An Automated Ticket-Writing Machine!

By: Benjamin Kosiborod and Victoria Staada

Problem Statement

Given information about a car and its parking spot, we wanted to predict the type of violation that is ticketed

Dataset

NYC OpenData

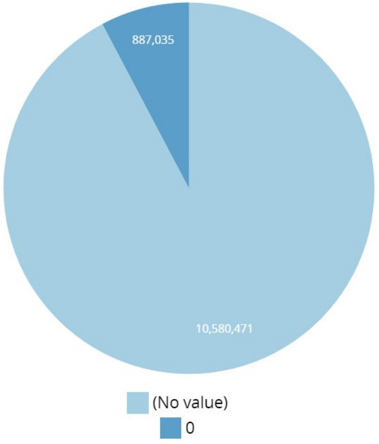
- NYC Parking Violations Issued - Fiscal Year 2019
  - ~11.5 million rows
  - 43 columns
  - 99 violation codes
    1. No Parking Street Cleaning Code 21 - **1,803,467 Violations (~16%)**
    2. Failure to Display Muni-Meter Receipt Code 38 - **1,165,883 Violations (~10%)**
    3. Photo School Zone Speed Violation Code 36 - **1,098,298 Violations (~10%)**
    4. No Standing Day Time Limits Code 14 - **1,013,584 Violations (~9%)**
    5. No Parking Day Time Limits Code 20 - **795,751 Violations (~7%)**

    = **~51%**

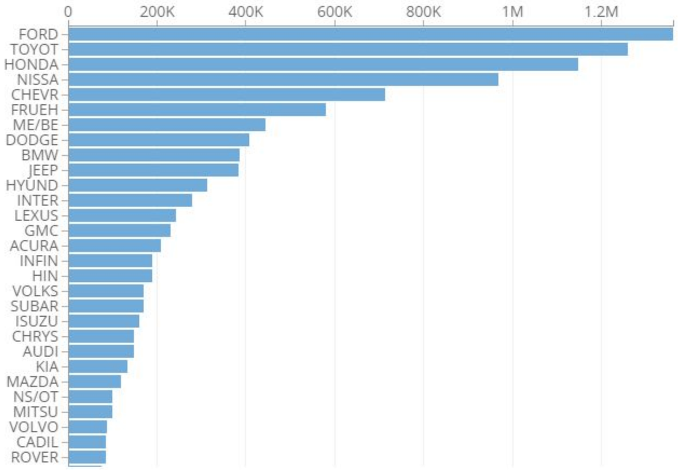
Approach & Methodology

- Data cleanup
  - Dropped columns we didn’t need/were unhelpful (43 → 17)
  - Dropped rows with bad data for our features (~11.5M → ~4.5M)
  - Imputed clean data when applicable (e.g. standardizing abbreviations)
    - Used Open Data NYC’s visualization tool to help
  - Dropped all rows with NaN values
  - Reset index
  - Clean data! 🧼
- Model training & testing
  - Sampled 100K rows from our data
  - Encoded data
  - Removed outliers
  - Trained & tested kNN, Decision Trees, Random Forest, AdaBoost, and FFNN models
  - Calculated metrics for each individual model
  - Plotted an ROC curve for all models

Unregistered Vehicles?



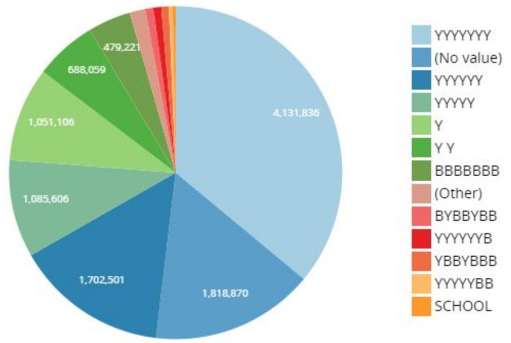
Vehicles Make



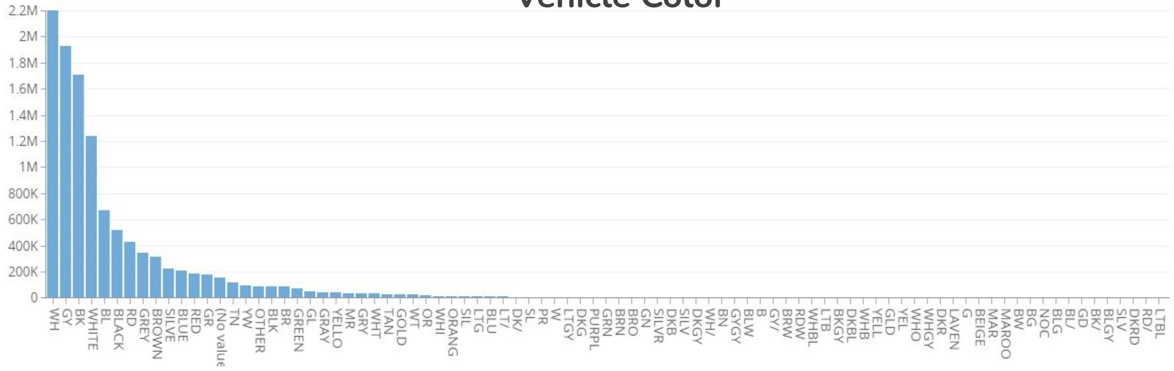
Challenges

- Poorly documented data
  - Days Parking In Effect (BBBBBBB, YYYYYYY, YBBYBBB, etc.)
  - 1000 vehicle colors recorded (W, WH, WHTE, WT, WHI, WHIE, etc.)
    - 12 colors
      - Red, Orange, Yellow, Green, Blue, Purple, Beige, Brown, Gray, Silver, White, Black
- Size of dataset/computing power
  - Personal computers ❌
  - Google Colab ❌
  - NEU Discovery ✔️
- NEU Discovery
  - Time limits
  - Memory limits
  - Custom Environment

Days Parking In Effect



Vehicle Color

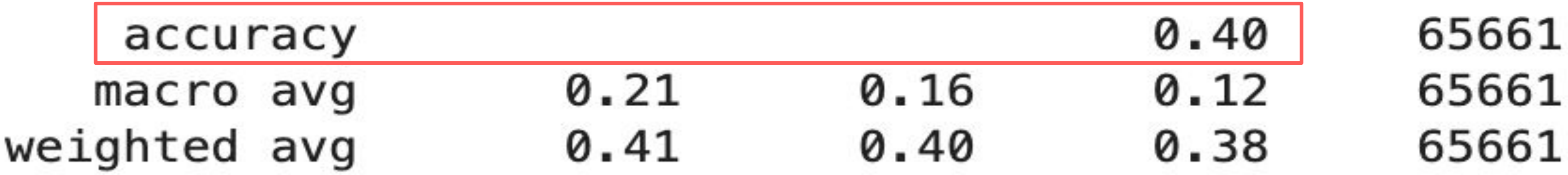


k-nearest neighbors

Default (k=5) worked best

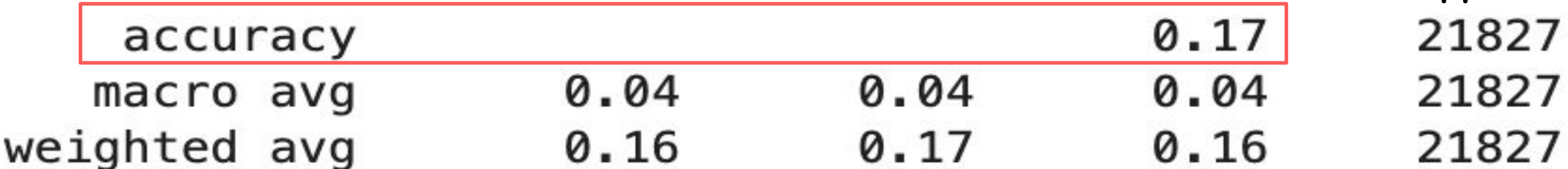
Expectedly low accuracy... our worst model by far.

Training



```
              Precision   Recall   F1 Score   Support
    accuracy                          0.40      65661
   macro avg     0.21      0.16       0.12      65661
weighted avg     0.41      0.40       0.38      65661
```

Testing



```
              Precision   Recall   F1 Score   Support
    accuracy                          0.17      21827
   macro avg     0.04      0.04       0.04      21827
weighted avg     0.16      0.17       0.16      21827
```

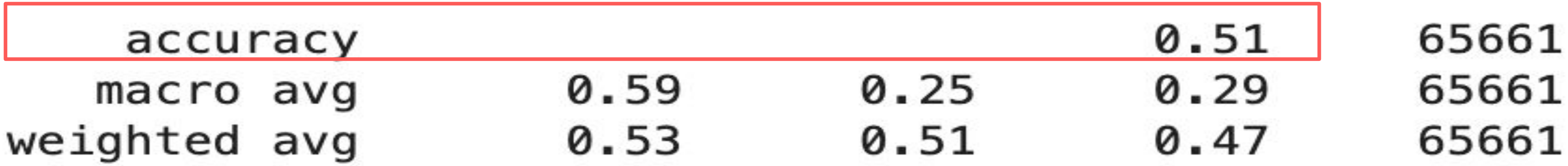
Decision Trees

max_depth = 13 worked best for our data

Our second best model. Likely overfitted the least.

Training

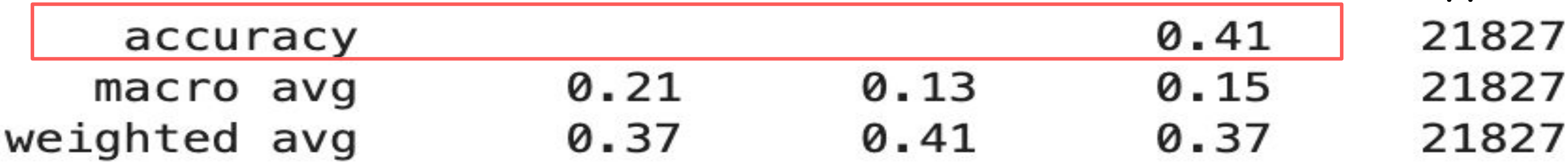
Precision Recall F1 Score Support



```
    accuracy                           0.51     65661
   macro avg       0.59      0.25      0.29     65661
weighted avg       0.53      0.51      0.47     65661
```

Testing

Precision Recall F1 Score Support



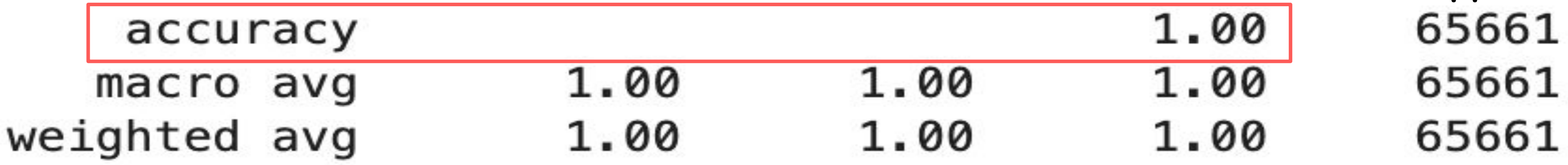
```
    accuracy                           0.41     21827
   macro avg       0.21      0.13      0.15     21827
weighted avg       0.37      0.41      0.37     21827
```

Random Forest (bagging)

Experimented with some hyperparameters (max_depth, criterion, n_estimators)

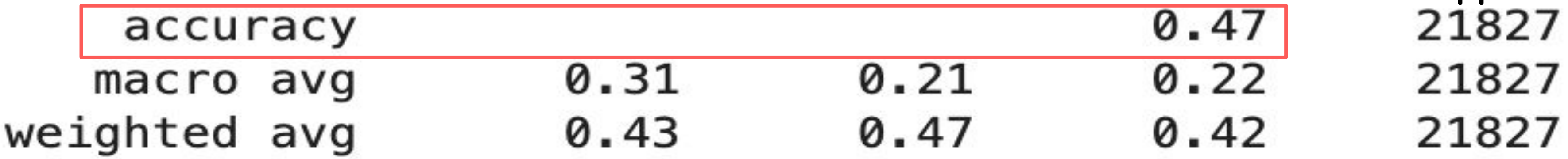
Our best model, but likely overfits to the training data. Perhaps additional parameter tuning is in order...

Training



```
                 Precision    Recall    F1 Score    Support
    accuracy                                1.00      65661
   macro avg          1.00      1.00        1.00      65661
weighted avg          1.00      1.00        1.00      65661
```

Testing



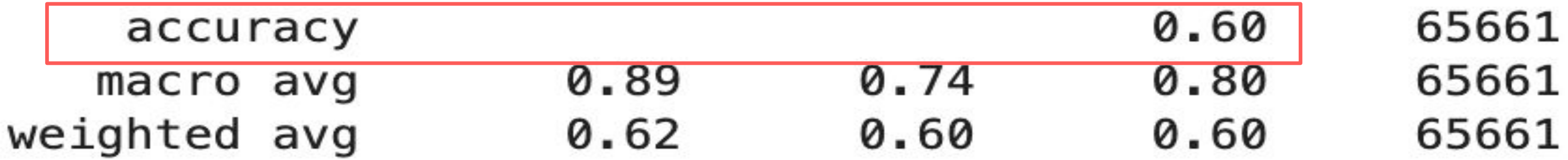
```
                 Precision    Recall    F1 Score    Support
    accuracy                                0.47      21827
   macro avg          0.31      0.21        0.22      21827
weighted avg          0.43      0.47        0.42      21827
```

AdaBoost

Used our Decision Tree estimator from earlier as our base estimator

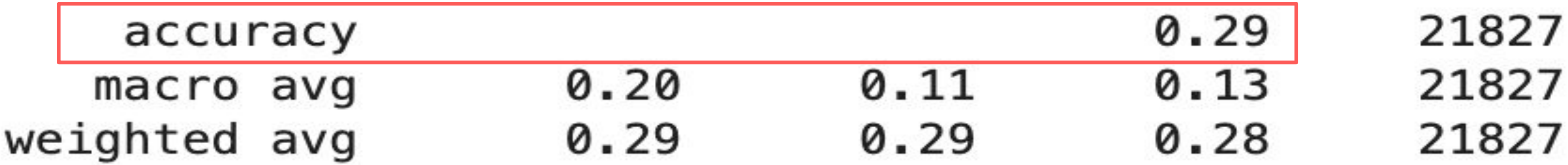
Unexpectedly low accuracy... Perhaps the dataset is more suited to bagging methods

Training



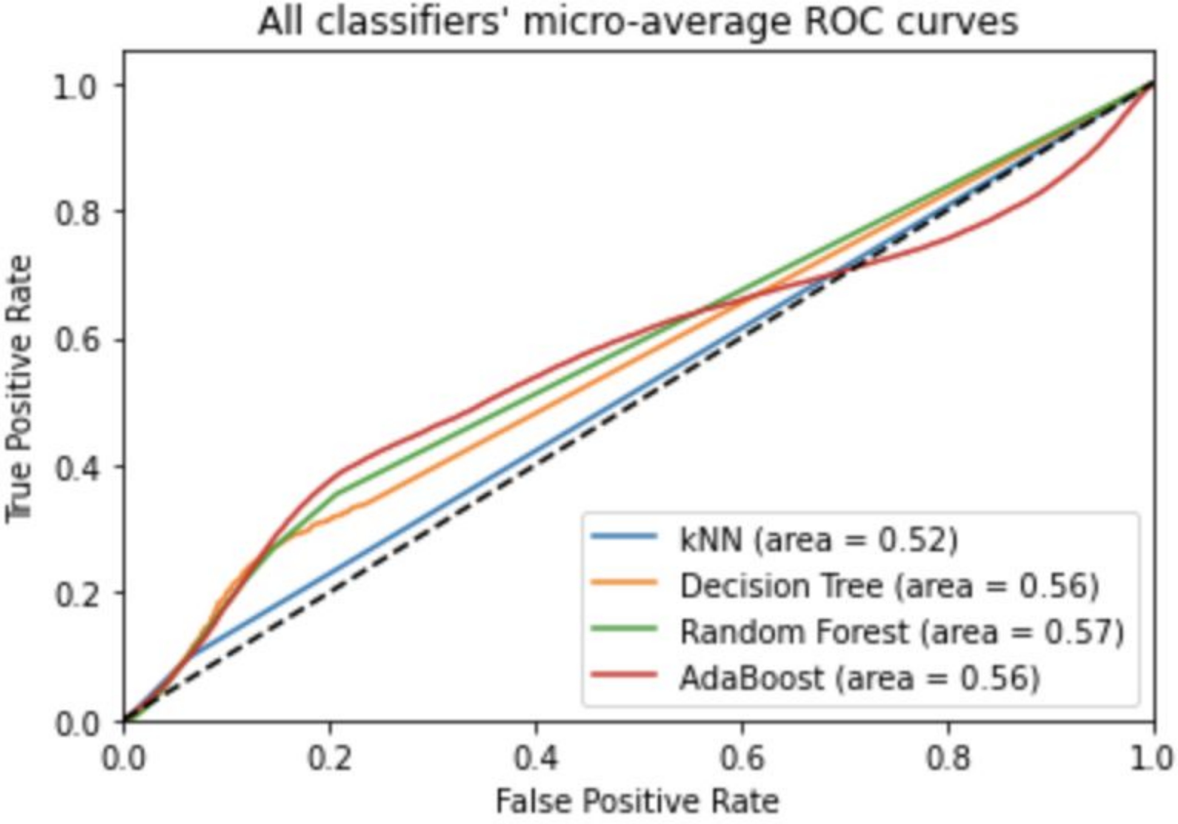
```
              Precision   Recall   F1 Score   Support
    accuracy                           0.60     65661
   macro avg       0.89     0.74       0.80     65661
weighted avg       0.62     0.60       0.60     65661
```

Testing



```
              Precision   Recall   F1 Score   Support
    accuracy                           0.29     21827
   macro avg       0.20     0.11       0.13     21827
weighted avg       0.29     0.29       0.28     21827
```

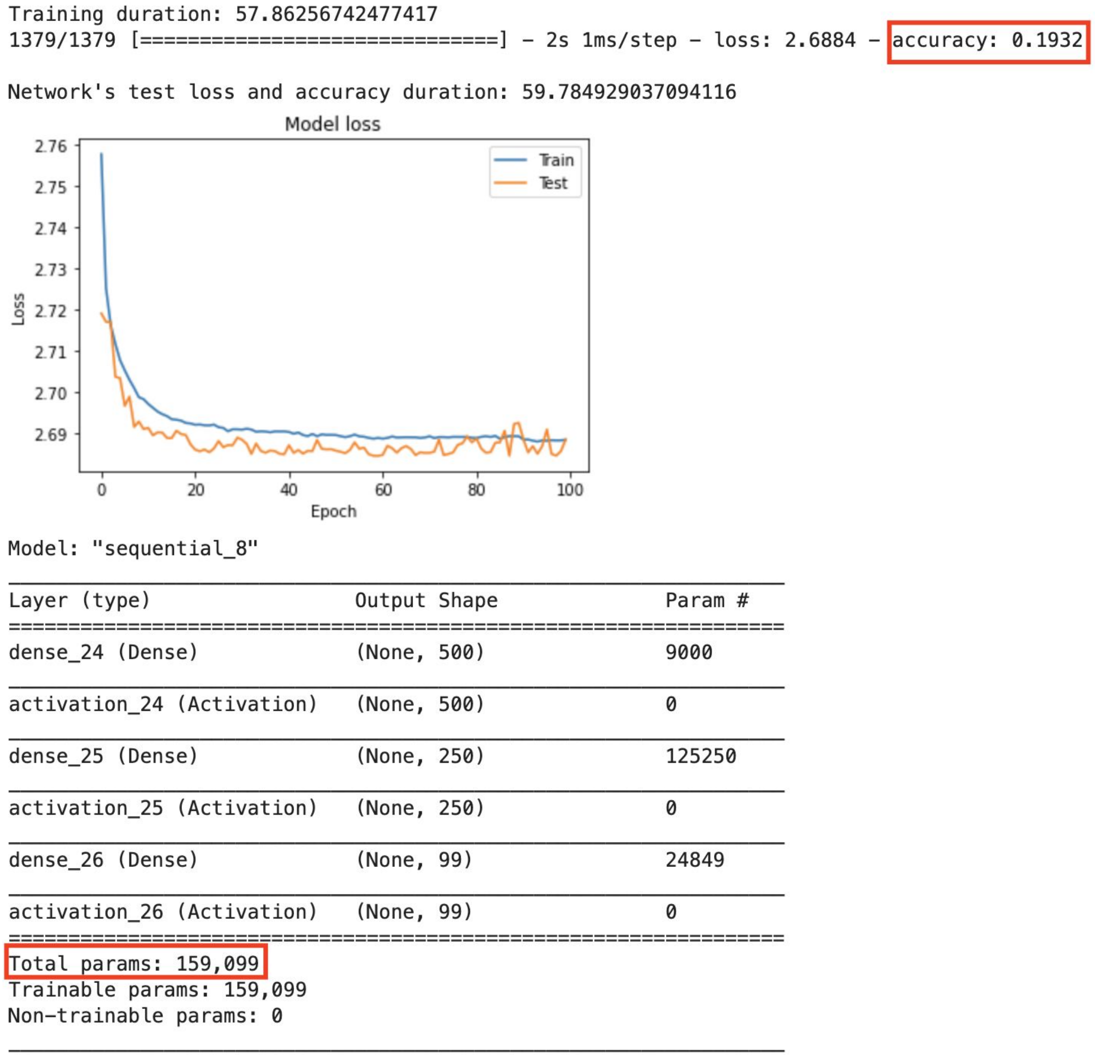
ROC Curve


All classifiers' micro-average ROC curves
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
kNN (area = 0.52)
Decision Tree (area = 0.56)
Random Forest (area = 0.57)
AdaBoost (area = 0.56)

Neural Network (FFNN)

Accuracy: ~19%
Dimensions
Input size: 17 (# of features)
Output size: 99 (# of classes)
# params: 159,099

Activation functions: sigmoid, softmax



```
Training duration: 57.86256742477417
1379/1379 [==============================] - 2s 1ms/step - loss: 2.6884 - accuracy: 0.1932

Network's test loss and accuracy duration: 59.784929037094116
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 500)               9000
_________________________________________________________________
activation_24 (Activation)   (None, 500)               0
_________________________________________________________________
dense_25 (Dense)             (None, 250)               125250
_________________________________________________________________
activation_25 (Activation)   (None, 250)               0
_________________________________________________________________
dense_26 (Dense)             (None, 99)                24849
_________________________________________________________________
activation_26 (Activation)   (None, 99)                0
=================================================================
Total params: 159,099
Trainable params: 159,099
Non-trainable params: 0
_________________________________________________________________
```

Questions?