MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENG
ENGENHARIA ELÉTRICA, MECÂNICA E DE COMPUTAÇÃO

Letícia Delfino Teixeira

PROBLEMA DE CLASSIFICAÇÃO DE VINHOS

Goiânia
2019

Introdução

Este trabalho tem como objetivo resolver o problema de classificação de vinhos.utilizando redes neurais.

Para realização deste trabalho foi utilizado o framework “Kaggle” e a base de dados “Classifying wine varieties”.

Informações do Problema

A base de dados "Classifying wine varieties" foi obtida através de resultados de análise química de vinhos cujas uvas cresceram em uma mesma região da Itália, mas que são derivadas de três cultivos diferentes. A análise feita determinou uma quantidade de 13 constituintes encontrados nos três tipos de vinhos.

O código utilizado para resolução está no github e pode ser encontrado aqui.

Objetivo

O objetivo é construir uma rede neural que possa classificar o tipo de vinho com base em suas características utilizando uma rede neural.

Classes

Classe	Valor correspondente
Classe 1	59
Classe 2	71
Classe 3	48

Caracterísitcas

Características	
1. Álcool	2. Fenol não flavonóide
3. Ácido málico	4. Proantocianimas
5. Cinza	6. Intensidade da cor

7. Alcalinidade das cinzas	8. Tom
9. Magnésio	10. OD280/OD315 de vinhos diluídos
11. Fenóis totais	12. Prolina
13. Flavonóides	

Amostras

O arquivo utilizado para retirar as amostras é o arquivo “Wine.csv” que se encontra na pasta deste documento.

```
#mostrando arquivo que está sendo usado
print(check_output(["ls", "../input"]).decode("utf8"))
```

```
Wine.csv
```

A tabela possui 178 amostras e 13 características, gerando um total de 178 linhas e 14 colunas. O conjunto de amostras com suas respectivas características pode ser visto abaixo:

```
#incianco leitura do arquivo
dados = pd.read_csv('../input/Wine.csv', header=None)
dados.columns = ['Classe','Álcool','Ácido málico','Cinza','Alcalinidade das cinzas','Magnésio','totalPhenols','Fenóis
print(dados)
```

```
    Classe  Álcool       ...        OD280/OD315  prolineProlina
0        1   14.23       ...               3.92            1065
1        1   13.20       ...               3.40            1050
2        1   13.16       ...               3.17            1185
3        1   14.37       ...               3.45            1480
4        1   13.24       ...               2.93             735
5        1   14.20       ...               2.85            1450
6        1   14.39       ...               3.58            1290
7        1   14.06       ...               3.58            1295
8        1   14.83       ...               2.85            1045
9        1   13.86       ...               3.55            1045
10       1   14.10       ...               3.17            1510
```

RNA

Dos grupos criados

Para realizar o treino e o teste da rede neural foram utilizadas todas as amostras da planilha.

Foram criados dois grupos para treinamento da rede. O primeiro diz respeito às características das amostras e vinho. São 13 características. Resultando em 178 amostras com 13 características
O segundo grupo diz respeito a saída esperada. A saída corresponde a classe da amostra de vinho, resultando em um conjunto com 178 linhas e uma única coluna.

```
#verificando tamanho dos conjuntos de treino
print(X.shape)
print(Y.shape)
```

```
(178, 13)
(178, 1)
```

Conversão de Dados

Como a rede neural não vai receber as classes no formato que fornecido pela planilha (1,2,ou 3), foi necessário criar uma função simples que realiza a conversão dos valores para uma matriz de três colunas que vai ser lida e interpretada pela rede neural.
A função consiste de três condições que geram a matriz com base na classe fornecida juntamente os demais dados e que foi salva no array de saída (Y).
As matrizes são salvas novamente no vetor de saída (Y) agora ,porém, com os valores desejado.

```
#criando a representação das classes
def classificador(classificacao):
    if classificacao == 1:
        return [1, 0, 0] #equivalente a classe um
    if classificacao == 2:
        return [0, 1, 0] #equivalente a classe dois
    if classificacao == 3:
        return [0, 0, 1] #equivalente a classe três
```

```
#salvando novas valores gerados pela função de classificação criada no array Y

Y = np.array([classificador(i[0]) for i in Y])
```

Dos Modelos

Para a criação do modelo foram utilizadas três camadas de de neurônios. A primeira camada possui 12 neurônios e sua função de ativação ‘relu’. A segunda camada possui 8 neurônios e também utiliza a função de ativação ‘relu’. Essa função foi escolhida por se tratar de uma função de fácil otimização.
A terceira camada possui 3 neurônios e sua função de ativação é a função ‘softmax’. Como o problema possui 3 classes foi necessário a utilização de 3 neurônios na camada de saída. Um determinado neurônio só será ativado se a saída gerada corresponder a sua respectiva classe. A escolha da função ‘softmax’ como função de ativação se deve ao fato dela ser

perfeita para problemas de classificação, uma vez que ela força a saída da rede neural representar a probabilidade do resultado encontrado pertencer a uma determinada classe.

```
#criando modelo
model = Sequential([
    Dense(12, input_dim=13, activation='relu'),
    Dense(8, input_dim=13, activation='relu'),
    Dense(3, activation='softmax')
              ])
```

Na compilação do modelo foi utilizada uma taxa de aprendizado baixa para que a rede de passos menores até encontrar o resultado na expectativa do resultado ser encontrado mesmo que lentamente. A função SGD corresponde ao Otimizador Estocástico de Gradiente Descendente.

```
#compilando modelo

sgd = optimizers.SGD(lr=0.001)
model.compile(
  optimizer='sgd',
  loss='categorical_crossentropy',
  metrics=['accuracy'],
)
```

Para a realização do treinamento foram utilizadas todas as amostras o número de épocas foi fixado em 100.

```
#treinando a rede usando o arquivo todo
model.fit(X,Y,epochs=1000, batch_size=178)
```

```
Epoch 8/1000
178/178 [==============================] - 0s 15us/step - loss: 9.6890 - acc: 0.3989
Epoch 9/1000
178/178 [==============================] - 0s 22us/step - loss: 9.6890 - acc: 0.3989
Epoch 10/1000
178/178 [==============================] - 0s 12us/step - loss: 9.6890 - acc: 0.3989
Epoch 11/1000
178/178 [==============================] - 0s 12us/step - loss: 9.6890 - acc: 0.3989
Epoch 12/1000
178/178 [==============================] - 0s 14us/step - loss: 9.6890 - acc: 0.3989
Epoch 13/1000
178/178 [==============================] - 0s 12us/step - loss: 9.6890 - acc: 0.3989
```

Resultados

```
predictions = model.predict(X)
print("predictions",predictions[0])
```

```
predictions [0. 1. 0.]
```

```
score = model.evaluate(X,Y,verbose=0)
print(score)
```

```
[9.688967544041322, 0.398876404494382]
```