UNIVERSIDADE FEDERAL DO CEARÁ

TECHNICAL REPORT

Aluno: Vítor Barbosa da Silva

1. Introdução

O dataset de classificação contém dados relacionados à transações em cartões de crédito de diferentes usuários em diferentes lugares da Europa. Possui colunas que apresentam anonimamente os dados das transações, como horário, local, etc. A atividade em cima do dataset consiste em analisar os dados e treinar o sistema para classificar se a transação tem caráter fraudulento ou não.

Já o dataset de regressão contém dados relacionados à passagens aéreas compradas. Possui colunas que apontam os dados da passagem, como voo, linha aérea, cidade de origem e de destino, preço, etc. A atividade em cima do dataset consiste em analisar os dados e treinar o sistema para analisar o quanto os atributos, como classe, linha aérea e etc, podem contribuir para a variação no preço da passagem.

2. Observações

Devido ao grande volume de dados tive que realizar uma amostragem estratificada para ser utilizada no knn, pois o tempo e o custo de processamento para gerar os resultados estava muito elevado. Assim, utilizei na terceira questão o valor de 5000 para cada uma das classes e na quarta questão o valor de 30000 mil para cada uma das classes, já que a implementação do knn nessa questão não seria necessariamente manual. Um ponto a ser destacado foi a dificuldade que tive para tratar os dados, já que as atividades exigem que os dados sejam numéricos e nem todos eram.

3. Resultados e discussão

1ª Questão.

Nesta questão o requisito era fazer um pré-processamento do dataset. iniciei carregando o dataset, após isso utilizei o comando `df = df.dropna()` para

excluir linhas que possuíam algum dado nulo ou branco. Em seguida precisei apenas realizar o salvamento do novo dataset criado na minha pasta de datasets

```
dataframe_ajustado = "../AV1/datasets/novo_creditcard.csv"
df.to_csv(dataframe_ajustado, index=False)
```

2ª Questão.

Nesta questão foi necessária a implementação da função KNN de forma manual:

```
def knn(X_train, y_train, X_test, k, dist_func, cov_inv=None):  new *
    y_pred = []

    for test_point in X_test:
        # Calculando distâncias do ponto de teste para todos os pontos de treino
        distances = []
        for train_point, train_label in zip(X_train, y_train):
            # Passar cov_inv apenas se a função de distância for Mahalanobis
            if dist_func == dist_mahalanobis:
                dist = dist_func(test_point, train_point, cov_inv)  # Passando cov_inv para Mahalanobis
            else:
                dist = dist_func(test_point, train_point)
            distances.append((dist, train_label[0]))

        distances.sort(key=lambda x: x[0])
        k_neighbors = [label for _, label in distances[:k]]
        majority_class = max(set(k_neighbors), key=k_neighbors.count)
        y_pred.append(majority_class)

    return np.array(y_pred)
```

Foi necessário fazer a separação do conjunto de treino e de teste utilizando a função train_test_split, onde o X armazenou os valores contidos nas demais colunas do dataset, excetuando-se a de Class, e y armazenou os valores de Class. Para k=7 as acurácias obtidas para cada distância foram as seguintes:

```
Acurácia do modelo dist_euclidiana: 99.85%
Acurácia do modelo dist_manhattan: 99.85%
Acurácia do modelo dist_mahalanobis: 98.60%
Acurácia do modelo dist_chebyshev: 99.75%
```

obs: Achei um pouco estranho os valores das distâncias euclidiana e Manhattan terem sido os mesmos, mas tentei outras abordagens e o valor delas continuava se repetindo.

3ª Questão.

Para essa questão, que pedia a melhor distância, eu escolhi a Euclidiana. Após isso fiz a normalização para verificar se iria interferir no valor da acurácia. O código utilizado foi o seguinte:

```
# Normalização Logarítmica
X_train_pos = X_train - np.min(X_train) + 1
X_test_pos = X_test - np.min(X_test) + 1
X_train_log = np.log1p(X_train_pos)
X_test_log = np.log1p(X_test_pos)

# Normalização Z-score (média zero e variância unitária)
scaler = StandardScaler()
X_train_z = scaler.fit_transform(X_train)
X_test_z = scaler.transform(X_test)
```

Em seguida utilizei a função KNN, ainda implementada de forma manual, para calcular o acertos e gerar o resultado a ser analisado:

```
Acurácia do KNN com diferentes normalizações:
Logarítmica: 92.60%
Z-score: 99.35%
```

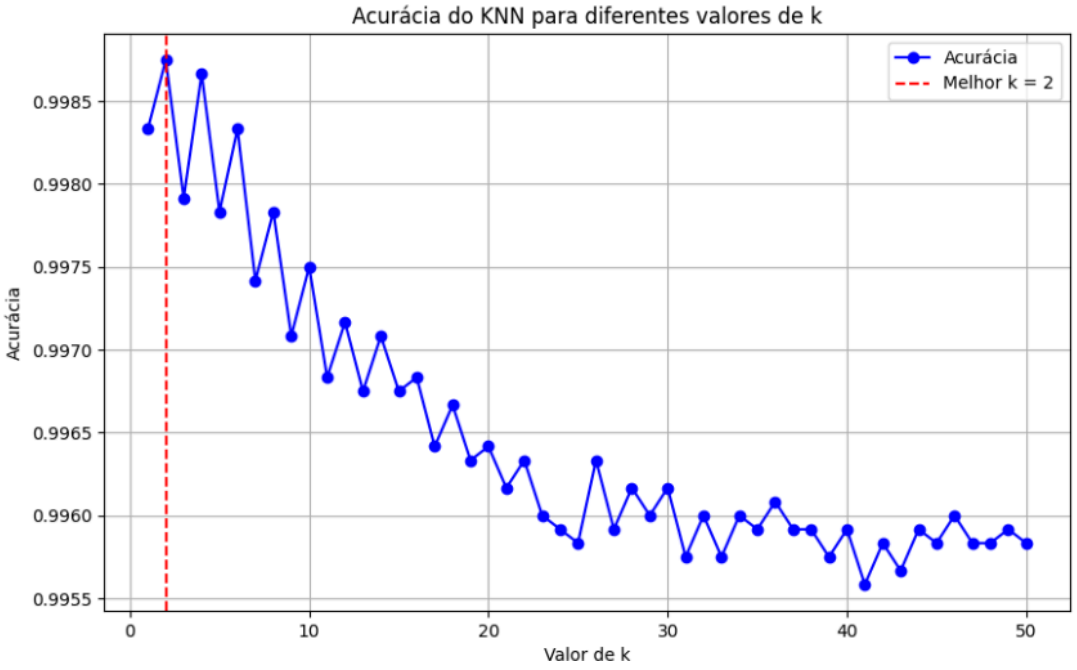
Após analisar o resultado, constatei que a normalização interfere no valor da acurácia.

4ª Questão.

Nesta questão é pedido que escolha a melhor parametrização, com base nos resultados a melhor foi a Z - score, então foi essa a escolhida. Após fazer

novamente a separação e normalização dos conjuntos, utilizei a implementação mais convencional do KNN, da biblioteca sklearn.

Utilizando um intervalo de 1 a 51, para verificação do melhor K, obtive o resultado de que o valor melhor é o k=2, com uma acurácia de 99,88%.



5ª Questão.

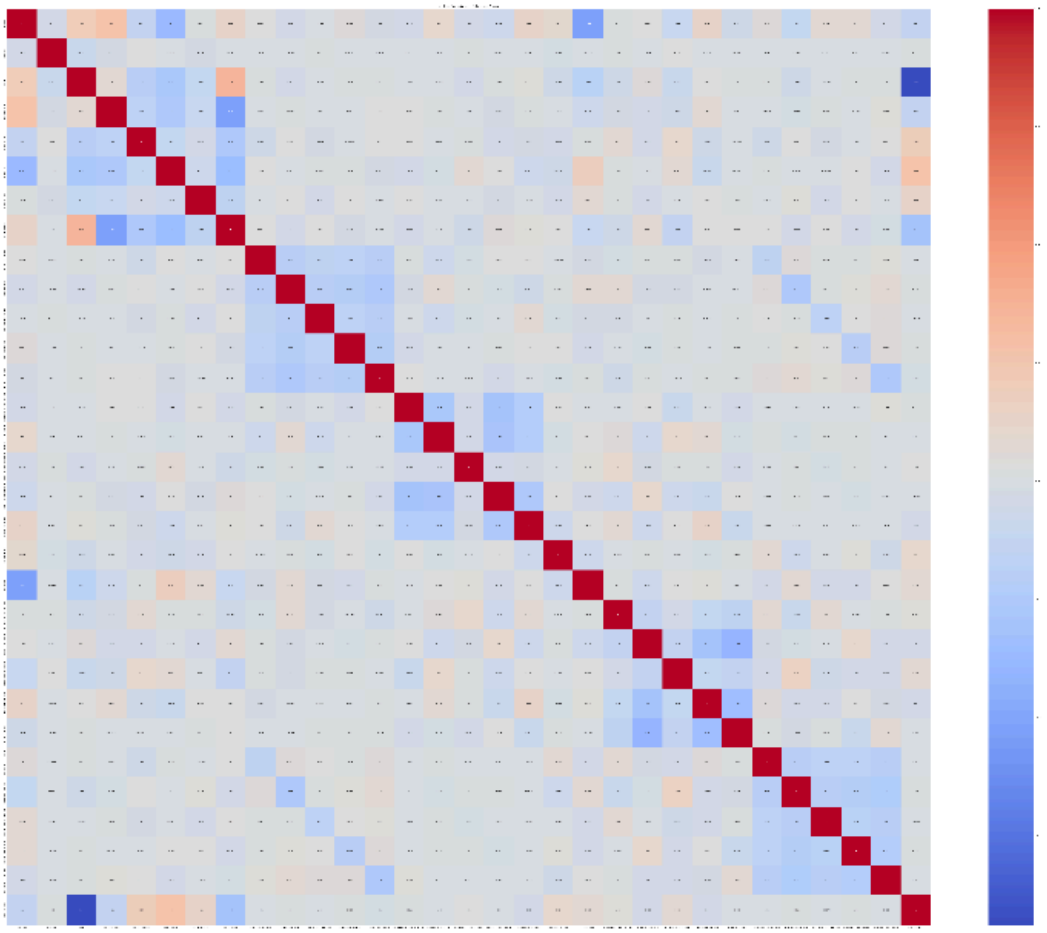
Nesta questão foi utilizada uma outra metodologia, a de regressão. Inicialmente comecei com o pré-processamento do dataset. Excluí linhas com valores NaN e também algumas colunas que não tinham relevância para a análise. As colunas foram a primeira que não tinha nome e a terceira que tinha o nome flight.

Tive que realizar uma conversão das colunas categóricas para gerar a matriz de correlações e verificar qual tinha mais relação com a coluna price. Para isso utilizei uma abordagem chamada one_hot encoder, onde converte dados categóricos do dataset para números:

```
df_one_hot = pd.get_dummies(df, columns=colunas_categoricas, drop_first=True)

matriz_correlacao = df_one_hot.corr()
```

Em seguida, gerei a matriz de correlações, para analisar.



Obs: Devido à grande quantidade de dados, é necessário dar o zoom para ver os dados da matriz. Porém só é possível no console, quando o código é gerado, assim a imagem é apenas para comprovar graficamente.

6ª Questão.

Nesta questão, utilizando-se do atributo mais relevante já previsto na questão anterior, foi implementada a regressão linear. Tive que converter os valores da coluna class e também implementadas as métricas:

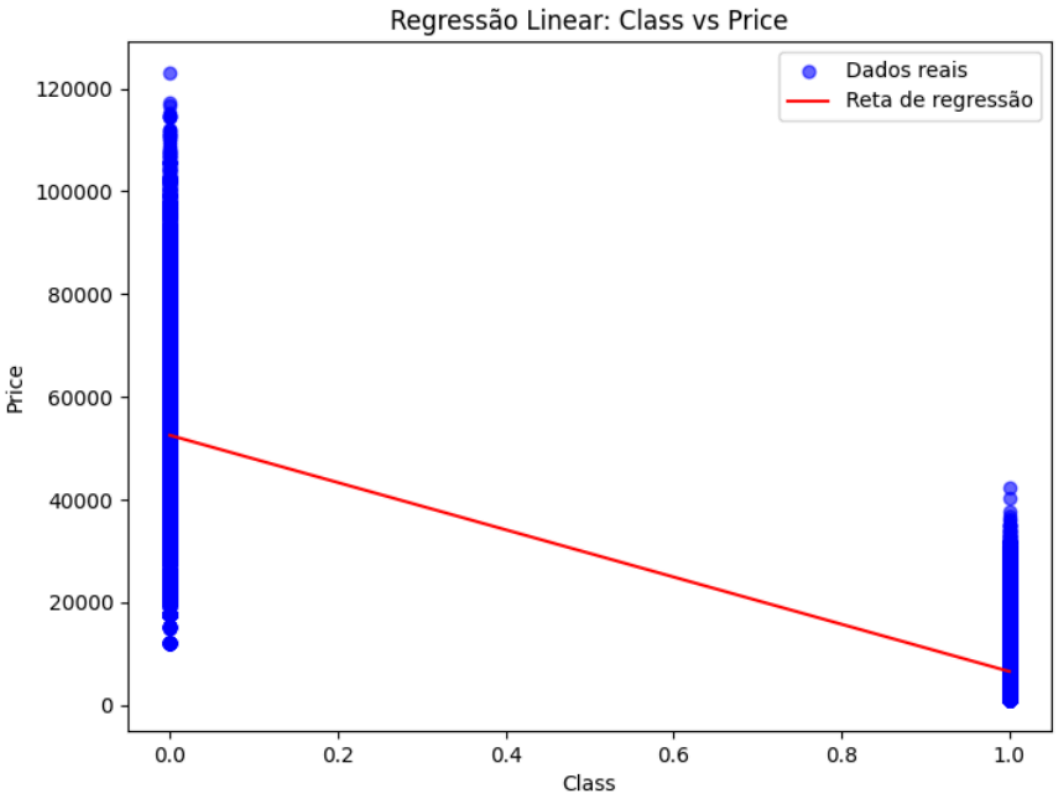
```
RSS = np.sum((y - y_pred) ** 2)
MSE = mean_squared_error(y, y_pred)
RMSE = np.sqrt(MSE)
R_squared = r2_score(y, y_pred)
```

A geração dos resultados das métricas foi a seguinte:

```
RSS: 18620.83 bilhões
MSE: 62037785.99
RMSE: 7876.41
R²: 0.88
```

Com base nesses valores percebi que os valores estão muito discrepantes, provavelmente devido à presença de outliers na coluna price. Porém a métrica R^2 indica uma forte relação entre as colunas class e price.

E o gráfico da reta de regressão e a nuvem de atributo foi o seguinte:



7ª Questão.

Nesta questão foi utilizada a função kfold e cross validation implementada manualmente:

```
def k_fold_cross_validation(X, y, k=5):
    fold_size = len(X) // k
    rss_list = []
    mse_list = []
    rmse_list = []
    r2_list = []

    for i in range(k):
        val_start = i * fold_size
          val_end = (i + 1) * fold_size if i != k - 1 else
len(X)
```

```
        X_val = X[val_start:val_end]
        y_val = y[val_start:val_end]
                X_train = np.concatenate([X[:val_start],
X[val_end:]], axis=0)
                y_train = np.concatenate([y[:val_start],
y[val_end:]], axis=0)
        model = LinearRegression()
        model.fit(X_train, y_train)
        y_pred = model.predict(X_val)

        rss = compute_RSS(y_val, y_pred)
        mse = compute_MSE(y_val, y_pred)
        rmse = compute_RMSE(y_val, y_pred)
        r2 = compute_R_squared(y_val, y_pred)

        rss_list.append(rss)
        mse_list.append(mse)
        rmse_list.append(rmse)
        r2_list.append(r2)

    avg_rss = np.mean(rss_list)
    avg_mse = np.mean(mse_list)
    avg_rmse = np.mean(rmse_list)
    avg_r2 = np.mean(r2_list)

    return avg_rss, avg_mse, avg_rmse, avg_r2
```

As métricas de medida também foram implementadas manualmente:

```
def compute_RSS(predictions,y):
    aux = np.square(y - predictions)
    RSS = np.sum(aux)
    return RSS
def compute_MSE(predictions,y):
    RSS = compute_RSS(predictions, y)
    MSE = np.divide(RSS, len(predictions))
    return MSE
def compute_RMSE(predictions,y):
    MSE = compute_MSE(predictions, y)
    RMSE = np.sqrt(MSE)
    return RMSE
def compute_R_squared(predictions,y):
    var_pred = np.sum(np.square(y - np.mean(y)))
    var_data = compute_RSS(predictions, y)
    r_squared = np.divide(var_pred, var_data)
    return r_squared
```

Nessas métricas os valores foram diferentes mas continuam indicando uma grande discrepância. Porém o R^2 representa uma forte ligação e indício de que a classe afeta no preço.

```
RSS médio: 3901501857943.18
MSE médio: 64990911.90
RMSE médio: 6966.19
R² médio: 0.90
```

4. Conclusões

Nem todos os resultados esperados foram satisfeitos, pois algumas métricas representaram erros. Mas acredito que os resultados obtidos nas demais questões foram satisfatórios. Considero um avanço no entendimento e execução dos conteúdos já vistos. Vale também ressaltar a importância da atividade proposta e do relatório para o entendimento e aprendizado dos alunos com relação ao conteúdo abordado.

5. Próximos passos

Tenho como sugestão pegar um dataset de alguém aleatoriamente e refazer juntamente com a turma expondo os pontos importantes e corrigindo juntamente com a turma o código.