CAPSTONE PROJECT

Classification of H1B Visa Applications

GROUP - 9 Final Report

Mentored By : Siddharth Koshta

Submitted By : Aakash Nanthan
Abdhul Rahuman H
Gopinath Karunanithi
Kamalakannan
Krishna S
Pavithra Parthasarathy

Table of Contents

1. Introduction to the project

1.1 Background
1.2 Problem Statement
1.3 Impact on business

2. Dataset and Domain

2.1 Dataset
2.2 Variable Categorization
2.3 Data Dictionary
2.4 Redundant columns

3. Data Exploration

3.1 Exploratory Data Analysis
3.2 Descriptive Statistics
3.3 Missing Value Analysis
3.4 Visualization
3.4.1 Numerical Variables
3.4.2 Categorical Variables
3.4.3 Bivariate Analysis
3.5 Data Scaling
3.6 Encoding
3.7 Imbalanced Data
3.8 Train Test Split

4. Model Building

4.1 Base Model Selection
4.1.1 Logistic Regression
4.1.2 K-NN
4.1.3 Decision Trees
4.2 Building the Base Model
4.2.1 Classification Report
4.3 Fixing Imbalance in the Dataset

4.3.1 Random Oversampling
4.3.2 Random Undersampling

4.3.3 SMOTE
4.4 Why SMOTE

5. Evaluating Model Performance
5.1 Model Performance Metrics
5.1.1 Accuracy
5.1.2 Precision
5.1.3 Recall
5.1.4 F1-Score
5.2 Choosing the Right Metrics
5.3 Comparing Models
5.2.1 Adaboost Classifier
5.2.2 Gradient Boosting Classifier
5.2.3 XGBClassifier
5.2.4 DecisionTree Classifier
5.2.5 RandomForest Classifier
5.4 Choosing the Best Model
5.5 Improving Performance
5.5.1 Hyperparameter Tuning

6. Conclusion
6.1 Conclusion

Chapter 1 - Introduction to the Project

1.1 Background

The United States Citizenship and Immigration Service (USCIS) issues a total of 22 types of visas in the United States of America for temporary workers. Amongst them, the H-1B visa is the most sought-after, due to its limited restrictions in work conditions, longer duration, ease of renewal, and the potential of turning it into permanent residency.

The H-1B visa is a non-immigrant visa that allows U.S. employers to hire foreign workers in specialty occupations that require specialized knowledge and a bachelor's degree or higher. The H-1B visa program is designed to help U.S. companies fill skill shortages and compete in the global market.

1.2 Problem Statement

The annual limit of H-1B visa applicants is fixed at 85,000 annually, and yet they receive applications more than twice the limit each year. With the number of applications increasing each year, we believe a predictive model that helps screen applicants might help reduce waiting times worldwide.

The approval of H-1B visa depends on various factors such as job offer, education qualifications, prevailing wage, and non-immigrant intent.

Using the above factors we will be building a model that helps predict the outcome of the H-1B visa application process.

1.3 Impact on business

The H-1B visa process has been the subject of much debate and scrutiny, particularly in recent years. The process can impact businesses in various ways, including in their hiring practices, recruitment, and workforce planning.

H-1B holders do not have the same freedom to switch jobs as their local peers. As a result, they have little power in negotiating for their own benefit. Since the application process is purely random, applicants with different skill levels have the same chance of getting an H1-B visa, which means cheap labor is imported on a large scale, while a great number of real highly-skilled workers are being rejected.

Chapter 2 - Dataset and Domain

2.1 Dataset

The Dataset is from the USCIS (United States Citizenship and Immigration Services) that contains all the H-1B petition, approve and denied records of the latest available data. The dataset has more than thirty-three lakh records on twenty-five variables including ‘Case Status’ ‘Employer’, ‘Job Title’, ‘State', ‘City’, ‘Full-Time’ etc.

Datasource:

https://www.kaggle.com/datasets/thedevastator/h-1b-non-immigrant-labour-visa

2.2. Data Dictionary

- case_year: The year in which the case was submitted
- case_status: The status of the case, either approved or denied
- case_submitted: The date on which the case was submitted
- decision_date: The date on which the decision was made
- emp_name: The name of the employer
- emp_city: The city in which the employer is located
- emp_state: The state in which the employer is located
- emp_zip: The zip code of the employer
- emp_country: The country in which the employer is located
- job_title: The title of the job for which the visa is being applied
- soc_code: The Standard Occupational Classification code for the job
- soc_name: The name of the Standard Occupational Classification for the job
- full_time_position: Whether the position is full-time or not
- prevailing_wage: The prevailing wage for the job
- pw_unit: The unit of the prevailing wage
- pw_level: The level of the prevailing wage
- wage_from: The minimum wage for the job
- wage_to: The maximum wage for the job
- wage_unit: The unit of the wage
- work_city: The city in which the job is located
- work_state: The state in which the job is located
- emp_h1b_dependent: Whether the employer is H-1B dependent or not
- emp_willful_violator: Whether the employer is a willful violator or not
- lng: The longitude of the job location
- lat: The latitude of the job location

2.3 Variable categorization

Column Name	Datatype	Non-Null Values
case_status	object	579397
case_year	int64	579397
case_submitted	object	579397
decision_date	object	579397
emp_name	object	579347
emp_city	object	579385
emp_state	object	579380
emp_zip	object	579383
emp_country	object	490916
job_title	object	579394
soc_code	object	579397
soc_name	object	579396
full_time_position	object	579395
prevailing_wage	float64	579396
pw_unit	object	579369

pw_level	object	465856
wage_from	float64	579397
wage_to	float64	579397
wage_unit	object	579393
work_city	object	579397
work_state	object	579397
emp_h1b_dependent	object	579395
emp_willful_violator	object	579395
lat	float64	579397
lng	float64	579397

From the table, we can see that emp_country and pw_level have the highest percentage of null values at 15.27 and 19.59 percent respectively.

We are treating null values in numerical columns by replacing them with the median and the null values in the categorical columns by replacing them with the mode of the columns respectively.

2.4 Redundant columns:

After viewing the dataset we are dropping and merging some features which are irrelevant for the EDA purpose. The features are given below:

- Merging case_submitted and decision_date into decision time
- Merging wage_to and wage_from into wage_range
- emp_name
- emp_city
- emp_state
- emp_zip
- emp_country
- job_title
- soc_name
- pw_unit
- work_city
- work_state
- lat
- lng

Using a bit of common sense and domain knowledge, we can say that the name of the employer (emp_name), the city of the employer (emp_city), the state of the employer(emp_state), the zip code, country, job title, work city and work state do not influence the outcome of the H1B visa application process.

Since soc_name and soc_code are conveying the same information to the model, we can remove one. We have removed soc_name since it's a categorical column. The soc_code column consists of six numbers in the format xx-xxxx, where the first two numbers represent the occupation. For example, soc codes that look like 11-xxxx represent management occupations, while 23-xxxx represents legal occupations.

Chapter 3 - Analyzing the data

3.1 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a critical step in data analysis that involves the initial exploration and examination of data to understand its structure, characteristics, and patterns. EDA is an iterative process that involves visualizing and summarizing data through descriptive statistics, graphs, and charts.

3.2 Descriptive Statistics

Descriptive statistics provide an overview of the data, including measures of central tendency, variability, and shape. This helps us to quickly understand the data without having to examine every individual data point.

	prevailing_wage	wage_from
count	5.786550e+05	5.786550e+05
mean	7.876254e+04	8.918352e+04
std	5.033532e+05	5.551086e+05
min	1.508000e+04	1.500000e+04
25%	5.963400e+04	6.500000e+04
50%	7.178100e+04	7.791700e+04
75%	9.023670e+04	1.000000e+05
max	2.016227e+08	2.496000e+08

Fig - 3.1

Above screenshots represent the descriptive statistics of the numeric data before and after scaling.

3.3 Missing Value Analysis

	Total	Percentage of Missing Values
emp_name	50	0.008631
emp_city	12	0.002072
emp_state	17	0.002935
emp_zip	14	0.002417
emp_country	88481	15.274464
job_title	3	0.000518
soc_name	1	0.000173
full_time_position	2	0.000345
prevailing_wage	1	0.000173
pw_unit	28	0.004834
pw_level	113541	19.600569
wage_unit	4	0.000691
emp_h1b_dependent	2	0.000345
emp_willful_violator	2	0.000345

Fig - 3.2

The above image represents the total missing values and missing value percentage

emp_country has 15.27 and pw_level has 19.60 percent of missing values. 85% of values in emp_country are USA, hence we are filling the missing values with USA. On the other hand, we are using bfill to fill in the missing in the pw_level since it has four different classes.

```
df['pw_level'].value_counts()
```

```
Level I      206508
Level II     167807
Level III     58403
Level IV      33138
Name: pw_level, dtype: int64
```

Fig - 3.3

```
df['emp_country'].value_counts()
```

```
USA           579386
CANADA             7
AUSTRALIA          2
CAMBODIA           1
CHINA              1
Name: emp_country, dtype: int64
```

Fig - 3.4

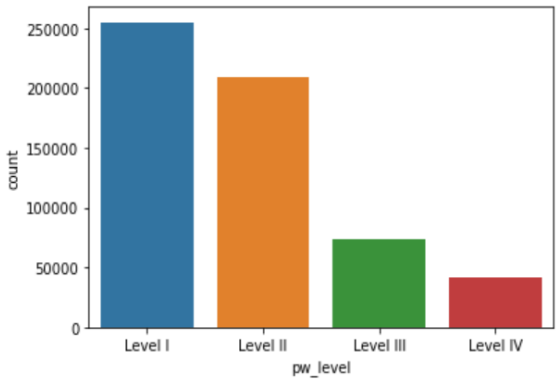

Fig - 3.5

The distribution did not change after filling the null values with b-fill.

	case_status	emp_name	emp_city	emp_state	job_title	soc_code	soc_name	full_time_position	pw_level	work_city	work_state
count	578655	578655	578655	578655	578655	578655	578655	578655	578655	578655	578655
unique	3	63547	4592	54	85538	688	818	2	4	4870	53
top	C	INFOSYS LIMITED	PLANO	CA	PROGRAMMER ANALYST	15-1132	SOFTWARE DEVELOPERS, APPLICATIONS	1	Level I	NEW YORK	CA
freq	506462	20146	32615	97141	44237	118303	117979	566676	248813	33913	110161

Fig - 3.6

3.4 Visualization

3.4.1 Numerical variables

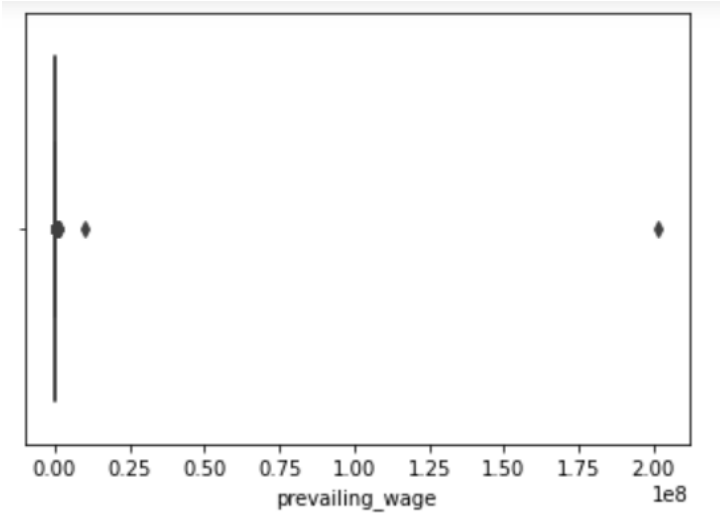

Fig - 3.7

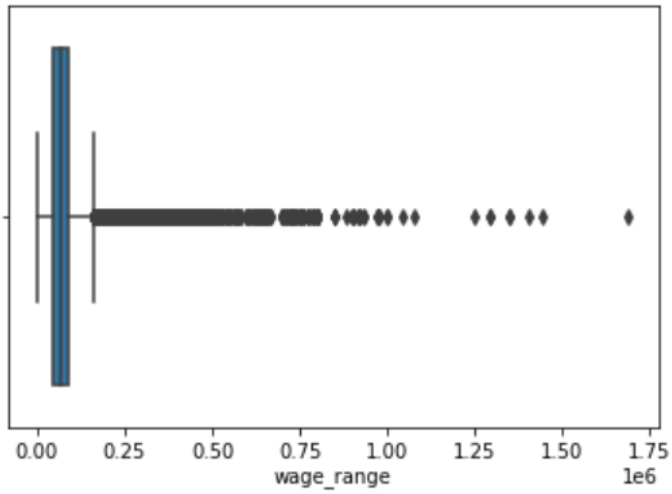

Fig - 3.8

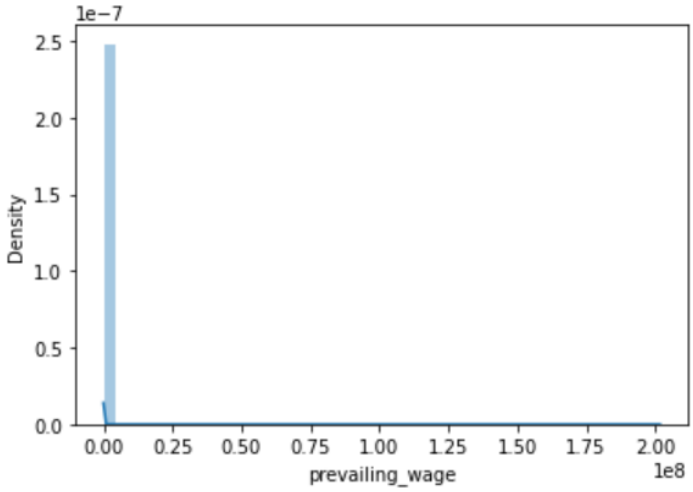

Fig - 3.9

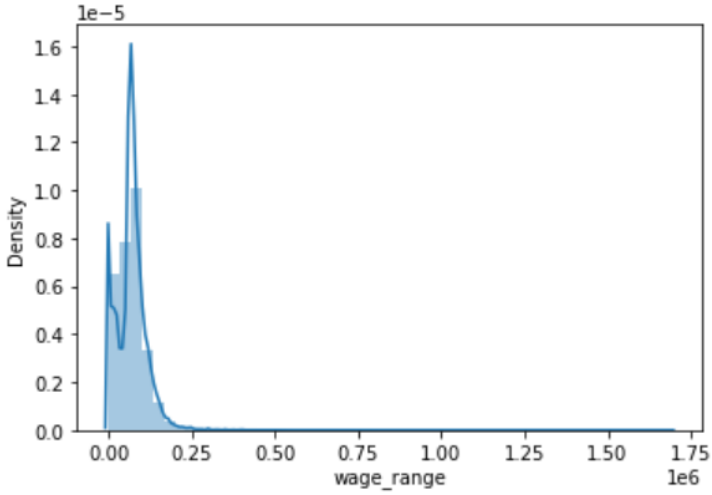

Fig - 3.10

3.4.2 Categorical variables

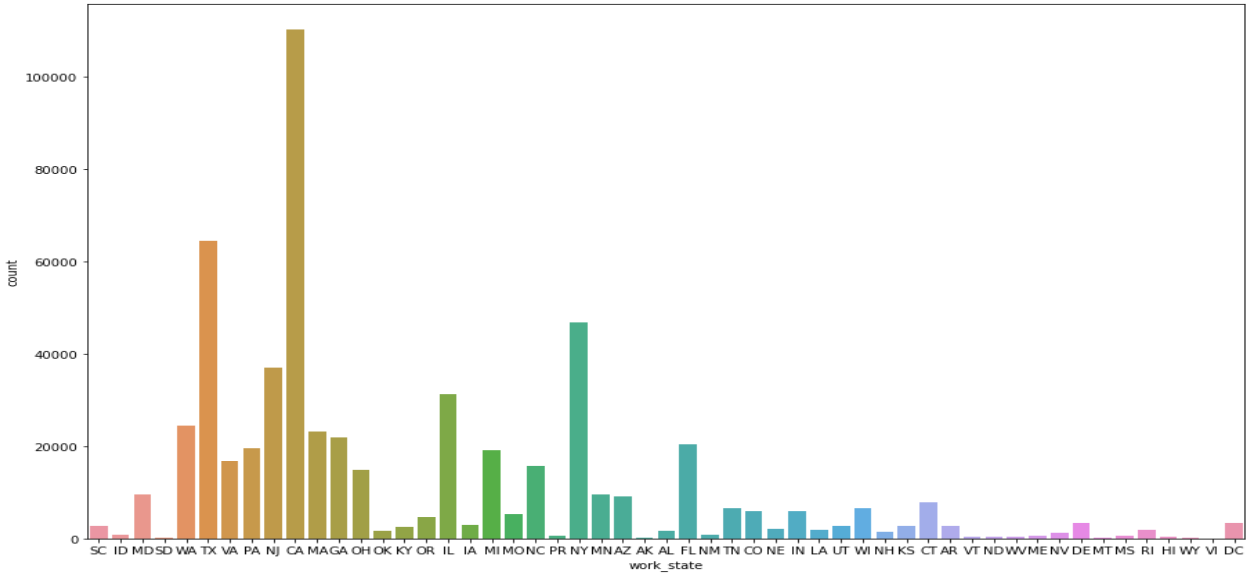

Fig - 3.11

Most of the employers are located in California.

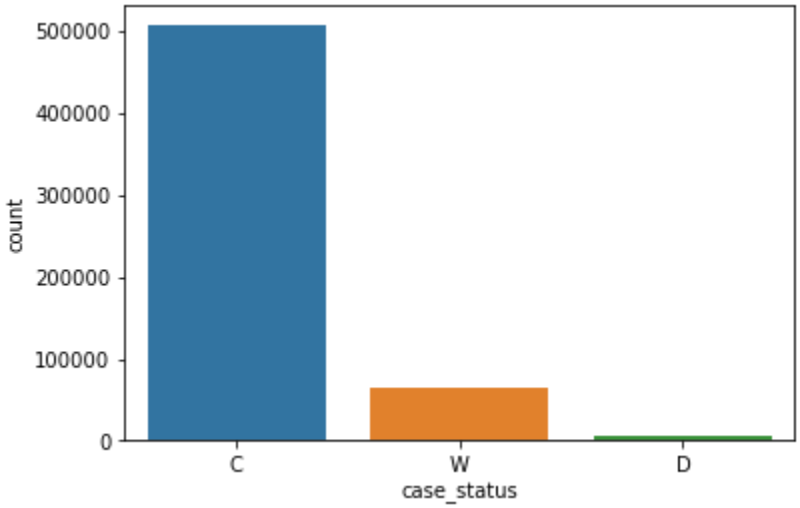

Fig - 3.12

There is an imbalance in the target variable classes. This imbalance can be handled using techniques such as SMOTE, Random Oversampling, and Random Undersampling.

3.4.3 Bivariate Analysis:

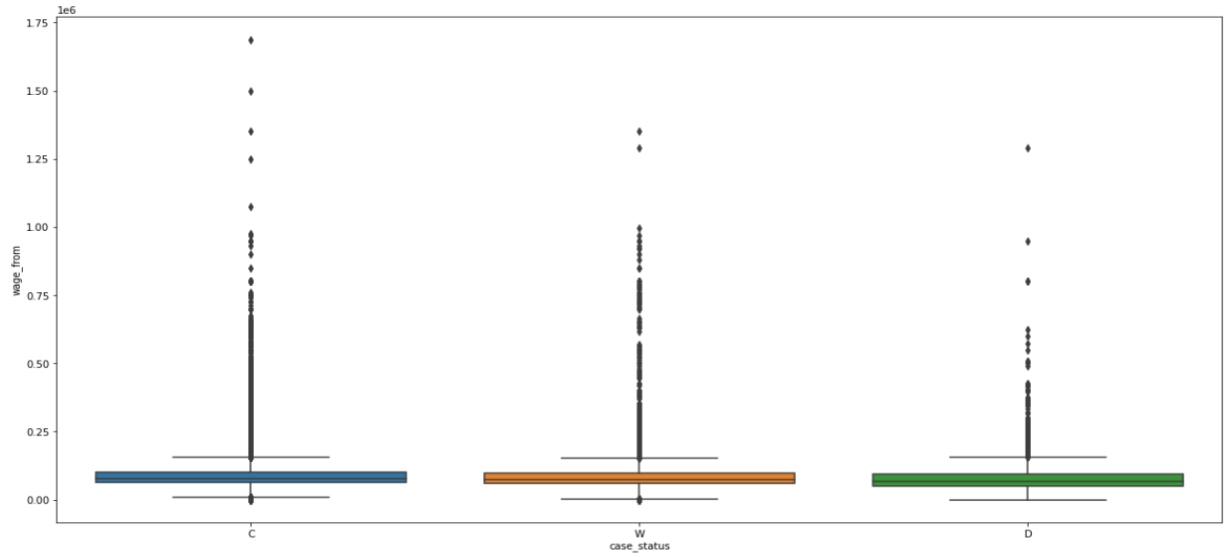

Fig - 3.13

The median salary offered for the approved, withdrawn, and denied applications are almost the same.

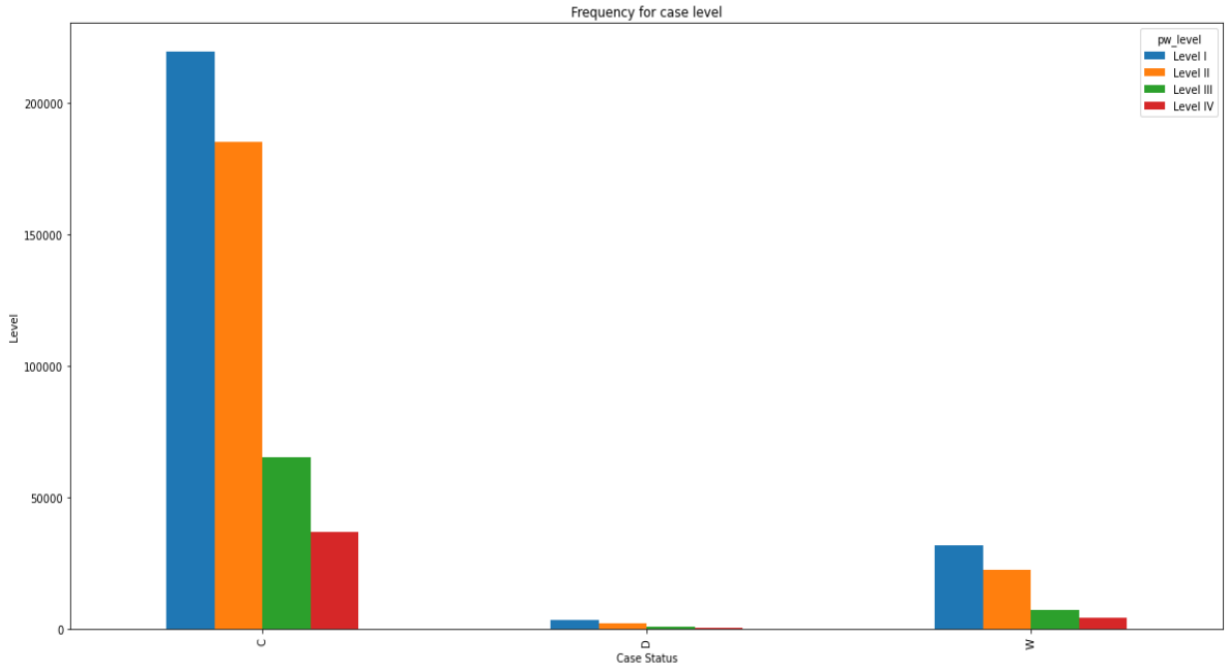

Fig - 3.14

Level - I applications have the most number of certified, denied, and withdrawn cases .

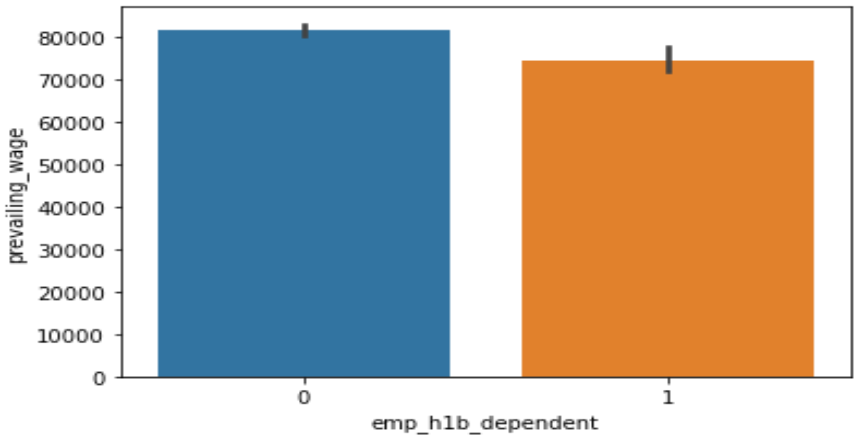

Fig - 3.15

Prevailing wages provided by H1-B dependent employers and independent employers are almost the same.

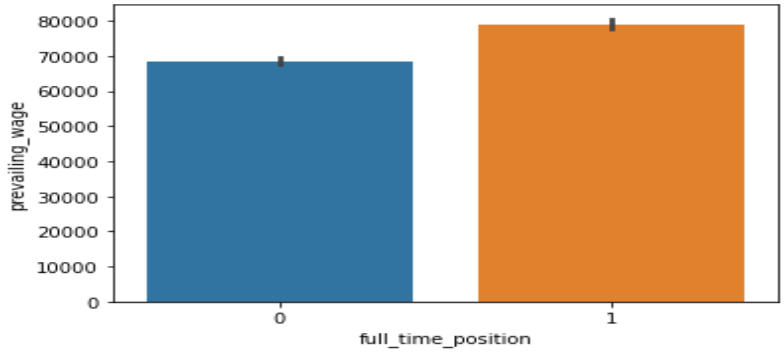

Fig - 3.16

Employees in full-time positions earn slightly more than part-time employees.

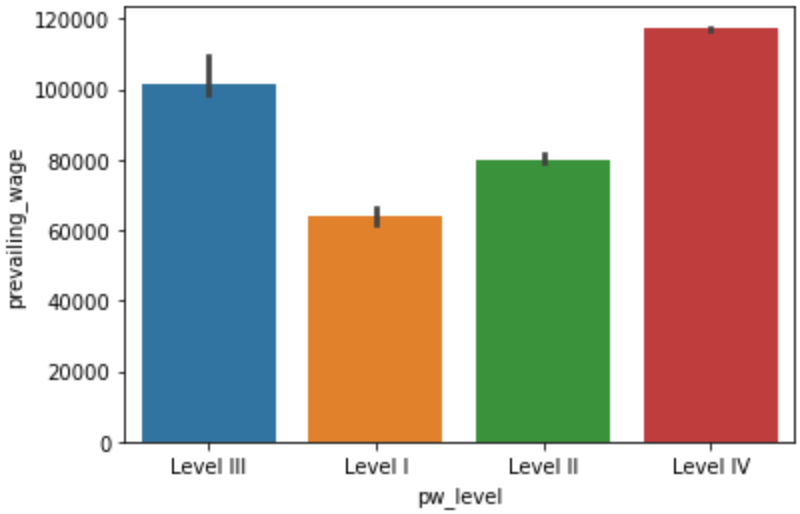

Fig - 3.17

There is a hierarchy in the wages and it is represented by the pw_level. Level-IV employees earn more than the employees of the other category.

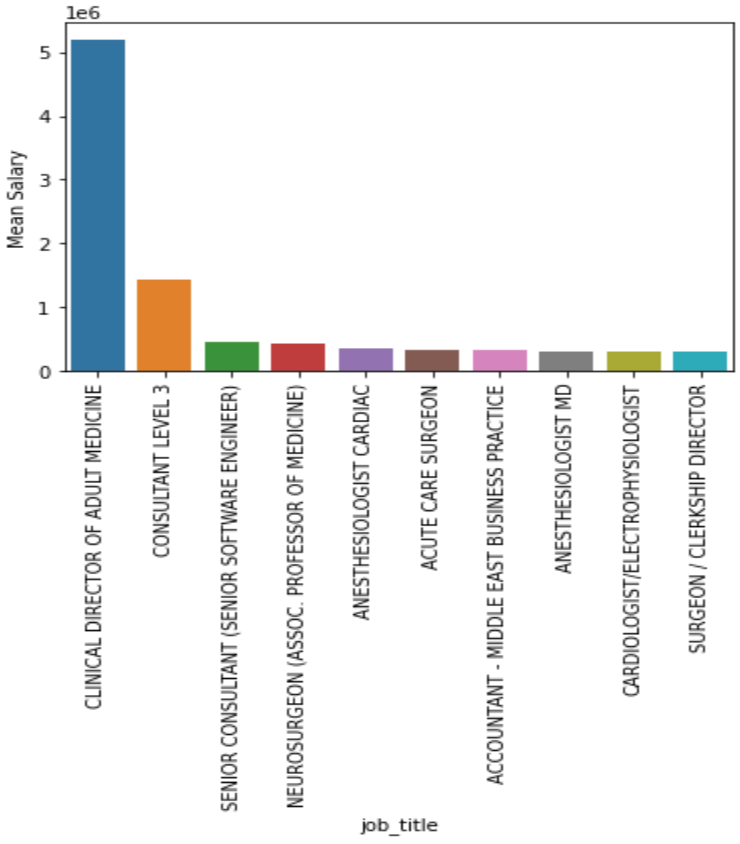

Fig - 3.18

The average prevailing wage of a Clinical Director of Adult Medicine is very high compared to other jobs.

job_title	ASSISTANT PROFESSOR	BUSINESS ANALYST	COMPUTER PROGRAMMER	COMPUTER SYSTEMS ANALYST	DEVELOPER	PROGRAMMER ANALYST	SENIOR SOFTWARE ENGINEER	SOFTWARE DEVELOPER	SOFTWARE ENGINEER	SYSTEMS ANALYST
emp_h1b_dependent										
0	4651	2965	1709	2106	605	9983	5392	6293	16125	2198
1	6	3933	5061	4105	4997	34254	1255	8875	10435	7928

Fig - 3.19

Many of the Assistant professors are H1-B visa dependents

3.5 Data Scaling:

Scaling is used to generalize the data points so that the distance between the data points is reduced. This is done to ensure all the data points are in the same scale. Scaling helps in making sure that no column gets a higher weightage than the other just because

	prevailing_wage	wage_range
count	579397.00	579397.00
mean	0.00	-0.00
std	1.00	1.00
min	-0.27	-1.51
25%	-0.06	-0.56
50%	-0.01	-0.02
75%	0.06	0.44
max	754.75	35.08

Fig - 3.20

3.6 Encoding:

In exploratory data analysis (EDA), encoding refers to the process of converting categorical variables into a numerical or binary format so that they can be analyzed using machine learning algorithms. This is necessary because most statistical and machine learning algorithms work only with numerical data.

```
cat['case_status'].unique()
```

```
array(['C', 'CW', 'D', 'W'], dtype=object)
```

CW and W both convey the same information to the model, we will use W to represent both.

```
cat['case_status'].replace('CW', 'W', inplace = True)
```

```
le = LabelEncoder()
cat['case_status'] = pd.DataFrame(le.fit_transform(np.array(cat['case_status'])))
```

```
cat['case_status'].unique()
```

```
array([0, 2, 1])
```

```
cat['pw_level'] = pd.DataFrame(le.fit_transform(np.array(cat['pw_level'])))
cat['pw_level'].unique()
```

```
array([0, 1, 2, 3])
```

Fig - 3.21

3.7 Imbalanced Data:

The target variable is the variable of interest that we want to predict or explain using the other variables in the dataset. It is also known as the dependent variable, response variable, or outcome variable.

```
x = dfs.drop('case_status', axis = 1)
y = dfs['case_status']
```

Fig - 3.22

3.8 Train Test Split:

After the independent and dependent variables are chosen, the data is split into training and testing datasets using the train_test_split function found in the scikit-learn library.

```
dfs.isnull().sum()
```

```
prevailing_wage         0
wage_range              0
case_status             0
full_time_position      0
pw_level                0
wage_unit               0
emp_h1b_dependent       0
emp_willful_violator    0
decision_time           0
dtype: int64
```

```
x = dfs.drop('case_status', axis = 1)
y = dfs['case_status']
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, train_size = 0.8, random_state = 100)
```

Fig- 3.23

Chapter 4 - Model Building

4.1 Base Model Selection

Since we are dealing with a classification problem, we have to build a classification model and evaluate the performance of the model accordingly. We have a few different classification algorithms such as:

1) Logistic Regression
2) K-NN
3) Decision Trees
4) Ensemble Techniques etc.

4.1.1 Logistic Regression

Logistic Regression is a classification algorithm that outputs the probability of an event happening. In our case, it outputs the probability of the visa getting approved or denied. We can use Logistic Regression from Scikit-Learn to classify the applications.

4.1.2 K-NN

K-NN works on the principle that birds of a feather flock together. In simple terms, it assumes that data points that are close together must be similar in one way or another. K-NN is a lazy learning algorithm, which means that it just stores the training data until the time comes for prediction. This makes K-NN extremely computationally intensive. Since we are working on a massive dataset, K-NN is not advised as it would not be very efficient.

4.1.3 Decision Trees

A decision tree is a graphical representation of all the outcomes for a given event, based on the given conditions. Decision Trees offer very good interpretability and efficiency, and can handle high-dimensional data. The only downside is that they are quite prone to overfitting. Comparing the options, we can see that a Decision Tree model would be a really good base model to work with.

4.2 Building the Base Model

```
log_reg = LogisticRegression().fit(xtrain, ytrain)
ypred_test_lr = log_reg.predict(xtest)
```

```
print(classification_report(ytest, ypred_test_lr))
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97    101548
           1       1.00      0.00      0.00      1354
           2       1.00      0.67      0.80     12976

    accuracy                           0.95    115878
   macro avg       0.98      0.56      0.59    115878
weighted avg       0.95      0.95      0.94    115878
```

Fig - 4.1

4.2.1 Classification Report

As mentioned in the previous section, we decided to use the Logistic Regression algorithm. From the classification report, we can see that the model has an overall accuracy of 95 percent.

The F1-scores for classes 0,1 and 2 are 0.97, 0.00, and 0.80 respectively. This tells us that our model is extremely good at predicting class 0 and class 2, but not at all good at predicting class 1. This is due to the huge imbalance in data as indicated by the support values for each class.

4.3 Fixing Imbalance in the Dataset

From the classification report, we can see that class 1 (Rejected) visa applications are very few in number compared to class 0 (Accepted) and class 2 (Withdrawn). This poses a serious problem as our model will not be able to predict if a data point belongs to class 1, if there are very few data points belonging to class 1. This is because Machine Learning models typically learn the patterns in the data and then predict the output. When there isn't enough data, predicting the output accurately becomes a problem.

To prevent the problems caused by data imbalance, we can use sampling techniques such as:

1) Random Oversampling
2) Random Undersampling
3) Synthetic Minority Oversampling Technique (SMOTE)

4.3.1 Random Oversampling

Random Oversampling is a technique used in machine learning to address the issue of class imbalance in datasets. Unlike random under-sampling, random oversampling involves increasing
The number of instances in the under-represented class match the number of instances in the under-represented class to balance the number of instances in each class.

When random oversampling is performed, new instances are generated by randomly selecting Instances from the under-presented class and duplicating them to increase their number. This is done until the number of instances in the under-represented class matches the number of instances in the over-represented class.

Advantages:

1. Improves Predictive Performance
2. No loss of information
3. Easy to implement

Disadvantages:

1. Overfitting
2. Time and resource-intensive
3. Can introduce bias.

4.3.2 Random Undersampling

Random under-sampling is a technique used in machine learning to address the issue of class imbalance in the datasets. Class imbalance occurs when one class of data is significantly more represented in the dataset than another class. Random under-sampling involves reducing the number of instances in the over-represented class to balance the number of instances in each class.

When random under-sampling is performed, a random subset of instances ia selected from the over-represented class, with the number of instances in the subset being equal to the number of instances in the under-represented class. This subset is then combined with all instances of the underrepresented class to create a balanced dataset.

Advantages:

1. Faster training time.
2. Reduced risk of overfitting
3. No introduction of bias

Disadvantages:

1. Loss of information
2. Reduced accuracy
3. Sensitivity to sampling ratio

4.3.3 SMOTE

Synthetic Minority Over Sampling Technique (SMOTE) is a technique used in machine learning To address the issue of class imbalance in the datasets. It involves generating synthetic examples of the under-represented class instead of duplicating existing examples, which helps to create a more diverse and representative dataset.

Advantages:

1. Creates synthetic data
2. Improved accuracy
3. Reduced bias

Disadvantages:

1. Can introduce noise
2. Resource-intensive
3. Sensitivity to parameters

4.4 Why SMOTE?

SMOTE addresses the limitations of random undersampling and oversampling methods. Random undersampling leads to the loss of potentially important information, while random oversampling increases the risk of overfitting.

In contrast, SMOTE generates synthetic samples from the minority class, rather than just duplicating or removing existing samples.

This approach ensures that the synthetic samples are based on the patterns and relationships present in the minority class, leading to a more diverse and representative dataset. As a result, SMOTE can help improve the performance of the models trained on imbalanced data.

Chapter 5 - Evaluating Model Performance

5.1 Model Performance Metrics

There are a lot of different metrics to measure model performance, such as:

1) Accuracy
2) Precision
3) Recall
4) F1-Score, etc.

5.1.1 Accuracy

Accuracy measures how well a model makes correct predictions overall. It is calculated as the number of correct predictions divided by the total number of predictions made.

It can be quite misleading when working with imbalanced datasets as it does not take class distribution into account. This means that accuracy does not give us a clear picture of how well the model performs in identifying minority classes.

5.1.2 Precision

Precision measures how often a model's positive predictions are correct. It is calculated as the number of true positive predictions divided by the total number of positive predictions (both True and False). In layman's terms, it measures how often the model is correct when it predicts that a data point belongs to a particular class.

Precision is very useful in understanding how many data points the model falsely predicted as positive. In the case of medical data, such as cancer, diabetes or heart failure prediction, False Positives cause unnecessary anxiety and result in further testing, increasing the cost borne by the patient.

5.1.3 Recall

Recall measures how often a model correctly identifies positive instances. It is calculated as the number of true positive predictions divided by the total number of actual positive predictions. Simply put, it measures how well the model can detect data points belonging to a particular class.

Recall is especially useful in understanding how many data points the model falsely predicted as negative. In the case of medical data, such as cancer, diabetes or heart failure prediction, False Negatives can delay treatment, leading to serious problems for the patient.

5.1.4 F1-Score

F1-Score measures the overall performance of the model, taking into account both precision and recall. It is calculated by taking the harmonic mean of precision and recall. It can be used instead of accuracy to understand the overall performance of the model when there is an imbalance in the data.

Since the F1-Score considers both precision and recall, a low precision or recall will automatically result in a low F1-Score. It is extremely useful when the cost of Falsely Predicting Positives and Falsely Predicting Negatives is high such as in medical applications.

5.2 Choosing the Right Metrics

Since the target column ‘case_status’ is highly imbalanced, it is better not to use accuracy as it can be misleading. Instead, we can use F1-Score, Precision, and Recall to gauge the performance of the model.

A False Positive in the case of H1B Visa Classification refers to cases where the model predicts that an application is approved, but in reality, the application is actually supposed to be rejected. This can lead to legal issues and affect future applications. Precision takes into account the number of False Positives and is useful when False Positive predictions can lead to problems in the real world.

A False Negative, on the other hand, refers to cases where the model predicts that an application is rejected, but in reality, the application is actually supposed to be approved. This can lead to delays, which we are trying to reduce. Recall takes into account the number of False Negatives and is useful when False Negative predictions can cause problems in the real world.

As a result, we will look at both the Recall and F1-Scores to measure the performance of our model.

5.3 Comparing Models

Comparing machine learning models is an essential step in developing a successful machine learning system. By selecting the best-performing model for a particular problem, we can improve the performance, efficiency, and interpretability of our system.

After SMOTE, the imbalance in the dataset has reduced significantly. The F1-score for class 1(Rejected) has also improved considerably. This is expected as our dataset now has a lot more data points belonging to class 1, making it easier for the model to learn the attributes that help the model classify when a data point belongs to class 1.

```
log_reg = LogisticRegression().fit(xtrain, ytrain)
ypred_test_lr = log_reg.predict(xtest)
```

```
print(classification_report(ytest, ypred_test_lr))
```

```
              precision    recall  f1-score   support

           0       0.65      0.79      0.71    101397
           1       0.62      0.68      0.65    101447
           2       1.00      0.69      0.82    101231

    accuracy                           0.72    304075
   macro avg       0.76      0.72      0.73    304075
weighted avg       0.76      0.72      0.73    304075
```

Fig - 5.1

```
log_reg = LogisticRegression().fit(xtrain, ytrain)
ypred_train_lr = log_reg.predict(xtrain)
```

```
print(classification_report(ytrain, ypred_train_lr))
```

```
              precision    recall  f1-score   support

           0       0.65      0.79      0.71    405394
           1       0.62      0.68      0.65    405344
           2       1.00      0.69      0.82    405560

    accuracy                           0.72   1216298
   macro avg       0.76      0.72      0.73   1216298
weighted avg       0.76      0.72      0.73   1216298
```

Fig - 5.2

Comparing the model's Recall and F1-Scores over the train and test data we can see that the scores are the same. From this, we can say that the model is not overfitting.
Tried different boosting techniques to improve the model's accuracy.

5.3.1 Adaboost Classifier

AdaBoost also called Adaptive Boosting is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is decision trees with one level which means Decision trees with only 1 split. These trees are also called Decision Stumps.

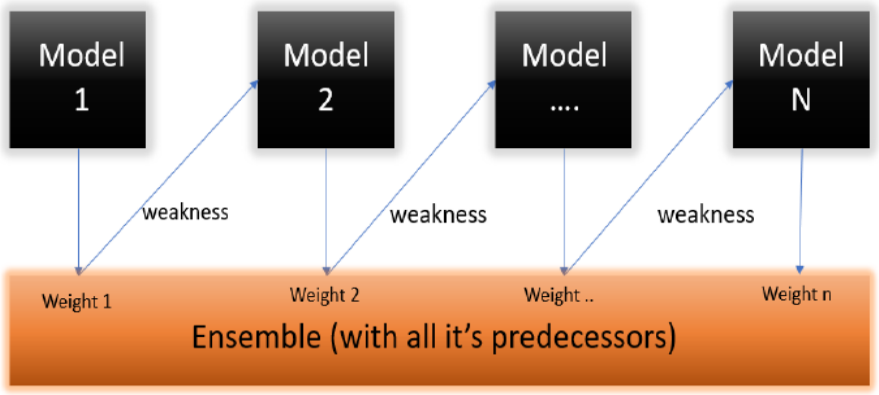

Fig - 5.3

What this algorithm does is that it builds a model and gives equal weights to all the data points. It then assigns higher weights to points that are wrongly classified. Now all the points which have higher weights are given more importance in the next model. It will keep training models until and unless a lower error is received.

```
from sklearn.ensemble import AdaBoostClassifier
```

```
ada = AdaBoostClassifier()
model_ada = ada.fit(xtrain,ytrain)
ypred_ada_test =  model_ada.predict(xtest)
```

```
print(classification_report(ytest, ypred_ada_test))
```

```
              precision    recall  f1-score   support

           0       0.75      0.89      0.82    101397
           1       0.70      0.73      0.72    101447
           2       0.99      0.75      0.85    101231

    accuracy                           0.79    304075
   macro avg       0.81      0.79      0.80    304075
weighted avg       0.81      0.79      0.80    304075
```

Fig - 5.4

The Adaboost model performs slightly better than the logistic regression model as seen above.

5.3.2 Gradient Boosting Classifier

In the case of Gradient Boosting Machines, every time a new weak learner is added to the model, the weights of the previous learners are frozen or cemented in place, left unchanged as the new layers are introduced. This is distinct from the approaches used in Ada Boosting where the values are adjusted when new learners are added.

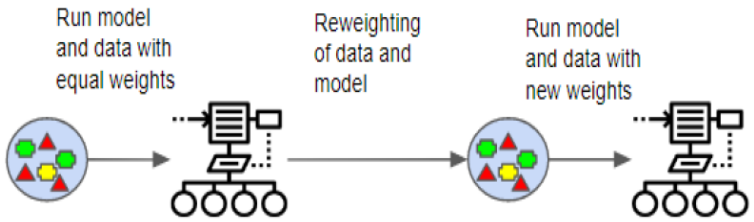

Fig - 5.5

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
gb = GradientBoostingClassifier()
gb_model = gb.fit(xtrain,ytrain)
ypred_gb_test = gb_model.predict(xtest)
```

```
print(classification_report(ytest, ypred_gb_test))
```

```
              precision    recall  f1-score   support

           0       0.77      0.88      0.82    101397
           1       0.78      0.76      0.77    101447
           2       0.99      0.86      0.92    101231

    accuracy                           0.84    304075
   macro avg       0.84      0.84      0.84    304075
weighted avg       0.84      0.84      0.84    304075
```

Fig - 5.6

With the gradient boosting classifier model, the Precision score of classes 1 and 2 has increased.

5.3.3 XGB Classifier

The XGB Classifier is based on Gradient Boosting. It combines the strengths of both decision trees and gradient boosting to achieve high prediction accuracy and efficiency.

It works by adding decision trees to the ensemble, with each new tree correcting the errors of the previous one. During training, the algorithm calculates the gradient of the loss function with respect to predictions of the model and uses this information to update the model parameters in a way that reduces loss.

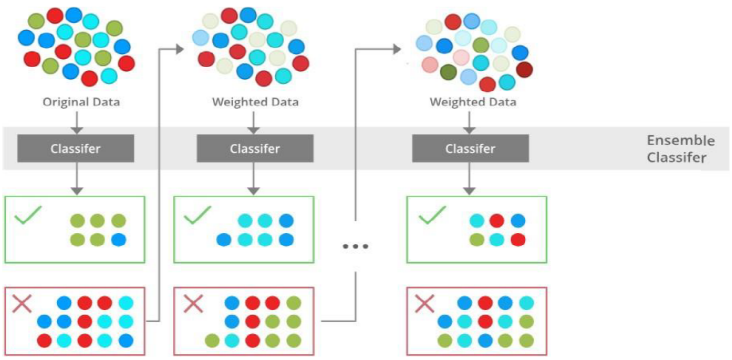

Fig - 5.7

```
from xgboost import XGBClassifier
```

```
xg = XGBClassifier()
xg_model = xg.fit(xtrain,ytrain)
ypred_xg_test = xg_model.predict(xtest)
```

```
print(classification_report(ytest, ypred_xg_test))
```

```
              precision    recall  f1-score   support

           0       0.80      0.92      0.86    101397
           1       0.83      0.80      0.81    101447
           2       0.99      0.88      0.93    101231

    accuracy                           0.87    304075
   macro avg       0.87      0.87      0.87    304075
weighted avg       0.87      0.87      0.87    304075
```

Fig - 5.8

With the XGBoost classifier the Recall and F1-Score has increased for all the classes.

5.3.4 DecisionTree Classifier

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature, each leaf node represents a class label and branches represent conjunctions of features that lead to those class labels.

The paths from the root to the leaf represent classification rules. The below diagram illustrates the basic flow of the decision tree for decision-making with labels.

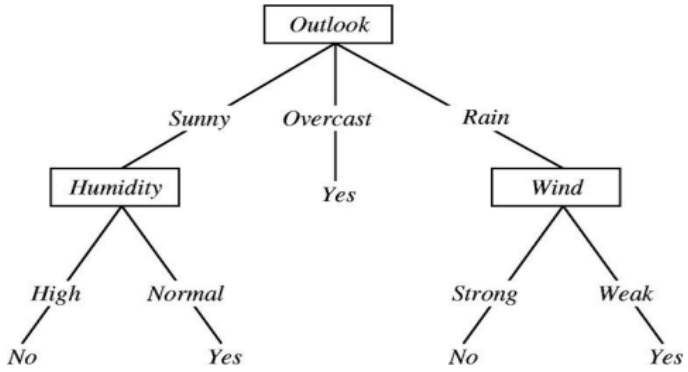

Fig - 5.9

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks.

```
from sklearn.tree import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier()
model_dt = dt.fit(xtrain,ytrain)
ypred_dt_test = model_dt.predict(xtest)
```

```
print(classification_report(ytest, ypred_dt_test))
```

```
              precision    recall  f1-score   support

           0       0.90      0.92      0.91    101397
           1       0.91      0.90      0.90    101447
           2       0.95      0.95      0.95    101231

    accuracy                           0.92    304075
   macro avg       0.92      0.92      0.92    304075
weighted avg       0.92      0.92      0.92    304075
```

Fig - 5.10

Recall and F1-Score have greatly increased for class 1 when using the Decision tree classifier.

5.3.5 Random Forest Classifier

Random Forest is an ensemble technique that combines multiple decision trees to reduce overfitting and improve prediction accuracy. Random Forest algorithms randomly sample a subset of data with replacement to create multiple training sets. These training sets are then used to train multiple decision trees where each tree is built by recursively partitioning the data into smaller subsets. Finally, the algorithm takes the majority vote of the predictions from all the decision trees.

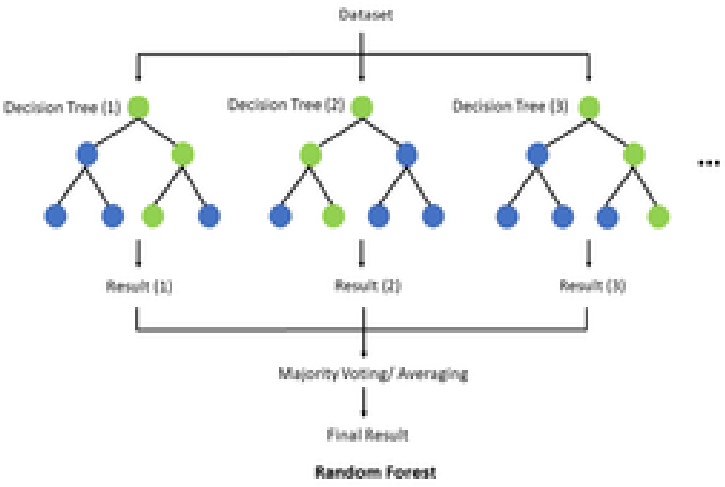

Fig - 5.11

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier()
model_rf = rf.fit(xtrain,ytrain)
ypred_rf_test = model_rf.predict(xtest)
```

```
print(classification_report(ytest, ypred_rf_test))
```

```
              precision    recall  f1-score   support

           0       0.91      0.94      0.92    101397
           1       0.92      0.92      0.92    101447
           2       0.98      0.94      0.96    101231

    accuracy                           0.93    304075
   macro avg       0.94      0.93      0.93    304075
weighted avg       0.94      0.93      0.93    304075
```

Fig - 5.12

There was a slight improvement in the scores and this is the best-performing model compared to the others.

5.4 Choosing the Best Model

From the classification reports, we can see that Random Forest Classifier performs the best in terms of Recall and F1-Score. This is to be expected as the Random Forest Classifier uses a combination of multiple Decision Trees, all working on different randomly picked subsets of data which reduces the impact of irrelevant and redundant features.

Random Forest models are also extremely robust, in that they are not so sensitive to noise and outliers in the data. This is why RandomForest models are less likely to overfit as compared to DecisionTree models.

Random Forest models are also non-parametric in nature, which means that they do not make any assumptions about the distribution of the data, unlike LogisticRegression. They are also highly scalable, which means that they can be used to train models on very large datasets with a lot of features.

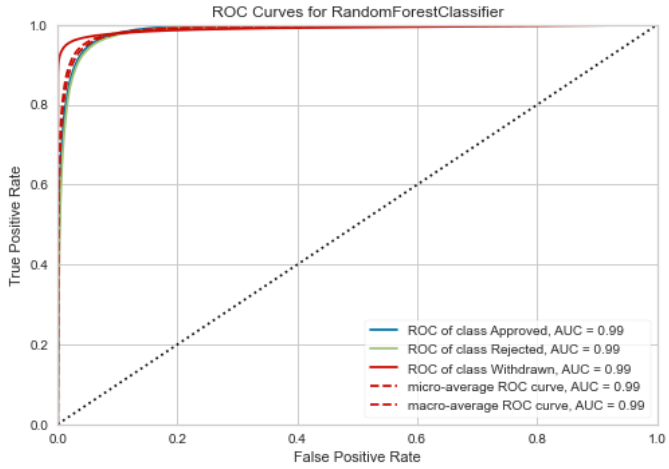

Fig - 5.13

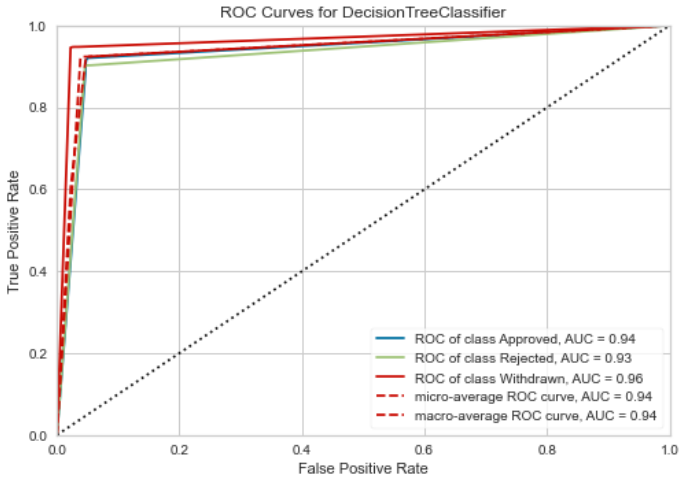

Fig - 5.14

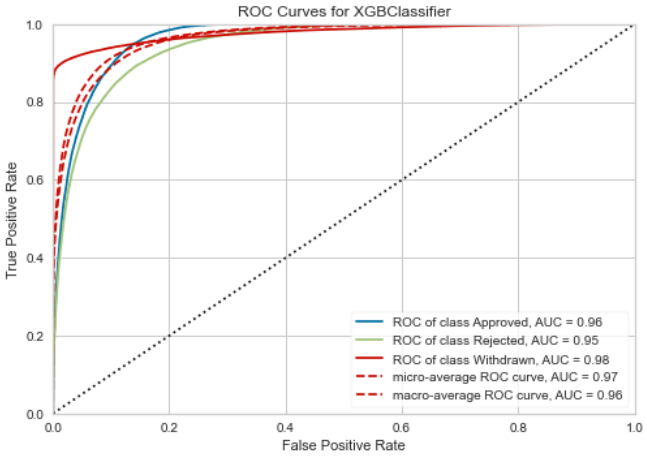

Fig - 5.15

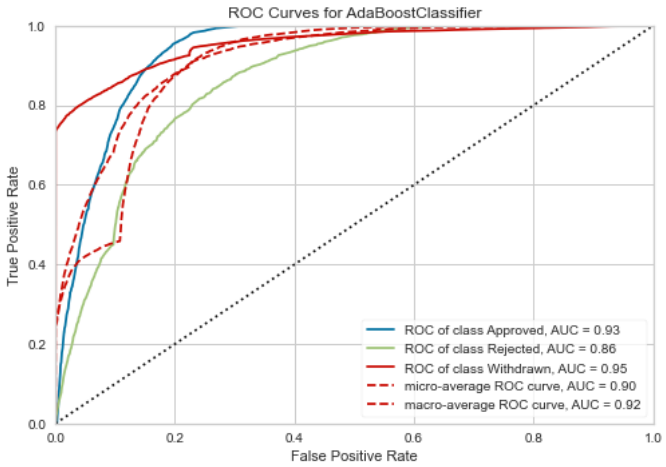

Fig - 5.16

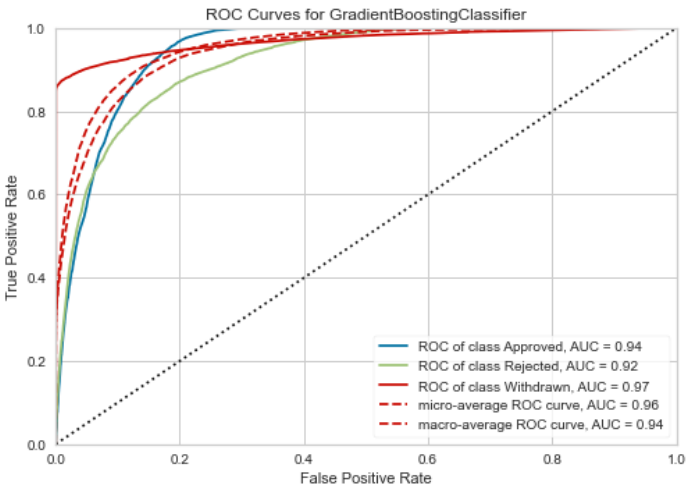

Fig - 5.18

5.5 Improving Performance

There are multiple techniques to improve the performance of our model such as:

1) Cleaning the data
2) Fixing Data Imbalance
3) Feature Selection/Elimination
4) Scaling the numerical variables
5) Trying different models
6) Hyperparameter Tuning
7) Cross-Validation
8) Pruning etc.

Since we already have a model that performs well, we will focus only on Hyperparameter Tuning.

5.5.1 Hyperparameter Tuning

Hyperparameters are values that are set before training the model. These values cannot be learned from the training data. They are used to control the behavior and performance of the model.

Hyperparameter Tuning is the process of finding the optimal set of hyperparameters for a model. The optimal values for hyperparameters depend on the dataset and the actual problem itself. Thus, tuning them requires a lot of experimentation and trial and error.

Multiple methods can be used to tune hyperparameters, but we will focus only on GridSearch and RandomSearch. GridSearch tests all possible combinations of hyperparameter values within a defined range, whereas RandomSearch randomly samples hyperparameter values from a defined range.

Unfortunately, due to the massive size of the dataset and a lack of resources, GridSearchCV crashed our kernel multiple times. However, we were able to run RandomSearchCV and get some values for hyperparameters.

Tuning our Random Forest model with these values, the performance of our model actually reduced. As a result, we decided to use the untuned model as it gives us really good performance already.

Chapter 6 - Conclusion

6.1 Conclusion

Selecting the best model for a machine learning problem is crucial in achieving high performance, efficiency, and interpretability. In this report, we compared various machine learning models such as Adaboost, Gradient Boosting, XGBoost, Decision Tree, and RandomForest classifiers, and found that the RandomForest classifier performed the best in terms of Recall and F1-Score.

To further improve the performance of the model, we explored techniques such as data cleaning, fixing data imbalance, feature selection/elimination, scaling, trying different models, hyperparameter tuning, cross-validation, and pruning. Among these, we focused on hyperparameter tuning and cross-validation, as they are essential in finding the optimal set of hyperparameters and ensuring that the model is not overfitting to the training data.

Overall, selecting the best machine learning model and optimizing its hyperparameters took a lot of experimentation and trial and error. However, by using appropriate techniques and methods, we were able to achieve high performance, efficiency, and interpretability.

6.2 Future Work

Due to limited time and resources, we were not able to do much in terms of improving our model. Given time and resources, we might have been able to achieve a model with a much higher Recall and F1-Score.

Future Work would include reducing the time taken by the model to make a prediction and reducing the amount of computational power required for running the model. This would make it easier for us to deploy the model on mobile phones and tablets.

Another idea is to have the model deployed online, with visa applicants being able to check whether their application is likely to be rejected or accepted and make changes to their application as required.

References

1) *Alsafy, Baidaa & Mosad, Zahoor & Mutlag, Wamidh. (2020). Multiclass Classification Methods: A Review.*

2) *Yogi, Abhishek & Dey, Ratul. (2022). CLASS IMBALANCE PROBLEM IN DATA SCIENCE: REVIEW. International Research Journal of Computer Science. 9. 56-60. 10.26562/irjcs.2021.v0904.002.*

3) *Chawla, Nitesh & Bowyer, Kevin & Hall, Lawrence & Kegelmeyer, W.. (2002). SMOTE: Synthetic Minority Over-sampling Technique. J. Artif. Intell. Res. (JAIR). 16. 321-357. 10.1613/jair.953.*

4) *Trimbach, S.. (2017). Giving the market a microphone: Solutions to the ongoing displacement of U.S. Workers through the H1B visa program. Northwestern Journal of International Law and Business. 37. 275-300.*

5) *A, Pranav & M, Prasanth & T, Suthan. (2019). Performing Examination on H1B Visa using Data Analytics Techniques to Enhance the Employability Skills. International Journal of Engineering and Advanced Technology. 9. 7506-7509. 10.35940/ijeat.A3128.109119.*

6) *Obi, Jude. (2023). A Comparative Study of Several Classification Metrics and Their Performances on Data. World Journal of Advanced Engineering Technology and Sciences. 8. 308-314. 10.30574/wjaets.2023.8.1.0054.*