Exercício 1

May 30, 2021

1 Exercício 1

Samuel Felipe Chenatti - 177065

1.1 Setup do ambiente

[1]:
```
import pandas as pd

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
!wget https://www.ic.unicamp.br/~wainer/cursos/1s2021/432/solar-flare.csv␣
↪--no-check-certificate -nc
```

```
O arquivo "solar-flare.csv" já existe, não será baixado.
```

1.2 Leitura dos dados

Antes de fazermos a leitura dos dados, trazemos algumas informações do dataset original como o nome das colunas:

[3]:
```
column_names = [
    'class',
    'target_spot_size',
    'spot_distribution',
    'activity',
    'evolution',
    'previous_activity',
    'hist_complex',
    'became_complex',
    'area',
```

```
    'largest_area',
    'c_class_target',
    'm_class_target',
    'x_class_target'
]
```

E a discretização dos valores categóricos (perceba que todas as colunas com exceção dos targets são categóricas):

[4]:
```
column_categories = [
    ['A', 'B', 'C', 'D', 'E', 'F', 'H'],
    ['X', 'R', 'S', 'A', 'H', 'K'],
    ['X', 'O', 'I', 'C'],
    [1, 2],
    [1, 2, 3],
    [1, 2, 3],
    [1, 2],
    [1, 2],
    [1, 2],
    [1, 2]
]
```

Finalmente, lemos os dados utilizando o Pandas:

[5]:
```
df = pd.read_csv(
    './solar-flare.csv',
    sep=' ',
    skiprows=1,
    header=None,
    names=column_names
)
```

Imprimos o ínicio e o fim do da DataSet utilizando os métodos apropriados:

[6]:
```
df.head(5)
```

[6]:

	class	target_spot_size	spot_distribution	activity	evolution	previous_activity	hist_complex	became_complex	area	largest_area	c_class_target	m_class_target	x_class_target
0	H	A	X	1	3	1	1	1	1	1	0	0	0
1	D	R	O	1	3	1	1	2	1	1	0	0	0
2	C	S	O	1	3	1	1	2	1	1	0	0	0
3	H	R	X	1	2	1	1	1	1	1	0	0	0
4	H	S	X	1	1	1	1	1	1	1	0	0	0

```
4                  1             1               2     1             1

   c_class_target  m_class_target  x_class_target
0               0               0               0
1               0               0               0
2               0               0               0
3               0               0               0
4               0               0               0
```

[7]:
```
df.tail(5)
```

[7]:
```
     class target_spot_size spot_distribution  activity  evolution  \
1061     H                S                  X         1          2
1062     H                S                  X         2          2
1063     C                S                  O         1          2
1064     H                R                  X         1          2
1065     B                X                  O         1          1

      previous_activity  hist_complex  became_complex  area  largest_area  \
1061                  1             1               1     1             1
1062                  1             1               2     1             1
1063                  1             2               2     1             1
1064                  1             1               2     1             1
1065                  1             1               2     1             1

      c_class_target  m_class_target  x_class_target
1061               0               0               0
1062               0               0               0
1063               0               0               0
1064               0               0               0
1065               0               0               0
```

Para facilitar a manipulação, separamos os dados e os targets em dois DataFrames separados:

[8]:
```
target_df = df[column_names[-3:]]
data_df = df[column_names[:-3]]
```

1.3 Converta os atributos categóricos para numéricos

Antes de continuarmos é importante ressaltar que o dado de treino não possui exemplos de todas as categorias para todas as colunas:

[9]:
```
set(column_categories[0]) - set(df['class'].unique())
```

[9]:
```
{'A'}
```

O comportamento padrão do `OneHotEncoder` para este caso é *inferir a discretização* da feature `class` e criar uma coluna numérica para cada possível valor.

Em tempo de inferência, se o encoder se deparar com um novo exemplo que possui a classe `A`, o comportamento será o de zerar todas as colunas de todas as classes.

Para garantir que durante a etapa de treino haja colunas para as classes faltantes no conjunto de treino, nós forçamos o encoder a ter conhecimento prévio de todos os possíveis valores:

[10]:
```
one_hot_encoder = OneHotEncoder(
    categories=column_categories
)
```

Finalmente, fazemos fit dos nossos dados em um array e o transformamos novamente num `DataFrame`:

[11]:
```
encoded_data = one_hot_encoder.fit_transform(data_df).toarray()

encoded_df = pd.DataFrame(
    encoded_data,
    columns=one_hot_encoder.get_feature_names(column_names[:-3])
)

encoded_df.head()
```

[11]:

	class_A	class_B	class_C	class_D	class_E	class_F	class_H	\
0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	1.0	

	target_spot_size_X	target_spot_size_R	target_spot_size_S	…	\
0	0.0	0.0	0.0	…	
1	0.0	1.0	0.0	…	
2	0.0	0.0	1.0	…	
3	0.0	1.0	0.0	…	
4	0.0	0.0	1.0	…	

	previous_activity_2	previous_activity_3	hist_complex_1	hist_complex_2	\
0	0.0	0.0	1.0	0.0	
1	0.0	0.0	1.0	0.0	
2	0.0	0.0	1.0	0.0	
3	0.0	0.0	1.0	0.0	
4	0.0	0.0	1.0	0.0	

	became_complex_1	became_complex_2	area_1	area_2	largest_area_1	\
0	1.0	0.0	1.0	0.0	1.0	
1	0.0	1.0	1.0	0.0	1.0	
2	0.0	1.0	1.0	0.0	1.0	
3	1.0	0.0	1.0	0.0	1.0	

```
4                0.0                1.0      1.0      0.0              1.0

   largest_area_2
0             0.0
1             0.0
2             0.0
3             0.0
4             0.0

[5 rows x 33 columns]
```

1.4 Centering and scaling

[12]:
```
scaler = StandardScaler()
scaler.fit(encoded_df.to_numpy())
scaler.transform(encoded_df.to_numpy())
```

[12]:
```
array([[ 0.        , -0.39994559, -0.49677321, …, -0.16120337,
         0.        ,  0.        ],
       [ 0.        , -0.39994559, -0.49677321, …, -0.16120337,
         0.        ,  0.        ],
       [ 0.        , -0.39994559,  2.01299098, …, -0.16120337,
         0.        ,  0.        ],
       …,
       [ 0.        , -0.39994559,  2.01299098, …, -0.16120337,
         0.        ,  0.        ],
       [ 0.        , -0.39994559, -0.49677321, …, -0.16120337,
         0.        ,  0.        ],
       [ 0.        ,  2.50034011, -0.49677321, …, -0.16120337,
         0.        ,  0.        ]])
```

1.5 PCA

[13]:
```
pca = PCA()
pca.fit(encoded_df.to_numpy())
```

[13]:
```
PCA()
```

[14]:
```
VARIANCE_TO_PRESERVE = 0.9

cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
dimensions_to_keep = np.count_nonzero(cumulative_variance <=␣
 ↪VARIANCE_TO_PRESERVE)

print(f'Restarão {dimensions_to_keep} dimensões se quiseremos preservar␣
 ↪{VARIANCE_TO_PRESERVE} de variância')
```

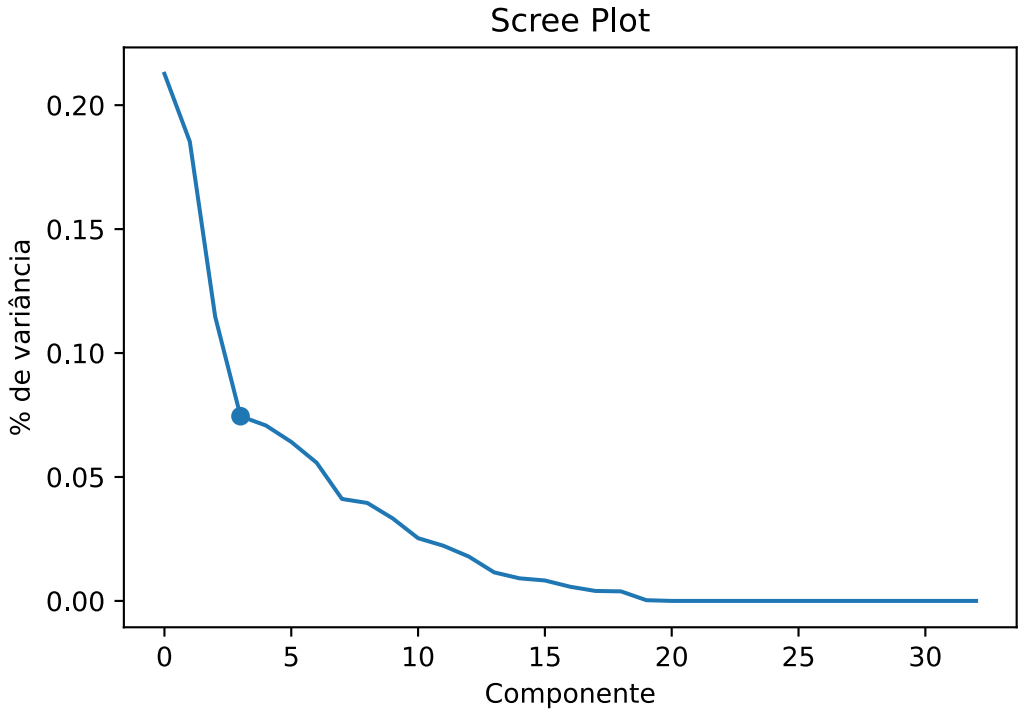
Restarão 10 dimensões se quiseremos preservar 0.9 de variância

[15]:
```
plt.title('Scree Plot')
plt.xlabel('Componente')
plt.ylabel('% de variância')

plt.plot(pca.explained_variance_ratio_)

plt.scatter(3, pca.explained_variance_ratio_[3])
```

[15]: <matplotlib.collections.PathCollection at 0x7f8b88b5d6a0>



[16]:
```
transformed_data = PCA(n_components=0.9).fit_transform(encoded_df.to_numpy())
transformed_data.shape
```

[16]: (1066, 11)

1.6 Validação cruzada e regressão linear

Split dos dados em treino/teste

[17]:
```
X_train, X_test, y_train, y_test = train_test_split(
    transformed_data,
    target_df.to_numpy(),
```

```
    train_size=0.7,
    shuffle=True
)
```

Ao definirmos as métricas que serão usadas para `cross_validate`, precisamos especificar que `MAE` e `RMSE` são métricas de erro; ou seja, quanto menor, melhor

Como consequência, dado que o Sklearn por padrão trabalha com maximização de objetivos, a biblioteca utilizará o negativo das métricas durante o `cross_validate`

[ ]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import make_scorer

SCORING = {
    'MAE': make_scorer(mean_absolute_error, greater_is_better=False),
    'RMSE': make_scorer(mean_squared_error, greater_is_better=False,␣
 ↪squared=False)
}
```

[64]:
```
def train(train_data: np.array, train_target: np.array, target_name: str = ''):
    """
    Treina e avalia o modelo em 5-fold

    Args:
        tain_data: os dados de treino
        train_target: a variável que queremos aproximar
        target_name: o nome da variável alvo

    Returns:
        Retorna um modelo fitado no conjunto de treino
    """
    model = LinearRegression()

    scores = cross_validate(
        model,
        train_data,
        train_target,
        scoring=SCORING,
        cv=5
    )

    rmse = np.abs(scores['test_RMSE'])
    mae = np.abs(scores['test_MAE'])

    print(f'Modelo fitado para o target {target_name}')
    print(f'RMSE para o 5-fold: {rmse}; médio: {rmse.mean() : .2f}')
    print(f'MAE para o 5-fold: {mae}; médio: {mae.mean() : .2f}')
```

```
    return model.fit(train_data, train_target)

for i in range(3):
    final_model = train(
        train_data=X_train,
        train_target=y_train[:, [i]],
        target_name=column_names[i-3]
    )

    rmse = mean_squared_error(y_test[:, [i]], final_model.predict(X_test),␣
 ↪squared=False)
    mae = mean_absolute_error(y_test[:, [i]], final_model.predict(X_test))

    print(f'RMSE avaliado no conjunto de teste: {rmse : .2f}')
    print(f'MAE avaliado no conjunto de teste: {mae : .2f}')
    print()
```

```
Modelo fitado para o target c_class_target
RMSE para o 5-fold: [0.74351979 1.03435036 0.87356639 0.63217747 0.71201207];
médio:  0.80
MAE para o 5-fold: [0.48774875 0.45242242 0.47362723 0.36776186 0.43944429];
médio:  0.44
RMSE avaliado no conjunto de teste:  0.71
MAE avaliado no conjunto de teste:  0.41

Modelo fitado para o target m_class_target
RMSE para o 5-fold: [0.45426935 0.16914627 0.15255222 0.15194575 0.44073959];
médio:  0.27
MAE para o 5-fold: [0.12240585 0.08132643 0.07908115 0.08117675 0.1274941 ];
médio:  0.10
RMSE avaliado no conjunto de teste:  0.27
MAE avaliado no conjunto de teste:  0.10

Modelo fitado para o target x_class_target
RMSE para o 5-fold: [0.07533596 0.02761945 0.02873407 0.11051275 0.08515321];
médio:  0.07
MAE para o 5-fold: [0.01804596 0.01492437 0.01483993 0.01818182 0.02062457];
médio:  0.02
RMSE avaliado no conjunto de teste:  0.11
MAE avaliado no conjunto de teste:  0.02
```