PSI 3571

P1 - Relatório

Aluno: Rodrigo Perrucci Macharelli - 9348877

Ênfase: Engenharia de Computação (semestral)

1. Regressor

1.1 Proposta

A proposta de regressor para esta P1 é a de um modelo que se utiliza de redes neurais artificiais para prever notas em provas baseado em um vetor de entrada de 32 elementos contendo informações gerais sobre o aluno.

As informações contidas nos dados de entrada em relação aos alunos são referentes a sua vida pessoal (endereço, estado de coabitação dos pais, trabalho do pai e da mãe, etc), atividades extra acadêmicas (relacionamentos, atividades, tempo livre fora da escola, etc) e atividades acadêmicas (faltas, notas das provas anteriores, reprovações, etc), tentando dar uma visão ampla sobre diversos aspectos que podem influenciar na nota de um aluno.

A saída do modelo do regressor representa a nota esperada para a nota final do aluno. Esse valor representa um número real entre 0 e 20.

O regressor foi implementado utilizando redes neurais artificiais com a técnica de *Deep Learning*, utilizando-se de um modelo com 5 camadas internas.

1.2 Dados e Processamento

Os dados foram obtidos a partir da base de dados da UCI, disponível em (http://archive.ics.uci.edu/ml/datasets/Student+Performance).

Consiste de diversos dados sobre alunos obtidos para alunos do ensino médio de escolas portuguesas, possuindo dois grupos de dados, sendo um para aulas de português e outro para aulas de matemática. Nota-se que há alunos matriculados em ambas as turmas.

Cada aluno é representado por 33 atributos, sendo eles de diferentes tipos, contendo valores binários, numéricos e nominais, a descrição de cada atributo e seus possíveis valores pode ser visto na tabela a seguir.

Atributos da Base de Dados

1. **school** - student's school (binary: "GP" - Gabriel Pereira or "MS" - Mousinho da Silveira)
2. **sex** - student's sex (binary: "F" - female or "M" - male)
3. **age** - student's age (numeric: from 15 to 22)
4. **address** - student's home address type (binary: "U" - urban or "R" - rural)
5. **famsize** - family size (binary: "LE3" - less or equal to 3 or "GT3" - greater than 3)
6. **Pstatus** - parent's cohabitation status (binary: "T" - living together or "A" - apart)
7. **Medu** - mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)
8. **Fedu** - father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)
9. **Mjob** - mother's job (nominal: "teacher", "health" care related, civil "services" (e.g. administrative or police), "at_home" or "other")
10. **Fjob** - father's job (nominal: "teacher", "health" care related, civil "services" (e.g. administrative or police), "at_home" or "other")
11. **reason** - reason to choose this school (nominal: close to "home", school "reputation", "course" preference or "other")
12. **guardian** - student's guardian (nominal: "mother", "father" or "other")
13. **traveltime** - home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)
14. **studytime** - weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)
15. **failures** - number of past class failures (numeric: n if 1<=n<3, else 4)
16. **schoolsup** - extra educational support (binary: yes or no)
17. **famsup** - family educational support (binary: yes or no)
18. **paid** - extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)
19. **activities** - extra-curricular activities (binary: yes or no)
20. **nursery** - attended nursery school (binary: yes or no)
21. **higher** - wants to take higher education (binary: yes or no)
22. **internet** - Internet access at home (binary: yes or no)
23. **romantic** - with a romantic relationship (binary: yes or no)
24. **famrel** - quality of family relationships (numeric: from 1 - very bad to 5 - excellent)
25. **freetime** - free time after school (numeric: from 1 - very low to 5 - very high)
26. **goout** - going out with friends (numeric: from 1 - very low to 5 - very high)
27. **Dalc** - workday alcohol consumption (numeric: from 1 - very low to 5 - very high)
28. **Walc** - weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)
29. **health** - current health status (numeric: from 1 - very bad to 5 - very good)
30. **absences** - number of school absences (numeric: from 0 to 93)
31. **G1** - first period grade (numeric: from 0 to 20)
32. **G2** - second period grade (numeric: from 0 to 20)
33. **G3** - final grade (numeric: from 0 to 20, output target)

A variável a ser obtida como saída do modelo (G3) representa a nota final do aluno, equivalente à terceira prova.

Para o pré-processamento dos dados, inicialmente foi feita a junção das tabelas para as matérias de português e matemática de forma a obter em apenas um dataset todas as informações.

```
# Junta as tabelas de estudantes
d1 = pd.read_csv("student-mat.csv", sep=";")
d2 = pd.read_csv("student-por.csv", sep=";")
dataset = pd.concat([d1, d2])
```

Com os dados obtidos efetua-se a transformação dos dados não numéricos de forma a representar todos os valores possíveis sem fazer com que o modelo entenda-os com diferentes pesos ou importância. Isso é feito pela técnica *One Hot Encoding*.

Esta técnica consiste em criar uma nova coluna para cada possível valor categórico, apenas utilizando o valor **1** para a coluna que representa o valor do atributo em questão.

O vetor de entrada passa de 32 valores para 58 com a expansão gerada pelo *One Hot*.

```
# Efetua OneHotEncoding
cols_to_encode = [0, 1, 3, 4, 5, 8, 9, 10, 11, 15, 16, 17, 18, 19,
20, 21, 22] # Colunas com valores binários ou categoricos
ct = ColumnTransformer([('one_hot_encoder', OneHotEncoder(),
cols_to_encode)],remainder='passthrough')
dataset=pd.DataFrame(ct.fit_transform(dataset))
```

Após a codificação da tabela separam-se os dados utilizados para treinamento e teste do modelo. Foram utilizados 80% dos dados para treinamento e o restante para testes.

```
# Separa entre train e test
train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)
```

Com os grupos separados, obtém-se os valores de entrada e saída respectivos para cada grupo. Os dados de entrada são formados por um vetor de tamanho 58 e o dado de saída a ser estimado é apenas um valor numérico.

```
# Separa entre train e test
train_dataset = dataset.sample(frac=0.8, random_state=0)
test_dataset = dataset.drop(train_dataset.index)
```

1.3 Implementação do Modelo

O modelo do regressor foi montado utilizando o framework *TensorFlow* (https://www.tensorflow.org/) em conjunto com a biblioteca *Keras*.

Trata-se de uma rede neural artificial simples, que implementa o conceito de *Deep Learning* como sua estrutura.

A rede é montada diretamente com auxílio das bibliotecas mencionadas anteriormente, podendo-se montar em formato de camadas sequenciais totalmente conectadas.

A rede possui 58 neurônios na entrada e 1 de saída, contendo 5 camadas intermediárias com 32, 64, 128, 64 e 32 neurônios cada.

Essa configuração foi estabelecida com base em testes de diversas configurações, verificando o modelo que tinha melhores resultados sobre o grupo de testes.

Cada uma das camadas utiliza a Retificação Linear (ReLU) como função de ativação.

```
# Cria o modelo de regressao
model = keras.Sequential([
    layers.Dense(32, activation='relu',
input_shape=[X_train.shape[1]]),
    layers.Dense(64, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(1)
])
```

Após a definição da estrutura da rede neural, é definido o otimizador, neste caso utilizado o chamado *Adam*, uma extensão do algoritmo do Gradiente Estocástico Descendente, utilizado de forma ampla e disponibilizado pelo framework. Utiliza taxa de aprendizado de 0.1%, valor também obtido a partir de testes práticos e comparações de desempenho.

O modelo é então compilado utilizando como função de perda (*loss)* o erro absoluto médio, de forma a fazer com que o modelo entenda todos os erros (distância do valor esperado para o estimado) com o mesmo peso, algo que não acontece no caso de utilizar o Erro Quadrático Médio, por exemplo.

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss="mae", optimizer=optimizer, metrics=["mae",
"mse"])
```

Com o modelo pronto basta efetuar o treinamento sobre os dados separados para tal finalidade.

1.4 Testes

Inicialmente efetua-se o treino do modelo, utilizando o grupo de dados montado anteriormente para este propósito, contendo 80% dos dados do dataset original.

```
#Efetua o treinamento da rede neural
model.fit(X_train, Y_train, epochs=600, validation_split=0.2,
verbose=0)

# Verifica acuracia com dataset de testes
loss, mae, mse = model.evaluate(X_test, Y_test, verbose=0)
```

O treinamento da rede é realizado 600 vezes e, em seguida, mostram-se os dados de qualidade do modelo, sendo estes o Erro Médio Absoluto e o Erro Quadrático Médio.

Esses valores são mostrados a seguir, juntamente com alguns exemplos de saídas comparadas com os valores esperados.

```
# Valores do treinamento
Loss:  1.1616320929458837 Mae:  1.1616321 Mse:  3.5699527
# Valores de exemplos
Valor original:  12.0 Valor do regressor:  12.386049
Valor original:  10.0 Valor do regressor:  10.011179
Valor original:  12.0 Valor do regressor:  12.7757
Valor original:  8.0 Valor do regressor:  9.597215
Valor original:  11.0 Valor do regressor:  10.346164
Valor original:  16.0 Valor do regressor:  17.137714
Valor original:  18.0 Valor do regressor:  15.131694
Valor original:  12.0 Valor do regressor:  11.941311
Valor original:  18.0 Valor do regressor:  18.042078
Valor original:  0.0 Valor do regressor:  0.04118424
Valor original:  0.0 Valor do regressor:  8.001872
```

Os resultados mostram, em sua maioria, um bom funcionamento do modelo, não deixando, porém, de apresentar erros, algumas vezes bastante expressivos.

Estes erros mais contundentes foram verificados em casos extremos, como por exemplo no último mostrado na imagem anterior, no qual a nota do aluno foi 0 e o modelo gerou um valor próximo de 8.

Este tipo de caso pode ser explicado por casos fora do comum, no qual o aluno pode ter tido problemas pontuais para realizar a prova, ou não tê-la feito, por exemplo.

2. Reconhecedor

2.1 Proposta

O reconhecedor implementado baseia-se em um problema de classificação de sons ambientes urbanos, podendo se enquadrar em 10 tipos diferentes de classes, ou seja, um classificador multiclasse.

Sua implementação foi feita com base em um modelo de redes neurais artificias de múltiplas camadas, *Deep Learning*, utilizando-se do framework *TensorFlow* em conjunto com *Keras*.

Dentre as classes de tipos de sons reconhecidos pelo modelo estão: latido de cachorro, sons de tiro, sirenes, música de rua e etc.

A inspiração para escolha deste projeto para o reconhecedor veio do fato de como os dados devem ser tratados para serem inseridos na rede neural, tratando-se de áudios e não valores tabelados, como foi o caso do regressor descrito na parte 1 do relatório. Mais detalhes sobre o procedimento adotado para tal estão descritos na próxima sessão.

2.2 Dados e Processamento

Os dados foram obtidos do *UrbanSound8K - Urban Sound Dataset*, contendo 8732 trechos de sons urbanos captados, com cerca de 4 segundos cada, em formato *WAV*.

Os dados estão dispostos em 10 pastas contendo arquivos de diversas classes em cada uma delas. É fornecida também uma tabela em formato *CSV* contendo as descrições de cada áudio (caminho do arquivo, classe correspondente, etc), facilitando o processamento.

As classes possíveis a serem reconhecidas são as seguintes:

0. air_conditioner 1. car_horn 2. children_playing 3. dog_bark 4. drilling	5. engine_idling 6. gun_shot 7. jackhammer 8. siren 9. street_music

Os dados de entrada do modelo neste caso não são tão óbvios quanto no caso do regressor e foram obtidos através do tratamento e extração de *Features* dos áudios.

Para isso utilizou-se da biblioteca *Librosa*, especializada em análise de áudio. Com ela pode-se obter dados característicos do áudio em questão, gerando valores numéricos que representam-no de alguma forma.

Para a caracterização dos áudios para este modelo, foram utilizadas 5 features para cada áudio, cada qual com 40 amostras, resultando em um vetor de entrada de 200 (40*5) valores numéricos capazes de representar a amostra de áudio.

Os *features* utilizados foram: MFCCS, melspectrogram, Chroma Stft (Power Spectrogram), Chroma Cqt e Chroma Cens (Chroma Energy Normalized).

Na próxima seção será mostrada a diferença de resultados utilizando apenas um feature para representação do excerto de áudio em contrapartida com a utilização dos 5 citados acima.

Os trechos de código referentes ao pré processamento dos dados pode ser visto a seguir.

Inicialmente os metadados são obtidos a partir da tabela fornecida e são criados os grupos de testes e treinamento de dados, que serão posteriormente preenchidos conforme os dados são processados.

```
# Cria o dataframe do csv contendo os metadados
data=pd.read_csv("UrbanSound8K/metadata/UrbanSound8K.csv")
# Montando os vetores de treino e teste
x_train=[]
x_test=[]
y_train=[]
y_test=[]
```

Com isto pronto pode-se efetuar o processamento de fato. O código comentado a seguir mostra as etapas envolvidas nesse processo.

```
# Leitura e preparação dos dados a partir da leitura dos metadados
path="UrbanSound8K/audio/fold"
for i in tqdm(range(len(data))):
    fold_no=str(data.iloc[i]["fold"]) # Obtém a pasta do áudio
    file=data.iloc[i]["slice_file_name"]  # Obtém o nome do áudio
    label=data.iloc[i]["classID"] # Obtém a classe a qual o áudio
pertence
    filename=path+fold_no+"/"+file # Gera o nome completo do arquivo

    y,sr=librosa.load(filename) # Carrega o arquivo de áudio

    # Extração dos features
```

```
    mfccs = np.mean(librosa.feature.mfcc(y, sr, n_mfcc=40).T,axis=0)
    melspectrogram = np.mean(librosa.feature.melspectrogram(y=y,
sr=sr, n_mels=40,fmax=8000).T,axis=0)
    chroma_stft=np.mean(librosa.feature.chroma_stft(y=y,
sr=sr,n_chroma=40).T,axis=0)
    chroma_cq = np.mean(librosa.feature.chroma_cqt(y=y,
sr=sr,n_chroma=40).T,axis=0)
    chroma_cens = np.mean(librosa.feature.chroma_cens(y=y,
sr=sr,n_chroma=40).T,axis=0)

features=np.reshape(np.vstack((mfccs,melspectrogram,chroma_stft,chro
ma_cq,chroma_cens)),(40,5))

    # Separação da última pasta para testes
    if(fold_no!='10'):
      x_train.append(features)
      y_train.append(label)
    else:
      x_test.append(features)
      y_test.append(label)
```

Perceba que neste momento faz-se a separação dos dados de teste e treinamento conforme a indicação do próprio fornecedor dos dados, isto é realizar *10-fold Cross Validation* com uma pasta para treino. Nota-se também que durante a implementação do modelo tentou-se uma técnica similar baseada nesta indicação.

Os dados processados resultam, após breve tratamento dos arrays, em vetores de entrada com 200 valores numéricos para cada arquivo de áudio. Estes valores foram salvos em tabelas no formato CSV para serem consumidos posteriormente pelo modelo.

```
# Gera numpy arrays a partir dos dados para serem salvos
x_train=np.array(x_train)
x_test=np.array(x_test)
y_train=np.array(y_train)
y_test=np.array(y_test)

x_train_2d=np.reshape(x_train,(x_train.shape[0],x_train.shape[1]*x_t
rain.shape[2]))
```

```
x_test_2d=np.reshape(x_test,(x_test.shape[0],x_test.shape[1]*x_test.
shape[2]))

# Cria os arquivos tratados
np.savetxt("train_data.csv", x_train_2d, delimiter=",")
np.savetxt("test_data.csv",x_test_2d,delimiter=",")
np.savetxt("train_labels.csv",y_train,delimiter=",")
np.savetxt("test_labels.csv",y_test,delimiter=",")
```

Vale ressaltar que este passo de pré-processamento é custoso computacionalmente. Para que fosse possível realizá-lo foi utilizado o ambiente do *Google Colab* (https://colab.research.google.com/), que permite a utilização de máquinas virtuais para processamento, além de já incluir nativamente a biblioteca *Librosa* para desenvolvimento em python, não requerindo a instalação da mesma.

Ao final do processo de processamento dos dados têm-se quatro tabelas representando os dados de entrada (X) de teste e treinamento e os dados de saída (Y) também de teste e treinamento.

2.3 Implementação do Modelo

O processo de implementação deste modelo foi feito de maneira incremental, inicialmente utilizando apenas um feature para caracterização dos áudios (devido ao custo computacional para processá-los) e em sequência com 5 features.

Além disso foi testada uma forma de geração do modelo baseado em votos e o modelo K-Fold. A idéia básica foi utilizar 9-Fold Cross Validation para gerar 9 modelos diferentes. A pasta restante foi utilizada apenas para testes e o resultado do modelo é baseado na resposta com maior número de votos dentre os 9 modelos gerados.

De início os dados são carregados a partir das tabela CSV geradas durante o processamento dos dados.

```
# Carrega os dados pre processados
x_train = genfromtxt('train_data.csv', delimiter=',')
y_train = genfromtxt('train_labels.csv', delimiter=',')
x_test = genfromtxt('test_data.csv', delimiter=',')
y_test = genfromtxt('test_labels.csv', delimiter=',')
```

Neste momento os dados representando os vetores y contém valores de 0 a 9, sendo assim os valores são transformados de forma semelhante à técnica *One Hot Encoding* mencionada na parte 1 do relatório.

```
# Transformação dos dados para vetores categoricos de 10 classes
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

No caso do classificador utilizando apenas um feature para identificar os áudios, basta criar o modelo de rede neural e realizar os testes.

A rede apresenta 3 camadas intermediárias com 256 nós cada, além da entrada de 40 nós (apenas uma feature) e saída com 10 nós (representando as 10 possíveis classes).

```
# Criação do modelo
model = Sequential()
model.add(Dense(units=256, activation='relu', input_dim=40))
model.add(Dropout(0.4))
model.add(Dense(units=256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(units=256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(units=10, activation='softmax'))
```

Para a implementação do classificador com votos utilizando 9-Fold, necessita-se da criação de um modelo para cada arranjo das possibilidades de treinamento. O código que implementa esta funcionalidade pode ser visto a seguir e se baseia em uma função da biblioteca *Scikit-Learn* com o nome *K-Fold*.

```
models = []
mean_acc_train = []
mean_acc_test = []

# Utilização do método K-Fold para gerar 9 modelos distintos
# Os dados de teste y_test referentes à pasta número 10 não sao
utilizados para  treino
for train_index, test_index in KFold(9).split(x_train):
    # Criação do modelo
    model = Sequential()
    model.add(Dense(units=256, activation='relu', input_dim=200))
    model.add(Dropout(0.4))
    model.add(Dense(units=256, activation='relu'))
    model.add(Dropout(0.4))
```

```
    model.add(Dense(units=256, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(units=10, activation='softmax'))

    # Compilação e treinamento
    model.compile(optimizer='adam', loss='categorical_crossentropy',
            metrics=['accuracy'])

    # Obtenção dos grupos de teste e treinamento
    X_train, X_test = x_train[train_index], x_train[test_index]
    Y_train, Y_test = y_train[train_index], y_train[test_index]

    # Treinamento do modelo

model.fit(X_train,Y_train,epochs=50,validation_data=(X_test,Y_test),
batch_size=50, verbose=0)

    # Medidas de desempenho
    train_loss_score = model.evaluate(X_train, Y_train)
    test_loss_score = model.evaluate(X_test, Y_test)
    print(f"TREINO: Acuracia: {train_loss_score[1]} - Loss
{train_loss_score[0]}")
    print(f"TESTE: Acuracia: {test_loss_score[1]} - Loss
{test_loss_score[0]}")

    mean_acc_train.append(train_loss_score[1])
    mean_acc_test.append(test_loss_score[1])

    # Guarda o modelo gerado
    models.append(model)
```

Neste caso é criado um modelo para cada configuração dos dados de teste e treinamento, sendo que os dados utilizados para o K-Fold não incluem os dados previamente selecionados para o grupo de teste, isto é, os dados pertencentes à pasta número 10 do dataset.

Além disso os modelos gerados apresentam 200 nós na camada de entrada do modelo.

Os modelos são guardados em um array para serem posteriormente utilizados em um sistema de votação cuja funcionalidade é bastante básica: cria-se um vetor de tamanho 10

(referente às 10 classes) inicializado com zeros; Para cada modelo guardado efetua-se a classificação com o modelo, e sua resposta é adicionada ao vetor de votação incrementando 1 no valor contido no índice correspondente da classe obtida como resposta; Por fim verifica-se qual índice (correspondente a classe) tem o maior número de votos, representando a classe escolhida.

```
# votacao
random_audios = np.random.randint(0, len(x_test), 100)
acertos = 0

for random_audio in random_audios:
    votacao = [0 for _ in range(10)]
    for model in models:
        vote =
np.argmax(model.predict(np.array([x_test[random_audio]])))
        votacao[vote] += 1

    if np.argmax(votacao) == np.argmax(y_test[random_audio]):
        acertos += 1

    print(f"Resultado: {np.argmax(votacao)} - Label:
{np.argmax(y_test[random_audio])}")

print(f"Acertos: {acertos/len(random_audios)}")
```

Para o classificador foram utilizadas as funções de ativação ReLU e Softmax, sendo a segunda bastante utilizada para reconhecedores, uma vez que as saídas se comportam como um distribuição de probabilidade cuja soma resulta 1. Sendo assim muito útil para avaliar qual das saídas (classes) é a mais provável de corresponder à classe real do item sendo analisado.

2.4 Testes

Para os testes com o classificador com apenas uma feature têm-se os seguintes resultados e alguns exemplos de classificações:

```
# Métricas
TREINO: Acuracia: 0.9423685877137429 - Loss 0.20084473463215663
TESTE: Acuracia: 0.5746714457103999 - Loss 1.3989687001548503

# Exemplos
Label:5 - Classificacao: 2
Label:5 - Classificacao: 2
Label:5 - Classificacao: 2
Label:0 - Classificacao: 2
Label:3 - Classificacao: 3
Label:3 - Classificacao: 3
Label:3 - Classificacao: 3
Label:3 - Classificacao: 3
Label:3 - Classificacao: 3
Label:3 - Classificacao: 3
label:9 - Classificacao: 0
Label:9 - Classificacao: 9
Label:9 - Classificacao: 9
Label:9 - Classificacao: 2
Label:9 - Classificacao: 2
Label:2 - Classificacao: 2
Label:2 - Classificacao: 2
Label:2 - Classificacao: 2
Label:2 - Classificacao: 8
Label:4 - Classificacao: 4
Label:9 - Classificacao: 9

# Valores sobre o conjunto de testes
Acuracia (50 pares): 54.90196078431373%
Acuracia (1000 pares): 57.46714456391876%
```

Com este modelo simples, e com a caracterização dos áudios utilizando apenas um feature obteve-se acurácia de aproximadamente 56%, o que é um valor aceitável, uma vez que a classificação aleatória destes itens teria acurácia esperada de 10% apenas.

Verifica-se agora os resultados para o classificador com votação e 5 features caracterizando os áudios.

Neste caso temos um total de 9 modelos gerados, cada um com suas respectivas métricas de acurácia durante o treinamento e testes, assim como uma média de todos os valores, representando a acurácia do modelo de votação

```
# Valores para os modelos separados
TREINO: Acuracia: 0.9558215761806477 - Loss 0.14143960449404053
TESTE: Acuracia: 0.41002277904328016 - Loss 3.8564476149357083

TREINO: Acuracia: 0.9576742197605251 - Loss 0.13995784586448148
TESTE: Acuracia: 0.5011389521640092 - Loss 2.8051559680729152

TREINO: Acuracia: 0.9554003989740667 - Loss 0.15199750558808475
TESTE: Acuracia: 0.508551881413911 - Loss 1.8428336749983762

TREINO: Acuracia: 0.9534055286406383 - Loss 0.14676580721015522
TESTE: Acuracia: 0.5370581527936146 - Loss 2.500124648604181

TREINO: Acuracia: 0.9588201766885153 - Loss 0.13282378623144706
TESTE: Acuracia: 0.44469783379523214 - Loss 2.720414173847729

TREINO: Acuracia: 0.96337988030778 - Loss 0.127705654678699
TESTE: Acuracia: 0.5210946408209807 - Loss 2.336384670197233

TREINO: Acuracia: 0.9655172413793104 - Loss 0.12114269077349224
TESTE: Acuracia: 0.5416191564862242 - Loss 2.2797659200389457

TREINO: Acuracia: 0.965944713559644 - Loss 0.11823339989558997
TESTE: Acuracia: 0.49828962412500327 - Loss 2.7277805858218414

TREINO: Acuracia: 0.9626674266342561 - Loss 0.12155091995405715
TESTE: Acuracia: 0.47434435596215957 - Loss 2.2538676792022985
```

```
# Média dos valores dos modelos
TREINO: Acuracia: 0.9598479069028204
TESTE: Acuracia: 0.49297970851160167
```

Analisando os modelos de forma independente percebe-se uma perda na acurácia dos modelos, devido a menor quantidade de dados para o treinamento (8 neste caso e 9 no caso anterior).

Porém fazendo o teste de classificação com o sistema de votação sobre o conjunto de dados de teste separado inicialmente, obtém-se acurácia de 66% sobre 100 exemplos.

Este resultado, assim como exemplos de classificação do modelo de votação podem ser vistos a seguir

```
# Exemplos com o modelo de votação
Resultado: 3 - Label: 3
Resultado: 7 - Label: 7
Resultado: 9 - Label: 0
Resultado: 7 - Label: 7
Resultado: 2 - Label: 9
Resultado: 1 - Label: 8
Resultado: 4 - Label: 9
Resultado: 4 - Label: 4
Resultado: 1 - Label: 1
Resultado: 6 - Label: 6
Resultado: 7 - Label: 7
Resultado: 2 - Label: 2
Resultado: 2 - Label: 2
Resultado: 7 - Label: 0

# Acurácia modelo de votação
Acurácia: 0.66
```

Sendo assim, a acurácia do modelo de votação com 5 features para sua entrada resultam numa taxa de acertos 10% maior do que no caso anterior.

Vale reforçar novamente que em ambos os casos os exemplos utilizados para as métricas dos modelos partem das mesmas pastas do dataset original, que por sua vez não foram utilizadas em nenhum momento pelos modelos para treinamento, sendo totalmente novos para este.

Bibliografia

1. Dataset Regressor:
   P. Cortez and A. Silva. Using Data Mining to Predict Secondary School Student Performance. In A. Brito and J. Teixeira Eds., Proceedings of 5th FUture BUsiness TEChnology Conference (FUBUTEC 2008) pp. 5-12, Porto, Portugal, April, 2008, EUROSIS, ISBN 978-9077381-39-7.
   http://archive.ics.uci.edu/ml/datasets/Student+Performance

2. TensorFlow
   https://www.tensorflow.org/
   https://www.tensorflow.org/guide/keras
3. Deep Learning
   https://machinelearningmastery.com/what-is-deep-learning/
4. Retificador Linear (ReLU)
   https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning
   https://www.tensorflow.org/api_docs/python/tf/keras/activations/relu
5. Adam Optimizer
   https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
6. Erro Quadratico Médio vs Erro bsoluto Médio
   https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d
7. UrbanSound8K - Urban Sound Dataset
   https://urbansounddataset.weebly.com/urbansound8k.html
8. Librosa
   https://librosa.github.io/librosa/
9. MFCCS
   https://en.wikipedia.org/wiki/Mel-frequency_cepstrum
10. Melspectrogram
    https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0
11. Chroma Stft
    https://librosa.github.io/librosa/generated/librosa.feature.chroma_stft.html
12. Chroma Cqt
    https://librosa.github.io/librosa/generated/librosa.feature.chroma_cqt.html
13. Chroma Sens
    https://librosa.github.io/librosa/generated/librosa.feature.chroma_cens.html
14. Google Colab
    https://colab.research.google.com/