Bharti AXA Assignment

Shrikrishna Padalkar

August 2021

1 Introduction

A travel insurance is a policy purchased by travelers to cover any loss arising while travelling including but not limited to baggage claim, delay in flight, accident suffered etc. Instead of paying out of pocket for travel related accidents, people pay annual premiums to a travel insurance company; the company then pays all or most of the costs associated with the travel accident or any other loss suffered.

The objective is to identify fraud claims made by the insured.

2 Procedure

- Data Description
- Target Distribution
- Missing Value Analysis
- Descriptive Stats
- Independent Variable Distribution
- Variable Pre-processing
- Variable Encoding
- Modelling
- Model Evaluation

Data	Dimensions	
Train Data	(52,310 X 11)	
Test Data	(22,421 X 10)	

Table 1: Data Distribution

2.1 Data Distribution

The train data will be used for model building process and the test data will be used for model evaluation. The table below shows the dimensions of the data. The train and test data comprise of 10 independent variables. **Claim** is the target variable which is a dichotomous variable where 1 represents **Fraudulent** claims and 0 represents **Genuine** claims.
Variable Types

- Continuous Variables The continuous variables are 'Duration', 'Net Sales', 'Commision (in value)', 'Age'.
- Categorical Variables The categorical variables are 'Product Name', 'Agency', 'Agency Type', 'Distribution Channel', 'Destination'.

2.2 Target Distribution

The **Claim** variable has the following distribution. Around 83% of the claims are genuine claims whereas around 17% of the claims are fraudulent. There is a class imbalance but not too much imbalance. **CODE:-**

```
def get_frequency_dist(cat_var, data=train_data):
    count = dict(Counter(data[cat_var]))

    count_df = pd.DataFrame({'Categories': list(count.keys()),\
                             'Counts': count.values()},\
                            columns=['Categories', 'Counts'])

    #Sort in descending order of count
    count_df = count_df.sort_values(by='Counts', ascending=False)

    #Obtain the % Counts
    total_count = sum(count_df['Counts'])
    count_df['Counts_%'] = count_df['Counts']*100/total_count #
                                   Broadcasting

    return count_df

#Target Event rate of train data
print("Train Data Target Distribution:- \n{}\n" .format(
                                get_frequency_dist('Claim')))
```

Class	Counts	% Counts
0	43,590	83.33
1	8,720	16.66

Table 2: Target Distribution

2.3 Missing Value Analysis

The variables having more than 30 per. missing values should not been considered. For the variables having ¡ 30% missing values, choose KNN imputation for imputing categorical variables and Median imputation for imputing continuous variables. The median value is not affected by outliers. A package named UtilityFunctions has been created containing some frequently required functions. The function for obtaining the missing value stats is present in the package. **CODE:-**

```
def find_missing_values(self):
        features = list(self.data.columns)
        missing_value_count = self.data.isnull().sum()
        missing_value_percentage = missing_value_count * 100/self.
                                   data.shape[0]

        missing_value_df = pd.DataFrame({'Features': features,

                                  'Missing Value Count':
                                  missing_value_count,

                             'Missing Value Percentage':
                             missing_value_percentage},

                                  columns = ['Features',
                                  'Missing Value Count',
                                  'Missing Value Percentage'
                                  ])

        #Sort in descending order of missing value count
        missing_value_df = missing_value_df.sort_values(
                                  by='Missing Value Count',
                                  ascending=False)

        return missing_value_df
```

2.4 Descriptive Stats

The descriptive statistics of each variable is shown below.

Features	Missing Count	% Missing	Data Types	Unique Categories
Agency	0	0	Categorical	16
Agency Type	0	0	Categorical	2
Distribution Channel	0	0	Categorical	2
Product Name	0	0	Categorical	25
Destination	0	0	Categorical	97

Table 3: Categorical Stats

Features	Missing Count	% Missing	Data Types	count	mean
ID	0	0	Integer	52310	6005.75
Duration	0	0	Integer	52310	58.26
Age	0	0	Integer	52310	39.56
Net Sales	0	0	Integer	52310	48.55

Table 4: Continuous Stats

2.5 Independent Variable Distribution

Categorical Variables Please refer the file **WOE_IV.xlsx** The variables Destination, Agency, Product Name have many categories. The number of categories have been reduced by clubbing the categories having 5% or less counts into **OTHERS**. A new grouped variable is created for each of these categorical variables.

CODE:-

```
class GroupingCatVars():
    '''
    Group some of the categories belonging to
    Destination, Agency and Product Name to "OTHERS"
    '''
    def __init__(self, data):
        self.data = data
        self.grouped_vars = [] #Store the newly created grouped
                                          variable

    def tag_others(self, cat_var, others_cat, majority_cat):
        '''
        Create a new grouped variable obtained vai
        clubbing the less frequent categories.
        If a new category appears in the unseen data then
        this category is tagged as "OTHERS"
        Input:-
        cat_var:- The categorical variable whose categories are to
                                          be grouped
        others_cat:- A list of categories that are to be tagged as
                                          "OTHERS"
        majority_cat:- A list comprising of categories that are
                                          present in Majority
                                          covering 95\% of the data
        '''
        self.data[cat_var + "_grouped"] = list(map(lambda cat: "
                                          OTHERS" if((cat

                                          in others_cat)
                                                   #Unseen category
                           or ((cat not in others_cat)
                           and (cat not in majority_cat))
                           else cat, self.data[cat_var]))

        #Update the list of grouped variables
        self.grouped_vars.append(cat_var + "_grouped")

    def driver(self):
        grouping_vars = ['Destination', 'Agency', 'Product Name']
        others_lists = [others_destination, others_agency,
                                          others_product_name]
        majority_lists = [majority_destination, majority_agency,
                                          majority_product_name]

        for var, others_list, majority_list in \
        zip(grouping_vars, others_lists, majority_lists):
            self.tag_others(var, others_list, majority_list)

        #Return the updated data with the grouped variables
        return self.grouped_vars
```

2.6 Variable Pre-Processing

WOE IV (Variable Importance) Please refer the file **WOE_IV.xlsx**. Weight of evidence is a technique used to check the relationship of the independent variable with the Categorical Target variable. IV(Information Value) determines how useful the variable is for predicting the target.

The code for computing WOE & IV is shown below. **CODE:-**

```python
def create_df(column_list, value_list):
    '''
    Input
    column_list:- Contains a list of column names for the data
                                      frame
    value_list:- It's a list of lists containing the set of values
                                      for each column
    '''
    df = pd.DataFrame(columns=column_list)
    for col, values in zip(column_list, value_list):
        df[col] = values

    return df
```

```python
def caluclate_woe_iv(woe_iv_df):
    woe_cat_wise = [] #Stores the woe calculations for each
                                      category

    total_events = sum(woe_iv_df["Event Count"])
    total_non_events = sum(woe_iv_df["Non Event Count"])

    woe_iv_df["Total Counts"] = woe_iv_df["Event Count"] +
                                      woe_iv_df["Non Event Count"]
    woe_iv_df["Event %"] = woe_iv_df["Event Count"]/total_events
    woe_iv_df["Non Event %"] = woe_iv_df["Non Event Count"]/
                                      total_non_events

    woe_iv_df["Event Rate"] = woe_iv_df["Event Count"]/woe_iv_df["
                                      Total Counts"]

    woe_cat_wise = [woe_sanity(ind, woe_iv_df, total_events,
                                      total_non_events) \
                    for ind in woe_iv_df.index]

    woe_iv_df["WOE"] = woe_cat_wise

    woe_iv_df["IV"] = (woe_iv_df["Non Event %"] - woe_iv_df["Event
                                      %"])*woe_iv_df["WOE"]

    return woe_iv_df
```

```
def woe_iv(cat_var, data, events='Good', non_events='Bad', target='
                                  Listing_Type'):
    data_grp = data[[cat_var, target]].groupby(cat_var)
    event_counts, non_event_counts = [], []
    categories = []

    for grp, df in data_grp:
        categories.append(grp)
        event_count = df.loc[df[target]==events, :].shape[0]
        non_event_count = df.shape[0] - event_count
        event_counts.append(event_count)
        non_event_counts.append(non_event_count)

    #Create a data frame for this distribution
    column_list = ["Categories", "Event Count", "Non Event Count"]
    value_list = [categories, event_counts, non_event_counts]
    woe_iv_df = create_df(column_list, value_list)

    #Compute percentage of events and non-event
    woe_iv_df = calculate_woe_iv(woe_iv_df)

    #Sort the data in descending order of counts
    woe_iv_df = woe_iv_df.sort_values(by="Total Counts", ascending=
                                      False)

    return woe_iv_df
```

2.7 Variable Encoding

We have used **One Hot Encoding** for converting the categorical variables into numeric form.

CODE:-

```
#One Hot Encoding
def OHE(data, cat_vars, train_test='train'):
    '''
    Input:-
    data:- The data set to be encoded
    cat_vars:- The list of categoricalvariables
    train_test:- The identifier for determining whether it's a
                                         train data or test data
    '''
    if(train_test=='train'):
        #Drop the first category if it's a train data
        encoded_data = pd.get_dummies(data[cat_vars], drop_first=
                                         True)
    else:
        #Don't drop first category for non-training data
        encoded_data = pd.get_dummies(data[cat_vars])

    encoded_features = list(set(encoded_data.columns).difference(
                                         set(cat_vars)))

    #Append the original features
    encoded_data = pd.concat([data, encoded_data], axis=1)

    #Return the encoded_data along-with the encoded features
    return encoded_data, encoded_features

encoded_data_train, encoded_features_train = OHE(train_data,
                                  categorical_vars)
encoded_data_test, encoded_features_test = OHE(train_data,
                                  categorical_vars, 'test')
```

2.8 Modelling

Various algorithms have been tried. Please find the code for modelling below:-

```
class Modeling():

    def __init__(self, X_train, X_test, y_train, model,
                                      name_of_classifier):
        self.X_train = X_train
        self.X_test = X_test
        self.y_train = y_train
        #self.y_test = y_test
        self.model = model
        self.clf_name = name_of_classifier

    def save_proba(self, probability, y, train_test):
        #Write the probability to a file
        probability_df = pd.DataFrame(probability, columns= ["
                                           Genuine", "Fraud"])

        if(train_test=='train'):
            probability_df["Actuals"] = y.tolist()

        probability_df.to_csv(os.getcwd() + "/Model Results/" +
                                           train_test + "
                                           Probabilities.csv")

    def auc_roc(self, X, y, train_test):
        predicted_probability = self.model.predict_proba(X)

        #Actuals for train data are present--> y_train
        if(train_test=='train'):
            #Save the probabilities
            self.save_proba(predicted_probability, y, train_test)

            fpr, tpr, _ = roc_curve(y,  predicted_probability[::,1]
                                               )

            auc = roc_auc_score(y, predicted_probability[::,1])
            plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
            plt.legend(loc=4)
            plt.xlabel("False Positive Rate")
            plt.ylabel("True Positive Rate")
            plt.savefig(os.getcwd() + "/Model Results/" +
                                               train_test + " AUC "
                                               + self.clf_name + ".
                                               png")
            plt.show()

        else:
            #Save the test probabilities
            self.save_proba(predicted_probability, y, train_test)

    def evaluate(self, CM, train_test):

        accuracy = CM.diagonal().sum()/CM.sum()
        precision = CM[1][1]/(CM[1][1] + CM[0][1])
        recall = CM[1][1]/(CM[1][1] + CM[1][0])
        f_measure = (2*precision*recall)/(precision+recall)
        print(train_test + " Data")
        print(CM)
        print("Accuracy:- {}\nPrecision:- {}\nRecall:- {}\nF-
                                         measure:- {}" \
              .format(accuracy,precision, recall, f_measure))

    def prediction(self):
        #Fit the model
        model = self.model.fit(self.X_train, self.y_train)

        #Predict on test set
        predicted_train = model.predict(self.X_train)
        predicted_test = model.predict(self.X_test)

        #Results
        result_train = pd.DataFrame({'Actual':self.y_train.ravel(),
                                          'Predicted':
                                         predicted_train.ravel()},
        columns=['Actual','Predicted'])

        result_test = pd.DataFrame({'Predicted':predicted_test.
                                         ravel()},
        columns=['Predicted'])

        #Save the model
        joblib.dump(model, os.getcwd() + "/Model Results/" + self.
                                         clf_name + ".pkl")

        #Print the AUC_ROC curve
        self.auc_roc(self.X_train, self.y_train, "train")
        self.auc_roc(self.X_test, '', "test")

        #Evaluate the model
        CM_train = CM(self.y_train.ravel(), predicted_train.ravel()
                                         , labels=None,
                                         sample_weight=None)
        #CM_test = CM(self.y_test.ravel(), predicted_test.ravel(),
                                         labels=None,
                                         sample_weight=None)

        self.evaluate(CM_train, "train")
        #self.evaluate(CM_test, "test")
```

2.9 Results

The results obtained via Logistic regression Please refer the file **Results Various Cut Off.csv** The event rate is 16.67%.

Probability Cut Off	Accuracy	Precision	Recall	F1 Score
15%	76.44%	39.78%	80.39%	53.22%
16.67%	78.59%	42.15%	76.28%	54.30%

Table 5: Caption

train$_{A}UC_{L}ogistic.png$

Figure 1: AUC ROC Curve