UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Larissa Vitória Vasconcelos Sousa Matrícula: 519221

1. Introdução

Nesse relatório, irei apresentar a implementação e análise dos modelos KNN (K-Nearest Neighbors) e Regressão Logística usando a base de dados "load_diabetes_clean_dataset" (Conjunto de dados de previsão de diabetes), disponibilizada no repositório (códigos) da disciplina de IA.

Com as seguintes colunas:

1) pregnancies: gravidez
2) glucose: glicose
3) diastolic: pressão diastólica
4) triceps: tríceps
5) insulin: insulina
6) bmi: IMC
7) dpf: dpf
8) age: idade
9) diabetes: diabetes

O modelo KNN é um algoritmo de aprendizado supervisionado usado tanto para classificação quanto para regressão. No KNN, a classificação ou previsão é baseada na proximidade das amostras de treinamento aos novos pontos de dados. O "K" no KNN representa o número de vizinhos mais próximos que serão considerados para fazer uma previsão. No caso da regressão, o KNN calcula a média ou a mediana dos valores de destino dos "K" vizinhos mais próximos para fazer uma previsão contínua.

A regressão logística é um modelo de aprendizado supervisionado usado para resolver problemas de classificação binária ou multiclasse. Apesar do nome "regressão", a regressão logística é usada para tarefas de classificação, não de regressão. A regressão logística utiliza uma função logística (também conhecida como função sigmoide) para mapear uma combinação linear das variáveis de entrada para uma probabilidade de pertencer a uma determinada classe. Essa probabilidade é então convertida em uma classe predita com base em um limite de decisão (geralmente 0,5).

2. Observações

Não estava mostrando todas as colunas a base de dados "load_diabetes_clean_dataset". Usei o “print”, o “.head” e não deu certo. Pesquisei e

vi que antes do

```
print(diabetes_df.head())
```

para esse problema, essa linha de código resolveria:

```
pd.set_option('display.max_columns', None) # para mostrar todas as colunas
```

que configura o número máximo de colunas a serem exibidas quando um DataFrame é impresso no console.

3. Resultados e discussão

Iniciarei com o código referente ao modelo **KNN (K-Nearest Neighbors)**:

```python
# KNN
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
from src.utils import load_diabetes_clean_dataset
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import classification_report, confusion_matrix

diabetes_df = load_diabetes_clean_dataset()
pd.set_option('display.max_columns', None) # para mostrar todas as colunas
diabetes_df

X = diabetes_df[["pregnancies","glucose","diastolic","triceps","insulin","bmi","dpf","age"]].values
y = diabetes_df["diabetes"].values

# Dividindo em conjuntos de treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

Comecei importando as bibliotecas necessárias, logo após li a base de dados, defini x e y, dividi em conjuntos de treino e teste.

```python
# Criando vizinhos
neighbors = np.arange(1, 12)
train_accuracies = {}
test_accuracies = {}
teste = True
for neighbor in neighbors:
    # Configurar um classificador KNN
    knn = KNeighborsClassifier(n_neighbors=neighbor)

    # Ajuste do modelo
    knn.fit(X_train, y_train)

    # Precisão
    train_accuracies[neighbor] = knn.score(X_train, y_train)
    test_accuracies[neighbor] = knn.score(X_test, y_test)

print("Acurácia do treino: ",train_accuracies, '\n',"Acurácia do teste: ", test_accuracies)
```

Defini os vizinhos, instanciei o KNN, ajustei o modelo, calculei a precisão e mostrei na tela as acurácias de treino e teste.

Saída:

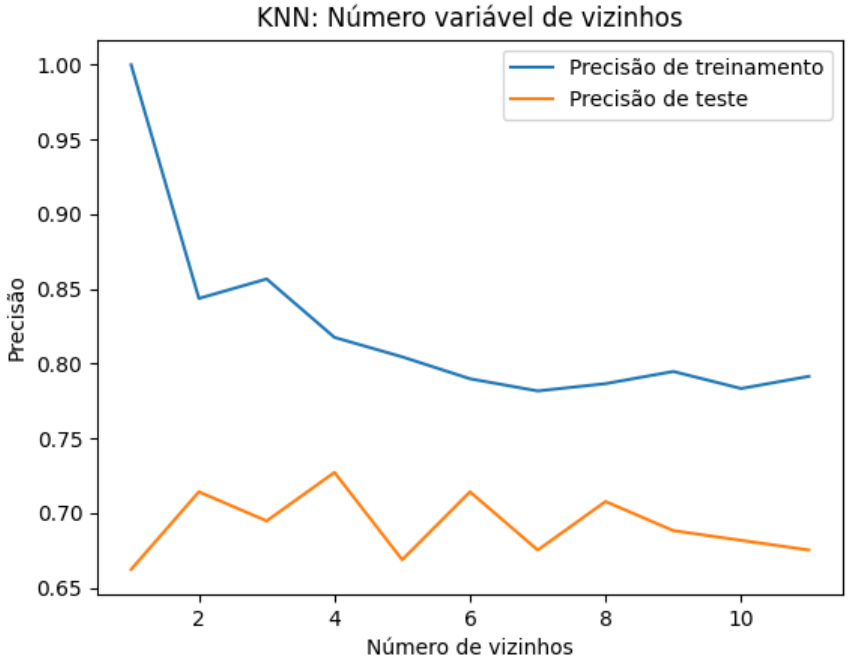
```
Acurácia do treino:  {1: 1.0, 2: 0.8436482084690554, 3: 0.8566775244299675, 4: 0.8175895765472313, 5: 
 Acurácia do teste:  {1: 0.6623376623376623, 2: 0.7142857142857143, 3: 0.6948051948051948, 4: 0.727272
```

A melhor acurácia de treino foi o 2 = 84,36%; e a melhor de teste foi o 4 = 72,72%.

Para fazer o gráfico e com ele descobrir o melhor k, comecei adicionando título, precisões de treinamento e teste e logo em seguida, plotei.

```python
# Adicionando um título
plt.title("\nKNN: Número variável de vizinhos")

# Plotando precisões de treinamento
plt.plot(neighbors, train_accuracies.values(), label="Precisão de treinamento")

# Plotando precisões de teste
plt.plot(neighbors, test_accuracies.values(), label="Precisão de teste")

plt.legend()
plt.xlabel("Número de vizinhos")
plt.ylabel("Precisão")

# Exibindo o gráfico
plt.show() # melhor k = 4
```

Saída:



Analisando o gráfico, vemos que o melhor k é o número 4, onde a precisão do teste atinge o seu máximo.

Para fazer a análise CrossValidation, li a base de dados, defini x e y, criei o objeto KFold com n_splits = 6, (ou seja, irei calcular 6 vezes) e instanciei o KNN. E, então, calculei o CrossValidation, média e desvio padrão.

```python
# Cross Validation
diabetes_cross = load_diabetes_clean_dataset()

X = diabetes_cross["dpf"].values.reshape(-1, 1)
y = diabetes_cross["diabetes"].values

# Criando o objeto Kfold
kf = KFold(n_splits=6, shuffle=True, random_state=5)

knn = KNeighborsClassifier()

# Calcular pontuações de validação cruzada 6 vezes
cv_scores = cross_val_score(knn, X, y, cv=kf)

# Print cv_scores
print("\nCross validation: ", cv_scores)

# Print da média
print("\nMédia: ", np.mean(cv_scores))

# Print o desvio padrão
print("\nDesvio padrão: ", np.std(cv_scores))
```

Saídas:

```
Cross validation:  [0.5546875 0.6015625 0.5546875 0.546875  0.5859375 0.5546875]
```

Essas são as pontuações de validação cruzada, que fornecem uma medida do desempenho do avaliador KNN em cada vez que executa (6 vezes) da validação cruzada. Cada número na lista representa a precisão alcançada pelo KNN em uma vez específica.

```
Média:  0.56640625
```

O desempenho médio do classificador KNN no conjunto de dados de diabetes é 0.56640625.

```
Desvio padrão:  0.02004531812283939
```

O desvio padrão (que indica a variabilidade entre as vezes executadas) dos scores de validação cruzada é 0.02004531812283939. Sabemos que, quanto menor é o desvio

padrão, mais consistente é o desempenho do modelo em diferentes partes do dataset, ou seja, esse resultado foi muito bom.

Para fazer a matriz de confusão, defini o x e y, x_treino, y_treino, etc., instancio o modelo knn, ajusto aos dados de treino, defino o y_pred, e, em seguida, mostro na tela os resultados.

```
# Confusion Matrix
X = diabetes_df.drop(['diabetes'],axis=1)
y = diabetes_df['diabetes'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42,stratify=y)

knn = KNeighborsClassifier(n_neighbors=6)

# Ajustando o modelo aos dados de treinamento
knn.fit(X_train, y_train)

# Prevendo os rótulos dos dados de teste: y_pred
y_pred = knn.predict(X_test)

# Gerando a matriz de confisão e o classification report
print("\nConfusion Matrix:\n ", confusion_matrix(y_test, y_pred))
print("\nClassification report:\n ", classification_report(y_test, y_pred))
```

Saídas:

```
Confusion Matrix:
  [[132  18]
 [ 49  32]]
```

Na primeira linha da matriz de confusão, temos os resultados para a classe 0 “não tem diabetes”, dos 150 (132 + 18) casos nesta classe, 132 foram corretamente classificados como “não tem diabetes” (verdadeiros negativos), e 18 foram classificados de forma errada como “tem diabetes” (falsos positivos). Na segunda linha, temos os resultados para a classe 1 “tem diabetes”, dos 81 (49 + 32) casos nesta classe, 49 foram erroneamente classificados como “não tem diabetes” (falsos negativos), e 32 foram corretamente classificados como “tem diabetes” (verdadeiros positivos).

```
Classification report:
              precision    recall  f1-score   support

           0       0.73      0.88      0.80       150
           1       0.64      0.40      0.49        81

    accuracy                           0.71       231
   macro avg       0.68      0.64      0.64       231
weighted avg       0.70      0.71      0.69       231
```

No Classification report, a precisão é a proporção de verdadeiros positivos em relação à soma de verdadeiros positivos e falsos positivos. Para a classe 0, a precisão é de aproximadamente 0.73, ou seja, 73% das classificações positivas estão corretas. Para a classe 1, a precisão é de aproximadamente 0.64, ou seja, 64% das classificações positivas estão corretas.

O Recall é a proporção de verdadeiros positivos em relação à soma de verdadeiros positivos e falsos negativos. Para a classe 0, o recall é de aproximadamente 0.88, ou seja, 88% dos casos positivos foram identificados corretamente. Para a classe 1, o recall é de aproximadamente 0.40, ou seja, 40% dos casos positivos foram identificados corretamente.

O F1-score é uma média harmônica entre precisão e recall. É uma medida balanceada que leva em consideração tanto falsos positivos quanto falsos negativos. Para a classe 0, o f1-score é de aproximadamente 0.80, ou seja, 80%. Para a classe 1, o f1-score é de aproximadamente 49%.

A acurácia é a proporção de classificações corretas em relação ao total de casos. A acurácia geral mostra que o modelo classificou corretamente 71% dos casos.

Agora iremos para o modelo de **Regressão Logística**:

Comecei importando as bibliotecas necessárias, logo após li a base de dados, defini x e y, instanciei o modelo regressão logística e ajustei o modelo.

```python
# REGRESSÃO LOGÍSTICA
from src.utils import load_diabetes_clean_dataset
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from src.utils import log_reg_diabetes

diabetes_df = load_diabetes_clean_dataset()
X = diabetes_df.drop(['diabetes'],axis=1)
y = diabetes_df['diabetes'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Instanciando o modelo
logreg = LogisticRegression()

# Ajuste do modelo
logreg.fit(X_train, y_train)
```

Previ as probabilidades, para mostrar na tela, gerei a matriz de confusão e o classification report. Crio o objeto KFold, instancio o modelo de regressão logística, calculo as pontuações de validação cruzada 6 vezes, usando o n_splits.

A ideia da KFold é simples: retornar os índices de cada partição de maneira aleatória. Esses índices são obtidos de acordo com o número de partições e são utilizados para construi-las a partir dos dados. E o Cross Validation permite obter as predições do modelo ao invés de apenas as métricas finais.

```python
# Prevendo probabilidades
y_pred_probs = logreg.predict_proba(X_test)[:, 1]

print("\nProbabilidades previstas: ", y_pred_probs[:10])

# Gerando a matriz de confusão e o classification report
print("\nMatriz de confusão:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Criando X e y
X = diabetes_df["dpf"].values.reshape(-1, 1)
y = diabetes_df["diabetes"].values

# Criando o objeto KFold
kf = KFold(n_splits=6, shuffle=True, random_state=5)

Logreg = LogisticRegression()

# Calculando as pontuações de validação cruzada 6 vezes
cv_scores = cross_val_score(Logreg, X, y, cv=kf)
```

Saídas:

```
Probabilidades previstas:  [0.23040852 0.28910622 0.73169399 0.63125227 0.45115251 0.07690538
 0.73115031 0.12432891 0.04388881 0.3482686 ]
```

A saída consiste em uma lista de valores que representam as probabilidades preditas pelo modelo para os primeiros 10 exemplos de teste. Cada valor na lista representa a probabilidade de um exemplo pertencer à classe positiva (diabetes) de acordo com as previsões do modelo. Essas probabilidades variam de 0 a 1 e indicam a confiança do modelo na classificação de cada exemplo como classe positiva (diabetes). Por exemplo, o segundo valor na saída é 0.73169399, o que significa que o modelo atribui uma probabilidade de aproximadamente 73% de o segundo exemplo de teste pertencer à classe positiva (diabetes).

```
Matriz de confusão:
 [[129  21]
 [ 39  42]]
```

Na primeira linha da matriz de confusão, temos os resultados para a classe 0 “não tem diabetes”, dos 150 (129 + 21) casos nesta classe, 129 foram corretamente classificados como “não tem diabetes” (verdadeiros negativos), e 21 foram classificados de forma errada como “tem diabetes” (falsos positivos). Na segunda linha, temos os resultados para a classe 1 “tem diabetes”, dos 81 (39 + 42) casos nesta classe, 39 foram erroneamente classificados como “não tem diabetes” (falsos negativos), e 42 foram corretamente classificados como “tem diabetes” (verdadeiros positivos).

```
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.86      0.81       150
           1       0.67      0.52      0.58        81

    accuracy                           0.74       231
   macro avg       0.72      0.69      0.70       231
weighted avg       0.73      0.74      0.73       231
```

No Classification report para a classe 0, a precisão é de aproximadamente 0.77, ou seja, 77% das classificações positivas estão corretas. Para a classe 1, a precisão é de aproximadamente 0.67, ou seja, 67% das classificações positivas estão corretas.

O Recall para a classe 0, é de aproximadamente 0.86, ou seja, 86% dos casos positivos foram identificados corretamente. Para a classe 1, o recall é de aproximadamente 0.52, ou seja, 52% dos casos positivos foram identificados corretamente.

O F1-score para a classe 0, é de aproximadamente 0.81, ou seja, 81%. Para a classe 1, o f1-score é de aproximadamente 58%.

A acurácia geral mostra que o modelo classificou corretamente 74% dos casos.

Calculei o CrossValidation, média e desvio padrão.

```python
# Print cv_scores
print("\nCv score: ", cv_scores)

# Print da média
print("\nMédia: ", np.mean(cv_scores))

# Print do desvio padrão
print("\nDesvio padrão: ", np.std(cv_scores))
```

Saídas:

O segundo número na lista é 0.7109375, ele representa a precisão alcançada pelo RL em uma vez específica.

```
Cv score:  [0.6484375 0.7109375 0.65625   0.625     0.65625   0.6328125]
```

O desempenho médio do classificador RL no conjunto de dados de diabetes é 0.6549479166666666.

```
Média:  0.6549479166666666
```

O desvio padrão é 0.027590651172418088.

```
Desvio padrão:  0.027590651172418088
```

```python
print("\nCurva ROC\n")

y_prob,y_test,_= log_reg_diabetes()

# Gerar valores de curva ROC: fpr, tpr, thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

plt.plot([0, 1], [0, 1], 'k--')

# Plotando tpr contra fpr
plt.plot(fpr, tpr)
plt.xlabel('Taxa de Falsos Positivos')
plt.ylabel('Taxa Verdadeiros Positivos')
plt.title('Curva ROC para previsão de diabetes')
plt.show()
```

Utilizamos a análise ROC para exibir graficamente, comparar e avaliar a acurácia. A curva ROC integra três medidas de precisão relacionadas: sensibilidade (verdadeiro positivo), especificidade (verdadeiro negativo) e ASC.

O padrão para os modelos de probabilidades é a divisão de 0.5 ou 50% para definição de uma classe, caso o resultado da classe apresente valor menor que o limite (0.0 a 0.49) o resultado (binário) será *0*, caso contrário, se der maior que o limite (0.5 a 1) o resultado será *1*. Porém, esse valor de limite ou conhecido também como *threshold* pode ser ajustado, de acordo com o comportamento desejado para as predições.

Do que precisamos para plotar a Curva ROC?

$$TPR = \frac{TP}{TP + TN}$$

- **TPR:** *True Positive Rate** ou Taxa de Verdadeiros Positivos
- **TP:** *True Positive* ou Verdadeiros Positivos
- **FN:** *False Negative* ou Falsos Negativos

Isso descreve o quão bom nosso modelo é acertar a previsão das classes positivas, quando a saída for positiva. O TPR também é conhecido como *Sensitivity* ou **Sensibilidade**.

$$FPR = \frac{FP}{FP + TN}$$

- **FPR:** *False Negative Rate** ou Taxa de Falsos Negativos
- **FP:** *False Positive* ou Falsos Positivos
- **TN:** *True Negative* ou Verdadeiros Negativos

Também conhecido como taxa de alarmes falsos, que é a frequência de classes positivas previstas, quando resultado verdadeiro é negativo. Pode-se calcular como o inverso da *Specificity* ou **Especificidade**.

$$Specificity = \frac{TN}{TN + FP}$$

Specificity: ou Especificidade

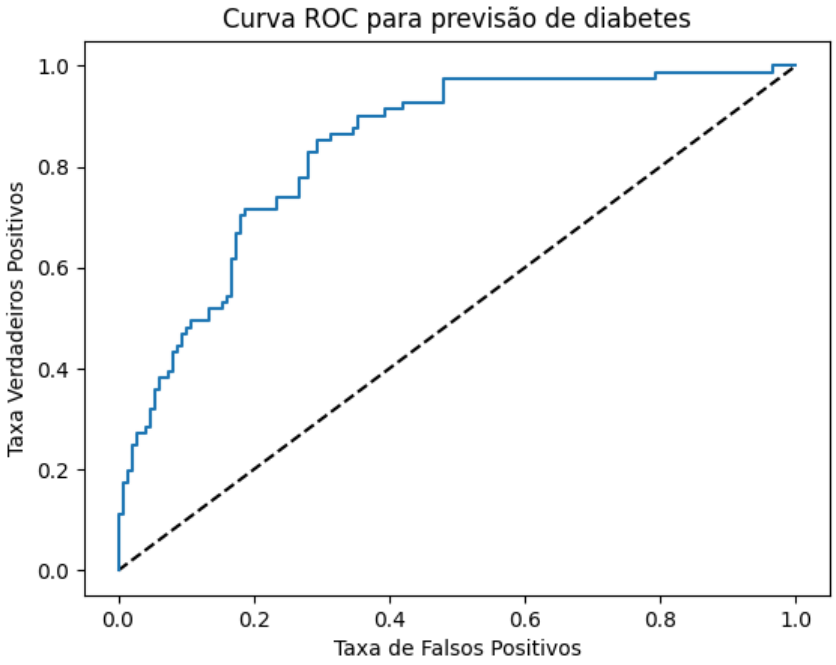
TN: *True Negative* ou Verdadeiros Negativos

FP: *False Positive* ou Falsos Positivos

Onde:

$$FPR = 1 - Specificity$$

Saída:



4. Conclusões

Métricas	KNN		REGRESSÃO LOGÍSTICA	
	Classe 0	Classe 1	Classe 0	Classe 1
Precision	0.73	0.64	0.77	0.67
F1-score	0.80	0.49	0.81	0.58
Acurácia geral	0.71		0.74	
Recall	0.88	0.40	0.86	0.52

Com base nas métricas apresentadas, é possível observar que o modelo de Regressão Logística teve um desempenho geral melhor em comparação com o modelo KNN.

- Analisando a precisão, o modelo de RL obteve uma precisão maior para ambas as classes (0 e 1) em comparação com o modelo KNN. Isso indica que o modelo de Regressão Logística teve uma melhor capacidade de classificar corretamente as amostras para cada classe.
- O F1-score para a classe 0 foi maior no modelo de Regressão Logística em comparação com o modelo KNN. Para a classe 1, o F1-score foi consideravelmente maior no modelo de Regressão Logística. O F1-score é uma medida que combina precisão e recall, portanto, um valor mais alto indica um melhor equilíbrio entre essas métricas.
- Sobre a acurácia geral, o modelo RL obteve uma acurácia geral de 0.74, enquanto o modelo KNN teve uma acurácia geral de 0.71. Isso significa que o modelo de Regressão Logística teve uma taxa de previsões corretas mais alta em relação ao total de amostras.
- Embora o modelo KNN tenha obtido um recall mais alto para a classe 0 (0.88), o modelo RL teve um recall mais alto para a classe 1 (0.52). O recall é importante para identificar a capacidade do modelo em encontrar corretamente as amostras de uma determinada classe.

Considerando esses pontos, o modelo de Regressão Logística apresentou um desempenho geral melhor em comparação com o modelo KNN, demonstrando uma maior capacidade de classificação e um equilíbrio melhor entre precisão e recall.

Medidas	KNN	REGRESSÃO LOGÍSTICA
Média	0.56640625	0.6549479166666666
Desvio padrão	0.02004531812283939	0.027590651172418088

A média e o desvio padrão são medidas estatísticas que fornecem informações sobre a distribuição dos valores.

Temos a média para o KNN igual a 0.56 e para a Regressão Logística igual a 0.65. Esses valores representam a média dos valores obtidos para as métricas em todas as execuções ou amostras consideradas. Já o desvio padrão representa a variação ou a diferença média entre cada valor e a média. Um desvio padrão baixo indica que os valores estão próximos da média (ex.: desvio padrão para o KNN igual a 0.02), enquanto um desvio padrão alto indica que os valores estão mais dispersos (ex.: desvio padrão para a RL igual a 0.027).

Aqui são as matrizes de confusão:

KNN		Valor predito	
		Não	Sim
Valor real	Não	132 (VN)	18 (FP)
	Sim	49 (FN)	32 (VP)

RL		Valor predito	
		Não	Sim
Valor real	Não	129 (VN)	21 (FP)
	Sim	39 (FN)	42 (VP)

5. Próximos passos