Sneha Dubey
Dr. Chen
CSCI 184
11 June 2024

Project Report

BACKGROUND

It's no secret that the American political climate nowadays is turbulent— with this country being made of people from diverse backgrounds and experiences, there are so many different ideologies competing to gain ground in the decision-making process. The way that they do this in the U.S. government is to vote. Members of the public vote for candidates that will represent them at the city level, state level, and ultimately the national level. By electing candidates from political parties whose ideologies and values align with their own, the public has the opportunity to make their voice heard in the public policy creation process.

The dataset that will be referenced within this project is a subset of the Cooperative Election Study (CES) 2022 Study Common Content (created largely by Brian Schaffner, Stephen Ansolabehere, and Marissa Shih; Harvard Dataverse). The original, full dataset can be found here. This study's dataset includes various demographic information pertaining to a nationally-representative sample of around 60,000 American voters; the researchers collected information such as each voter's home zip code, birthday, gender, education level, race, religion, past votes, and political affiliations.

This project takes into account these aforementioned diverse backgrounds and experiences of individual citizens of the United States, and aims to predict which political party they are most likely to vote with. Understanding people and how they think, and therefore how they vote, is critical to understanding the political climate in this country; a person's political identity is crucial in the outcome of each election they participate in.

DESIGN

This project aims to explore the relationship between an individual's demographic information and their political party affiliation, and to successfully train an optimal classification model to further predict a voter's party allegiance based on their characteristics. The steps followed in this project are as follows:

1. Preprocess the CES 2022 Study data, dropping all unnecessary features and encoding data to be model-friendly; split into training and testing data

2. Train the following classification models on the same training data, minimising the differences in their finetuning

Decision Tree	Logistic Regression
Random Forest	Naive Bayes
Support Vector Machine	XGBoost
K-Nearest Neighbours	Neural Network

   The aforementioned models were chosen for both their popularity with those in the industry and applicability to this specific situation.
3. Test each of the models using the testing data, noting the amount of success each has (based on their performance metrics); determine the most-successful model
4. Create a visual representation of each model's performance compared to one-another

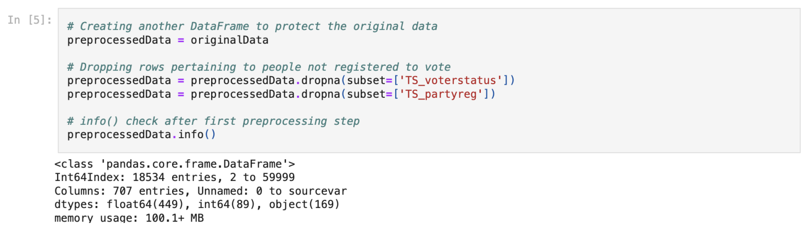
Because this study was so comprehensive in its gathering of about 706 feature variables, the dataset was heavily preprocessed such that only the features that discuss physical/ideological characteristics of each of the 60,000 voters, with the target variable being TS_partyreg (which political party each of the voters are registered with) were considered. This means that all features deemed to be distractions/noise (i.e. those not pertaining to tangible *characteristics* describing each individual voter but rather their actions, etc.), such as those pertaining to how the voters have voted in the past, from consideration (e.g. 2016/2020 presidential elections, senate elections, congressional elections, governors, etc.), were removed. This left the dataset with approximately 327 features, all pertaining *directly* to each surveyed individual's characteristics/ideologies, and 1 target (their party registration). Due to the amount of features, their precise names and the characteristics they stand for are not mentioned in this report, but can be found within the project source code.

For this project, a variety of python libraries were referenced in the process of model creation, training, and evaluation. These include pandas, numpy, sklearn, matplotlib, and xgboost. Therefore, in the replication of this project it will be necessary to have the corresponding libraries and toolkits installed.

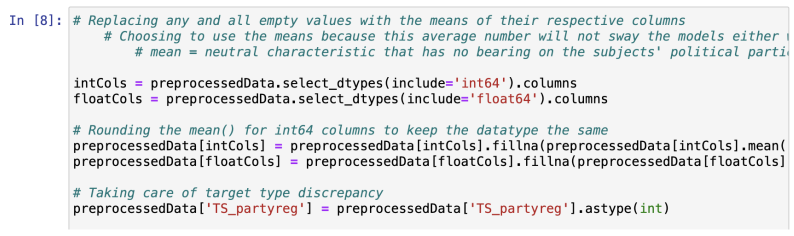
IMPLEMENTATION

Please reference the included python3 jupyter notebook for the full code. Included below are snippets of each step.

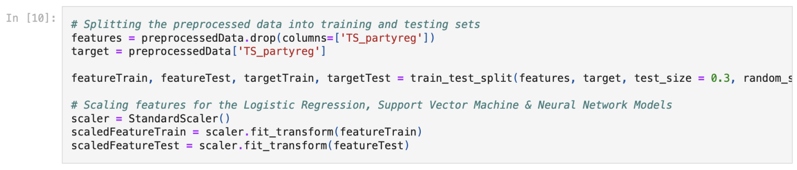
Data Cleaning:



```
In [5]:  # Creating another DataFrame to protect the original data
         preprocessedData = originalData

         # Dropping rows pertaining to people not registered to vote
         preprocessedData = preprocessedData.dropna(subset=['TS_voterstatus'])
         preprocessedData = preprocessedData.dropna(subset=['TS_partyreg'])

         # info() check after first preprocessing step
         preprocessedData.info()

        <class 'pandas.core.frame.DataFrame'>
        Int64Index: 18534 entries, 2 to 59999
        Columns: 707 entries, Unnamed: 0 to sourcevar
        dtypes: float64(449), int64(89), object(169)
        memory usage: 100.1+ MB
```

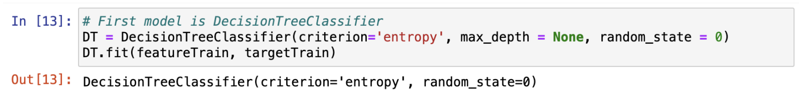


```
In [8]: # Replacing any and all empty values with the means of their respective columns
            # Choosing to use the means because this average number will not sway the models either 
                # mean = neutral characteristic that has no bearing on the subjects' political partic

        intCols = preprocessedData.select_dtypes(include='int64').columns
        floatCols = preprocessedData.select_dtypes(include='float64').columns

        # Rounding the mean() for int64 columns to keep the datatype the same
        preprocessedData[intCols] = preprocessedData[intCols].fillna(preprocessedData[intCols].mean(
        preprocessedData[floatCols] = preprocessedData[floatCols].fillna(preprocessedData[floatCols]

        # Taking care of target type discrepancy
        preprocessedData['TS_partyreg'] = preprocessedData['TS_partyreg'].astype(int)
```

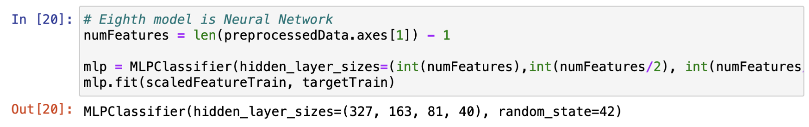


```
In [10]: # Splitting the preprocessed data into training and testing sets
         features = preprocessedData.drop(columns=['TS_partyreg'])
         target = preprocessedData['TS_partyreg']

         featureTrain, featureTest, targetTrain, targetTest = train_test_split(features, target, test_size = 0.3, random_s

         # Scaling features for the Logistic Regression, Support Vector Machine & Neural Network Models
         scaler = StandardScaler()
         scaledFeatureTrain = scaler.fit_transform(featureTrain)
         scaledFeatureTest = scaler.fit_transform(featureTest)
```

Model Training:

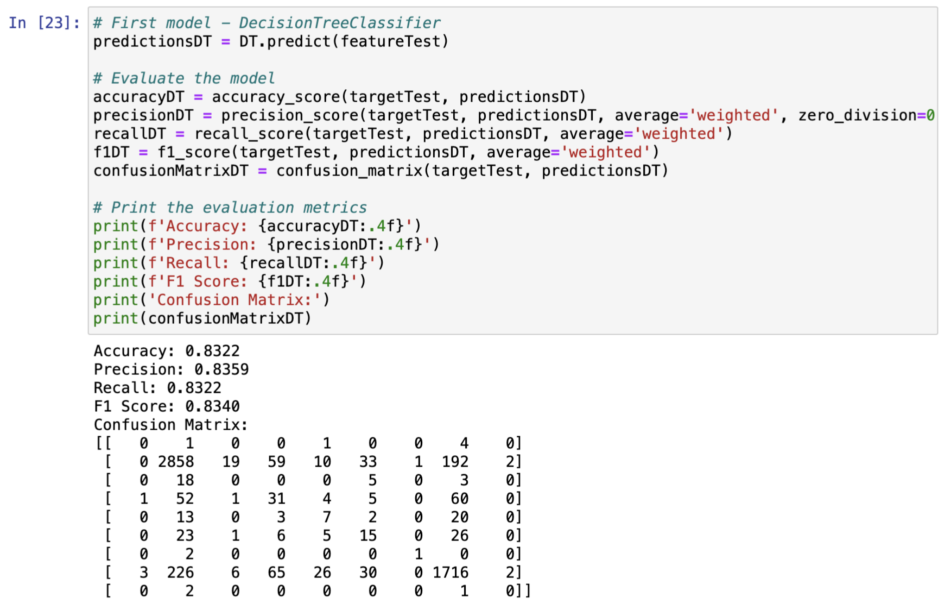


```
In [13]: # First model is DecisionTreeClassifier
         DT = DecisionTreeClassifier(criterion='entropy', max_depth = None, random_state = 0)
         DT.fit(featureTrain, targetTrain)

Out[13]: DecisionTreeClassifier(criterion='entropy', random_state=0)
```



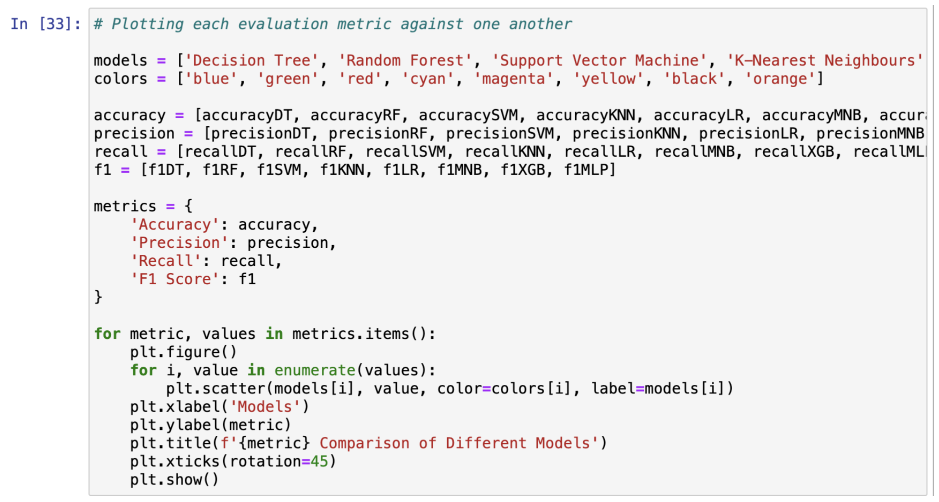
```
In [20]: # Eighth model is Neural Network
         numFeatures = len(preprocessedData.axes[1]) - 1

         mlp = MLPClassifier(hidden_layer_sizes=(int(numFeatures),int(numFeatures/2), int(numFeatures,
         mlp.fit(scaledFeatureTrain, targetTrain)

Out[20]: MLPClassifier(hidden_layer_sizes=(327, 163, 81, 40), random_state=42)
```

Model Evaluation:



```
In [23]: # First model - DecisionTreeClassifier
         predictionsDT = DT.predict(featureTest)

         # Evaluate the model
         accuracyDT = accuracy_score(targetTest, predictionsDT)
         precisionDT = precision_score(targetTest, predictionsDT, average='weighted', zero_division=0
         recallDT = recall_score(targetTest, predictionsDT, average='weighted')
         f1DT = f1_score(targetTest, predictionsDT, average='weighted')
         confusionMatrixDT = confusion_matrix(targetTest, predictionsDT)

         # Print the evaluation metrics
         print(f'Accuracy: {accuracyDT:.4f}')
         print(f'Precision: {precisionDT:.4f}')
         print(f'Recall: {recallDT:.4f}')
         print(f'F1 Score: {f1DT:.4f}')
         print('Confusion Matrix:')
         print(confusionMatrixDT)

         Accuracy: 0.8322
         Precision: 0.8359
         Recall: 0.8322
         F1 Score: 0.8340
         Confusion Matrix:
         [[   0    1    0    0    1    0    0    4    0]
          [   0 2858   19   59   10   33    1  192    2]
          [   0   18    0    0    0    5    0    3    0]
          [   1   52    1   31    4    5    0   60    0]
          [   0   13    0    3    7    2    0   20    0]
          [   0   23    1    6    5   15    0   26    0]
          [   0    2    0    0    0    0    1    0    0]
          [   3  226    6   65   26   30    0 1716    2]
          [   0    2    0    0    0    0    0    1    0]]
```

Comparing Performance:



```
In [33]: # Plotting each evaluation metric against one another

         models = ['Decision Tree', 'Random Forest', 'Support Vector Machine', 'K-Nearest Neighbours'
         colors = ['blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black', 'orange']

         accuracy = [accuracyDT, accuracyRF, accuracySVM, accuracyKNN, accuracyLR, accuracyMNB, accur
         precision = [precisionDT, precisionRF, precisionSVM, precisionKNN, precisionLR, precisionMNB
         recall = [recallDT, recallRF, recallSVM, recallKNN, recallLR, recallMNB, recallXGB, recallML
         f1 = [f1DT, f1RF, f1SVM, f1KNN, f1LR, f1MNB, f1XGB, f1MLP]

         metrics = {
             'Accuracy': accuracy,
             'Precision': precision,
             'Recall': recall,
             'F1 Score': f1
         }

         for metric, values in metrics.items():
             plt.figure()
             for i, value in enumerate(values):
                 plt.scatter(models[i], value, color=colors[i], label=models[i])
             plt.xlabel('Models')
             plt.ylabel(metric)
             plt.title(f'{metric} Comparison of Different Models')
             plt.xticks(rotation=45)
             plt.show()
```

RESULTS

Below are the listed performance metrics for each of the models, as well as visual representations of their performance compared to others'.

Decision Tree:

Accuracy: 0.8322 Recall: 0.8322

Precision: 0.8359 F1 Score: 0.8340

Random Forest:

Accuracy: 0.8935 Recall: 0.8935

Precision: 0.8513 F1 Score: 0.8679

Support Vector Machine:

Accuracy: 0.5708 Recall: 0.5708

Precision: 0.3258 F1 Score: 0.4148

K-Nearest Neighbours:

Accuracy: 0.5235 Recall: 0.5235

Precision: 0.4731 F1 Score: 0.4902

Logistic Regression:

Accuracy: 0.8759 Recall: 0.8759

Precision: 0.8612 F1 Score: 0.8676

Naive Bayes:

Accuracy: 0.0277 Recall: 0.0277

Precision: 0.4824 F1 Score: 0.0382

XGBoost:

Accuracy: 0.0002 Recall: 0.0002

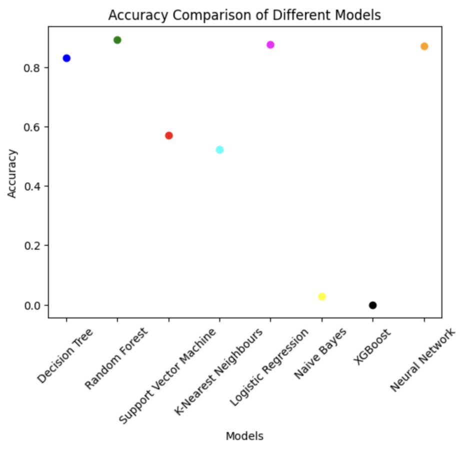
Precision: 0.0000 F1 Score: 0.0000

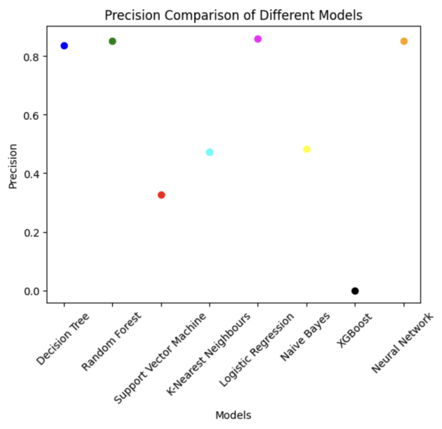
Neural Network:

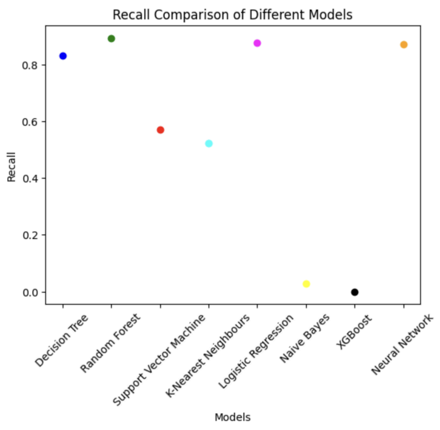
Accuracy: 0.8720 Recall: 0.8720

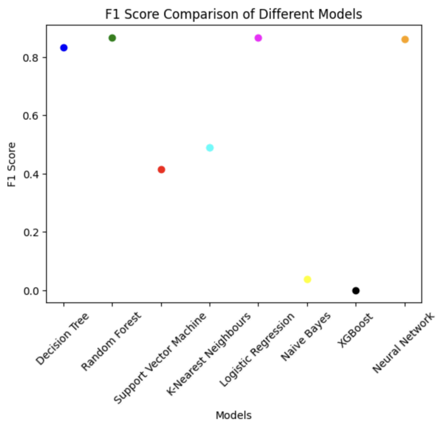
Precision: 0.8537 F1 Score: 0.8621

Plots:









INTERPRETATION

These results indicate that the final order of models, from most successful at categorising each individual into their appropriate political party based on their characteristics/ideologies to the least, is as follows:

1. Random Forest
2. Logistic Regression
3. Neural Network
4. Decision Tree
5. K-Nearest Neighbours
6. Support Vector Machine
7. Naive Bayes
8. XGBoost

It appears as though the Random Forest Classifier has, on average, the highest performance metrics, and XGBoost has the lowest. The overall success of the Random Forest model may be due to the nature of the algorithm; many different trees are generated and the result is calculated by a majority vote. This classification method ensures that the majority of the decision trees obtained this result, and therefore it is supported adequately. On the other hand, XGBoost relies on one trained decision tree to make the classification. While this final tree has been heavily trained (with each iteration improving upon the errors of the last), it appears as though it was not the ideal model to deal with this specific data. Perhaps this model would be more advantageous in a different scenario.