San Francisco Crime Category Prediction _ combined

July 11, 2018

```
In [35]: from sklearn.metrics import log_loss, accuracy_score
         from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import MinMaxScaler
         from sklearn.pipeline import make_pipeline
         from sklearn.ensemble import RandomForestClassifier

         import matplotlib.pyplot as plt
         import pandas as pd
         import numpy as np
         from collections import Counter
         import operator
```

1 Introducao

Nesse notebook iremos analisar os dados relativos aos crimes ocorridos em San Francisco e criaremos modelos para prever a categoria dos crimes ocorridos. Os dados foram disponilizados no Kaggle **aqui** .

2 Analise de dados

Nessa etapa, carregamos o dataset de treino baixado do kaggle utilizando pandas e analisamos o formato dos dados, numero de instancias e o numero de classes. Criamos tambem visualizacoes para entender melhor o comportamento temporal dos dados e a distribuicao de classes.

```
In [2]: data_original = pd.read_csv("train_original.csv", sep=",")
        data_original.shape
```

```
Out[2]: (878049, 9)
```

Pelo tamanho do numero de instancias, fica dificil utilizar um classificador como SVM, pois o dataset eh muito grande e o treinamento demoraria muito. Uma boa escolha pode ser o modelo random forest.

```
In [3]: data_original.head()
```

```
Out[3]:                  Dates        Category                     Descript  \
        0  2015-05-13 23:53:00        WARRANTS                WARRANT ARREST
        1  2015-05-13 23:53:00  OTHER OFFENSES      TRAFFIC VIOLATION ARREST
        2  2015-05-13 23:33:00  OTHER OFFENSES      TRAFFIC VIOLATION ARREST
        3  2015-05-13 23:30:00   LARCENY/THEFT  GRAND THEFT FROM LOCKED AUTO
        4  2015-05-13 23:30:00   LARCENY/THEFT  GRAND THEFT FROM LOCKED AUTO

           DayOfWeek PdDistrict      Resolution                    Address  \
        0  Wednesday   NORTHERN  ARREST, BOOKED         OAK ST / LAGUNA ST
        1  Wednesday   NORTHERN  ARREST, BOOKED         OAK ST / LAGUNA ST
        2  Wednesday   NORTHERN  ARREST, BOOKED  VANNESS AV / GREENWICH ST
        3  Wednesday   NORTHERN            NONE   1500 Block of LOMBARD ST
        4  Wednesday        PARK            NONE  100 Block of BRODERICK ST

                   X          Y
        0 -122.425892  37.774599
        1 -122.425892  37.774599
        2 -122.424363  37.800414
        3 -122.426995  37.800873
        4 -122.438738  37.771541
```

Podemos quebrar a data em atributos como dia, mes e ano e utilizar como features no classificador.

```
In [4]: data = pd.read_csv("train.csv", sep=",")
```

```
In [5]: data.shape
```

```
Out[5]: (878049, 8)
```

2.0.1 Dataset modificado com a data transformada em atributos

```
In [6]: data.head()
```

```
Out[6]:    Year  Month  Day  Hour        Category  DayOfWeek           X          Y
        0  2015      5   13    23        WARRANTS  Wednesday -122.425892  37.774599
        1  2015      5   13    23  OTHER OFFENSES  Wednesday -122.425892  37.774599
        2  2015      5   13    23  OTHER OFFENSES  Wednesday -122.424363  37.800414
        3  2015      5   13    23   LARCENY/THEFT  Wednesday -122.426995  37.800873
        4  2015      5   13    23   LARCENY/THEFT  Wednesday -122.438738  37.771541
```

2.1 Analise do numero de classes

O objetivo para esses dados eh prever a categoria do crime. Analisando o numero de classes dos dados, podemos perceber outro desafio em relacao aos dados. Sao 39 o numero de classes, e bastante desbalanceadas, o que torna ambos undersampling e oversampling desafiadores.

```
In [7]: np.unique(data["Category"])
```

```
Out[7]: array(['ARSON', 'ASSAULT', 'BAD CHECKS', 'BRIBERY', 'BURGLARY',
               'DISORDERLY CONDUCT', 'DRIVING UNDER THE INFLUENCE',
               'DRUG/NARCOTIC', 'DRUNKENNESS', 'EMBEZZLEMENT', 'EXTORTION',
               'FAMILY OFFENSES', 'FORGERY/COUNTERFEITING', 'FRAUD', 'GAMBLING',
               'KIDNAPPING', 'LARCENY/THEFT', 'LIQUOR LAWS', 'LOITERING',
               'MISSING PERSON', 'NON-CRIMINAL', 'OTHER OFFENSES',
               'PORNOGRAPHY/OBSCENE MAT', 'PROSTITUTION', 'RECOVERED VEHICLE',
               'ROBBERY', 'RUNAWAY', 'SECONDARY CODES', 'SEX OFFENSES FORCIBLE',
               'SEX OFFENSES NON FORCIBLE', 'STOLEN PROPERTY', 'SUICIDE',
               'SUSPICIOUS OCC', 'TREA', 'TRESPASS', 'VANDALISM', 'VEHICLE THEFT',
               'WARRANTS', 'WEAPON LAWS'], dtype=object)
```

```
In [8]: len(np.unique(data["Category"]))
```

```
Out[8]: 39
```

2.1.1 Categoria de crime por numero de instancias

```
In [9]: data.groupby('Category')['Category'].count().sort_values(ascending=True)
```

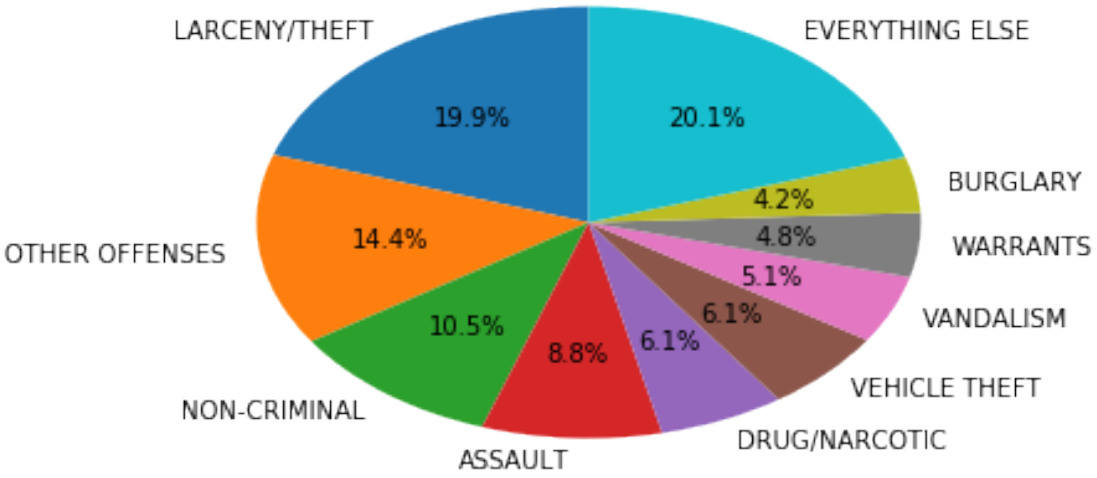
```
Out[9]: Category
        TREA                          6
        PORNOGRAPHY/OBSCENE MAT      22
        GAMBLING                    146
        SEX OFFENSES NON FORCIBLE   148
        EXTORTION                   256
        BRIBERY                     289
        BAD CHECKS                  406
        FAMILY OFFENSES             491
        SUICIDE                     508
        EMBEZZLEMENT               1166
        LOITERING                  1225
        ARSON                      1513
        LIQUOR LAWS                1903
        RUNAWAY                    1946
        DRIVING UNDER THE INFLUENCE 2268
        KIDNAPPING                 2341
        RECOVERED VEHICLE          3138
        DRUNKENNESS                4280
        DISORDERLY CONDUCT         4320
        SEX OFFENSES FORCIBLE      4388
        STOLEN PROPERTY            4540
        TRESPASS                   7326
        PROSTITUTION               7484
        WEAPON LAWS                8555
        SECONDARY CODES            9985
        FORGERY/COUNTERFEITING    10609
        FRAUD                     16679
        ROBBERY                   23000
```

```
MISSING PERSON                 25989
SUSPICIOUS OCC                 31414
BURGLARY                       36755
WARRANTS                       42214
VANDALISM                      44725
VEHICLE THEFT                  53781
DRUG/NARCOTIC                  53971
ASSAULT                        76876
NON-CRIMINAL                   92304
OTHER OFFENSES                126182
LARCENY/THEFT                 174900
Name: Category, dtype: int64
```

2.1.2 Pie chart de categorias de crime

Para visualizar a distribuicao de tipos de crime, podemos fazer um pie chart. Esse gráfico ilustra a frequência histórica de cada crime. Foram escolhidos os 9 crimes mais frequentes e o restante é representado pela categoria "EVERYTHING ELSE".

```
In [36]: count = Counter(data.Category)
         key = sorted(count, key=count.__getitem__, reverse=True)
         value = sorted(count.values(), reverse=True)

         labels = key[:9]
         labels.append("EVERYTHING ELSE")
         frequency = value[:9]
         frequency.append(sum(value[9:]))

         plt.pie(frequency, labels=labels, startangle=90, autopct='%.1f%%')
         plt.show()
```



2.2 Analise temporal

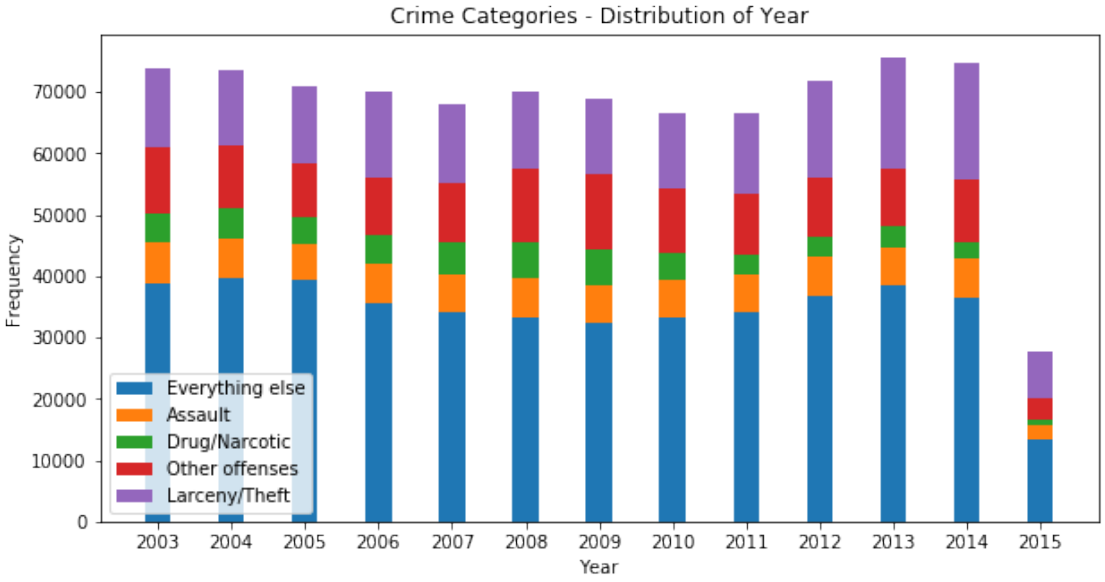
Pode-se analisar a distribuicao temporal dos dados criando graficos de barra empilhados por categoria de crime para ano, mes, dia da semana e hora.

2.2.1 Analise de categoria de crimes por ano

Verifica-se que a quantidade de crimes cometidos oscilou pouco, ocorrendo uma leve redução entre os anos 2003 e 2011. Em 2013, porém, registrou-se o maior número de crimes comparados aos últimos anos. Em 2015, os crimes foram registrados até o mês de maio, o que tornou a análise incompleta para o mesmo ano.

```
In [38]: list_years = np.sort(data.Year.unique())

         larceny = []
         other = []
         narcotic = []
         assault = []
         eelse = []

         for i in list_years:
             count = Counter(data.Category.loc[data.Year == i])
             value = sorted(count.values(), reverse=True)
             larceny.append(count.get("LARCENY/THEFT"))
             other.append(count.get("OTHER OFFENSES"))
             narcotic.append(count.get("DRUG/NARCOTIC"))
             assault.append(count.get("ASSAULT"))
             eelse.append(sum(value) - larceny[-1] - other[-1] - narcotic[-1] - assault[-1])

         N = len(list_years)

         ind = np.arange(N)    # the x locations for the groups
         width = 0.35       # the width of the bars: can also be len(x) sequence

         plt.figure(figsize = (10,5))

         p1 = plt.bar(ind, eelse, width)
         p2 = plt.bar(ind, assault, width, bottom = eelse)
         p3 = plt.bar(ind, narcotic, width, bottom = list(np.array(eelse) + np.array(assault)))
         p4 = plt.bar(ind, other, width, bottom = list(np.array(eelse) + np.array(assault) + np
         p5 = plt.bar(ind, larceny, width, bottom = list(np.array(eelse) + np.array(assault) +

         plt.xlabel('Year')
         plt.ylabel('Frequency')
         plt.title('Crime Categories - Distribution of Year')
```

```
plt.xticks(ind, list_years)
plt.legend((p1[0], p2[0], p3[0], p4[0], p5[0]), ('Everything else', 'Assault', 'Drug/
plt.show()
```

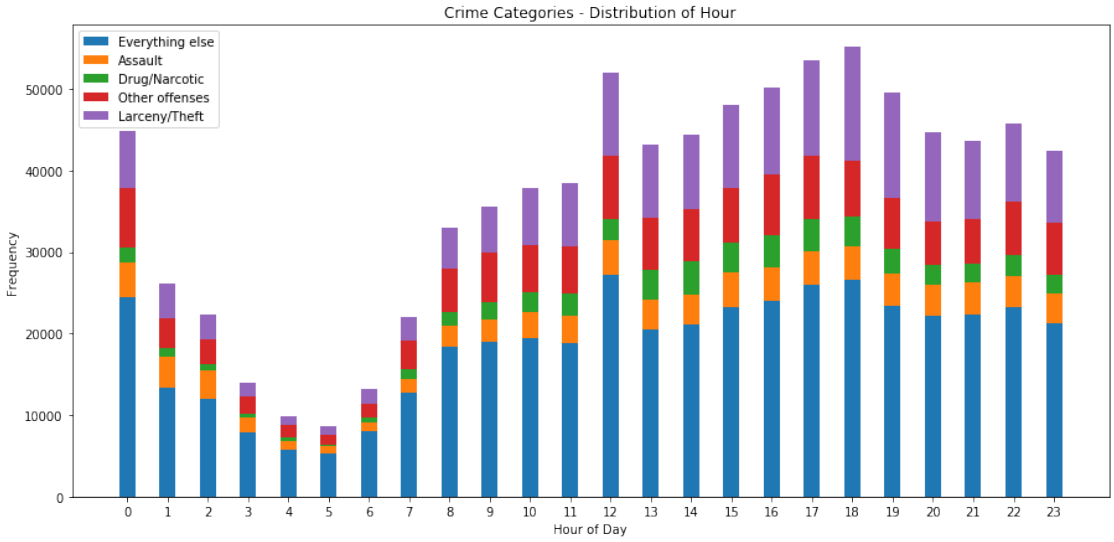


2.2.2 Analise de categoria de crimes por hora do dia

Verifica-se que o tipo de crime cometido varia bastante ao longo do dia em comparacao a variacao anual. Em determinadas horas do dia, "outras ofensas" torna-se mais frequente do que "furtos/roubos". O horário "pico" de crimes são às 18 h.

```
In [44]: list_hour = np.sort(data.Hour.unique())

         larceny = []
         other = []
         narcotic = []
         assault = []
         eelse = []

         for h in list_hour:
             count = Counter(data.Category.loc[data.Hour == h])
             value = sorted(count.values(), reverse=True)
             larceny.append(count.get("LARCENY/THEFT"))
             other.append(count.get("OTHER OFFENSES"))
             narcotic.append(count.get("DRUG/NARCOTIC"))
             assault.append(count.get("ASSAULT"))
             eelse.append(sum(value) - larceny[-1] - other[-1] - narcotic[-1] - assault[-1])
```

```
N = len(list_hour)

ind = np.arange(N)    # the x locations for the groups
width = 0.4       # the width of the bars: can also be len(x) sequence

plt.figure(figsize = (15,7))

p1 = plt.bar(ind, eelse, width)
p2 = plt.bar(ind, assault, width, bottom = eelse)
p3 = plt.bar(ind, narcotic, width, bottom = list(np.array(eelse) + np.array(assault)))
p4 = plt.bar(ind, other, width, bottom = list(np.array(eelse) + np.array(assault) + np
p5 = plt.bar(ind, larceny, width, bottom = list(np.array(eelse) + np.array(assault) +

plt.xlabel('Hour of Day')
plt.ylabel('Frequency')
plt.title('Crime Categories - Distribution of Hour')
plt.xticks(ind, list_hour)
plt.legend((p1[0], p2[0], p3[0], p4[0], p5[0]), ('Everything else', 'Assault', 'Drug/

plt.show()
```



3 Predicao categoria de crime

3.1 Utilizando coordenadas X e Y

Primeiramente, treinamos random forest apenas nos dados de localizacao (atributos X e Y) por um esquema 80% treino 20% teste, feitos estratificadamente para manter a distribuicao das classes.

```
In [10]: X_train, X_test, y_train, y_test = train_test_split(data[["X","Y"]], data["Category"]
         scaler = MinMaxScaler()
```

```
In [11]: clf = make_pipeline(MinMaxScaler(), RandomForestClassifier(n_estimators=150,  max_dep

In [12]: clf.fit(X_train, y_train)

Out[12]: Pipeline(memory=None,
              steps=[('minmaxscaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('random
                     max_depth=20, max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split...n_jobs=4,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False))])
```

3.1.1 Log Loss

Como a metrica de desempenho pedido pelo Kaggle eh o log loss, temos que pegar como saida do classificador as probabilidades previstas.

```
In [13]: pred = clf.predict_proba(X_test)

In [14]: pred.shape

Out[14]: (175610, 39)

In [15]: log_loss(y_test, pred)

Out[15]: 2.5838000471708034
```

3.1.2 Ajustando parametros

Como nao sabemos os melhores parametros para o modelos, iremos ajusta-lo no esquema 3-fold cross validation utilizando busca em grid. Primeiramente utilizaremos tambem apenas os dados das coordenadas X e Y. O pipeline que implementamos na classe ClassifierPipeline, aplica scaling de 0 a 1 nos dados das coordenadas e uma busca em grid do classificador desejado. Atentar que o output da funcao eh o log loss negado porque a busca em grid sempre maximiza o valor do score por default, portanto valores que devem ser minimizados possuem seus valores negados.

```
In [16]: from model_tuning import ClassifierPipeline
```

Nos eh retornado a melhor combinacao de parametros do grid colocado. O resultado foi um pouco melhor do que o parametro que colocamos na divisao 80/20.

```
In [17]: rf_parameters = {
                 "n_estimators":[150, 200, 250],
                 "max_depth": [10,15,20]
         }
         pipeline = ClassifierPipeline(RandomForestClassifier(), rf_parameters, n_jobs=2)
         pipeline.fit(data[["X","Y"]], data["Category"])
```

```
Best score: -2.478
Best parameters set:
        clf__max_depth: 10
        clf__n_estimators: 250
```

Eh possivel tambem visualizar os resultados para cada combinacao desejada.

```
In [18]: pipeline.get_cv_results()
```

```
Out[18]:    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
         0     107.001906      0.843753        14.374518        0.039568
         1     140.700231      1.172925        18.863728        0.183730
         2     175.722402      1.478251        23.556669        0.428332
         3     121.343305      1.389591        17.369820        0.502810
         4     160.164636      0.602939        22.831981        0.435058
         5     204.226629      3.761786        30.541275        1.958017
         6     128.117592      3.876016        19.788930        0.124079
         7     169.841316      2.587281        25.984079        0.217235
         8     209.408009      2.666703        32.485311        2.662955

           param_clf__max_depth param_clf__n_estimators  \
         0                   10                     150
         1                   10                     200
         2                   10                     250
         3                   15                     150
         4                   15                     200
         5                   15                     250
         6                   20                     150
         7                   20                     200
         8                   20                     250

                                                 params  split0_test_score  \
         0  {'clf__max_depth': 10, 'clf__n_estimators': 150}          -2.506033
         1  {'clf__max_depth': 10, 'clf__n_estimators': 200}          -2.506696
         2  {'clf__max_depth': 10, 'clf__n_estimators': 250}          -2.505753
         3  {'clf__max_depth': 15, 'clf__n_estimators': 150}          -2.515952
         4  {'clf__max_depth': 15, 'clf__n_estimators': 200}          -2.515615
         5  {'clf__max_depth': 15, 'clf__n_estimators': 250}          -2.511040
         6  {'clf__max_depth': 20, 'clf__n_estimators': 150}          -2.835732
         7  {'clf__max_depth': 20, 'clf__n_estimators': 200}          -2.803182
         8  {'clf__max_depth': 20, 'clf__n_estimators': 250}          -2.788637

            split1_test_score  split2_test_score  mean_test_score  std_test_score  \
         0          -2.448943          -2.483142        -2.479373        0.023459
         1          -2.447674          -2.484952        -2.479775        0.024372
         2          -2.448543          -2.480879        -2.478392        0.023422
         3          -2.433258          -2.505791        -2.485001        0.036822
         4          -2.429887          -2.498634        -2.481379        0.037064
         5          -2.429340          -2.496182        -2.478854        0.035534
         6          -2.670048          -2.788839        -2.764874        0.069731
         7          -2.654783          -2.760485        -2.739484        0.062377
         8          -2.638065          -2.739939        -2.722215        0.062736
```

```
   rank_test_score
0                3
1                4
2                1
3                6
4                5
5                2
6                9
7                8
8                7
```

3.2 Utilizando Dia da Semana, Mes e Coordenadas

Agora, utilizaremos tambem os dados temporais. Para isso, temos que transformar as colunas categoricas para numericas, no caso utilizaremos inicialmente dia da semana e mes.

```
In [47]: tempdf = data[['Category','DayOfWeek', 'Month', 'X', 'Y']]
         tempdf.head()
```

Out[47]:

	Category	DayOfWeek	Month	X	Y
0	WARRANTS	Wednesday	5	-122.425892	37.774599
1	OTHER OFFENSES	Wednesday	5	-122.425892	37.774599
2	OTHER OFFENSES	Wednesday	5	-122.424363	37.800414
3	LARCENY/THEFT	Wednesday	5	-122.426995	37.800873
4	LARCENY/THEFT	Wednesday	5	-122.438738	37.771541

Binarizando as features categoricas utilizando uma funcao de pandas.

```
In [48]: df = pd.get_dummies(tempdf, columns=['Month', 'DayOfWeek'])
         df.head()
```

Out[48]:

	Category	X	Y	Month_1	Month_2	Month_3	Month_4	\
0	WARRANTS	-122.425892	37.774599	0	0	0	0	
1	OTHER OFFENSES	-122.425892	37.774599	0	0	0	0	
2	OTHER OFFENSES	-122.424363	37.800414	0	0	0	0	
3	LARCENY/THEFT	-122.426995	37.800873	0	0	0	0	
4	LARCENY/THEFT	-122.438738	37.771541	0	0	0	0	

	Month_5	Month_6	Month_7	...	Month_10	Month_11	\
0	1	0	0	...	0	0	
1	1	0	0	...	0	0	
2	1	0	0	...	0	0	
3	1	0	0	...	0	0	
4	1	0	0	...	0	0	

	Month_12	DayOfWeek_Friday	DayOfWeek_Monday	DayOfWeek_Saturday	\
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	0	

```
3         0                 0                 0                   0
4         0                 0                 0                   0

   DayOfWeek_Sunday  DayOfWeek_Thursday  DayOfWeek_Tuesday  \
0                 0                   0                  0
1                 0                   0                  0
2                 0                   0                  0
3                 0                   0                  0
4                 0                   0                  0

   DayOfWeek_Wednesday
0                    1
1                    1
2                    1
3                    1
4                    1

[5 rows x 22 columns]
```

3.2.1 Ajustando parametros

Fazendo busca em grid para os melhores parametros.

```
In [22]: rf_parameters = {
                "n_estimators":[250, 300],
                "max_depth": [5,10]
         }
         pipeline = ClassifierPipeline(RandomForestClassifier(), rf_parameters, n_jobs=2)
         pipeline.fit(df.drop(columns=["Category"]), df['Category'])
```

```
Best score: -2.580
Best parameters set:
        clf__max_depth: 10
        clf__n_estimators: 300
```

3.3 Utilizando dia da semana e coordenadas

O resultado nao melhorou adicionando mes e dia da semana. Vamos tentar utilizar apenas dia da semana.

```
In [28]: tempdf = data[['Category','DayOfWeek', 'X', 'Y']]
         tempdf.head()
```

```
Out[28]:          Category  DayOfWeek           X          Y
         0        WARRANTS  Wednesday -122.425892  37.774599
         1  OTHER OFFENSES  Wednesday -122.425892  37.774599
         2  OTHER OFFENSES  Wednesday -122.424363  37.800414
         3   LARCENY/THEFT  Wednesday -122.426995  37.800873
         4   LARCENY/THEFT  Wednesday -122.438738  37.771541
```

```
In [29]: df = pd.get_dummies(tempdf, columns=['DayOfWeek'])
         rf_parameters = {
                 "n_estimators":[250, 300],
                 "max_depth": [5,10]
         }
         pipeline = ClassifierPipeline(RandomForestClassifier(), rf_parameters, n_jobs=2)
         pipeline.fit(df.drop(columns=["Category"]), df['Category'])
```

```
Best score: -2.522
Best parameters set:
        clf__max_depth: 10
        clf__n_estimators: 250
```

3.4 Utilizando hora e coordenadas

E por ultimo, utilizando apenas hora e coordenadas.

```
In [31]: tempdf = data[['Category','Hour', 'X', 'Y']]
         tempdf.head()
```

```
Out[31]:          Category  Hour           X          Y
         0        WARRANTS    23 -122.425892  37.774599
         1  OTHER OFFENSES    23 -122.425892  37.774599
         2  OTHER OFFENSES    23 -122.424363  37.800414
         3   LARCENY/THEFT    23 -122.426995  37.800873
         4   LARCENY/THEFT    23 -122.438738  37.771541
```

```
In [32]: df = pd.get_dummies(tempdf, columns=['Hour'])
         rf_parameters = {
                 "n_estimators":[250, 300],
                 "max_depth": [5,10]
         }
         pipeline = ClassifierPipeline(RandomForestClassifier(), rf_parameters, n_jobs=2)
         pipeline.fit(df.drop(columns=["Category"]), df['Category'])
```

```
Best score: -2.547
Best parameters set:
        clf__max_depth: 10
        clf__n_estimators: 300
```

4 Conclusoes

Observamos que o melhor resultado foi obtido sem considerar os dados temporais. Isso indica a necessidade de investigar modelos que considerem a ordem temporal dos dados para que os padroes de variacoes temporais sejam detectados. Apesar do banco ser bastante grande, foi possivel treinar o modelo random forest e obter uma melhora de performance quando os parametros do modelo foram ajustados.