MO432 – Aprendizado Supervisionado

Prof. Jacques Wainer

Giovanni Vitale Mazzanati – RA 102447

Exercício 3 – Relatório Final

Logo após a leitura do arquivo, iniciei o pré-processamento. Além da remoção da coluna de datas, como fora solicitado, foram criadas duas novas colunas: a primeira com o valor do ouro na semana seguinte, nomeada "Valor+1"; a segunda com um binário para indicar se o valor do ouro subiu (1) ou não (0) naquela semana específica.

Com isso, decidi tratar o problema de forma separada. Um problema de regressão para encontrar o valor da semana seguinte – encontrar "Valor+1" – e outro problema de classificação para indicar – entre 0 ou 1 – se o valor do ouro subirá ou não.

No mais, o único outro pré-processamento realizado foi a separação prévia dos valores de treino (com 996 entradas) e de teste (as 100 primeiras entradas). Fiz testes com normalização (média zero e desvio padrão um) no conjunto completo, apenas no conjunto de treino e apenas nas colunas de valores do conjunto de treino, mas nenhum dos testes apresentou alteração em qualquer resultado e, portanto, mantive os valores na forma original.

Os testes para selecionar os algoritmos foram feitos com o conjunto de treino completo, ou seja, todos os 996 valores. A regressão linear tradicional e a regressão linear com regularização L1 (Lasso) foram testadas apenas como regressores, enquanto a regressão logística foi testada apenas como classificador. Para ambos foram testados: regressão linear com regularização L2 (Ridge), SVM linear, SVM com kernel RBF, KNN, MLP, Árvore de decisão (Decision Tree), Random Forest e Gradient Boosting.

Foram testadas as seguintes variações de hiperparâmetros, além dos valores *default* de cada algoritmo:

- Regressão linear: nenhum;
- Ridge: alpha loguniform entre 10^{-3} e 10^{3};
- Lasso: alpha loguniform entre 10^{-3} e 10^{3};
- Regressão logística: nenhum;
- SVM linear: C loguniform entre 2^{-5} e 2^{15}, épsilon 0.1 ou 0.3;
- SMV "rbf": C loguniform entre 2^{-5} e 2^{15}, épsilon 0.1 ou 0.3, gamma loguniform entre 2^{-9} e 2^{3};
- KNN: k_neighbors 10 valores aleatórios entre 1 e 100;
- MLP: número de neurônios entre 5 e 20, aumentando de 3 em 3;
- Árvore de Decisão: alpha aleatório entre 0 e 0.04;
- Random Forest: n_list $10, 100$ e 1000, features $5, 10$ e 22;
- GBM: n_list aleatório entre 5 e 100, learning rate aleatório entre $0{,}01$ e $0{,}30$, depth 2 ou 3.

A tabela abaixo exibe o melhor resultado encontrado para cada algoritmo. A primeira coluna exibe o valor do coeficiente R^2, tomado como *score* do problema de regressão – encontrar o valor do ouro na semana seguinte. A segunda coluna exibe a acurácia do problema de classificação, que é o *score* deste problema de definir se o valor do ouro vai subir ou não. Por fim, a terceira coluna exibe a raiz do erro quadrático médio (RMSE) do problema de regressão.

Tabela 1: resultados dos algoritmos testados.

Modelo	Score (R^2) Regressão	Score (acurácia) Classificação	RMSE Regressão
Regressão Linear	0,9971	-	22,88
Linear L2 (Ridge)	0,9971	0,5592	22,88
Linear L1 (Lasso)	0,9971	0,5592	22,88
Regressão logística	-	0,5592	-
SVM Linear	0,9969	0,5592	24,89
SVM kernel rbf	0,8036	0,5592	368,86
KNN	0,9976	0,7108	55,40
MLP	0,9971	0,5592	22,58
Árvore de Decisão	0,9999	0,9759	61,04
Random Forest	0,9993	0,9759	55,69
Gradient Boosting	0,9982	0,7098	52,79

De modo geral, todos os modelos responderam bem à regressão. Dado que os dados apresentam forte correlação, o algoritmo escolhido para esse problema foi a regressão linear. Para o problema de classificação, em contrapartida, os resultados não foram tão satisfatórios. Todos os modelos lineares, ambos os SVM's e o MLP classificaram todos os valores como 1, ou seja, indicaram que o valor do ouro sempre vai subir. Os algoritmos Decision Tree e Random Forest apresentam uma acurácia demasiadamente alta para o problema, indicando um provável *overfitting* nos dados. Assim, os únicos modelos considerados para o problema de classificação foram o KNN e o gradient boosting (GBM), mesmo com acurácia na casa dos 70%.

Para esses dois problemas, testei mais alguns parâmetros, dessa vez com divisão por séries temporais. Além dos parâmetros dos algoritmos, também testei possibilidades para o número de *splits*. Para o KNN, testei o número de vizinho (*k_neighbors*) em valores consecutivos de 1 a 25 e o número de *splits* de 2^1 a 2^9, contínua no expoente, ou seja, $2, 4, 8, \dots, 512$ divisões do conjunto de treino. Os melhores valores encontrados foram 512 (2^9) *splits* com 12 vizinhos.

Já para o GBM testei para *n_estimators* os valores $1, 10, 100, 1000$, para *learning_rate* os valores $0{,}001, 0{,}01, 0{,}1, 1, 10$ e *max_depth* $1, 2, 3, 4, 5$, além das mesmas possibilidades para o número de *splits*. O melhor resultado foi *n_estimators* = 1, *learning_rate* = 0,001 e *max_depth* = 1, além dos mesmos 512 *splits*. Por fim, o algoritmo escolhido foi o *gradient boosting*, pois este conseguiu um *score* de 71% no conjunto de treino, contra 64% no KNN.

Para finalizar, apliquei os dois modelos escolhidos – regressão linear para prever a variação e GBM para prever se subirá – no conjunto de teste. O problema de regressão apresentou correlação de $R^2 = 0{,}92$ e $\boldsymbol{RMSE = 43,86}$. O problema de classificação apresentou $\boldsymbol{accuracy = 58}\%$.

Em anexo, segue o código da alternativa escolhida para resolução do problema.

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

from sklearn.metrics import mean_squared_error, accuracy_score, r2_score
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingClassifier
```

```
# Lendo o arquivo e o exibindo.

df = pd.read_csv("https://www.ic.unicamp.br/~wainer/cursos/1s2021/432/ouro2.csv")

# Imprime o início e o fim
df
```

	Data	Valor
0	13/06/2021	1868.0
1	06/06/2021	1879.6
2	30/05/2021	1892.0
3	23/05/2021	1905.3
4	16/05/2021	1878.9
...	...	...
1091	16/07/2000	280.1
1092	09/07/2000	281.1
1093	02/07/2000	283.2
1094	25/06/2000	290.1
1095	18/06/2000	282.7

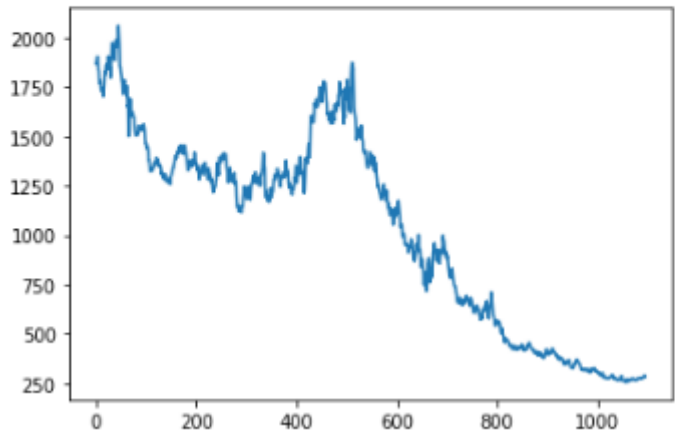
1096 rows × 2 columns

```
# Excluindo as datas

df = df.drop(["Data"], axis=1)
df
```

	Valor
0	1868.0
1	1879.6
2	1892.0
3	1905.3
4	1878.9
...	...
1091	280.1
1092	281.1
1093	283.2
1094	290.1
1095	282.7

1096 rows × 1 columns

```
# Visualizando graficamente os dados
# Ordem decrescente (mais recente - mais antiga) de datas

plt.plot(df)
plt.show()
```



```
df["Valor+1"] = df["Valor"].shift(1)
df[np.isnan(df)] = 1868
df
```

	Valor	Valor+1
0	1868.0	1868.0
1	1879.6	1868.0
2	1892.0	1879.6
3	1905.3	1892.0
4	1878.9	1905.3
...	...	...
1091	280.1	277.7
1092	281.1	280.1
1093	283.2	281.1
1094	290.1	283.2
1095	282.7	290.1

1096 rows × 2 columns

```
# Cria uma coluna para medir a variação do preço
# Cria uma coluna que diz se o preço subiu ou desceu

sobe_ou_desce = []

for i in range(len(df) - 1):
  if df.iloc[i, 0] - df.iloc[i + 1, 0] >= 0:
    sobe_ou_desce.append(1)
  else:
    sobe_ou_desce.append(0)

sobe_ou_desce.append(0)

df["SobeDesce"] = sobe_ou_desce
df
```

	Valor	Valor+1	SobeDesce
0	1868.0	1868.0	0
1	1879.6	1868.0	0
2	1892.0	1879.6	0
3	1905.3	1892.0	1
4	1878.9	1905.3	1
...	...	...	...
1091	280.1	277.7	0
1092	281.1	280.1	0
1093	283.2	281.1	0
1094	290.1	283.2	1
1095	282.7	290.1	0

1096 rows × 3 columns

```
# Separando os atributos de saída
# Regressão para a coluna "Variação"
# Classificação para a coluna "SobeDesce"

y_valor = df.iloc[:100, 0]
y_reg = df.iloc[:100, 1]
y_clas = df.iloc[:100, 2]

x_valor = df.iloc[100:, 0]
x_reg = df.iloc[100:, 1]
x_clas = df.iloc[100:, 2]

x_valor = np.array(x_valor).reshape(-1, 1)
y_valor = np.array(y_valor).reshape(-1, 1)

print(y_valor.shape)
print(y_reg.shape)
print(y_clas.shape)

print(x_valor.shape)
print(x_reg.shape)
print(x_clas.shape)
```

```
(100, 1)
(100,)
(100,)
(996, 1)
(996,)
(996,)
```

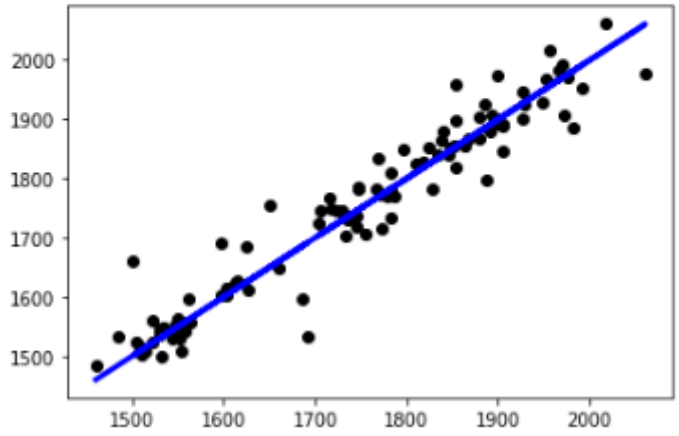
```
# Analisando mais informações

print(f"O ouro subiu {x_clas.sum()} vezes,")
print(f"o que representa {100 * x_clas.sum() / 996:.2f} % das ocasiões.")
```

```
O ouro subiu 557 vezes,
o que representa 55.92 % das ocasiões.
```

Regressão Linear

```
# Treinando o conjunto de treino
# Calculando o RMSE no conjunto de treino
lin_reg_def = cross_val_score(LinearRegression(),
                          x_valor,
                          x_reg,
                          cv=5,
                          scoring="neg_root_mean_squared_error")
print(f"RMSE do conjunto de treino: {np.mean(-lin_reg_def):.2f}")

# Calculando o score r2 no conjunto de treino
reg_lin = LinearRegression().fit(x_valor, x_reg)
print(f"Score regressão no treino: {reg_lin.score(x_valor, x_reg):.2f} \n")

# Fazendo a previsão no conjunto teste
pred_lin_reg = reg_lin.predict(y_valor)
pred_lin_reg = np.array(pred_lin_reg).reshape(-1, 1)

# Calculando o RMSE no conjunto teste
print(f"RMSE do conjunto de teste: {mean_squared_error(y_reg, pred_lin_reg) ** 0.5:.2f}")
# Calculando o score r2 no conjunto de teste
print(f"Score regressão no teste: {reg_lin.score(pred_lin_reg, y_reg):.2f}")

# Um gráfico para ajudar
plt.scatter(y_valor, y_reg,  color='black')
plt.plot(y_valor, pred_lin_reg, color='blue', linewidth=3)
plt.show()
```

```
RMSE do conjunto de treino: 22.88
Score regressão no treino: 1.00

RMSE do conjunto de teste: 43.86
Score regressão no teste: 0.92
```



Gradient Boosting

```
# Treinando o conjunto de treino
# Calculando o RMSE no conjunto de treino
gbm_clf_def = cross_val_score(GradientBoostingClassifier(),
                          x_valor,
                          x_clas,
                          cv=TimeSeriesSplit(n_splits=2**9).split(x_valor),
                          scoring="neg_root_mean_squared_error")
print(f"RMSE do conjunto de treino: {np.mean(-gbm_clf_def):.2f}")

# Calculando o score r2 no conjunto de treino
clas_gbm = GradientBoostingClassifier().fit(x_valor, x_clas)
print(f"Score classificação no treino: {clas_gbm.score(x_valor, x_clas):.2f} \n")

# Fazendo a previsão no conjunto teste
pred_gbm_clas = clas_gbm.predict(y_valor)
pred_gbm_clas = np.array(pred_gbm_clas).reshape(-1, 1)

# Calculando o RMSE no conjunto teste
print(f"RMSE do conjunto de teste: {mean_squared_error(y_clas, pred_gbm_clas) ** 0.5:.2f}")
# Calculando o score r2 no conjunto de teste
print(f"Score regressão no teste: {clas_gbm.score(pred_gbm_clas, y_clas):.2f}")
# Accuracy (acurácia) do conjunto de teste
print(f"Accuracy: {accuracy_score(y_clas, pred_gbm_clas):.2f}")

# Um gráfico para ajudar
plt.scatter(y_valor, y_clas,  color='black')
plt.plot(y_valor, pred_gbm_clas, color='blue', linewidth=3)
plt.show()
```

```
RMSE do conjunto de treino: 0.53
Score classificação no treino: 0.71

RMSE do conjunto de teste: 0.65
Score regressão no teste: 0.42
Accuracy: 0.58
```