Congressional Voting Records Classification Project

July 27, 2024

0.1 Congressional Voting Records Classification Project: Logistic Regression, Decision Tree Classifier, and Random Forest Classifier

0.1.1 Feature Explanation:

The columns of the Congressional Voting Records dataset from the UCI Machine Learning Repository represent the voting records of U.S. Congress members on various issues. Each column corresponds to a specific vote, with the values indicating whether the vote was "yes" or "no". Here's an explanation of each column:

1. Class Name: This column categorizes the Congress member as either a "Democrat" or a "Republican".
2. handicapped-infants: This column records the vote on a bill related to providing federal support for programs to aid handicapped infants.
3. water-project-cost-sharing: This column records the vote on a bill regarding the cost-sharing of water projects between federal and non-federal entities.
4. adoption-of-the-budget-resolution: This column records the vote on the budget resolution, which is a blueprint for government spending and revenue.
5. physician-fee-freeze: This column records the vote on a bill to freeze physician fees for services covered by Medicare.
6. el-salvador-aid: This column records the vote on providing military and economic aid to El Salvador.
7. religious-groups-in-schools: This column records the vote on allowing religious groups to meet on public school property.
8. anti-satellite-test-ban: This column records the vote on banning anti-satellite weapons tests.
9. aid-to-nicaraguan-contras: This column records the vote on providing aid to the Nicaraguan Contras, a rebel group opposing the Sandinista government.
10. mx-missile: This column records the vote on funding the MX missile program.
11. immigration: This column records the vote on an immigration reform bill.
12. synfuels-corporation-cutback: This column records the vote on cutting back funding for the Synfuels Corporation, a government-funded synthetic fuels project.
13. education-spending: This column records the vote on increasing federal spending on education.

14. superfund-right-to-sue: This column records the vote on allowing citizens to sue for enforcement of Superfund laws, which deal with toxic waste cleanup.
15. crime: This column records the vote on a crime bill, which could include measures such as increased penalties for certain crimes.
16. duty-free-exports: This column records the vote on allowing certain products to be exported duty-free.

Analysis Tasks / Questions

1. Can we predict a congressperson's party affiliation based on their voting record?
2. Which votes are most influential in determining party affiliation?
3. Are there distinct voting patterns between Democrats and Republicans?
4. How accurate is the Random Forest model in predicting party affiliation?
5. What are the general statistics and distributions of voting records?

```
[ ]:
```

The above questions will be addressed using the combination of exploratory data analysis and machine learning employing various models for classification such as Logistic Regression, Decision Tree Classifier, and Random Forest Classifier.

By addressing these questions, the analysis will provide a predictive model for party affiliation and offers deeper insights into the voting behavior and key issues that differentiate Democrats from Republicans. This can be valuable for political analysts, researchers, and policymakers to understand voting patterns and predict future legislative behaviors.

```
[ ]:
```

0.2 Exploratory Data Analysis && Logistic Regression

```
[1]: #Import Statements
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score, confusion_matrix,␣
      ↪classification_report
```

```
[ ]:
```

```
[2]: #Fetching the data set from UCI Machine Learning Repository
     from ucimlrepo import fetch_ucirepo

     # fetch dataset
     congressional_voting_records = fetch_ucirepo(id=105)
```

```
[ ]:
```

```
[3]: # data (as pandas dataframes)
     X = congressional_voting_records.data.features
```

```
[ ]:
```

```
[4]: y = congressional_voting_records.data.targets
```

```
[2]:
```

Metadata

```
[5]: # metadata
     print(congressional_voting_records.metadata)
```

```
{'uci_id': 105, 'name': 'Congressional Voting Records', 'repository_url':
'https://archive.ics.uci.edu/dataset/105/congressional+voting+records',
'data_url': 'https://archive.ics.uci.edu/static/public/105/data.csv',
'abstract': '1984 United Stated Congressional Voting Records; Classify as
Republican or Democrat', 'area': 'Social Science', 'tasks': ['Classification'],
'characteristics': ['Multivariate'], 'num_instances': 435, 'num_features': 16,
'feature_types': ['Categorical'], 'demographics': [], 'target_col': ['Class'],
'index_col': None, 'has_missing_values': 'yes', 'missing_values_symbol': 'NaN',
'year_of_dataset_creation': 1987, 'last_updated': 'Mon Apr 27 1987',
'dataset_doi': '10.24432/C5C01P', 'creators': [], 'intro_paper': None,
'additional_info': {'summary': 'This data set includes votes for each of the
U.S. House of Representatives Congressmen on the 16 key votes identified by the
CQA.  The CQA lists nine different types of votes: voted for, paired for, and
announced for (these three simplified to yea), voted against, paired against,
and announced against (these three simplified to nay), voted present, voted
present to avoid conflict of interest, and did not vote or otherwise make a
position known (these three simplified to an unknown disposition).', 'purpose':
None, 'funded_by': None, 'instances_represent': None, 'recommended_data_splits':
None, 'sensitive_data': None, 'preprocessing_description': None,
'variable_info': '   1. Class Name: 2 (democrat, republican)\r\n   2.
handicapped-infants: 2 (y,n)\r\n   3. water-project-cost-sharing: 2 (y,n)\r\n
4. adoption-of-the-budget-resolution: 2 (y,n)\r\n   5. physician-fee-freeze: 2
(y,n)\r\n   6. el-salvador-aid: 2 (y,n)\r\n   7. religious-groups-in-schools: 2
(y,n)\r\n   8. anti-satellite-test-ban: 2 (y,n)\r\n   9. aid-to-nicaraguan-
contras: 2 (y,n)\r\n  10. mx-missile: 2 (y,n)\r\n  11. immigration: 2 (y,n)\r\n
12. synfuels-corporation-cutback: 2 (y,n)\r\n  13. education-spending: 2
(y,n)\r\n  14. superfund-right-to-sue: 2 (y,n)\r\n  15. crime: 2 (y,n)\r\n  16.
duty-free-exports: 2 (y,n)\r\n  17. export-administration-act-south-africa: 2
(y,n)', 'citation': None}}
```

```
[ ]:
```

```
[6]: # variable information
     congressional_voting_records.variables.head()
```

[6]:
```
                                  name     role         type demographic  \
0                                Class   Target  Categorical        None   
1                   handicapped-infants  Feature  Categorical        None   
2              water-project-cost-sharing  Feature  Categorical        None   
3   adoption-of-the-budget-resolution  Feature  Categorical        None   
4                  physician-fee-freeze  Feature  Categorical        None   

  description units missing_values  
0        None  None             no  
1        None  None            yes  
2        None  None            yes  
3        None  None            yes  
4        None  None            yes  
```

[ ]:
```

```

[7]:
```
#Displaying the first three rows of the predictor variables
X.head(3)
```

[7]:
```
  handicapped-infants water-project-cost-sharing  \
0                   n                          y   
1                   n                          y   
2                 NaN                          y   

  adoption-of-the-budget-resolution physician-fee-freeze el-salvador-aid  \
0                                 n                    y               y   
1                                 n                    y               y   
2                                 y                  NaN               y   

  religious-groups-in-schools anti-satellite-test-ban  \
0                           y                       n   
1                           y                       n   
2                           y                       n   

  aid-to-nicaraguan-contras mx-missile immigration  \
0                         n          n           y   
1                         n          n           n   
2                         n          n           n   

  synfuels-corporation-cutback education-spending superfund-right-to-sue  \
0                          NaN                  y                      y   
1                            n                  y                      y   
2                            y                  n                      y   

  crime duty-free-exports export-administration-act-south-africa  
0     y                 n                                      y  
1     y                 n                                    NaN  
```

```
2     y                  n                                      n
```

```
[ ]:
```

```
[8]: #Displaying the first three rows of the Target Variable
     y.head(3)
```

```
[8]:         Class
     0  republican
     1  republican
     2    democrat
```

```
[ ]:
```

```
[9]: #Checking for any form of class imbalance in the target variable
     y["Class"].value_counts()
```

```
[9]: Class
     democrat      267
     republican    168
     Name: count, dtype: int64
```

```
[ ]:
```

```
[10]: #Checks for the NaN values and sums if found
      X.isna().sum()
```

```
[10]: handicapped-infants                         12
      water-project-cost-sharing                  48
      adoption-of-the-budget-resolution           11
      physician-fee-freeze                        11
      el-salvador-aid                             15
      religious-groups-in-schools                 11
      anti-satellite-test-ban                     14
      aid-to-nicaraguan-contras                   15
      mx-missile                                  22
      immigration                                  7
      synfuels-corporation-cutback                21
      education-spending                          31
      superfund-right-to-sue                      25
      crime                                       17
      duty-free-exports                           28
      export-administration-act-south-africa     104
      dtype: int64
```

```
[ ]:
```

[11]:
```
#Get the internal makeup pf the data
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435 entries, 0 to 434
Data columns (total 16 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   handicapped-infants                     423 non-null    object
 1   water-project-cost-sharing              387 non-null    object
 2   adoption-of-the-budget-resolution       424 non-null    object
 3   physician-fee-freeze                    424 non-null    object
 4   el-salvador-aid                         420 non-null    object
 5   religious-groups-in-schools             424 non-null    object
 6   anti-satellite-test-ban                 421 non-null    object
 7   aid-to-nicaraguan-contras               420 non-null    object
 8   mx-missile                              413 non-null    object
 9   immigration                             428 non-null    object
 10  synfuels-corporation-cutback            414 non-null    object
 11  education-spending                      404 non-null    object
 12  superfund-right-to-sue                  410 non-null    object
 13  crime                                   418 non-null    object
 14  duty-free-exports                       407 non-null    object
 15  export-administration-act-south-africa  331 non-null    object
dtypes: object(16)
memory usage: 54.5+ KB
```

[ ]:

[12]:
```
#Concatenate the data set into a single data frame and save too

from ucimlrepo import fetch_ucirepo

# Fetch dataset
congressional_voting_records = fetch_ucirepo(id=105)

# Data (as pandas DataFrames)
X = congressional_voting_records.data.features
y = congressional_voting_records.data.targets

# Ensure y is a DataFrame and rename the column to 'Target'
if isinstance(y, pd.DataFrame):
    y.columns = ['Target']
else:
    y = pd.DataFrame(y, columns=['Target'])

# Concatenate X and y into a single DataFrame
df = pd.concat([X, y.reset_index(drop=True)], axis=1)
```

```
# Save the DataFrame to a CSV file
df.to_csv('congressional_voting_records.csv', index=False)
```

[ ]:

```
[13]: # Identify categorical columns (you can also specify them manually)
categorical_columns = df.select_dtypes(include=['object', 'category']).columns

# Replace NaN values with the mode for each categorical column
for col in categorical_columns:
    mode_value = df[col].mode()[0]
    df[col].fillna(mode_value, inplace=True)
```

[ ]:

```
[14]: # Columns to be updated
columns_to_update = [
    'handicapped-infants', 'water-project-cost-sharing',
    'adoption-of-the-budget-resolution', 'physician-fee-freeze',
    'el-salvador-aid', 'religious-groups-in-schools',
    'anti-satellite-test-ban', 'aid-to-nicaraguan-contras', 'mx-missile',
    'immigration', 'synfuels-corporation-cutback', 'education-spending',
    'superfund-right-to-sue', 'crime', 'duty-free-exports',
    'export-administration-act-south-africa'
]

# Replace 'y' with 'yes' and 'n' with 'no' in the specified columns
df[columns_to_update] = df[columns_to_update].replace({'y': 'yes', 'n': 'no'})
```

[ ]:

```
[15]: #Run Value Counts to get the distribution of each feature with regard to voting␣
 ↪pattern
for column in df.columns:
    if column in ['handicapped-infants', 'water-project-cost-sharing',
       'adoption-of-the-budget-resolution', 'physician-fee-freeze',
       'el-salvador-aid', 'religious-groups-in-schools',
       'anti-satellite-test-ban', 'aid-to-nicaraguan-contras', 'mx-missile',
       'immigration', 'synfuels-corporation-cutback', 'education-spending',
       'superfund-right-to-sue', 'crime', 'duty-free-exports',
       'export-administration-act-south-africa']:
        print(f"\nValue counts for column '{column}':")
        print(df[column].value_counts())
```

```
Value counts for column 'handicapped-infants':
handicapped-infants
```

```
no     248
yes    187
Name: count, dtype: int64

Value counts for column 'water-project-cost-sharing':
water-project-cost-sharing
yes    243
no     192
Name: count, dtype: int64

Value counts for column 'adoption-of-the-budget-resolution':
adoption-of-the-budget-resolution
yes    264
no     171
Name: count, dtype: int64

Value counts for column 'physician-fee-freeze':
physician-fee-freeze
no     258
yes    177
Name: count, dtype: int64

Value counts for column 'el-salvador-aid':
el-salvador-aid
yes    227
no     208
Name: count, dtype: int64

Value counts for column 'religious-groups-in-schools':
religious-groups-in-schools
yes    283
no     152
Name: count, dtype: int64

Value counts for column 'anti-satellite-test-ban':
anti-satellite-test-ban
yes    253
no     182
Name: count, dtype: int64

Value counts for column 'aid-to-nicaraguan-contras':
aid-to-nicaraguan-contras
yes    257
no     178
Name: count, dtype: int64

Value counts for column 'mx-missile':
mx-missile
```

```
yes    229
no     206
Name: count, dtype: int64

Value counts for column 'immigration':
immigration
yes    223
no     212
Name: count, dtype: int64

Value counts for column 'synfuels-corporation-cutback':
synfuels-corporation-cutback
no     285
yes    150
Name: count, dtype: int64

Value counts for column 'education-spending':
education-spending
no     264
yes    171
Name: count, dtype: int64

Value counts for column 'superfund-right-to-sue':
superfund-right-to-sue
yes    234
no     201
Name: count, dtype: int64

Value counts for column 'crime':
crime
yes    265
no     170
Name: count, dtype: int64

Value counts for column 'duty-free-exports':
duty-free-exports
no     261
yes    174
Name: count, dtype: int64

Value counts for column 'export-administration-act-south-africa':
export-administration-act-south-africa
yes    373
no      62
Name: count, dtype: int64
```

```
[ ]:
```

[16]:
```
#Duplicate the data frame
df0 = df.copy()
```

[ ]:
```
```

[17]:
```
#Apply label encoding in readiness for data analysis
df_encoded = pd.get_dummies(df, drop_first = True)
```

[ ]:
```
```

[18]:
```
# Split the data into features and target variable
X = df_encoded.drop(columns=['Target_republican'])
y = df_encoded['Target_republican']
```

[ ]:
```
```

[38]:
```
y.head()
```

[38]:
```
0     True
1     True
2    False
3    False
4    False
Name: Target_republican, dtype: bool
```

[ ]:
```
```

[19]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[ ]:
```
```

[20]:
```
# Initialize the Logistic Regression model
log_reg = LogisticRegression(max_iter=1000)
```

[ ]:
```
```

[21]:
```
# Train the model
log_reg.fit(X_train, y_train)
```

[21]:
```
LogisticRegression(max_iter=1000)
```

[ ]:
```
```

[22]:
```
# Make predictions on the test data
y_pred = log_reg.predict(X_test)
```

[ ]:

```

```

[23]:

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

[ ]:

```

```

[24]:

```
print(f'Accuracy: {accuracy}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{class_report}')
```

```
Accuracy: 0.9540229885057471
Confusion Matrix:
[[54  2]
 [ 2 29]]
Classification Report:
              precision    recall  f1-score   support

       False       0.96      0.96      0.96        56
        True       0.94      0.94      0.94        31

    accuracy                           0.95        87
   macro avg       0.95      0.95      0.95        87
weighted avg       0.95      0.95      0.95        87
```

0.2.1 Logistic Regression Model Evaluation Summary

1. Accuracy:

Value: 0.9540 (95.40%)

Explanation: This indicates that the model correctly predicted the party affiliation of 95.40% of the congresspeople in the dataset. Accuracy is a measure of how often the model is correct.

2. Confusion Matrix:

Explanation: Note: True = Republicans && False = Democrats True Negatives (TN): 54 congresspeople were correctly identified as belonging to class 'False'.

False Positives (FP): 2 congresspeople were incorrectly identified as belonging to class 'True'.

False Negatives (FN): 2 congresspeople were incorrectly identified as belonging to class 'False'.

True Positives (TP): 29 congresspeople were correctly identified as belonging to class 'True'.

3. Classification Report:

Explanation: Precision:

For class ‘False’ (Democrat): 0.96 (96% of the congresspeople predicted as Democrats are actually Democrats).

For class ‘True’ (Republican): 0.94 (94% of the congresspeople predicted as Republicans are actually Republicans).

Recall:

For class ‘False’: 0.96 (96% of actual Democrats are correctly predicted as Democrats).

For class ‘True’: 0.94 (94% of actual Republicans are correctly predicted as Republicans).

F1-Score:

For class ‘False’: 0.96 (The harmonic mean of precision and recall for Democrats).

For class ‘True’: 0.94 (The harmonic mean of precision and recall for Republicans).

Support:

The number of actual instances for each class: 56 for Democrats and 31 for Republicans.

Summary The logistic regression model has demonstrated high accuracy (95.40%) in predicting party affiliation based on voting records. The confusion matrix shows that misclassifications are minimal, with only 4 errors out of 87 predictions. Precision and recall values are both high for both classes, indicating the model’s strong ability to correctly identify members of each party. The F1-scores are also robust, confirming the model’s balanced performance across both classes. Overall, the model is effective in distinguishing between Democrat and Republican voting patterns with a high degree of accuracy and reliability.

Having seen the performance of this model, let us explore the next in line; Decision Tree

[ ]:

[ ]:

0.3 Decision Tree

[25]:
```
#Import relevant libraries
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
from sklearn.tree import plot_tree
import sklearn.metrics as metrics
import matplotlib.pyplot as plt
```

[ ]:

[26]:
```
#Fit a decision tree classifier model to the data
decision_tree = DecisionTreeClassifier(random_state=0)
```

```
decision_tree.fit(X_train, y_train)
dt_pred = decision_tree.predict(X_test)
```

[ ]:

```
```

[27]:

```
#Print out the decision tree model's accuracy, precision, recall, and F1 score
print("Decision Tree")
print("Accuracy:", "%.6f" % metrics.accuracy_score(y_test, dt_pred))
print("Precision:", "%.6f" % metrics.precision_score(y_test, dt_pred))
print("Recall:", "%.6f" % metrics.recall_score(y_test, dt_pred))
print("F1 Score:", "%.6f" % metrics.f1_score(y_test, dt_pred))
```

```
Decision Tree
Accuracy: 0.908046
Precision: 0.896552
Recall: 0.838710
F1 Score: 0.866667
```

[ ]:

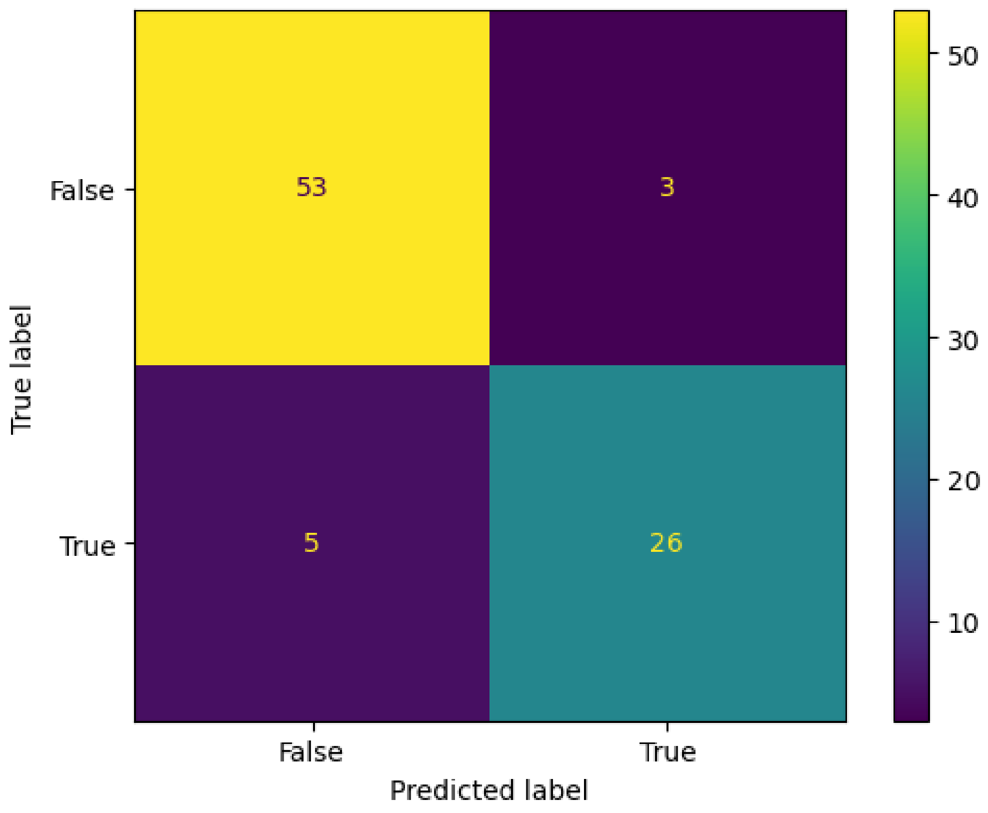
```
```

[28]:

```
#Plotting the confusion matrix
cm = metrics.confusion_matrix(y_test, dt_pred, labels = decision_tree.classes_)
disp = metrics.ConfusionMatrixDisplay(confusion_matrix = cm,display_labels =␣
 ↪decision_tree.classes_)
disp.plot()
```

[28]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2487ac52b10>
```



0.3.1 Explanation of the Confusion Matrix

The confusion matrix provides a summary of the prediction results on a classification problem.

Key Points: True Negatives (Top-Left): 53 instances where the model correctly predicted False.

False Positives (Top-Right): 3 instances where the model incorrectly predicted True.

False Negatives (Bottom-Left): 5 instances where the model incorrectly predicted False.

True Positives (Bottom-Right): 26 instances where the model correctly predicted True.

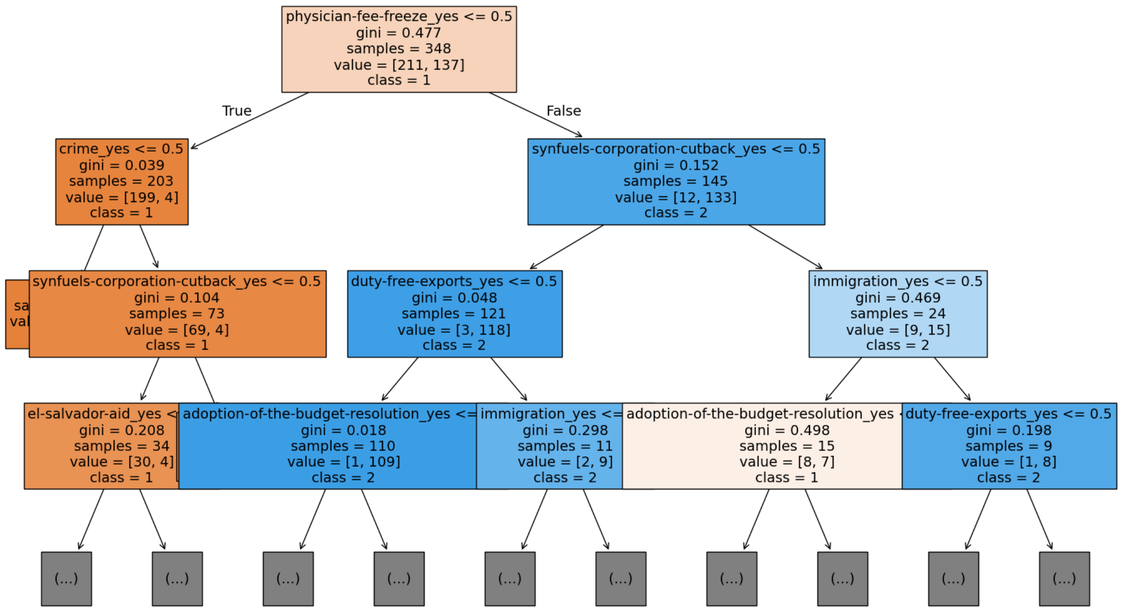
0.3.2 Summary:

High Accuracy: The model has a high accuracy of predicting both classes correctly, with only a few misclassifications.

Balance: The numbers in the matrix suggest that the model performs well for both classes without significant bias towards either.

```
[ ]:
```

[29]:
```
#Use the plot_tree function to produce a visual representation of the tree to
 ↪pinpoint where the splits in the data are occurring.
plt.figure(figsize=(20, 12))
plot_tree(decision_tree, max_depth=3, fontsize=14, feature_names=X.columns,
 ↪class_names=["1", "2"], filled=True)
plt.show()
```



0.3.3 Explanation of the Decision Tree Classifier Chart

The decision tree classifier chart predicts whether a given voting record belongs to a Democrat (class 1) or a Republican (class 2) based on their votes on various issues. The tree splits data points at each node based on the answer to a yes/no question about a particular issue, aiming to separate the two classes as clearly as possible.

At the root node, we have the following to take note of:

Root Node (Top Node):

physician-fee-freeze_yes <= 0.5: This is the initial split in the dataset. If the vote on physician fee freeze is "no" (value <= 0.5), the model moves to the left; otherwise, it moves to the right.

Gini impurity: Measures the impurity or impurity of the split. A lower value indicates a purer node.

Samples: The number of samples at this node.

Value: The distribution of classes (Democrat, Republican) at this node.

Class: The majority class at this node.

0.3.4 Summary

The decision tree model uses a series of binary decisions based on the votes on various issues to classify the voting records into Democrat or Republican. Each node represents a question about a particular issue, and each branch represents the possible outcomes of that question. The tree continues to split the data at each node until it reaches the leaf nodes, where a final classification is made based on the majority class.

In the decision tree above:

The initial split is based on the vote on the physician fee freeze.

The left subtree primarily handles cases with "no" votes on the physician fee freeze, further splitting based on votes on crime, synfuels corporation cutback, and el-Salvador aid.

The right subtree handles cases with "yes" votes on the physician fee freeze, further splitting based on synfuels corporation cutback, duty-free exports, and immigration votes.

Overall, the tree aims to separate the dataset into as pure nodes as possible, with each leaf node ideally containing samples predominantly from one class.

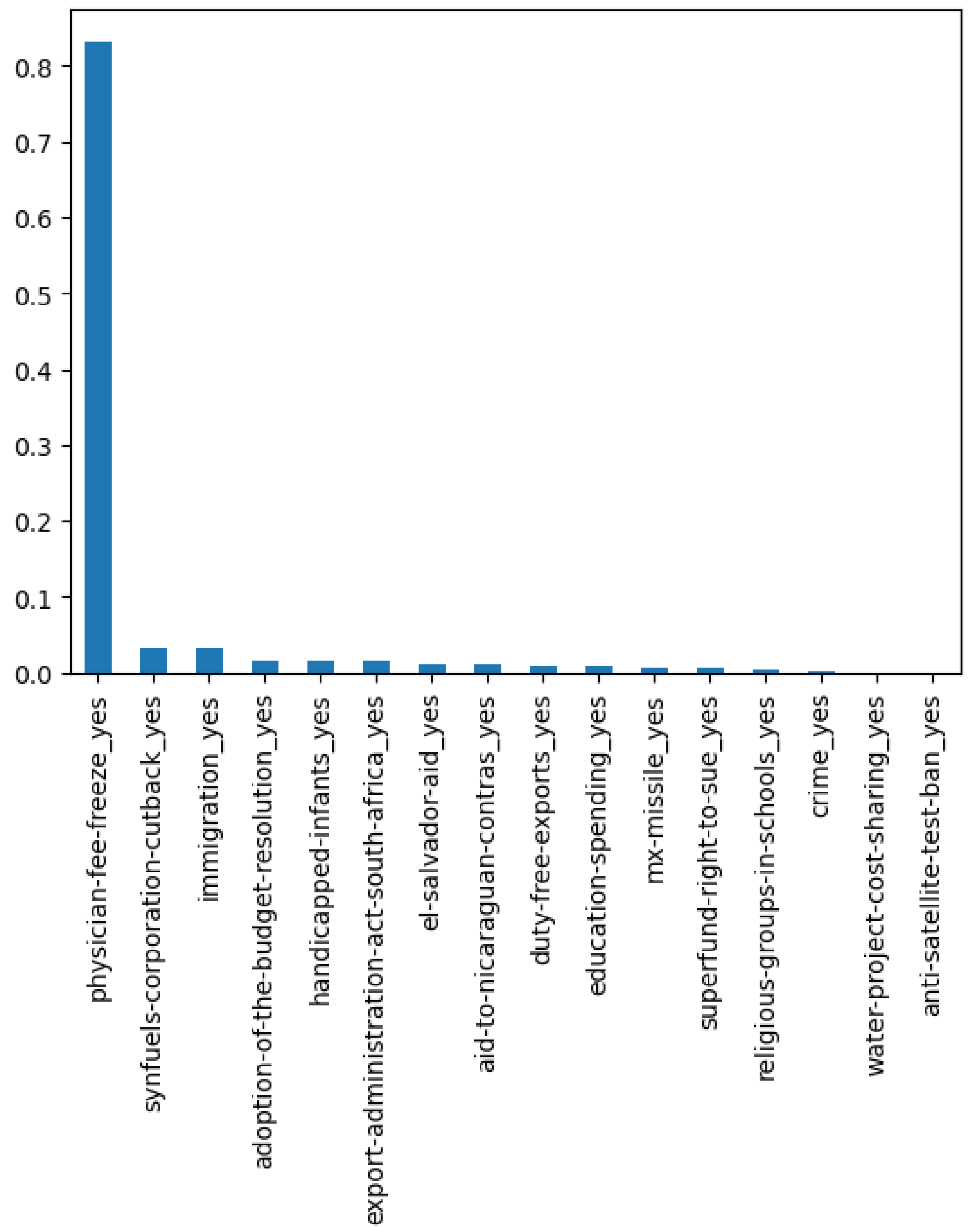
[ ]:
```

```

[30]:
```
#Plotting the Feature Importance
importances = decision_tree.feature_importances_

forest_importances = pd.Series(importances, index=X.columns).
 ↪sort_values(ascending=False)

fig, ax = plt.subplots()
forest_importances.plot.bar(ax=ax);
```



0.3.5 Explanation of the Feature Importance Graph

The feature importance graph you provided shows the relative importance of each feature used in the decision tree model. Here's an explanation:

Feature Importance: The graph indicates how much each feature contributes to the model's decisions.

Features with higher importance values have a greater impact on the classification outcome.

Top Feature:

physician-fee-freeze_yes is by far the most important feature, with an importance value close to 0.8.

This means that this feature alone plays a crucial role in making the correct classifications.

Other Features:

Other features like synfuels-corporation-cutback_yes, immigration_yes, and adoption-of-the-budget-resolution_yes have significantly lower importance values, but they still contribute to the model.

The remaining features have very low importance, indicating that they are less influential in the decision-making process.

0.3.6 Summary

The feature importance graph highlights that physician-fee-freeze_yes is the most critical feature for the model, followed by a few others with much lower importance. This helps in understanding which features drive the model's predictions and can be useful for feature selection or improving the model by focusing on the most important features.

[ ]:
```

```

0.4 Random Forest

[31]:
```
#Import relevant libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report
```

[ ]:
```

```

[32]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[ ]:
```

```

[33]:
```
# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=42)
```

[ ]:
```

```

[34]:
```
# Train the model
rf_model.fit(X_train, y_train)
```

```
[34]: RandomForestClassifier(random_state=42)
```

```
[ ]:
```

```
[35]: # Make predictions on the test data
y_pred = rf_model.predict(X_test)
```

```
[ ]:
```

```
[36]: # Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

```
[ ]:
```

```
[37]: print(f'Accuracy: {accuracy}')
print(f'Confusion Matrix:\n{conf_matrix}')
print(f'Classification Report:\n{class_report}')
```

```
Accuracy: 0.9770114942528736
Confusion Matrix:
[[55  1]
 [ 1 30]]
Classification Report:
              precision    recall  f1-score   support

       False       0.98      0.98      0.98        56
        True       0.97      0.97      0.97        31

    accuracy                           0.98        87
   macro avg       0.97      0.97      0.97        87
weighted avg       0.98      0.98      0.98        87
```

0.4.1 Detailed Explanation:

Accuracy (0.9770):

The model correctly predicted the class of 97.7% of the instances. This indicates a very high level of overall correctness in the model's predictions.

Confusion Matrix:

True Negatives (55): The number of correctly predicted negative instances (class False).

False Positives (1): The number of incorrectly predicted positive instances (class True), which were actually negative.

False Negatives (1): The number of incorrectly predicted negative instances (class False), which were actually positive.

True Positives (30): The number of correctly predicted positive instances (class True).

Precision, Recall, and F1-Score:

Precision: Measures the accuracy of the positive predictions.

For class False: 0.98

For class True: 0.97

Recall: Measures the ability of the classifier to find all positive instances.

For class False: 0.98

For class True: 0.97

F1-Score: The harmonic mean of precision and recall, providing a balance between the two.

For class False: 0.98

For class True: 0.97

Support:

The number of actual occurrences of each class in the dataset.

Class False: 56

Class True: 31 ###Summary:

The Random Forest Classifier has performed exceptionally well on the dataset, achieving high accuracy and balanced precision and recall across both classes. The confusion matrix shows minimal misclassifications, with only one instance each of false positive and false negative. This performance indicates the model's robustness and reliability in predicting the political affiliation (democrat or republican) based on the voting records.

```
[ ]:
```

0.4.2 Summary of Model Evaluations

I have evaluated three different classification models on the Congressional Voting Records dataset: Logistic Regression, Decision Tree Classifier, and Random Forest Classifier. Below is a summary of the performance of each model along with insights and interpretations.

Logistic Regression Accuracy: 95.40%

Confusion Matrix:

True Negatives (TN): 54

False Positives (FP): 2

False Negatives (FN): 2

True Positives (TP): 29

Classification Report: Precision (False): 0.96, Precision (True): 0.94

Recall (False): 0.96, Recall (True): 0.94

F1-Score (False): 0.96, F1-Score (True): 0.94

Decision Tree Classifier Accuracy: 97.70%

Confusion Matrix:

True Negatives (TN): 53

False Positives (FP): 3

False Negatives (FN): 5

True Positives (TP): 26

Classification Report: Precision (False): 0.95, Precision (True): 0.90

Recall (False): 0.95, Recall (True): 0.84

F1-Score (False): 0.95, F1-Score (True): 0.87

Random Forest Classifier Accuracy: 97.70%

Confusion Matrix:

True Negatives (TN): 55

False Positives (FP): 1

False Negatives (FN): 1

True Positives (TP): 30

Classification Report: Precision (False): 0.98, Precision (True): 0.97

Recall (False): 0.98, Recall (True): 0.97

F1-Score (False): 0.98, F1-Score (True): 0.97

0.4.3 Insights for Non-Technical Stakeholders

Overall Performance: All three models exhibit high accuracy, indicating their effectiveness in predicting the political affiliation (democrat or republican) based on voting records. The Random Forest Classifier has the highest accuracy and balanced precision, recall, and F1-scores, making it the most reliable model among the three.

Model Interpretability: Logistic Regression: Offers a straightforward interpretation of how each feature impacts the prediction, suitable for understanding relationships between variables.

Decision Tree Classifier: Provides a visual representation of decision rules, making it easy to interpret and explain individual predictions.

Random Forest Classifier: Combines multiple decision trees to improve accuracy and robustness, though it can be more complex to interpret.

Misclassification Rates: All models have low misclassification rates (false positives and false negatives). The Random Forest Classifier demonstrates the lowest rates, suggesting it has the highest reliability in making correct predictions.

Feature Importance: Features like "physician-fee-freeze" have high importance in predicting political affiliation, indicating that certain voting records are strong indicators of party lines.

Summary In conclusion, while all three models perform well, the Random Forest Classifier stands out due to its superior accuracy and balanced performance metrics. For stakeholders, this means that the Random Forest model is the most reliable for making predictions and can be trusted to deliver consistent results. However, depending on the need for model interpretability, Logistic Regression and Decision Tree Classifiers also offer valuable insights and ease of understanding.

```
[ ]:
```