Bank Marketing with Machine Learning

Zewei Chu

April 19, 2015
Contents

0.1 Abstract ........................................ 5
0.2 Introduction ................................... 6
0.3 Related Work ................................. 9
 0.3.1 Bank Marketing Campaigns with Machine Learning .................. 9
 0.3.2 Classic Machine Learning Algorithms ................................ 9
 0.3.3 Big Data Tools .................................. 10
0.4 Machine Learning .............................. 11
 0.4.1 Decision Tree ................................. 11
 0.4.2 Regression ...................................... 14
 0.4.3 Boosting ........................................ 17
0.5 An Overview of Bank Marketing ................. 21
 0.5.1 Expert Designed Model ........................... 21
 0.5.2 Data Driven Model ............................. 21
 0.5.3 Big Data ....................................... 22
0.6 Experiment Overview and Setup ............... 24
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6.1 Understanding the business</td>
<td>25</td>
</tr>
<tr>
<td>0.6.2 Data Selection</td>
<td>26</td>
</tr>
<tr>
<td>0.6.3 Build the feature table</td>
<td>29</td>
</tr>
<tr>
<td>0.6.4 Labeling samples</td>
<td>31</td>
</tr>
<tr>
<td>0.7 Gradient Boosted Decision Tree</td>
<td>33</td>
</tr>
<tr>
<td>0.7.1 Continuify Discrete Features</td>
<td>33</td>
</tr>
<tr>
<td>0.7.2 XGBoost</td>
<td>35</td>
</tr>
<tr>
<td>0.8 Logistic Regression</td>
<td>37</td>
</tr>
<tr>
<td>0.8.1 Practical Analysis of Logistic Regression</td>
<td>37</td>
</tr>
<tr>
<td>0.8.2 Discretize Continuous Features</td>
<td>38</td>
</tr>
<tr>
<td>0.8.3 Feature Combinations</td>
<td>39</td>
</tr>
<tr>
<td>0.9 Evaluations</td>
<td>40</td>
</tr>
<tr>
<td>0.9.1 biased training and testing data sets</td>
<td>40</td>
</tr>
<tr>
<td>0.9.2 ROC and AUC</td>
<td>41</td>
</tr>
<tr>
<td>0.10 Discussion</td>
<td>44</td>
</tr>
<tr>
<td>0.10.1 Decision Tree</td>
<td>44</td>
</tr>
<tr>
<td>0.10.2 Logistic Regression</td>
<td>45</td>
</tr>
<tr>
<td>0.11 Future Work</td>
<td>47</td>
</tr>
<tr>
<td>0.12 Conclusion</td>
<td>48</td>
</tr>
<tr>
<td>0.13 Acknowledgement</td>
<td>49</td>
</tr>
<tr>
<td><strong>Bibliography</strong></td>
<td>50</td>
</tr>
</tbody>
</table>
0.1 Abstract

Bank market campaigns today mostly rely on human expert's opinions on choosing potential customers. This method is time consuming and lacks accuracy. As banks have very structured and detailed client information and transaction records, it is desirable to build data driven decision making systems with guaranteed high successful rate at campaigns. Machine Learning models and techniques have huge potential to show their power in such problem settings. This project report shows how machine learning algorithms can be applied in such practical problem settings, especially with big data sets. The report discusses three case studies I conducted regarding machine learning models in bank marketing campaigns in details. The report also addresses some practical annoying problems existing in different training algorithms. Some simple but useful solutions are provided and discussed. This work can serve as a reference to future bank marketing campaign system modeling and other data driven decision making systems designs.
0.2 Introduction

Machine Learning techniques have been applied to various scenarios of real life applications. However, in some certain sense, machine learning, especially data mining with big data sets, is still a dark science for most people. Even some data scientists or engineers do not understand the real magic happened behind the model training and how different models work on specific problem settings. The purpose of my project is to explore the applications of machine learning algorithms in building data drive decision making models in bank marketing campaigns. This report can serve as a general guideline or reference on how to build similar bank campaign systems at future applications. More generally, the machine learning models and tricks applied as described in this report can be applied to other data drive decision making systems as well with proper adjustments.

Section 0.3 first reviews some related work on machine learning models’ applications on bank campaigns and discuss how my work differs from previous ones.

Understanding the theoretical backgrounds of machine learning algorithm is the cornerstone of building good models. Section 0.4 reviews three classic and powerful machine learning algorithms, decision tree, regression and boosting. The review is very concise and mainly serves to help readers get intuitions about the design principle of such algorithms. Such basic understanding of the theoretical aspects of machine learning models help readers decide their corresponding application scenarios and solve specific problems arising in different cases.

Section 0.5 then generally discusses the current situation of bank marketing campaigns. Most bank campaigns still rely on human experts opinions to decide which potential customer to call (or to send message to). This way of marketing is time consuming and inaccurate. The success of machine learning in other areas attracts the banking industry to invest on decision making models with machine learning models, but the lack of talents and experience makes such projects difficult to proceed. This report is a good reference
to future model builders in the banking industry.

Section 0.6 introduces the problem setting of my three case studies and how I gather necessary data for model training. The experiment setup work and data preparation is mostly trivial but some smart work is preferred to get the work done quickly. One thing worth noticing is that my first case study is based on a data set of Terabytes and the complexity of this study is much higher than most previous studies.

Section 0.7 and section 0.8 gives details on how to build decision making models with Gradient Boosting Decision Tree and Logistic Regression. Some useful tricks such as dimension reduction, feature engineering, and dimension increasing are introduced. They have proved to be very helpful in my case studies.

The major challenge in my three case studies are:

- One case study is of extremely large scale.
- Directly applying classic algorithms will not yield models with great performance. Some proper handling of data format and feature engineering is of huge help.

Section 0.9 discusses some practical issues about evaluation. In real projects, evaluation can be a problem due to insufficiency of test data, and bias of test data, or the difference between training data distribution and test environment. I will give some advice on how these problem can be tackled. The detailed information regarding the performance of my third case study is introduced in this section.

Section 0.10 discusses some more issues worth noticing in machine learning and model building. Finally I give some insights on future work and draw the conclusions.

In summary, the contributions of this project are:

- Two-day on-line experiments show the model of my case study will generate 300,000 CNY per day profit for the bank.
• Review and apply machine learning algorithms on building data driven decision making systems in bank marketing campaign systems.

• Introduce practical problems in real big data model building and give solutions.

• Work on extremely large scale data sets and show some useful techniques to enhance the performance of classic models.

• This work can serve as a reference for other data drive decision making systems in bank (or other) marketing campaign.
0.3 Related Work

This section introduces some previous work related to my case studies and project, including machine learning algorithms applied in bank marketing campaigns, theoretical research on machine learning algorithms, and tools built for big data analysis.

0.3.1 Bank Marketing Campaigns with Machine Learning

S. Moro [S. Moro and Cortez, 2011] works on a case study of bank marketing campaigns with machine learning decision making systems. My third case study in this thesis also uses his data sets. His case studies shows how Naive Bayesian, Decision Tree and SVM works on bank marketing campaigns. All three algorithms performs quite well on his data sets. His case study is relatively of small scale with only 45211 instances. My first case study in this thesis uses a bank campaign data set of more than 50 million instances. Hence my case study will show results of more interests and applicability.

0.3.2 Classic Machine Learning Algorithms

Another related background work is all classic algorithms used in my three case studies, including Decision Tree Quinlan [1985], Salzberg [1993], Regression models Freedman [2005], and boosting Freund and Schapire [1997, 1999]. Such three classic machine learning models have a long history and have proven to be very useful model in prior research and experiments. In big data scenarios, my three case studies show that they are still very powerful at training good models. Further, big data sets usually represents the whole population even better and do not suffer severe overfitting issues.
0.3.3 Big Data Tools

XGBoost designed by Tianqi Chen [Chen 2014] and Vowpal Wabbit [Alekh Agarwal 2013] designed by John Langford are two primary tools I used in this project. These two tools are optimized for learning boosted decision tree and logistic regression models on extremely large scale data sets. They are very fast at training big data sets and also support parallelization.

In my three case studies I never used any distributed systems such as Hadoop or Spark. Such big scale of distributed systems are not very realistic for most current bank’s technology departments. However, a server with more than 10 cores and 100G memory is reasonable to be equipped with.
0.4 Machine Learning

Machine Learning algorithms are our best weapons to understand big data problems. Before diving into big data case studies, I first review three important classes of machine learning algorithms here.

- **Decision Tree.** Classic algorithm to split data set into categories as much as possible. The model is easy and powerful, but suffers from the problem of overfitting.
- **Regression.** Very good model relates the label to the weighted sum of features.
- **Boosting.** This is a wonderful algorithm to transform a bunch of weak learners into strong learners.

The purpose of this section is not to go through all details of every algorithm, instead I want to omit theoretical analyses as much as possible, but focus on the intuitive ideas how each algorithm works on different scenarios. Such analyses, I believe, is critical to the understanding of real problems and picking the best algorithm with appropriate input data formats.

### 0.4.1 Decision Tree

This subsection briefly review the concepts of decision tree, the ID3 algorithm of training decision trees, and also discuss some existing problem in decision trees and the corresponding solutions. Naively speaking, decision tree is just a set of binary rules to repeatedly split data sets into categories, and gives final prediction at each leaf nodes.

Decision tree is a very famous but also simple algorithm. For many data scientists, decision tree is the first machine learning algorithm they learn. The intuitive idea of decision tree learning is to build a disjunction of conjunction of constraints so that each conjunction of constraints will give one prediction. The previous sentence sounds complicated,
but the beauty of a decision tree is that it can be represented by a simple tree structure, in a sense that each node represents an IF-ELSE logic. This feature also makes decision tree very intuitive to be explained in human language and easy to understand.

To train a decision tree based on a training data set, an important idea is to recursively pick a feature and split the data set by the attribute of this specific feature. As soon as a node is created, the data set is split into two parts and we can further create nodes with the same method. This is the general idea of ID3 algorithm.

Before talking about more details of ID3 algorithm, let us first review the concept of Entropy and Information Gain. The entropy is defined as

\[
Entropy(S) = -p^+ \log_2 p^+ - p^- \log_2 p^- .
\]

Here \(p^+\) and \(p^-\) represents a positive test and negative test, i.e., yes or no in the specific feature. Notice that here I simplify all features by yes or no. Some authors may allow more than two possibilities in one feature, but a multi-attributed feature can be easily transferred into multiple yes-no features. The information gain is then defined as

\[
IG(A) = Entropy(S) - Entropy(S, A^+) - Entropy(S, A^-)
\]

The idea of information gain is that it measures how much entropy can be decreased if we are given the answer of a specific feature.

The information gain introduced above is the target with which the ID3 algorithm wants to optimize. At each node creating step, ID3 always picks the feature that will create maximal information gain. The ID3 algorithm is described in [1].

If we think carefully about the ID3 decision tree algorithm, it is not difficult to observe that it has some inherent pitfalls. One big notorious problem with decision tree model is overfitting. Overfitting is the phenomenon when a model performs better and better in the training data set, its performance at test data set or other more general data sets...
become worse. This phenomenon can be caused by multiple reasons, but some common explanations are noise and insufficiency of training data. Noise is some problematic training data. These data may be misclassified or recorded with wrong features. Thus if our model is based on fulfilling the requirement given by such noise information, the model will tend to be flawed. Insufficiency of training data can also cause the problem of overfitting, because small amounts of training data may not be good enough to represent the general cases. Hence even our model is perfect at predicting results on the training data set, it may not apply very well in more general cases. It is understandable that if we keeps making the decision tree with more and more nodes, it will ultimately get 100% correct at the training data, but the model we get will also be extremely complicated. Science always prefer simple theories instead of complicated ones, because we have reasons to believe that such complicated decision trees only applies in specific scenarios, in this case, the training set, but does not reveal the
true distribution of the data.

To attack the problem of overfitting, people have done a lot of research and proposed several solutions. One method is called error-reduced pruning. The idea is to split a validation set outside of the training set, when a decision tree is built, we iteratively test on each node to see if pruning the node will reduce the validation error. If this is the case, the node will be pruned. The idea behind this process is that if the training set contains some bias, it is not likely that this bias also appears in the validation set. Another method used by C4.5 algorithm is to first convert the learned tree into a set of rules, and try to prune the rule by removing any preconditions that result in its estimated accuracy.

Luckily, overfitting is not very likely to happen in the case studies discussed in the report, as the training data sets are of extremely large scale, and this prevents overfitting. Big data sets often represents the whole population very well without much bias and even some noise will not affect the quality of the data set too much. Also, in my case study, I used the gradient boosting decision tree algorithm, which boosted decision tree algorithm with a bunch of trees, then further increase the accuracy of the algorithm.

In summary, the idea of decision tree is to repeatedly split data set into categories and assign appropriate labels to the final step of splitting, i.e., the leaf node. Bearing this in mind, it is not difficult to think about that decision tree is capable of picturing any weird combination of data samples with enough deep trees. Thus it often suffers the problem of overfitting because of noisy data samples, insufficiency of training data, or building too complicated tree models.

0.4.2 Regression

This subsection talks about regression models, including linear regression and logistic regression.

The basic idea of regression, whether it is linear regression or logistic regression, is that it models the relation between
features and labels as a weighted (log) summation. The model believes that each feature will more or less make some contribution to the final classification, and our task is to determine how much each factor contributes.

The basic model of linear regression is

\[ Y = \beta_0 + \beta_1 X + \epsilon, \]

where \( X \) represents features and \( Y \) represents the corresponding label, \( \beta_0 \) and \( \beta_1 \) are two constants, namely, the intercept and the slope, of the model, and \( \epsilon \) is the error term representing the difference between the real label and estimated label. Our task then is to determine the value of \( \beta_0 \) and \( \beta_1 \). If we are able to calculate estimates \( \hat{\beta}_0 \) and \( \hat{\beta}_1 \), then we can predict the label by

\[ \hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x. \]

If we define \( e_i = y_i - \hat{y}_i \), and the residual sum of squares (RSS) as

\[ RSS = \sum_{i=1}^{n} e_i^2, \]

or equivalently,

\[ RSS = \sum_{i=1}^{n} (y_1 - \hat{\beta}_0 - \hat{\beta}_1 x_1)^2, \]

it is very intuitive and reasonable that we can minimize RSS to gain a good linear regression model. Simply differentiating this formula will give us the optimal parameters,

\[ \hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \]

\[ \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}, \]

where \( \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \) and \( \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \) are the sample means.

Logistic Regression share the similar thoughts as Linear Regressions. However, instead of modeling labels as a linear function of features, it models the logit odds of labels as a linear function of features. The model thus becomes

\[ \log \frac{p(x)}{1 - p(x)} = \beta_0 + x \beta. \]
Here \( p \) stands for the probability that feature \( x \) gives a positive label.

What is the advantage of modeling the logit odds as linear function of features rather than the just the odds? Intuitively speaking, linear functions are unbounded, and the same amount of change of a feature value always results the same amount of change in predictions. This fact does not fulfill our anticipation of the label. Ideally, changing \( p \) by the same amount should require larger change in \( x \) when \( p \) is big enough than when \( p \) is close to \( 1/2 \). Also, the logit function restricted the value of \( p(x) \) to be within \( 0 \) and \( 1 \), which is consistent to our knowledge that probability should be a value between \( 0 \) and \( 1 \).

Solving the logistic regression formula gives

\[
p(x) = \frac{e^{\beta_0 + x \beta}}{1 + e^{\beta_0 + x \beta}} = \frac{1}{1 + e^{-(\beta_0 + x \beta)}}.
\]

If we think about the meaning of \( p(x) \), it actually represents the probability of the features give a positive label. We may apply similar tricks to gain the optimal solution of parameters \( \beta_0 \) and \( \beta_1 \) like linear regression. In this case, we define the likelihood function of a logistic regression model by

\[
L(\beta_0, \beta) = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}.
\]

The log-likelihood will turn products into sums:

\[
\ell(\beta_0, \beta) = \sum_{i=1}^{n} (y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i)))
= \sum_{i=1}^{n} \log(1 - p(x_i)) + \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{1 - p(x_i)}
= \sum_{i=1}^{n} \log(1 - p(x_i)) + \sum_{i=1}^{n} y_i (\beta_0 + x_i \beta)
= \sum_{i=1}^{n} -\log(1 + e^{\beta_0 + x_i \beta}) + \sum_{i=1}^{n} y_i (\beta_0 + x_i \beta)
\] (1)

If we apply similar trick as in linear regression, we can differentiate the log-likelihood with one component of \( \beta \), say
\[ \frac{\partial \ell}{\partial \beta_j} = -\sum_{i=1}^{n} \frac{1}{1 + e^{\beta_0 + x_i \beta_j}} \frac{e^{\beta_0 + x_i \beta_j} x_{ij}}{1 + e^{\beta_0 + x_i \beta_j}} + \sum_{i=1}^{n} y_i x_{ij} \]

We cannot simply let it equal to zero and solve it directly as it is a transcendental function, however, there are plenty of numerical methods, such as Newton’s method, to approximated the optimal solutions.

I believe this I have given enough fundamental and theoretical information for linear and logistic regressions. In my three case studies introduced later, I will apply logistic regression algorithm to build models for all three case studies. The reason has been explained in previous text. In summary, the general idea of logistic regression is to use linear function to approximate the logit odds of the probability that such feature gives positive label. The training algorithm will assign appropriate weights to each feature. The larger the absolute value of a feature, the more it contributes to the labeling decision. Positive weight means this feature indicates positive label and vice versa.

### 0.4.3 Boosting

The idea of boosting comes from the questions of whether a "weak" learner which is just slightly better than random guessing can be "boosted" into an arbitrary accurate strong learner. Y. Freund and R. Schapire give an algorithm AdaBoost, which stands for "Adaptive Boosting" [Freund and Schapire 1999]. This section briefly review the AdaBoosting algorithm.

The idea of AdaBoost is very simple. Our target is to train a bunch of "weak" learners \( h_t \) and the final "strong" learner is a weighted combination of them with appropriate weights.
$H = \alpha_1 h_1$. It remains to consider how to train these bunch of "weak" learners and how to decide the weight of each learner. The intuitive ideas given by AdaBoost are:

- Higher weighted shall be assigned to learners with lower error rates.
- Samples in training data set have weights. Higher weights shall be assigned to previously poorly predicted samples so that the later learners can concentrate on difficulty samples.

The algorithm of AdaBoost is shown in Algorithm 2.

**Algorithm 2 AdaBoost algorithm**

1: **procedure** ADABoost\((x_1, y_1), \ldots, (x_m, y_m), T)\)
2: $D_1(i) \leftarrow 1/m$
3: **for** $t = 1, \ldots, T$ **do**
4: Train weak learner $h_t$ with distribution $D_t$
5: $\epsilon_t \leftarrow Pr_{D_t}[h_t(x_i) \neq y_i]$
6: $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$
7: $D_{t+1}(i) \leftarrow \frac{D_t(i) \exp^{-\alpha_t y_i h_t(x_i)}}{Z_t}, Z_t$ being a normalization factor
8: **end for**
9: $H(x) \leftarrow \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x))$
10: **end procedure**

Let us analyze the AdaBoost algorithm step by step.

We are given a set of sample data with features and corresponding labels \([-1, +1]\), and a integer $T$ representing the number of weak learners we want to train. Each sample will initially be assigned to the same weight $1/m$, where $m$ is the number of samples. AdaBoost will then repeatedly train this weighted data set to get a series of weak learners.

Every time after a weak learner is trained, an error rate $\epsilon_t = Pr_{D_t}[h_t(x_i) \neq y_i]$ adjusted with the sample weights is also estimated. This error rate $\epsilon_t$ provides very important information about the goodness of the weak learner. The smaller the error rate $\epsilon_t$, the larger weight we should assign to the weak learner.
Hence the next step is to calculate the weight \( \alpha_t = \frac{1-\epsilon_t}{\epsilon_t} \) as the weight assigned to the latest weak learner acquired. Notice that \( \alpha_t > 0 \) if \( \epsilon < \frac{1}{2} \), which means the weak learner is better than random guessing, and \( \alpha_t < 0 \) if \( \epsilon > \frac{1}{2} \).

Finally the sample data set distribution will be adjusted according to the performance of previous weak learner by 
\[
D_{t+1}(i) = \frac{D_t(i) \exp^{-\alpha_t h_t(x_i)}}{Z_t}.
\]
Naively speaking, the weight of correctly predicted samples are decreased while the weights of misclassified samples are increased. Thus at the next round, the new weak learner will put more attention on these misclassified samples.

The final output \( H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x)) \) gives prediction based on the weighted votes of a series weak learners from \( h_1 \) to \( h_T \).

Much theoretical and experimental studies have proven the power of AdaBoost as a "boosting" algorithm turning weak learners into a strong learner. Y. Freund and R. Schapire [1997] proved that the training error of the final hypothesis is bounded by 
\[
\prod_{t=1}^{T} 2\sqrt{\epsilon_t (1-\epsilon_t)} \leq \exp^{-2\sum (\frac{1}{2} - \epsilon_t)^2}
\]

Y. Freund and R. Schapire [Freund and Schapire [1997]] also proved the generalization error of the final hypothesis is bounded by
\[
\Pr[H(x) \neq y] + O(\sqrt{\frac{Td}{m}})
\]
. Here \( d \) stands for the VC-dimension of the weak hypothesis space and \( m \) is the sample size. Observing this upper bound, we may expect the generalization error increases with the addition of rounds of training after crossing a number limit. However, most experiments show that this never happens and the generalization error keeps dropping after adding training rounds.

Alternatively, Schapire analyzed the margins of training
samples. The margin of a sample \((x, y)\) is defined as
\[
y \frac{\sum_t \alpha_t h_t(x)}{\sum_t \alpha_t}
\]
. The sign of the margin represents the prediction of the final hypothesis while the absolute value shows the confidence of the prediction. Schapire proves that the generalization error is also bounded by
\[
Pr[\text{margin}(x, y) < \theta] + O\left(\sqrt{\frac{d}{m\theta^2}}\right)
\]
. This upper bounded reveals a very important relation, in the sense that AdaBoost is really good at increasing prediction confidence for each sample as the increase of number of rounds, the first term will get smaller and smaller after rounds of training, while the second term is inversely proportional to \(\theta\), the confidence threshold. Hence we can expect when \(T\) gets really large, the first term gets very small for large \(\theta\) and the second term will also be small for large \(\theta\).

In summary, boosting (AdaBoost specifically) ensembles a bunch of relatively weak learners and largely enhance the prediction accuracy of the new learner. In my study, boosting is combined with decision tree algorithm to build very strong learner in experiments.
0.5 An Overview of Bank Marketing

Bank marketing is a prosperous area in a sense that banks usually have strong ability to generate profits. Banks sell (financial) products to their customers through various campaigns, however, most campaigns turn out to have very low successful rates. Hence if machine learning techniques can be applied properly to increase the successful rate of a campaign, we will be able to gain tremendous profit.

0.5.1 Expert Designed Model

Current bank campaigns are usually based on expert designed models. A normal bank campaign is usually led and conducted by a group of business experts who has very good understanding of the business nature. Most of the time they can give great advices on what kind of customers to choose and call (in a telemarketing). However, this model has obvious shortages, for instance, the quality of each campaign is largely determined by the ability of the experts. It is more or less like a horse gambling game, although experienced gamblers usually perform better than novices, the outcome is never guaranteed. Also, when the data amount becomes huge, such as conducting daily campaigns on credit card transactions, it is impossible for an expert to judge which transaction to choose as a campaign candidate, but an automatic mechanism has to be designed and built.

0.5.2 Data Driven Model

The data driven model is a good substitute for the expert designed model, in a sense that it can largely utilize the information we have in our databases and the decision of picking candidates can be done by machines. Various machine learning algorithms will help us build great models surpassing human beings. However, practical difficulties exist in the design and implementation of such data driven systems as discussed in the next subsection 0.5.3.
0.5.3 Big Data

"Big Data" is a very hot topic now, the essence of big data is to mine some "golden" information from big amount of data and serve various purposes. Although tremendous theoretical research has been done on machine learning algorithms at data mining, very few research has been done to reveal how these techniques can be applied to real scenarios. Some companies like Google, Facebook, Microsoft invest a lot at big data researches to serve business uses, and some succeeded and generated profits from it.

These successful stories at utilizing big data provide a very promising future for other industries, including banks. Hence many banks today also invest heavily on big data research, hoping that they can make money from it.

In my opinion, bank data are extremely suitable for data mining, for the following reasons:

- Banks maintain very structured and accurate customer and transaction data, which is very convenient for data analysis.

- The research of bank data can be transferred into real money very easily, for instance, research customer behavior can help banks do market campaign with higher accuracy, and higher successful rates in market campaigns often means huge amount of real money.

However, according to what I have seen from the banking industry, most banks (in China and Hong Kong) encountered huge difficulties on big data research. With tons of golds in the databases, but banks cannot mine useful information from it. The following reasons lead to current difficulties:

- Short of investment: banks are not technological companies, although they assert that they have made huge investment on technology department, the industry itself impedes its success at technology sides. Not
many experts at machine learning and data mining are choosing a career in a bank. Thus this problem is actually caused by the short of talented people.

- The technology used in banks are “old” technology compared technological industry. For instance, their data is kept in structured databases, and most applications also rely on structured databases, but we cannot expect doing data mining with these databases efficiently. Also, many “data analysts” in banks are experts of SAS, SPSS, excel VBA, python, etc. These languages are not good choices when we are dealing with terabytes of petabytes of data.

- Another difficulty of data mining comes from the lack of communication among departments in banks. As I will discuss in later parts of the report, data mining is not only a science, but also an art. Training beautiful models for bank campaigns does not only require good understanding in machine learning models, the ability to work on huge data sets, but also good senses of business natures. Team work and good leader are essential to the success of such projects.

Hence one purpose of this project is to show some pilot work to address these difficulties, so that other researchers can take these case studies as a model method to build models from big data of banks.
0.6 Experiment Overview and Setup

This section gives an overview of my three case studies on bank marketing campaigns. Two case studies are extremely large scale and uses real data from two banks in China, another study is relatively smaller in scale and based on the data set of S. Moro [S. Moro and Cortez [2011].

The three case studies are:

- Credit card transaction installment recommendation model. Every time when a transaction is made by the client, the bank wants to build a decision making system to automatically send campaign SMS messages to the client asking whether they want to pay the transaction back in installments by months. If the client replies positively, the bank will make money by earning interests from the clients. The restriction is that they are only allowed to send a certain amount of messages each day, so the target is to maximize the successful rate at such campaigns.

- Gold related wealth management product sales campaign. The bank has some gold related wealth management product for sale. They want to sell these products to their potential clients through telemarketing, the problem being who to call as they have limited number of telephone operators and the higher the successful rate, the more money they can make.

- Bank Marketing Campaign. This is a data set from S. Moro [S. Moro and Cortez [2011].

The focus of the following sections about the case studies will be focused on the first case study, i.e. the credit card transaction installment recommendation model. These three case studies can serve as a guideline for similar campaign projects implemented in the banking industry.
0.6.1 Understanding the business

As mentioned previously, training a model is not only a science, but an art as well. There is no standard rules for good art, but some common methodology can be applied to achieve good performances. The first step, but also the key to train great model, is to fully understand the underlying business we are dealing with.

Many data scientists actually failed to pay enough attention to this essential preparation step of data mining. Most data scientists have good understanding about the theoretical backgrounds of machine learning algorithms, however, not many of them understand the scenarios where learning algorithms are most applicable, and the advantages or disadvantages of specific algorithms. Only after we fully understand the problem we are tackling can we pick the best algorithms to attack them.

Another reason why we need to understand business nature is that our machine learning algorithms are not smart enough to deal with all practical situations. The problem with current theoretical research is that they focus too much on solving specific problems of settings, but in reality, data scientists are facing much more complicated situations than these basic settings people have studied intensively and extensively. What is more important, the problem settings are not fixed in real problems. The target of most theoretical study is to enhance the machine learning algorithms with many restrictions, but we also need to understand that in real cases, we can try to loose the restrictions as much as we can but not focus on solving a very difficult problem with restricted information.

To better understand the business model, what we do before training models is that we consulted as many business experts at the area we are working on as possible. For the credit card transaction installment recommendation model, we interviewed these experts who got involved in multiple campaigns in the history. The experts of business nature can often provide very important information for the model building. Their experience and past analysis will give in-
formation about clients with specific features tend to be potential clients with higher probabilities than those without them. Notice that such information can be taken as part of the solutions, and one task before we start training the model is to guarantee that these known solutions should be added to the training features. Otherwise the model we trained may not even beat the expert designed model as we obviously ignored important features.

Sometimes although all features seem to be included in the training data set, the learning algorithms still failed to detect the important information. In such cases, we have to think about what makes the learning algorithm ignore such obvious essential features. The reason varies in different cases and it tests the data scientist’s ability to understand the business nature and how the learning algorithms study these rules. Sometimes the algorithm fails to detect it because there are more important features than them, for example, if the decision tree does not go deep enough, they may not detect some features, it may solve the problem to simply add some levels of decision rules in such cases. Sometimes the feature is a combination of features, while algorithms such as linear regression can only calculate the weight of single features. In such case, we may consider merging these features to make sure this important feature is included in the data sets.

Doing such analysis will even help us detect features we did not expect beforehand and is a direction to go in solving real data mining problems, but studying theoretical foundations will not teach us how to achieve it. It requires both good understanding of the algorithms and also the business natures.

0.6.2 Data Selection

In theoretical study, the training and test data sets are usually provided as given information. However, in solving real problems, they are part of the jobs of data scientists. It involves two separate meanings:
Collect as much features as possible to build the model.

Preprocess the data format before training, as mentioned previously, the input data format will affect the quality of the final model.

Feature collection is a mainly a job of communication. In building the credit card transaction installment recommendation system, we want to collect as much transaction related information, personal information, and personal wealth information as possible. We could even collect personal information from sources such as mobile applications. The more information we can get for each transaction, the better model we can expect to train from.

Ideally we wish that the whole technology department can provide their databases and give us detailed explanations on each table and columns, but because of various reasons of collaboration and security concerns, we are only allowed to build our model based on very limited databases.

For the credit card transaction installment recommendation system, we finally collected 5 tables for model training. I list these tables as followed:

- **Client Information Table.** This table contains all information related to a specific client, such as client number, birthday, birth city and country, gender, current city and country, education level, self-declared annual salary, personal credit limit, etc.

- **Account Information Table.** This table contains all account level information, such as account number, account type(normal, VIP, other special kinds), account holder’s client number, account level credit limit, current credit limit, account setup date, etc.

- **Card Information Table.** This table contains all information related to a specific credit card, such as card number, account number to which this card belongs, card type, card credit limit, current credit limit, amount due since last payment date, cycle date at each month, etc.
• Transaction Information Table. This table keeps records of all information related to each transaction. Such as the card number with which the transaction is made, the amount and currency type of the transaction, the location, time, date of the transaction, the merchant number, merchant type, the product or service nature.

• Campaign Information Table. The bank’s old expert made decision system has already conducted instant responsive campaigns for over two years. This table basically contains the transaction numbers in each campaign and the result of the campaigns.

Obviously we can do better if we could collect more information such as the saving account balance of each client, the other financial assets each client holds, all historical transaction made with their debit cards, client’s stock, futures, options trading records, etc. While these information is usually kept by other separate technology departments owned other department of the bank, it is really hard to coordinate the cooperation among these departments. However, such four tables have been great enough for us to build an excellent model surpassing all talented experts in the bank.

Notice that the data amount of this case study is huge. Approximated 1,000,000 transactions happened each day, and we are provided with the past 6 months transaction records and campaign results. The old expert designed decision making system normally picks about 300,000 transactions for campaigns through text message, asking whether they want to pay back the transaction by monthly installments. This number limit is set by some existing regulations. Historical data shows that the successful rate of each campaign message is approximately 1%.

In the case study of gold related wealth management product sales campaign, we uses three tables as followed:

• Client Information Table. Similar as previous.

• Client Financial Asset Information Table. This table keeps records of all kinds of financial assets belonging
to each client, including stock assets, futures assets, gold related assets, saving accounts, etc.

- Client Transaction Information. This tables keeps records of all transactions of financial assets of each client. The former table can be viewed as the static status of a client’s assets, while this table serves as the dynamic behavior of the client’s recent trading events.

In the bank marketing campaign study, the data set is provided by S. Moro. The features include age, job, marital status, education, whether having credit default, account balance, housing loan, personal loan, contact information type, last contact month, last contact day of a week, last contact duration in seconds, number of contacts performed before this campaign, number of days passed since last campaign, outcome of previous campaign of each client.

### 0.6.3 Build the feature table

After all raw data is ready, the next step is to merge different data tables to build a structured feature-label table as training inputs. This in general is not a difficult task. We only needs to merge tables with appropriate key columns. I still want to discuss some tricky details though, as in the first case study we are working on extremely large scale data sets.

The first task is to extract relevantly more important features from each table. Ideally as discussed, we want to keep as much features as possible, but let us be realistic. We are dealing with more than 50,000,000 transactions as training set, if we consider each identical feature in a column as a single feature, i.e. age 20 and age 21 are two different features, we will end up with features of billions of dimensions. Although theoretically we can still train such high dimensional data sets with suitable computation units and memory, or even connecting thousands of machines with Hadoop, most banks are not interested enough to invest that much on a single model building task. With some sense of simplification, we can achieve similar per-
formance with limited computation power. In our case, we finally built the model with a 40-core, 256G memory, 10T disk server, and the raw data is more than 10 Terabytes.

As described in the subsection 0.6.2, we extracted the appropriate columns of features from each table, and then merged them together to create a large table as raw training data. It is just "raw" data in a sense that we will have to adjust the data format according to different training algorithms in later stages. These two tasks of "extraction" and "merging" are not as trivial as they seem to be. If we naively extract the corresponding columns line by line and merge them together without any smart algorithms (some divide-and-conquer tricks), chances are it will take days to extract a file with more than 200,000,000 lines, and it will get even longer to merge tables. Some simple GNU/Linux commands can boost these processes.

```
split -1 20000000 -d $input $input"."

for i in {0..9}; do
  cut -d ";" -f1,2,3,4,5 > $input"_extracted.0"$i &
done

echo "wait all threads to finish"
wait

for i in {0..9}; do
  cat $input"_extracted.0"$i >> $output

done
```

This way we can save the time spent on feature extraction by parallelizing the executions. Theoretically speaking, the more cores you have in your machine, the more time you can save by splitting into more portions.

Another reminder of feature selection is that the features we choose to train models must be the features available in real application scenarios. There will be a lot of engineering or communication aspects to consider. As mentioned previously, ideally we want to get as much information about each transaction as possible, and these features may come
from various sources of departments within a company, or even from external sources such as social network or mobile applications. If this is the case, we have to make sure that when the model we built gets on-line someday, all these features will still be available for predictions.

Last but not least, there is an important rule that needs special attention. All features of each sample must be gathered exactly at the time it is labeled, i.e., we have to avoid the case that the campaign was made 2 months ago but the features such as the assets of the client is as of today. It is very easy to understand that this kind of "time travel" will introduce huge bias to the training data. What is worse, it may also include some indications of the "answer" in the features, which gives much higher weights to these "answer" features. Sometimes if we find the training error (and test error) is extremely low in the experiment, we have to pay extra attention to those highly weighted rules and check if they "time traveled" and revealed the answer somehow.

### 0.6.4 Labeling samples

Labeling samples is not even a problem to discuss in theoretical researches, as most problem settings give specific training and testing data sets. In our case study, there are still some details worth discussing.

Consider the Credit Card Transaction Installment Recommendation problem, we are given a table which records the outcomes of historical campaign information, which states the transactions successfully campaigned and failed. There is not doubt that they represent positive and negative labels. What about the data outside these records, i.e. those transactions that have never been campaigned? The question is, we can take those clients who voluntarily choose to pay back their transaction by installments as positive samples, since they are even more "positive" than our positive samples. However, those negative samples are not actually negative, chances are that they do not know such a business being operated by the campaign. They probably never know that they can pay their money back in several months
instead of right at next due date. If the bank chose to send a campaign message to them, they may still accept the offer and become positive samples.

In our case study, we take all such positive cases as positive, but take only "campaigned" negative samples as negative. This setting is reasonable in a sense that all samples are correctly labeled. The only problem is that we introduced some kind of bias into our data set due to the bigger portion of positive cases than real scenarios, but this is still in the acceptable range and more importantly, positive samples are too scarce and adding positive samples makes it slightly better.

Labeling is an even bigger problem in the Gold related wealth management product sales model. The problem is that the bank does not keep any record of the clients being called, but only the clients who bought such products. In such case, neither the buyers are true positive samples nor the rest of the clients are negative samples. The buyers who received a call and chose to buy the products should be considered as positive samples, but those voluntarily bought the product are more "positive" than positive samples. As for the rest of the client, since we have no records of campaign name list, we cannot say for sure if they can become positive cases if we called them in the campaign.

In this scenario, we can choose some positive only learning algorithms such as K-Nearest-Neighbor (kNN) algorithm, but the performance of kNN is satisfactory enough. Alternatively, we set all buyers as positive samples, and randomly picked 10 times number of clients as negative samples. This way we trained a model with pretty good performance.
0.7 **Gradient Boosted Decision Tree**

This section talks about how we applied the *Gradient Boosted Decision Tree (GBDT)* algorithm in three case studies.

As I have said, the focus of this report is not on the theoretical analysis of machine learning algorithms, but instead to show how we applied these algorithms in real cases and achieve good performances of models. In practical scenarios, we often encounter very different challenges than most theoretical studies. For instance, decision tree is not capable of dealing with very high dimensional features, as it will normally build very complicated trees with high dimensional features. Such decision trees of complexity are not realistic in applications as the algorithm will be too slow. Also, complicated decision tree models, and as we have discussed in [0.4.1](#), are usually accompanied with overfitting. That will obviously make our model a bad one and impossible to work with.

Subsection [0.7.1](#) discusses a dimension reduction trick for decision trees. This trick has proved to work quite well in high dimensional decision tree training. Subsection [0.7.2](#) introduces a very powerful tool XGBoost with Gradient Descent Decision Tree algorithm.

### 0.7.1 Continuify Discrete Features

High dimensional data is a nightmare for decision trees. If we think about the nature of decision tree algorithm, it is just a set of top-down IF-ELSE rules. The main idea of ID3 algorithm is to select the feature that reduces entropy most at each round of node creation. Hence if the data set is of high feature dimensions (more than a billion in Credit Card Transaction Installment Recommendation model), it is nearly impossible to iterate all features step by step. Furthermore, high dimensional data means we need decision tree of large complexity to model. Such complicated trees take even more time to build and are more likely to be of the problem of overfitting.
To solve the problem, we introduce a very useful trick to largely reduce the dimension of the features. The trick is to "continuify discrete features".

Notice that in the credit card transaction installment recommendation model, features such as client number, merchant number (at which the transaction is made) are of extremely high dimension (more than millions), because each single value of a feature will be regarded as a single feature in decision tree model training. Although they share a lot of similarities, they indeed are different features. An intuitive explanation is that some clients are more likely to pay back the transaction amount in installments, so it is reasonable that checking the client number will help identify successful cases. If we do not do anything on this problem, we will end up with a decision tree like "If client is A -> If client is B -> If client is C...", which are in essence tedious similar rules. To solve this problem, intuitively we want to make such discrete features disappear and replace them with a single (continuous) feature, but also incorporate the information behind such discrete features to the new feature.

The new feature we use to replace such huge number of discrete features are the (historical) successful rate of this feature. For example, if a client A made 100 transactions in the history and chose to pay back the money by transaction 10 times, we can replace the client number of A by 0.1. To represent the confidence level of this probability, we also add the denominator, which is 100 in the previous example, as a new feature. With these two numbers, we almost shows how probable this client (the original feature) is likely to be a positive sample, and how confident we are at this conjecture. Notice that it is important to prevent the error of "time traveling", i.e., all successful rates and historical samples should be those before the time this sample is collected. For example, if client A made a transaction at January 1st of 2015, when calculating the successful rate of the client A, we should count all transactions up to January 1st 2015 and as for the successful ones, the cases when he/she chose to pay by installments. It requires we sort all transaction by date and time before continuifying such features.

Notice that for already continuous features, such as trans-
action amounts, there is no need to continuify them any more.

When the data set is huge, it is recommended that we continuify features in parallel. In my case study, I split big table by columns, and continuify such columns in parallel. This again saves us more than 10 hours of time. You may wonder why I keep talking about doing parallel programming at simple data formatting, because it is actually essential in engineering projects. Most of the time it does not matter if we improved the accuracy of algorithm by 0.1 percent or not, but it matters if we can deliver a model in a limited period of time. Time really matters. It is a different problem in scientific research though, in science, we usually do not care too much about time and investment, as long as the new model can improve the performance with 0.1 percent.

0.7.2 XGBoost

XGBoost is a wonderful tool implementing the gradient boosted decision tree algorithm. It is specifically designed for big data analysis, in a sense that it can efficiently train a model with millions of samples. In my case study of more than 50,000,000 samples and about 70 features, the training finishes within 3 hours. With some poorly designed tools, it may causes weeks of time to train such a model, and it is a disaster for an engineering project.

XGBoost is very easy to use, when the labeled data is ready, split them into training set and testing set, just train the data with XGBoost and it will output the model after training. During the training it will show evaluation metrics such as accuracy or AUC so we could get an idea how the training is going. XGBoost can also output the model in a human readable form, such that it shows how the decision tree looks like. Notice that GBDT is a boosting algorithm, hence it will output a list of trees, and the final output is the weighted vote of all trees. Also notice that the prediction of XGBoost is real number between 0 and 1, which stands for the probability that this combination of features will give a positive label.
Finally the case study of Credit Card Transaction Installment Recommendation model achieves training and test AUC of over 0.85, and the two day SMS campaign shows that the name list (those with highest scores) provided by XGBoost model gives 1.35% successful rate, while the old expert designed decision making system only has 0.75% successful rate. Our model is 80% better than the old model. Also, note that the model trained in this way is designed to optimized the successful rate of the campaign. If we want to optimize the money we can make from such campaigns, just simply multiply the score predicted by our model with the transaction amount, and re-ranking the list gives us the name list of largest financial income. In terms of incomes, our model is 60% better than the old model. This improvement means big financial gain for the bank, which calculates to about 300,000 CNY/day.
0.8 Logistic Regression

The basic concepts and ideas of logistic regression has been introduced at 0.4.2. Understanding the theoretical aspects of logistic regression is the foundation of applying it properly in real scenarios, but more work needs to be done if we want to build models with the best performances.

This section discusses all details one needs to know about applying logistic regression to build decision making models and shows how they are used in my three case studies.

0.8.1 Practical Analysis of Logistic Regression

We have discussed that logistic regression is very powerful at handling high dimensional data. Actually, the higher the dimension of features, the better performance we can expect from logistic regression. Adding more features, even irrelevant features usually will not cause troubles to the model training with logistic regression. The final model will simply assign weights of very small absolute values to such irrelevant features, and the performance is still guaranteed by the algorithm. Think another way around, if our training data set does not provide features of enough dimensions, this may cause trouble for our model. Hence one direction to think on enhancing model performance is to increase the dimension of the features of our training data set.

Another restriction of regression is that this model always "thinks" linearly or approximate-linearly. Unlike decision tree, which is capable of picturing any weird kind of data distribution, regression always thinks that it can model all data in a (approximate) linear function, and split the data into two categories with an appropriate threshold. The reality does not always support this hypothesis. Let me tell a very straightforward example I encountered in Gold related wealth management product sales campaign. Given two simple features, the age and the gender of a client. Let us further simplify the feature age by young and old. The feature age or the feature gender alone may not gain
much weight themselves, especially when comparing them to more relevant features like people’s saving account information. However, we may expect that senior ladies are more likely to be interested in buying gold related product rather than any other three categories, i.e., young boys, young girls, and senior men. Our logistic regression model is not good at capturing such important features because it is not in its feature space. As we will further discuss in 0.8.3, manually doing feature combination will sometimes improve model performance dramatically.

0.8.2 Discretize Continuous Features

Discretization of continuous features is a simple but useful way to increase the dimension of features. A good example of continuous feature is the amount of saving in a client’s account. If we do not do anything to this feature, logistic regression will typically assign a weight to this feature and the product of the weight and the amount of saving a client has gives indication of whether the client would like to buy some products. However, is it correct and can we get more information outside of this amount of saving? Regression models always think that every feature is either positively or negatively related to the label by some coefficients, but what if in some range of values, the feature is positively related with the label while in other ranges, they are negatively related. In the previous sample, it may be the case that a client with certain amount of wealth is more likely to buy some gold related wealth management product, while clients with more wealth or less are not that interested. Similar situations may exist in many other continuous features.

One useful trick is to discretize all continuous features in data sets. In my case study, I used the following formula to do the translation,

\[ c = \text{floor}(\log_2(d) \times 10). \]

This way we transformed a certain range of continuous value of a feature into a single feature, and an original continuous feature is transformed into a series of features with values 0
and 1. Hence the dimension is increased and logistic regression can show its ability of dealing with high dimensional data.

### 0.8.3 Feature Combinations

As I have mentioned in the senior lady example, sometimes logistic regression failed to detect important features because the feature itself is not inside the feature space. In the senior lady example, both age and gender are features of data sets, but we do not have the feature of senior lady directly from it. A solution to this problem is to do some feature combinations manually. Some people call this Feature Engineering. In this case, we can add four features, young boys, young girls, senior men, senior ladies as 0,1 features, and append such features at the end of other features, then logistic regression will be able to detect if some feature among them is significantly relevant to the labels.

We can always append such feature combinations at the end of other features, as adding dimensionality will not make a worse model usually. Still it is not reasonable to enumerate all possible combination of features in large scale data sets, as the combinations are more than exponential of number of features. As in big data cases, we are usually dealing with millions of features, it is impossible that we do this kind of manual work. In terms of what kind of features we should consider combining, it is where we should apply the knowledge of business nature, or we can ask the opinions of business experts and try those features that are more likely to be relevant. This is also the reason why I said building machine learning models is not only a science but also an art.
0.9 Evaluations

This section discusses some practical issues at evaluation of models and shows some key statistics of the model in my third case study.

The evaluation is often not an issue in ideal problem settings, as the training data set and test set are both given without any concerns. However, actually, such ideal assumptions may not be adopted in practical applications. Realistic restrictions such as fundings, time, human resources and environmental restrictions impedes appropriate evaluation from taking places. This section discusses a realistic problem existing in model evaluations.

0.9.1 biased training and testing data sets

This is a very common situation in data mining that the data sets we used for training and testing are inherently biased. A good example is the data sets I used to train the Credit Card Transaction Installment recommendation model. The data sets come from historical campaigns, while historical campaign customers are picked by the old expert made decision systems. It is obvious that such data sets contain bias caused by the old decision making systems. We have discussed this problem in Subsection 0.6.4. Our training set is composed of all positive samples (regardless whether campaigned or not) and true negative samples (those unsuccessful campaigned transactions). In such case, any evaluation made on training or testing sets is unconvincing, while we do not have an unbiased set to do evaluations.

We have two options to solve this problem. One is to build an unbiased evaluation set from the biased training set. This project is possible if we try to pick some samples from the biased set sharing nearly the same distribution in all features. However, this work is not easy and it is not guaranteed we can construct such unbiased set with limited testing data. Thus we chose to go another way. We applied
Figure 1: Recommendation Overlapping of Two Models.

our model to the whole population, and generate a recommended transaction list. Observe that the whole population of transactions is obviously an unbiased set. Then we take the overlapping of our name list with another name list of similar size given by the old decision making system. Now we can evaluate the performance of the old decision making system, the performance of the overlapping set of the two recommendation systems, and based on such two performances, we can deduce the performance of our new system. More precisely, as shown is Fig. 1, assume the left circle is the name list given by the old system, and the right circle is the name list given by our new model. Assume the successful rate in the old name list is $p$, and the successful rate in the new name list is $q$, we could draw such conclusion about the successful rate of the middle set by

$$1 - r = (1 - p)(1 - q).$$

Thus we solve the equation that

$$q = 1 - \frac{1 - r}{1 - p}.$$

This calculation proves to be very close to real observations.

### 0.9.2 ROC and AUC

When good test data sets are available, ROC analysis[2005] is a very useful tool to evaluate the performance of a model.
Figure 2 reviews four categories of samples in testing samples. True positives are those positive samples and labeled correctly as positive, false positives are those negative samples but falsely classified as positive, false negative are those truly positive samples but falsely classified as negative, true negatives are those negatives and labeled correctly as negatives. We also define two important concepts, false positive rate (FP Rate) as the ratio of false positive samples and all truly negative samples, and true positive rate (TP Rate) as the ratio of true positive samples and all truly positive samples. These concepts are very straightforward to understand. We pay special attention to the relationship between FP Rate and TP Rate. If we represent FP Rate as x axis and TP Rate as Y axis, we get the famous Receiver Operating Characteristics (ROC). Since we get a curve of FP Rate and TP Rate, integrating this curve gives the Area Under Curve (AUC). AUC is regarded as a key indicator of the model performance. The value of AUC is between 0 and 1 as both FP Rate and TP Rate falls between 0 and 1.
Figure 4: ROC Curve with Logistic Regression Model

Figure 3 and 4 show the ROC Curve of Gradient Boosted Decision Tree and Logistic Regression on my third case study, i.e., the bank marketing campaign data set from S. Moro and Cortez [2011]. The blue lines represents the ROC curve of my models and the green lines is the baseline as random guessing. The intuitive idea is, the bigger area under the ROC curve, the better the performance of the model. Because if the false positive rate remains low when the true positive rate gets very big, it indicates that the model is very good at classifying positive samples. Hence it should be a very good model.

In my third case study, The model trained with Gradient Boosted Decision Tree achieves an AUC of 0.9156, which is higher than any model given in S. Moro and Cortez [2011], while Logistic Regression model gives an AUC of 0.7913, which also indicates good performance at test cases.
0.10 Discussion

Section 0.4, 0.7 and 0.8 have discussed most theoretical aspects and practical issues with decision tree, logistic regression and boosting models. Readers should have a pretty good sense about their theories and application scenarios, and also know how to deal with some common issues related. The purpose of this section is to further discuss the essence of these models and summarize their advantages and disadvantages.

0.10.1 Decision Tree

The essence of decision tree is that it can repeatedly use a single feature to categorize a single data set into two subsets. If we are patient enough, this process never ends until all data subsets are "pure", which means each subset is of the same label, or the subsets have no more features to further split. Hence theoretically, decision tree is very strong at classifying all kinds of data sets. The only problem with decision tree is that the model can grow into huge complexity if we do not give any level constraints. Such complicated trees often suffers the problem of overfitting due to noisy or insufficiency of training data. Hence although we have some techniques such as pruning to address the problems of overfitting, it is still recommended that we never train a tree with too many levels.

A big benefit of working on huge data set is that the training data is often comprehensive and of the same distribution with the population, in another word, they do not have problem of insufficiency. Hence training big data set with not too deep decision tree is not very likely to suffer the problem of overfitting.

Another problem with decision tree is that it lacks the capability of training high dimensional features. The more dimensions, the more complicated the decision tends to be, and the more likely overfitting will occur. The method of continuify discrete features we introduced before solves this
problem perfectly, in a sense it keeps all information we need pertaining to a class of features, but also reduce the dimension dramatically. Think about this, why do we have to keep all discrete values of a feature as single features? Because we want to know exactly how likely this single value will indicate a positive label. After we transform each single value of feature into its percentage of positive labels, they all merge into a single feature but we still have the whole information of how much this feature is likely to indicate a positive label.

Another reason why it works so well with my Credit Card Transaction Installment Recommendation model is that I used decision tree with Boosting. As shown in 0.4.3, boosting is great at turning weak learners into strong learners, and it really helps in this case.

### 0.10.2 Logistic Regression

Logistic regression is actually a more popular algorithm in the industry, mostly because its ability to handle high dimensional features. This algorithm shares some similar thoughts with Support Vector Machine in a sense that they both model features and labels in a linear relationship, i.e. each feature will contribute to the final label of a sample.

The major problem with logistic regression is that it always regard features as independent variables. However, some features are actually dependent inherently. In such cases, feature combination will sometimes cause larger weight or lower weight than simple multiplication of weights. In the eyes of logistic regression model, all features form a perfect space and they together gives perfect prediction of the label. Unlike decision tree, if we give some weird shape of data distribution, regression (or SVM) is not able to perfectly split data into two categories. This is the reason why sometimes they need manual helps and feature combination will help regression get out of the dilemma.

The following table summaries the application scenarios of decision tree and logistic regressions and we shall adjust
data format into their corresponding appetites.

<table>
<thead>
<tr>
<th>Decision Tree</th>
<th>Logistic Regression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous Features</td>
<td>Discrete Features</td>
</tr>
<tr>
<td>Low Dimension</td>
<td>High Dimension</td>
</tr>
<tr>
<td>Isolated Features</td>
<td>Feature Combinations</td>
</tr>
</tbody>
</table>

Another perspective of understanding why these little tricks such as feature combination, continuify and discretization are of great help in model training is due to the fact that machine learning models are not perfect enough to capture the essence of data distribution. They are designed for solving a certain kind of problems. If we want them to solve problems outside their capabilities, we, as intelligent human beings, have to offer some aid and make the problem within their capabilities. This is the art that we have to learn by thorough understanding of learning models and business natures.
0.11 Future Work

Applying machine learning algorithms in bank marketing campaigns is still a relatively new field. Not many research or case studies have been done at this area. Hence a very useful future work is to keep collecting data sets, especially real data sets from banks, and train machine learning models from such data sets. Keep digging useful features in this classification problem will make future model building easier and of better performance.

Different Machine Learning algorithms, such as Neural Network (Deep Learning), K-Nearest-Neighbor, Support Vector Machines, can also be experimented such bank marketing campaign data sets. It will be interesting to find appropriate algorithms in different application scenarios.

Similar machine learning algorithm and methodologies can be applied to many other problems in banking, such as credit control and risk managements. They can also be applied to problems outside the field of banking, for instance, on-line advertising in the games or other industries.

Machine learning of large scale is another interesting topic to explore. Most big data companies today provide big database storage services, distributed system installation such as Hadoop or Spark, but not many provide model building services. Model training on distributed system requires large financial investment and is not practical for many companies, and will also take years for even big banks to proceed big data learning projects. Hence it is worth studying easier ways to train good models with restricted resources.

Finally, it will be very interesting to develop some systems that has the functionality of model training, model updating, and on-line decision making. Such a system will be very useful for many banks on marketing campaigns.
0.12 Conclusion

My final year project shows in details how machine learning with big data techniques can be applied in bank marketing campaign problems. The results of my three case studies shows how machine learning with big data can build decision making systems better than any old expert made models. Such methods and models have huge potential to be applied widely in the banking and also other similar industries. My case studies also shows that with some proper handling of data formatting and feature engineering, the performance of classic machine learning models can be enhanced dramatically.
0.13 Acknowledgement

I want to thank Mr. Yupeng Li, Prof. Francis Lau and Prof. Chuan Wu for their supervision of this whole final year project, and also thank Mr. Wenyuan Dai, Mr. Yuqiang Chen and Mr. Yang Bai for their generous advice on my three case studies.
Bibliography


Tianqi Chen. *Introduction to boosted trees*, 2014.


R. Laureano S. Moro and P. Cortez. *A data-driven approach to predict the success of bank telemarketing. decision support systems."
