Predicting if the customer is satisfied by the resolution.

Customer Satisfaction is the key feature which helps in the business growth. We have seen many companies in the past which were collapsed as they didn't pay much attention to their customer support system. For a company, the customers are as much precious as their product is. In united states, Customer Federal Saving Bank basically collects the data of the customer complaints and company response to their complaints to analyse the data and helps the government to make new financial laws. This dataset is publicly available on "consumerfinance.gov".

What makes this case study interesting?

This case study works as the two-way benefit:

- For companies: It helps to find out on which factors the particular company needs to work to be sounded in the business market.
- For customers: It helps customer to select the company for the products they want and which company provides the best customer support.

CONTENTS

The case study is integrated using the supervised learning.

What is Supervised Learning?

In this type of learning, we have labelled input data. This means that the data presented to the model already contains the correct answer. We are giving this pre-labelled data to make the model learn from it. This means if data with similar features is given to the machine in the future, it will **recognize** it.

The type of supervised learning used is **Classification**.

Classification Supervised Learning: It specifies the category to which data elements belong to and is best used when the output has finite and discrete values. It predicts a category for an input variable also.

In our dataset we have to find out whether the customer is satisfied or not from the resolution so it comes under two class classification supervised learning.

Collecting Data

As you know, machines initially learn from the data that you give them. It is of the utmost importance to collect reliable data so that your machine learning model can find the correct patterns. The quality of the data that you feed to the machine will determine how accurate your model is. If you have incorrect or outdated data, you will have wrong outcomes or predictions which are not relevant.

The first step was to load the dataset which have done using pandas.

```
df = pd.read_csv("consumer_complaints.csv")
print(df.columns)
```

```
Index(['date_received', 'product', 'sub_product', 'issue', 'sub_issue',
       'consumer_complaint_narrative', 'company_public_response', 'company',
       'state', 'zipcode', 'tags', 'consumer_consent_provided',
       'submitted_via', 'date_sent_to_company', 'company_response_to_consumer',
       'timely_response', 'consumer_disputed?', 'complaint_id'],
      dtype='object')
```

Above, is the columns name present in the dataset.

```
print(df.shape)
```

```
(555957, 18)
```

Exploring Dataset

The dataset contains 555957 observations and 18 features available for analysis. After the analysis, it was observed that there are many columns and rows which have null values.

Description of columns:

Date received: The date the CFPB received the complaints.

Product: The type of product customer identified in the complaint.

Sub-Product: The type of sub-product customer identified in the complaint.

Issue: The type of issue customer identified in the complaint.

Sub-Issue: The type of sub-issue customer identified in the complaint.

Company: The complaint is about which company,

State: The state of the mailing address provided by the customer.

Zip code: The mailing zip code provided by the customer.

Submitted_via : How the complaint is submitted to the CFPB.

Date_sent_to_company: The date at which CFPB sent complaint to the company.

Company response to consumer: The response of company to the consumer.

Timely response: Whether the company gave the timely response or not.

Customer disputed? : Whether the consumer disputed the company response .

Complain id: The unique identification no. of the complaints.

Feature Extraction

In this section, important features were extracted from the features present in the dataset and date columns were transformed.

Respective columns were dropped (customer_complaint_narartive, company_public_response, tags, company_public_response) as these columns had more than 75% of null. Moreover, they were not adding much value to the analysis.

```
print(df['date_received'].head(3))
```

```
0    08/30/2013
1    08/30/2013
2    08/30/2013
Name: date_received, dtype: object
```

The dtype of date columns is object. That means currently pandas seeing the date columns as string.
To convert these strings into internal datetimes, we can use the pan-das function to_datetime.
Upcoming step was to extract some information from the date columns. Firstly day, year, month were extracted from the date columns as this would give us better insights about the data i.e. in which year, month the companies got the maximum complaints.

```
#Adding columns year,month,day for date
dfnew['date_received'] = pd.to_datetime(dfnew['date_received'])
#extract the year
dfnew['date_received_year'] = dfnew['date_received'].dt.year
#extract the month
dfnew['date_received_month'] = dfnew['date_received'].dt.month
#extract the day
dfnew['ddate_received_day'] = dfnew['date_received'].dt.day
```

```
# Dropping the date_received and date_sent_to_company
dfnew = dfnew.drop(['date_received', 'date_sent_to_company'], axis = 1)
```

Dropping the date columns as we have already transform them into year, month and day Columns.
customer_disputed column were treated as the target variable, as if the customer were satisfied by the company resolution, he/she haven't disputed again.

```
: print(df['consumer_disputed?'].value_counts())

  No     443823
  Yes    112134
  Name: consumer_disputed?, dtype: int64
```

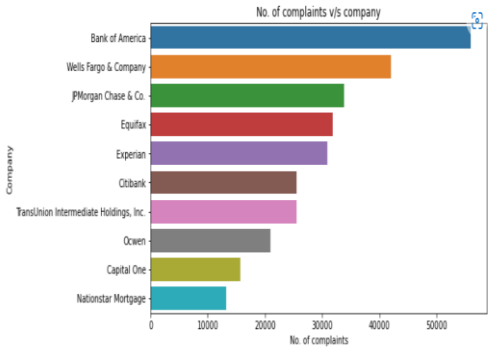
This is imbalanced dataset.
New Dataset has following columns:

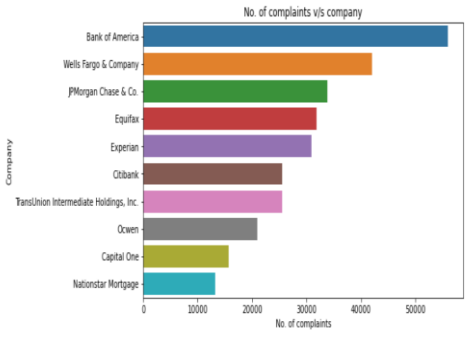
```
print(dfnew.columns)
```

```
Index(['product', 'sub_product', 'issue', 'sub_issue', 'company', 'state',
       'zipcode', 'submitted_via', 'company_response_to_consumer',
       'timely_response', 'consumer_disputed?', 'complaint_id',
       'date_sent_to_company_year', 'date_sent_to_company_month',
       'date_sent_to_company_day', 'date_received_year', 'date_received_month',
       'ddate_received_day'],
      dtype='object')
```

Data Visualization

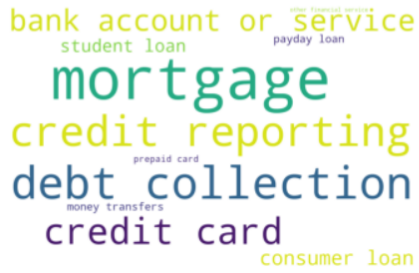
Since most of the columns are categorical, so mainly barplots or countplots were used using seaborn because it would be easier to interpret!
Below mention are few examples of the plots in the notebook, you can check more graphs in the attached link of notebook and follow by the conclusion.

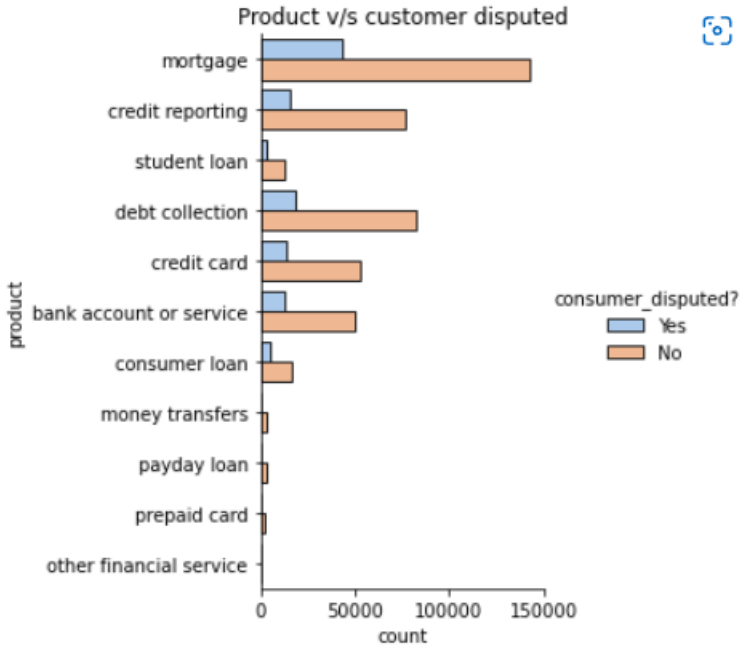




To find out the most occurring category of product found in the complaints, word cloud plot was used.

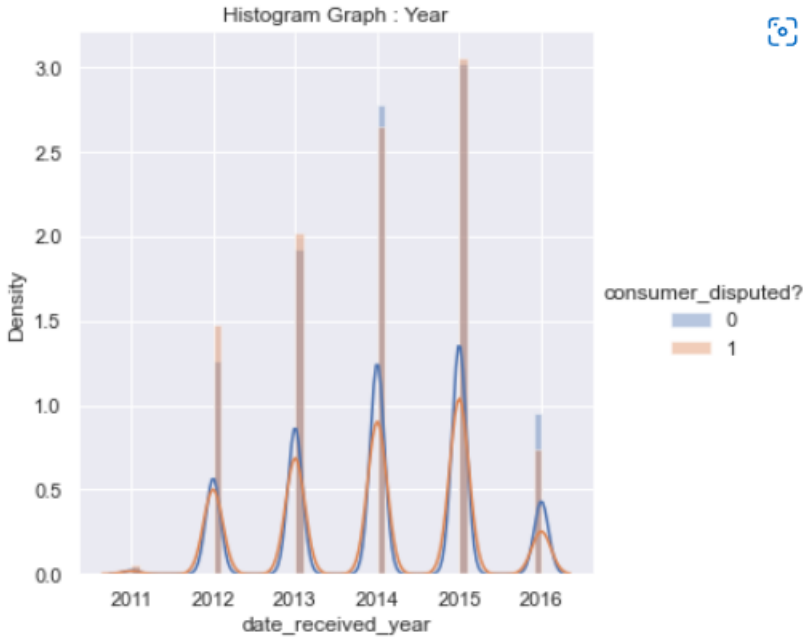
```
from wordcloud import WordCloud
data = df['product'].value_counts().to_dict()
wc = WordCloud(width = 1200, height = 800, background_color ='white',min_font_size = 10).generate_from_frequencies(data)

plt.imshow(wc)
plt.axis('off')
plt.show()
```



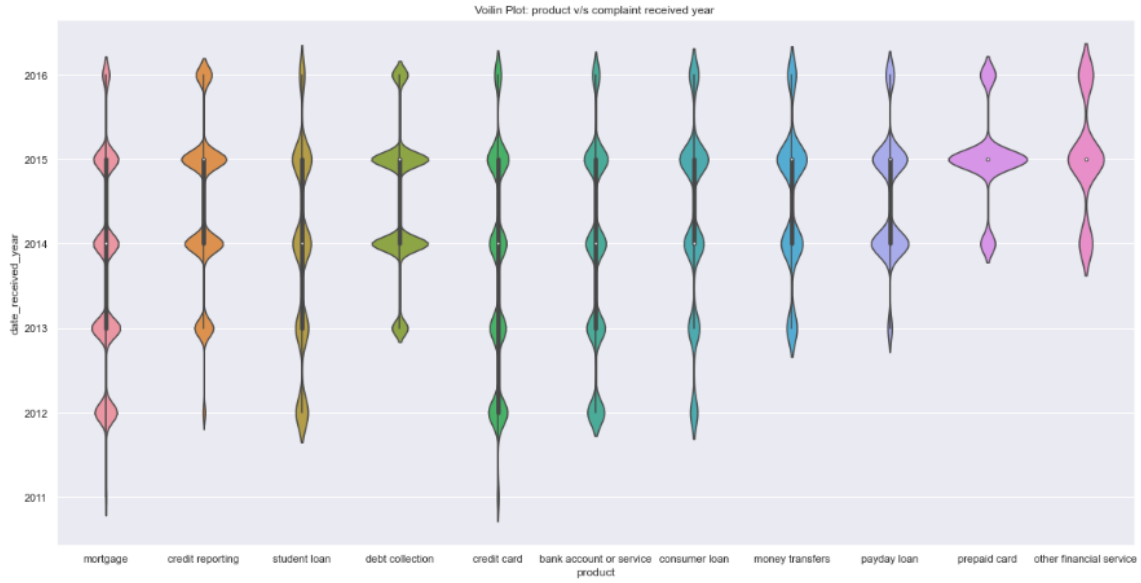


Above plot is plot b/w product and count of complaints, bar graph shows us that the customer satisfaction rate.
From the above plot we can conclude that there is large no. of customers which are not satisfied by the service provided by financial institution for the Mortgage product followed by Debt collection and credit reporting compared to any other product. As it may be because these products have higher no. of complaints.

Above plot is b/w the top companies which got the max. complaints and the no. of complaints wi
th the bar plot shows us the timely response.
It can be inferred that from the top companies which got higher no. of complaints, Bank of America lacks in timely_response and timely_response is very important for the customer satisfaction.



Above graph shows us the no. of complaints in each year.
In 2014 and 2016, the customer satisfaction rate is high as compared to other years.



Above are the violin plot b/w complaint received year and product.

Major findings from the plot:

- **Mortgage :**
  - **2011:** Complaints from Mortgage product were not received .
  - In year 2012,2013,2014,2015 companies got moderate no. of complaints.
  - In year 2016 the no. of complaints were less than that of other years except 2011.
- **Credit Reporting :**
  - In 2015 ,companies got higher no. of complaints than any other year followed by 2014,2013,2016,2012.
- **Student Loan**
  - In every year company got the complaints from student loan product but seems slightly decrease in no. of complaints in 2016 and with zero complaints in 2011.
- **Debt collection :**
  - Max no. of complains in 2014 and 2015
- **Credit Card :**
  - In every year company got the complaints from Credit Card product but seems slightly decrease in no. of complaints in 2016 and with zero complaints in 2011.
- **Bank Account or Services :**
  - n every year company got the complaints from Bank Account or Services product but seems slightly decrease in no. of complaints in 2016 and with zero complaints in 2011.
- **Customer loan :**
  - Max no. of complains in 2015 followed by 2014
- **Money Transfers :**
  - Max no. of complains in 2015 followed by 2014
- **Pay day loan :**
  - Max no. of complains in 2015 followed by 2014
- **Prepaid Card :**
  - Max no. of complains in 2015 and it is highest among all other product in that year.

From the previous observations we found that in 2015 companies got the max. no. of complaints. Regarding that the max. no. of complaint were from the products credit reporting, debt collection and prepaid card.

Conclusion from Data Visualisation

- The most of the complaints were regarding the Mortgage product followed by Credit re-p orting, Debt Collection, Bank account or service.
- The major issues which were highlighted in the complaints are Loan modification, collection, foreclosure, incorrect information on credit report, Loan servicing.

- Bank of America got highest no. of complaints followed by Wells Fargo, JPMorgan Chase. The reason can be that these banks provide large no. of product services.
- Among the top companies which got the highest complaints, Equifax is the company which provide only credit card product and other companies provides 5-6 products. It means that Equifax have worst customer satisfaction rate. Acc. to my view companies could tend to get more complains when they provide larger no. of the products but Equifax is not one of them.
- From the previous observations we found that in 2015 companies got the max. no. of complaints. Regarding that the max. no. of complaint were from the products credit reporting, debt collection and prepaid card.

Data Pre-processing

It is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

But why we need it?

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this we use data pre-processing task.

```
#converting each word into lowercase
dfnew['product'] = dfnew['product'].str.lower()
dfnew['sub_product'] = dfnew['sub_product'].str.lower()
dfnew['issue'] = dfnew['issue'].str.lower()
dfnew['sub_issue'] = dfnew['sub_issue'].str.lower()
dfnew['company'] = dfnew['company'].str.lower()
dfnew['company_response_to_consumer'] = dfnew['company_response_to_consumer'].str.lower()
dfnew['timely_response'] = dfnew['timely_response'].str.lower()
dfnew['submitted_via'] = dfnew['submitted_via'].str.lower()
```

The lower() function method converts all the uppercase characters in the string into the lowercase characters.

```
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

Here all the words presented in the contracted form are deconstructed with the help of regular expression.

The

Feature Engineering

Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric that's why I transformed the categorical data using the OneHotEncoding and Response Coding.

```
def hotencoder(dfeature):
    labelencoder = LabelEncoder()
    int_label_sub = labelencoder.fit_transform(dfeature)
    int_labe_sub = int_label_sub.reshape(len(int_label_sub),1)
    one_hot_encod_sub = OneHotEncoder(sparse=False)
    one_hot_enc = one_hot_encod_sub.fit_transform(int_labe_sub)
    return one_hot_enc
```

Code for One HotEncoding

Initially label encoding were done and top of that onehotencoding was done .You can directly go with onehotencoding .

```
#Response Coding
def responsedict(feature):
    #unique values of feature
    unique_labels = X_train[feature].unique()
    key = []
    val =[]
    for i in unique_labels:

        #Count where the target_variable is 1.
        positive = X_train.loc[:,feature][((X_train[feature] == i) & (y_train == 1))].count()
        #Count where the target_variable is 0.
        negative = X_train.loc[:,feature][((X_train[feature] == i) & (y_train == 0))].count()
        total_count = positive + negative
        #negative probability
        neg_prob = negative/total_count
        #positive probability
        pos_prob = positive/total_count
        key.append(i)
        val.append([neg_prob,pos_prob])

    prob_dict = dict(zip(key,val))
    return prob_dict

def feature_name(feature,df):
    feature_dict = responsedict(feature)
    unique_label = X_train[feature].unique()
    fea = []
    for i,val  in df.iterrows():
        if val[feature] in unique_label:
            fea.append(feature_dict[val[feature]])
        else:
            fea.append([0.5,0.5])

    return fea
```

Code for Response Coding

In responsedict function, the positive and negative value is calculated, to create the response table. Response table is used to represent the number of data points belonging to each output class for a given category. Once we have the response table, we encode this information by adding the same number of features in the dataset as the cardinality of the class labels to represent the probability of the data point with given category, belonging to a particular class, which is stored in prob_dict for the further computation. Prob_dict is computed only on the training dataset.

In feature_name function, for a particular feature, the prob_dict is revoked using the responsedict function and also unique label(category) present for particular feature is stored in the unique_label for training sample. We don't do any computation for test samples. We just search for the values using the particular category of the feature as the key in the proba_dict and assign the corresponding value to the category and if it in not present in the proba_dict, then we assign the default value which is 0.5,0.5 is in our case. The no. of values depends on the no. of category of target variable which is 2 in our case i.e 0 or 1.

One hot encoding works well unless your categorical variable doesn't take large values. But zipcode, state and company column have large values that's why I implemented response coding for them. When one hot encoded column was used, memory space error as their values is too big to handle for 16GB memory.

Normalization

It is key step to normalize the values of columns having numerical values. Normalization is done to adjust values measured on different scales to notionally common scale. It also improves the model performance.

```
#normalizing the column having numeric data
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['date_sent_to_company_year'].values.reshape(-1, 1))

X_train_normalized_company_year = normalizer.transform(X_train['date_sent_to_company_year'].values.reshape(-1,1))
X_test_normalized_company_year = normalizer.transform(X_test['date_sent_to_company_year'].values.reshape(-1,1))

normalizer.fit(X_train['date_sent_to_company_month'].values.reshape(-1, 1))

X_train_normalized_company_month = normalizer.transform(X_train['date_sent_to_company_month'].values.reshape(-1,1))
X_test_normalized_company_month = normalizer.transform(X_test['date_sent_to_company_month'].values.reshape(-1,1))

normalizer.fit(X_train['date_sent_to_company_day'].values.reshape(-1, 1))

X_train_normalized_company_day = normalizer.transform(X_train['date_sent_to_company_day'].values.reshape(-1,1))
X_test_normalized_company_day = normalizer.transform(X_test['date_sent_to_company_day'].values.reshape(-1,1))

normalizer.fit(X_train['date_received_year'].values.reshape(-1, 1))

X_train_normalized_received_year = normalizer.transform(X_train['date_received_year'].values.reshape(-1,1))
X_test_normalized_received_year = normalizer.transform(X_test['date_received_year'].values.reshape(-1,1))

normalizer.fit(X_train['date_received_month'].values.reshape(-1, 1))

X_train_normalized_received_month = normalizer.transform(X_train['date_received_month'].values.reshape(-1,1))
X_test_normalized_received_month = normalizer.transform(X_test['date_received_month'].values.reshape(-1,1))

normalizer.fit(X_train['ddate_received_day'].values.reshape(-1, 1))

X_train_normalized_received_day = normalizer.transform(X_train['ddate_received_day'].values.reshape(-1,1))
X_test_normalized_received_day = normalizer.transform(X_test['ddate_received_day'].values.reshape(-1,1))
print(X_train_normalized_received_day.shape)
normalizer.fit(X_train['complaint_id'].values.reshape(-1, 1))

X_train_normalized_complaint_id= normalizer.transform(X_train['complaint_id'].values.reshape(-1,1))
X_test_normalized_complaint_id = normalizer.transform(X_test['complaint_id'].values.reshape(-1,1))
print(X_train_normalized_complaint_id.shape)
```

After all the pre-processing, I stacked the data using “NumPy. Stack”.

```
OneTrainXi = np.hstack((one_hot_enc_product_train,one_hot_enc_sub_product_train,one_hot_enc_issue_train,one_hot_enc_sub_issue_tra
                        one_hot_enc_timely_response_train,one_hot_enc__response_to_consumer_train,X_train_normalized_complaint_id
OneTestXi = np.hstack((one_hot_enc_product_test,one_hot_enc_sub_product_test,one_hot_enc_issue_test,one_hot_enc_sub_issue_test,te
```

```
print(OneTrainXi.shape)
print(OneTestXi.shape)
```

```
(95928, 106)
(23982, 106)
```

But here we are missing one thing that our dataset is imbalanced dataset so we need to either do oversampling or under sampling. But under sampling is not a good choice as we may lose some information. That’s why I selected oversampling which I have done using SMOTE.

But how does SMOTE oversample the data.

SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Specifically, a random example from the minority class is first chosen. Then *k* of the nearest neighbours for that example are found (typically *k*=5). A randomly selected neighbour is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.

```
from imblearn.over_sampling import SMOTE
ros =SMOTE(random_state=777)
ros_xtrain, ros_train_y = ros.fit_resample(OneTrainXi, y_train)
```

Model Training

Algorithms are the key factor used to train the ML models. The data feed into this that helps the model to learn from and predict with accurate results. Hence, choosing the right algorithm is important to ensure the performance of your machine learning model.

The next important factor is the quantity of the data we are using to train the models. Quantity depends on the different factors like for deep learning-based ML models, a huge quantity of datasets is required for algorithms.

Just like quantity, the quality of machine learning training data set is another key factor, you need to keep in mind while developing an ML model.

I dropped all those rows which have null values in any of the columns because I had ample of data points. If I would have limited data points, I had deal with the null values differently like for numerical columns, I would replace the null values with the mean of the values of the columns and for categorical values, we could take an unknown category for columns which had null values.

I have tried with no. of models and you can see the results below. I have hyper tuned each model.

Model	Train AUC	Test AUC	Train Accuracy	Test Accuracy
Logistic Regression	0.83	0.622	0.74	0.63
Decision Tree	0.90	0.584	0.81	0.59
Xgboost	0.96	0.610	0.90	0.76
RandomForest	0.92	0.619	0.75	0.57
Stacking	0.95	0.599	0.87	0.66

While selecting the finalised model we need to take into consideration that the difference between the test and train performance metrics should not be large. We can take 0.2 as the upper limit. That's why I selected logistic regression.

Parameters used to hyper tune logistic regression model:

penalty : *{'l1', 'l2', 'elasticnet', 'none'}, default='l2'*

Specify the norm of the penalty:

C : *float, default=1.0*

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

solver : *{'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, default='lbfgs'*

Algorithm to use in the optimization problem. Default is 'lbfgs'. To choose a solver, you might want to consider the following aspects:

max_iter : *int, default=100*

Maximum number of iterations taken for the solvers to converge.

```
from sklearn.linear_model import LogisticRegression
# dict of parameters
grid={"C":np.logspace(-3,3,7),
     "solver" : ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
     "penalty": ['l1', 'l2'],
     "max_iter": [100,200,300]}
logreg=LogisticRegression()
#Random search
logreg_cv=RandomizedSearchCV(logreg,grid,cv=10)
logreg_cv.fit(ros_xtrain, ros_train_y)

print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
```

```
tuned hpyerparameters :(best parameters)  {'solver': 'saga', 'penalty': 'l2', 'max_iter': 200, 'C': 1000.0}
```

Here, RandomizedSearchCV was used which is a kind of hyper tuning technique.

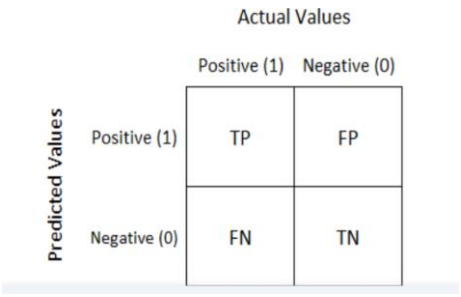
But why we need to do hyper tuning? Because it is the powerful tool to enhance the supervised learning models-improving accuracy, precision and other metrics by searching the optimal model parameters based on different scoring method.

Performance metrics

As this case study is related to the classification problem, we have ample of choices for the performance metrics but take this into account that performance of metrics is highly get affected by the imbalanced dataset. We have already balanced our dataset using SMOTE that's why we are free to choose any performance metrics.

I selected accuracy, confusion metrics and roc-auc score.

Confusion Matrix

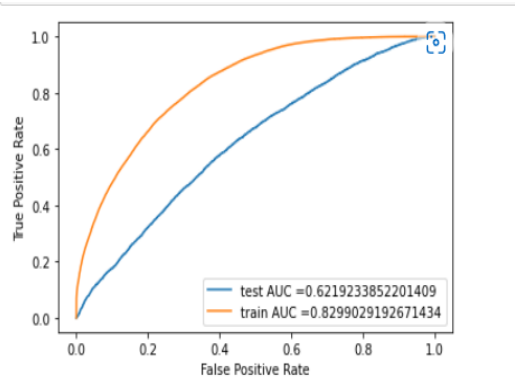


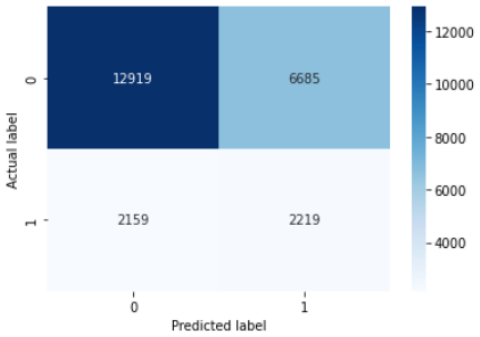
By using the confusion metrics, we can find out the precision, accuracy, Recall, F1score, Roc-Auc value.

ROC-AUC Curve

The **Area Under the Curve (AUC)** is the measure of the ability of a classifier to distinguish between classes.

From the list of models, I found the logistic regression model were working well on the dataset.





Above are the ROC-AUC plot and test confusion matrix respectively.

Feature Importance

Firstly, I tried to get the feature importance score using the model in built function but it was very hard to interpret that which feature corresponds to which score because I have transformed the categorical features using one-hot encoding.

Then I tried it using PCA

```
print(abs(principal.components_[0]))
```

This printed the eigen values corresponding to each eigen vectors.

The importance of each feature is reflected by the magnitude of the corresponding values in the eigenvectors (higher magnitude - higher importance).

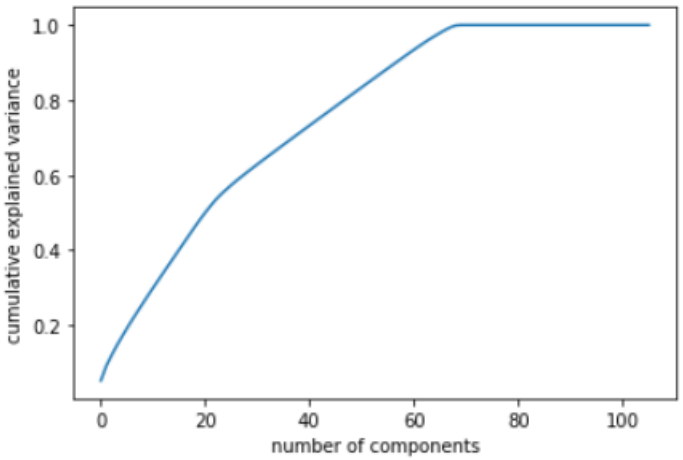
I concluded that complaint_id, date_sent_to_company_year, date_sent_to_company_month, date_sent_to_company_day, date_received_year, date_received_month,ddate_received_day features not contribute at all in the model training as we get zero eigen values corresponding to them.

Conclusion

For Principal Component 1 important features in desc. order are

1. sub-issue
2. issue
3. sub-product
4. product
5. company
6. zipcode
7. state
8. company response
9. timely response
10. submitted_via

```
plt.plot(np.cumsum(principal.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



From the above plot we could infer that after 70 features variance is not changing much. So we can drop all other features. Unnecessary features decrease training speed, decrease model interpretability, and most importantly, decrease generalization performance on the test set.

Thank you for going through the notebook! I hope it was helpful somehow.