold-out method to optimize Forest level parameters. We first run a grid search regarding to the parameter n_estimators, which is the number of trees in a random forest. It turns out that the greater n_estimators we choose, the better the model performs. Due to the time-consuming nature of the model, we choose n_estimators to be 500, and the ROC and AUC remain stable. In fact, the model also performs fairly well when we choose 200 n_estimators, with shorter training time but a slightly lower AUC.



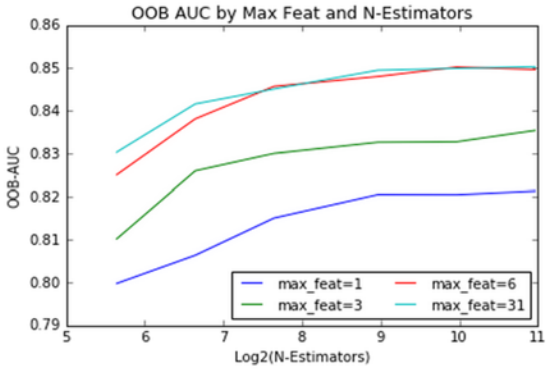
RandomForestClassifier *(bootstrap=True, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)*

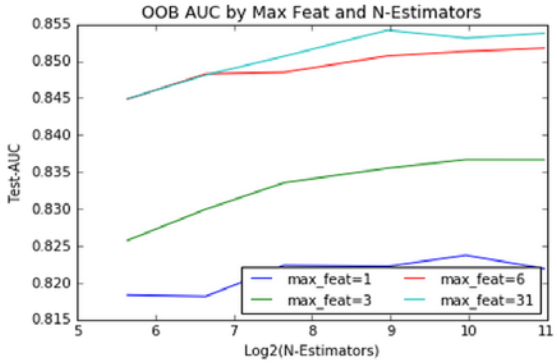
1.3 Out-of-bag Error

Due to the large amount of data, we spent about a whole day to run the OOB evaluation and holdout evaluation but did not produce any outputs. Therefore, we randomly select 10% of the samples to generate a biased result. According to the plot below, the forest performs better with more trees(n_estimators), but this effect tapers off asymptotically. Having too few features is suboptimal(max_feature =1, 3), and when max_features are 31 and 6, the model performs well.

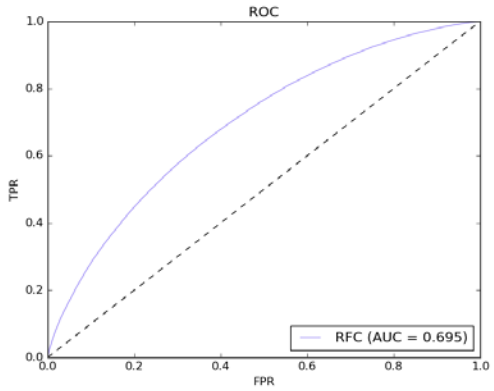
According to the test sets AUC in these models, we can see similar pattern in the holdout evaluation as well as in the OOB evaluation, but there are also some differences. At a certain level of max_features, more trees

in the random forest(n_estimators) hurts the performance of the model, which reflects in the decrease of AUC. The best model choice is log2(n_estimators)=9, for which the n_estimators is about 500 and max_features is 31.





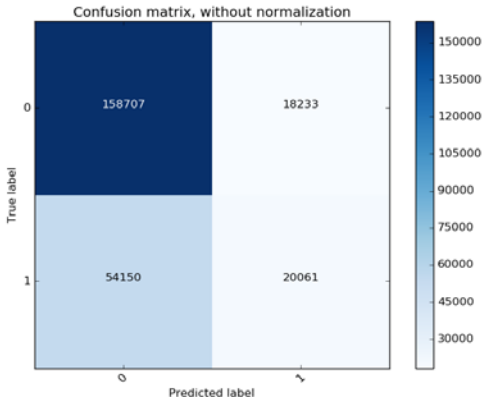
From the results, the optimal parameters are: n_estimators=500 and max_features=31(None) for the Random Forest model. The resulting ROC graph is as below.



RandomForestClassifier *(bootstrap=True, class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)*

1.4 Evaluation Metrics

According to the confusion matrix, we have precision = 0.52, while the base rate = 0.29, so we have lift that's approximately 1.79. Compared to other models, it has a higher recall = 0.27. We may say this model is better at identifying true labels as it has fewer false negatives, but at the cost of a higher false positive rate.



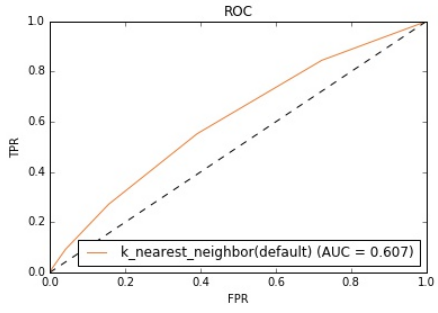
Confusion matrix of Random Forest, without normalization

2. K-Nearest Neighbors Algorithm (k-NN)

k-NN algorithm is easy to understand and implement. However, memory storage and time complexity are big issues because k-NN retrieves the whole dataset when making predictions. Time complexity for k-NN is O(n), so k-NN runs very slow for our data. Moreover, the optimal parameter k depends on each dataset, so we should be careful choosing k to achieve a good balance between bias and variance.

2.1 Baseline model

To see how k-NN performs and get a baseline model, we first ran k-NN with a default setting on our scaled data. The result is presented below.



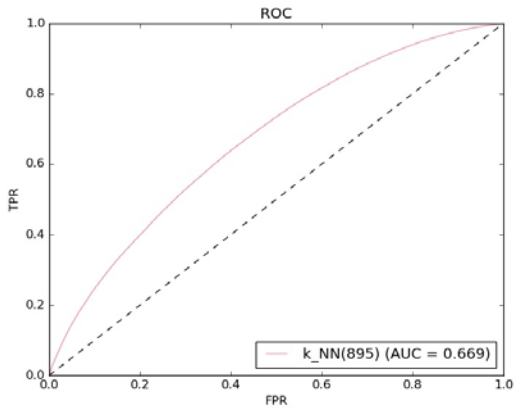
KNeighborsClassifier*(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='uniform')*

2.2 Grid Search

To improve model performance, we first looked carefully into the parameters. First, the default 5 neighbors are too small compared to how much data we have. Secondly, since weights for closer neighbors should be greater, we decided to change the parameter "weights" from default "uniform" to "distance". For the other parameters, we decided to run a grid search to find optimal ones.

A grid search was done on hyperparameters "n_neighbors" (number of neighbors), "leaf_size" (leaf size passed to BallTree or KDTree), and "metric" (distance metric). For "n_neighbors", a rule of thumb for selecting it is that the optimal k is equal to the square root of the number of records in the training data. In our data, it means the k should be 895. Since more neighbors means much more time to compute, we used n_neighbors ={5, 100, 500, 895, 1000} in our search. The other parameters are set to: leaf_size={30, 100}. metric= {Euclidean, Manhattan, Minkowski(default)}.

After running grid search on the scaled dataset, we obtained a different set of parameters and achieved a much better AUC. The result is presented below.

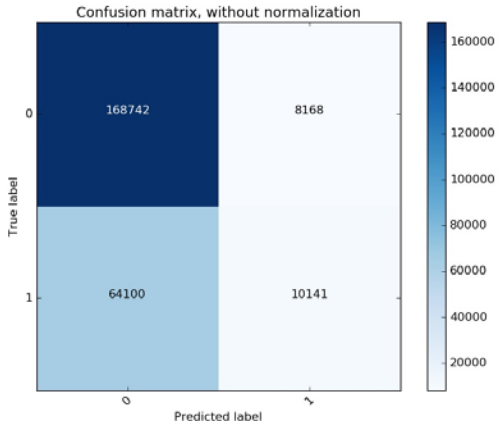


KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=895, p=2, weights='distance')

2.3 Evaluation Metrics

We calculated recall and precision from the confusion matrix below. The precision is 0.56 and the recall is 0.14, which is no better than other models we explored. With a base rate at 0.29, the lift is around 1.93. Because

the recall and AUC are not ideal compared to other methods and k-NN is expensive to run, we won't use this model for our business problem.



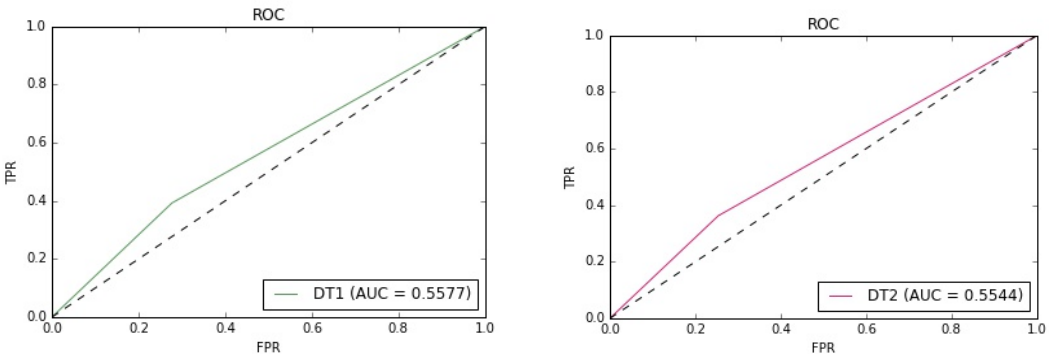
Confusion matrix of k-NN, without normalization

3. Decision Tree

Decision tree is a straightforward model that is easy to build and performs fast with large dataset. However, with default settings it tends to overfit, so it requires extra effort on selecting hyper-parameters. Small changes in data often result in very different solutions, so it is also quite unstable.

3.1 Baseline Model

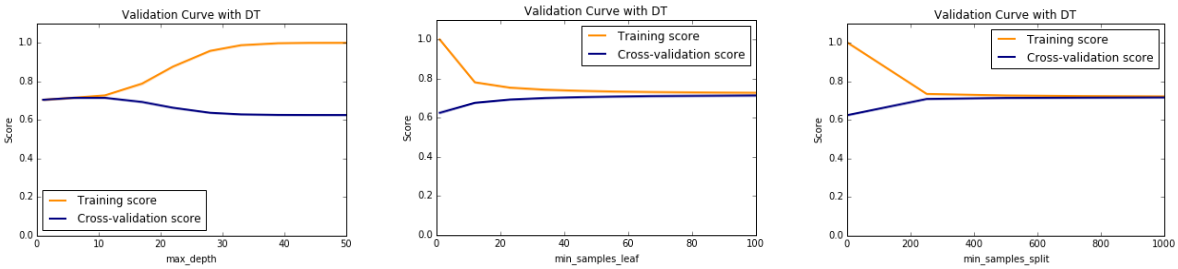
We began by building a baseline model using default settings, and performed on both scaled and unscaled data. Model performance is rather poor with default settings, which suggests that there may exist problems of overfitting. We also plot feature importance (see appendix) and the most important features are: price of the project, the fundraising duration and total number of students reached. The ROC curves created by baseline models are below.



DecisionTreeClassifier*(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')*

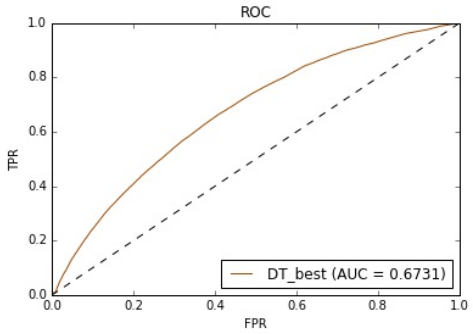
3.2 Grid search on hyper-parameter

To see if we can improve our baseline model, we then ran a grid search on hyper-parameters: min split size, min leaf size and max_depth, which controls model complexity and chances to overfit. To help choose a reasonable range for grid search, we first drew validation curves to look for informative trends as below. In order to save time, we use only 2 fold cross-validation considering our large sample size. Validation curve shows the gap between training score and cross-validation score, which suggests overfitting when the gap is large, and the negligible effect the parameter has on model performance when the gap closes.



It can be shown that min leaf size here does not have a significant influence on model performance, as it converges fast at rather small value. Overfitting also arises when max_depth is larger than 10, so we cannot grow the tree too deep. Finally, min split size converges at a larger value, so it is most likely to have a different value from default settings.

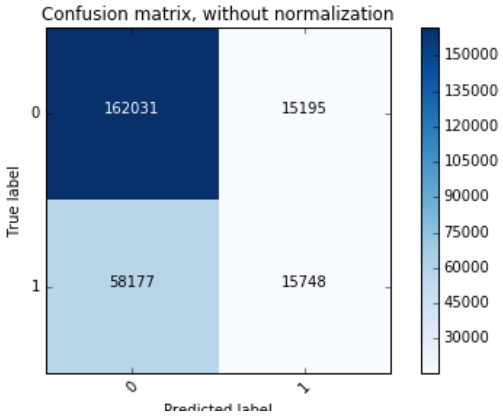
Based on these results, we now run a grid search of these 3 parameters. Max depth is set to a range from 1 to 20, min leaf size from 1 to 100, and min split size from 200 to 1000, each with 5 values. We set up 2-fold cross-validation and run grid search on the scaled data. The model with the best score now has parameters min_samples_leaf=80, min_samples_split= 600, max_depth=10. And AUC improves quite remarkably to 0.67.



DecisionTreeClassifier *(class_weight=None, criterion='gini', max_depth=10, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=80, min_samples_split=600, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')*

3.3 Evaluation Metrics

Looking at the confusion matrix below, we have precision = 0.51, while the base rate = 0.29, so we have lift that is approximately 1.75. Compared to logistic regression, recall here is 0.21, which is higher. We may say this model is a little better at identifying true labels as it reduces false negatives, but at the cost of a higher false positive rate.



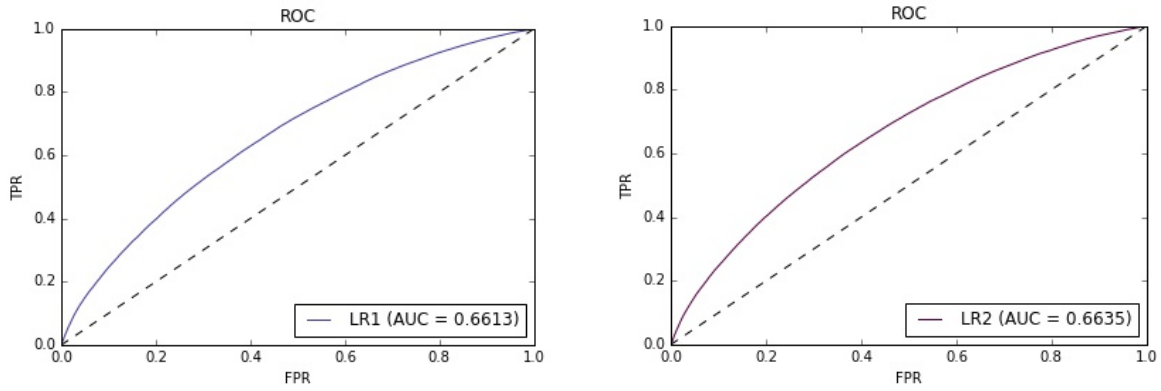
Confusion matrix of Decision Tree, without normalization

4. Logistic Regression

Logistic Regression is a simple and robust algorithm with strong statistical properties, and works well on both small and large dataset.

4.1 Baseline Model

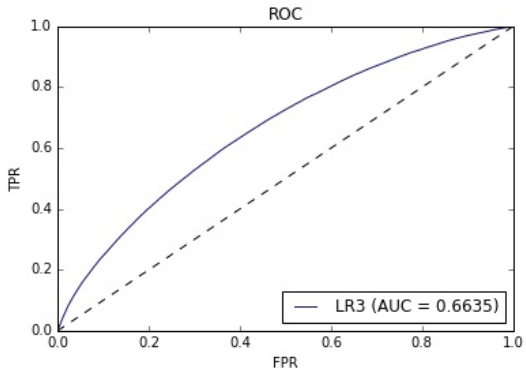
We first ran two logistic regressions with default settings, one on the original dataset, the other on the scaled dataset. There is not much variation on the model performance as we can see below.



LogisticRegression *(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1, penalty='l2', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)*

4.2 Grid search on hyper-parameter

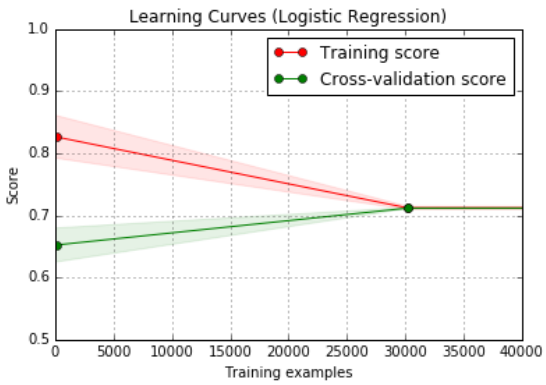
To see if we can improve our baseline model, we then ran a grid search on hyper-parameters 'C', which is the regularization strength, and 'penalty'- the regularization method. We set up 5-fold cross-validation and run the grid search on the scaled data. The model with the best score,as below, has parameters different from the default setting, with less regularization and 'L1' penalty, but ROC curve and AUC remain the same.

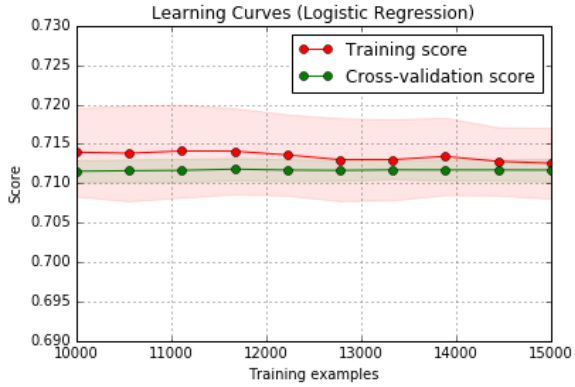


LogisticRegression *(C=100, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1, penalty='l1', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)*

4.3 Sample size and learning curve

Our cleaned data has a sample size of nearly 1 million, which took quite a long time to process each time. However, using default settings and 10-fold cross validation, the learning curves begin to converge at around 20,000, which suggests that, for complex calculations that takes a lot of time and memory, we could choose a smaller sample size.





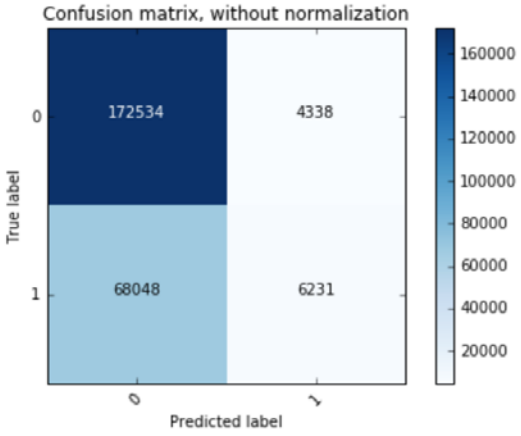
4.4 Model Interpretation

By observing the coefficients, approximately a $100 increase in total prices of the project will lead to a 5% decrease of odds of getting funded. Notably, compared to a long essay(more than 350 words) which may have

provided more details about the project , short essays do have approximately 20% lower odds of getting successfully funded.

4.5 Evaluation Metrics

Besides calculating AUC, we can have a closer look at how our model behaves by looking at the confusion matrix. Here we have precision = 0.59, while the base rate = 0.29, so we have lift at approximately 2. But recall here is only 0.08, which is fairly low. If we think false positives are more important(or costly) this is a good choice. But if we care more about false negatives, this model may not be good enough.



Confusion matrix of Logistic Regression, without normalization

5. Support Vector Machine (SVM)

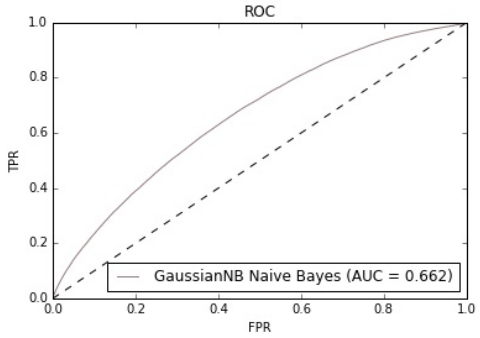
Although SVM is robust to noise, we did not employ this model because it is too expensive to run for our dataset. In our preliminary analysis, this model performed too slow and took relatively much longer time(compared to the other models) to run merely 5000 observations. The AUC score is approximately 0.644, which is not ideal either. Furthermore, the classifier in sklearn does not provide direct probability estimates. According to the sklearn website, they need to be calculated using an expensive five-fold cross-validation, which is really too complex for a large dataset like ours. In addition, the probability estimates may be inconsistent with the scores. Therefore we decided not to run this model.

6. Naive Bayes

Naive Bayes classifier performs well in classification and ranking problems. It is quite efficient in storage and runs very quick. It is easy to implement with no parameters to tune. However, we should note that the model assumes independence between features, so the probability estimates are usually larger for the correct class and smaller for the incorrect classes. For this very reason, Naive Bayes also suffers from high bias and low variances. Therefore we would implement this model for ranking problems while not actually using the true values of probability estimates.

6.1 Baseline Model

We first employed the default Gaussian Naive Bayes algorithm to the normalized data.



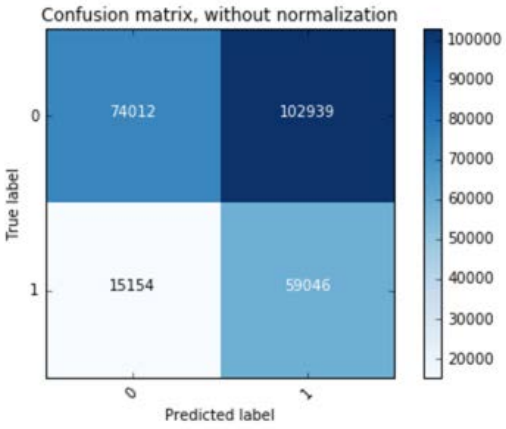
ROC Curve of Gaussian Naive Bayes model, GaussianNB(priors=None)

As one can see from the graph above, the AUC for Naive Bayes is acceptable, but not as good as some of the other models we discussed. Because there is no parameter to tune for Naive Bayes in scikit-learn, the above graph is the final result for this model. Donorschoose.org could adopt this algorithm if running time and storage are top concerns for the firm since Naive Bayes runs very fast. Otherwise, the other algorithms outperform Naive Bayes.

6.2 Evaluation Metrics

We calculated recall and precision from the confusion matrix below. The calculated recall is 0.80, which is incredibly high, but the precision is only 0.36. With the base rate at 0.29, the lift is 1.24. Even though we desire

a higher recall, but the low precision will cause a large amount of false alarms. So this model is not practical in real life.



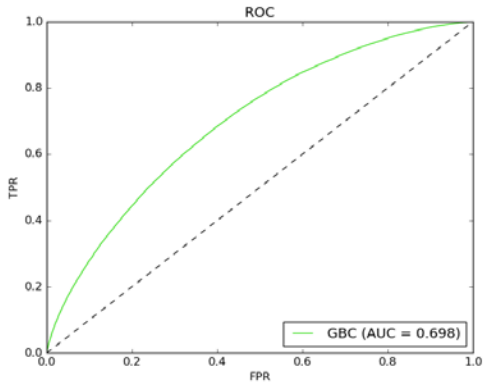
Confusion matrix of Naive Bayes, without normalization

7. Gradient Boosting

With Gradient Boosting Classifier, there are lots of flexibility with the choice of loss functions. Many parameters can interact and influence heavily on the approach. Overfitting can occur if values of parameters are not suitable, and it is computationally intensive.

7.1 Baseline Model

For this model, we will just do a preliminary study and make improvements in the future. We first run Gradient Boosting model with default settings on the original dataset. We can see the model performs well and outputs a relatively high AUC value.



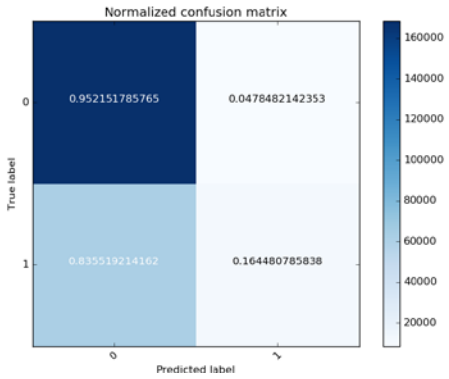
7.2 Evaluation metrics

According to the confusion matrix, we have precision=0.59, with base 0.29, the lift is approximately 2. The recall here is 0.16. Compared to other models, this model is not good at identifying true labels as it has more false negatives.

[[168568 8471]

[61922 12190]]

Confusion matrix, without normalization



Confusion matrix of Gradient Boosting, with normalization

Deployment

Judging from the above modeling process, the best model is the Random Forest model. Compared to the other models, it has the right balance between precision and recall, with recall relatively higher than the other models. We wanted to stress recall in the trade-off, meaning that we would rather have a model that doesn't misclassified a failure as success, since we are looking to improve the success rates of failing projects.

With every new project posted, we will input its features into our Random Forest classifier and calculate a probability estimation. Depending on the output, if the estimated failure rate is higher than a threshold, we will

first suggest the person to write a longer essay, or to decrease the amount of money posted. Then we will place this project at a reasonable rank as discussed before based on the probability.

To monitor the effectiveness of this method, we will split similar projects into two groups. For a certain period of time, we will post one of the groups in the rank that we were using before. Then after the projects in this group are closed, we will post the other group of projects to the website in the new order we proposed. After that, we compare the success rates for the two groups of project. If the second group's rate is higher, then we are safe to say that our new implementation is working and our profit will be increased in the long term. If not, our risks will be a slightly lower success rate than before and decreased revenue in the short term.

There are no major ethical problems in our problem. However, we do have concerns that the goals of this website may not be in accordance with the donors. Besides helping teachers and students in need, the website earns profit by splitting money with each project. The donors have the choice whether to donate only to the project or also to the website. So if the website has a strong incentive to maximize expected revenue, they will let in as many projects as possible, and help them get funded successfully by tempering results, in a way to attract more donations and donors. The website also may not attempt to verify the authenticity of information listed in each project. Any teacher could post a request for donation and mark their need level as 'highest poverty', and it's costly to judge whether these are exaggerating or totally fake. Besides, projects predicted to have low probability of getting funded maybe just because they do have questionable authenticity. Trying to get them funded will be a waste of resources and would not improve social welfare.

Future improvements

For the Random Forest classifier, it took a long time to do OOB evaluation when using the whole datasets so we did not finish in time. In the future, we plan to use the whole datasets to do OOB evaluation for model selection, which might give us a better result.

For the Gradient boosting classifier with default settings, the AUC is 0.698, which is relatively high compared to other models with default settings. According to the confusion matrix, the precision is 0.59, which is higher than other models, but the recall is only 0.16, which does not satisfy our goal. In the future, we will attempt to use grid search to tune the hyperparameters of Gradient boosting classifier to improve the recall value and better serve our purpose.

We also employed Adaptive Boosting classifier. With default settings, the AUC is 0.682, which is quite high compared to some of the other models. The precision is 0.56 and the recall is 0.177, which does not satisfy our goal. Because Adaptive Boosting runs efficiently and has a high AUC, it is a promising model. In the future, we will consider using grid search to find better hyperparameters and see whether we can improve the recall value.

Appendix

Dataset:

1. Donorschoose.org https://research.donorschoose.org/t/opedata-layout-and-docs/18

Additional Figures:

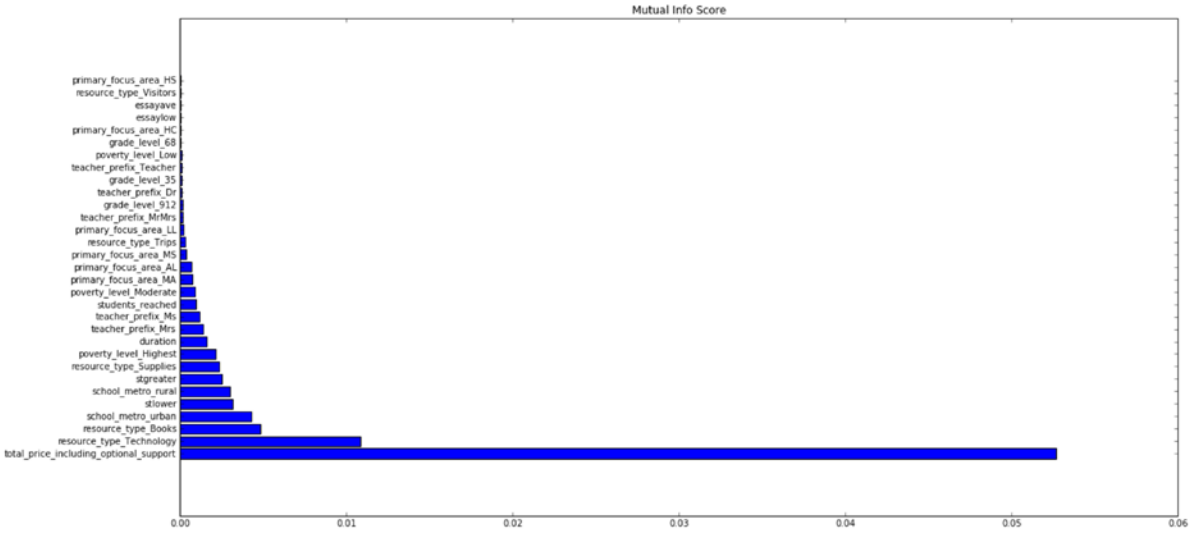


Figure 1. Mutual Information Score of all features for feature selection purposes

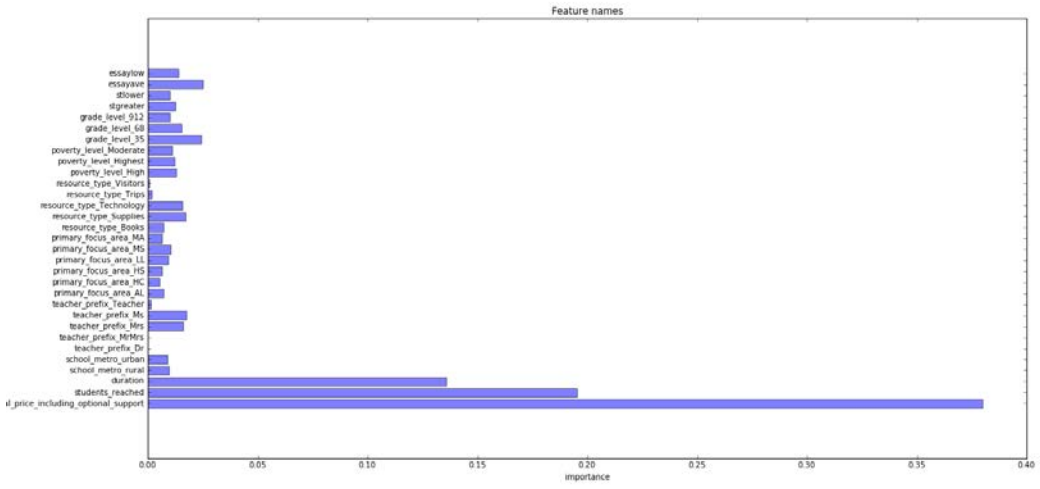


Figure 2. Feature importance from Decision Tree

Contributions:

1. Han Zhao: Logistic Regression model, Decision Tree classifier
2. Xue Yang: Random Forest classifier, Gradient boosting classifier
3. Li Lin Qin: Data cleaning, essay clustering, text feature extraction
4. Yiran Xu: k-NN model, Naive Bayes model, Adaboosting classifier

References:

1. Provost, Foster, and Tom Fawcett. *Data Science for Business*. O'Reilly Media, Inc. 2013. Print.
2. "Random Forest." *Wikipedia*. Wikimedia Foundation, n.d. Web. 07 Dec. 2016.
3. "Support Vector Machines." *1.4. Support Vector Machines — Scikit-learn 0.18.1 Documentation*. N.p., n.d. Web. 07 Dec. 2016.
4. Rose, Brandon. "Document Clustering" *Brandonrose.org*. Web. 07 Dec. 2016.
5. "U.S.: GDP per Capita by State in 2015 | Statista." *Statista*. Web. 08 Dec. 2016.
6. "Sklearn.neighbors.KNeighborsClassifier." *Sklearn.neighbors.KNeighborsClassifier — Scikit-learn 0.18.1 Documentation*. Web. 08 Dec. 2016.
7. Hassanat, Ahmad Basheer, Mohammad Ali Abbadi, and Ghada Awad Altarawneh. "Solving the Problem of the K Parameter in the KNN Classifier Using an Ensemble Learning Approach." *International Journal of Computer Science and Information Security* 12.8 (2014): n. pag. Web. 8 Dec. 2016.