INF 552: Project Report

Santander Customer Satisfaction Kaggle Competition

Pavan Vasan (lingambu@usc.edu), Suresh Kasipandy(kasipand@usc.edu), Murat Zhumagali (zhumagal@usc.edu), Xin Chenz (xinchenz@usc.edu)

Computer Science and Data Informatics Program, University of Southern California, Los Angeles, California

Abstract: Prediction is at the heart of almost every scientific discipline, and the study of generalization (that is, prediction) from data is the central topic of machine learning. Machine learning and statistical methods are used throughout the scientific world for their use in handling the "information overload" that characterizes our current digital age. Machine learning developed from the artificial intelligence community, mainly within the last 30 years, at the same time that statistics has made major advances due to the availability of modern computing. However, parts of these two fields aim at the same goal, that is, of prediction from data.

1. Introduction:

1.1 Tagline:

Which customers are happy customers?

1.2 Purpose:

From frontline support teams to C-suites, customer satisfaction is a key measure of success. Unhappy customers don't stick around. What's more, unhappy customers rarely voice their dissatisfaction before leaving. The purpose of this contest is to help Santander Bank in identifying dissatisfied customers early in their relationship. Doing so would allow Santander to take proactive steps to improve a customer's happiness before it's too late. In this competition, you'll work with hundreds of anonymized features to predict if a customer is satisfied or dissatisfied with their banking experience.

1.3 Details of Data:

We were provided with an anonymized dataset containing a large number of numeric variables. The "TARGET" column is the variable to predict. It equals 1 for unsatisfied customers and 0 for satisfied customers. The task was to predict the probability that each customer in the

test set is an unsatisfied customer. The data set was provided in the form of csv files along with a template as to how the format of the submission file is to be.

1.4 Strategy Used:

We worked on the problem using the XGBoost (Extreme Gradient Boosting) and Random Forest techniques and then used the programs to implement ensemble averaging by taking the values calculated by each method and calculating a weighted sum of those values for each point in the dataset.

1.5 Result:

The accuracy score achieved through the strategy was 0.820682 with the highest score being 0.829072 in the leaderboard. (private Scoreboard)

2. Details of the methods used for the project:

2.1 eXtreme Gradient Boosting (Form of Gradient Boosting) :

2.1.1 *Short note on eXtreme Gradient Boosting*

XGBoost is short for "Extreme Gradient Boosting", where the term "Gradient Boosting" is proposed in the paper Greedy Function Approximation: A Gradient Boosting Machine, by Friedman. XGBoost is based on this original model.

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Regression tree (also known as classification and regression tree):

- Decision rules same as in decision tree
- Contains one score in each leaf value

Regression Tree Ensemble Method:

- Invariant to scaling of inputs, so you do not need to do feature normalization. Learn higher order interaction between features. It is scalable, and are used in Industry.
- Now here comes the question, what is the model for random forests? It is exactly tree ensembles! So random forests and boosted trees are not different in terms of model, the difference is how we train them. This means if you write a predictive service of tree

ensembles, you only need to write one of them and they should directly work for both random forests and boosted trees.

2.1.2 *Model Specifications*

To understand the parameters behind Gradient Boosting using XGBoost library, one needs to have a basic understanding of the model behind. Suppose we have K trees, the model is:

$$\sum_{k=1}^{K} f_k$$

where fk each is the prediction from a decision tree. The model is a collection of decision trees. Having all the decision trees, we make prediction by:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$$

where is the feature vector for the i-th data point. Similarly, the prediction at the t-th step can be defined as:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i)$$

To train the model, we need to optimize a loss function. Here we tend to use Log Loss for binary classification:

$$- \; L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{i,j} \log(p_{i,j})$$

Meanwhile we use M log loss for multi-classification.

Regularization is another important part of the model. A good regularization term controls the complexity of the model which prevents overfitting.

We define the following:

$$\Omega = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where, T is the number of leaves, and w^2_j is the score on the j-th leaf.

Put loss function and regularization together, we have the objective of the model:

Obj = L + Ω

where loss function controls the predictive power, and regularization controls the simplicity. In XGBoost, we use gradient descent to optimize the objective.

Given an objective $Obj(y, \hat{y})$ to optimize, gradient descent is an iterative technique which calculate

$$\partial_{\hat{y}} Obj(y, \hat{y})$$

at each iteration. Then we improve yˆ along the direction of the gradient to minimize the objective.

Recall the definition of objective Obj = L + Ω. For a iterative algorithm we can re-define the objective function as

$$Obj^{(t)} = \sum_{i=1}^{N} L(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) = \sum_{i=1}^{N} L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \sum_{i=1}^{t} \Omega(f_i)$$

To optimize it by gradient descent, we need to calculate the gradient.

$$\partial_{\hat{y}_i^{(t)}} Obj^{(t)}$$

$$\partial^2_{\hat{y}_i^{(t)}} Obj^{(t)}$$

The performance can also be improved by considering both the first and the second order gradient.

- $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$
- $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

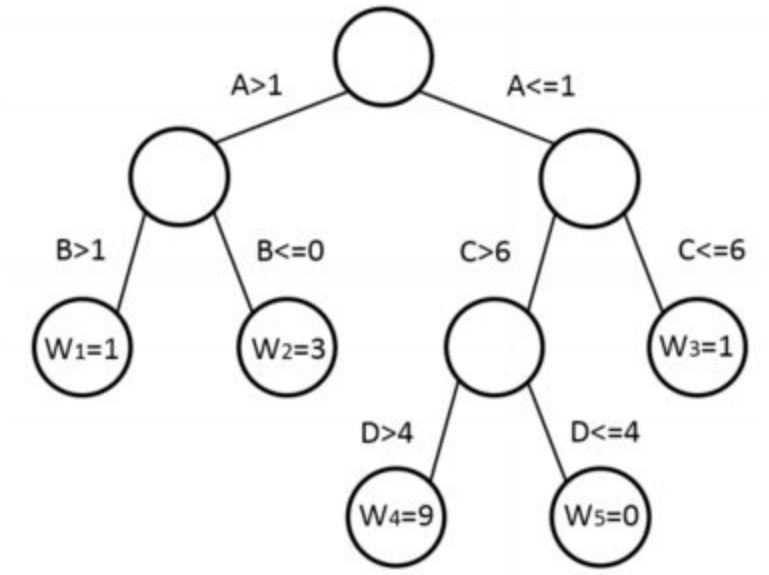
Since we don't have derivative for every objective function, we calculate the second order taylor approximation of it
Where,
Remove the constant terms, we get

$$Obj^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

This is the objective at the t-th step. Our goal is to find a ft to optimize it.

2.1.3 Tree Building Algorithm

The tree structures in XGBoost leads to the core problem:
how can we find a tree that improves the prediction along the gradient?
Every decision tree looks like this



Each data point flows to one of the leaves following the direction on each node.
The core concepts are:

- Internal Nodes
  - Each internal node split the flow of data points by one of the features.
  - The condition on the edge specifies what data can flow through.

- Leaves
- Data points reach to a leaf will be assigned a weight.
- The weight is the prediction.

Two key questions for building a decision tree are
1. How to find a good structure?
2. How to assign prediction score?
We want to solve these two problems with the idea of gradient descent.
Let us assume that we already have the solution to question 1.
We can mathematically define a tree as

$$f_t(x) = w_{q(x)}$$

where is a "directing" function which assign every data point to the q(x)-th leaf.
This definition describes the prediction process on a tree as

- Assign the data point x to a leaf by q
- Assign the corresponding score wq(x) on the q(x)-th leaf to the data point.

Define the index set

$$I_j = \{i|q(x_i) = j\}$$

This set contains the indices of data points that are assigned to the j -th leaf.
Then we rewrite the objective as

$$Obj^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2$$

$$= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T$$

Since all the data points on the same leaf share the same prediction, this form sums the prediction by leaves.
It is a quadratic problem of w_j, so it is easy to find the best w_j to optimize Obj.

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

The corresponding value Obj of is

$$Obj^{(t)} = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T$$

The leaf score

$$w_j = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda}$$

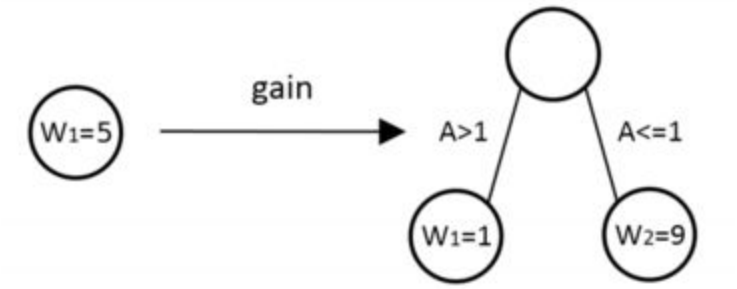
relates to

- The first and second order of the loss function g and h
- The regularization parameter λ

Now we come back to the first question: How to find a good structure?
We can further split it into two sub-questions:
1. How to choose the feature to split?
2. When to stop the split?

In each split, we want to greedily find the best splitting point that can optimize the objective.
For each feature
1. Sort the numbers
2. Scan the best splitting point.
3. Choose the best feature.

Now we give a definition to "the best split" by the objective.
Every time we do a split, we are changing a leaf into a internal node.



Let

- I be the set of indices of data points assigned to this node
- I_L and I_R be the sets of indices of data points assigned to two new leaves.

Recall the best value of objective on the j-th leaf is

$$Obj^{(t)} = -\frac{1}{2}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma$$

The gain of the split is

$$gain = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma$$

To build a tree, we find the best splitting point recursively until we reach to the maximum depth. Then we prune out the nodes with a negative gain in a bottom-up order.
XGBoost can handle missing values in the data.
For each node, we guide all the data points with a missing value

- to the left subnode, and calculate the maximum gain
- to the right subnode, and calculate the maximum gain
- Choose the direction with a larger gain

Finally every node has a "default direction" for missing values.
To sum up, the outline of the algorithm is

- Iterate for n round times
- Grow the tree to the maximum depth
  - Find the best splitting point
  - Assign weight to the two new leaves
- Prune the tree to delete nodes with negative gain

2.2 Random Forest:

2.2.1 Introduction to Random Forest:

Random forest is a notion of the general technique of random decision forests that are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes/classification) or mean prediction/regression of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

A random forest is an ensemble of decision trees which will output a prediction value, in this case survival. Each decision tree is constructed by using a random subset of the training data. After you have trained your forest, you can then pass each test row through it, in order to output a prediction.

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges as. to a limit as the number of trees in the forest becomes large. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. Using a random selection of features to split each node yields error rates that compare favorably to Adaboost , but are more robust with respect to noise. Internal estimates monitor error, strength, and correlation and these are used to show the response to increasing the number of features used in the splitting. Internal estimates are also used to measure variable importance. These ideas are also applicable to regression.

2.2.2 Preliminary Technique: Decision Tree Learning

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy). Leaf node (e.g., Play) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

2.2.3 Algorithm

- *Bootstrap aggregating/Tree Bagging:*

Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

Given a Training set of N examples and A class of learning models (e.g. decision trees, neural networks…). Bagging Algorithm trains multiple (k) models on different samples and average their predictions then predicts/tests by averaging the results of k models.

The goal of the algorithm is improving the accuracy of one model by using its multiple copies. Average of misclassification errors on different data splits gives a better estimate of the predictive ability of a learning method.

- *Training:*

  In each iteration t, t=1,…T;
  Randomly sample with replacement N samples from the training set.
  Train a chosen “base model” (decision tree) on the samples.
  Test:
  For each test example, Start all trained base models.
  Predict by combining results of all T trained models:
  Regression: averaging
  Classification: a majority vote

Fig 1: Decision Trees in Random Forest

2.2.4 Random subspace method - Bagging to Random Forest

Random subspace method is an ensemble classifier that consists of several classifiers each operating in a subspace of the original feature space, and outputs the class based on the

outputs of these individual classifiers. Random subspace method has been used for decision trees (random decision forests),linear classifiers, support vector machines, nearest neighbours and other types of classifiers. This method is also applicable to one-class classifiers.
The algorithm is an attractive choice for classification problems where the number of features is much larger than the number of training objects, such as fMRI data or gene expression data

The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated.

The Random subspace method may benefit from using random subspaces for both constructing and aggregating the classifiers. When the number of training objects is relatively small compared with the data dimensionality, by constructing classifiers in random subspaces one may solve the small sample size problem. The subspace dimensionality is smaller than in the original feature space, while the number of training objects remains the same. Therefore, the relative training sample size increases. When data have many redundant features, one may obtain better classifiers in random subspaces than in the original feature space. The combined decision of such classifiers may be superior to a single classifier constructed on the original training set in the complete feature space.

2.2.4 How Random Forest Works

Each tree is grown as follows:

- If the number of cases in the training set is N, sample N cases at random. but with replacement, from the original data. This sample will be the training set for growing the tree.
- If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
- Each tree is grown to the largest extent possible. There is no pruning.

The forest error rate depends on two things:

- The correlation between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate. Reducing m reduces both the correlation and the strength. Increasing it increases both.

When the training set for the current tree is drawn by sampling with replacement, about one-third of the cases are left out of the sample. This oob (out-of-bag) data is used to get a running unbiased estimate of the classification error as trees are added to the forest. It is also used to get estimates of variable importance.

After each tree is built, all of the data are run down the tree, and proximities are computed for each pair of cases. If two cases occupy the same terminal node, their proximity is increased by one. At the end of the run, the proximities are normalized by dividing by the number of trees. Proximities are used in replacing missing data, locating outliers, and producing illuminating low-dimensional views of the data.

3. Conclusion :

Based on the results shown in the program, we can conclude that Gradient Boosting using the XGBoost Library and Random Forests are effective techniques in order to predict a given value given a dataset and the classifications to which each of those data points have been grouped.

References

1. https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
2. https://www.kaggle.com/kushal1412/santander-customer-satisfaction/simple-random-forest-satisfaction
3. http://blog.yhat.com/posts/random-forests-in-python.html
4. https://www.kaggle.com/c/titanic/details/getting-started-with-random-forests
5. https://www.kaggle.com/c/digit-recognizer/forums/t/2299/getting-started-python-sample-code-random-forest
6. http://www.analyticbridge.com/profiles/blogs/random-forest-in-python
7. http://blog.datadive.net/random-forest-interpretation-with-scikit-learn/
8. http://101.datascience.community/tag/random-forest/
9. http://www.analyticsvidhya.com/blog/2015/09/random-forest-algorithm-multiple-challenges/
10. https://www.kaggle.com/tqchen/otto-group-product-classification-challenge/understanding-xgboost-model-on-otto-data
11. https://www.youtube.com/watch?v=loNcrMjYh64&app=desktop
12. http://ocw.mit.edu/courses/sloan-school-of-management/15-097-prediction-machine-learning-and-statistics-spring-2012/
13. http://www.slideshare.net/ShangxuanZhang/xgboost

Note about the team members contributions :

Every member in the team put in their best efforts and contributed to the project equally. Though members worked in groups of two on the important aspects of the project, everyone helped each other in solving the problems faced in order to progress further in completing the project.