Matheus Fernandes Kovaleski 00274744

Trabalho 1 de Aprendizado de Máquina

Objetivo do trabalho:

O trabalho visa o desenvolvimento de um modelo baseado em AM para prever a nota dos alunos ao fim do um curso a partir de questionários que tentam mapear o estado emocional dos mesmos. Para tanto, utilizamos o conjunto de dados Student life disponível na plataforma kaggle.

Dados utilizados:

O dataset é bem vasto em quantidade de informações disponíveis para se usar, tento 9 pastas, que correspondem dados retirados de interfaces diferentes, onde cada pasta possui uma série de arquivos referentes a cada iteração feita com os usuários. Como a utilização de todos os dados presentes no dataset tornaria a execução do trabalho muito mais complexa, e tendo em vista que para o problema escolhido a necessidade de todos eles seria desnecessária, foi feito um filtro onde só foram usados 2 arquivos presentes em 2 pastas. Para se ter noção do comportamento dos estudantes foi usado os dados presentes no arquivo de LonelinessScale que é um questionário que tenta mapear o sentimento de solidão dos usuários através das seguintes perguntas:

"1. I feel in tune with the people around me",
"2. I lack companionship",
"3. There is no one I can turn to",
"4. I do not feel alone",
"5. I feel part of a group of friends",
"6. I have a lot in common with the people around me",
"7. I am no longer close to anyone",
"8. My interests and ideas are not shared by those around me",
"9. I am an outgoing person",
"10. There are people I feel close to",
"11. I feel left out",
"12. My social relationships are superficial",
"13. No one really knows me well",
"14. I feel isolated from others",
"15. I can find companionship when I want it",
"16. There are people who really understand me",
"17. I am unhappy being so withdrawn",
"18. People are around me but not with me",
"19. There are people I can talk to",
"20. There are people I can turn to".

Também se faz uso do arquivo de notas dos usuários, que possui as notas no formato GPA.

Para que o conjunto de dados funcione bem funcionasse bem, visto que são 2 arquivos separados foi feito um merge dos dados utilizando a coluna de userID(uid), tendo feito isso a quantidade de instâncias restantes no conjunto de dados ficou em 48, sendo distribuídas em 4 classes referentes a 4 classes de notas: 0, 2, 3, 4. Para os dados utilizados, foi feito alguns passos de pré processamento, primeiro o uso de marge, como

descrito anteriormente, segundo o drops de colunas desnecessárias como outros atributos de GPA, logo em seguida eu converto todas as respostas presentes no dados para um valor numérico que representa uma resposta, no caso often = 1, rarely = 2, sometimes = 3 e never = 4, após isso por questão de simples conveniência no uso dos modelos é feito uma conversão de tipos, onde os dados referentes ao GPA são convertidos de float para string.

Para a execução dos experimentos foi utilizado a linguagem python com o uso de algumas bibliotecas como, tensorflow para a implementação de uma rede neural, pandas para os dados, numpy para facilitar alguns cálculos, sklearn para implementar os modelos e algumas métricas, além de algumas bibliotecas do sistema operacional para a leitura dos arquivos que continham os dados.

Os modelos usados para o experimento foram: arvore de decisão, naive bayes e regressão linear, também foi feita a implementação de um modelo de rede neural, como descrito anteriormente, mas descreveremos esse mais a frente. Os parâmetros utilizados para maximizar a acurácia dos modelos foram os valores random state e o valor de k no número de folds. Para se descobrir o melhor valor de random state foi feito uma variação de 0 a 999 verificando o valor médio dos modelos de forma geral. Para o cálculo de médias foi usado o modelo geométrico, visto que ele melhor representa uma média das acurácias, no caso o valor encontrado foi de random_state = 590. Para o k no número de fold foi feito 6 execuções usando a mesma metodologia, chegando ao valor de k = 5.

o código implementado para o kfold foi o seguinte:

```
def kfold_cross_validation(X, y, classes, y_train, dttest, n_splits=5, epochs=10):
    num_samples = len(X)
    fold_size = num_samples // n_splits
    accuracies = []

    num_classes = len(classes)

    model = buildModel(X.shape[1], num_classes)
    accByModel = []
    dt = DecisionTreeClassifier()
    nB = GaussianNB()
    LR = LinearRegression()
    mConf = []
    class_mapping = {class_name: index for index, class_name in enumerate(classes)}

    for modelI in range(3):
        for i in range(n_splits):
            start = i * fold_size
            end = (i + 1) * fold_size if i < n_splits - 1 else num_samples

            X_train = pd.concat([X.iloc[:start], X.iloc[end:]]).reset_index(drop=True)
            dttrain = pd.concat([y.iloc[:start], y.iloc[end:]]).reset_index(drop=True)
            X_test = X.iloc[start:end].reset_index(drop=True)
            dttestK = y.iloc[start:end].reset_index(drop=True)

            dttest_mapped = [class_mapping.get(class_name, -1) for class_name in dttestK]

            accuracy = 0
            if(modelI == 0):
                accuracy = decisionTreeK(X_train, dttrain, X_test, dttest_mapped, dt)
            elif(modelI == 2):
                #accuracy = neuralNetwork1(X_train, dttrain, X_test, dttestK, classes, model)
                accuracy = linearRegression(X_train, dttrain, X_test, dttest_mapped, LR)
            else:
                accuracy = naiveBayes(X_train, X_test, dttrain, dttest_mapped, nB)
            accuracies.append(accuracy)
```

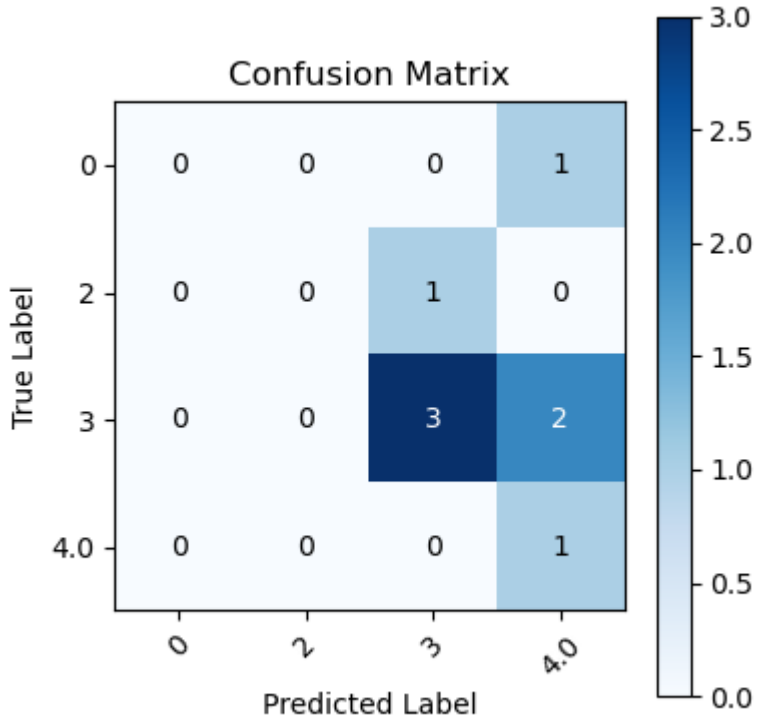
No caso o pipeline do uso do kfold para cada modelo é para cada i fold é usado para o treinamento do modelo onde em cada iteração do código variando de 0 a 2 onde cada número representa um modelo, onde 0 é a árvore de decisão, 1 é naive Bayes e 3 e regressão linear, os folds são passados para o modelo específico e após o treinamento é feito a avaliação do modelo usando o dados de teste pegos no split entre train e test feito no pré processamento. De forma mais visual, segue um pseudocódigo da descrição.

```
Função kfold_cross_validation(X, y, classes, y_train, dttest, n_splits, epochs)
    num_samples = tamanho de X
    fold_size = num_samples dividido por n_splits
    num_classes = tamanho de classes

    Inicialize uma lista vazia accByModel para armazenar as acurácias dos modelos
    Inicialize uma lista vazia accuracyBymodelTrain para armazenar as acurácias de treino dos modelos
    Crie modelos DecisionTreeClassifier dt, GaussianNB nB e LinearRegression LR
    Inicialize uma lista vazia mConf para armazenar as matrizes de confusão

    Crie um dicionário class_mapping para mapear classes para índices

    Para cada modeloI de 0 a 2
        Inicialize uma lista vazia accuracies para armazenar as acurácias de cada fold
        Para cada fold de 0 a n_splits - 1
            Calcule os índices de início (start) e fim (end) para o fold atual
            Divida os conjuntos de dados em treino e teste com base nos índices
            Calcule as previsões do modelo atual para o fold atual (usando o modelo correspondente)
            Calcule a acurácia para esse fold e adicione à lista accuracies

        Adicione a lista accuracies à lista accuracyBymodelTrain

        Calcule a acurácia de teste para o modelo atual (usando o modelo correspondente)
        Calcule a matriz de confusão para o modelo atual

        Adicione a acurácia de teste à lista accByModel
        Adicione a matriz de confusão à lista mConf

    Retorne accByModel, accuracyBymodelTrain, mConf
```

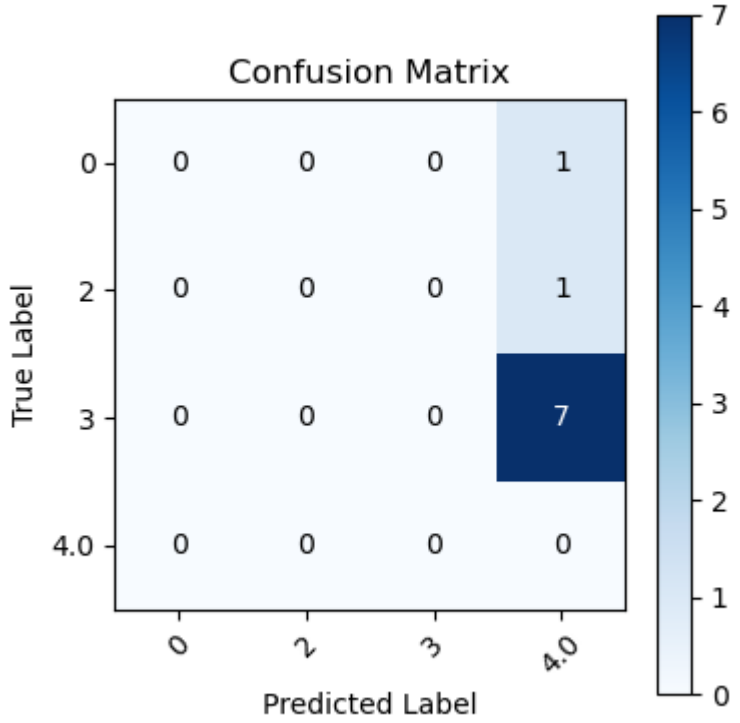
Avaliação dos Resultados

Abaixo temos a matriz de confusão relacionada a cada modelo, e mais abaixo temos um gráfico para avaliarmos as acurácias obtidas pelos modelos quando usados os dados de teste.

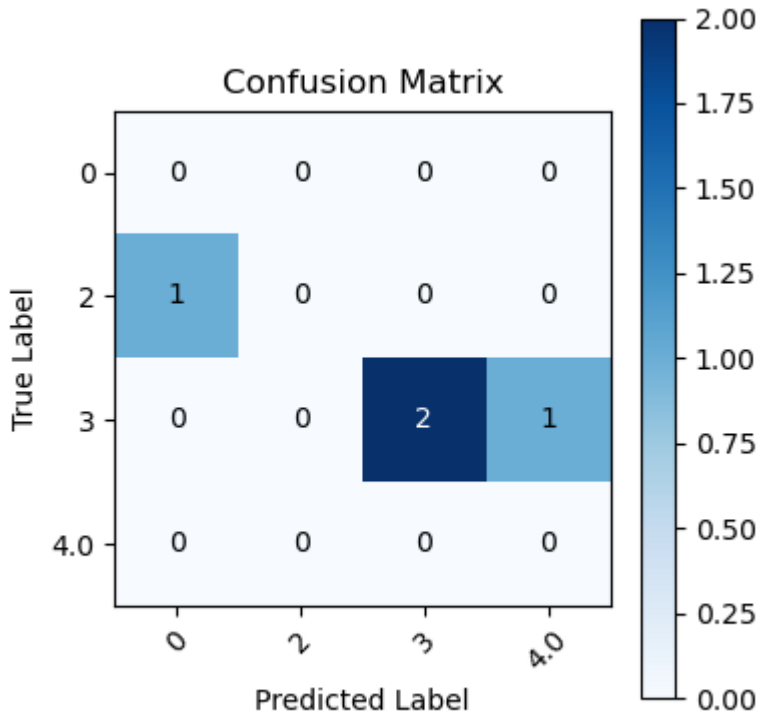
DT



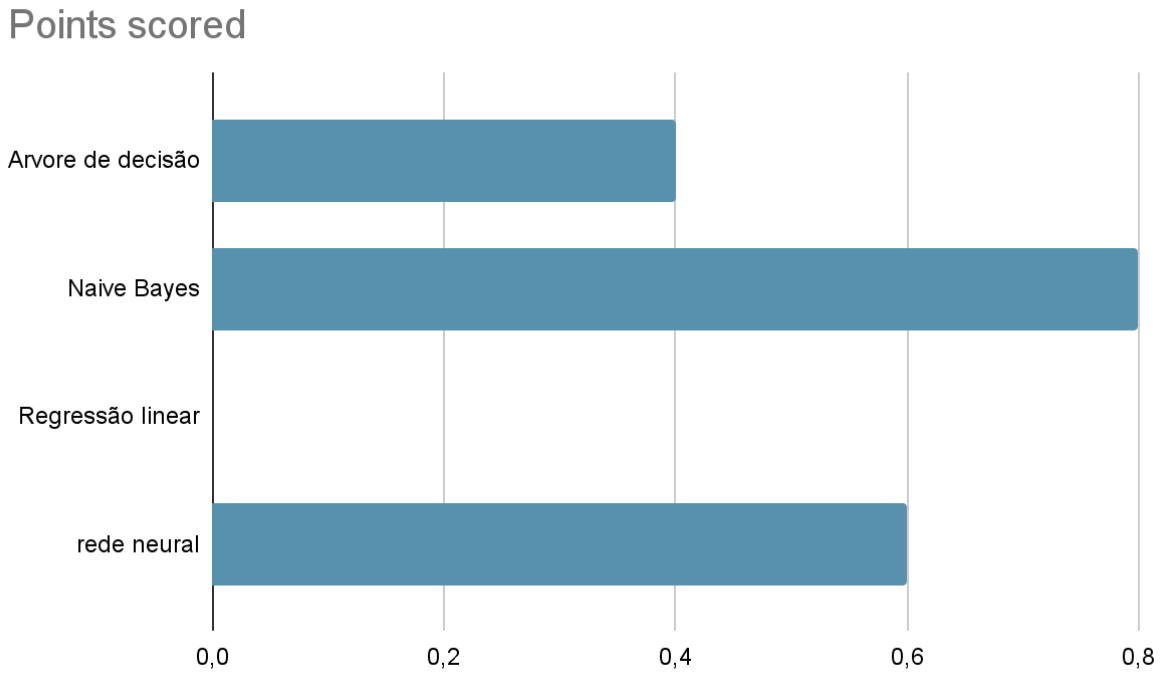
NB



RL



Avaliando as matrizes e comparando com o gráfico de avaliação das acurácias, podemos concluir que provavelmente alguns valores não foram contabilizados, pois existe uma divergência no número de instâncias por matrix criada.



Avaliando o gráfico acima, podemos concluir que dentre os 3 modelos implementados originalmente o que se sai melhor foi o naive bayes, visto que pelos dados de entrada, ele é o que melhor consegue criar uma correlação entre um “sintoma” de solidão com a nota obtida no final. Como esperado, visto que o algoritmo funciona de forma parecida com o anterior, o segundo melhor foi Árvore de decisão, dado que ele separa os dados em subconjuntos, ele se sairia melhor para o problema. O Algoritmo de regressão linear foi o que se saiu pior pois não existe uma correlação muito forte entre os dados de entrada e o resultado de saída.

Visto que o resultados objetivos em sua maioria foi insatisfatório, foi decidido a implementação de um outro modelo, usando redes neurais, com o seguinte formato.

```
def buildModel(shape, num_classes):
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(21, activation='relu', input_shape=(shape,)),
        tf.keras.layers.Dense(63, activation='relu'),
        tf.keras.layers.Dense(num_classes, activation='softmax')
    ])

    model.compile(loss='categorical_crossentropy',
                  optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  metrics=['accuracy'])
    return model
```

Esse modelo se sai melhor que os outros pois ele tenta reconhecer padrões entre os dados que podem não ser muito claros para os outros modelos implementados.

Conclusão

O experimento foi realizado de forma quase satisfatória visto que, como mencionado anteriormente, não foi possível fazer uma análise mais aprofundada de qual combinação de informação daria o melhor resultado nas predições. Outra limitação encontrada foi a dificuldade encontrada no tratamento dos dados para o processamento dos mesmo além da limitação do uso de algoritmos supervisionados, visto que provavelmente modelos não supervisionados conseguiriam ter uma acurácia melhor. Contudo, os resultados, mesmo se considerarmos que a análise foi feita de forma superficial, parecem demonstrar que algo nessa linha é possível de se fazer.