Lista 1 de Machine Learning

Gabriel Dias Vilela

IMPA, Verao 2025

Sumário

1 Exercicio 1a

Falso. A diferena entre o erro de teste e o de treino, mesmo no caso em que o erro de teste baixo, pode trazer informaes teis sobre o grau de complexidade ideal para o modelo no contexto do problema, bem como auxiliar na identificao de overfitting etc...

2 Exercicio 1b

Verdadeiro. A hiptese sobre a distribuio dos erros gera como consequncia uma hiptese sobre a distribuio da varivel t que por sua vez comparada a distribuio observada de t (t_{obs}) para analisar H_0.

3 Exercicio 1c

Falso. Como o banco pretende atribuir peso maior aos erros relacionados a no-identificao de transaes fraudulentas, necessrio avaliar a acurcia do modelo apenas na analise de amostras fraudulentas tambm.

4 Exercicio 1d

Verdadeiro. No caso limite, basta tomar $\lambda = 0$. Neste caso, a regresso de ridge se comportar como uma regresso linear e, portanto, ter a mesma performance.

5 Exercicio 1e

Verdadeiro. Os intervalos de confiana apresentados so deduzidos tendo como hiptese que os erros so independentes, identicamente distribudos e seguem distribuio normal $N(0, \sigma^2)$. No caso em que essa hiptese no vlida, utilizar uma distribuio gerada via bootstrap para avaliar intervalos de confiana se mostra mais vantajoso devido ao seu carter "emprico"e sua proximidade com a realidade dos dados.

6 Exercicio 2a

Reescrevendo a varincia:

$$\Sigma = V(\epsilon) = \mathbb{E}\left[(\epsilon - \mathbb{E}[\epsilon])(\epsilon - \mathbb{E}[\epsilon])^\top\right] = \mathbb{E}[B_{n\times n}]$$

Note que:

$$\epsilon - \mathbb{E}[\epsilon] = \begin{bmatrix} \epsilon_1 - \mathbb{E}[\epsilon_1] \\ \epsilon_2 - \mathbb{E}[\epsilon_2] \\ \vdots \\ \epsilon_n - \mathbb{E}[\epsilon_n] \end{bmatrix}$$

Portanto:

$$(\epsilon - \mathbb{E}[\epsilon])(\epsilon - \mathbb{E}[\epsilon])^\top = \begin{bmatrix} (\epsilon_1 - \mathbb{E}[\epsilon_1])^2 & (\epsilon_1 - \mathbb{E}[\epsilon_1])(\epsilon_2 - \mathbb{E}[\epsilon_2]) & \cdots & (\epsilon_n - \mathbb{E}[\epsilon_n])(\epsilon_1 - \mathbb{E}[\epsilon_1]) \\ (\epsilon_1 - \mathbb{E}[\epsilon_1])(\epsilon_2 - \mathbb{E}[\epsilon_2]) & (\epsilon_2 - \mathbb{E}[\epsilon_2])^2 & \cdots & (\epsilon_n - \mathbb{E}[\epsilon_n])(\epsilon_2 - \mathbb{E}[\epsilon_2]) \\ \vdots & \vdots & \ddots & \vdots \\ (\epsilon_n - \mathbb{E}[\epsilon_n])(\epsilon_1 - \mathbb{E}[\epsilon_1]) & (\epsilon_n - \mathbb{E}[\epsilon_n])(\epsilon_2 - \mathbb{E}[\epsilon_2]) & \cdots & (\epsilon_n - \mathbb{E}[\epsilon_n])^2 \end{bmatrix}$$

Analisando as hipteses:

1. Resduos no correlacionados:

   Seja $i, j \in \{1, \dots, n\}$, $i \neq j$, e tome a expresso:

   $$\mathbb{E}\left[(\epsilon_i - \mathbb{E}[\epsilon_i])(\epsilon_j - \mathbb{E}[\epsilon_j])\right] = \mathbb{E}[\epsilon_i\epsilon_j - \mathbb{E}[\epsilon_i]\epsilon_j - \mathbb{E}[\epsilon_j]\epsilon_i + \mathbb{E}[\epsilon_i]\mathbb{E}[\epsilon_j]]$$

   Por hiptese, temos que $\mathbb{E}[\epsilon] = 0 \implies \mathbb{E}[\epsilon_i] = 0, \forall i \in \{1, \dots, n\}$.

   Assim:

   $$\mathbb{E}\left[(\epsilon_i - \mathbb{E}[\epsilon_i])(\epsilon_j - \mathbb{E}[\epsilon_j])\right] = \mathbb{E}[\epsilon_i\epsilon_j]$$

   Portanto $b_{ij} = b_{ji} = (\epsilon_i - \mathbb{E}[\epsilon_i])(\epsilon_j - \mathbb{E}[\epsilon_j]) = \epsilon_i\epsilon_j$.

   Logo, os elementos Σ_{ij} da matriz $\Sigma_{n\times n} = \mathbb{E}[B_{n\times n}]$, $i \neq j$, sero da forma $\mathbb{E}[\epsilon_i\epsilon_j]$.

   Assim, a hiptese $\mathbb{E}[\epsilon_i\epsilon_j] = 0, \forall i \neq j$, anula todos os entradas da matriz Σ, exceto sua diagonal principal.

2. Homoscedasticidade

   Seja $V(\epsilon_i) = V(\epsilon_j) = \sigma^2, \forall i, j \in \{1, \dots, n\}$.

   Tendo em vista os clculos j feitos, a diagonal principal da matriz Σ ter todos os seus elementos dados por σ^2.

Por fim, conclumos que, dado que as hipteses 1 e 2, Σ vale:

$$\Sigma = \sigma^2 I_{n\times n}, \quad \text{onde } V(\epsilon_i) = \sigma^2, \forall i \in \{1, \dots, n\}$$

7 Exercicio 2b

(i) Seja $\hat{\beta}_\Sigma = \arg\min_\beta (y - X\beta)^\top \Sigma^{-1}(y - X\beta) = \arg\min_\beta G(\beta)$. Tomando:

$$\frac{\partial G(\beta)}{\partial \beta} = \frac{\partial}{\partial \beta}[(y - X\beta)^\top \Sigma^{-1}(y - X\beta)] = \frac{\partial}{\partial \beta}\left[y^\top \Sigma^{-1} y - 2y^\top \Sigma^{-1} X\beta + \beta^\top X^\top \Sigma^{-1} X\beta\right],$$

Conclumos que:

$$\frac{\partial G(\beta)}{\partial \beta} = -2y^\top \Sigma^{-1} X + 2\beta^\top X^\top \Sigma^{-1} X$$

Como a funo convexa (mostrado em *), para β ser tal que minimize $G(\beta)$, ele deve ser soluo da seguinte equao:

$$-2y^\top \Sigma^{-1} X + 2\beta^\top X^\top \Sigma^{-1} X = 0,$$

o que implica em:

$$y^\top \Sigma^{-1} X = \beta^\top X^\top \Sigma^{-1} X \implies \quad \hat{\beta_\Sigma} = (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} y.$$

(*): Mostraremos que a funo G convexa a partir de sua segunda derivada:

$$\frac{\partial^2 G(\beta)}{\partial \beta^2} = \frac{\partial}{\partial \beta}\left[-2y^\top \Sigma^{-1} X + 2\beta^\top X^\top \Sigma^{-1} X\right] = 2(X^\top \Sigma^{-1} X).$$

Como $X^\top \Sigma^{-1} X$ positiva semi-definida, G convexo e o β calculado um ponto de mnimo global de G.

(ii) Primeiramente, suponha que X no seja uma varivel aleatria neste contexto analisado. Alm disso, note que β no varivel aleatria, mas sim uma matriz fixa que gera os valores amostrais isentos de erros. Assim, desenvolvendo o valor esperado de $\hat{\beta}_\Sigma$:

$$\mathbb{E}[\hat{\beta}_\Sigma] = \mathbb{E}\left[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} y\right] = \mathbb{E}\left[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1}(X\beta + \epsilon)\right].$$

$$= \mathbb{E}\left[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} X\beta + (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \epsilon\right].$$

$$= \mathbb{E}\left[I\beta + (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \epsilon\right].$$

$$= \beta + \mathbb{E}\left[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \epsilon\right]$$

Como X no se trata de uma varivel aleatria neste contexto, temos que:

$$\mathbb{E}\left[(X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \epsilon\right] = (X^\top \Sigma^{-1} X)^{-1} X^\top \Sigma^{-1} \mathbb{E}[\epsilon] = 0.$$

Logo:

$$\mathbb{E}[\hat{\beta}_\Sigma] = \beta.$$

(iii) Por motivos anlogos ao item anterior, note que beta no varivel aleatria, mas sim uma matriz fixa. Alm disso, como estamos analisando a varincia dado X, podemos assumi-lo como fixo tambm (apesar de que, no contexto geral do problema, X j tido como fixo).

$$V[\hat{\beta}_{\Sigma} \mid X] = V\left[(X^{\top}\Sigma^{-1}X)^{-1}X^{\top}\Sigma^{-1}(X\beta + \epsilon) \mid X\right].$$

Seja $M = (X^{\top}\Sigma^{-1}X)^{-1}X^{\top}\Sigma^{-1}$. Assim:

$$V[\hat{\beta}_Z \mid X] = V[\beta + M\epsilon \mid X] = MV[\epsilon]M^{\top} = M\Sigma M^{\top}$$

Substituindo M na expresso acima, obtemos:

$$V[\hat{\beta}_Z \mid X] = (X^{\top}\Sigma^{-1}X)^{-1}X^{\top}\Sigma^{-1}\Sigma\Sigma^{-1}X(X^{\top}\Sigma^{-1}X)^{-1}.$$

Simplificando:

$$V[\hat{\beta}_Z \mid X] = (X^{\top}\Sigma^{-1}X)^{-1}.$$

(iv) Tome $(X^{\top}\Sigma^{-1}X)^{-1}X^{\top}\Sigma^{-1}\epsilon = \alpha$. Note que, condicionado a X, ϵ e α seguem distribuies com a mesma natureza (ou seja, α tambm segue uma distribuio normal), afinal a matriz $(X^{\top}\Sigma^{-1}X)^{-1}X^{\top}\Sigma^{-1}$ no se trata de uma varivel aleatria (seu valor fixo e invarivel em funo de reamostragens) e portanto a forma da curva de distribuio de alfa determinado pela nica varivel aleatria relacionada a ela, ϵ. Seja tambm $\hat{\beta}_{\Sigma} = \alpha + \beta$. De maneira anloga a forma da curva da distribuio de $\hat{\beta}_{\Sigma}$ determinada apenas por α visto que β no uma varivel aleatria. Assim, $\hat{\beta}_{\Sigma}$ segue uma distribuio $N(\mu, \sigma^2)$. Como j calculado, $E[\hat{\beta}_{\Sigma}] = \beta$ e $V[\hat{\beta}_Z \mid X] = (X^{\top}\Sigma^{-1}X)^{-1}$. Logo, $\hat{\beta}_{\Sigma}$ segue a distribuio $N(\beta, (X^{\top}\Sigma^{-1}X)^{-1})$

8 Exercicio 3a

Insira sua soluo aqui, incluindo qualquer conta, cdigo ou figura relevante para a sua soluo.

9 Exercicio 3b

Insira sua soluo aqui, incluindo qualquer conta, cdigo ou figura relevante para a sua soluo.

10 Exercicio 3c

Insira sua soluo aqui, incluindo qualquer conta, cdigo ou figura relevante para a sua soluo.

11 Exercicio 3d

Insira sua soluo aqui, incluindo qualquer conta, cdigo ou figura relevante para a sua soluo.

12 Exercicio 3e

Insira sua soluo aqui, incluindo qualquer conta, cdigo ou figura relevante para a sua soluo.

13 Exercicio 4a

Insira sua soluo aqui, incluindo qualquer conta, cdigo ou figura relevante para a sua soluo.

14 Exercicio 4b

Insira sua soluo aqui, incluindo qualquer conta, cdigo ou figura relevante para a sua soluo.

15 Exercicio 5a

necessrio normalizar as features pois assim evitaremos que a escala e redimensionamentos interfiram na classificao.

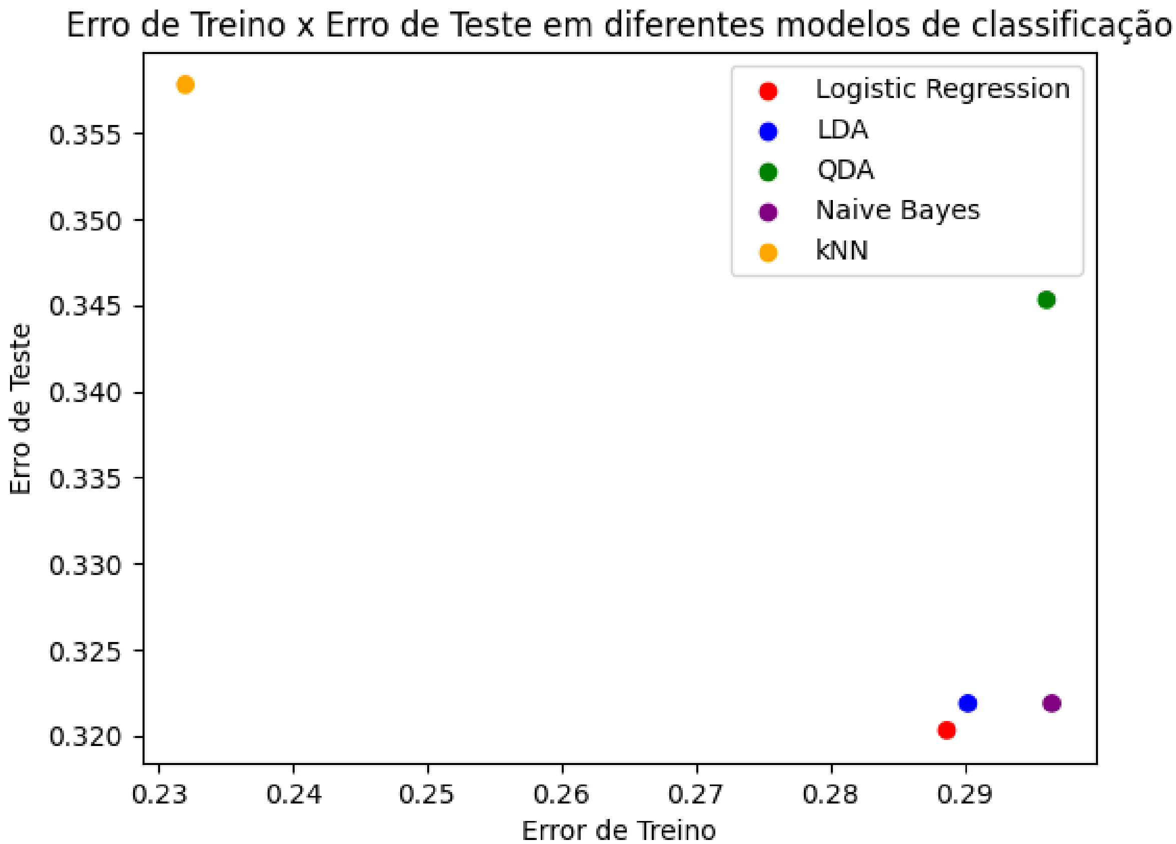
16 Exercicio 5b

```
# Regress o Log stica
lr = LR()
fitted_lr = lr.fit(X_train, y_train.values.ravel())

y_pred_train_lr = fitted_lr.predict(X_train)
y_pred_test_lr = fitted_lr.predict(X_test)

# LDA
lda = LDA()
fitted_lda = lda.fit(X_train, y_train.values.ravel())

y_pred_train_lda = fitted_lda.predict(X_train)
y_pred_test_lda = fitted_lda.predict(X_test)

# QDA
qda = QDA()
fitted_qda = qda.fit(X_train, y_train.values.ravel())

y_pred_train_qda = fitted_qda.predict(X_train)
y_pred_test_qda = fitted_qda.predict(X_test)

# Naive Bayes
nb = NB()
fitted_nb = nb.fit(X_train, y_train.values.ravel())

y_pred_train_nb = fitted_nb.predict(X_train)
y_pred_test_nb = fitted_nb.predict(X_test)

# KNN com k=5
knn = kNN(n_neighbors=5)
fitted_knn = knn.fit(X_train, y_train.values.ravel())

y_pred_train_knn = fitted_knn.predict(X_train)
y_pred_test_knn = fitted_knn.predict(X_test)
```

17 Exercicio 5c

Em ordem, o cdigo utilizado e a imagem gerada a partir dele.

```
color_list = ['red', 'blue', 'green', 'purple', 'orange']

taxa_erro_treino_lr = (y_pred_train_lr != y_train.values.ravel()).sum() / len(
                                          y_train)
taxa_erro_teste_lr = (y_pred_test_lr != y_test.values.ravel()).sum() / len(y_test)

taxa_erro_treino_lda = (y_pred_train_lda != y_train.values.ravel()).sum() / len(
                                          y_train)
taxa_erro_teste_lda = (y_pred_test_lda != y_test.values.ravel()).sum() / len(
                                          y_test)

taxa_erro_treino_qda = (y_pred_train_qda != y_train.values.ravel()).sum() / len(
                                          y_train)
taxa_erro_teste_qda = (y_pred_test_qda != y_test.values.ravel()).sum() / len(
                                          y_test)

taxa_erro_treino_nb = (y_pred_train_nb != y_train.values.ravel()).sum() / len(
                                          y_train)
taxa_erro_teste_nb = (y_pred_test_nb != y_test.values.ravel()).sum() / len(y_test)

taxa_erro_treino_knn = (y_pred_train_knn != y_train.values.ravel()).sum() / len(
                                          y_train)
taxa_erro_teste_knn = (y_pred_test_knn != y_test.values.ravel()).sum() / len(
                                          y_test)

plt.scatter(
    [taxa_erro_treino_lr, taxa_erro_treino_lda, taxa_erro_treino_qda,
                                              taxa_erro_treino_nb,
                                              taxa_erro_treino_knn],
    [taxa_erro_teste_lr, taxa_erro_teste_lda, taxa_erro_teste_qda,
                                              taxa_erro_teste_nb,
                                              taxa_erro_teste_knn],
    c=color_list
)
for i, model in enumerate(['Logistic Regression', 'LDA', 'QDA', 'Naive Bayes', '
                                          kNN']):
    plt.scatter([], [], c=color_list[i], label=model)

plt.xlabel('Error de Treino')
plt.ylabel('Erro de Teste')
plt.title('Erro de Treino x Erro de Teste em diferentes modelos de classifica o
                                          ')
plt.legend()
plt.show()
```



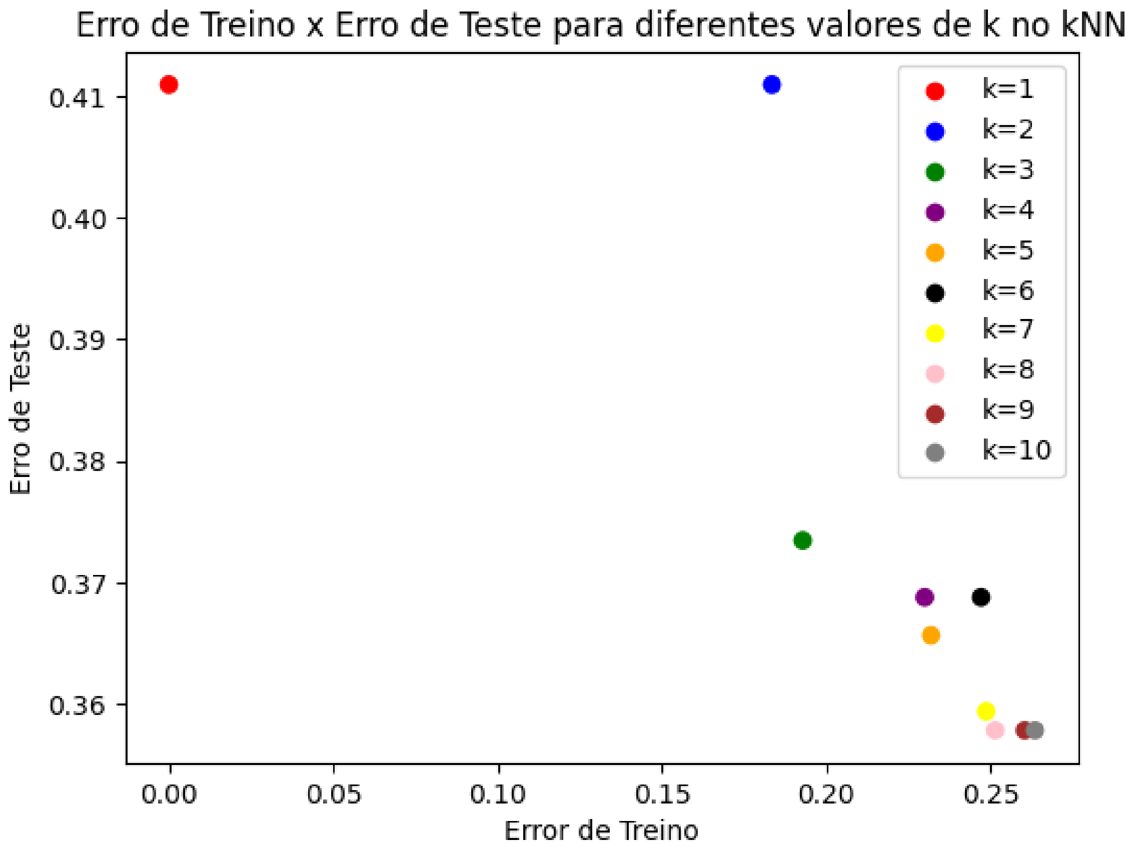
Erro de Treino x Erro de Teste em diferentes modelos de classificação
Logistic Regression
LDA
QDA
Naive Bayes
kNN
Erro de Teste
0.355
0.350
0.345
0.340
0.335
0.330
0.325
0.320
0.23
0.24
0.25
0.26
0.27
0.28
0.29
Error de Treino

18 Exercicio 5d

De acordo com o grafico, podemos notar que o erro de treino decresce junto ao valor de k. Ou seja, modelos com k menor tendem a fittar melhor os dados de treino.
Contudo, possvel observar que a diminuio do valor de k tende a aumentar o erro de teste, o que indica que essa diminuio est gerando overfitting no modelo.

Em ordem, o cdigo utilizado e a imagem gerada a partir dele.

```
## Multiplos KNN

color_list = ['red', 'blue', 'green', 'purple', 'orange', 'black', 'yellow', 'pink
                                    ', 'brown', 'gray']

error_list = []
for i in range(1, 11):
    knn = kNN(n_neighbors=i)

    fitted_knn = knn.fit(X_train, y_train.values.ravel())

    y_pred_treino_knn = fitted_knn.predict(X_train)
    y_pred_test_knn = fitted_knn.predict(X_test)

    error_treino = (y_pred_treino_knn != y_train.values.ravel()).sum() / len(
                                        y_train)
    erro_teste = (y_pred_test_knn != y_test.values.ravel()).sum() / len(y_test)

    error_list.append((error_treino, erro_teste))

plt.scatter(
    x=[error[0] for error in error_list],
    y=[error[1] for error in error_list],
    c=color_list
)
for i, k in enumerate(range(1, 11)):
    plt.scatter([], [], c=color_list[i], label=f'k={k}')

plt.xlabel('Error de Treino')
plt.ylabel('Erro de Teste')
plt.title('Erro de Treino x Erro de Teste para diferentes valores de k no kNN')
plt.legend()
plt.show()
```


Erro de Treino x Erro de Teste para diferentes valores de k no kNN
Erro de Teste
Error de Treino
0.41
0.40
0.39
0.38
0.37
0.36
0.00
0.05
0.10
0.15
0.20
0.25
k=1
k=2
k=3
k=4
k=5
k=6
k=7
k=8
k=9
k=10

19 Exercicio 6a

O modelo de lasso requer normalizao previa dos dados pois tal modelo penaliza os coeficientes de acordo com o quo alto so seus mdulos e, portanto, pode erroneamente penalizar alguns coeficientes devido a natureza da feature associada a eles.
Por exemplo, features com o mdulo da media alto (em relao as outras features do modelo) tendem a ter uma varincia maior e um mdulo maior. Portanto, o coeficiente associado a elas tende a ser mais baixo. Logo, tal coeficiente ser pouco afetado por penalizaes do modelo de lasso devido a natureza de sua feature. Para evitar isso, normalizamos essa feature subtraindo-a de sua media e dividindo-a por sua varincia.

20 Exercicio 6b

Abaixo se encontram os cdigos utilizados para realizar a seleo a partir de cada mtodo pedido.

Best subset selection

```
from itertools import combinations

def get_all_possible_subsets(n, features):
    return list(combinations(features, n))

## Best subset selection
overall_best_model = None
overall_max_coef = 0
best_model_by_numb_pred_bs = {}
for numb_pred in range(1, 14):

    # Get all possible subsets and remove duplicates
    possibilities = get_all_possible_subsets(numb_pred, X_train.columns)
    possibilities = [tuple(sorted(possibility)) for possibility in possibilities]
    possibilities = list(set(possibilities))

    possibilit_max_coef = 0
    for possibilit in possibilities:
        r2_coef = 0
        # We perform a 5-fold cross validation
        for fold_index in range(0, 5):
            x_fold_train = X_train.loc[cv_fold != fold_index, possibilit]
            y_fold_train = y_train.loc[cv_fold != fold_index]
            fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()

            y_fold_validation = y_train.loc[cv_fold == fold_index]
            x_fold_validation = X_train.loc[cv_fold == fold_index, possibilit]

            y_fold_validation_pred = fitted_lm.predict(x_fold_validation)

            # R2 coefficient on the validation set
            r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) **
                                                    2) / np.sum((
                                                    y_fold_validation - np.mean(
                                                    y_fold_validation)) ** 2)

        # avg of the r2 coefficient on the 5 folds for this subset of features
        avg_r2_coef = r2_coef / 5

        # If the model with this subset of features is better than the previous
                                                best model, we save its features
                                                and update the best r2
                                                coefficient
        if possibilit_max_coef < avg_r2_coef:
            possibilit_max_coef = avg_r2_coef
            best_subset = possibilit

    best_model_by_numb_pred_bs[numb_pred] = {
        "features": best_subset,
        "r2coef": possibilit_max_coef
```

```
    }

    # If the model with this number of features is better than the previous best
                                                  model, we save its features and
                                                  update the best r2 coefficient
    if overall_max_coef < possibilit_max_coef:
        overall_max_coef = possibilit_max_coef
        overall_best_model = best_subset
```

Melhor conjunto de features encontrado:
Conjunto: ['Abdomen', 'Biceps', 'Forearm', 'Height', 'Hip', 'Neck', 'Thigh', 'Weight', 'Wrist']
R^2: 0.7039921033347978

Forward stepwise selection:

```
# Forward stepwise selection
overall_best_model_forward = None
overall_max_coef_forward = 0
best_model_by_numb_pred_forward = {}
previous_stage_best_features = []

for numb_pred_forward in range(1, 14):
    available_features = set(X_train.columns) - set(previous_stage_best_features)
    possibilit_max_coef = 0
    for feature in available_features:
        # We add a new feature to the previous best subset of features to test the
                                                      model with this new subset
        new_features = previous_stage_best_features + [feature]
        r2_coef = 0
        # We perform a 5-fold cross validation
        for fold_index in range(0, 5):
            x_fold_train = X_train.loc[cv_fold != fold_index, new_features]
            y_fold_train = y_train.loc[cv_fold != fold_index]
            fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()

            y_fold_validation = y_train.loc[cv_fold == fold_index]
            x_fold_validation = X_train.loc[cv_fold == fold_index, new_features]

            y_fold_validation_pred = fitted_lm.predict(x_fold_validation)

            r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) **
                                                     2) / np.sum((
                                                     y_fold_validation - np.mean(
                                                     y_fold_validation)) ** 2)
        avg_r2_coef = r2_coef / 5

        # If the model with this subset of features is better than the previous
                                                     best model, we save its features
                                                     and update the best r2
                                                     coefficient
        if possibilit_max_coef < avg_r2_coef:
            possibilit_max_coef = avg_r2_coef
            best_subset = new_features
```

```python
    # Save the best model for this number of features
    best_model_by_numb_pred_forward[numb_pred_forward] = {
        "features": best_subset,
        "r2coef": possibilit_max_coef
    }
    previous_stage_best_features = best_subset

    # Save the best overall model
    if overall_max_coef_forward < possibilit_max_coef:
        overall_max_coef_forward = possibilit_max_coef
        overall_best_model_forward = best_subset
```

Melhor conjunto de features encontrado:
Conjunto: ['Abdomen', 'Wrist', 'Hip', 'Forearm', 'Neck', 'Chest', 'Height', 'Knee', 'Weight', 'Biceps', 'Thigh']
R^2: 0.7004284501872192

Backward stepwise selection

```python
# Backward stepwise selection
overall_best_model_backward = None
overall_max_coef_backward = 0
best_model_by_numb_pred_backward = {}
previous_stage_best_features = list(X_train.columns)


# Include the case where no variable is removed
r2_coef = 0
for fold_index in range(0, 5):
    x_fold_train = X_train.loc[cv_fold != fold_index]
    y_fold_train = y_train.loc[cv_fold != fold_index]
    fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()

    y_fold_validation = y_train.loc[cv_fold == fold_index]
    x_fold_validation = X_train.loc[cv_fold == fold_index]

    y_fold_validation_pred = fitted_lm.predict(x_fold_validation)

    r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) ** 2) / np.
                                              sum((y_fold_validation - np.mean(
                                              y_fold_validation)) ** 2)

best_model_by_numb_pred_backward[13] = {
    "features": list(X_train.columns),
    "r2coef": r2_coef / 5
}

for numb_removed_pred_backward in range(1, 13):

    available_features = set(previous_stage_best_features)

    possibilit_max_coef = 0
    for feature in available_features:
        # We remove a feature from the previous best subset of features to test
```

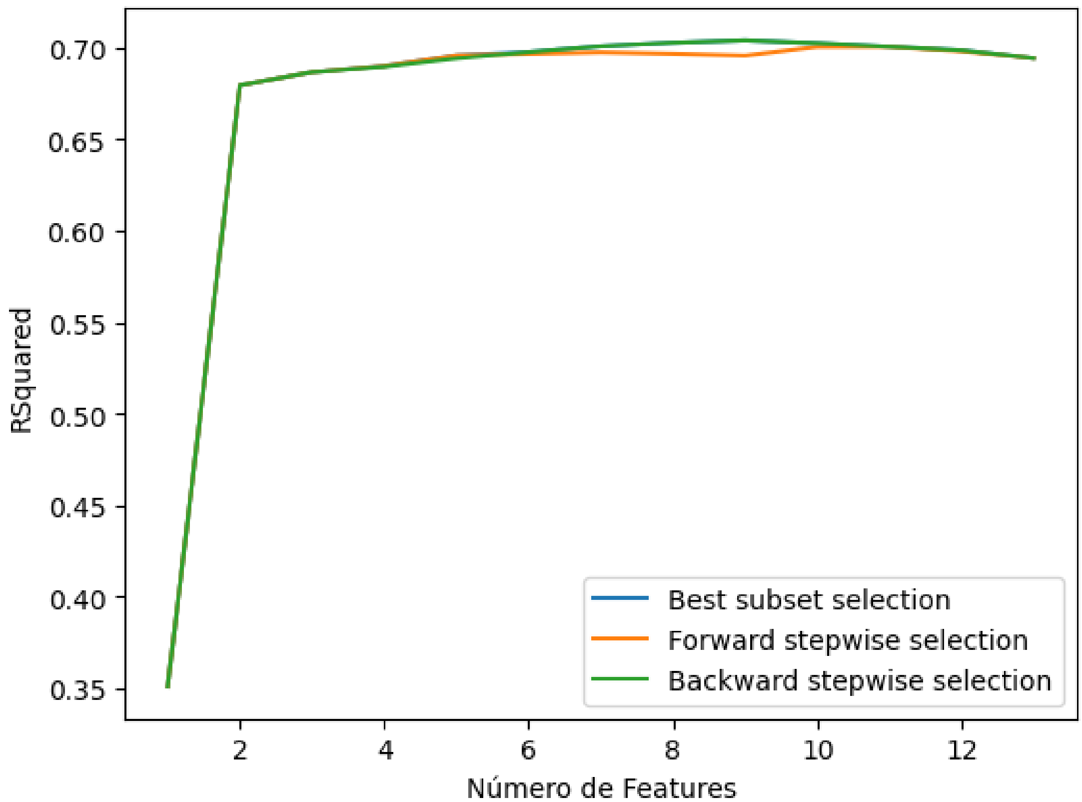
```python
                                                      the model with this new subset
        new_features = list(set(previous_stage_best_features) - set([feature]))
        r2_coef = 0
        # We perform a 5-fold cross validation
        for fold_index in range(0, 5):
            x_fold_train = X_train.loc[cv_fold != fold_index, new_features]
            y_fold_train = y_train.loc[cv_fold != fold_index]
            fitted_lm = sm.OLS(y_fold_train, x_fold_train).fit()

            y_fold_validation = y_train.loc[cv_fold == fold_index]
            x_fold_validation = X_train.loc[cv_fold == fold_index, new_features]

            y_fold_validation_pred = fitted_lm.predict(x_fold_validation)

            r2_coef += 1 - np.sum((y_fold_validation - y_fold_validation_pred) **
                                                          2) / np.sum((
                                                          y_fold_validation - np.mean(
                                                          y_fold_validation)) ** 2)
        avg_r2_coef = r2_coef / 5

        # If the model with this subset of features is better than the previous
                                                      best model, we save its features
                                                      and update the best r2
                                                      coefficient
        if possibilit_max_coef < avg_r2_coef:
            possibilit_max_coef = avg_r2_coef
            best_subset = new_features

    # Save the best model for this number of features
    best_model_by_numb_pred_backward[13 - numb_removed_pred_backward] = {
        "features": best_subset,
        "r2coef": possibilit_max_coef
    }
    previous_stage_best_features = best_subset

    # Save the best overall model
    if overall_max_coef_backward < possibilit_max_coef:
        overall_max_coef_backward = possibilit_max_coef
        overall_best_model_backward = best_subset
```

Melhor conjunto de features encontrado:
Conjunto: ['Weight', 'Wrist', 'Forearm', 'Hip', 'Neck', 'Thigh', 'Height', 'Biceps', 'Abdomen']
R^2: 0.7039921033348018

21 Exercicio 6c

A seguir, cdigo usado e o grfico gerado por ele:

```
plt.plot(
    list(best_model_by_numb_pred_bs.keys()),
    [best_model_by_numb_pred_bs[key]["r2coef"] for key in
                                          best_model_by_numb_pred_bs.keys()],
    label="Best subset selection"
)

plt.plot(
    list(best_model_by_numb_pred_forward.keys()),
    [best_model_by_numb_pred_forward[key]["r2coef"] for key in
                                          best_model_by_numb_pred_forward.keys
                                          ()],
    label="Forward stepwise selection"
)

plt.plot(
    list(best_model_by_numb_pred_backward.keys()),
    [best_model_by_numb_pred_backward[key]["r2coef"] for key in
                                          best_model_by_numb_pred_backward.keys
                                          ()],
    label="Backward stepwise selection"
)

plt.xlabel('Number of Predictors')
plt.ylabel('R2 Coefficient')
plt.legend()
plt.show()
```


RSquared
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
2
4
6
8
10
12
Número de Features
Best subset selection
Forward stepwise selection
Backward stepwise selection