MINERAÇÃO DE DADOS COMPLEXOS

EXTECAMP

Dataset:
Chicago Bicycle Sharing Data

Grupo "E Éramos Quatro":
Felipe Pereira
Anderson Rocha
Alexandre Guidin
~~Miguel Filho~~

Introdução

O foco deste trabalho é a realização da análise do dataset e a criação do modelo de predição para o problema proposto.

O problema proposto é o de Compartilhamento de Bicicletas. Neste conjunto, temos uma série de atributos e, como target, devemos prever a quantidade de bicicletas que serão alugadas em um determinado período.

Para isto, realizamos o trabalho que se segue.

1. Análise dos dados

O problema foi proposto por um desafio do Kaggle (https://www.kaggle.com/yingwurenjian/chicago-divvy-bicycle-sharing-data/). O autor do dataset disponibilizou 2 datasets para o problema:

-Raw Data:

Todos os dados de aluguel de bicicletas em Chicago, de 2013 a 2017. Também possui os dados de clima. São os dados puros, por isso, possuem muitos valores nulos.

-**Data**:

Contém dados que foram limpos pelo próprio autor do desafio, retirando registros com valores faltantes e também com corridas de mais de uma hora. Utilizamos este dataset.

Mesmo pegando um dataset já trabalhado, tivemos que realizar algumas modificações no mesmo, para torná-lo passível de ser utilizado para predições. Vamos falar sobre isto nos próximos tópicos.

Dimensionalidade :

O dataset proposto possui as seguintes dimensões:

9.495.235 registros
23 atributos

Este dataset foi importado em um banco de dados PostgreSQL para transformação.

Estatísticas:

Através de consultas SQL, pudemos verificar que não existe nenhum valor nulo no dataset. Também não parece existir nenhum valor corrompido ou incorreto, pois todos os valores estão dentro de faixas esperadas para o atributo.

Modelagem de dados:

Os dados disponibilizados foram os dados de cada aluguel em si. Porém nosso desafio é prever a quantidade de aluguéis em um determinado período. Por isso, precisamos alterar a modelagem dos dados.

O primeiro desafio foi definir o que seria o “determinado período”. Decidimos iniciar a análise com predições por hora, pois isto nos ajudaria a ter mais dados para o modelo do que se fossemos fazer com predições por dia.

Tomada esta decisão, foi preciso olhar para o dataset com outros olhos, afim de transformarmos os dados para que fossem passíveis de predição. Para isto realizamos algumas transformações nos mesmos.

Transformação intermediária:

Transformamos as variáveis categóricas Usertype, Gender e Events em variáveis discretas:

Usertype:

Usertype possui 3 valores distintos: Customer, Dependent e Subscriber. Transformamos cada categoria em um atributo diferente: is_customer, is_dependent e is_subscriber. Cada um desses novos atributos possui apenas 2 valores possíveis: (0 - False) e (1 - True).

Gender:

Gender possui 2 valores distintos: Female e Male. Transformamos cada categoria em um atributo diferente: is_female e is_male. Cada um desses novos atributos possui apenas 2 valores possíveis: (0 - False) e (1 - True).

Events:

O atributo events é o responsável pelas características de weather. Os valores distintos que ele possui são: tstorms, unknown, cloudy, snow or rain, not_clear e clear. Transformamos cada categoria em um atributo diferente: is_tstorms, is_unknown, is_cloudy, is_rain, is_not_clear e is_clear. Cada um desses novos atributos possui apenas 2 valores possíveis: (0 - False) e (1 - True).

Estas transformações nos possibilitaram agregar os valores por qualquer período determinado. Porém, percebemos que mais uma transformação nos dados era necessária, afim de tornar o modelo justo.

Transformação final:

Nosso modelo intermediário possuía todos os atributos do modelo limpo fornecido no desafio do Kaggle. Além disso, as variáveis categóricas estavam disponíveis em um formato discreto, permitindo agegação por qualquer período escolhido. Porém uma escolha “moral” se tornou necessária.

Ao pensar sobre o problema que gostaríamos de resolver (predição de aluguel de bicicletas), nos pareceu incoerente manter algumas informações por 2 aspectos: em primeiro lugar, se você quer prever algo, significa que você não sabe de antemão o que vai ocorrer, com isso você não possui informações detalhadas sobre o que irá ocorrer, por isso não é factível que você tenha de antemão algumas informações como quantas mulheres e homens irão alugar as bicicletas, por exemplo. Outro problema é: se você tivesse essa informação, a quantidade de aluguel de bicicletas seria simplesmente a soma desses valores, não fazendo sentido criar um modelo de predição para este problema. Mesmo que você decidisse criar um modelo, mas alimentasse o modelo com estas informações, o modelo logo iria aprender que o target é a soma simples de algumas features (como (is_female + is_male) ou (is_customer + is_dependent + is_subscriber)).

Tomada esta decisão, e também tomada a decisão de agregar os dados por um período, também fez com que algumas features se tornassem desnecessárias ou inutilizáveis.

Com isto em mente, algumas features foram intencionalmente descartadas para tornar o modelo factível. Foram descartadas as seguintes features:

- Usertype
- Gender
- Starttime
- Stoptime
- Tripduration
- Events
- From_station_id
- From_station_name
- Latitude_start
- Longitude_start
- Dpcapacity_start
- Is_customer
- Is_dependent
- Is_subscriber
- Is_female
- Is_male

Nossa modelagem de dados final ficou com o seguinte aspecto:

- **year**: ano
- **month**: mês
- **week**: número da semana no ano

- **day**: dia do mês
- **is_weekend**: 1 = é fim de semana / 0 = não é fim de semana
- **mean_temperature**: média da temperatura naquela hora
- **median_temperature**: mediana da temperatura naquela hora
- **tstorms**: 1 = houve thunder storms naquela hora / 0 = não houve
- **unknown**: 1 = houve unknown naquela hora / 0 = não houve
- **cloudy**: 1 = houve cloudy naquela hora / 0 = não houve
- **rain_or_snow**: 1 = houve rain_or_snow naquela hora / 0 = não houve
- **not_clear**: 1 = houve not clear naquela hora / 0 = não houve
- **clear**: 1 = houve clear naquela hora / 0 = não houve
- **rentals**: quantidade de aluguéis naquela hora (variável target)

Normalização:

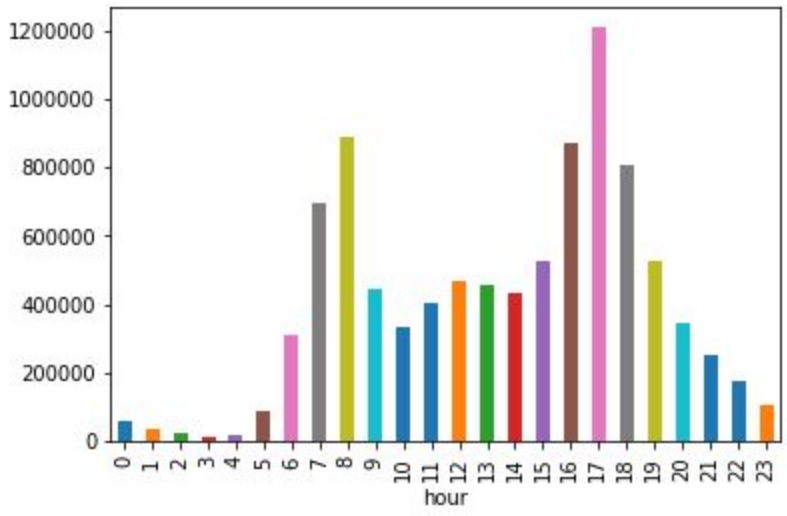
Os dados foram normalizados para influenciar positivamente na criação do modelo:

```
#Pré-Processamento dos dados

#Normalização
data.year = preprocessing.scale(data.year)
data.month = preprocessing.scale(data.month)
data.week = preprocessing.scale(data.week)
data.day = preprocessing.scale(data.day)
data.hour = preprocessing.scale(data.hour)
data.mean_temperature = preprocessing.scale(data.mean_temperature)
data.median_temperature = preprocessing.scale(data.median_temperature)
```

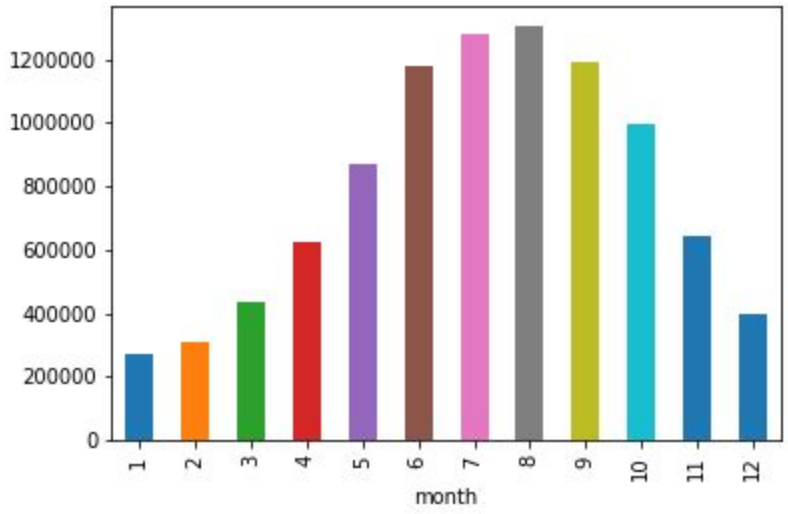
Relação entre aluguel e hora do dia:

Conforme pode-se esperar, as bicicletas são mais alugadas durante os horários de pico, que correspondem aos horários em que as pessoas estão realizando um trajeto de deslocamento casa->trabalho ou trabalho->casa:



Relação entre aluguel e temperatura:

Através da análise do aluguel ao longo do ano, podemos verificar o efeito das estações e consequentes condições climáticas sobre o aluguel de bicicletas. Os dados mostram uma preferência pelo aluguel durante os meses mais quentes do ano:



Correlação entre os atributos:

Podemos verificar uma correlação positiva com o target, principalmente, entre os atributos “hora” e “temperatura”:

	year	month	week	day	hour	is_weekend	mean_temperature
year	1.000000	-0.012281	-0.011520	0.000423	-0.002638	0.001325	0.076302
month	-0.012281	1.000000	0.969205	-0.006392	-0.006376	-0.014476	0.257745
week	-0.011520	0.969205	1.000000	-0.007392	-0.006186	-0.015134	0.255064
day	0.000423	-0.006392	-0.007392	1.000000	-0.003438	-0.026462	-0.016198
hour	-0.002638	-0.006376	-0.006186	-0.003438	1.000000	0.002205	0.084455
is_weekend	0.001325	-0.014476	-0.015134	-0.026462	0.002205	1.000000	-0.009158
mean_temperature	0.076302	0.257745	0.255064	-0.016198	0.084455	-0.009158	1.000000
median_temperature	0.076302	0.257745	0.255064	-0.016198	0.084455	-0.009158	1.000000
tstorms	0.003549	-0.001537	-0.001183	-0.012698	0.017825	-0.010013	0.075140
unknown	-0.003490	-0.003441	-0.002941	0.001871	-0.001143	-0.004803	0.003980
cloudy	0.023567	0.041586	0.036959	-0.001572	0.047427	0.006108	0.174433
rain_or_snow	-0.012664	-0.061673	-0.058947	-0.005611	0.003428	0.006926	-0.152806
not_clear	-0.032328	0.002691	0.000403	-0.004731	-0.029590	-0.004740	-0.019605
clear	-0.009926	0.004487	0.008848	0.013531	-0.065340	-0.010734	-0.105052
rentals	0.168986	0.147929	0.144008	-0.134087	0.252474	-0.002065	0.475935

median_temperature	tstorms	unknown	cloudy	rain_or_snow	not_clear	clear	rentals
0.076302	0.003549	-0.003490	0.023567	-0.012664	-0.032328	-0.009926	0.168986
0.257745	-0.001537	-0.003441	0.041586	-0.061673	0.002691	0.004487	0.147929
0.255064	-0.001183	-0.002941	0.036959	-0.058947	0.000403	0.008848	0.144008
-0.016198	-0.012698	0.001871	-0.001572	-0.005611	-0.004731	0.013531	-0.134087
0.084455	0.017825	-0.001143	0.047427	0.003428	-0.029590	-0.065340	0.252474
-0.009158	-0.010013	-0.004803	0.006108	0.006926	-0.004740	-0.010734	-0.002065
1.000000	0.075140	0.003980	0.174433	-0.152806	-0.019605	-0.105052	0.475935
1.000000	0.075140	0.003980	0.174433	-0.152806	-0.019605	-0.105052	0.475935
0.075140	1.000000	-0.000701	-0.179072	-0.030399	-0.009348	-0.029537	-0.015352
0.003980	-0.000701	1.000000	-0.014758	-0.002505	-0.000770	-0.002434	-0.000146
0.174433	-0.179072	-0.014758	1.000000	-0.639909	-0.196783	-0.621764	0.193332
-0.152806	-0.030399	-0.002505	-0.639909	1.000000	-0.033406	-0.105551	-0.147542
-0.019605	-0.009348	-0.000770	-0.196783	-0.033406	1.000000	-0.032459	-0.007363
-0.105052	-0.029537	-0.002434	-0.621764	-0.105551	-0.032459	1.000000	-0.112553
0.475935	-0.015352	-0.000146	0.193332	-0.147542	-0.007363	-0.112553	1.000000

2. Criação do modelo

Separamos o dataset em 80% para treinamento (sendo 60% treinamento e 20% validação) e 20% para o teste final:

```
#Modelos a serem testados
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.linear_model import LinearRegression, BayesianRidge, LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor, RandomForestRegressor

#Retirada da variável target das features de predição
X = data.drop('rentals',1)
y = data.rentals

#Separação de conjunto de testes
X_model, X_test, y_model, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

#Separação de conjunto de validação
X_train, X_val, y_train, y_val = train_test_split(X_model, y_model, test_size=0.2, random_state=0)
```

Também decidimos testar vários modelos diferentes para verificar a eficiência de cada um:

```
#Treinamento de modelos
lr = LinearRegression(n_jobs=5, fit_intercept=True)
logr = LogisticRegression(penalty='l2', C=1.0, fit_intercept=True, random_state=0, solver='liblinear', n_jobs=5)
dt = DecisionTreeRegressor(max_depth=10, criterion='mse', splitter='best', random_state=0, presort=True)
dtr = AdaBoostRegressor(dt,n_estimators=500, learning_rate=0.1, random_state=0)
rf = RandomForestRegressor(n_estimators=100, criterion='mse', max_features='auto', random_state=0, n_jobs=5)
blr = BayesianRidge(n_iter=1000, fit_intercept=True)

#Criação de vetor de modelos
algs = []
algs.append(lr)
algs.append(logr)
algs.append(dt)
algs.append(dtr)
algs.append(rf)
algs.append(blr)

#Fit dos modelos
for alg in algs:
    print('Fitting: ', type(alg).__name__)
    alg.fit(X_model, y_model)
```

```
Fitting:  LinearRegression
Fitting:  LogisticRegression
```

```
/home/felipe/anaconda3/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:1228: UserWarning: 'n_jobs' > 1 does
not have any effect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 5.
  " = {}.".format(self.n_jobs))
```

```
Fitting:  DecisionTreeRegressor
Fitting:  AdaBoostRegressor
Fitting:  RandomForestRegressor
Fitting:  BayesianRidge
```

Para cada modelo, testamos alguns hiper-parâmetros diferentes:

- **Linear Regression**:
    - **Fit_Intercept**: Se há ou não o intercept no eixo Y pela função linear. Testamos tanto com interceptação (True) quanto sem interceptação (False), mas os resultados não foram afetados. Mantivemos o valor padrão (True).

- **LogisticRegression**:
    - **Penalty**: a penalidade para evitar overfit. Tanto L1 quanto L2 possuem performance igual neste caso.
    - **C**: Foram testados valores de 0.001 a 100, e o resultado não foi alterado. Continuamos com o padrão (1).
    - **fit_intercept**: Utilizado para colocar um bias na função de classificação. Não influenciou nos resultados.
    - **solver**: Foram testados outros modelos de classificação (sag e saga), mas ficamos com o padrão de liblinear.

- **Decision Tree**:
    - **Criterion**: O tipo de função usado para medir a qualidade de um split (qual feature é mais relevante). MAE e MSE são muito parecidas, com um ligeiro ganho para MSE.
    - **Splitter**: O tipo de função usado realizar o split. O modo ‘best’ (escolha do melhor split) obteve melhor resultado do que o modo ‘random’ (escolha aleatória do split).
    - **Max_Depth**: Profundidade máxima da árvore de decisão.

- Presort: Ordena os dados para acelerar o split. Recomendado quando não existem muitos dados (o que é o nosso caso).

- **AdaBoost Regressor**:
  - **N_Estimators**: A quantidade de objetos do ensemble, no caso, a quantidade de árvores de decisão.
  - **Learning Rate**: A velocidade do learning rate na descida do gradiente. Testamos os valores 1, 0.1 e 0.01. Ficamos com o valor 0.1, que apresentou resultados melhores.

- **RandomForestClassifier**:
  - **N_estimators**: A quantidade de árvores a serem colocadas na floresta. Testamos com 10, 100 e 1000. A quantidade de 100 obteve uma melhor relação de custo/benefício.
  - **Criterion**: O tipo de função usado para medir a qualidade de um split (qual feature é mais relevante). Gini e Entropy são muito parecidas, com um ligeiro ganho para Entropy.
  - **Max_Features**: Testado com auto (raiz quadrada da quantidade total) e também com a quantidade total. O teste com auto ficou melhor.

- **BayesianRidge**:
  - **N_iter**: A quantidade de iterações do modelo.
  - **Fit Intercept**: Se o modelo irá usar um bias ou não.

Testamos os modelos em validação para verificar as melhoras obtidas:

	Name	Type	R2	MAE	MSE
0	LinearRegression	Train	0.323172	187.213935	72950.151816
1	LogisticRegression	Train	-0.083556	200.826578	116788.336282
2	DecisionTreeRegressor	Train	0.893314	57.073984	11498.862146
3	AdaBoostRegressor	Train	0.960174	52.439189	4292.542740
4	RandomForestRegressor	Train	0.991510	14.508301	915.028547
5	BayesianRidge	Train	0.323140	187.200312	72953.635601
6	LinearRegression	Validation	0.320537	186.327603	71867.391533
7	LogisticRegression	Validation	-0.081194	201.124391	114358.864777
8	DecisionTreeRegressor	Validation	0.881600	59.083956	12523.314164
9	AdaBoostRegressor	Validation	0.959540	52.593823	4279.526486
10	RandomForestRegressor	Validation	0.991866	14.580894	860.304890
11	BayesianRidge	Validation	0.320621	186.288606	71858.594126

3. Acurácia de modelos

Para os testes finais, utilizamos o dataset de teste separado para isso e que foi rodado apenas uma vez. Analisamos a performance dos modelos de acordo com as métricas de erro:

36	LinearRegression	Test	0.312564	189.855222	74930.489941
37	LogisticRegression	Test	-0.094791	204.497545	119332.151213
38	DecisionTreeRegressor	Test	0.874726	63.518099	13654.849082
39	AdaBoostRegressor	Test	0.930690	50.334499	7554.829703
40	RandomForestRegressor	Test	0.939110	39.605608	6637.032840
41	BayesianRidge	Test	0.312393	189.854716	74949.132372

Nossa conclusão é de que os 2 melhores modelos são o AdaBoostRegressor e o RandomForestRegressor, pois possuem um maior R2 e menor MAE e MSE.

Entre os 2, o RandomForest apresentou um comportamento de overfit, pois foi extremamente bem no treinamento e validação e, embora tenha ido bem no teste, não foi tão bem quanto nas outras etapas.

Já o AdaBoost nos pareceu mais maduro pois, apesar de possuir métricas ligeiramente piores que o Random Forest, o AdaBoost manteve sua performance em todas as etapas (treinamento, validação e teste), com pouca variação entre elas, o que indica um modelo mais maduro e menos propenso a estar em overfit.