Summary of Findings

Introduction

The prediction question that I'm attempting regarding the dataset is predict the officer ethnicity given information about the allegations. This would be a classification problem because my target column would be categorical. Before diving in to prediction, I will drop rows in the dataset that have missing values. Upon looking at the dataset, I discovered that I have to classify my prediction into Hispanic, White, Black, Asian, American Indian as these are the unique variables in my target column mos_ethnicity. As of evaluating the model, since I care about how many times my predictions are correct, i.e. the proportion of the predictions that are correct, I will be the using accuracy of the model.

Baseline Model

For my baseline model, I would like to use board_disposition, contact_reason, mos_gender, complainant_ethnicity and mos_age_incident as my features. Among those 5 features, every feature is nominal except for mos_age_incident, which is a quantitative feature. For nominal features, I will be using OneHotEncoder to transform them into sparse matrices. For the quantitative feature, I will standardize it using StandardScaler. My baseline model has an accuracy of 0.57 provided by the R^2 score. Upon looking into the importance of each individual feature in my baseline model, I found that complainant_ethnicity appears to be the most important with a score of 0.01, and contact_reason appears to be the least important with a score of 0. However, this does not mean that contact_reason does not matter because I trained the model using all columns, in which some of them are raw(there was no feature engineering done on those columns). As I start to add more features in my final model, these importances will change. As of the performance of my baseline model, I think it is decent given a score of 0.57, which means that my predictions are correct 57% of the time. Ideally, I want to build a model that has a score as close as possible to 1, which means that 100% of my predictions are correct. However, given the score of my baseline model, I think it is not bad and I can definitely improve this score by adding new features in my final model.

Final Model

ADD precinct

The first thing that I will be doing in the final model is to add the precinct feature. This is good for my predictions because people generally live closer to people that are of the same ethnicity. To my understanding, each precinct governs one particular area in the city. Therefore precinct is a good feature because people generally want to work at places not too far from home, which means that officers who work in the same precinct are likely to be of the same ethnicity. I will be using OneHotEncoder for this column. After adding this feature, the score improved to 0.73.

ADD rank_abbrev_now - rank_abbrev_incident

I think that how much promotions or how fast the promotions are for an officer is a good feature for predicting the ethnicity of officers. To do this, I would first transform the columns into numbers representing the rank hierarchy and then perform the substraction. After that, I will be using FunctionTransformer for the output. After adding this feature, the score improved to 0.77.

ADD Binary outcome_description

It is possible that the outcome of the incident is related to the ethnicity of the officer. It is likely that officers of one ethnicity are more likely to make arrests than those of another ethnicity. I will be transforming the outcome_description column to a binary column, with 1 meaning arrest and 0 meaning no arrest. I will use FunctionTransformer. After adding this feature, the score improved to 0.78.

ADD first_name and last_name

Last names usually carry tons of information about people's origin. For example, people from Africa would have different last names than people from Asia. I will be using OneHotEncoder for these columns. After adding this feature, the score improved to 0.96.

Search

I performed a search using GridSearchCV. The model I used is a Pipeline with DecisionTreeClassifier. The best parameters are :{max_depth: 2500, min_samples_leaf: 2, min_samples_split: 5}.

Fairness Evaluation

My interesting subset of data is the mos_age_incident column. I will treat any age > 40 as old and <= 40 as young. Thus I have a young subset and an old subset, which will be my X and Y. For my parity measures, I will be using recall since false negatives are important in my model. In other words, I care more about false negatives than false positives. I want to know more the proportion of the two groups that are predicted correctly. My null hypothesis is that the sensitivities of the two groups are the same(my model is fair), and my alternative hypothesis is that the sensitivities of the two groups are different(my model is unfair). I will be using difference in means as my test statistic. I got a p-value of 0.944 which means that I cannot reject my null hypothesis which means that my model is fair.

Code

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import GridSearchCV

%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures
```

Introduction

In [2]:
```
# checking out the dataset
df = pd.read_csv('allegations_202007271729-Copy1.csv')
df.head()
```

Out[2]:

	unique_mos_id	first_name	last_name	command_now	shield_no	complaint_id	month_received	year_received	month_closed	year_closed	...	mos_age_incid
0	10004	Jonathan	Ruiz	078 PCT	8409	42835	7	2019	5	2020	...	
1	10007	John	Sears	078 PCT	5952	24601	11	2011	8	2012	...	
2	10007	John	Sears	078 PCT	5952	24601	11	2011	8	2012	...	
3	10007	John	Sears	078 PCT	5952	26146	7	2012	9	2013	...	
4	10009	Noemi	Sierra	078 PCT	24058	40253	8	2018	2	2019	...	

5 rows × 27 columns

In [3]:
```
# checking out the features(columns)
df.columns
```

Out[3]:
```
Index(['unique_mos_id', 'first_name', 'last_name', 'command_now', 'shield_no',
       'complaint_id', 'month_received', 'year_received', 'month_closed',
       'year_closed', 'command_at_incident', 'rank_abbrev_incident',
       'rank_abbrev_now', 'rank_now', 'rank_incident', 'mos_ethnicity',
       'mos_gender', 'mos_age_incident', 'complainant_ethnicity',
       'complainant_gender', 'complainant_age_incident', 'fado_type',
       'allegation', 'precinct', 'contact_reason', 'outcome_description',
       'board_disposition'],
      dtype='object')
```

In [4]:
```
# drop rows with missing values
df = df.dropna(axis = 0)
```

In [5]:
```
# checking out the target
df['mos_ethnicity'].unique()
```

Out[5]:
```
array(['Hispanic', 'White', 'Black', 'Asian', 'American Indian'],
      dtype=object)
```

Baseline Model

In [6]:
```
# Numeric columns and associated transformers
num_feat = ['mos_age_incident']
num_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# Categorical columns and associated transformers
cat_feat = ['board_disposition', 'contact_reason', 'mos_gender', 'complainant_ethnicity']
cat_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown = 'ignore'))
])

# preprocessing pipeline (put them together)
preproc = ColumnTransformer(
    transformers=[
        ('num', num_transformer, num_feat),
        ('cat', cat_transformer, cat_feat)
    ], remainder = 'drop')

pl = Pipeline(steps=[('preprocessor', preproc), ('clf', DecisionTreeClassifier())])
```

In [7]:
```
# split the dataset
X = df.drop('mos_ethnicity', axis = 1)
y = df.mos_ethnicity
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test
# train the model
pl.fit(X_train, y_train)
# score for the model
pl.score(X_test, y_test)
```

Out[7]:
```
0.5663969538315088
```

In [8]:
```
# look into each individual feature and their importances
importance = dict(zip(X.columns, pl.named_steps['clf'].feature_importances_))
for i in list(df.columns.drop(['mos_ethnicity', 'mos_age_incident', 'board_disposition', 'contact_reason', 'mos_gender
', 'complainant_ethnicity'])):
    importance.pop(i)
importance
```

Out[8]:
```
{'mos_gender': 0.0024161749025932537,
 'mos_age_incident': 0.01374031518291352,
 'complainant_ethnicity': 0.005253176526791074,
 'contact_reason': 0.01103227666092285,
 'board_disposition': 0.0069556292156181965}
```

Final Model

ADD precinct

After adding the precinct feature, the score of my model improved to 0.73, which means that precinct is indeed a good feature for my model.

In [9]:
```
# Numeric columns and associated transformers
num_feat = ['mos_age_incident']
num_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# Categorical columns and associated transformers
cat_feat = ['board_disposition', 'contact_reason', 'mos_gender', 'complainant_ethnicity', 'precinct']
cat_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown = 'ignore'))
])

# preprocessing pipeline (put them together)
preproc = ColumnTransformer(
    transformers=[
        ('num', num_transformer, num_feat),
        ('cat', cat_transformer, cat_feat)
    ], remainder = 'drop')

pl = Pipeline(steps=[('preprocessor', preproc), ('clf', DecisionTreeClassifier())])
```

In [10]:
```
# split the dataset
X = df.drop('mos_ethnicity', axis = 1)
y = df.mos_ethnicity
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test
# train the model
pl.fit(X_train, y_train)
# score for the model
pl.score(X_test, y_test)
```

Out[10]:
```
0.7315564017134698
```

ADD rank_abbrev_now - rank_abbrev_incident

It turns out that the ranks have something to do with the officer's ethnicity. After adding this feature, the score improved to 0.77

In [11]:
```
def helper(row):
    if row in ['POM', 'POF', 'PO', 'PSA', 'SRG']:
        return 0
    elif row in ['SGT', 'SSA', 'SDS', 'SCS']:
        return 1
    elif row in ['DT3', 'DT2', 'DT1', 'DTS', 'DCS', 'DI']:
        return 2
    elif row in ['LT', 'LSA', 'LCD', ]:
        return 3
    elif row in ['INS', 'CPT']:
        return 4
    elif row in ['DC', 'AC']:
        return 5
    else:
        return 6
def helper2(cols):
    return (cols.iloc[:,0].apply(helper) - cols.iloc[:,1].apply(helper)).to_frame()
```

In [12]:
```
# Numeric columns and associated transformers
num_feat = ['mos_age_incident']
num_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# rank columns and associated transformers
rank_transformer = FunctionTransformer(helper2)
rank_feat = ['rank_abbrev_now', 'rank_abbrev_incident']

# Categorical columns and associated transformers
cat_feat = ['board_disposition', 'contact_reason', 'mos_gender', 'complainant_ethnicity', 'precinct']
cat_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown = 'ignore'))
])

# preprocessing pipeline (put them together)
preproc = ColumnTransformer(
    transformers=[
        ('num', num_transformer, num_feat),
        ('cat', cat_transformer, cat_feat),
        ('rank', rank_transformer, rank_feat)
    ], remainder = 'drop')

pl = Pipeline(steps=[('preprocessor', preproc), ('clf', DecisionTreeClassifier())])
```

In [13]:
```
# split the dataset
X = df.drop('mos_ethnicity', axis = 1)
y = df.mos_ethnicity
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test
# train the model
pl.fit(X_train, y_train)
# score for the model
pl.score(X_test, y_test)
```

Out[13]:
```
0.7749881009043312
```

ADD Binary outcome_description

After adding this feature, the score improved to 0.78, which means that this feature is a good feature.

In [14]:
```
def helper3(row):
    if 'Arrest' in row:
        return 1
    else:
        return 0
def helper4(cols):
    return (cols.iloc[:,0].apply(helper3)).to_frame()
```

In [15]:
```
# Numeric columns and associated transformers
num_feat = ['mos_age_incident']
num_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# rank columns and associated transformers
rank_transformer = FunctionTransformer(helper2)
rank_feat = ['rank_abbrev_now', 'rank_abbrev_incident']

# outcome column and associated transformers
out_transformer = FunctionTransformer(helper4)
out_feat = ['outcome_description']

# Categorical columns and associated transformers
cat_feat = ['board_disposition', 'contact_reason', 'mos_gender', 'complainant_ethnicity', 'precinct']
cat_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown = 'ignore'))
])

# preprocessing pipeline (put them together)
preproc = ColumnTransformer(
    transformers=[
        ('num', num_transformer, num_feat),
        ('cat', cat_transformer, cat_feat),
        ('rank', rank_transformer, rank_feat),
        ('out', out_transformer, out_feat)
    ], remainder = 'drop')

pl = Pipeline(steps=[('preprocessor', preproc), ('clf', DecisionTreeClassifier())])
```

In [16]:
```
# split the dataset
X = df.drop('mos_ethnicity', axis = 1)
y = df.mos_ethnicity
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test
# train the model
pl.fit(X_train, y_train)
# score for the model
pl.score(X_test, y_test)
```

Out[16]:
```
0.780580675868634
```

ADD last_name

After adding this feature, the score improved to 0.96, which means that this feature is a good feature.

In [17]:
```
# Numeric columns and associated transformers
num_feat = ['mos_age_incident']
num_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

# rank columns and associated transformers
rank_transformer = FunctionTransformer(helper2)
rank_feat = ['rank_abbrev_now', 'rank_abbrev_incident']

# outcome column and associated transformers
out_transformer = FunctionTransformer(helper4)
out_feat = ['outcome_description']

# Categorical columns and associated transformers
cat_feat = ['board_disposition', 'contact_reason', 'mos_gender', 'complainant_ethnicity', 'precinct', 'last_name']
cat_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown = 'ignore'))
])

# preprocessing pipeline (put them together)
preproc = ColumnTransformer(
    transformers=[
        ('num', num_transformer, num_feat),
        ('cat', cat_transformer, cat_feat),
        ('rank', rank_transformer, rank_feat),
        ('out', out_transformer, out_feat)
    ], remainder = 'drop')

pl = Pipeline(steps=[('preprocessor', preproc), ('clf', DecisionTreeClassifier())])
```

In [18]:
```
# split the dataset
X = df.drop('mos_ethnicity', axis = 1)
y = df.mos_ethnicity
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) # 70% training and 30% test
# train the model
pl.fit(X_train, y_train)
# score for the model
pl.score(X_test, y_test)
```

Out[18]:
```
0.959662065683008
```

Search

The best parameters are :{max_depth: 2500, min_samples_leaf: 2, min_samples_split: 5}

In [19]:
```
parameters = {
    'clf__max_depth': [None, 1000, 1500, 2000, 2500, 3000, 3500],
    'clf__min_samples_split':[2,5,7,15,20],
    'clf__min_samples_leaf':[2,5,7,15,20]
}
```

In [20]:
```
search = GridSearchCV(pl, parameters, cv = 5)
search.fit(X_train, y_train)
```

Out[20]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
                                        ColumnTransformer(transformers=[('num',
                                                                         Pipeline(steps=[('scaler',
                                                                                          StandardScaler())]),
                                                                         ['mos_age_incident']),
                                                                        ('cat',
                                                                         Pipeline(steps=[('onehot',
                                                                                          OneHotEncoder(handle_unknow
n='ignore'))]),
                                                                         ['board_disposition',
                                                                          'contact_reason',
                                                                          'mos_gender',
                                                                          'complainant_ethnicity',
                                                                          'precinct',
                                                                          'last_name']),
                                                                        ('rank',
                                                                         FunctionTransformer(func=<function helper2 a
t 0x7fc0b8a98430>),
                                                                         ['rank_abbrev_now',
                                                                          'rank_abbrev_incident']),
                                                                        ('out',
                                                                         FunctionTransformer(func=<function helper4 a
t 0x7fc0b8be2670>),
                                                                         ['outcome_description'])])),
                                       ('clf', DecisionTreeClassifier())]),
             param_grid={'clf__max_depth': [None, 1000, 1500, 2000, 2500, 3000,
                                            3500],
                         'clf__min_samples_leaf': [2, 5, 7, 15, 20],
                         'clf__min_samples_split': [2, 5, 7, 15, 20]})
```

In [28]:
```
best = search.best_params_
best
```

Out[28]:
```
{'clf__max_depth': 2500,
 'clf__min_samples_leaf': 2,
 'clf__min_samples_split': 5}
```

Fairness Evaluation

I will first divide the dataset into the two groups that I mentioned.

In [22]:
```
pre = pl.predict(X_test)
fair = pd.DataFrame()
fair['age'] = X_test['mos_age_incident'].apply(lambda x :'old' if x > 40 else 'young')
fair['predict'] = pre
fair['actual'] = y_test
fair['compare'] = fair['predict'] == fair['actual']
fair.groupby('age')['compare'].mean()
```

Out[22]:
```
age
old      0.958763
young    0.959766
Name: compare, dtype: float64
```

I can see that there is a small difference between the two groups. Now I need to know if this difference is significant. I will use a significance level of 0.05.

In [23]:
```
obs = fair.groupby('age')['compare'].mean().diff().abs().iloc[-1]
obs
```

Out[23]:
```
0.0010034126983040625
```

In [24]:
```
n_repetitions = 500

differences = []
for _ in range(n_repetitions):

    shuffled_age = (
        fair['age']
        .sample(replace=False, frac=1)
        .reset_index(drop=True)
    )

    shuffled = (
        fair
        .assign(**{'Shuffled age': shuffled_age})
    )

    group_means = (
        shuffled
        .groupby('Shuffled age')
        .mean()
        .loc[:, 'compare']
    )
    difference = group_means.diff().abs().iloc[-1]

    differences.append(difference)
np.count_nonzero(differences >= obs) / n_repetitions
```

Out[24]:
```
0.944
```

Since I got a p value larger than 0.05, I cannot reject my null hypothesis which means that my model is fair.