Pedro Henrique Andrade Trindade

[2]:
```
import pandas as pd
"""
Código para ler o csv do dataset
"""
column_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width',␣
 ↪'class']

iris_data = pd.read_csv('iris.data', header=None, names=column_names)

print(iris_data.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width        class
0           5.1          3.5           1.4          0.2  Iris-setosa
1           4.9          3.0           1.4          0.2  Iris-setosa
2           4.7          3.2           1.3          0.2  Iris-setosa
3           4.6          3.1           1.5          0.2  Iris-setosa
4           5.0          3.6           1.4          0.2  Iris-setosa
```

Questão 1: Considerando cada uma das duas técnicas descritas acima, encontre três classificadores binários, cada um classificando uma espécie contra as outras duas (logo, você fará seis classificadores no total, três por QM e outros três por SVM). Forneça a taxa de erro de cada um destes classificadores, tanto no conjunto de treinamento quanto no conjunto de teste.

Resposta: Segue o código de implementação do classificador linear implementado com Mínimo Quadrados e também utilizando a estratégia do SVM. Os erros para cada caso estão sendo printados em códigos posteriores abaixo.

[3]:
```
import cvxpy as cp
import numpy as np
import pandas as pd

def linear_least_squares_classifier(df, target_class):
    # Vamos preparar os dados com features e labels
    X = df.iloc[:, :-1].values  # Pegamos todas as colunas menos a última
    y = np.where(df['class'] == target_class, 1, -1)  # 1 para a classe de␣
 ↪interesse, -1 do contrário

    # número de features no dataset
    n_features = X.shape[1]
```

```
    # Vamos resolver o problema com o cvxpy
    a = cp.Variable(n_features)  # Vetor de pesos
    b = cp.Variable()            # termo de bias

    # Aqui é a definição dos mínimos quadrados
    objective = cp.Minimize(cp.sum_squares(X @ a + b - y))
    problem = cp.Problem(objective)

    # Então resolvemos o problema
    problem.solve()

    return a.value, b.value
```

[4]:
```
"""
    Apenas um código demonstrando o treinamento e pesos sendo gerados
"""
iris_data = pd.read_csv('iris.data', header=None, names=['sepal_length',␣
 ↪'sepal_width', 'petal_length', 'petal_width', 'class'])
a, b = linear_least_squares_classifier(iris_data, 'Iris-setosa')

print("Weights (a):", a)
print("Bias (b):", b)
```

```
Weights (a): [ 0.1312861   0.48494601 -0.44552275 -0.12670283]
Bias (b): -0.7550609184556691
```

[14]:
```
"""
    Código utilitário para ajudar a separar teste/validação
    e calcular o erro
"""
def train_test_split_pandas(df, test_size=0.2, random_state=11):
    if random_state is not None:
        np.random.seed(random_state)

    # Vamos embaralhar o dataset
    df_shuffled = df.sample(frac=1, random_state=random_state).
 ↪reset_index(drop=True)

    # Separa em treino / validação
    n = len(df_shuffled)
    n_val = int(n * test_size)

    df_val = df_shuffled.iloc[:n_val]
    df_train = df_shuffled.iloc[n_val:]

    return df_train, df_val
```

```
def calculate_classification_error(y_true, y_pred):
    return np.mean(y_true != y_pred) * 100
```

[36]:
```
iris_types = ['Iris-setosa','Iris-versicolor', 'Iris-virginica']

iris_data = pd.read_csv('iris.data', header=None,
                        names=['sepal_length', 'sepal_width',
                               'petal_length', 'petal_width', 'class'])

df_train, df_val = train_test_split_pandas(iris_data, test_size=1/3)

for iris_type in iris_types:
       print(f"\nQM Para o tipo {iris_type}")
       a, b = linear_least_squares_classifier(df_train, target_class=iris_type)

       X_train = df_train.iloc[:, :-1].values
       y_train_true = np.where(df_train['class'] == iris_type, 1, -1)

       # Aqui é a predição do modelo para o conjunto de treino
       y_train_pred = np.sign(X_train @ a + b)
       error = calculate_classification_error(y_train_true, y_train_pred)
       print(f"Erro de validação para conjunto de treinamento {iris_type}:␣
 ↪{error:.2f}%")

       X_val = df_val.iloc[:, :-1].values
       y_val_true = np.where(df_val['class'] == iris_type, 1, -1)

       # Aqui é a predição do modelo para o conjunto de teste
       y_val_pred = np.sign(X_val @ a + b)

       error = calculate_classification_error(y_val_true, y_val_pred)
       print(f"Erro de validação para conjunto de teste {iris_type}: {error:.
 ↪2f}%")
```

```
QM Para o tipo Iris-setosa
Erro de validação para conjunto de treinamento Iris-setosa: 0.00%
Erro de validação para conjunto de teste Iris-setosa: 0.00%

QM Para o tipo Iris-versicolor
Erro de validação para conjunto de treinamento Iris-versicolor: 24.00%
Erro de validação para conjunto de teste Iris-versicolor: 30.00%

QM Para o tipo Iris-virginica
Erro de validação para conjunto de treinamento Iris-virginica: 6.00%
Erro de validação para conjunto de teste Iris-virginica: 10.00%
```

[16]:
```
def svm_classifier(df, target_class):
    # Basicamente o mesmo processo do classificador anterior
    X = df.iloc[:, :-1].values
    y = np.where(df['class'] == target_class, 1, -1)

    _, n_features = X.shape

    a = cp.Variable(n_features)
    b = cp.Variable()

    # aqui a gente muda a função objetivo em relação ao classificador anterior
    hinge_loss = cp.sum(cp.pos(1 - cp.multiply(y, X @ a + b)))

    problem = cp.Problem(cp.Minimize(hinge_loss))
    problem.solve()

    return a.value, b.value
```

[37]:
```
"""
   Código que printa os erros para o classificador SVM
"""
iris_types = ['Iris-setosa','Iris-versicolor', 'Iris-virginica']

iris_data = pd.read_csv('iris.data', header=None,
                        names=['sepal_length', 'sepal_width',
                               'petal_length', 'petal_width', 'class'])

df_train, df_val = train_test_split_pandas(iris_data, test_size=1/3)

for iris_type in iris_types:
       print(f"\nSVM Para o tipo {iris_type}")
       a, b = svm_classifier(df_train, target_class=iris_type)

       X_train = df_train.iloc[:, :-1].values
       y_train_true = np.where(df_train['class'] == iris_type, 1, -1)

       y_train_pred = np.sign(X_train @ a + b)
       error = calculate_classification_error(y_train_true, y_train_pred)
       print(f"Erro de validação para conjunto de treinamento {iris_type}:␣
 ↪{error:.2f}%")

       X_val = df_val.iloc[:, :-1].values
       y_val_true = np.where(df_val['class'] == iris_type, 1, -1)

       y_val_pred = np.sign(X_val @ a + b)

       error = calculate_classification_error(y_val_true, y_val_pred)
```

```
        print(f"Erro de validação para conjunto de teste {iris_type}: {error:.
↪2f}%")
```

```
SVM Para o tipo Iris-setosa
Erro de validação para conjunto de treinamento Iris-setosa: 0.00%
Erro de validação para conjunto de teste Iris-setosa: 0.00%

SVM Para o tipo Iris-versicolor
Erro de validação para conjunto de treinamento Iris-versicolor: 24.00%
Erro de validação para conjunto de teste Iris-versicolor: 32.00%

SVM Para o tipo Iris-virginica
Erro de validação para conjunto de treinamento Iris-virginica: 2.00%
Erro de validação para conjunto de teste Iris-virginica: 4.00%
```

Questão 2: Combine os classificadores desenvolvidos acima para obter dois classificadores de 3 classes, um desenvol- vido por quadrados mínimos e o segundo por otimizaçao linear, e forneça a matriz de confusão para os conjuntos de treinamento e de teste.

[22]:
```
def train_linear_multiclass_classifier(df):
    classes = df['class'].unique() # Lista de classes únicas
    classifiers = {}

    for target_class in classes:
        # Treino de um classificador linear para cada classe
        a, b = linear_least_squares_classifier(df, target_class)
        classifiers[target_class] = (a, b)

    return classifiers
```

[23]:
```
def predict_multiclass(X, classifiers):
    scores = {}
    for class_name, (a, b) in classifiers.items():
        scores[class_name] = X @ a + b  # Score de decisão para cada classe

    # seleciona a classe com maior score
    return np.array([max(scores, key=lambda k: scores[k][i]) for i in␣
↪range(len(X))])
```

[24]:
```
def train_svm_multiclass_classifier(df):
    classes = df['class'].unique()
    classifiers = {}

    for target_class in classes:
        a, b = svm_classifier(df, target_class)
        classifiers[target_class] = (a, b)
```

```
    return classifiers
```

[27]:
```
"""
   Código para simples teste do erro de validação para o SVM
   e QM linear multiclasse.
"""
iris_data = pd.read_csv('iris.data', header=None,
                        names=['sepal_length', 'sepal_width',
                               'petal_length', 'petal_width', 'class'])
df_train, df_val = train_test_split_pandas(iris_data, test_size=1/3,␣
 ↪random_state=42)

classifiers = train_linear_multiclass_classifier(df_train)

X_val = df_val.iloc[:, :-1].values
y_val_pred = predict_multiclass(X_val, classifiers)

y_val_true = df_val['class'].values
error = calculate_classification_error(y_val_true, y_val_pred)
print(f"Erro de validação para o QM: {error:.2f}%")

classifiers = train_svm_multiclass_classifier(df_train)
y_val_pred = predict_multiclass(X_val, classifiers)

error = calculate_classification_error(y_val_true, y_val_pred)
print(f"Erro de validação para o SVM: {error:.2f}%")
```

```
Erro de validação para o QM: 20.00%
Erro de validação para o SVM: 4.00%
```

[29]:
```
# Código para computar a matriz de confusão
def compute_confusion_matrix(classifiers, df_val):
    X_val = df_val.iloc[:, :-1].values
    y_true = df_val['class'].values
    y_pred = predict_multiclass(X_val, classifiers)

    classes = sorted(np.unique(y_true))
    cm = np.zeros((len(classes), len(classes)), dtype=int)

    for i, true_class in enumerate(classes):
        for j, pred_class in enumerate(classes):
            cm[i, j] = np.sum((y_true == true_class) & (y_pred == pred_class))

    return pd.DataFrame(cm, index=classes, columns=classes)
```

[33]:
```
"""
   Código para calcular matriz de confusão do QM e do SVM
```

```
"""
iris_data = pd.read_csv('iris.data', header=None,
                        names=['sepal_length', 'sepal_width',
                               'petal_length', 'petal_width', 'class'])

df_train, df_val = train_test_split_pandas(iris_data, test_size=1/3)

classifiers = train_linear_multiclass_classifier(df_train)

cm = compute_confusion_matrix(classifiers, df_train)
print("Matriz de confusão para o dataset de treino do QM:")
print(cm)
cm = compute_confusion_matrix(classifiers, df_val)
print("Matriz de confusão para o dataset de validação do QM:")
print(cm)

classifiers = train_svm_multiclass_classifier(df_train)
print("\n------------------------------------------------------------")
cm = compute_confusion_matrix(classifiers, df_train)
print("Matriz de confusão para o dataset de treino do SVM:")
print(cm)
cm = compute_confusion_matrix(classifiers, df_val)
print("Matriz de confusão para o dataset de validação do SVM:")
print(cm)
```

```
Matriz de confusão para o dataset de treino do QM:
                 Iris-setosa  Iris-versicolor  Iris-virginica
Iris-setosa               35                0               0
Iris-versicolor            0               20              11
Iris-virginica             0                2              32
Matriz de confusão para o dataset de validação do QM:
                 Iris-setosa  Iris-versicolor  Iris-virginica
Iris-setosa               14                1               0
Iris-versicolor            0                9              10
Iris-virginica             0                1              15

------------------------------------------------------------
Matriz de confusão para o dataset de treino do SVM:
                 Iris-setosa  Iris-versicolor  Iris-virginica
Iris-setosa               35                0               0
Iris-versicolor            0               29               2
Iris-virginica             0                2              32
Matriz de confusão para o dataset de validação do SVM:
                 Iris-setosa  Iris-versicolor  Iris-virginica
Iris-setosa               14                1               0
Iris-versicolor            2               15               2
Iris-virginica             0                0              16
```

Questão 3: Compare os resultados obtidos nos itens anteriores. Qual classificador apresentou melhor desempenho, em geral?

Resposta:

Notamos pelos experimentos que a performance geral do classificador multiclasse com SVM foi melhor do que o multiclasse com o QM, o que é um resultado interessante, dado que individualmente a taxa de erro dos dois tipos de classificadores foram próximas.

Mas o SVM nesse contexto se mostrou melhor em separar as Iris do tipo Iris-virginica, melhorando a fronteira de decisão, diminuindo o erro de classificação entre Iris-versicolor e Iris-virginica, por exemplo.