Predicting Outcomes of Allegations:

NYPD Civilian Complaints

Summary of Findings

Introduction

In this project, we will be predicting the outcome of an allegation using the NYPD Civilian Complaints dataset. This is a classification prediction problem, because the outcome of an allegation is a categorical variable. There are two possible allegation outcomes based on what is available in the dataset. The outcome can either be in the favor of the complainant or not. After observing the dataset, an "unsubstantiated" or "exonerated" outcome is not in the favor the complainant, since that means the person **being** accused has evidence of innocence or no evidence of being guilty. The "substantiated" outcome is in favor of the complainant, because that means there is evidence that provides truth to the allegation. These two possible outcomes means we will be performing a binary classification. We chose to predict the outcome of an allegation, because it is very valuable for someone to know how an allegation can carry out so that they do not waste time if there is no chance of the complainant winning their case. It can save a lot of time for all parties. We are choosing accuracy as our evaluation metric, because it is the most common one for classification predictions. Accuracy is best, because we are very interested in knowing how "correct" our prediction model is. This is very important in evaluating it, so that we know if we can properly classify an allegation. If it is wrong at an alarming rate, then this would be bad if someone were to use it to decide whether or not to actually file an allegation. If our model classifies it as unsubstantiated when it really is substantiated, then an officer is possibly off the hook for mistreating that civilian. If our model classifies it as substantiated when it really is unsubstantiated, then time would be wasted.

Baseline Model

The features we will be first using in our baseline model is the fado_type, complainant_ethnicity, mos_ethnicity, complainant_age_incident, and year_closed. There will need to be some data cleaning using these five columns. For the complainant_ethnicity column, all the NaN, "Refused," and "Other Race" values will be replaced with "Unknown." We are replacing "Other Race" as well, because this category is way too vague to predict accurately for a complainant that does not have a specific race for the prediction model. The board_disposition column also needs to be cleaned, in the sense that the parentheses after the outcomes can be taken away, since we are counting substantiated as in favor of the complainant and unsubstantiated and exonerated as in disfavor of the complainant.

The features of fado_type, complainant_ethnicity, and mos_ethnicity in our model are nominal categorical variables, as there is no inherent order to ethnicity, allegation outcome, or reason for the allegation. Therefore, we will be performing one-hot encodings on all these columns. This will be through a ColumnTransformer. The features of complainant_age_incident and year_closed are quantitative variables. We are performing StandardScaler transformers on these variables. We will also be using a Decision Tree Classifier to classify our prediction.

The performance of our model is overfitting to the training data, with 85.21% accuracy. However, to the testing data (unseen data), it has a 72.55% accuracy. The current model isn't terrible, but it can be better.

Ideally, we want to have as high of an accuracy as possible, so anything above 72.55% for unseen data would be better.

Final Model

We added the mos_age_incident and mos_gender categorical features to our prediction model. We thought these features would be good to add, since the age of an officer can be used to justify the truth of an allegation. If an older officer has had a stunning reputation with behavior and whatnot, especially with more time on the job, the truth of an allegation may be doubted more. We chose to use mos_gender, because society sees female officers as "nicer," so allegations on them may be doubted.

The model we chose is the Decision Tree Classifier with a max depth of 10 and a minimum number of samples split of 7. To select this, we used GridSearchCV. The testing accuracy ended up being around 79.66%, which has been the highest percentage accuracy among all the models and features we tried out, although not by too much.

Fairness Analysis

In our Fairness Analysis, we were trying to see if our model would be fair or more biased in predictions for male or female complainants. We only wanted to observe male and female-identifying complainants in order to make analysis simpler, since these two gender categories were the most commonly found in the dataset. We then conducted a permutation test to see if any differences in accuracy between the two groups is significant.

Null Hypothesis: Our model is fair. Its precision for male complainants and female complainants are roughly the same, and any differences are due to random chance.

Alternative Hypothesis: Our model is unfair. Its precision for female complainants is lower than its precision for male complainants.

We concluded with a p-value of 0.52. With our chosen significance level of 0.05, this means we fail to reject our null hypothesis, since our p-value is greater than our significance level. This means that we don't have sufficient evidence to prove our model is unfair. It is possible that our differences are due to random chance.

Code

In [213…

```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

Baseline Model

In [214…

```
#imports
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [215…

```
fp = os.path.join('allegations.csv')
allegations = pd.read_csv(fp)

# board_disposition
def clean_sub(word):
    if 'Substantiated ' in word:
        return word[:13]
    else:
        return word

# change unsub..., exonerated, and sub... to 0's and 1's
allegations['board_disposition'] = allegations['board_disposition'].apply(clean_sub)
allegations['board_disposition'] = allegations['board_disposition'].replace({'Substantiate
```

In [216…

```
#cleaning complainant ethnicity (mos ethn. does not need to be cleaned)
allegations['complainant_ethnicity'] = allegations['complainant_ethnicity'].fillna('Unknow
allegations['complainant_ethnicity'] = allegations['complainant_ethnicity'].replace({'Othe
```

In [217…

```
# imputation on complainant_age_incident column
# replaced missing values with the column mean
allegations['complainant_age_incident'] = allegations['complainant_age_incident'].fillna(a
```

In [218…

```
X = allegations[['complainant_ethnicity', 'mos_ethnicity', 'fado_type', 'complainant_age_i
y = allegations['board_disposition']

# One-hot encodings
pipe = Pipeline([
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
])

categ_cols = ['complainant_ethnicity', 'mos_ethnicity', 'fado_type']

# Transformer pipeline
ct = ColumnTransformer([
    ('std-scaler', StandardScaler(), ['complainant_age_incident', 'year_closed']),
    ('categorical_columns', pipe, categ_cols)
])

# create pipeline for the model
pl = Pipeline([('ct', ct), ('dtc', DecisionTreeClassifier())])
```

In [219…

```
# train_test_split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# fit the model
pl.fit(X_train, y_train)

# get accuracy of the model on the testing data
pl.score(X_test, y_test)
```

Out[219…

```
0.7255395683453237
```

In [220…

```
# get accuracy of the model on the training data
# overfitting!
pl.score(X_train, y_train)
```

Out[220…
```
0.8521464545527221
```

Final Model

In [306…
```
# two new features added on top of CATEGORICAL features from baseline model
# added: mos_age_incident and mos_gender
```

In [222…
```
X = allegations[['complainant_ethnicity', 'mos_ethnicity', 'fado_type', 'mos_age_incident'
y = allegations['board_disposition']

# One-hot encodings
pipe = Pipeline([
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False)),
])

categ_cols = ['complainant_ethnicity', 'mos_ethnicity', 'fado_type', 'mos_gender']

# Transformer pipeline
ct = ColumnTransformer(
    transformers = [
        ('std-scaler', StandardScaler(), ['mos_age_incident']),
        ('categorical_columns', pipe, categ_cols)
    ]
)

# Pipeline for the model
pl = Pipeline([('ct', ct), ('dtc', DecisionTreeClassifier())])
```

In [223…
```
# train_test_split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20)

# fit the model
pl.fit(X_train, y_train)

# get accuracy of the model on the testing data
pl.score(X_test, y_test)
```

Out[223…
```
0.7468525179856115
```

In [224…
```
# get accuracy of the model on the training data

pl.score(X_train, y_train)
```

Out[224…
```
0.7679682230382973
```

Grid Search Time!

In [262…
```
# import
from sklearn.model_selection import GridSearchCV
```

In [272…
```
# Performing GridSearchCV
```

```
categ = ['rank_abbrev_incident', 'rank_abbrev_now', 'rank_now', 'rank_incident',
        'mos_ethnicity', 'mos_gender', 'complainant_ethnicity', 'complainant_gender',
        'fado_type', 'allegation', 'contact_reason']
quant_cols = ['shield_no', 'month_received', 'year_received', 'month_closed',
             'year_closed', 'mos_age_incident', 'complainant_age_incident']

X = allegations[categ+quant_cols]
y = allegations['board_disposition']

categoricals = Pipeline([
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse=False))
])

# Transformer pipeline
ct = ColumnTransformer([
    ('as-is', 'passthrough', quant_cols),
    ('categ_cols', categoricals, categ)
])

# Pipeline for the model
pl = Pipeline([('ct', ct), ('dtc', DecisionTreeClassifier())])
```

In [273…

```
# Looking at parameter names
pl.get_params().keys()
```

Out[273…

```
dict_keys(['memory', 'steps', 'verbose', 'ct', 'dtc', 'ct__n_jobs', 'ct__remainder', 'ct__
sparse_threshold', 'ct__transformer_weights', 'ct__transformers', 'ct__verbose', 'ct__verb
ose_feature_names_out', 'ct__as-is', 'ct__categ_cols', 'ct__categ_cols__memory', 'ct__cate
g_cols__steps', 'ct__categ_cols__verbose', 'ct__categ_cols__ohe', 'ct__categ_cols__ohe__ca
tegories', 'ct__categ_cols__ohe__drop', 'ct__categ_cols__ohe__dtype', 'ct__categ_cols__ohe
__handle_unknown', 'ct__categ_cols__ohe__sparse', 'dtc__ccp_alpha', 'dtc__class_weight',
'dtc__criterion', 'dtc__max_depth', 'dtc__max_features', 'dtc__max_leaf_nodes', 'dtc__min_
impurity_decrease', 'dtc__min_samples_leaf', 'dtc__min_samples_split', 'dtc__min_weight_fr
action_leaf', 'dtc__random_state', 'dtc__splitter'])
```

In [280…

```
# trying diff combos of hyperparameters!
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# dictionary of hyperparameter values to try
hyperparameters = {
    'dtc__max_depth': [2, 4, 6, 8,10, None],
    'dtc__min_samples_split': [5, 7, 10],
}
searcher = GridSearchCV(pl, hyperparameters, cv=5)

# fitting instance of an estimator
searcher.fit(X_train, y_train)
```

Out[280…

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('ct',
                                        ColumnTransformer(transformers=[('as-is',
                                                                         'passthrough',
                                                                         ['shield_no',
                                                                          'month_receive
d',
                                                                          'year_received',
                                                                          'month_closed',
                                                                          'year_closed',
                                                                          'mos_age_inciden
t',
                                                                          'complainant_age
_incident']),
                                                                                                          ('categ_cols',
                                                                                                           Pipeline(steps=
[('ohe',

OneHotEncoder(handle_unknown='ignore',

sparse=False))]),
                                                                                                            ['rank_abbrev_inc
ident',
                                                                                                             'rank_abbrev_no
w',
                                                                                                             'rank_now',
                                                                                                             'rank_incident',
                                                                                                             'mos_ethnicity',
                                                                                                             'mos_gender',
                                                                                                             'complainant_eth
nicity',
                                                                                                             'complainant_gen
der',
                                                                                                             'fado_type',
                                                                                                             'allegation',
                                                                                                             'contact_reaso
n'])])),
                                                        ('dtc', DecisionTreeClassifier())]),
              param_grid={'dtc__max_depth': [2, 4, 6, 8, 10, None],
                          'dtc__min_samples_split': [5, 7, 10]})
```

In [281…
```python
# the best parameters
searcher.best_params_
```

Out[281…
```
{'dtc__max_depth': 10, 'dtc__min_samples_split': 7}
```

In [305…
```python
# the accuracy using best estimator on testing set
searcher.best_estimator_.score(X_test, y_test)
```

Out[305…
```
0.7965612104539203
```

Fairness Analysis

Null Hypothesis: Our model is fair. Its precision for male complainants and female complainants are roughly the same, and any differences are due to random chance.

Alternative Hypothesis: Our model is unfair. Its precision for female complainants is lower than its precision for male complainants.

In [288…
```python
# imports
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
```

In [302…
```python
# getting only the male and female complainant data
df = allegations[(allegations['complainant_gender'] == 'Female') |
        (allegations['complainant_gender'] == 'Male')]

# getting relevant columns/features
categ = ['rank_abbrev_incident', 'rank_abbrev_now', 'rank_now', 'rank_incident',
        'mos_ethnicity', 'mos_gender', 'complainant_ethnicity', 'complainant_gender',
        'fado_type', 'allegation', 'contact_reason']
quant_cols = ['shield_no', 'month_received', 'year_received', 'month_closed',
          'year_closed', 'mos_age_incident', 'complainant_age_incident']
```

```
X = df[categ+quant_cols]
y = df['board_disposition']


# getting the observed stat
X_train, X_test, y_train, y_test = train_test_split(X, y)

results = X_test
results['prediction'] = searcher.best_estimator_.predict(X_test)
results['tag'] = y_test

results['is_male'] = (results['complainant_gender'] == 'Male').replace({True: 'male', Fals

# recall of male complainants
males = results[results['is_male'] == 'male']
recall_males = metrics.recall_score(males['tag'], males['prediction'])

# recall of female complainants
females = results[results['is_male'] == 'female']
recall_fem = metrics.recall_score(females['tag'], females['prediction'])

obs = recall_males - recall_fem
obs
```

Out[302…

```
-0.003193275157293174
```

In [301…

```
# conducting permutation test
diff_in_acc = []
for _ in range(100):
    s = (
        results[['is_male', 'prediction', 'tag']]
        .assign(is_male=results.is_male.sample(frac=1.0, replace=False).reset_index(drop=T
        .groupby('is_male')
        .apply(lambda x: metrics.accuracy_score(x['tag'], x['prediction']))
        .diff()
        .iloc[-1]
    )

    diff_in_acc.append(s)
```

In [303…

```
# calculate p-value
(diff_in_acc >= obs).mean()
```

Out[303…

```
0.52
```

Permutation Test Conclusion

Because the p-value of 0.52 is greater than our chosen significance level of 0.05, we fail to reject our null hypothesis. This means that we cannot say our model is not fair. We cannot say that the precision is not roughly the same for male and female complainants, and differences could be due to random chance.

The End