University of Chicago

MACS 33002: Introduction to Machine Learning
Final Project

Predicting Bank Turnover with Machine Learning Techniques

Authors and Contribution

Sixue Liu: Model Selection, Coding, Method Description, Results

Jiewen Luo: Introduction, Method Description, Analysis and Results

Yuanxiaoyue Yang: Literature Review, Data and Method Description, Discussion

March 12, 2020

1 Introduction

The banking industry in the United States is highly competitive. There are currently 5294 domestic and international banks providing banking services in the United States. Given such an intensively competitive market, the demand for banking services becomes highly fragmented. This brings a great challenge for banks to maintain and expand their customer base. Provided that relatively higher costs are associated with acquiring new customers and re-acquiring deflected customers, retaining current customers can help banks save their customer managing budget (Verbeke et al., 2011). Besides, in the banking industry, long-term account holders are more likely to generate high profits for the bank (Keramati et al., 2016). Therefore, focusing on existing customers appears to be the most effective way of managing customer relationships in the banking industry. However, because of low shifting costs and highly available alternatives, customers have high flexibility in shifting from one bank to another. Various reasons can cause customers to leave a bank, including the accessibility of frontier technology and products, service satisfaction, interest rate, geographical location, and the variety of products and services (Kumar and Ravi, 2008). This makes managing and maintaining existing customers even harder.

From a bank's perspective, it would be helpful if a bank is able to predict whether a customer is more likely to be a churner or not. Therefore, it is very important for a bank to form a better understanding of how characteristics of potential churners and non-churners differ. In this way, the bank can identify the group of customers it should target on. Such an understanding can improve the bank's strategies in managing customer relationships or attracting new customers.

In order to help a bank to identify its target group of customers, our study aims at assessing the likelihood of customer churn using a publicly available bank customer dataset. We will utilize various machine learning techniques, including logistic regressions, decision tree and random forest, k-nearest neighbors, support vector machine, and quadratic

discriminant analysis to identify the hidden patterns of customer churning behavior and establish a relationship between customer churn and associated customer features. We will then evaluate the prediction results from each technique and select the one with the best general performance. By establishing an optimized machine learning model to form a better understanding of customer churning behavior, our goal is to detect customers that are at risk of leaving the bank in the future, to analyze causes that drive customers' churning behavior, and to eventually help banking firms to develop effective strategies to improve customer retention in the future.

2 Literature Review

This section provides a brief review of existing machine learning literature on various models that have been used in prediction of customer churn. Ngai et al. (2009) classified 900 articles relevant to applications of data mining techniques in customer relationship management from academic literature and identified that classification and association models are the two most commonly used.

Widely used models for solving classification problems include Decision Tree (DT), Logistic Regression, and support vector machine (SVM). Keramati et al. (2016) used a DT method to identify features of churners from electronic banking services and the results show that the model successfully identified features of 5 groups of churners. There are also studies that use DT to study churn prediction problems in telecommunication industry (Keramati et al., 2014; Huang et al., 2012). Nie et al. (2011) used both DT and logistic regression to predict customer churn using credit card data collected from a real Chinese bank and found that results achieved from logistic regression are slightly better than results from DT. Support vector machine (SVM) is also extensively used for customer churn prediction tasks. Huang et al. (2012) compared the performance of DT and SVM in predicting telecommunication customer churn and concluded that the preference between the two methods depends on the decision makers' objectives. Other methods, such as Neural

Networks, Random Forest, Naive Bayes, and K-Nearest Neighbors, have also been used in various similar studies (Zhao and Tai, 2014; Rajamohamed and Manokaran, 2018; Zoric, 2016).

Many recent studies have started employing methodology that combines different techniques. A 2008 study developed an ensemble system based on majority voting that constituted of several machine learning techniques such as random forest, decision tree, logistic regression, support vector machine, multilayer perception, and radiao basis function (Kumar and Ravi, 2008). They tested their method using data of credit card costumers from a Latin American bank and concluded that the majority voting achieved good overall accuracy in prediction. Farquad et al. (2014) proposed a modified SVM method incorporating rules from Naive Bayes Tree and tested it using a bank credit card dataset. It turned out that this comprehensive hybrid approach improved the performance of previous SVM models. Keramati et al. (2014) also proposed a hybrid methodology that employed Artificial Neural Network, K-Nearest Neighbors (KNN), DT, and SVM and found that the hybrid method achieves a considerably higher than 95% accuracy for precision and recall measures, which outperforms any single method.

3 Machine Learning Methods

We will use multiple machine learning models in our project to predict whether bank customers will churn or not based on their demographic characteristics and other related variables. In this section, we will briefly discuss the mechanism of each model we will use. Since whether the customer churn or not is a binary variable, we will mainly use logistic regression and classification methods.

3.1 Logistic Regression

The first approach is logistic regression. A typical approach for Logit model of customer j in group h at time t can be expressed as

$$Y_{jht} = f(X, D, O)'\beta + \epsilon_{jt} \tag{1}$$

In this case, f gives us the interaction between the observables (X), demographics (D), and other related variables (O). We will use both Ridge and LASSO regressions, which are commonly used penalized regression techniques to prevent over-fitting that may result from simple linear regression. We will also explore more on how these penalized regressions fit with the Logit model. Besides, we will also test which of the two penalized regression method will be more suitable in our bank customer churning model.

3.1.1 Ridge Regression ($l2$ penalty)

Ridge is a penalized regression method. The loss function is given by adding a penalty equivalent to square of the magnitude of the coefficients:

$$\min_{\beta,\beta_0} \frac{1}{2}\beta^T\beta + C\sum_{i=1}^{n} log(exp(-y_i(X_i^T\beta + \beta_0)) + 1) \tag{2}$$

The first term in equation 2 means putting a constraint on the coefficients β. If the coefficients take large values, the penalty term C will regularize the coefficients to penalize the optimization function. Notice that C is the inverse of regularization strength. A smaller value of C leads to a stronger regularization. In this case, ridge regression is able to shrink the coefficients when they tend to be large and it helps to reduce the model complexity and multicollinearity.

3.1.2 LASSO Regression ($l1$ penalty)

LASSO is another penalized regression method that is very similar to ridge regression. By Patrick et al. (2015), the loss function is given by

$$\min_{\beta,\beta_0} || \beta ||_1 + C\sum_{i=1}^{n} log(exp(-y_i(X_i^T\beta + \beta_0)) + 1) \tag{3}$$

In this LASSO expression, C is the inverse of regularization strength governing how strictly additional regressors are penalized. The smaller value of C indicates a stronger penalization. The first term in equation (3) means that LASSO takes the least absolute shrinkage approach. Unlike ridge regression which takes the square of the coefficients, magnitudes are also taken into account in a LASSO regression. This regularization method can lead to zero coefficients, which means some of the features can be completely neglected. Therefore, other than avoiding over-fitting, LASSO can also help in selecting correct features.

3.2 Support Vector Machines

Support Vector Machines (SVMs) include a set of supervised machine learning methods that can be used for various classification tasks. namely support vector classification(SVC). When dealing with binary classification, SVC divides observations into two classes with a hyperplane. The main goal is to find the optimal separated hyperplanes through nonlinear mapping to derive correct classifications (Farquad et al., 2014). Therefore, SVC has great ability to model nonlinearities. The problem that SVC is trying to solve is as follow:

$$\min_{\beta,b,\zeta} \frac{1}{2}\beta^T\beta + C\sum_{i=1}^{n}\zeta_i \tag{4}$$

$$\text{Subject to } y_i(\beta^T\phi(x_i)+b) \geq 1-\zeta_i,\ \zeta_i \geq 0, i = 1,..,n$$

where: $\zeta_i = \max(0, 1 - y_i(\beta x_i - b))$, $K(x_i, x_j) = \phi(x_i)^T\phi(x_i)$ is the kernel function, and C is the inverse of regularization strength.

3.3 Quadratic Discriminant Analysis

Quadratic Discriminant Analysis is a type of classifier that can separate two classes of observations by a quadratic decision surface. QDA assumes each class of observations

rule has a Gaussian distribution, but relax the assumption of identical covariance of each class.Given these two assumption,we arrive at the classification function:

$$\text{G(x)=argmax}_k \delta_k(x);$$

where

$$\delta_k(x) = -\tfrac{1}{2} \mid \sum_k \mid -\tfrac{1}{2}(x-\mu_k)^T \sum_k^{-1}(x-\mu_k) + log\pi_k$$

Class $k \in \{1, 2\}$;$\sum_k$ = the covariance matrix for class k; μ_k = the mean for class k.

Since some of the features in our study are potentially correlated, regularized discriminant analysis should be applied. The only difference between a regularized QDA and a non-regularized QDA is the different convairances they used. In this case, the regularized covariance shrinks $\sum_k$ to a pooled covariance $\sum$ as defined by as followed:

$$\hat{\sum}_k(\lambda) = \lambda\hat{\sum}_k + (1-\lambda)\hat{\sum} \tag{5}$$

where λ is the tuning parameter determines the level of shrinkage.

3.4 K-nearest Neighbors

K-nearest Neighbors (KNN) is one of the most frequently used classification algorithm. KNN uses a data sample in which the algorithm separates the data points into several classes in order to predict the classification of a new data point. The prediction process is very simple: To assign the most appropriate class to a data point, the algorithm will firstly calculate the Euclidean distance between the data point and other data points in order to identify its k (a positive integer) nearest neighbors. Then the data point is classified by a plurality vote of its neighboring data points, where it is assigned to the class its k nearest neighbors belong to. The accuracy of prediction highly depends on the choice of k.

There are several reasons why KNN would be a preferred algorithm. One benefit of KNN is the simplicity of its classification rule. Moreover, KNN is a non-parametric method, which means it does not make any assumptions on the underlying distribution of the data, allowing for more flexibility in data structure. Another important feature of KNN is that it is a lazy learning algorithm. A lazy algorithm performs local approximation, which is extremely suitable for large and continuously changing datasets. This makes KNN particularly suitable for the churning prediction problem in our study, because the bank's customer database is likely to constantly change with new customers added.

3.5 Classification Trees

Classification trees approximate functions by partitioning the characteristic space into a series of hypercubes and reporting the average value of the function in each of those partitions. We will use two classification tree estimators: decision tree and random forest (Breiman, 2001) in our analysis.

3.5.1 Decision Tree (with Bagging)

Decision tree is suitable for the goal of finding specific features of churners, as it produces easily understood and visualized classification rules (Keramati et al., 2014). Another reason why decision tree is particularly suitable for churn prediction problems is that these problems often involve using data of customer's demographic information (age, gender, career, education, etc.), transaction method, the length of the customer association, and other account information. As decision tree performs well with numerical and categorical data, it turns out to be a suitable method (Nie et al., 2011). In order to improve stability of tree performance, we will use the bagging method, also known as bootstrap aggregation (Breiman, 1996). By aggregating many decision trees, bagging can significantly improve predictive performance and help overcome some of the disadvantages of a single decision tree. At the same time, bagging can preserve the interpretability of decision tree.

3.5.2 Random Forest

Random forest is another ensemble classification learning method. The basic idea behind random forest is to split a tree for multiple times. Each split will choose a random subsample of m features from the full set of p features. At each split, only the m features are considered and m is usually set to be $\sqrt{p}$. This randomization process can help mitigate the strength of certain strong predictors and will help reduce variance and correlation between predictors. However, compared with bagging, random forest is unable to explicitly illustrate segment results, which is a potential disadvantage.

4 Data

The dataset we will use for our analysis is a free public dataset of bank customer information called "Bank Turnover Dataset" obtained from Kaggle.com[1]. This cross-sectional dataset contains information of 10,000 customers from an anonymous European bank. We also group the variables into 3 different categories based on the type of the information. Here we provide a list of all variables and their descriptions:

Group 1-Demographic Characteristics:

- Gender: A dummy variable indicating the customer's gender
- Geography: A categorical variable indicating the customer's geographic location
- Age: The customer's current age
- Estimated Salary: The customer's estimated salary

Group 2-Usage of bank services:

- Has Credit Card: A dummy variable indicating whether the customer owns a credit card through the bank
- Is Active Member: A dummy variable indicating whether the customer is an active member at the bank

[1]The link to the dataset is: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling.

- Exited: A dummy variable indicating whether the customer exited from the bank services during a six-month period.
- Tenure: The duration of time the customer has been using the bank services
- Balance: The customer's current account balance at the bank
- Number of Products: The number of bank accounts or bank account affiliated products the customer has

Group 3-Quality of customer:

- Credit Score: The customer's current credit score

We provide percentage share of each category among the total number of observations for categorical and dummy variables and summary statistics for numerical variables in Appendix I. One important issue to notice is that the randomness of this data sample is unknown. We will use bootstrap method to mitigate this problem.

5 Analysis and Results

5.1 Data Preprocessing

During the data preprocessing phase, raw data were separated into a vector dataset stored the variable interest (customer churn), and a features dataset contained the remaining feature variables; categorical variables such as gender and geography were recoded into a set of separate binary variables; redundant or useless features including row number, customer Id, and surname were eliminated. These processed samples were then split into two subsets, including 75% training data and 25% testing data for future model assessment.

As mentioned before, our data might not be random, so we are concerned that our data may not meet the normal distribution assumption. Therefore, we will apply a bootstrap method to improve accuracy of our results. Specifically, we resample 8000 observations out of the original 10000 observations in order to improve the randomness of our data. Another potential benefit of bootstrap is to increase the robustness of our estimates.

5.2 Model Selection

We performed two stage processes for selecting the optimal classifier for customer churn. In the first stage, 5-fold cross-validation was applied for searching the ideal tuning parameter for each classifier. Among a list of hyperparameter C in $\{0.01, 0.1, 1, 5, 10\}$ under both l1 and l2 penalty schema, Ridge logistic regression with penalty parameter C = 0.01 yielded the highest level accuracy. Among a list of hyperparameter C in $\{0.1, 1, 10, 100, 1000\}$, SVC with parameter C = 10 yielded the highest level accuracy. As for regularized discriminant analysis, the best parameter $\lambda = 0.1$ was selected from the list of $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. For a list of k in $\{1, 3, 5, 7, 9\}$, the 1-nearest-neighbor classifier outperformed all the other KNN classifiers. When considering the optimal level of tree depth in the list $\{4, 6, 8, 10\}$ using both Gini and entropy selection criteria, the best performer was generated when applied entropy criterion and set the tree depth to be 8. Also, we apply bagging method to select the features in pruning the decision tree, in order to increase the stability. So we choose the max features as 5. Finally, among a list of tree numbers in $\{40, 60, 80, 100\}$, random forest model with 60 trees brought us the highest level of accuracy. The best performer of each classifier was then selected into next stage for further evaluation.

In the second stage, we used the test dataset to assess the performance scores of each candidate classifier and then chose our best classifier based on statistics performance and model interpretability. The performances of each classifier candidate are presented in Table 1. In our case, the random forest model is the best performer with 92.2% accuracy; KNN is in the second place with an accuracy of 89.3%; decision tree has a slighter lower accuracy of 84.5%; QDA and Ridge logistic regression have relatively lower level of accuracy. Further examination of precision rates and recall rates yielded mostly consistent results as our accuracy assessment. However, KNN has the highest level of recall rate, and decision tree achieves pretty great performance in precision. Overall, as shown in Figure 1 below, while random forest certainly performs the best, decision tree, SVC, and KNN classifiers yielded

similar statistical results.

Model	**Accuracy Rate**	**Best Parameters Set**	**Precision**	**Recall**
Logistic Regression ($l2$)	0.8120	$l2$ and $C = 0.01$	0.6153	0.1411
SVM	0.8630	$\lambda = 1$	0.7338	0.4861
QDA	0.8335	parameter = 0.1	0.6797	0.3048
KNN	0.8925	$k = 1$	0.7286	0.7305
Decision Tree	0.8450	entropy and max depth = 8	0.6900	0.3980
Random Forest	0.9220	number of trees = 60	0.9003	0.6826

Table 1. Comparison of Model Performances

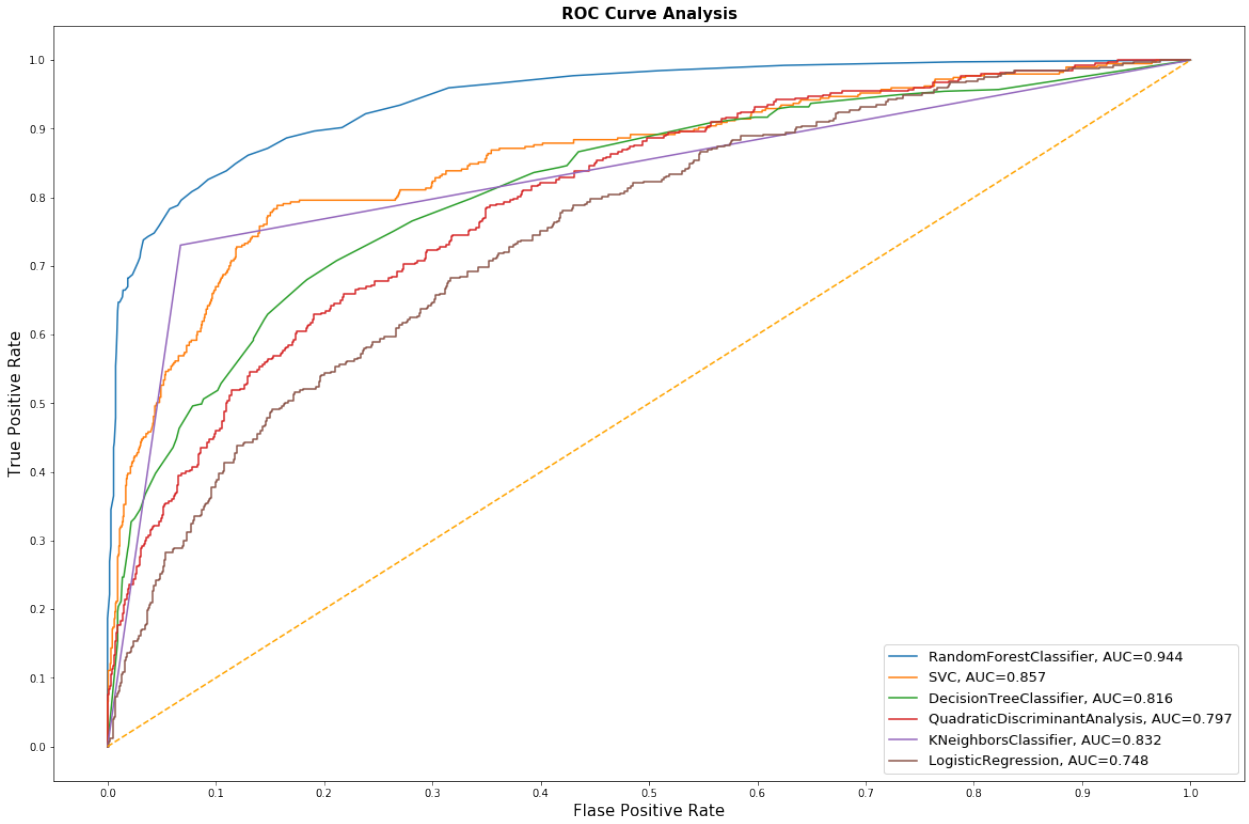


Figure 1. Comparison of ROC curves

Based on this picture, it's no wonder that random forest performs the best. We think we should not only focus on the model performance, but pay attention to the interpretability of the model. We believe that what differentiates our project is we are considering a business related problem. And what we are doing is to dig deeper the potential factors of customer churning, in order to increase "our client's" retention rate in the future. The reason why random forest performs the best is it combines the result of multiple trees. Decision tree method has much better interpretability compared with both random forest and KNN, we utilized the decision tree model for the following customer churn analysis.

5.3 Customer Churn Analysis: Results from Decision Tree

Figure 2 below is a simplified illustration of the decision tree results. The complete illustration can be found at the end of Appendix II.

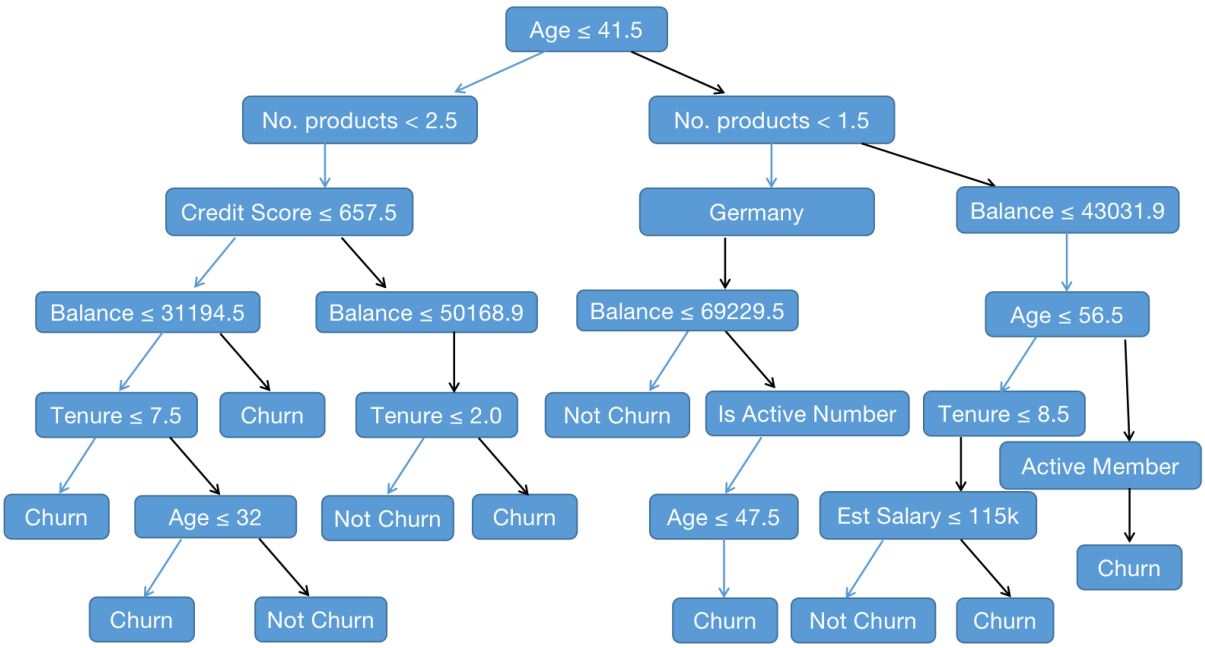


Figure 2. Decision tree results

The proceeding context illustrates all possible scenarios for customer churn based on our decision tree model:

1. When under 41.5-year-old customers own less than 2.5 bank products and with a credit score lower than or equal to 409, they will almost certainly churn.
2. When age between 37.5 years and 41.5 years customers own less than 2.5 bank products but with a credit score above 409, they are likely to churn if they are non-Germany and their account balance is in between $169555.91 and $172042.12.
3. When age between 37.5 years and 41.5 years customers own less than 2.5 bank products and with a credit score between 409 and 623, they are likely to churn if they are non-Germany and they have over $202018.81 account balance and less than 2.5 years tenure.
4. When age between 37.5 years and 41.5 years customers own less than 2.5 bank products and with a credit score above 623, they will almost certainly churn if they are non-non-Germany.
5. When age between 37.5 years and 41.5 years customers own less than 2.5 bank products, they are likely to churn if they are Germany and they have over $150385.56 account balance and 755.5-763 credit score.
6. When under 41.5-year-old customers own more than 2.5 bank products, they are likely to churn if they have less than 657.5 credit score but above $31194.52 account balance.
7. When under 41.5-year-old customers own more than 2.5 bank products and have less than 657.5 credit score, they are likely to churn if they also have less than $31194.52 account balance and less than 7.5 years tenure.
8. When under 32-year-old customers own more than 2.5 bank products and have less than 657.5 credit score, they are likely to churn if they also have less than $31194.52 account balance and over 7.5 years tenure.
9. When under 41.5-year-old customers own more than 2.5 bank products and have greater than 657.5 credit score, they are likely to churn if they also have more than $50168.98 account balance and more than 2 years tenure.

10. When under 41.5-year-old customers own more than 2.5 bank products and have greater than 657.5 credit score, they are likely to churn if they have less than $50168.98 account balance and less than 1.5 years tenure.
11. When under 41.5-year-old customers own more than 2.5 bank products and have greater than 822 credit score, they are likely to churn if they have less than $50168.98 account balance and more than 1.5 years tenure.
12. When over 41.5-year-old customers own less than 2.5 bank products, they are likely to churn if they are Germany.
13. When age between 41.5 years to 50.5 years customers own less than 2.5 bank products, they are likely to churn if they are non-Germany and no-active members.
14. When age between 41.5-year-old customers own over 2.5 bank products, they also have a high risk of leaving the bank.

6 Concluding Remarks and Discussion

Based on our previous analysis of model performances, we conclude that decision tree is the most recommended method for bank customer churning prediction problem. As our project provides a case of machine learning application in business management, the highly visualized and easily understood result from decision tree has significant advantages over other techniques. The clear segmentation of customers helps to identify needs of different customer groups, which will help the bank to design different types of service targeted on different customer groups accordingly. The better understanding of the features of churners can also help bank managers to develop better strategies aimed on customers with features growing more similar to the churner groups. Such strategies will help the bank to improve its retention rate.

An extended implication of our study is that our results have shown that simple machine learning methods can be highly applicable in service-based businesses. The machine

learning methods we used in our study can be applied in almost any service industry, such as telecommunications or network providers. A proper analysis of customer behaviors using machine learning techniques can help these service companies to improve their strategies in customer relationship management. Compared with techniques that require high level of expertise in the machine learning field, the techniques we selected in our study are easy to digest but can be powerful tools in business applications. Therefore, these techniques are of high practical value.

We also acknowledge that there are some unavoidable limitations in our study. Firstly, as our data is downloaded from a public website, the data source is unclear. Specifically, we do not know which bank this dataset belongs to. However, as customer information is certainly credential, it is reasonable to hide the name of the bank when publishing the dataset online. Also, the purpose of our study is to find the best machine learning technique for a churning prediction problem. We do not aim on providing professional data analysis services to any particular bank, so it does not matter too much if we do not know which bank's data we are using.

Another drawback of our data is that we have a limited number of features. We only have customer-related variables but do not have any bank-related ones. However, as our project aims at predicting customer behaviors within some individual bank instead of trying to understand general patterns of churning behavior in the entire banking industry, bank-related variables are less relevant in our case. Plus, considering the real world scenario, the variables we have in our dataset are likely the only type of information available in a bank's customer database. One can imagine that a bank can hardly obtain similar information from its competitors. Therefore, the limited amount of information we have can still make our project a valid business application. If we are able to obtain customer information from multiple banks along with more bank-related information, we will be more capable of studying the generalized patterns of customer churning behaviors in the banking industry in future research.

References

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5.

Farquad, M. A., Ravi, V., and Raju, S. B. (2014). Churn prediction using comprehensible support vector machine: An analytical CRM application. *Applied Soft Computing Journal*, 19:31–40.

Huang, B., Kechadi, M. T., and Buckley, B. (2012). Customer churn prediction in telecommunications. *Expert Systems with Applications*, 39(1):1414–1425.

Keramati, A., Ghaneei, H., and Mirmohammadi, S. M. (2016). Developing a prediction model for customer churn from electronic banking services using data mining. *Financial Innovation*, 2(1).

Keramati, A., Jafari-Marandi, R., Aliannejadi, M., Ahmadian, I., Mozaffari, M., and Abbasi, U. (2014). Improved churn prediction in telecommunication industry using data mining techniques. *Applied Soft Computing Journal*, 24:994–1012.

Kumar, D. A. and Ravi, V. (2008). Predicting credit card customer churn in banks using data mining. *International Journal of Data Analysis Techniques and Strategies*, 1(1):4–28.

Ngai, E. W., Xiu, L., and Chau, D. C. (2009). Application of data mining techniques in customer relationship management: A literature review and classification. *Expert Systems with Applications*, 36(2 PART 2):2592–2602.

Nie, G., Rowe, W., Zhang, L., Tian, Y., and Shi, Y. (2011). Credit card churn forecasting by logistic regression and decision tree. *Expert Systems with Applications*, 38(12):15273–15285.

Patrick, B., Denis, N., Stephen P., R., and Miaoyu, Y. (2015). Machine learning methods for demand estimation. *The American Economic Review*, 105(5):481.

Rajamohamed, R. and Manokaran, J. (2018). Improved credit card churn prediction based on rough clustering and supervised learning techniques. *Cluster Computing*, 21(1):65–77.

Verbeke, W., Martens, D., Mues, C., and Baesens, B. (2011). Building comprehensible customer churn prediction models with advanced rule induction techniques. *Expert systems with applications*, 38(3):2354–2364.

Zhao, S. X. and Tai, Q. Y. (2014). Applied research on data mining in bank customer churn. *Applied Mechanics and Materials*, 687-691:5023–5027.

Zoric, A. B. (2016). Predicting customer churn in banking industry using neural networks. *Interdisciplinary Description of Complex Systems*, 14(2):116–124.

Appendix I. Summary Statistics Tables

Variables	**Percentage Share**
Gender	
Female	45.43%
Male	54.57%
Geography	
France	50.14%
Germany	25.09%
Spain	24.77%
Has Credit Card	
Yes	70.55%
No	29.45%
Is Active Member	
Yes	51.51%
No	48.49%
Exited	
Yes	20.37%
No	79.63%
Total Number of Observations	10000

Table 2. Summary Statistics for Categorical and Dummy Variables

Variables	**Mean**	**Standard Deviation**	**Min**	**Median**	**Max**
Age	38.92	10.49	18	37	92
Credit Score	650.53	96.65	350	652	850
Tenure	5.01	2.89	0	5	10
Balance	76,485.89	62,397.41	0	97,199	250,898.1
Estimated Salary	100,090.2	57,510.49	11.58	100,193.91	199,992.5
Number of Products	1.53	0.58	1	1	4
Total Number of Observations					10000

Table 3. Summary Statistics for Numerical Variables

Appendix II. Python Code

All the codes along with the output are attached starting on the next few pages.

In [0]:
```
!pip install -U -q PyDrive

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:
```
!pip install -U -q graphviz
!pip install -U -q pydotplus
```

In [0]:
```
link = 'https://drive.google.com/open?id=1XjREQWN8p_-Q9Uz-XOSFtAGre6Sjq5vM'
fluff, id = link.split('=')
file = drive.CreateFile({'id':id})
file.GetContentFile('bank_churn.csv')
```

In [0]:
```
import pandas as pd
import numpy as np
from sklearn import model_selection
```

In [223]:
```
bank = pd.read_csv('bank_churn.csv')
bank.head()
```

Out[223]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	N
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	
3	4	15701354	Boni	699	France	Female	39	1	0.00	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	

In [0]:
```
#bagging
from sklearn.utils import resample

bank = resample(bank, replace=True, n_samples=8000, random_state=0)
```

In [0]:
```
y = bank['Exited']
```

In [226]:
```
bank['Gender'] = bank['Gender'] == 'Female'
bank = pd.get_dummies(bank, columns=['Geography'], drop_first=True)
to_drop = ['RowNumber','CustomerId','Surname','Exited']
X = bank.drop(to_drop, axis=1)
X.head()
```

Out[226]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
2732	623	True	48	1	108076.33	1	1	0
9845	590	True	38	9	0.00	2	1	1
3264	738	False	35	5	161274.05	2	1	0
4859	794	True	22	4	114440.24	1	1	1
9225	594	True	32	4	120074.97	2	1	1

In [227]:
```
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8000 entries, 2732 to 9669
Data columns (total 15 columns):
RowNumber            8000 non-null int64
CustomerId           8000 non-null int64
Surname              8000 non-null object
CreditScore          8000 non-null int64
Gender               8000 non-null bool
Age                  8000 non-null int64
Tenure               8000 non-null int64
Balance              8000 non-null float64
NumOfProducts        8000 non-null int64
HasCrCard            8000 non-null int64
IsActiveMember       8000 non-null int64
EstimatedSalary      8000 non-null float64
Exited               8000 non-null int64
Geography_Germany    8000 non-null uint8
Geography_Spain      8000 non-null uint8
dtypes: bool(1), float64(2), int64(9), object(1), uint8(2)
memory usage: 835.9+ KB
```

In [0]:
```
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, t
```

In [0]:
```
import random
seed_value = 123
random.seed(seed_value)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

Logistic = LogisticRegression()
KNN = KNeighborsClassifier()
RF = RandomForestClassifier()
DT = DecisionTreeClassifier()
SVM = SVC()
QDA = QuadraticDiscriminantAnalysis()
```

In [230]:
```
model_names = ['Logistic','KNN','Random Forest', 'Decision Tree', 'Support
model_list = [Logistic, KNN, RF, DT, SVM, QDA]
count = 0

for classifier in model_list:
    cv_score = model_selection.cross_val_score(classifier, X_train, y_train
    print('Model accuracy of ' + model_names[count] + ' is ' + str(cv_score
    count += 1
```

```
Model accuracy of Logistic is 0.8116666666666668
Model accuracy of KNN is 0.8368333333333332
Model accuracy of Random Forest is 0.9131666666666666
Model accuracy of Decision Tree is 0.8766666666666666
Model accuracy of Support Vector Machines is 0.8618333333333332
Model accuracy of Quadratic Discriminant Analysis is 0.8363333333333334
```

In [0]:
```
from sklearn.model_selection import GridSearchCV

def print_grid_search_metrics(gs):
    print ("Best score: " + str(gs.best_score_))
    print ("Best parameters set:")
    best_parameters = gs.best_params_
    for param_name in sorted(parameters.keys()):
        print(param_name + ':' + str(best_parameters[param_name]))
```

In [232]:
```
parameters = {
    'penalty':('l1', 'l2'),
    'C':(0.01, 0.1, 1, 5, 10)
}
Grid_LR = GridSearchCV(LogisticRegression(solver='liblinear'),parameters, c
Grid_LR.fit(X_train, y_train)
```

Out[232]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=
False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=N
one,
                                          max_iter=100, multi_class='aut
o',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='libl
inear',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': (0.01, 0.1, 1, 5, 10), 'penalty': ('l1', 'l
2')},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
```

In [233]:
```
print_grid_search_metrics(Grid_LR)
```

```
Best score: 0.8138333333333334
Best parameters set:
C:0.01
penalty:l2
```

In [0]:
```
best_LR_model = Grid_LR.best_estimator_
```

In [235]:
```
parameters = {
    'n_estimators' : [40,60,80,100]
}
Grid_RF = GridSearchCV(RandomForestClassifier(),parameters, cv=5)
Grid_RF.fit(X_train, y_train)
```

Out[235]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=
0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth
=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.
0,
                                              n_estimators=100, n_jobs=No
ne,
                                              oob_score=False,
                                              random_state=None, verbose=
0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'n_estimators': [40, 60, 80, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
```

In [236]:
```
print_grid_search_metrics(Grid_RF)
```

```
Best score: 0.9155
Best parameters set:
n_estimators:60
```

In [0]:
```
best_RF_model = Grid_RF.best_estimator_
```

In [238]:
```
parameters = {
    'n_neighbors':[1, 3, 5, 7, 9]
}
Grid_KNN = GridSearchCV(KNeighborsClassifier(),parameters, cv=5)
Grid_KNN.fit(X_train, y_train)
```

Out[238]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=3
0,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=No
ne,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid={'n_neighbors': [1, 3, 5, 7, 9]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
```

In [239]:
```
print_grid_search_metrics(Grid_KNN)
```

```
Best score: 0.8675
Best parameters set:
n_neighbors:1
```

In [0]:
```
best_KNN_model = Grid_KNN.best_estimator_
```

In [242]:
```
parameters = {
    'criterion':['gini', 'entropy'],
    'max_depth': [4, 6, 8],
    'max_features': ['sqrt', 'log2', 5]
}
Grid_DT = GridSearchCV(DecisionTreeClassifier(),parameters, cv=5)
Grid_DT.fit(X_train, y_train)
```

Out[242]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight
=None,
                                              criterion='gini', max_depth
=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.
0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 6, 8],
                         'max_features': ['sqrt', 'log2', 5]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
```

In [243]:
```
print_grid_search_metrics(Grid_DT)
```

```
Best score: 0.861
Best parameters set:
criterion:entropy
max_depth:8
max_features:5
```

In [0]:
```
best_DT_model = Grid_DT.best_estimator_
```

In [245]:
```
parameters = {'C': [0.1, 1, 10, 100, 1000],
              }
Grid_SVC = GridSearchCV(SVC(probability=True),parameters, cv=5)
Grid_SVC.fit(X_train, y_train)
```

Out[245]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=True, random_state=None, shrinking
=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.1, 1, 10, 100, 1000]}, pre_dispatch='2*n
_jobs',
             refit=True, return_train_score=False, scoring=None, verbose=
0)
```

In [246]:
```
print_grid_search_metrics(Grid_SVC)
```

```
Best score: 0.8713333333333333
Best parameters set:
C:10
```

In [0]:
```
best_SVM_model = Grid_SVC.best_estimator_
```

In [248]:
```
parameters = {'reg_param': [0.1, 0.2, 0.3, 0.4, 0.5]
              }
Grid_QDA = GridSearchCV(QuadraticDiscriminantAnalysis(),parameters, cv=5)
Grid_QDA.fit(X_train, y_train)
```

Out[248]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=QuadraticDiscriminantAnalysis(priors=None, reg_par
am=0.0,
                                                     store_covariance=Fal
se,
                                                     tol=0.0001),
             iid='deprecated', n_jobs=None,
             param_grid={'reg_param': [0.1, 0.2, 0.3, 0.4, 0.5]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=Fals
e,
             scoring=None, verbose=0)
```

In [249]:
```
print_grid_search_metrics(Grid_QDA)
```

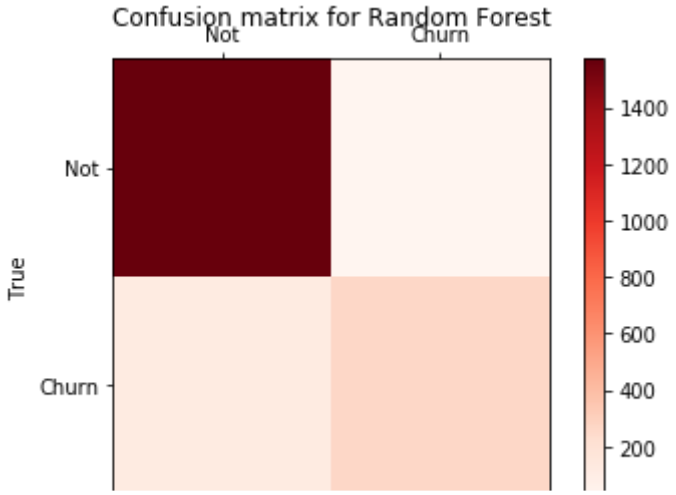
```
Best score: 0.8355
Best parameters set:
reg_param:0.1
```

In [0]:
```
best_QDA_model = Grid_QDA.best_estimator_
```

In [0]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from matplotlib import pyplot as plt

def cal_evaluation(classifier, cm):
    tn = cm[0][0]
    fp = cm[0][1]
    fn = cm[1][0]
    tp = cm[1][1]
    accuracy  = (tp + tn) / (tp + fp + fn + tn + 0.0)
    precision = tp / (tp + fp + 0.0)
    recall = tp / (tp + fn + 0.0)
    print (classifier)
    print ("Accuracy is: " + str(accuracy))
    print ("Precision is: " + str(precision))
    print ("Recall is: " + str(recall))

def draw_confusion_matrices(confusion_matricies):
    class_names = ['Not','Churn']
    for cm in confusion_matrices:
        classifier, cm = cm[0], cm[1]
        cal_evaluation(classifier, cm)
        fig = plt.figure()
        ax = fig.add_subplot(111)
        cax = ax.matshow(cm, interpolation='nearest',cmap=plt.get_cmap('Red
        plt.title('Confusion matrix for ' + classifier)
        fig.colorbar(cax)
        ax.set_xticklabels([''] + class_names)
        ax.set_yticklabels([''] + class_names)
        plt.xlabel('Predicted')
        plt.ylabel('True')
        plt.show()
```

In [252]:
```
confusion_matrices = [
                      ("Random Forest", confusion_matrix(y_test,best_RF_mod
                      ("Logistic Regression", confusion_matrix(y_test,best_
                      ("K nearest neighbor", confusion_matrix(y_test, best_
                      ("Decision Tree", confusion_matrix(y_test, best_DT_mo
                      ('Support Vector Machines', confusion_matrix(y_test,
                      ('Quadratic Discriminant Analysis', confusion_matrix(
]

draw_confusion_matrices(confusion_matrices)
```

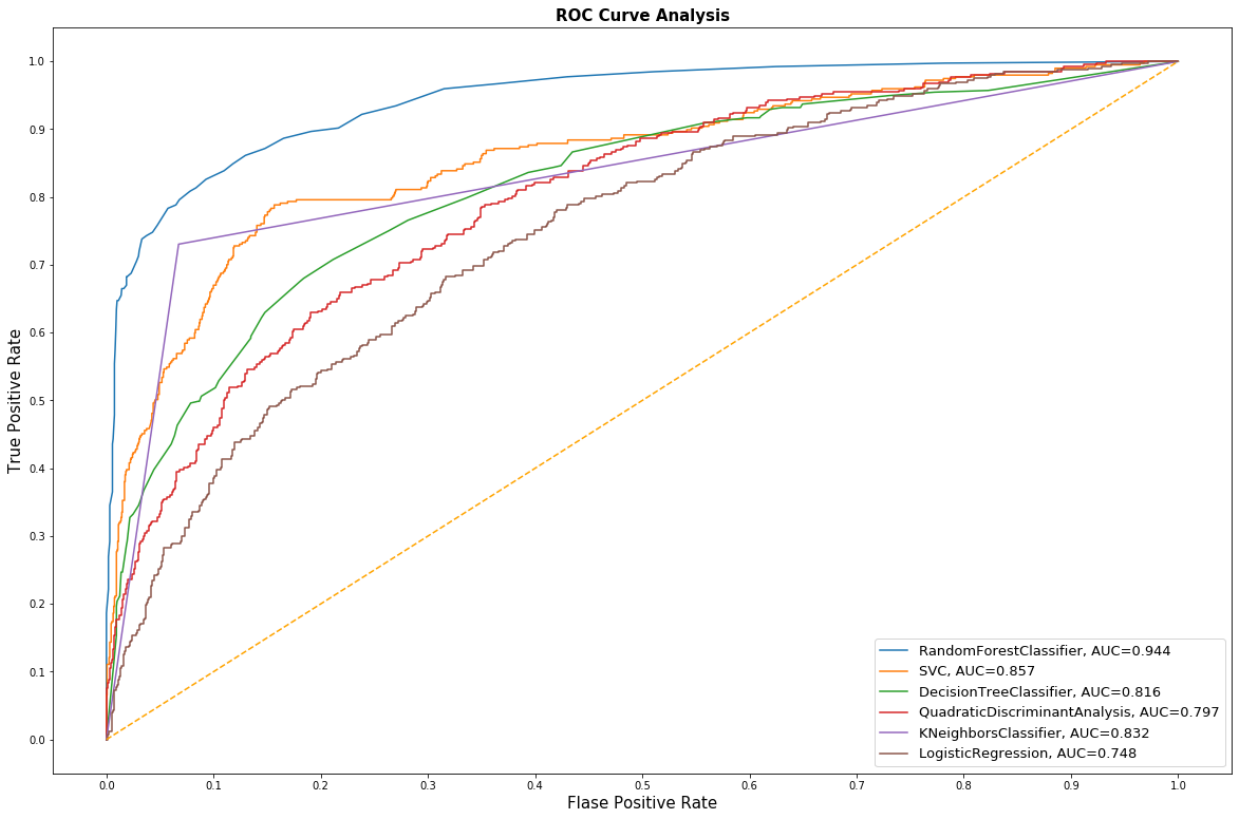
```
Random Forest
Accuracy is: 0.922
Precision is: 0.9003322259136213
Recall is: 0.6826196473551638
```



In [0]:
```
```

In [0]:
```
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn import metrics

# Instantiate the classfiers and make a list
classifiers = [best_RF_model,
        best_SVC_model,
        best_DT_model,
        best_QDA_model,
        best_KNN_model,
        best_LR_model]

# Define a result table as a DataFrame
result_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])

# record the prediction results
for cls in classifiers:
    yproba = cls.predict_proba(X_test)[::,1]

    fpr, tpr, _ = roc_curve(y_test,  yproba)
    auc = roc_auc_score(y_test, yproba)
    result_table = result_table.append({'classifiers':cls.__class__.__name_
                                        'fpr':fpr,
                                        'tpr':tpr,
                                        'auc':auc}, ignore_index=True)

# Set name of the classifiers as index labels
result_table.set_index('classifiers', inplace=True)
```

In [254]:
```
fig = plt.figure(figsize=(20,13))

for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(i, result_table.loc[i]['auc']))

plt.plot([0,1], [0,1], color='orange', linestyle='--')

plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("Flase Positive Rate", fontsize=15)

plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)

plt.title('ROC Curve Analysis', fontweight='bold', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')

plt.show()
fig.savefig('multiple_roc_curve.png')
```



In [0]:
```
feature_cols = X.columns
```

In [191]:
```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

#fig = plt.figure()
dot_data = StringIO()
export_graphviz(best_DT_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('decision tree.png')
Image(graph.create_png())

#fig.savefig('diabetes.png')
```

Out[191]: