**CUSTOMER CHURN ANALYSIS PROJECT REPORT:**

**Problem Definition:**

In this article, we will go through some consumer data and see how we can leverage data insights and predictive modeling in order to improve customer retention. In our analysis, we will use Python and variety of Machine Learning algorithms for prediction. **The best way to avoid customer churn is to know your customers, and the best way to know your customer is through historical and new customer data. Our customer data from IBM Sample Data Sets with the aim of building and comparing several customer churn prediction models.**

**Data Analysis:**

We have to make sure we update the categorical variables to numerical variables as the machine learning techniques we will apply requires all customers attributes to be numerical. We can see our dataset but we also want to make sure the data is clean, so as part of the cleaning process, we look at missing values and data types.

The dataset has a lot of categorical variables rendered as text values (‘Yes’,’No’,etc.) in the columns. Converting the categorical values into numeric values to facilitate ML algorithms to process the data. Also, converting Total Charges to a numerical data type.

df.TotalCharges = pd.to\_numeric(df.TotalCharges, errors='coerce')

df.isnull().sum()

From the above output, it can be seen that, there are 11 missing values for Total Charges. Replacing 11 rows from the data set. Also predictor variable here is Churn. Therefore, it is necessary to convert the predictor variable in binary numeric variable.

**Explanatory data analysis:**

The explanatory approach is a method to make people understand something by describing or illustrating. In this stage we able to understand and finds the overview of the data headings, data shapes, data types, unique values, missing values, perform basic statistics, univariate and bivariate anaysis , missing value treatment and outliers treatment.

**Data preprocessing:**

Data as such can't be used. We'll transform the data so that we can feed it to a machine learning algorithm

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import roc\_auc\_score

from sklearn.model\_selection import GridSearchCV

### Methodology:

Our first step was to split our data into training and test sets using train-test-split, which would allow us to cross-validate our results later. We also stratified the train-test-split, to ensure that the same proportion of our target variable was found in both our training and test sets. It is important to test a predictor on data held-out from training, preprocessing (such as standardization, feature selection, etc.)

### Splitting the data:

X is the data with the independent variables, Y is the data with the dependent variable. The test size variable determines in which ratio the data will be split. It is quite common to do this in a 80 Training / 20 Test ratio. Also ned to stratify the train-test-split to have a balanced split.

**Building Machine Learning Models:**

**Predictive modeling:**

We will consider several different models to predict customer churn. To ensure we are not over-fitting to our data, we will split the 7,043 customer records into a training and test set, with the test set being 20% of the total records. It is important to scale the variables in regression so that all of them are within a range of 0 to 1.

Logistic regression requires there to be little or no multicollinearity among the independent variables. Therefore, as discussed in the begining, Total Charges will be removed.

df\_dummies = df\_dummies.drop("TotalCharges", axis=1) *# removing Total Charges to avoid multicolinearity.*

*# Using the data frame where we had created dummy variables*

y = df\_dummies['Churn'].values

X = df\_dummies.drop(columns = ['Churn'])

*# Scaling all the variables to a range of 0 to 1*

**from** **sklearn.preprocessing** **import** MinMaxScaler

features = X.columns.values

scaler = MinMaxScaler(feature\_range = (0,1))

scaler.fit(X)

X = pd.DataFrame(scaler.transform(X))

X.columns = features

**from** **sklearn.model\_selection** **import**

**Logistic regression:**

Using machine learning algorithm and the dependent variable here churn 1 or churn 0 is categorical. The trained model can be used to predict if a customer churned or not for the test dataset. The results are saved in “prediction\_test” and afterwards the accuracy score is measured and printed.

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.metrics** **import** confusion\_matrix, accuracy\_score

**from** **sklearn** **import** metrics

lr\_model = LogisticRegression(solver='lbfgs').fit(X\_train, y\_train)

*#result = lr\_model.fit(X\_train, y\_train)*

lr\_prediction = lr\_model.predict\_proba(X\_test)

lr\_model.score(X\_test, y\_test)

0.8224431818181818

*#Accuracy*

Accuracy = (502+77)\*100/(502+55+70+77)

print("Accuracy **{:0.2f}**%:",format(Accuracy))

Accuracy {:0.2f}%: 82.24431818181819

The classifier is predicting a 1 (having a Low Birth Weight child) any time the probability in the second column is greater than 0.5. The positive class probability is returned by the models in the second column (index=1):

**Calibration Plot:**

Once I have the class probabilities and labels, I can compute the bins for a calibration plot. Here I use sklearn.calibration.calibration\_curve that returns the (x,y) coordinates of the bins on the calibration plot.

**from** **sklearn.calibration** **import** calibration\_curve

lr\_y, lr\_x = calibration\_curve(y\_test, lr\_prediction[:,1], n\_bins=20)

Note that although I asked for 20 bins for logistic regression, 2 bins out of 20 don’t have any data. The reason is a combination of that logistic regresion being a simple model, that there are only two features, and that I have 704 points of data in the validation set.

**Cross-validation:**

Cross validation ensures all samples will appear in the training and test sets, so 100% of your data gets used at some point for training and for testing. Use 5-fold cross-validation instead, and take the average of all of my data to make the calibration plot.

**from** **sklearn.linear\_model** **import** LogisticRegressionCV

clf = LogisticRegressionCV(cv=5, random\_state=1234, solver='lbfgs').fit(X\_train, y\_train)

clf\_prediction = clf.predict\_proba(X\_test)

clf.score(X\_test, y\_test)

0.8238636363636364

**from** **sklearn.model\_selection** **import** cross\_val\_score

accuracies = cross\_val\_score(estimator=clf, X=X\_train, y=y\_train, cv=5)

print(accuracies.mean())

0.7989887667898536

print(accuracies.std())

0.003849477406942357

The result is: 0.003 which is 0.3%. This is extremely low, which means that our model has a very low variance, which is actually very good since that means that the prediction that we obtained on one test set is not by chance. Rather, the model will perform more or less similar on all test sets.

By default random forest picks up 2/3rd data for training and rest for testing for regression and almost 70% data for training and rest for testing during classification.By principle since it randomizes the variable selection during each tree split it's not prone to overfit unlike other models.

rlf = RandomForestClassifier(random\_state=101, n\_estimators=100).fit(X\_train, y\_train)

rlf\_prediction = rlf.predict\_proba(X\_test)

rlf.score(X\_test, y\_test)

0.7798295454545454

**from** **sklearn.model\_selection** **import** cross\_val\_score

all\_accuracies = cross\_val\_score(estimator=rlf, X=X\_train, y=y\_train, cv=5)

print(all\_accuracies.mean())

0.7827107976888315

print(all\_accuracies.std())

0.004729220104909373

**Concluding Remarks:**

The result is: 0.004 which is 0.4%. This is extremely low, which means that our model has a very low variance, which is actually very good since that means that the prediction that we obtained on one test set is not by chance. Rather, the model will perform more or less similar on all test sets.

clf\_y, clf\_x = calibration\_curve(y\_test, clf\_prediction[:,1], n\_bins=20)

rlf\_y, rlf\_x = calibration\_curve(y\_test, rlf\_prediction[:,1], n\_bins=20)

fig, ax = plt.subplots()

*# only these two lines are calibration curves*

plt.plot(clf\_x,clf\_y, marker='o', linewidth=1, label='lr')

plt.plot(rlf\_x, rlf\_y, marker='o', linewidth=1, label='rf')

*# reference line, legends, and axis labels*

line = mlines.Line2D([0, 1], [0, 1], color='black')

transform = ax.transAxes

line.set\_transform(transform)

ax.add\_line(line)

fig.suptitle('Calibration plot for Telecom data')

ax.set\_xlabel('Predicted probability')

ax.set\_ylabel('True probability in each bin')

plt.legend()

plt.show()

bin\_total(y\_test, clf\_prediction[:,1], n\_bins=20)

array([198, 85, 45, 59, 43, 33, 29, 34, 22, 25, 24, 19, 23,

19, 27, 12, 6, 1, 0, 0, 0])

bin\_total(y\_test, rlf\_prediction[:,1], n\_bins=20)

array([213, 70, 59, 47, 39, 42, 27, 27, 22, 18, 26, 21, 22,

18, 7, 12, 10, 6, 7, 11, 0])

I think it’s safe to say that, in this example, the random forest is better calibrated than logistic regression.