nypd

June 7, 2021

1 NYPD Allegations

- **See the main project notebook for instructions to be sure you satisfy the rubric!**
- See Project 03 for information on the dataset.
- A few example prediction questions to pursue are listed below. However, don't limit yourself to them!
  - Predict the outcome of an allegation (might need to feature engineer your output column).
  - Predict the complainant or officer ethnicity.
  - Predict the amount of time between the month received vs month closed (difference of the two columns).
  - Predict the rank of the officer.

Be careful to justify what information you would know at the "time of prediction" and train your model using only those features.

2 Summary of Findings

2.0.1 Introduction

In the nypd dataset, we will use DecisionTreeClassifier to predict the board disposition of a complaint case given certain variables. We will target 'board disposition', and assess our model with accuracy, or its R-squared value.

2.0.2 Baseline Model

Our baseline model will have 10 features. We have no quantitative variables. Our ordinal vairables are: **'rank_incident'**. Our nominal variables are: **'mos_ethnicity', 'mos_gender', 'complainant_ethnicity', 'complainant_gender', 'fado_type', 'allegation', 'precinct', 'contact_reason', 'outcome_description'.**

The R-squared value of our model's prediction of data is about **0.750**. We believe our model performance is decent because our R-squared value shows that about 76% of the values in board disposition can be explained by our model.

2.0.3 Final Model

After evaluating the accuracy of our baseline model, we decided to add **'mos_age_incident'** and **'complainant_age_incident'**, and engineer them using StdScalerByGroup (with ranks as groups) as new features. We added them because we believe that the ages of both officer and

complainant provide significant influence on the board's disposition. We believe there could be a possible trend hidden in these two variables. For example, the board might be more lenient towards younger officers than older officers. The opposite could be true, where older officers might have more leeway due to their veteran status.

After running GridSearchCV on our DecisionTreeClassifier, our ideal parameters are: **max_depth = 5** and **min_samples_leaf=5.** We chose DecisionTreeClassifier in the end because we wanted to use a model to predict the decision of the board based on the values of our model's variables.

The R-squared value of our final model's prediction of the data is about **0.759**.

2.0.4 Fairness Evaluation

The "interesting subset" we will look at is **'complainant_age_incident'**. We are testing to see if our model is more fair towards younger or older people, with the cutoff of "young" people being 45 years old. Our parity measure is accuracy, to test if our model is less accurate when predicting for younger or older groups.

```
Null Hypothesis: The model is fair, the accuracy of the two subsets are the same
Alternative Hypothesis: The model is unfair, the accuracy of younger complainants is differ
Significance Level: 0.01
```

After running a permutation test of 100 trials on the accuracy parity of our model on complainant age groups (old (>45) vs young (<=45)), our resulting p-value is **0.71**. Therefore, there is not enough statistically significant evidence to believe that our model is unfair. Thus, we **Fail to Reject** our null hypothesis.

3 Code

[183]:
```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelBinarizer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.impute import SimpleImputer

from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import train_test_split
from sklearn.base import BaseEstimator, TransformerMixin
```

[2]:
```
fp = os.path.join('data','allegations_202007271729.csv')
ny = pd.read_csv(fp)
```

[3]:
```
# STDScalerByGroup class function
class StdScalerByGroup(BaseEstimator, TransformerMixin):

    def __init__(self):
        pass

    def fit(self, X, y=None):
        df = pd.DataFrame(X)
        stats = df.groupby(df.iloc[:,0]).agg(['mean','std'])
        self.grps_ = dict(stats)

        return self

    def transform(self, X, y=None):
        try:
            getattr(self, "grps_")
        except AttributeError:
            raise RuntimeError("You must fit the transformer before tranforming␣
↪the data!")

        def z_scaler(df, group_col, data_col):
            set = df[[group_col, data_col]]
            stats = pd.concat([self.grps_[(data_col, 'mean')], self.
↪grps_[(data_col, 'std')]],axis=1)
            stats.columns = stats.columns.droplevel()
            stats = dict(stats)
            set['mean'] = set[group_col].apply(lambda x: stats['mean'][x])
            set['std'] = set[group_col].apply(lambda x: stats['std'][x])
            set['zscore'] = (set[data_col] - set['mean']) / set['std']
            return set['zscore']

        df = pd.DataFrame(X)
        for each in df.columns[1:]:
            df[each] = z_scaler(df, df.columns[0], each)

        return df.drop(columns=df.columns[0])
```

[174]:
```
pd.set_option('display.max_columns', None)
nyp = ny[ny.columns[14:27]]
```

3.0.1 Baseline Model

[142]:
```
rankings = {'Police Officer': 0, 'Detective': 1, 'Sergeant': 2,
                'Lieutenant': 3, 'Captain': 4, 'Inspector': 5,
                    'Deputy Inspector': 6, 'Chiefs and other ranks': 7}
nypd = nyp.copy()
nypd['rank_incident'] = nypd['rank_incident'].transform(lambda x: rankings[x])
nypd['board_disposition'] = nypd['board_disposition'].transform(lambda x: 1 if␣
 ↪x.startswith('Substantiated') else 0)
# STDScalerByGroup: mos_age_incident (grouped by rank_incident),␣
 ↪complainaint_age_incident (grouped by complainant gender)
# OneHotEncoding: mos_ethnicity, complainant_ethnicity, fado_type, rank_incident
#                     allegation, precinct, contact_reason, outcome_description
nulls = nypd[nypd.columns[nypd.isna().sum() > 0]]
for each in nulls.columns:
    if nypd[each].dtypes == 'float64':
        nypd[each] = nypd[each].fillna(0)
    else:
        nypd[each] = nypd[each].fillna('None')

nypd['complainant_age_incident'] = nypd['complainant_age_incident'].
 ↪transform(lambda x: 0 if x < 0 else x)
```

[256]:
```
def analyze(nypd):
    X = nypd.drop('board_disposition', axis=1)
    y = nypd.board_disposition
    Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,test_size=0.3)
    preproc = ColumnTransformer(
        transformers=[
            ('one_hot', OneHotEncoder(handle_unknown='ignore'),␣
 ↪['mos_ethnicity','complainant_ethnicity','fado_type', 'rank_incident',
                                                                    ␣
 ↪'allegation','precinct','contact_reason','outcome_description',␣
 ↪'mos_gender', 'complainant_gender'])
        ], remainder='passthrough')
    pl = Pipeline(steps=[('preprocess', preproc), ('tree',␣
 ↪DecisionTreeClassifier(max_depth=5))])

    pl.fit(Xtrain,ytrain)
    train_pred = pl.predict(Xtrain)
    test_pred = pl.predict(Xtest)

    rmse_train = np.sqrt(np.mean((train_pred - ytrain)**2))
    rmse_test = np.sqrt(np.mean((test_pred - ytest)**2))
    return 'R-Squared: '+ str(pl.score(Xtest, ytest))
```

[257]:
```
analyze(nypd)
```

[257]: 'R-Squared: 0.7502997601918465'

3.0.2 Final Model

[272]:
```
def best_param(nypd):
    X = nypd.drop('board_disposition', axis=1)
    y = nypd.board_disposition
    Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,test_size=0.3)
    age_transformer = Pipeline(steps=[
        ('before_fill', SimpleImputer(strategy='constant',fill_value=0)),
        ('std', StdScalerByGroup()),
        ('after_fill', SimpleImputer(strategy='constant',fill_value=0)),
    ])
    preproc = ColumnTransformer(
        transformers=[
            ('StdScaleAndImpute', age_transformer,␣
↪['rank_incident','mos_age_incident', 'complainant_age_incident']),
            ('one_hot', OneHotEncoder(handle_unknown='ignore'),␣
↪['mos_ethnicity','complainant_ethnicity','fado_type',
                                                                  ␣
↪'rank_incident', 'allegation','precinct','contact_reason',
                                                                  ␣
↪'outcome_description', 'mos_gender', 'complainant_gender']),
        ], remainder='passthrough')
    pl = Pipeline(steps=[('preprocess', preproc),('print', print()), ('tree',␣
↪DecisionTreeClassifier())])
    pl.fit(Xtrain,ytrain)
    train_pred = pl.predict(Xtrain)
    test_pred = pl.predict(Xtest)
    parameters = {
        'tree__max_depth': [5,6,7,8,9,14,16,18,None],
        'tree__min_samples_leaf': [5,7,10,14,16,18,20],
    }
    clf = GridSearchCV(pl,parameters,cv=5)
    clf.fit(Xtrain,ytrain)
    print(clf.best_params_)
```

[273]:
```
best_param(nypd)
```

{'tree__max_depth': 5, 'tree__min_samples_leaf': 5}

[258]:
```
def final_analyze(nypd):
    X = nypd.drop('board_disposition', axis=1)
    y = nypd.board_disposition
    Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,test_size=0.3)
    age_transformer = Pipeline(steps=[
        ('before_fill', SimpleImputer(strategy='constant',fill_value=0)),
```

```
        ('std', StdScalerByGroup()),
        ('after_fill', SimpleImputer(strategy='constant',fill_value=0)),
    ])
    preproc = ColumnTransformer(
        transformers=[
            ('StdScaleAndImpute', age_transformer,␣
↪['rank_incident','mos_age_incident', 'complainant_age_incident']),
            ('one_hot', OneHotEncoder(handle_unknown='ignore'),␣
↪['mos_ethnicity','complainant_ethnicity','fado_type',
                                                                  ␣
↪'rank_incident', 'allegation','precinct','contact_reason',
                                                                  ␣
↪'outcome_description', 'mos_gender', 'complainant_gender']),
        ], remainder='passthrough')
    pl = Pipeline(steps=[('preprocess', preproc), ('tree',␣
↪DecisionTreeClassifier(max_depth=5,min_samples_leaf=5))])
    pl.fit(Xtrain,ytrain)
    train_pred = pl.predict(Xtrain)
    test_pred = pl.predict(Xtest)

    rmse_train = np.sqrt(np.mean((train_pred - ytrain)**2))
    rmse_test = np.sqrt(np.mean((test_pred - ytest)**2))
    return 'R-Squared: '+ str(pl.score(Xtest, ytest))
```

[266]:
```
final_analyze(nypd)
```

[266]:
```
'R-Squared: 0.758792965627498'
```

3.0.3 Fairness Evaluation

[268]:
```
X = nypd.drop('board_disposition', axis=1)
y = nypd.board_disposition
Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,test_size=0.3)
age_transformer = Pipeline(steps=[
    ('before_fill', SimpleImputer(strategy='constant',fill_value=0)),
    ('std', StdScalerByGroup()),
    ('after_fill', SimpleImputer(strategy='constant',fill_value=0)),
])
preproc = ColumnTransformer(
    transformers=[
        ('StdScaleAndImpute', age_transformer,␣
↪['rank_incident','mos_age_incident', 'complainant_age_incident']),
        ('one_hot', OneHotEncoder(handle_unknown='ignore'),␣
↪['mos_ethnicity','complainant_ethnicity','fado_type',
                                                        'rank_incident',␣
↪'allegation','precinct','contact_reason',
```

```
 ↪'outcome_description', 'mos_gender', 'complainant_gender']),
    ], remainder='passthrough')
pl = Pipeline(steps=[('preprocess', preproc), ('tree',␣
 ↪DecisionTreeClassifier(max_depth=5,min_samples_leaf=5))])
pl.fit(Xtrain,ytrain)
train_pred = pl.predict(Xtrain)
test_pred = pl.predict(Xtest)
```

[255]:
```
results = Xtest
results['prediction'] = test_pred
results['tag'] = ytest
results['young'] = (results.complainant_age_incident <= 45).replace({True:␣
 ↪'young',False: 'old'})
observed = Xtest.groupby('young').apply(lambda x: metrics.accuracy_score(x.tag,␣
 ↪x.prediction)).diff().iloc[-1]
test = []
for i in range(100):
    accuracy = results.assign(young=results.young.sample(frac=1,replace=False).
 ↪reset_index(drop=True))
    test.append(accuracy.groupby('young').apply(lambda x: metrics.
 ↪accuracy_score(x.tag, x.prediction)).diff().iloc[-1])

pd.Series(test <= observed).mean()
```

```
<ipython-input-255-ddbf089d05e3>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  results['prediction'] = test_pred
<ipython-input-255-ddbf089d05e3>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  results['tag'] = ytest
<ipython-input-255-ddbf089d05e3>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  results['young'] = (results.complainant_age_incident <= 45).replace({True:
'young',False: 'old'})
```

```
[255]: 0.71
```