Análise da situação de alunos matriculados na disciplina LoP utilizando MLP

Introdução

Este trabalho foi desenvolvido por Igor Carvalho de Brito Batista, Rony de Sena Lourenço e Thatiana Jéssica da Silva Ribeiro.

O objetivo se trata de analisar a probabilidade de aprovação de alunos da turma de Lógica de Programação da Escola de Ciências e Tecnologia (ECT) da UFRN. Para realizar determinada análise, foi utilizado uma base de dados contendo informações dos semestres 2017.2, 2018.1, 2018.2 e 2019.1.

Metodologia

O modelo de aprendizado de máquina utilizado neste trabalho foi o Multilayer Perceptron, algoritmo responsável pelo aprendizado de rede.

A partir do funcionamento dos neurônios biológicos do sistema nervoso animal, foi estabelecido na área da Inteligência Artificial um modelo computacional de um neurônio. Com apenas um neurônio, pouca coisa se pode fazer, mas os combinando em uma estrutura em camadas, em cada uma terá um número diferente de neurônios, teremos assim a formação de uma rede.

O vetor de valores de uma determinada entrada passa por uma camada inicial chamada input layer, que é a responsável por encaminhar esse vetor para uma outra camada chamada head layer, responsável por produzir um número de hiperplanos no espaço. Logo após, é encaminhado para a camada de saída chamada output layer, responsável por mostrar o resultado.

Para que esse modelo funcione, assim como qualquer outro modelo em rede, é necessário passar pela etapa do treinamento. Essa etapa tem por objetivo fazer com que o modelo aprenda os padrões a partir de uma determinada amostra, dessa forma, quando um dado desconhecido for fornecido à rede, ela será apta à designar a qual classe a amostra pertence. Depois, foi possível fazer a validação do resultado deste algoritmo com os dados de teste, durante esta fase, foi possível determinar o funcionamento da rede com alguns dados que não foram utilizados na etapa de treinamento.

Como o intuito é prever a probabilidade de aprovação, então buscamos selecionar os atributos que ajudassem o modelo a entender o comportamento de alunos exemplares e, dessa forma, prever com uma maior porcentagem de acertos.

Códigos

Para o desenvolvimento da atividade, foi utilizado o código cedido pelo professor em aula. O código faz uso das bibliotecas numpy, matplotlib, scikit-learn, keras e pandas.

A biblioteca pandas foi utilizada para importar o dataset em csv contendo todos os dados referente a notas, quantidade de submissões e situação dos alunos. Em seguida, as colunas foram selecionadas de acordo com a necessidade e houve a separação do conjunto de teste. Todos os dados da tabela passaram por uma normalização para que os mesmos se apresentassem em ordem de grandeza próxima e não causasse problemas decorrentes de valores extremos.

```
[ ]  X = dataset.iloc[:,[2,43,44,45,46,47,48,49,50,51]].values
     y = dataset.iloc[:, 11].values

     print(X[0:6,:])
```

```
[[24.  8.  4.  2.  0.  0. 10.  5. 18.  0.]
 [72.  1.  0.  3.  0.  0. 12.  3. 10.  0.]
 [36.  0.  0.  0.  1.  0.  0.  2.  3.  6.]
 [ 4.  1.  1.  1.  3.  0.  5.  5.  3.  0.]
 [16.  1.  1.  0.  3.  0.  5.  5.  7.  0.]
 [52.  3.  1.  0.  2.  0.  7.  4. 10.  0.]]
```

```
[ ]  # Splitting the dataset into the Training set and Test set
     from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

```
[ ]  # Feature Scaling
     from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

Figura 01 - Escolha das colunas da tabela, separação do conjunto de teste e normalização dos dados.

Na figura a seguir é possível observar a seleção dos parâmetros da rede neural. No primeiro passo é possível criar camadas, dentro de cada camada é possível inserir a quantidade de neurônios que serão utilizados para fazer a filtragem dos dados bem como a função de ativação a ser utilizada. O último trecho do código presente na imagem apresenta a quantidade de épocas passadas por parâmetro. Como resposta do treinamento, são exibidas a acurácia do treinamento e o erro a cada época.

```
[ ]  # Adding the input layer and the first hidden layer
     classifier.add(Dense( activation = 'relu', input_dim = 10, units = 12, kernel_initializer = 'uniform'))


     # Adding the first hidden layer
     classifier.add(Dense( activation = 'relu', units = 10, kernel_initializer = 'uniform' ))

      #Adding the second hidden layer
     #classifier.add(Dense( activation = 'relu', units = 2, kernel_initializer = 'uniform' ))


     # Adding the output layer
     classifier.add(Dense( activation = 'sigmoid', units = 1, kernel_initializer = 'uniform'))
```

```
[ ]
     # Compiling the ANN
     classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
[ ]  # Fitting the ANN to the Training set
     classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

```
Epoch 1/100
312/312 [==============================] - 0s 1ms/step - loss: 0.6930 - acc: 0.5000
Epoch 2/100
312/312 [==============================] - 0s 123us/step - loss: 0.6909 - acc: 0.7404
Epoch 3/100
312/312 [==============================] - 0s 121us/step - loss: 0.6810 - acc: 0.8173
Epoch 4/100
312/312 [==============================] - 0s 122us/step - loss: 0.6529 - acc: 0.8397
Epoch 5/100
312/312 [==============================] - 0s 119us/step - loss: 0.6006 - acc: 0.8365
Epoch 6/100
312/312 [==============================] - 0s 119us/step - loss: 0.5338 - acc: 0.8397
Epoch 7/100
312/312 [==============================] - 0s 121us/step - loss: 0.4685 - acc: 0.8494
Epoch 8/100
312/312 [==============================] - 0s 128us/step - loss: 0.4197 - acc: 0.8494
Epoch 9/100
312/312 [==============================] - 0s 132us/step - loss: 0.3905 - acc: 0.8494
Epoch 10/100
312/312 [==============================] - 0s 123us/step - loss: 0.3747 - acc: 0.8526
Epoch 11/100
312/312 [==============================] - 0s 123us/step - loss: 0.3661 - acc: 0.8526
Epoch 12/100
```

Figura 2 - Parâmetros e treinamento do conjunto de dados.

Por fim, o conjunto de teste é utilizado para fazer uma predição e com base no resultado é possível calcular uma matriz de confusão e a taxa de acerto. A figura abaixo exemplifica o uso do conjunto de teste.

```
[ ]  # Part 3 - Making the predictions and evaluating the model

     # Predicting the Test set results
     y_pred = classifier.predict(X_test)
     print(y_pred[0:10])

     y_pred = (y_pred > 0.5)
     print(y_pred[0:10])

     # Making the Confusion Matrix
     from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(y_test, y_pred)
```

```
[[0.92793417]
 [0.7874966 ]
 [0.8196142 ]
 [0.9827299 ]
 [0.14972156]
 [0.81995237]
 [0.6000233 ]
 [0.46273944]
 [0.9640522 ]
 [0.0460121 ]]
[[ True]
 [ True]
 [ True]
 [ True]
 [False]
 [ True]
 [ True]
 [False]
 [ True]
 [False]]
```

```
[ ]  print("Matriz de Confusão:")
     print(cm)
     print("Taxa de acerto:")
     print((cm[0,0]+cm[1,1])/len(y_test) )
     #print(len(y_test))
```

```
Matriz de Confusão:
[[50 12]
 [17 56]]
Taxa de acerto:
0.7851851851851852
```

Figura 3 - Uso do conjunto de treinamento para cálculo da matriz de confusão e taxa de acerto.

Experimentos

Em um dos testes realizados foram utilizadas as colunas de número 2, 11, 12, 13, 14, 15, 16 e 17 que correspondem, respectivamente, a nota da primeira prova, a situação do aluno, quantidade de submissões da primeira lista, quantidade de submissões da segunda lista, quantidade de submissões da terceira lista,

quantidade de submissões da quarta lista, quantidade de submissões da quinta lista e a quantidade de questões submetidas para prova 1.

Acidentalmente foi colocada a coluna que representa a situação do aluno. Como esperado, a saída da rede apresentou um resultado claramente manipulado: 100% de acurácia, valores baixos para o erro e uma taxa de acerto de 100%.

```
[ ]  # Adding the input layer and the first hidden layer
     classifier.add(Dense( activation = 'relu', input_dim = 8, units = 12, kernel_initializer = 'uniform'))

     # Adding the first hidden layer
     classifier.add(Dense( activation = 'relu', units = 8, kernel_initializer = 'uniform' ))

     # Adding the second hidden layer
     #classifier.add(Dense( activation = 'relu', units = 2, kernel_initializer = 'uniform' ))

     # Adding the output layer
     classifier.add(Dense( activation = 'sigmoid', units = 1, kernel_initializer = 'uniform'))

[ ]  # Compiling the ANN
     classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

[ ]  # Fitting the ANN to the Training set
     classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)

     Epoch 1/100
     357/357 [==============================] - 4s 11ms/step - loss: 0.6928 - acc: 0.6218
     Epoch 2/100
     357/357 [==============================] - 0s 225us/step - loss: 0.6879 - acc: 0.8908
     Epoch 3/100
     357/357 [==============================] - 0s 223us/step - loss: 0.6590 - acc: 0.9552
     Epoch 4/100
     357/357 [==============================] - 0s 243us/step - loss: 0.5797 - acc: 0.9972
     Epoch 5/100
     357/357 [==============================] - 0s 236us/step - loss: 0.4557 - acc: 1.0000
     Epoch 6/100
     357/357 [==============================] - 0s 227us/step - loss: 0.3335 - acc: 1.0000
     Epoch 7/100
     357/357 [==============================] - 0s 229us/step - loss: 0.2365 - acc: 1.0000
     Epoch 8/100
     357/357 [==============================] - 0s 257us/step - loss: 0.1629 - acc: 1.0000
     Epoch 9/100
     357/357 [==============================] - 0s 240us/step - loss: 0.1096 - acc: 1.0000
     Epoch 10/100
     357/357 [==============================] - 0s 244us/step - loss: 0.0743 - acc: 1.0000
     Epoch 11/100
     357/357 [==============================] - 0s 234us/step - loss: 0.0514 - acc: 1.0000
     Epoch 12/100
     357/357 [==============================] - 0s 233us/step - loss: 0.0368 - acc: 1.0000
     Epoch 13/100
```

Figura 4 – Parâmetros da rede e acurácia no primeiro treinamento.

```
[ ]  print("Matriz de Confusão:")
     print(cm)
     print("Taxa de acerto:")
     print((cm[0,0]+cm[1,1])/len(y_test) )
     print(len(y_test))
```

```
Matriz de Confusão:
[[38  0]
 [ 0 52]]
Taxa de acerto:
1.0
90
```

Figura 6 – Resultado da matriz de confusão e taxa de acerto do primeiro treinamento.

A matriz de confusão é responsável por avaliar os resultados e contabilizar a quantidade de verdadeiros positivos, falsos positivos, verdadeiros negativos e falsos negativos. A figura abaixo ilustra a representação de uma matriz de confusão.

Figura 7 – Representação da matriz de confusão.

Na matriz de confusão obtida após o primeiro treinamento é possível observar não há falsos positivos e nem falsos negativos, o que condiz com o treinamento realizado de forma equivocada.

Análise de sensibilidade à mudanças dos parâmetros da rede

Alguns cenários de teste foram proposto utilizando o MLP para este problema de classificação de alunos aprovados/reprovados na disciplina de LoP. Estes cenários são descritos na Tabela X. Foram escolhidos como atributos do problema somente dados referentes ao primeiro mês de aula (submissões de listas e laboratórios) e notas da primeira prova (colunas 2,3,4,12,13,14,15, 42, 47 do csv fornecido).

Tabela 1 - Cenários analisados

	Situação 1	Situação 2	Situação 3	Situação 4	Situação 5	Situação 6	Situação 7
Quantidade de camadas	3	3	3	3	4	4	3
Neurônios por camada	4, 6, 1	8, 12, 1	8, 12, 1	8, 12, 1	8, 12, 24, 1	8, 12, 24, 1	2, 4, 1
Batch size	5	5	5	20	20	20	20
Época	200	200	400	400	400	400	100
% amostras para teste	20 %	20 %	20 %	20 %	20 %	10%	30 %
Matriz de confusão	30 8 10 42	27 11 7 45	22 16 14 38	27 11 8 44	27 11 17 35	10 9 8 18	31 7 9 43
Taxa de acerto	80 %	80 %	66,67%	78, 89%	68, 89%	62, 22%	82, 22%

As linhas da matriz de confusão correspondem ao que foi predito pelo algoritmo de machine learning, o MPL, e as colunas correspondem aos dados que são verdadeiros. Nesse caso, a função de ativação escolhida é uma sigmóide, portanto, só existe duas possibilidades de saída: o valor 0, equivale a situação na

qual o aluno é reprovado e o valor 1 equivale a situação na qual o aluno é aprovado. Contando-se a quantidade de 0 e 1 no conjunto de dados utilizando o comando unique, counts = np.unique(y_test, return_counts=True), foi possível notar que com o conjunto de teste de 30%, temos o seguinte output:

```
[0 1] [62 73]
```

Assim, é possível definir que o output da matriz de confusão equivale a: primeira linha - alunos reprovados; segunda linha - alunos aprovados.

```
[[54  8] => somatório = 62
 [16 57]]=> somatório = 73
```

Agora, é possível analisar os resultados obtidos na Tabela de forma quantitativa e buscar uma interpretação apropriada.

- Situação 1 e 2: é possível notar que alterando a quantidade de neurônios na camada inicial e na hidden layer, a taxa de acerto permaneceu a mesma. Entretanto, percebe-se que o algoritmo passou a errar mais a quantidade de alunos reprovados e acertar mais a quantidade de alunos aprovados (reprovados = 30 classificados corretamente na situação 1 contra 27 na situação 2 e aprovados = 42 classificados corretamente na situação 1 contra 45 na situação 2).
- Situação 2 e 3: manteve-se a mesma configuração para as camadas, alterando-se somente a quantidade de épocas. O aumento da quantidade de épocas causou a redução na taxa de acerto da rede, pois acabou gerando uma situação de overfitting.
- Situação 3 e 4: nessa etapa, foi proposto alterar o batch size, ou seja, o número de exemplos de treinamento usados em uma iteração. Conclui-se então que ao utilizar mais exemplos em uma iteração, a taxa de acerto aumenta, pois ele recebe mais ‘conhecimento’ acerca do problema.
- Situação 4 e 5: nesse caso, foi mantido a quantidade de neurônios na input layer e na primeira hidden layer, adicionando-se uma segunda hidden layer com 24 neurônios. Verificou-se que a predição do algoritmo em relação à classe dos reprovados manteve-se a mesma, entretanto na classe dos aprovados 9 amostras a mais foram classificadas incorretamente.
- Situação 5 e 6: Nesse caso, foi reduzido o tamanho do conjunto de testes de 20% para 10%, aumentando assim o tamanho do conjunto de treinamento. Era esperado que isso fosse fazer com que a taxa de acerto aumentasse, pois a rede teria mais dados para treinar e aprender melhor, entretanto não

foi o obtido, pois a taxa de acerto caiu em aproximadamente 6,6%. Isso indica que aparentemente houve um overfitting, o tamanho do conjunto de treinamento aumentou, mas a rede acabou “decorando” os padrões. Assim, quando um novo dado proveniente do conjunto de teste é classificado, a rede acaba não acertando, fazendo, portanto, com que a taxa de acerto caísse.

- Situação 7: foi observado que reduzindo a quantidade de neurônios e épocas o custo computacional foi reduzido bastante, e a taxa de acerto foi a maior encontrada. Também foi alterado a quantidade de amostras para teste para 30%, que é o que comumente é utilizado conforme observado em sala.