ITM 6899

Capstone Project

Student Name: Trang Nguyen

Advisor: Dr. Chongqi Wu

Date: May 16th 2018

Overview & Purpose

Units: 1

Development and writing of business analytics project.

The purpose of the project is to practice the knowledge obtained during the study of the business analytics program with a specification in classification problem in Machine learning. The project will use a fabled company and dataset to demonstrate a real-world business problem, after the course, student will learn in a real industry classification problem, the classification project cycle and related techniques, algorithms, metrics, evaluation methods and so on.

Remarks and Requirements:

1. **Must enroll in ITM 6899 Capstone Project with Dr. Chongqi Wu to receive grade for the Capstone Project.**
2. **Familiar with Python and Jupyter notebook.**
3. **Students may work in group of up to 4 students.**
4. **Submit a project report which addresses all the eight questions at the end of this file. Project report must also include one page of executive summary.**
5. **Submit your Jupyter notebook along with the report.**
6. **Use this file as the cover of your project report.**

Objectives

1. To get familiar with data exploration steps in a Machine Learning problem
2. To practice the visualization methods of different feature analysis
3. To learn the typical type of machine learning problem - binary classification and its model building cycle, as well as evaluation metrics
4. To learn the importance of deal with the missing data, outlier in the dataset
5. To get an understanding of what is an imbalanced data classification problem

Business Problem Description

Beta is an online e-commerce company. The company is interested to know in an early stage, after their customer convert to a paid customer, whether they could become a VIP consumer of their website or not within a month (30 days). The have a dataset where is observed and aggregated during their first 7 days since the first date they made their first purchase. The dataset and its features are explained as below. Once they have the classifier, they could target those VIP customers with personalized treatment.

The Task

Use this dataset to build a binary classifier to use the first 7 days of data since a customer convert, whether they will become a VIP customer for the business within 30 days since their first conversion. The definition VIP by day 30 conversion is defined as a customer spend equal or more than $500 by day 30.

Dataset Description

1. IsVIP_500: target variable, class label, 1 means is a VIP by day 30, 0 means not.
2. payment_7_day: total payment made by day 7 of conversion
3. dau_days: distinct days of customer login to the website.
4. days_between_install_first_pay: number of days since the user registered on the website
5. total_txns_7_day: total transactions the customer made on the website in the first 7 days.
6. total_page_views: number of product items the customer viewed on the website in the first 7 days.
7. total_product_liked: number of product items they have marked like during their views in the first 7 days
8. product_like_rate: the products liked divided by viewed products
9. total_free_coupon_got: number of free coupons the customer got during the first 7 days after conversion.
10. total_bonus_xp_points: total bonus points customer got during the first 7 days, where they could use it as cash with certain redeem rate.

Task Break Down and Submission Requirements

Please use jupyter notebook and python as program language to answer the following questions. One may use text, code, plot to support your answers.

1. Describe the data with statistical information about each feature.
2. Visualization of each feature and the target variable (class distribution).
3. Is there any missing data in the dataset? If so, how to deal with it? What are the methods to deal with missing data?
4. Is there any outlier in the data set? If so, how to deal with it? What are the methods to deal with outliers?
5. Show how to build the classifier and how you will evaluate the results? What is the metrics you will use?
6. What algorithms you plan to use and why? (at least 2 or more)
7. How would compare the results of the algorithms? Which is better, why?
8. Given the current model, how would you improve the results? The data is imbalanced, please give a few methods that the academic will use for imbalanced data, pick one to experiment with your current dataset and see how the result will change.

Project Report

1. Describe the data with statistical information about each feature.

Firstly, we read the csv file into Jupyter notebook and print out the first 5 observations. We also separate the data characters into 2 groups: training features and target. Training features includes nine variables used for binary classifier; and target is IsVIP_500, recorded as the factor to classify.

```
In [72]: file = r'Website_VIP_User_data_10000.csv'
         df = pd.read_csv(file)
         #drop the Observation number
         df=df.drop(df.columns[0], axis=1)

In [73]: df.head(5)
```

Out[73]:

	IsVIP_500	payment_7_day	dau_days	days_between_install_first_pay	total_txns_7_day	total_page_views	total_product_liked	product_like_rate	total_free_cou
0	0	15.98	6	8	3	266	95	0.357143	
1	0	8.48	7	372	3	484	118	0.243802	
2	0	3.49	7	439	2	504	151	0.299603	
3	0	3.99	7	570	2	513	165	0.321637	
4	0	38.00	3	1741	4	31	6	0.193548	

```
In [56]: training_features = ['payment_7_day', 'dau_days', 'days_between_install_first_pay', 'total_txns_7_day', 'total_page_views', 'tota
         target = 'IsVIP_500'
```

Next, we examine the number of VIP customers in the dataset. Among 10001 customers, there are only 155 potential VIP customers, which might be a big challenge to train the dataset and predict the potential customers. Next, the mean values of each features are presented by each group of customers. For example, the VIP customers always make higher payment and more transactions in company website, and they also have much more views and likes on the company products. Finally, we summarize the descriptive statistics of all training features to have overall view on dataset.

Descriptive statistics

```
In [6]: pd.crosstab(index=df[target], columns="count")
```

Out[6]:

col_0	count
IsVIP_500	
0	9846
1	155

```
In [7]: df.groupby(df[target]).mean()
```

Out[7]:

	payment_7_day	dau_days	days_between_install_first_pay	total_txns_7_day	total_page_views	total_product_liked	product_like_rate	total_free_coupo
IsVIP_500								
0	20.734284	4.705159	647.884725	3.206886	365.892139	91.833841	-inf	6.7
1	85.172581	5.993548	495.451613	6.309677	955.451613	268.309677	0.210652	9.6

```
In [8]: df[training_features].describe()
```

Out[8]:

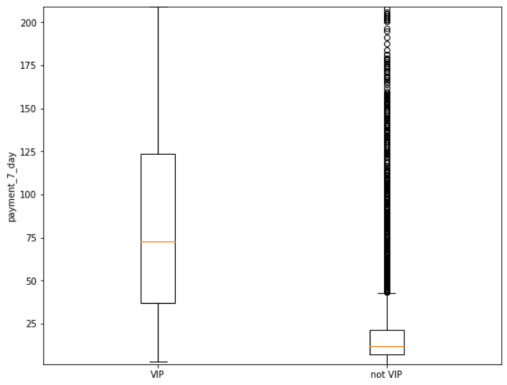
	payment_7_day	dau_days	days_between_install_first_pay	total_txns_7_day	total_page_views	total_product_liked	product_like_rate	total_free_coupo
count	10001.000000	10001.000000	10001.000000	10001.000000	10001.000000	10001.000000	1.000100e+04	10001.0
mean	21.732978	4.725127	645.522248	3.254975	375.029397	94.568943	-inf	6.8
std	28.614734	2.193795	867.677542	2.228897	534.204200	156.700696	NaN	7.4
min	1.510000	1.000000	1.000000	2.000000	-9.000000	-19.000000	-inf	0.0
25%	7.150000	3.000000	17.000000	2.000000	56.000000	2.000000	1.107383e-01	2.0
50%	11.990000	5.000000	97.000000	3.000000	187.000000	39.000000	2.159624e-01	6.0
75%	22.860000	7.000000	1226.000000	3.000000	478.000000	122.000000	2.854220e-01	11.0
max	208.930000	7.000000	3242.000000	83.000000	7620.000000	2151.000000	1.900000e+01	331.0

2. Visualization of each feature and the target variable (class distribution).

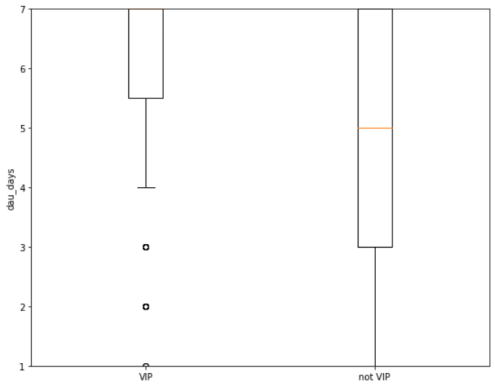
To visualize all training features, I use box plot to display the center and spread of these features grouped by the target variable. These graphs allow us to quickly obtain a sense of the range and skew of each feature.

These following plots show that most variables have outliers on both the high and low ends; therefore, the mean value was more meaningful than the median in this case.

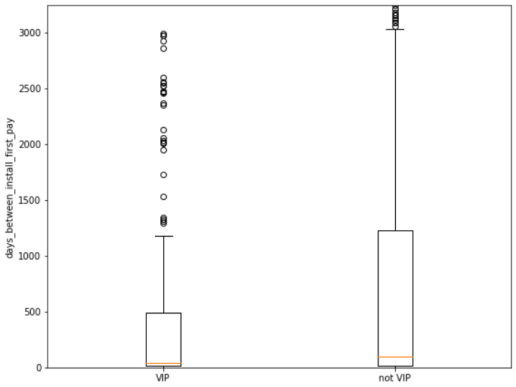
Out[66]: Text(0,0.5,'payment_7_day')



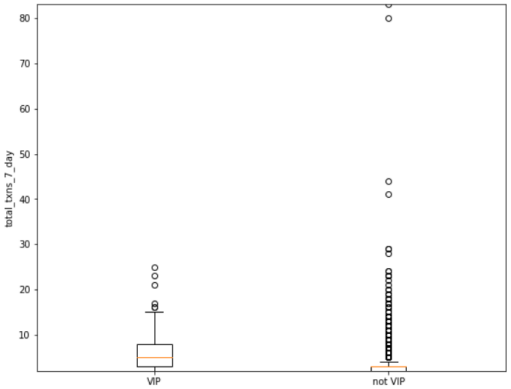
Out[62]: Text(0,0.5,'dau_days')



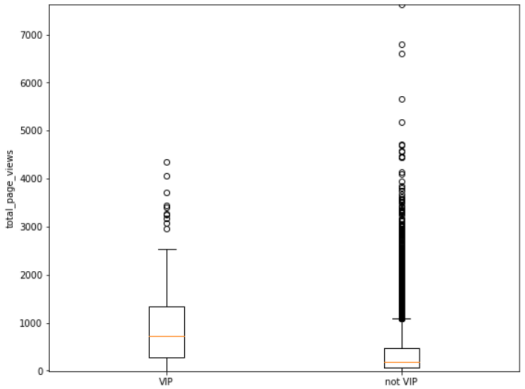
Out[60]: Text(0,0.5,'days_between_install_first_pay')



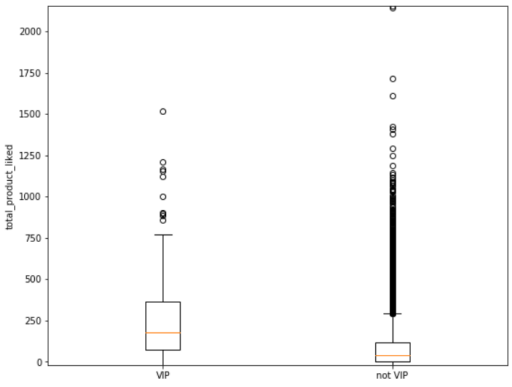
Out[59]: Text(0,0.5,'total_txns_7_day')



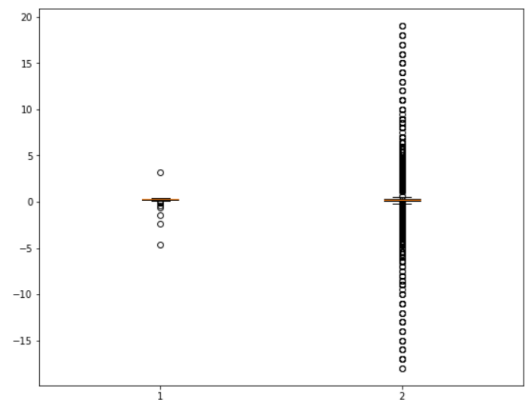
Out[65]: Text(0,0.5,'total_page_views')



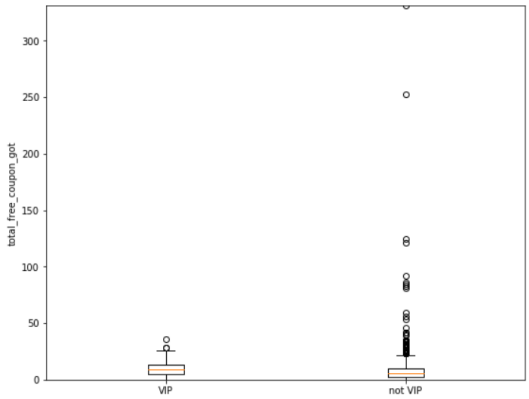
Out[67]: Text(0,0.5,'total_product_liked')



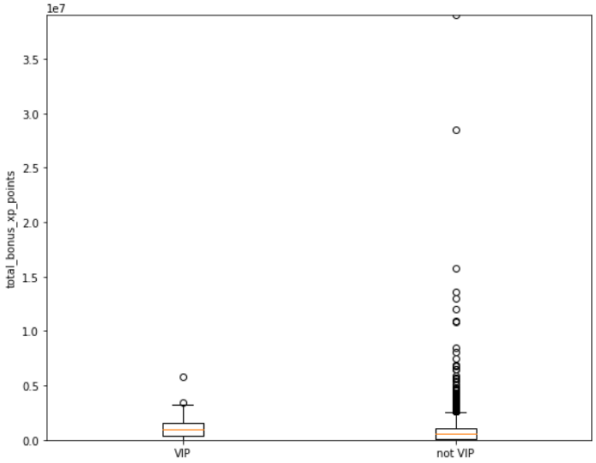
```
ValueError: Axis limits cannot be NaN or Inf
```




```
Out[69]: Text(0,0.5,'total_free_coupon_got')
```




```
Out[70]: Text(0,0.5,'total_bonus_xp_points')
```



3. Is there any missing data in the dataset? If so, how to deal with it? What are the methods to deal with missing data?

There are missing values in the dataset. When visualizing the Product_like_rate feature, we got value error because of some negative infinity. These missing values may cause errors with some machine learning algorithms; therefore, we try to impute them by using a model to replace them. There are some methods to deal with this problem:

- A value from randomly selected records
- A mean, median or mode value for the column
- A value estimated by another predictive model

In [8]:
```
from numpy import inf
a= df['product_like_rate']
a[a== -inf] = 0 #Replace negative infinity by 0
```

```
C:\Users\Trang Nguyen\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [9]:
```
#Statistical description
df.describe()
```

Out[9]:

	Unnamed: 0	IsVIP_500	payment_7_day	dau_days	days_between_install_first_pay	total_txns_7_day	total_page_views	total_product_liked	produc
count	10001.000000	10001.000000	10001.000000	10001.000000	10001.000000	10001.000000	10001.000000	10001.000000	10
mean	5000.000000	0.015498	21.732978	4.725127	645.522248	3.254975	375.029397	94.568943	
std	2887.184355	0.123530	28.614734	2.193795	867.677542	2.228897	534.204200	156.700696	
min	0.000000	0.000000	1.510000	1.000000	1.000000	2.000000	-9.000000	-19.000000	
25%	2500.000000	0.000000	7.150000	3.000000	17.000000	2.000000	56.000000	2.000000	
50%	5000.000000	0.000000	11.990000	5.000000	97.000000	3.000000	187.000000	39.000000	
75%	7500.000000	0.000000	22.860000	7.000000	1226.000000	3.000000	478.000000	122.000000	
max	10000.000000	1.000000	208.930000	7.000000	3242.000000	83.000000	7620.000000	2151.000000	

4. Is there any outlier in the data set? If so, how to deal with it? What are the methods to deal with outliers?

As we can see in the box-and-whisker plot, there is so many outliners of each variables in this data set.

To deal with this common problem, there are some following methods:

- We can use a log transform to transform the data (ideally to more Gaussian distribution). Then we determine a min and max value to identify which values are outliners.
- Z-score: This method relies on the mean and standard deviation of a group of data
- IQR: the benefit of using the interquartile range method is that it uses a robust measure of dispersion.

In [34]:
```
def remove_outliers_iqr(df, col_name):
    quartile_1 = df[col_name].quantile(0.25)
    quartile_3 = df[col_name].quantile(0.75)
    iqr = quartile_3 - quartile_1
    lower_bound = quartile_1 - (iqr*1.5)
    upper_bound = quartile_3 + (iqr*1.5)
    df= df.loc[(df[col_name] < upper_bound) & (df[col_name] > lower_bound)]
    return df
```

In [35]:
```
remove_outliers_iqr(df, 'payment_7_day')
remove_outliers_iqr(df, 'dau_days')
remove_outliers_iqr(df, 'days_between_install_first_pay')
remove_outliers_iqr(df, 'total_txns_7_day')
remove_outliers_iqr(df, 'total_page_views')
remove_outliers_iqr(df, 'total_product_liked')
remove_outliers_iqr(df, 'product_like_rate')
remove_outliers_iqr(df, 'total_free_coupon_got')
remove_outliers_iqr(df, 'total_bonus_xp_points')
```

9989	0	3.99	7	392	2	288	56	0.194444
9990	0	17.99	7	38	3	745	331	0.444295
9991	0	55.99	7	60	2	431	115	0.266821
9993	0	9.24	1	2673	3	14	-10	-0.714286
9994	0	22.98	4	26	4	85	18	0.211765
9995	0	11.99	7	251	3	880	188	0.213636
9996	0	23.99	5	20	2	636	165	0.259434
9997	0	9.99	6	279	2	190	28	0.147368
9998	0	9.99	7	896	3	441	98	0.222222
9999	0	16.50	7	206	3	1318	317	0.240516
10000	0	13.00	7	857	2	355	85	0.239437

9772 rows × 10 columns

- Normalization:

```
In [78]: from sklearn import preprocessing
         normalizer = preprocessing.Normalizer().fit(df[training_features])

In [79]: df_transform = normalizer.transform(df[training_features])
         df_transform

Out[79]: array([[  1.25333330e-05,   4.70588224e-06,   6.27450965e-06, ...,
                    2.80112038e-07,   7.05882336e-06,   9.99999975e-01],
                 [  6.29967989e-06,   5.20020745e-06,   2.76353882e-04, ...,
                    1.81117025e-07,   1.11433017e-05,   9.99999893e-01],
                 [  4.04215763e-06,   8.10747949e-06,   5.08454785e-04, ...,
                    3.47003799e-07,   1.15821136e-05,   9.99999685e-01],
                 ...,
                 [  9.70279259e-06,   6.79875357e-06,   8.70240457e-04, ...,
                    2.15833447e-07,   1.65112587e-05,   9.99999525e-01],
                 [  4.16664168e-05,   1.76766617e-05,   5.20198901e-04, ...,
                    6.07359826e-07,   2.02018991e-05,   9.99994004e-01],
                 [  1.94319692e-05,   1.04633680e-05,   1.28101520e-03, ...,
                    3.57901925e-07,   1.49476686e-05,   9.99999030e-01]])
```

5. Show how to build the classifier and how you will evaluate the results? What is the metrics you will use?

There are five steps to build the binary classifier.

- Step 1: Collect the data
- Step 2: Exploring and preparing data (create training and test data)
- Step 3: Training a model on data
- Step 4: Evaluate the model performance
- Step 5: Improve the model performance

The most common metrics to evaluate the model performance is the accuracy of the model predictions. We evaluate the results by confusion table, which includes:

- True Positive (TP): Correctly classified as the class of interest
- True Negative (TN): Correctly classified as not the class of interest
- False Positive (FP): Incorrectly classified as the class of interest
- False Negative (FN): Incorrectly classified as not the class of interest

$$\textbf{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Furthermore, we also use sensitivity and specificity to evaluate the model performance.

- Sensitivity of model (true positive rate) measures the proportion of positive examples that were correctly classified

$$\textbf{Sensitivity} = \frac{TP}{TP+FN}$$

- Specificity of model (true negative rate) measures the proportion of negative examples that were correctly classified

$$\textbf{Specificity} = \frac{TN}{TN+FP}$$

Beyond the accuracy, we also use Precision and Recall to evaluate the model performance:

- Precision (positive predictive rate) is defined as the proportion of positive examples that are truly positive.

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

- Recall is a measure of hoe complete the results are. It is defined as the number of true positives over the total number of positives. A model with high recall captures a large portion of the positive examples, meaning that it has wide breadth.

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

6. What algorithms you plan to use and why?

I am going to use k-NN method and Logistic Regression model to build binary classifier because the provided data is categorized as supervised learning. In addition, both of the algorithms are suitable for numeric variables (features).

- **k-NN**

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning) or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice.

1. Nearest Neighbors Classification (k-NN)

```
In [24]: # split into train and test
         X_train, X_test, y_train, y_test = train_test_split(df[training_features], df[target], test_size=0.25, random_state=1)

In [25]: from sklearn.neighbors import KNeighborsClassifier
         # instantiate learning model (k = 9)
         neigh = KNeighborsClassifier(n_neighbors=9)

In [26]: # fitting the model
         neigh.fit(X_train, y_train)

Out[26]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=9, p=2,
                    weights='uniform')

In [28]: # predict the response
         pred = neigh.predict(X_test)
         print(neigh.predict(X_test))

         [0 0 0 ..., 0 0 0]
```

- **Logistic Classifiers**

Logistic regression algorithm is used for the binary classification. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variables. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. The calculated probabilities used to find the target class. In general, the high probability class treated as the final target class.

7. Compare the results of the algorithms. Which is better, why?

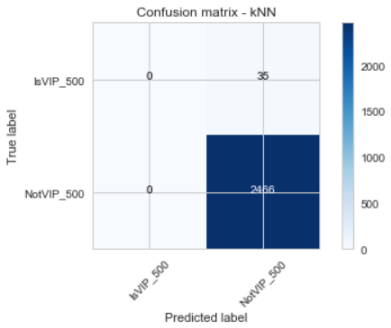
The accuracy rate of k-NN is slightly higher than that of Logistic Regression model. Meanwhile, the recall of Logistic Regression model is much better than k-NN. The accuracy rate of k-NN is 98.4%, but the recall is zero. It means that k-NN model fails to predict the VIP customers that we

are interested in. The recall of Logistic Regression model is 2.56%, which means this model can predict more accurately the potential VIP customers. Therefore, Logistic Regression model is better than k-NN in general.

- **k-NN:**

```
In [32]: # evaluate accuracy
from sklearn.metrics import confusion_matrix
con_mat = confusion_matrix(y_test, pred, labels = [1,0])
plt.figure()
plot_confusion_matrix(con_mat, classes=("IsVIP_500", "NotVIP_500"),
                      title="Confusion matrix - kNN")
plt.show()
```

```
Confusion matrix, without normalization
[[   0   35]
 [   0 2466]]
```



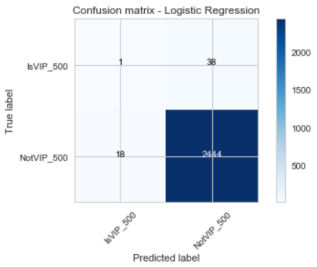

```
In [33]: #accuracy
print('Accuracy')
print(neigh.score(X_train, y_train))
#recall
print('Recall')
print (recall_score(y_test, pred))
```

```
Accuracy
0.984
Recall
0.0
```

- **Logistic Regression**

```
In [42]: con_matrix3 = confusion_matrix(y_test3, predicted, labels = [1,0])
plt.figure()
plot_confusion_matrix(con_matrix3, classes=("IsVIP_500", "NotVIP_500"),
                      title="Confusion matrix - Logistic Regression")
plt.show()
```

```
Confusion matrix, without normalization
[[   1   38]
 [  18 2444]]
```




```
In [41]: #accuracy
print('Accuracy')
print(model.score(X_train3, y_train3))
#recall
print('Recall')
print (recall_score(y_test3, predicted))
```

```
Accuracy
0.980933333333
Recall
0.025641025641
```

8. Given the current model, how would you improve the results? The data is imbalanced, please give a few methods that the academic will use for imbalanced data, pick one to experiment with your current dataset and see how the result will change.

Given the current models of k-NN and Logistic Regression, we could improve the model performance by feature scaling or normalization. In addition, the common method to improve model in k-NN is to try different value of k to get the most accurate performance.

However, the given dataset is imbalanced because the majority and minority class instances have huge differences:

- Total observations = 10,001
- VIP Observations = 155
- Not VIP observations = 9,846

Hence, dealing with imbalanced data might be the best solution to improve the accuracy. There are a few methods that the academic will use for imbalanced data as follow:

- **Random Under-sampling**: aims to balance class distribution by randomly eliminating majority class examples.
- **Random Over-sampling**: increase the number of instances in the minority class by randomly replicating them to obtain a higher representation of minority class.
- **Cluster-Based over sampling**: Apply K-means clustering algorithm.
- **Synthetic Minority Oversampling Technique (SMOTE)**: SMOTE produces synthetic minority class samples by selecting some of the nearest minority neighbors of a minority sample which is named S and generates new minority class samples along the lines between S and each nearest minority neighbor.

I would like to apply SMOTE to the Logistic Regression model:

Dealing with imbalanced data - SMOTE

In [43]:
```
from sklearn.datasets import make_classification
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE, ADASYN
```

In [44]:
```
X_train4, X_val, y_train4, y_val = train_test_split(df[training_features], df[target], test_size=0.25, random_state=12)
```

In [45]:
```
sm = SMOTE(random_state=12, ratio = 1.0)
x_train_res, y_train_res = sm.fit_sample(X_train4, y_train4)
```

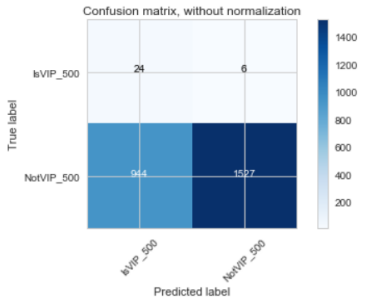
```
C:\Users\Trang Nguyen\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:77: DeprecationWarning: Function _ratio_float is
deprecated; Use a float for 'ratio' is deprecated from version 0.2. The support will be removed in 0.4. Use a dict, str, or a c
allable instead.
  warnings.warn(msg, category=DeprecationWarning)
```

In [46]:
```
#Import Library LOGISTIC REGRESSION
from sklearn.linear_model import LogisticRegression

# Create logistic regression object
model4 = LogisticRegression()

# Train the model using the training sets
model4.fit(x_train_res, y_train_res)

#Predict Output
predicted= model4.predict(X_val)
```

Then, we check the model performance by creating confusion matrix and calculating the accuracy rate and recall. As we can see below, the overall accuracy rate of Logistic Regression model (74.14%) is lower than the previous models, but still acceptable. Importantly, the recall has been significantly improved from 2.56% to 80%. This improvement is crucially important because we can accurately predict the potential customers for the company and have appropriate strategies to enhance Customer Relationship Management (CRM) system.

In [48]:
```
con_matrix4 = confusion_matrix(y_val, predicted, labels = [1,0])
plt.figure()
plot_confusion_matrix(con_matrix4, classes=("IsVIP_500", "NotVIP_500"),
                      title="Confusion matrix, without normalization")
plt.show()
```

```
Confusion matrix, without normalization
[[   24     6]
 [  944 1527]]
```



In [50]:
```
print ('Accuracy')
print (model4.score(x_train_res, y_train_res))
print('Recall')
print (recall_score(y_val, predicted))
```

```
Accuracy
0.741423728814
Recall
0.8
```