ml_project

April 7, 2020

```
In [1]: import numpy as np
        import pandas as pd
        import scipy.linalg as la
        import scipy.stats as stats
        import matplotlib.pyplot as plt
        import umap
        import pickle
        import time
        import xgboost

        from sklearn import linear_model, model_selection, metrics
        from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
        from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as GDA
        from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
        from sklearn.manifold import TSNE
        from sklearn.model_selection import GridSearchCV,train_test_split,cross_val_score
        from sklearn.metrics import roc_auc_score as RAS
        from sklearn.metrics import roc_curve as ROC
        from sklearn.model_selection import train_test_split
        from sklearn.neighbors import KNeighborsClassifier as KNC
```

```
   jupyter nbconvert --to latex "Race and Incarceration in America.ipynb"
--TagRemovePreprocessor.remove_input_tags='{"hide_input"}'; pdflatex "Race and
Incarceration in America"
```

1 Introduction

Last semester I took a look a data set containing the information of more than 7.5 million individuals that have been processed by the criminal justice system. I found that racial minorities were more likely to receive extreme sentences, agreeing with existing research around bias in the criminal justice system. In this project I will be exploring the data from a machine learning perspective. My goal is to determine if this data can be classified in such a way that is predictive of race. The idea is that perhaps racial bias can be detected in various systems by seeing how effective different machine learning techniques are at classifying inmate data by race.

2 Data

2.1 Source and Credibility

The data that I will be using in this analysis is from one source. It is a database hosted on Data.gov and maintained by the State of Connecticut Department of Corrections. This source is highly credible because it is a primary source for the data. This organization is an official government agency which collects, maintains, and reports on this data.

2.2 Gathering and Cleaning

All the data which I am using in this report are freely available to the public. Collection and cleaning was relatively simple as the source data was well maintained. The file that I obtained from the Connecticut Department of Corrections is a very well maintained database. The largest issue I had with this file was mild inconsistency with the way in which certain data was encoded (ex. race was encoded as both `WHITE` and `WHITE\t`). This was the data that I spent the most time working to engineer as it is a data set that I intend to use for different regression-related analyses. The files are

```
individuals.csv
regression_df.csv.
```

2.3 About the Data

This data set contains individual information for 7.77 million people that have been processed by the justice system and recorded by the Connecticut Department of Corrections. Each individual is recorded along with their age, gender, race, offense, and sentence length, among other things [6].

I also created one-hot encoded versions of this data set in order to run regressions on the data. Because of the size of the data, the regression data sets are only random subsets of the larger data set.

Because there is so much to consider in what is found in the data set, I chose not to engineer more features as to avoid unneeded complexity.

i ought to disclose the sample sizes among different races that are found in the Connecticut Department of Justice data. The sample size for American Indians and Asians is much smaller than that of Whites, Hispanics, and Blacks, hence we may see some irregular outcomes in the analysis related to these racial groups.

```
In [2]: cols = ['RACE','GENDER','AGE','OFFENSE','FACILITY','DETAINER','SENTENCE DAYS']
        features = ['GENDER','AGE','OFFENSE','FACILITY','DETAINER','SENTENCE DAYS']
        df = pd.read_csv('individuals.csv',usecols=cols)

In [3]: races = {'BLACK':'Blacks','WHITE':'Whites',
                 'HISPANIC':'Hispanic','AMER IND':'American Indian',
                 'ASIAN':'Asian'
                }
        for rac in races.keys():
            mask = df.RACE == rac
            print(f'Sample size for {races[rac]}: {len(df[mask])}')
```

```
Sample size for Blacks: 3287596
Sample size for Whites: 2393949
Sample size for Hispanic: 2039297
Sample size for American Indian: 21133
Sample size for Asian: 35660
```

3 Possible questions

can we use ml techniques to correctly classify this data? which ones fail and why? can we create a predictive model for sentence lengths? should sentencing be offloaded to a ml algorithm? what does it mean to have an effective classfifier for this data set.

```
In [6]: # chunksize = 100000
        # rdf = pd.read_csv('regression_df.csv', chunksize=chunksize)
```

```
In [2]: cols = ['RACE','GENDER','AGE','OFFENSE','FACILITY','DETAINER','SENTENCE DAYS']
        features = ['GENDER','AGE','OFFENSE','FACILITY','DETAINER','SENTENCE DAYS']
        df = pd.read_csv('individuals.csv',usecols=cols)
```

4 ML Techniques

TSNE is having poor results and UMAP doesn't seem to capture any good groupings. I could block by crime and examine a few "big" crimes the try several different perplexities, however I do not think that clusting will be helpful in this case. Especially becasue I have not yet found a kernel that converts crimes to a spacial variable.

there are other techniques that are not applicable to this data. For example, since this data set is not a time series models like ARMA and HMM are not applicable here.

4.1 Random Forest Classifiers

```
In [44]: samp = df.sample(20000)
         samp.RACE = pd.factorize(samp['RACE'])[0] + 1
         samp.GENDER = pd.factorize(samp['GENDER'])[0] + 1
         samp.OFFENSE = pd.factorize(samp['OFFENSE'])[0] + 1
         samp.DETAINER = pd.factorize(samp['DETAINER'])[0] + 1
         samp.FACILITY = pd.factorize(samp['FACILITY'])[0] + 1
         samp_y = samp.RACE
         samp_X = samp[['GENDER','AGE','OFFENSE','FACILITY','DETAINER','SENTENCE DAYS']]
```

```
In [46]: param_grid = {
             'n_estimators': np.arange(100,400,20),
             'max_depth': np.arange(10,100,10),
             'max_features': np.arange(1,6)
         }
```

```
        clf = RandomForestClassifier(oob_score=True)

        s = time.time()
        clf = GridSearchCV(clf, param_grid, scoring=None, cv=5)
        clf = clf.fit(samp_X, samp_y)
        e = time.time()
        print(f'time to train is {(e-s)/60} minutes')

        clf = clf.best_estimator_
        clf = clf.fit(samp_X, samp_y)
        print(f'oob score is {clf.oob_score_}')

        with open('RandomForestClf.pickle', "wb+") as f:
            pickle.dump(clf, f)
```

```
time to train is 223.73655876318614 minutes
oob score is 0.70645
```

```
In [4]: with open('RandomForestClf.pickle',"rb") as f:
            clf = pickle.load(f)
        print(clf.oob_score_)
        print(clf.feature_importances_)
```

```
/home/ethan/.local/lib/python3.7/site-packages/sklearn/base.py:306: UserWarning: Trying to unpi
  UserWarning)


0.70645
[0.01022266 0.26942225 0.2124008  0.17300606 0.0338177  0.30113052]


/home/ethan/.local/lib/python3.7/site-packages/sklearn/base.py:306: UserWarning: Trying to unpi
  UserWarning)
```

here we can see that the most important features are sentence length, age, and offense. And the high OoB score is promising.

```
In [5]: samp = df.sample(200000)
        samp.RACE = pd.factorize(samp['RACE'])[0] + 1
        samp.GENDER = pd.factorize(samp['GENDER'])[0] + 1
        samp.OFFENSE = pd.factorize(samp['OFFENSE'])[0] + 1
        samp.DETAINER = pd.factorize(samp['DETAINER'])[0] + 1
        samp.FACILITY = pd.factorize(samp['FACILITY'])[0] + 1
        samp_y = samp.RACE
        samp_X = samp[['GENDER','AGE','OFFENSE','FACILITY','DETAINER','SENTENCE DAYS']]
        clf.score(samp_X, samp_y)
```

```
Out[5]: 0.420315
```

However the score is not great. Random chance would be .2, so it does do better than chance, though not much better.

4.2 Gradient Descent Boosted Classification

```
In [34]: samp = df.sample(10000)
         samp.RACE = pd.factorize(samp['RACE'])[0] + 1
         samp.GENDER = pd.factorize(samp['GENDER'])[0] + 1
         samp.OFFENSE = pd.factorize(samp['OFFENSE'])[0] + 1
         samp.DETAINER = pd.factorize(samp['DETAINER'])[0] + 1
         samp.FACILITY = pd.factorize(samp['FACILITY'])[0] + 1
         samp_y = samp.RACE
         samp_X = samp[['GENDER','AGE','OFFENSE','FACILITY','DETAINER','SENTENCE DAYS']]

In [35]:

In [ ]: param_grid = {
             'learning_rate': np.linspace(.01,1,10),
             'subsample': np.linspace(.05,1,10),
             'max_depth': np.arange(1,5)
         }

         clf = GradientBoostingClassifier(n_estimators=5000)

         s = time.time()
         # clf = GridSearchCV(clf, param_grid, cv=5)
         clf = clf.fit(samp_X, samp_y)
         e = time.time()

         # clf = clf.best_estimator_
         # clf = clf.fit(samp_X, samp_y)

         print(f'time is {(e-s)/60}')

         with open('GradientBoostedClf1.pickle', "wb+") as f:
             pickle.dump(clf, f)

In [6]: samp = df.sample(200000)
         samp.RACE = pd.factorize(samp['RACE'])[0] + 1
         samp.GENDER = pd.factorize(samp['GENDER'])[0] + 1
         samp.OFFENSE = pd.factorize(samp['OFFENSE'])[0] + 1
         samp.DETAINER = pd.factorize(samp['DETAINER'])[0] + 1
         samp.FACILITY = pd.factorize(samp['FACILITY'])[0] + 1
         samp_y = samp.RACE
         samp_X = samp[['GENDER','AGE','OFFENSE','FACILITY','DETAINER','SENTENCE DAYS']]
         clf.score(samp_X, samp_y)
```

```
Out[6]: 0.296365
```

```
In [8]: with open('GradientBoostedClf.pickle', "rb") as f:
            clf = pickle.load(f)
        #     print(f'oob improvement {clf.oob_improvement_}')
        #     to_print = zip(clf.feature_importances_,features)
            print(clf.feature_importances_)
```

```
[0.02090027 0.24716962 0.25775903 0.10433119 0.03748648 0.33235341]


/home/ethan/.local/lib/python3.7/site-packages/sklearn/base.py:306: UserWarning: Trying to unpi
  UserWarning)
/home/ethan/.local/lib/python3.7/site-packages/sklearn/base.py:306: UserWarning: Trying to unpi
  UserWarning)
```

Here we see the same feature importances as we did with the random forest

```
In [39]: clf.get_params()
```

```
Out[39]: {'criterion': 'friedman_mse',
          'init': None,
          'learning_rate': 0.34,
          'loss': 'deviance',
          'max_depth': 4,
          'max_features': None,
          'max_leaf_nodes': None,
          'min_impurity_decrease': 0.0,
          'min_impurity_split': None,
          'min_samples_leaf': 1,
          'min_samples_split': 2,
          'min_weight_fraction_leaf': 0.0,
          'n_estimators': 600,
          'n_iter_no_change': None,
          'presort': 'auto',
          'random_state': None,
          'subsample': 1.0,
          'tol': 0.0001,
          'validation_fraction': 0.1,
          'verbose': 0,
          'warm_start': False}
```

4.3 XG Boost

```
In [5]: samp = df.sample(10000)
        samp.RACE = pd.factorize(samp['RACE'])[0] + 1
        samp.GENDER = pd.factorize(samp['GENDER'])[0] + 1
        samp.OFFENSE = pd.factorize(samp['OFFENSE'])[0] + 1
```

```
        samp.DETAINER = pd.factorize(samp['DETAINER'])[0] + 1
        samp.FACILITY = pd.factorize(samp['FACILITY'])[0] + 1
        samp_y = samp.RACE
        samp_X = samp[['GENDER','AGE','OFFENSE','FACILITY','DETAINER','SENTENCE DAYS']]
```

```
In [7]: param_grid = {
            'learning_rate':np.linspace(.01,1,5),
            'reg_alpha':np.linspace(.01,1,5),
            'reg_lambda':np.linspace(.01,1,5),
            'gamma':np.linspace(.01,1,5)
        }

        clf = xgboost.XGBClassifier(verbose=2)
        s = time.time()
        clf = GridSearchCV(clf, param_grid,cv=5)
        clf = clf.fit(samp_X, samp_y)

        with open('XBG_fitted_grid.pickle', 'wb+') as f:
            pickle.dump(clf, f)

        clf = clf.best_estimator_
        clf = clf.fit(samp_X, samp_y)
        e = time.time()

        print(f'time was {(e-s)/(60*60)} hours')

        with open('XGBoostClf.pickle', 'wb+') as f:
            pickle.dump(clf, f)
```

```
time was 1.7242353409528732 hours
```

```
In [18]: samp = df.sample(200000)
         samp.RACE = pd.factorize(samp['RACE'])[0] + 1
         samp.GENDER = pd.factorize(samp['GENDER'])[0] + 1
         samp.OFFENSE = pd.factorize(samp['OFFENSE'])[0] + 1
         samp.DETAINER = pd.factorize(samp['DETAINER'])[0] + 1
         samp.FACILITY = pd.factorize(samp['FACILITY'])[0] + 1
         samp_y = samp.RACE
         samp_X = samp[['GENDER','AGE','OFFENSE','FACILITY','DETAINER','SENTENCE DAYS']]
         # clf.score(samp_X,samp_y)
```

```
In [19]: with open('XGBoostClf.pickle', "rb") as f:
             clf = pickle.load(f)
         #     print(f'oob improvement {clf.oob_improvement_}')
         #     to_print = zip(clf.feature_importances_,features)
             print(clf.score(samp_X,samp_y))
             print(clf.feature_importances_)
```

```
0.41374
[0.2596316  0.132617   0.1624777  0.12035192 0.18768501 0.13723671]
```

This score is very promiseing since the score is about double chance. There actually is a lot of correct classification going on here.

Interestingly however, it seems that the feature importances are very different for this model. Gender is the most important feature nad every other feature is about equally important.

4.4 K-Nearest Neighbors

In [ ]:

4.5 KD Trees

In [ ]: