Santander Customer Transaction Prediction

SHIVAM MISHRA

17 September 2019

Contents

1. INTRODUCTION

1.1 Background

At, Santander, the mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals.

Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

1.2 Problem Statement

In this challenge, we need to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

1.2 Data

Table 1.1: Sample Data (Columns: 1-7)

ID_code	target	var_0	var_1	var_2	var_3	var_4
train_0	0	8.9255	-6.7863	11.9081	5.093	11.4607
train_1	0	11.5006	-4.1473	13.8588	5.389	12.3622
train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825
train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846
train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772

Table 1.2: Sample Data (Columns: 196-202)

var_193	var_194	var_195	var_196	var_197	var_198	var_199
1.691	18.5227	-2.3978	7.8784	8.5635	12.7803	-1.0914
10.9516	15.4305	2.0339	8.1267	8.7889	18.356	1.9518
1.6858	21.6042	3.1417	-6.5213	8.2675	14.7222	0.3965
1.4214	23.0347	-1.2706	-2.9275	10.2922	17.9697	-8.9996
9.1942	13.2876	-1.5121	3.9267	9.5031	17.9974	-8.8104

```
In [134]: train=pd.read_csv("train.csv")

In [6]: train.shape
Out[6]: (200000, 202)
```

Look at the shape of the data. There are 202 variables in the train dataset and 200000 observations.

The details of dataset attributes are as follows -

- ID_code – ID of the data starting from train_0 to train_199999.
- target- Object variable indicating if a customer will make a transaction if the target value is zero then customer will not make transaction and if it is one then customer will make a transaction.
- var_0 to var_199 – All these are float variable and Independent. These features will be used to predict the target variable of the test dataset.

Now let's have a look at the data type of train attributes.

```
ID_code     object
target       int64
var_0      float64
var_1      float64
var_2      float64
            ...
var_195    float64
var_196    float64
var_197    float64
var_198    float64
var_199    float64
Length: 202, dtype: object
```

Here, the datatype of target attribute is int which is actually object. Datatype of ID_code is object. Simillarly datatype of vaiables from var_0 to var_199 is float.

2. Methodology

2.1 Pre Processing

Before developing any model we first need to look into the data. By, saying look into the data, I mean to explore the data. Look at the datatypes of attributes, find the minimum and maximum values of variables compare it with its mean value. Convert the required datatypes. Finding and imputing missing values using various methods like mean, median, etc. This is nothing but data preprocessing where we analyze the data, clean the data and transform the data. Data preprocessing is the probably most or one of the most important things in model development. So, it needs to be taken care of.

2.1.1 Missing Value Analysis

Missing value analysis is a method or technique to find out if there are missing values in the attributes of the dataset. When we applied missing value analysis on our dataset we found out that there were no missing values in our dataset.

- is.null( ).sum() in python
- function(x){sum(is.na(x))} in R

We got the following result after applying missing value analysis on our dataset

Missing value analysis

```
In [11]: train.isnull().sum().sum()
Out[11]: 0
```

as we can see that there are no missing values in the data

Now, you can see that there were no missing values.

2.1.2 Outlier Analysis

In statistics, an outlier is defined as a data point that differs significantly from other observations. Outlier analysis is a technique to find these points. Outlier analysis can only be done on a numerical variable.
Causes of Outliers

- Poor data quality/contamination
- Low-quality measurements, malfunctioning equipment, manual error
- Correct but exceptional data

Let's look at the summary of the data and try to find out if the outlier exists in our dataset.

```
train.describe()
```

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
mean	0.100490	10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	5.408949	16.545850
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	3.418076
min	0.000000	0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	5.349700
25%	0.000000	8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	13.943800
50%	0.000000	10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	16.456800
75%	0.000000	12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	19.102900
max	1.000000	20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	27.691800

8 rows × 201 columns

This is a summary of the first 9 variables. Look at the summary of var_0, you can see that it has mean value as 3.04 and max value is 20.31. That means max value is not differing with mean value by very large value. So, by these details we can say that there are no outliers in var_0 variable. Similarly, you can look at the other variables and find out that there are no outliers.

```
train.describe()
```

var_190	var_191	var_192	var_193	var_194	var_195	var_196	var_197	var_198	var_199
00000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
3.234440	7.438408	1.927839	3.331774	17.993784	-0.142088	2.303335	8.908158	15.870720	-3.326537
4.559922	3.023272	1.478423	3.992030	3.135162	1.429372	5.454369	0.921625	3.010945	10.438015
-14.093300	-2.691700	-3.814500	-11.783400	8.694400	-5.261000	-14.209600	5.960600	6.299300	-38.852800
-0.058825	5.157400	0.889775	0.584600	15.629800	-1.170700	-1.946925	8.252800	13.829700	-11.208475
3.203600	7.347750	1.901300	3.396350	17.957950	-0.172700	2.408900	8.888200	15.934050	-2.819550
6.406200	9.512525	2.949500	6.205800	20.396525	0.829600	6.556725	9.593300	18.064725	4.836800
18.440900	16.716500	8.402400	18.281800	27.928800	4.272900	18.321500	12.000400	26.079100	28.500700

Similarly, you can see last 10 variables also do not contain outliers. For example, look at the var_199. It has mean value as 10.438015 and the max value is 28.500 that means they are not differing by large margin hence it does not contain outliers.

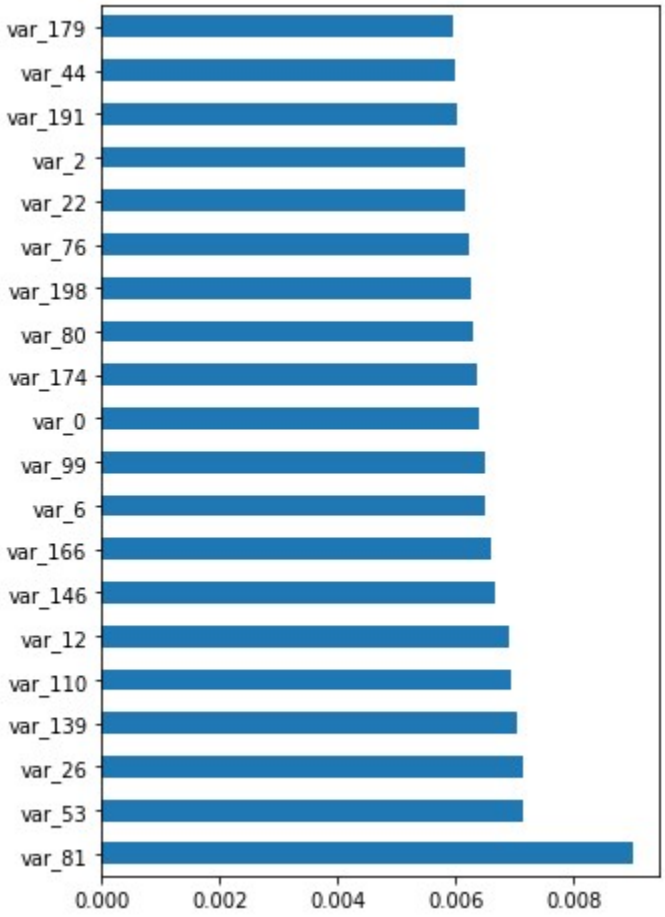
2.1.3 Feature Importance

Feature Importance is used to find out what features are contributing the most in order to predict the result. To find out what features are most important we used “ExtraTreesClassifier”.
We used the below code to find out important features.

```
X = train.iloc[:,2:202]  #independent columns
y = train.iloc[:,1]    #target column i.e price range
model = ExtraTreesClassifier(random_state=42)
model.fit(X,y)
print(model.feature_importances_)
#plot graph of feature importances for better visualization
plt.subplots(figsize=(40,45))
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(200).plot(kind='barh')
plt.show()
```

Here, X is the variable that contains all the independent variables and y is also a variable that contains dependent variable i.e target.

Let’s look at the top 20 features that we got from the above code.



You can see that var_81,var_53,var_26, and var_139 are some of the most important features.

2.1.4 Feature Selection

Feature selection is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. In machine learning and statistics feature selection is also known as variable selection, attribute selection or variable subset selection. In our dataset ID_code which is an object, this variable is of no use hence we dropped this column from our train and test dataset. And selected the rest of the features.

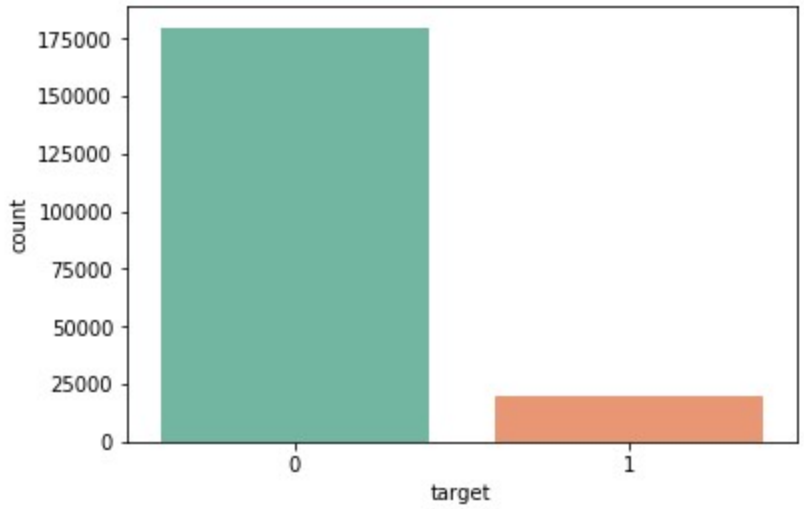
2.1.4 Handle Imbalanced data

If you will look at the count of the target variable then you will find out that number of zero's is much more than the number of ones. This is nothing but target class imbalance problem. Have a look at the count plot of the target variable.

```
Count
0   179902
1    20098
Name: target, dtype: int64
```

```
sns.countplot(train['target'],palette="Set2")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2dd5eeb320>
```



As you can see that there are 20098 occurrences of positive class i.e 1 and 179902 occurrences of negative class i.e 0. This clearly shows imbalance of data. The problem with imbalanced data is that model might be biased towards majority class which is zero in this case.
So, to handle imbalanced data there are several methods like undersampling, oversampling and synthetic data generation.

i. Undersampling:-

This method works with the majority class. It reduces observations from majority class in order to make dataset balanced.

But the problem with this model is that while reducing the observations from majority class we may lose the important data. Which is also called as underfitting.

ii. Oversampling:-

This method works with minority class. It replicates the observation of minority class to make balance between minority and majority class. But the problem with this method is that as we are replicating the data there will be many similar observations and will cause overfitting.

iii. Synthetic Data Generation:-

In simple words instead of replicating the same data, this method creates new artificial data.
SMOTE(synthetic minority oversampling technique) is a powerful and widely used method to create synthetic data.
So, we have used SMOTE to handle imbalance data.
Library of SMOTE

- in R- ROSE
- in python-SMOTE

```
sm = SMOTE(random_state=42)
X_train_smote,y_train_smote=sm.fit_sample(X_train,y_train)
X_test_smote,y_test_smote=sm.fit_sample(X_test,y_test)
print(X_train_smote.shape)
print(X_test_smote.shape)
```

2.2 Modeling

2.2.1 Model Selection

As we know that we need to predict whether a customer will make a transaction or not we can understand that this is a classification problem. So, we knew that we have to use classification models to predict the target variable. Hence we used the following regression models to predict the result.

1. In python
- Logistic Regression
- Decision Tree
- Lightgbm

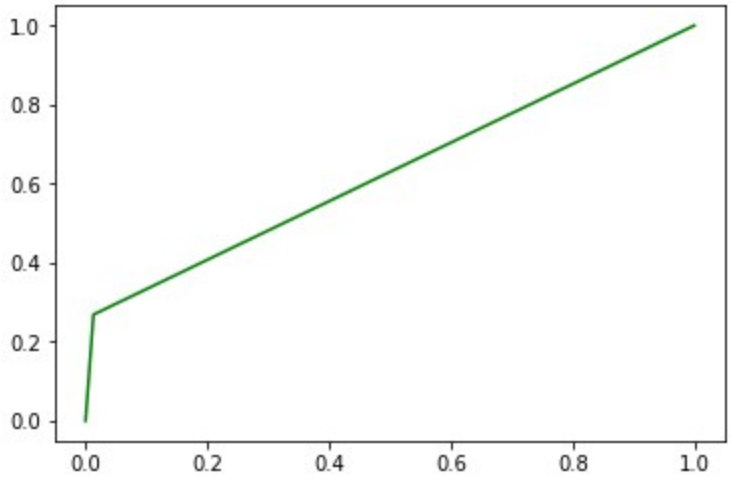
2. In R
- Logistic Regression
- Random Forest
- Lightgbm

2.2.2 Logistic Regression on imbalanced data

```
#Logistic Regression
logit=LogisticRegression(random_state=42)
logit_model=logit.fit(X_train,y_train)
logit_pred=logit_model.predict(X_test)
FPR,TPR,thresholds=roc_curve(y_test,logit_pred)
plt.plot(FPR,TPR,'g')
```

Roc curve

```
[<matplotlib.lines.Line2D at 0x7f2db25f18d0>]
```



Classification Report

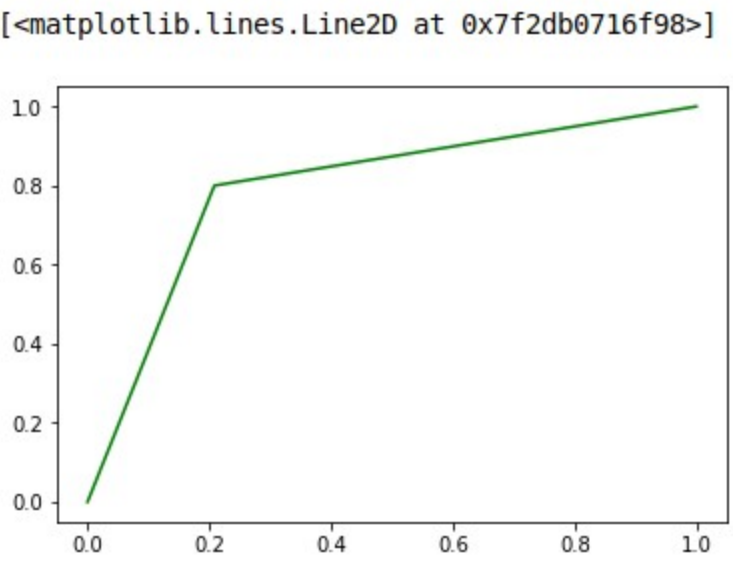
```
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     35980
           1       0.70      0.27      0.39      4020

    accuracy                           0.91     40000
   macro avg       0.81      0.63      0.67     40000
weighted avg       0.90      0.91      0.90     40000
```

Look at the ROC curve and classification report. Even though the model is giving accuracy of 91% but has very low recall rate for positive class. That means number of times positive class was predicted right is very less.

Logistic Regression on synthetic data

```
smote=LogisticRegression(random_state=42)
smote_model=smote.fit(X_train_smote,y_train_smote)
smote_pred=smote_model.predict(X_test_smote)
FPR,TPR,thresholds=roc_curve(y_test_smote,smote_pred)
plt.plot(FPR,TPR,'g')
```

ROC curve



Classification Report

```
              precision    recall  f1-score   support

           0       0.80      0.79      0.79     35980
           1       0.79      0.80      0.80     35980

    accuracy                           0.80     71960
   macro avg       0.80      0.80      0.80     71960
weighted avg       0.80      0.80      0.80     71960
```

Now, look at the ROC curve and classification report. You can see that roc curve has improved a lot and recall value has also improved that means Logistic Regression is performing well on synthetic data.

2.2.3 Decision Tree

We have divided train data into 80% train and 20% test datasets for the decision tree model. Let's look at the decision tree model development code in python.

```
C50_model = tree.DecisionTreeClassifier(criterion='entropy').fit(X_train, y_train)
#predict new test cases
C50_Predictions = C50_model.predict(X_test)
```

Here, X_train is subset data from the train dataset for training and has all independent variables. Similarly, y_train is a training dataset with only the target variable.
X_test is test data that is a subset of the train dataset and has all the independent variables.

2.2.4 Random Forest

For Random Forest also we have divided train data into 80% train and 20% test datasets. Then we used ROSE in R to create new synthetic data as train_rose and test_rose. Let's look at the random forest model development code in R.

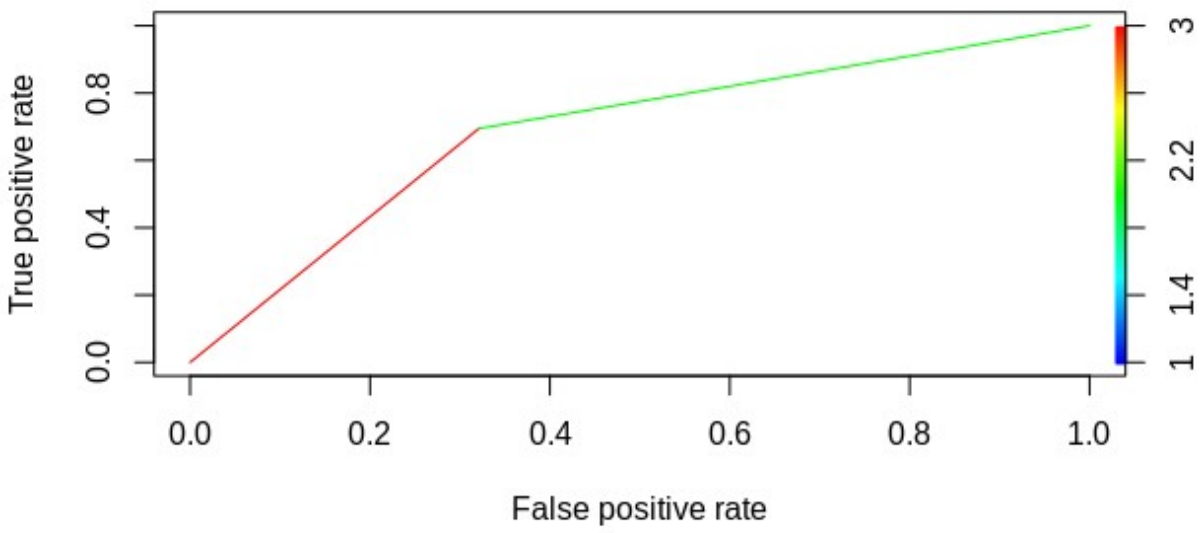
```
RF_model = randomForest(target ~ ., train_rose, importance = TRUE, ntree = 100,seed=2)
RF_Predictions = predict(RF_model, test_rose[,-1])
```

Here, train_rose is the new data that we created by using ROSE for training. Similarly, test_rose new test dataset is a training dataset.

Now let's look at how this model performed in R

```
Examples are labelled as positive when predicted is greater than 0.5

precision: 0.507
recall: 1.000
F: 0.337
```

Roc curve:



So, if you will look at the roc curve and precision, F-1 score you can say that random forest is not performing very good.

2.2.4 Lightgbm

In the lightgbm model, first of all, we need to set parameters in order to tune our model.

```
param = {'objective' : "binary",
         'boost':"gbdt",
         'metric':"auc",
         'boost_from_average':"false",
         'num_threads':8,
         'learning_rate' : 0.01,
```

```
        'num_leaves' : 13,
        'max_depth':-1,
        'tree_learner' : "serial",
        'feature_fraction' : 0.05,
        'bagging_freq' : 5,
        'bagging_fraction' : 0.4,
        'min_data_in_leaf' : 80,
        'min_sum_hessian_in_leaf' : 10.0,
        'verbosity' : 1}
```

So, these are the parameters we used to tune our lightgbm model.
Let's look at the lightgbm code.

```
num_rounds=20000
lgbm=
lgb.train(param,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_test],verbose_eval=400,early_stopping_rounds =
3000)
lgbm
```

Here, num_rounds is nothing but the number of iterations we want lightgbm to perform in order to get optimal results. lgb_train is train dataset and lgb_test is test dataset. verbose_eval will basically print the AUC score of train and test dataset for every 400^{th} iteration. early_stopping round will stop lightgbm model if it doesn't see any improvement in AUC score of the test dataset consistently for 3000 rounds.

3. Conclusion

3.1 Model Evaluation

Now that we have three models for predicting whether a customer will make a transaction or not, we need to decide which one to choose. There are several criteria that are used for evaluating and comparing models. We can compare our models using the following criteria:

1. AUC score
2. Precision
3. Recall

3.1.1 AUC score

AUC stands for the area under roc curve. That is AUC measures entire area under the roc curve. The value of AUC ranges from 0 to 1. A model whose predictions are 100 percent correct has an AUC score of 1 and model whose predictions are 100 percent wrong has AUC score of 0.

3.1.2 Precision

Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actually positive. Precision is a good measure to determine when the costs of False Positive is high. As in our case if we predicted that certain customer will make a transaction but in reality he is not going to make any transaction and we will lose all our expenditure on that customer. Precision is given by.

Precision= True positive/(True positive+False Positive)
= True positive/Total predicted positive

3.1.3 Recall

Recall actually calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative. For example, if we

predict that a customer will not make any transaction and if he may make a transaction then we might lose that customer. The recall is given by

Recall= True positive/(True positive+False Negative)
= True positive/Total actual positive

Performance Measures of the models.

Model	Auc Score	Report
Logistic Regression	79.20	precision recall f1-score 0 0.80 0.79 0.79 1 0.79 0.80 0.80 accuracy 0.80 macro avg 0.80 0.80 0.80 weighted avg 0.80 0.80 0.80
Decision Tree	64.58	precision recall f1-score 0 0.62 0.75 0.68 1 0.68 0.53 0.60 accuracy 0.64 macro avg 0.65 0.64 0.64 weighted avg 0.65 0.64 0.64
LightGBM	89	precision recall f1-score 0 0.86 0.93 0.89 1 0.92 0.85 0.89 accuracy 0.89 macro avg 0.89 0.89 0.89 weighted avg 0.89 0.89 0.89

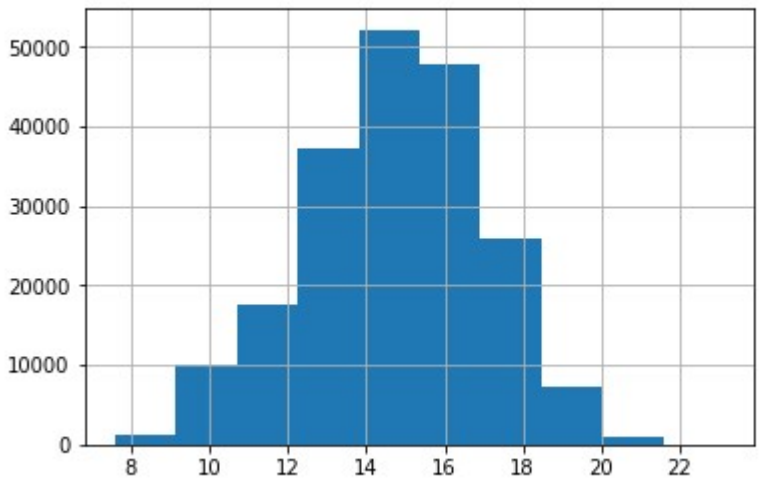
3.2 Model Selection

As we can see from the above table that LighGBM model is giving us the best AUC score. Similarly LightGBM is also giving best Precision and Recall score. So based on these result we can see that LightGBM outperforms other models and we selected this model for predicting target variables.

4. Visualizations

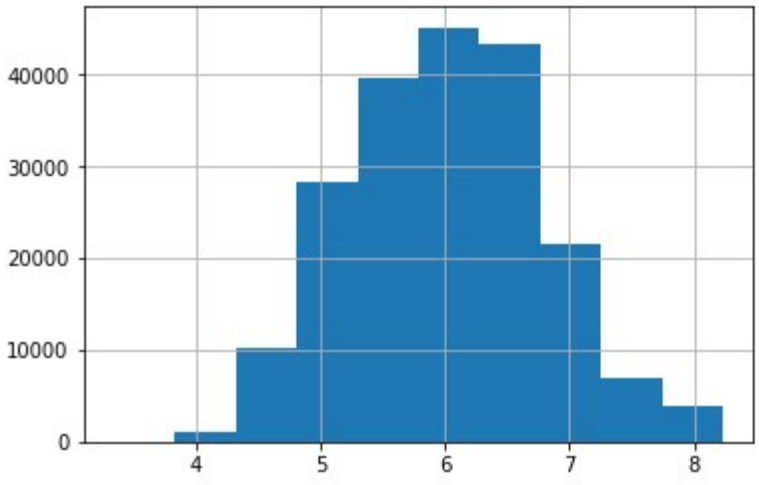
4.1 Histogram of the var_81

```
: <matplotlib.axes._subplots.AxesSubplot at 0x7f94e17c5e80>
```



4.2 Histogram of the var_53

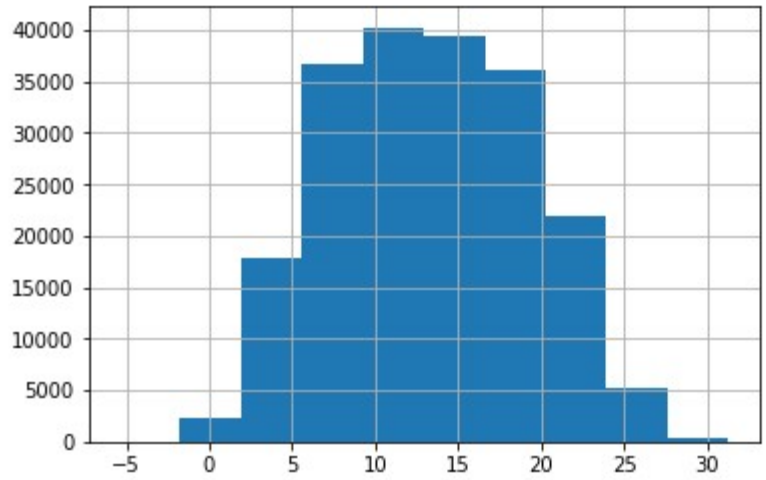
```
: <matplotlib.axes._subplots.AxesSubplot at 0x7f94e17d62e8>
```



As you can see that histogram of var_81 and var_53 is similar which are the most important variables. This indicates that distribution of data of important variables is similar.

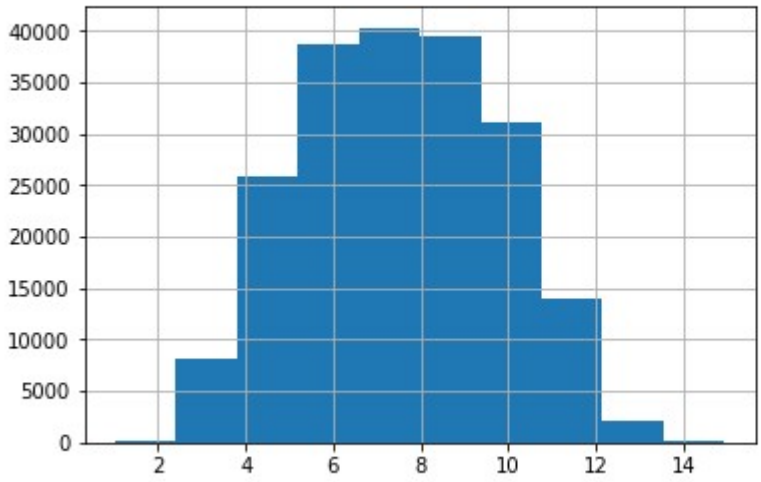
4.3 Histogram of the var_20

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x7f94ec2580b8>
```



4.4 Histogram of the var_14

```
: <matplotlib.axes._subplots.AxesSubplot at 0x7f94e2c01e80>
```



Similarly look at the histogram of var_20 and var_14 which are least important variables. You can see that the distribution of data of these variables is similar.