Dillon Mavunga and Anna Taylor
Professor Sommer
Machine Learning
8 December 2024

Hamilton County Voter Turnout

Our Problem

Our problem we are addressing is using the Hamilton County voter data to help predict voter turnout. We also want to see which features are most important when predicting turnout. The scope of this project can be very helpful for multiple groups of people. Our problem can help Hamilton County Board of Elections, those running for office in Hamilton County, and Hamilton County voters, to properly identify individuals that may or may not vote and give some reasoning behind their choice. This information will be outlined in our algorithms below.

We set out weekly meetings to discuss our plans on assessing the data and which paths we would take to obtain our goals in finding trends within voter turnout. Each meeting we discussed a possible algorithm we could use to run these models, and we would work on the implementation to make sure these models would be able to work well on our data. Once we figured out how to implement the algorithms, we then focused on analyzing results to find the most accurate and well-trained model to bring our results into a real-world application.

The Nature of Our Data

Our data is the population of registered voters in Hamilton County, where each instance represents an individual and information about themselves. Our dataset had over 600,000 instances and 44 features. This information covers name, birth year, and a unique voter identification number, along with location data such as precinct name, precinct number, and address. There is also detailed information about government location classification data. This dataset also includes voter data such as their voter registration date and how they voted in all elections since 2020. We used all these features when implementing our algorithms. Because this is a government dataset, this allowed us to have specific and accurate amounts of data, with minimal missing values.

A special quality of our dataset is its ability to be updated every day. Hamilton County Board of Elections updates this dataset daily Monday-Friday before 9 am. This ensured that we were constantly downloading data that was up to date. As we got deeper into our project and evaluated our data, we began to realize the possibility of overfitting with our data. As we go through our approaches, we will show our answers and what we believe got us to this point.

Our Approaches

First, we started with loading in data and implementing preprocessing methods. We made sure to import the libraries needed to run these algorithms and ensure high data quality. These libraries included pandas, which we used for data manipulation, reading in our data, dropping NAs to keep readability consistent, and using functions such as “get_dummies()” for one hot encoding. We also used NumPy for mathematical operations such as checking counts of missing values. Scikit-learn allowed us to run the algorithms (decision tree, random forest, logistic regression, grid search, kmeans/kmodes clustering) we implemented by using a specific function within its own library. We used matplotlib to visualize the results of our algorithms, specifically the decision tree visualization.

When we started our project, we put a large focus on working with clustering to find hidden knowledge and patterns within our data through its unsupervised learning methods. However, after further investigation, we were unsuccessful when trying Kmeans and Kmodes clustering. We decided to shift our focus into regression. We used regression methods and predictive measures for voter turnout, with 0 being nonvoters, and 1 being voters. We attempted four different types of algorithms: decision tree, logistic regression, random forest, and grid search

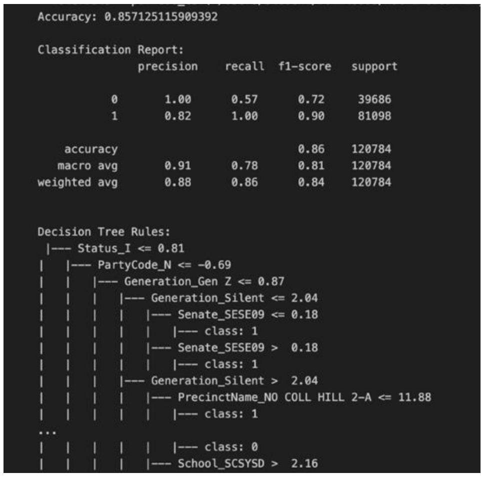
First Algorithm: Decision Tree

Our first algorithm run was a decision tree. Through our decision tree, we were looking to find which features within our dataset were most important when predicting voter turnout. Our decision tree took about a minute to run, so it was not very time consuming. Within our decision tree, we used one hot encoding to transform our categorical feature values into binary values. Matplotlib allowed us to visualize the splits of features within our data by outputting the decision.

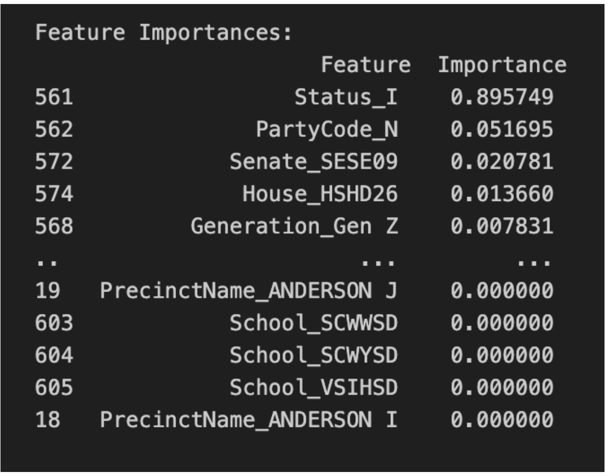
Decision Tree 1

Originally, we focused on including everything but our target column feature, which was the 2024 election data instance. From this, we got an overall 86% accuracy within our model. This Decision Tree model analyzes all the other columns and splits on different values which is used for predicting someone to vote in the 2024 election. Our model gave us some interesting accurate results with 100% recall for predicting voters to vote and 100% precision of voters to not vote. Here, there is a possibility of overfitting, partially due to the skewness of our data. This means that our model favors predicting voters to vote and does not account for about 43% of non-voters. Our model was 82% correct when predicting someone to vote. Fortunately, our model predicts very well for the voters that voted but the model has room for improvement to not forget about the remaining non-voters that need to be classified correctly. Our results also gave us the important feature columns where inactivity status, nonpartisan voter status, and Cincinnati Senate-9 seemed to be biggest feature factors in our model.

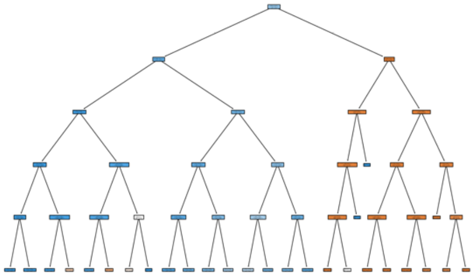
We also focused heavily on our support column, where our support column represents the number of actual instances for each class in the dataset. For example, we had 39686 people in Class 0, and 81098 people in Class 1.



```
Accuracy: 0.857125115909392

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.57      0.72     39686
           1       0.82      1.00      0.90     81098

    accuracy                           0.86    120784
   macro avg       0.91      0.78      0.81    120784
weighted avg       0.88      0.86      0.84    120784


Decision Tree Rules:
 |--- Status_I <= 0.81
|   |--- PartyCode_N <= -0.69
|   |   |--- Generation_Gen Z <= 0.87
|   |   |   |--- Generation_Silent <= 2.04
|   |   |   |   |--- Senate_SESE09 <= 0.18
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- Senate_SESE09 >  0.18
|   |   |   |   |   |--- class: 1
|   |   |   |--- Generation_Silent >  2.04
|   |   |   |   |--- PrecinctName_NO COLL HILL 2-A <= 11.88
|   |   |   |   |   |--- class: 1
...
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- School_SCSYSD >  2.16
```

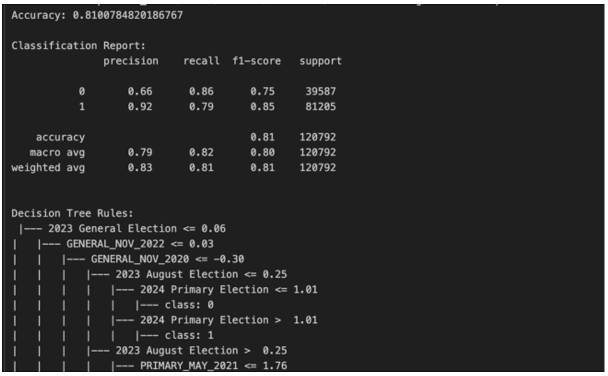


```
Feature Importances:
                          Feature  Importance
561                      Status_I    0.895749
562                   PartyCode_N    0.051695
572                 Senate_SESE09    0.020781
574                  House_HSHD26    0.013660
568              Generation_Gen Z    0.007831
..                            ...         ...
19   PrecinctName_ANDERSON J    0.000000
603                 School_SCWWSD    0.000000
604                 School_SCWYSD    0.000000
605                 School_VSIHSD    0.000000
18   PrecinctName_ANDERSON I    0.000000
```

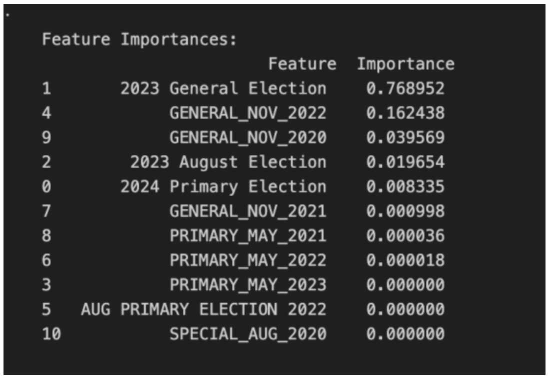


Decision Tree 2

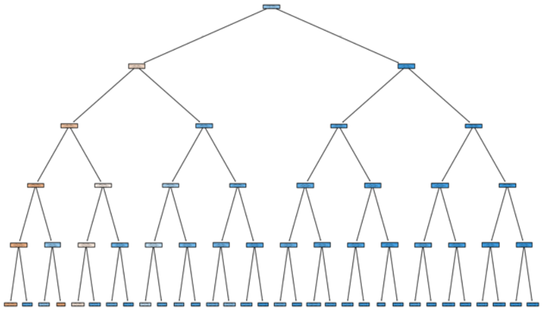
In addition, we ran a new Decision Tree algorithm that only put election types in our feature columns. We found that the overall model was 81% accurate. This model is better at predicting people to not vote out of actual people that did not vote. Out of all our model predictions for this decision tree, we found that it is also good at predicting accurately that they did vote. The elections that the model found to be most impactful were the 2023 General Election, 2022 General November Election and 2020 General November Election.



```
Accuracy: 0.8100784820186767

Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.86      0.75     39587
           1       0.92      0.79      0.85     81205

    accuracy                           0.81    120792
   macro avg       0.79      0.82      0.80    120792
weighted avg       0.83      0.81      0.81    120792


Decision Tree Rules:
 |--- 2023 General Election <= 0.06
|   |--- GENERAL_NOV_2022 <= 0.03
|   |   |--- GENERAL_NOV_2020 <= -0.30
|   |   |   |--- 2023 August Election <= 0.25
|   |   |   |   |--- 2024 Primary Election <= 1.01
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- 2024 Primary Election >  1.01
|   |   |   |   |   |--- class: 1
|   |   |   |--- 2023 August Election >  0.25
|   |   |   |   |--- PRIMARY_MAY_2021 <= 1.76
```



```
Feature Importances:
                       Feature  Importance
1        2023 General Election    0.768952
4             GENERAL_NOV_2022    0.162438
9             GENERAL_NOV_2020    0.039569
2         2023 August Election    0.019654
0        2024 Primary Election    0.008335
7             GENERAL_NOV_2021    0.000998
8             PRIMARY_MAY_2021    0.000036
6             PRIMARY_MAY_2022    0.000018
3             PRIMARY_MAY_2023    0.000000
5    AUG PRIMARY ELECTION 2022    0.000000
10            SPECIAL_AUG_2020    0.000000
```



When comparing our 2 decision tree models we found that both models were close in accuracy, however we received better results when including all features in the dataset. To sum up, past election voting history is not enough when predicting turnout as our model performed better when including other categorical type columns as well.

In addition, because of no “100%” values within decision tree 2, we thought to base the rest of our data off the same feature columns from this tree. However, we have decided to use the features from the first decision tree as a basis for the rest of our project because when using decision tree 2, we found that all values from algorithm to algorithm were providing us with the exact same accuracy scores (a possibility from overfitting).

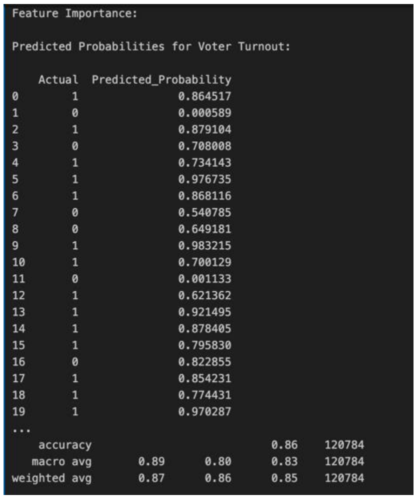
We also focused heavily on our support column, where our support column represents the number of actual instances for each class in the dataset. For example, we had 39587 people in Class 0, and 81098 people in Class 1. This let us know throughout our project that although similar values were being output, they were still slightly different in several instances.

Second Algorithm: Logistic Regression

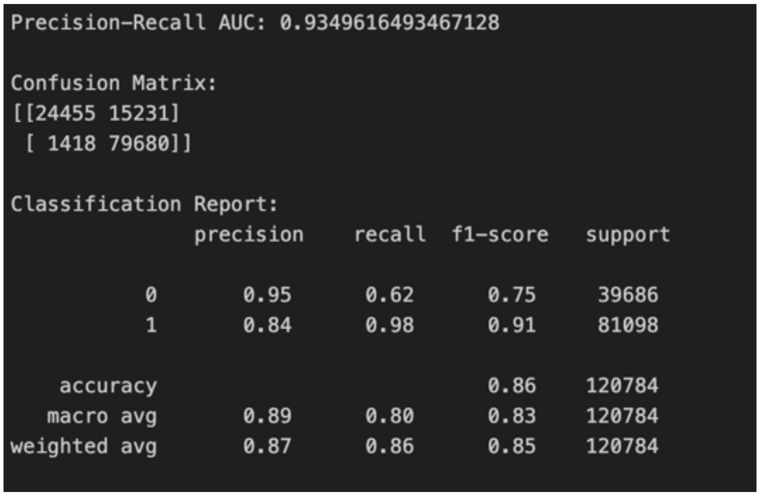
Our next algorithm was Logistic Regression. This model was used to predict the individual probabilities of each instance’s voting type (voted or did not vote). Our Logistic Regression model took us about 3 minutes to run.

As seen below, the model was 86% accurate when predicting these probabilities. We also included a confusion matrix, precision and recall measures to analyze the accuracy of the model.

The model clearly has a very high recall score for predicting voters to vote. This would signify that out of all voters that voted, the model is very good at predicting those people to vote. It also has a high precision for those that did not vote. This would represent that when our model predicts someone to not vote, it was typically correct in that prediction.



```
Feature Importance:

Predicted Probabilities for Voter Turnout:

    Actual  Predicted_Probability
0        1               0.864517
1        0               0.000589
2        1               0.879104
3        0               0.708008
4        1               0.734143
5        1               0.976735
6        1               0.868116
7        0               0.540785
8        0               0.649181
9        1               0.983215
10       1               0.700129
11       0               0.001133
12       1               0.621362
13       1               0.921495
14       1               0.878405
15       1               0.795830
16       0               0.822855
17       1               0.854231
18       1               0.774431
19       1               0.970287
...
    accuracy                           0.86    120784
   macro avg       0.89      0.80      0.83    120784
weighted avg       0.87      0.86      0.85    120784
```



```
Precision-Recall AUC: 0.9349616493467128

Confusion Matrix:
[[24455 15231]
 [ 1418 79680]]

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.62      0.75     39686
           1       0.84      0.98      0.91     81098

    accuracy                           0.86    120784
   macro avg       0.89      0.80      0.83    120784
weighted avg       0.87      0.86      0.85    120784
```

Third Algorithm: Random Forest

Our third algorithm was the random forest algorithm. Our random forest model ran multiple decision trees to find the best combination of features. We ended up using this model's precision and recall values to show how our model could be used in the real world, in addition to the features of most importance for our data. This algorithm took about 4 minutes to run.

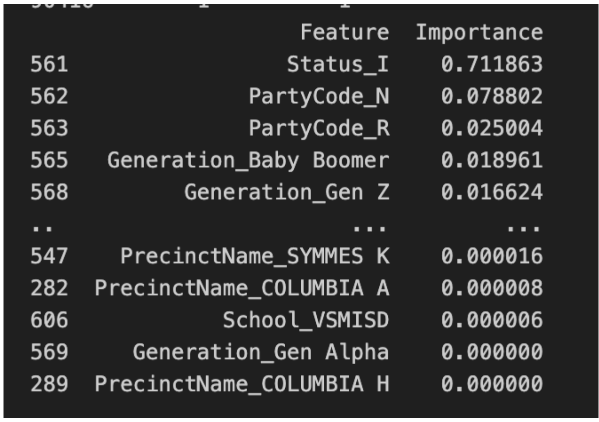
The decision tree has shown us that the features of most importance include inactive voters, party code of nonpartisan, and party code of republican. Two generations of importance were also Baby Boomers and Gen Z.

For non-voters (class 0), the confusion matrix analysis shows that the model predicts class 0 with a precision of 0.93 and a recall of 0.63. The precision of non-voters in this instance indicates that 93% of those predicted as non-voters are non-voters. Recall shows that 63% of actual non-voters were correctly identified by the model. When comparing both precision and recall, our model is effective at identifying non-voters out of all instances.

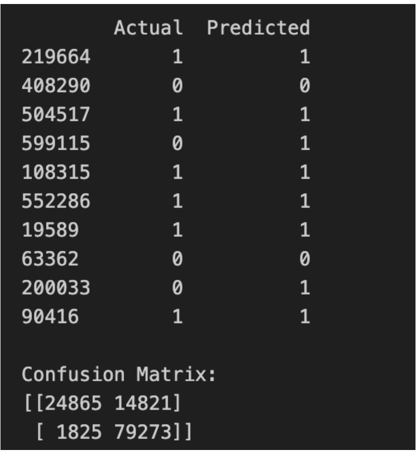
For voters (class 1), the confusion matrix analysis shows that the model predicts class 1 with a higher precision of 0.84 and a recall of 0.98. This means 84% of those predicted as voters are actual voters, and the model successfully identifies 98% of actual voters. Our recall value hints that our model is very good at identifying voters who will vote, insisting that all votes are captured.

Regarding real-world implications, the best use of our data would be using our high precision rate (0.93) for class 0 (non-voters). This can help a candidate focus on targeting likely non-voters with confidence. This could be useful in designing campaigns to convert non-voters into voters or understanding barriers to participation. On the other hand, our class 1 recall value of 0.98 could be incredibly useful in ensuring voter turnout by targeting the 98% of voters that were correctly identified as likely to participate on election day

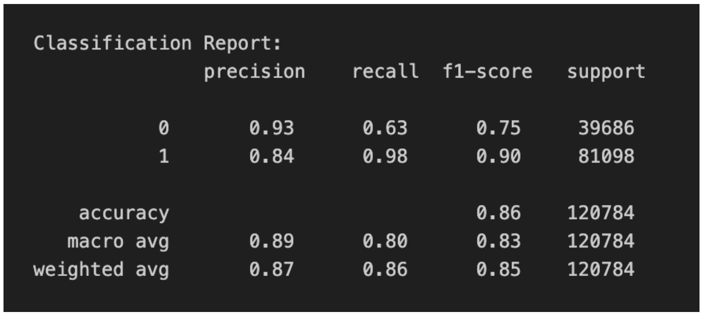
With an overall accuracy of 86%, the model can help identify likely voters and non-voters with reasonable confidence.



```
                    Feature  Importance
561                Status_I    0.711863
562             PartyCode_N    0.078802
563             PartyCode_R    0.025004
565   Generation_Baby Boomer    0.018961
568         Generation_Gen Z    0.016624
..                       ...         ...
547     PrecinctName_SYMMES K    0.000016
282  PrecinctName_COLUMBIA A    0.000008
606             School_VSMISD    0.000006
569     Generation_Gen Alpha    0.000000
289  PrecinctName_COLUMBIA H    0.000000
```



```
        Actual  Predicted
219664       1          1
408290       0          0
504517       1          1
599115       0          1
108315       1          1
552286       1          1
19589        1          1
63362        0          0
200033       0          1
90416        1          1

Confusion Matrix:
[[24865 14821]
 [ 1825 79273]]
```



```
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.63      0.75     39686
           1       0.84      0.98      0.90     81098

    accuracy                           0.86    120784
   macro avg       0.89      0.80      0.83    120784
weighted avg       0.87      0.86      0.85    120784
```

Fourth Algorithm: Grid Search (2-fold and 5-fold)

Our fourth, and final algorithm we created was our grid search algorithm. Grid search allowed for us to find the best hyperparameters for our decision tree model. We ran our algorithm with both a 2-fold cross validation, and a 5-fold cross validation. However, this was very time consuming, as our model was totaling 144 & 360 folds. It took hours for this algorithm to run, but it provided us with plenty of information on providing the best methods for the most optimal model.

Grid Search

For our grid search, we used decision tree hyperparameters. For or two-fold grid search, our best hyperparameter algorithm recommends using the gini for splitting nodes, no specified max depth of the tree, a minimum of 5 samples per leaf, and a minimum of 2 samples required to split an internal node. These hyperparameters can be used to identify relationships and balance our tree depth.

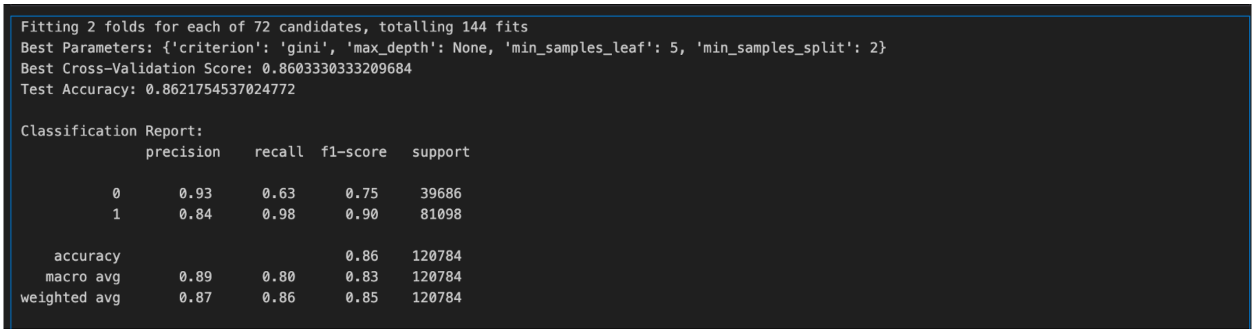
Our cross-validation results are promising, as they show there is consistent performance across folds. Our best cross validation accuracy is 0.8603, with a test accuracy of 0.8621.

Our five-fold grid search with our best hyperparameters recommends using the entropy for splitting nodes, no specified max depth of the tree, a minimum of 5 samples per leaf, and a minimum of 2 samples required to split an internal node. These hyperparameters can be used to identify relationships and balance our tree depth.

Our cross-validation results are promising, as they show there is consistent performance across folds. Our best cross validation accuracy is 0.8611, with a test accuracy of 0.8622.

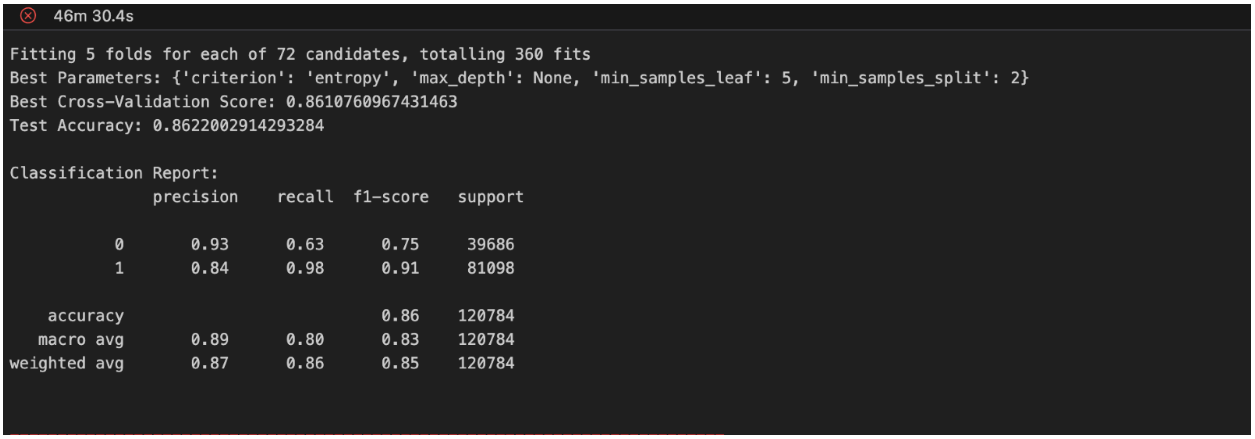
The main difference between our two-fold and five-fold is splitting criteria in five-fold is to use entropy and in two-fold to use Gini impurity. This provided much information regarding our data, which we found useful for decision trees since the basis two algorithms that we have ran have been decision trees and random forest.

Grid Search 2-Fold



```
Fitting 2 folds for each of 72 candidates, totalling 144 fits
Best Parameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split': 2}
Best Cross-Validation Score: 0.8603330333209684
Test Accuracy: 0.8621754537024772

Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.63      0.75     39686
           1       0.84      0.98      0.90     81098

    accuracy                           0.86    120784
   macro avg       0.89      0.80      0.83    120784
weighted avg       0.87      0.86      0.85    120784
```

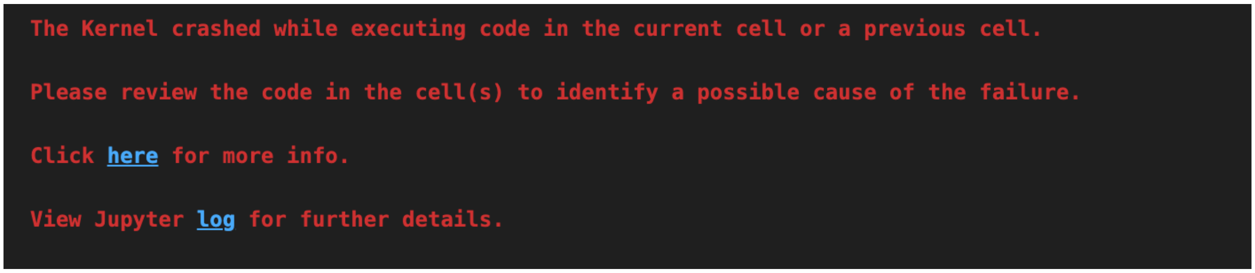
Grid Search 5-Fold



```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 5, 'min_samples_split': 2}
Best Cross-Validation Score: 0.8610760967431463
Test Accuracy: 0.8622002914293284

Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.63      0.75     39686
           1       0.84      0.98      0.91     81098

    accuracy                           0.86    120784
   macro avg       0.89      0.80      0.83    120784
weighted avg       0.87      0.86      0.85    120784
```

Algorithms We Did Not Include

Lastly, we had some algorithms that did not perform as we would have liked. As mentioned earlier, we had attempted to focus on clustering. We wanted to use the clustering methods to analyze distances between centroids of the clusters to see if our data had any hidden patterns. However, when we tried KModes and Kmeans clustering algorithms, they did not respond in the way we wanted them to. KModes was either not running or taking a long time. KMeans crashed our kernel as we tried to run the model. We quickly shifted our focus over to regression, and happily, revealed a ton of useful information about our data, as well as producing highly accurate predictive models!



```
The Kernel crashed while executing code in the current cell or a previous cell.

Please review the code in the cell(s) to identify a possible cause of the failure.

Click here for more info.

View Jupyter log for further details.
```

Results

When evaluating and comparing our models, we found that all of our models had very similar accuracy scores. To ensure that we were not calling for the same values we went in a deeper dive into our models to see what may possibly be causing this issue. First, we looked into the training and testing sets we had created to ensure that we were outputting separate classification reports for each algorithm. Then, we looked to our support instances and precision and recall scores. Here, we noticed that we had slight differences in our accuracy scores, AUC ROC, support numbers and confusion matrix distributions to let us know that indeed they were different in some way. These results opened a new light for our future work investigation. The results are convincing us that our model could be overfitting and or both, skewing the accuracies due to feature importances.

Despite these findings, we still believe our models are useful enough to predict voter turnout. Although all our model results were very similar, we believe that our best model is logistic regression. Due to its real-world implications whether it is targeting non-voters or voters, the ability to distinguish between each instance individually provides a ton of information for parties to seek out using models. In this case as mentioned above the logistic regression model is good at correctly identifying who is likely to participate in voting. Our model is also giving us the probabilities of each individual’s likelihood of voting which would be useful to distinguish between people who do not vote at all, vote every now and then, or are likely to vote. Voting campaigns or initiatives would be able to use our model to assess the likelihood of someone voting and doing so accurately, which was the intent of us choosing this dataset.

Potential future work

Regarding future work and practices, there are a few things we could expand upon in the future. The first is the information in our dataset. Due to our overfitting, we would definitely look forward to implementing different data within our models to see if this outputs anything different. In addition, as seen above, many of our important features laid out in our dataset were location based. Upon further discovery, we ended up finding another dataset that we could one day combine with our current dataset to give even more location information. Data from the Ohio Secretary of State not only holds data from elections in 2000 and on but includes more specific location data. This data gets as specific as cities, townships, villages, and wards. This data could allow us to be even more accurate about important areas within Hamilton County or the rest of Ohio. An example would be for our Decision Tree where we would be able to pair this data with Ohio Secretary of State which could give us more information about individuals, we would expect our accuracy scores to increase.

Another thing we had mentioned as of last week was combining our categorical columns and voting columns within our decision tree for our feature columns. After further investigation, we were able to do this by creating a variable that included all instances within our set, minus our target value. This gave us a high accuracy rate of 86%. (Explained further in our decision tree section and figures 1, 2, & 3)

Lastly, something we could do in the future is build out correct clustering algorithms to assess our data in a different way. We were running into some time-complexity issues within our first couple weeks of work, but if we can incorporate methods like PCA dimensionality reduction and proper clustering algorithms we would expect better and more accurate results.