Customer Listings

Shrikrishna Padalkar

June 2021

1 Problem Statement

Identify good and bad item listings in the dataset link provided below. This will help a company avoid bad listings from being put up on their platform and pull them down if already there so that customer impact and dissatisfaction is minimal. This is a classification problem.

- Data Statistics
- Data Cleaning
- Derived Variables
- Splitting Data(Train/Test)
- Exploratory Data Analysis
- Modelling

1.1 Data Statistics

- The raw data consists of 3,57,997 records and 70 variables.
-
- Target Variable

  The target Variable is "Listing_Type" which contains two classes Good and Bad.
  Code:-

```
#Check the target distribution
def get_distribution(cat_var, data):
    '''
    Input
    cat_var:- A categorical variable
    Results
    counts_df:- A data frame containing the counts and the %
                                        of entries
    '''
    counts = dict(Counter(data[cat_var]))

    counts_df = pd.DataFrame({"Category": list(counts.keys()),
                        "Values": list(counts.values())},
                        columns=["Category", "Values"])
    total = sum(counts_df["Values"])

    counts_df["Percentage"] = counts_df["Values"]*100/total

    #Sort in descending order of counts
    counts_df = counts_df.sort_values(by="Values", ascending=
                                        False)
    total_df = pd.DataFrame({"Category": "Total", "Values":
                                        total, "Percentage":100}
                                        ,
        columns=["Category", "Values", "Percentage"], index=[
                                            counts_df.shape[0]])
    counts_df = pd.concat([counts_df, total_df])

    return counts_df
```

1.2 Data Cleaning

- Target Variable

  The target variable consists of many garbage entries. Considering only the 2 targets "Good" and "Bad".
- Numeric Variables

  There are many string values and non-numeric values in the numeric variables. The numeric values present in the string format have been converted to integers.

– Variable Amenities:-

Each cell in the amenities column contains a sub of amenities from a superset. We can create a column for each amenities. Place a 1 or 0 to mark the presence(/absence) of the amenity. We can create a variable consisting of total number of Amenities.

CODE:-

```
#Create a variable Total Number of Amenities
all_unique_amenities = set()

#regex = re.compile("[%s]" %(re.escape(string.punctuation)
                                    ))
pattern = '"|{|}'
#Replace the garbage amenities like single characters,
                                    only punctuations to ''

def get_all_unique_amenities(amenity):
    global all_unique_amenities
    if(type(amenity)==str):
        #Remove the unwanted strings
        amenity = re.sub(pattern, '', amenity)

        amenity = str.upper(amenity) #Convert to upper
                                            case

        #Convert the amenity_list to set
        if(',' in amenity):
            amenity_set = set(amenity.split(','))
        else:
            amenity_set = set(amenity)
            #Replace single characters or only
                                                punctuations
                                                 or special
                                                characters
                                                with ''

        all_unique_amenities = all_unique_amenities.union(
                                            amenity_set)
        return len(amenity_set)

no_amenities = [get_all_unique_amenities(amenity_set) for
                                    amenity_set in
                                    train_data["Amenities"]]

print("Number of unique amenities:- %d" %len(
                                    all_unique_amenities))
```

– Variable Country_Code and Country

Country_Code is the code for each of the unique Countries in the Country variable. They are the same, so use either of the 2. We have used Country_Code.

– Variables to Ignore The variables having missing values ¿30% have been dropped. There are a few categorical variables having many categories, these have been dropped.

Variable	Type	Reason
Host name	Categorical	Cannot be used due to too many categories
Geolocation	String	Latitude and Longitude can be used instead
Country	Categorical	Use Contry_Code instead of Country
Experiences_Offered	Categorical	Majority(95%)$valuesare'NONE'$
Host_Location	Categorical	Cannot be used due to too many categories
Zipcode	Categorical	Cannot be used due to too many categories
Summary	Categorical	Cannot be used due to too many categories

Table 1: Caption

1.3 Derived Variables

- Review_Proximity

  It is the difference between Last_Review and First_Review in days. This has been converted to year.

- Co-Ordinates

  The co-ordinates are obtained using the below formula.

Co-Ordinate	**Formula**
X	$\cos(lat) * \cos(long)$
Y	$\cos(lat) * \sin(long)$
Z	$\sin(lat)$

  Check whether the Latitude and Longitude have missing values for same indices(/rows).

CODE:-

```
#Check whether Latitude and Longitude are missing for the
                                    same observations
missing_longitude_index = set(considered_data.loc[
                                    considered_data["
                                    Latitude"].isnull()==
                                    True, "Longitude"].index
                                    .tolist())

missing_latitude_index = set(considered_data.loc[
                                    considered_data["
                                    Longitude"].isnull()==
                                    True, "Latitude"].index.
                                    tolist())

#Empty sets imply that Latitude and Longitude are missing
                                    for the same set of
                                    indices
print(set(missing_longitude_index).difference(set(
                                    missing_latitude_index))
                                    )
print(set(missing_latitude_index).difference(set(
                                    missing_longitude_index)
                                    ))
```

The latitude and the longitude can be used to determine the coordinate(x,y,z) of the location.

– Number of Amenities

The variable Amenities contains collection of amenities for a particular user. We can use the number of amenities present in the set. Another option can be extracting each amenity from the set and create a variable corresponding to every unique amenity across the entire variable.

1.4 Splitting Data(Train/Test)

We split the data into Training(80%) and Testing(20%) data sets. We carry out the transformations on Train Data and apply these transformations on the Test Data. The assumption is Test Data is the representative of the Train Data. viz. Binning is performed on Train Data and the bins obtained are applied on the Test Data.

The activity of finding less frequent groups has to be done only on training dataset. These groups are to be tagged as "OTHERS" in both test dataset and train dataset. Below is list of variables that have been grouped.

Variables
Bed_Type
Property_Type
Cancellation_Policy
$\text{Host}_{R}esponse\_Time$
Country_Code

Table 2: Caption

```
target = "Listing_Type"
independent_vars = list(set(considered_variables).difference(
                                    set([target])))

X = considered_data[independent_vars]
y = considered_data["Listing_Type"]
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.2, random_state=
                                    42)
```

- Target Distribution - Train Data

Category	**Values**	**Percentage**
Good	2,40,950	90.67
Bad	24,769	9.32
Total	2,65,719	100

- Target Distribution - Test Data

Category	**Values**	**Percentage**
Good	60,266	90.72
Bad	6,164	9.27
Total	66,430	100

1.5 Exploratory Data Analysis

- Uni-variate Analysis

  Please refer the file **Statistics_considered_data.csv** Based on the frequency distribution of the categorical variables, less frequent categories are clubbed into a new group. Those categories covering 95% of the cummulative counts have been retained, rest are tagged as "OTHERS".

- Bi-variate Analysis

	Beds	Bathrooms	Bedrooms	
Beds	1.0	0.54	0.74	
Bathrooms	0.54	1.0	0.59	
Bedrooms	0.74	0.58	1.0	

Table 3: Caption

For continuous variables we have used Karl Pearsons's coefficient of correlation. Then table below shows the correlation amongst some of the variables in the train data.

CODE:-

```
#Find the correlation between the numeric variables
def correlation(data, label=="base_data"):

    f, ax = plt.subplots(figsize=(7,5))
    data_corr = data[numeric_variables].corr()

    sbn.heatmap(data_corr,
                mask = np.zeros_like(data_corr,\
                dtype=np.bool),
                cmap = sbn.diverging_palette(220, 20,\
                as_cmap=True),\
                square = True,
                ax = ax)
    plt.savefig(os.getcwd() + "/Plots/" + label + "
                                        _Correlation.png")
```

– **WOE & IV**

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.

We compute the WOE and IV values for each of the categorical variables. The categories that are less frequent have been clubbed into a new category named "OTHERS". This is helpful for dimensionality reduction.

CODE:-

```
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
```

```
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
```

Variable	WOE	IV	# Categories
Bed_Type	5.48	0.001	6
Bed_Type_grouped	1.99	0.00	3
Cancellation$_{P}olicy$	9.66	0.47	6
Cancellation_Policy_grouped	9.66	0.47	12
Country_Code	25.97	0.02	21
Country_Code_grouped	12.19	0.02	13
Host_Response_Time	7.12	0.33	5
Host_Response_Time_grouped	3.36	0.13	4
Property_Type	59.65	0.016	41
Property_Type_grouped	5.62	0.00	6

Table 4: Categorical Variables

```
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
```

Information Value	Variable Predictiveness
Less than 0.02	Not useful
[0.02, 0.1)	Weak
[0.1, 0.3)	Medium
[0.3, 0.5)	Strong
Greater than 0.5	Suspecious

Table 5: Caption

Variables	Missing Value Counts	Missing Value Percentage
Reviews_per_month	65787	24.75
Host_Response_Time	62620	23.56
Extra_People	46918	17.65

Table 6: Caption

- **IV Values**
  The IV values(/Information Values) suggest the predictive power of a variable. The classification of predictive powers based on the IV Values is shown below.

1.6 Variable Selection

- **Missing Values** The variables having more than 30 per. missing values have not been considered. For those variables where the percentage of missing values is less than 10% we can impute these variables. We choose KNN imputation for categorical variables and Median imputation for continuous variables. The median value does not change after capping the outliers.
- **Imputation** The categorical variables have been imputed by KNN imputation technique. It's a lazy evaluation technique, since for each missing record distance needs to be calculated. The continuous variables are imputed via median.

1.7 Modelling

```
class Modeling():

    def __init__(self, X_train, X_test, y_train, y_test, 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= ["
                                        Bad", "Good"])
        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)

        #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()


    def evaluate(self, CM, train_test):

        accuracy = CM.diagonal().sum()/CM.sum()
        precision = CM[0][0]/(CM[0][0] + CM[0][1])
        recall = CM[0][0]/(CM[0][0] + 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, f=1):
```

```
        #X_train, X_test, y_train, y_test = train_test_split(
                                            self.X, self.y,
                                            test_size=test_size/
                                            100, random_state=42
                                            )

        #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({'Actual':self.y_test.ravel
                                            (), 'Predicted':
                                            predicted_test.ravel
                                            ()},
        columns=['Actual','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, self.y_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")
```

– Test Results

Confusion Matrix

	Predicted Bad	Predicted Good
Actual Bad	2,369	829
Actual Good	59,533	74,577

– Test Results
Confusion Matrix

		Predicted Bad	Predicted Good	
	Actual Bad	607	217	
	Actual Good	14,651	18,910	