Luqmaan Hakeem

125018043@sastra.ac.in

Abstract

This study explores the application of logistic regression in forecasting the results of Dota 2 games, utilizing a comprehensive collection of in-game data from the Kaggle Dota 2 Win Prediction dataset

Dota 2 win prediction

Using machine learning algorithms

|  |  |  |
| --- | --- | --- |
| **S.no** | **Sections** | **Page No.** |
| **1** | **Abstract** | **2** |
| **2** | **Introduction** | **3** |
| **3** | **Related work and Background** | **4** |
| **4** | **Methodology** | **10** |
| **5** | **Implementation and discussion** | **11** |
| **6** | **Result** | **20** |
| **7** | **Learning Outcome** | **23** |
| **8** | **Conclusion** | **24** |

**Table of contents**

**Abstract**

This study explores the application of logistic regression in forecasting the results of Dota 2 games, utilizing a comprehensive collection of in-game data from the Kaggle Dota 2 Win Prediction dataset. The dataset contains individual player details like hero picks, K/D/A stats, gold, XP, as well as team actions like placing wards and controlling objectives. Logistic regression, a model for classification based on probabilities, is used to predict the chances of either the Radiant or Dire team winning using these characteristics. Models such as Random Forest and SVM are also employed to compare the performances between each models. Methods used to enhance model performance involve feature engineering, which includes gathering player stats and identifying key in-game occurrences like first blood and tower kills. The straightforward nature and ease of understanding logistic regression offer clear understanding of how various game elements impact the likelihood of a victory, pinpointing crucial factors influencing match results. Evaluation of the model is carried out using metrics like accuracy, precision, and recall, while cross-validation guarantees the reliability. This project shows how machine learning can be used to give instant win forecasts depending on the situation in the game.

**Introduction:**

The whole project is to predict the outcome of a Dota 2 game given all the game’s characteristics up to a certain point in the game. There are two teams in Dota 2 games: Radiant and Dire. The goal is to evaluate the chances team Radiant victory. Game’s data is presented with numeric and categorical features, event logs, time series, etc. Working with game’s data and learning to see patterns and regularities can prove very useful in further work.

In a regular Dota 2 game each of two teams - Radiant and Dire - consists of 5 players. For each game player chooses a hero and consequently a role. Dota 2 is a team game so the team’s composition matters. The map contains bases of each team (fountain) and 3 lanes on each side, shops, Roshan's lair and other elements.

The dataset which is taken a kaggle competition consists of rich variety of features such as kills , time series data like gold abilities used etc. It provides us with historical contexts which can be beneficial when it comes to forecasting and training. The dataset includes the following csv files which have specific usecases

* train\_features.csv: basic training set, prepared by organizers
* test\_features.csv: basic test set, prepared by organizers
* train\_targets.csv: answers for games from the training set (whether Radiant eventually won)
* train\_matches.jsonl: raw training data
* test\_matches.jsonl: raw test data

Since the dataset is already pre-processed there is no need to do additional data preprocessing. Further implementation is discussed in the implementation section. The accuracy which was achieved in the end using logistic regression is 81%.

**Related Work**

This project is inspired from the famous kaggle competition from mlcourse.ai: <https://www.kaggle.com/competitions/mlcourse-dota2-win-prediction>

References used to implement the project:

* <https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html>
* <https://github.com/cjlin1/liblinear>
* <https://oneapi-src.github.io/oneDAL/daal/algorithms/optimization-solvers/solvers/stochastic-average-gradient-accelerated-method.html>

**Background:**

The following is a basic understanding of different concepts used in the project:  
**Principal Component Analysis (PCA)**

**Working Principle**: PCA is a statistical technique used for dimensionality reduction while preserving as much variance as possible in high-dimensional datasets. The key steps involved in PCA are:

1. **Standardization**:
   * PCA starts by standardizing the dataset. This means that we scale the data such that it has a mean of zero and a standard deviation of one. This is crucial for PCA since it is sensitive to the variances of the initial variables.
2. **Covariance Matrix Computation**:
   * After standardization, PCA computes the covariance matrix to understand how variables relate to each other. The covariance matrix expresses the correlations between different features in the dataset.
3. **Eigenvalue and Eigenvector Calculation**:
   * The next step is to calculate the eigenvalues and eigenvectors of the covariance matrix.
   * **Eigenvalues** indicate the amount of variance captured by each principal component.
   * **Eigenvectors** represent the direction of the axes (the principal components) in the transformed space.
4. **Sorting Eigenvalues and Eigenvectors**:
   * Eigenvalues are sorted in descending order, and the corresponding eigenvectors are arranged accordingly. The top eigenvectors (the ones associated with the largest eigenvalues) will form the new feature space.
5. **Selecting Principal Components**:
   * Based on the sorted eigenvalues, we select a subset of the top kkk eigenvectors to form a new matrix of features (principal components). The number of components is usually chosen based on the desired level of explained variance.
6. **Transforming the Data**:
   * Finally, the original data is projected onto the new feature space formed by the selected principal components. This results in a reduced-dimensionality dataset, capturing most of the variance in the original data.

### Logistic Regression

**Working Principle**: Logistic Regression is a statistical method for binary classification that models the probability of a binary outcome based on one or more predictor variables. It employs the logistic function (sigmoid function) to map predicted values to probabilities.

1. **Model Representation**:
   * The logistic regression model predicts the probability
   * P(Y=1∣X) of the binary target variable Y given feature vector X. The model is represented as: P(Y=1∣X)=1/(1+e^(−(β0+β1X1+β2X2+...+βnXn))
   * Here, β0​ is the intercept and β1,β2,...,βn are the coefficients for the features X1,X2,...,Xn​.
2. **Loss Function**:
   * Logistic regression uses the log loss (cross-entropy loss) as the loss function, which measures the performance of the model by comparing the predicted probabilities with the actual class labels.
3. **Optimization**:
   * To find the best-fitting model, logistic regression applies optimization algorithms (like gradient descent) to minimize the loss function by adjusting the coefficients (β\betaβ).
4. **Classification**:
   * The output probability is converted into class labels (0 or 1) by applying a threshold (usually 0.5). If the predicted probability is greater than or equal to 0.5, the instance is classified as 1; otherwise, it is classified as 0.

### Regularization

**Working Principle**: Regularization is a technique used to prevent overfitting in machine learning models by adding a penalty term to the loss function. It discourages overly complex models by constraining the values of the coefficients.

1. **Types of Regularization**:
   * **L1 Regularization (Lasso)**: Adds the absolute value of the coefficients as a penalty term to the loss function. This can lead to sparse models, where some coefficients are exactly zero, effectively performing feature selection.
     1. Loss=Log Loss+λ∑∣βj∣
   * **L2 Regularization (Ridge)**: Adds the squared values of the coefficients as a penalty term. This encourages smaller coefficient values but does not produce sparse models. Loss=Log Loss+λ∑βj^2​
   * **Elastic Net**: Combines both L1 and L2 penalties.
2. **Regularization Strength**:
   * The regularization strength is controlled by a hyperparameter (often denoted as λ or C). A higher value of λ increases the penalty, leading to simpler models, while a lower value allows for more complexity.

### Solvers Used in Logistic Regression

**1. liblinear**:

* **Working Principle**: liblinear is an optimization solver specifically designed for logistic regression and linear Support Vector Machines. It uses a coordinate descent algorithm to optimize the objective function.
* **Advantages**:
  + Suitable for small to medium-sized datasets.
  + Efficient for problems with a large number of features.
* **Limitations**:
  + Not well-suited for multi-class problems unless combined with a one-vs-all approach.
  + It does not handle very large datasets as efficiently as some other solvers.

**2. Saga**:

* **Working Principle**: saga is an extension of the Stochastic Average Gradient (SAG) algorithm and is designed for large-scale linear models. It is suitable for both L1 and L2 regularization.
* **Advantages**:
  + Can handle large datasets efficiently.
  + Suitable for both binary and multi-class classification.
  + Allows for a wider variety of regularization techniques (L1, L2, and Elastic Net).
* **Limitations**:
  + May require more iterations to converge compared to other solvers, especially for smaller datasets.

. Before we get into the results let us define the basics of the two models mentioned above.  
**Random Forest**

**Random Forest** is an ensemble learning method that combines multiple decision trees to improve classification accuracy and control overfitting. It constructs a multitude of decision trees during training and outputs the mode of the classes (classification) of the individual trees.

* **Working Principle**:
  1. **Bootstrap Sampling**: For each tree in the forest, a random sample of the dataset is selected with replacement (bootstrap sample).
  2. **Random Feature Selection**: At each split in the tree, a random subset of features is considered for splitting the nodes. This reduces variance and helps in improving model robustness.
  3. **Voting**: Once all trees are built, the final prediction is made based on the majority vote from all trees.

### Support Vector Machine (SVM)

**Support Vector Machine** is a supervised learning algorithm that constructs a hyperplane or set of hyperplanes in a high-dimensional space for classification. The goal is to find the hyperplane that maximizes the margin between the classes.

* **Working Principle**:
  1. **Hyperplane Creation**: SVM identifies the hyperplane that best separates the classes in the feature space.
  2. **Margin Maximization**: The algorithm aims to maximize the distance (margin) between the hyperplane and the nearest data points from either class, known as support vectors.
  3. **Kernel Trick**: To handle non-linear data, SVM can use kernel functions (like linear, polynomial, RBF) to transform the input space into a higher-dimensional space where a linear separator can be found.

### Methodology

**1. Feature Engineering**:  
Feature engineering plays a crucial role in improving model performance. Aggregated player statistics, such as team-wide sums of kills, assists, and last hits, are used to represent overall team strength. Specific event-based features such as "first blood claimed" and "roshans killed" are also treated as important predictive features. Time-series data (e.g., gold, XP progression) are aggregated into summary statistics (mean, variance) to capture game momentum.

**2. Experimental Design**:  
The dataset is split into training and testing sets to ensure unbiased evaluation. Cross-validation is employed during training to minimize overfitting. The logistic regression model is trained using key game features to predict match outcomes, with hyperparameter tuning to optimize model performance.

Model evaluation focuses on several metrics:

* **Accuracy**: Measures the overall correctness of the predictions.
* **Precision and Recall**: Used to assess the model's ability to correctly classify wins and losses.
* **AUC-ROC Curve**: Evaluates the model’s ability to distinguish between Radiant and Dire wins.

Finally, feature importance analysis is conducted using the coefficients of the logistic regression model to interpret which in-game factors most influence match outcomes. This helps in identifying the most impactful aspects of gameplay (e.g., gold accumulation, hero selection) in determining the likelihood of a team’s victory.

**3. Environments and tools used:**

This project uses sklearn primarily for model training and prediction and matplotlib for plotting graphs and numpy for creating dataframes. Python is used to implement the project. The code is executed in the kaggle environment.

**Implementation and discussion:**

Since the dataset is already in numerical values there is no need to preprocess them. I created dataframes from the given dataset

|  |
| --- |
| df\_train\_features = pd.read\_csv(os.path.join(PATH\_TO\_DATA,'train\_features.csv'), index\_col='match\_id\_hash') df\_train\_targets = pd.read\_csv(os.path.join(PATH\_TO\_DATA, 'train\_targets.csv'), index\_col='match\_id\_hash') |

Then I split the dataframes where 30% will be used for validation and the remaining 70% of the data will be used for training

|  |
| --- |
| x\_train, x\_valid, y\_train, y\_valid = train\_test\_split(X, y, test\_size=0.3,random\_state=17) |

### 

### To start, I scaled the feature set using StandardScaler to normalize the data, ensuring that each feature has a mean of 0 and a standard deviation of 1. This step is essential since logistic regression can be sensitive to varying scales of input features. After scaling, I applied Principal Component Analysis (PCA) with 50 components. PCA is a dimensionality reduction technique that helps in capturing the most significant patterns in the data while reducing the number of features, which can improve the performance of the model.

### For the classification task, I used the LogisticRegression model from the sklearn library. The logistic regression model was initialized with a regularization strength C of 1, using the L2 penalty(Ridge regression). L2 regularization helps prevent overfitting by adding a penalty to large coefficients. The maximum number of iterations was set to 100 to ensure the model converges, while the solver selected was liblinear, which is efficient for smaller datasets. The random\_state ensures that the results are reproducible, while n\_jobs=-1 allows the model to use all available processors for faster computation. The verbose parameter was set to 1 to display progress logs during training.

### Once the logistic regression model was initialized, I fit it using the reduced feature set X\_pca obtained from the PCA transformation. Finally, I computed the accuracy of the model on the training data using the score() method, which returned an accuracy score of the model based on the transformed data. The comparison between the accuracy of different experiments will be explored in the result section.

|  |
| --- |
| #Logistic Regression + Ridge Regression with PCA C = 1 penalty = 'l2' max\_iter = 100 solver = 'liblinear' random\_state = 17 n\_jobs = -1 verbose = 1  clf\_lr = LogisticRegression(C=C,  penalty=penalty,  max\_iter=max\_iter,  random\_state=random\_state,  verbose=verbose,  n\_jobs=n\_jobs,  solver=solver) scaler = StandardScaler() X\_scaled = scaler.fit\_transform(X) pca = PCA(n\_components=50) X\_pca = pca.fit\_transform(X\_scaled)  clf\_lr.fit(X\_pca, y)   accuracy\_pca = clf\_lr.score(X\_pca, y) print(accuracy\_pca) |

### Fitting the model with the data without PCA is quite similar to the above approach instead we just use the train data which was split with ridge regression.

### More about the solvers, I primarily used liblinear and saga solvers for the logistic regression model. Liblinear uses Coordinate Descent or Trust Region Newton method to optimize the objective function.This makes it suitable for medium sized dataset. While saga(Stochastic Average Gradient Augmented) is a relatively new solver and it uses Stochastic Gradient Descent. I used saga to experiment with implementing the model with pca and without pca to compare the results and I found out that without pca worked much better and saga solver did do better than liblinear. I demonstrated this by shuffle splitting the data and cross validating the different subsets.

### cv = ShuffleSplit(n\_splits=5, test\_size=0.3, random\_state=17)

### This splits the data into 5 different train-validation pairs (with 30% of the data reserved for validation each time). The following implementation is quite similar to what we already did but it is different in terms of using different iterations and instead of fitting we just find the cross validation score

|  |
| --- |
| #calculating ROC-AUC for each split C = 1 penalty = 'l2' max\_iter = 50 solver = 'liblinear' random\_state = 17 n\_jobs = -1 verbose = 1  clf\_lr\_1 = LogisticRegression(C=C,  penalty=penalty,  max\_iter=max\_iter,  random\_state=random\_state,  verbose=verbose,  n\_jobs=n\_jobs,  solver=solver)  cv\_scores\_lr1 = cross\_val\_score(clf\_lr\_1, X, y, cv=cv, scoring='roc\_auc') |

### Similarly I implemented the saga solver to each split and calculated the roc\_auc score for it by reducing the regularization strength to 0.1

|  |
| --- |
| C = 0.1 penalty = 'l2' solver = 'saga' max\_iter = 150 random\_state = 17 n\_jobs = -1 verbose = 1 class\_weight = 'balanced'  clf\_lr\_2 = LogisticRegression(C=C,  penalty=penalty,  max\_iter=max\_iter,  random\_state=random\_state,  verbose=verbose,  n\_jobs=n\_jobs,  class\_weight=class\_weight,  solver=solver)  # calculate ROC-AUC for each split cv\_scores\_lr2 = cross\_val\_score(clf\_lr\_2, X, y, cv=cv, scoring='roc\_auc') |

### After calculating these scores I compared the two solvers and found out that cv\_score\_lr2 performs well.

|  |
| --- |
| array([ True, False, False, True, False]) |

### For further experimentation I thought of extending the data set by adding new columns which might help improve the accuracy of the model. There is a dataset for the match and using this dataset I was able to calculate the tower kills for each team and did this for every match in the dataset. The match data set had information of the player at each point of time(such as kills, deaths, assists etc) and to demonstrate this I plotted a line plot to show the variation of gold that a player had throughout the match. The new columns displays the number of tower kills of each team and the diff\_tower column displays how good or bad they performed in destroying the towers with respect to the other team.



### After extending the dataframe we will calculate the cross validation score for this data using the model which uses saga as it’s solver. Then we compare the results of the model without extending with the model which used the extended data and we found out that the model performed the same



Moving on I implemented the model to predict on the actual test dataset using the model which used saga solver with C as 0.1.

|  |
| --- |
| clf\_lr\_2.fit(df\_train\_features\_extended.values, y) df\_lr\_final= pd.DataFrame(  {'radiant\_win\_prob': clf\_lr\_2.predict\_proba(df\_test\_features\_extended.values)[:, 1]},  index=df\_test\_features.index, ) df\_lr\_final.to\_csv('lr\_pred.csv')  df\_lr\_final.head() |

Just for comparison I also implemented random forest and SVM to demonstrate that logistic regression works better compared to these two models

Moving on to the implementation of random forest and svm:

|  |
| --- |
| #Random forest from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score, roc\_auc\_score, f1\_score from sklearn.model\_selection import cross\_val\_score, train\_test\_split rf\_clf = RandomForestClassifier(n\_estimators=100,  max\_depth=15,  min\_samples\_split=5,  random\_state=17)  rf\_clf.fit(x\_train, y\_train) y\_pred\_rf = rf\_clf.predict(x\_valid)  accuracy\_rf = accuracy\_score(y\_valid, y\_pred\_rf) roc\_auc\_rf = roc\_auc\_score(y\_valid, rf\_clf.predict\_proba(x\_valid)[:, 1]) # For binary classification f1\_rf = f1\_score(y\_valid, y\_pred\_rf)  print(f"Random Forest Accuracy: {accuracy\_rf}") print(f"Random Forest ROC-AUC: {roc\_auc\_rf}") print(f"Random Forest F1 Score: {f1\_rf}") |

### n\_estimators: The number of trees in the forest and I initialized it with 100 trees

### max\_depth: The maximum depth of each tree which is 15.

### min\_samples\_split: The minimum number of samples required to split an internal node is 5.

### After declaring these parameters I trained the model and fit the model with the train datasets and predicted with the validation set. The results will be explored in the next section.

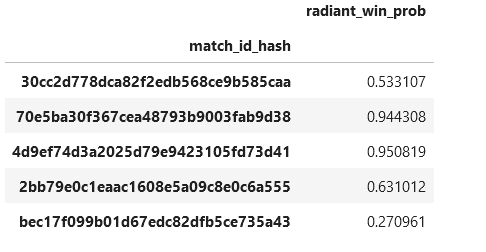
**For SVM I had to scale the data because the model tends to be sensitive to the feature scale and then I split the dataset. I used the ‘rbf’ kernel to implement the SVM with regularization strength as 1. Rbf kernel is particularly powerful for handling non-linear classification tasks because it can map the data into a higher-dimensional space where it becomes easier to separate classes with a linear boundary. It is a type of gaussian kernel.**

**The results of this model will be explored in the next section.**

|  |
| --- |
| from sklearn.svm import SVC from sklearn.preprocessing import StandardScaler  # We need to scale the data for SVM as it's sensitive to the feature scale scaler = StandardScaler() X\_scaled = scaler.fit\_transform(X)  X\_train\_scaled, X\_valid\_scaled, y\_train, y\_valid = train\_test\_split(X\_scaled, y, test\_size=0.3, random\_state=17)  # Instantiate the SVM classifier with RBF kernel svm\_clf = SVC(C=1.0, kernel='rbf', gamma='scale', probability=True, random\_state=17)  svm\_clf.fit(X\_train\_scaled, y\_train)  y\_pred\_svm = svm\_clf.predict(X\_valid\_scaled)  accuracy\_svm = accuracy\_score(y\_valid, y\_pred\_svm) roc\_auc\_svm = roc\_auc\_score(y\_valid, svm\_clf.predict\_proba(X\_valid\_scaled)[:, 1]) f1\_svm = f1\_score(y\_valid, y\_pred\_svm)  print(f"SVM Accuracy: {accuracy\_svm}") print(f"SVM ROC-AUC: {roc\_auc\_svm}") print(f"SVM F1 Score: {f1\_svm}") |

### 

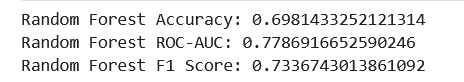
### Results

The final result displays the probability that radiant won in a given match\_hash\_id using logistic regression (saga solver) and here is the results I got:  


Now let us discuss the results we got in multiple experiments that I implemented to improve the accuracy:

* **(a) Logistic Regression with PCA**:
  + **Settings**: Used PCA to reduce dimensions to 50.
  + **Results**: Achieved accuracy of
* **(b) Logistic Regression without PCA**:
  + **Settings**: All 245 features used.
  + **Results**: Achieved accuracy of 70%
* **(c) Logistic Regression + Lasso Regularization without PCA**:
  + **Settings**: Lasso with alpha=1.0.
  + **Results**: Accuracy achieved 0.7254106627890032 or 72.5%
* **(d) Logistic Regression + Ridge Regression with PCA**:
  + **Settings**: Ridge with alpha=1.0 and PCA applied.
  + **Results**: Accuracy was 0.7085318210459988 or 70%
* **(e) Logistic Regression with PCA using Gradient Descent**:
  + **Settings**: PCA features and SGD as the solver.
  + **Results**: Achieved accuracy of 0.7091619407687461
* **(f) Logistic Regression without PCA and with Gradient Descent**:
  + **Settings**: Full feature set with SGD.
  + **Results**: Accuracy was 0.727393974531776 or 72.7%

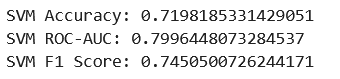
**Random Forest results**:



Compared to logistic regression with saga solver this does not perform as well as the lr model and here is the result for the lr model

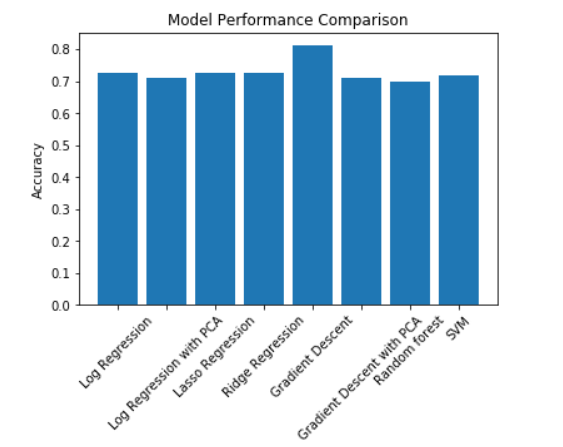
As you can see the log regression model works pretty well and 81% score is pretty good for the classification with 246 features.

**SVM results**:



This performs slightly better than the random forest model but still not as good as the log regression model with saga solver

Here is a graphical representation of the comparison between different models and experiments:



Based on the results shown in the bar graph, we observe a comparison of model performance across several algorithms in terms of accuracy. The highest accuracy is achieved by Logistic regression with Gradient Descent(saga), followed closely by the Logistic Regression variants (with and without PCA) and Ridge Regression, which all show competitive performance. The Random Forest model, while performing well, lags slightly behind the best-performing models, indicating it may not be the optimal choice for this dataset. The SVM model performs better than random forest but still not better than Log regression. Lasso Regression and Logistic Regression with PCA show slightly lower performance but remain competitive. Overall, Gradient Descent demonstrates the best performance, while PCA-augmented models do not show a significant improvement over their base models.

**Learning and Outcome**

Colab link: <https://colab.research.google.com/#fileId=https%3A//storage.googleapis.com/kaggle-colab-exported-notebooks/dota-2-lr-18525476-e084-4df5-bf1d-32a6c4c42a9d.ipynb%3FX-Goog-Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-com%2540kaggle-161607.iam.gserviceaccount.com/20241020/auto/storage/goog4_request%26X-Goog-Date%3D20241020T055124Z%26X-Goog-Expires%3D259200%26X-Goog-SignedHeaders%3Dhost%26X-Goog-Signature%>

Code repo: <https://github.com/rghdrizzle/Dota2-Win-prediction>

Dataset: <https://www.kaggle.com/competitions/mlcourse-dota2-win-prediction/data>

Here are the few things that I learnt while working on the project:

* + SVM kernals
  + Implementation of logistic regression
  + Different types of solvers
  + How to analyze the dataset and the results
  + Regularization

**Conclusion**

To conclude, we used the large dataset from kaggle to successfully build a logistical regression model to predict the results of a Dota 2 game. We also employed different other models such as SVM and Random forest to compare the results and analyze the performance. We could also infer more about the team’s performance from understanding the model and the results. There are some limitations to the project as not being able to fully utilize the SVM model to experiment further and for feature reduction using PCA, it infact only reduced the performance according to my experimentation. We could also observe that the Logistic regression model outperformed the random forest model and we could infer that in this case logistic regression gives far better performance. We did accomplish to reach a desired accuracy score which is 80%+ using logistic regression.