Experimentação Regressão Logística

Pedro Henrique Honorio Saito

122149392

1. Datasets Normalizados e Padronizados

Para a geração dos *dataset* padronizado, decidi aplicar a classe `StandardScaler` do `sklearn.preprocessing` conforme a seguinte implementação:

```python
from sklearn.preprocessing import MinMaxScaler, StandardScaler

def preprocess(data, type: str):
  if type == 'standardize':
    standardizer = StandardScaler()
    return standardizer.fit_transform(data)
  scaler = MinMaxScaler()
  return scaler.fit_transform(data)
```

De modo que a classe `StandardScaler` realiza a transformação abaixo:

$$z = \frac{x - \mu}{s}$$

Onde

- x : Observação original.
- μ : Média para aquela *feature*.
- s : Desvio padrão amostral.
- z : Nova observação da variável após a transformação.

2. Regressão Logística com Gradiente Descendente

Para usarmos o método do gradiente descendente no lugar do cálculo da regressão logística por meio do `LogisticRegression` diretamente, usaremos a classe `SGDClassifier` que implementa um modelo linear genérico treinado via *Stochastic Gradient Descent (SGD)*.

Para a regressão logística multilcasse, desejamos minimizar a **log-loss** descrita por:

$$\mathcal{L}(\{\boldsymbol{w}_k\}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_{i,k}\log\left(p_{i,k}\right),$$

onde, para cada amostra i :

- $y_{i,k}$: Indicador *one-hot*:

$$y_{i,k} = \begin{cases} 1 & \text{se a amostra } i \text{ pertence à classe } k \\ 0 & \text{caso contrário.} \end{cases}$$

- $p_{i,k}$: Probabilidade prevista de i pertencer à classe k, obtida via **softmax**:

$$p_{i,k} = \frac{\exp(\boldsymbol{w}_k^{\mathrm{t}}\boldsymbol{x} + b_k)}{\sum_{j=1}^{K} \exp\left(\boldsymbol{w}_j^{\mathrm{t}}\boldsymbol{x}_i + b_j\right)}.$$

Em termos gerais:

- Para cada amostra i calculamos o vetor de **logits** dado por: $z_{i,k} = \boldsymbol{w}_k^{\mathrm{t}}\boldsymbol{x}_i + b_k$ para $k = 1, 2, 3$.
- Aplicamos a **softmax** para normalizar os **logits** nas probabilidades $p_{i,1}, p_{i,2}$ e $p_{i,3}$ que somam 1.
- A perda em i é $-\sum_k y_{i,k} \log(p_{i,k})$.
- Tomamos a média sobre todas as n amostras.

3. *K-Fold* ($k = 5$)

Por fim, aplicamos o método *K-Fold Cross-Validation* para avaliar a robustez do modelo em diferentes particionamentos dos dados. Para esse fim, utilizamos a classe `KFold` do `sklearn.model_selection` da seguinte forma:

```python
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shuffle=True, random_state=42)
for train_idx, test_idx in kf.split(X):
  # ...
```

A seguir, comparamos as métricas de **Precisão**, **Revocação (Recall)** e **F1-Score** para os *datasets* normalizados e padronizados:

Classes	Precisão	Revocação	F1-Score
Setosa	1	1	1
Versicolor	0,978	0,597	0,728
Virginica	0,713	0,983	0,822

Tabela 1: *Métrica para dataset normalizado.*

Classes	Precisão	Revocação	F1-Score
Setosa	1	1	1
Versicolor	0,917	0,820	0,860
Virginica	0,828	0,921	0,868

Tabela 2: *Métrica para dataset padronizado.*

Interpretação dos resultados:

1. **Setosa**: Em ambos os *datasets*, o modelo classifica todas as amostras de Setosa corretamente (Precisão = Revocação = 1.00). Isso nos informa que as nossas *features* separaram muito bem essa classe das demais.

2. **Versicolor**:

- No *dataset* normalizado, a Revocação média (0,597) é significativamente menor do que a Precisão (0,978), ou seja, o modelo comete poucos falsos positivos para Versicolor, mas deixa de reconhecer casos reais da classe.
- No *dataset* padronizado, essa diferença é menos marcante (Revocação = 0,820 e Precisão = 0,917), mostrando que a padronização ajuda na captura melhor da variabilidade interna.

3. **Virginica**: Com dados normalizados, a Precisão (0,713) é inferior à Revocação (0,983), isto é, o modelo tende a rotular como Virginica amostras de outras classes, mas não perde verdadeiras observações de Virginica.

4. Comparação das Matrizes de Confusão ($K = 5$)

Por fim, para cada um dos K folds, vamos analisar separadamente os resultados da matriz de confusão apenas para o *dataset* normalizado via *MinMax* para facilitar:

	Setosa	**Versicolor**	**Virginica**
Setosa	10	0	0
Versicolor	0	5	4
Virginica	0	0	11

Tabela 3: *Matriz de confusão para fold 1.*

	Setosa	**Versicolor**	**Virginica**
Setosa	13	0	0
Versicolor	0	4	6
Virginica	0	0	7

Tabela 4: *Matriz de confusão para fold 2.*

	Setosa	**Versicolor**	**Virginica**
Setosa	12	0	0
Versicolor	0	8	2
Virginica	0	0	8

Tabela 5: *Matriz de confusão para fold 3.*

	Setosa	**Versicolor**	**Virginica**
Setosa	8	0	0
Versicolor	0	5	5
Virginica	0	0	12

Tabela 6: *Matriz de confusão para fold 4.*

	Setosa	**Versicolor**	**Virginica**
Setosa	7	0	0
Versicolor	0	8	3
Virginica	0	1	11

Tabela 7: *Matriz de confusão para fold 5.*

	Setosa	**Versicolor**	**Virginica**
Setosa	50	0	0
Versicolor	0	30	20
Virginica	0	1	49

Tabela 8: *Matriz de confusão agregada.*

Portanto, percebemos que a separação entre as demais classes é trivial pra o modelo. Por outro lado, o modelo apresenta mais dificuldade na distinção entre *Versicolor* e *Virginica*. Como podemos ver, *Versicolor* sofre elevado número de falsos negativos (Revocação média $\approx 0,60$), enquanto *Virginica* raramente é "perdida" (Revocação $\approx 0,98$).

A variabilidade entre os folds (de 4 a 6 erros de Versicolor por fold) destaca a sensibilidade à partição dos dados, mas o padrão de confundir Versicolor com *Virginica* é consistente. Desse modo, para este modelo linear via SGD com *log-loss*, investir em *features* que destacam as diferenças de *Versicolor* é essencial para reduzir sistematicamente as confusões observadas em todos os folds.

Antes de finalizarmos, aqui está a matriz de confusão agregada para o *dataset* padronizado:

	Setosa	**Versicolor**	**Virginica**
Setosa	50	0	0
Versicolor	0	41	9
Virginica	0	4	46

Tabela 9: *Matriz de confusão agregada.*

De um modo geral, a padronização melhora substancialmente a capacidade do modelo de reconhecer *Versicolor*, ao custo de um aumento pequeno nos erros sobre *Virginica*. Em síntese, se o objetivo é

minimizar sobretudo a **perda de *Versicolor***, a padronização é a ideal. Por outro lado, a **normalização** via *MinMax* tende a sacrificar sistematicamente *Versicolor* em favor da *Virginica*.

5. Código

Python

```python
import numpy as np
from sklearn import datasets
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

iris = datasets.load_iris()
X, y = iris.data, iris.target
X = MinMaxScaler().fit_transform(X)

def preprocess(data, type: str = None):
  if type == 'standardize':
    standardizer = StandardScaler()
    return standardizer.fit_transform(data)
  scaler = MinMaxScaler()
  return scaler.fit_transform(data)

X = preprocess(X, type='standardize')

accuracies = []
reports = []
cms = []

kf = KFold(n_splits=5, shuffle=True, random_state=42)

for train_idx, test_idx in kf.split(X):
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    model = SGDClassifier(
        loss='log_loss',
        learning_rate='constant',
        eta0=0.01,
        max_iter=1000,
        tol=1e-3,
        random_state=42
    )
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
```

```
    accuracies.append(accuracy_score(y_test, y_pred))
    reports.append(classification_report(y_test, y_pred,
    target_names=iris.target_names, output_dict=True))
    cms.append(confusion_matrix(y_test, y_pred))

import pandas as pd

print("Acurácias por fold:", np.round(accuracies, 4))
print("Acurácia média:", np.mean(accuracies).round(4), "\n")

dfs = [
    pd.DataFrame(report).T.loc[ ['setosa','versicolor','virginica'],
                                ['precision','recall','f1-score'] ]
    for report in reports
]

mean_report = pd.concat(dfs, keys=range(len(dfs)), names=['fold','class']) \
                .groupby('class').mean()

print("Métricas médias por classe:")
print(mean_report)

for fold, (cm, df) in enumerate(zip(cms, dfs), start=1):
    print(f"\nMétrica para {fold=}:")
    print(df)
    print(f"\nMatriz de confusão para {fold=}:")
    print(cm)

total_cm = sum(cms)
print("\nMatriz de confusão agregada:")
print(total_cm)
```