UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
INTELIGÊNCIA ARTIFICIAL

Trabalho Prático 5
Redes Neurais Artificiais

Francisco Luiz Vicenzi
Gustavo Emanuel Kundlatsch
PROFESSOR
Mauro Roisenberg

Florianópolis
Dezembro de 2020

1 Descrição do problema

O problema a ser resolvido neste trabalho consiste na idetinficação de objetos em um Raio-X de aeroporto. Para isso, foi disponibilizado um conjunto de dados contendo imagens de objetos de vestuário. Segundo o enunciado, todas as imagens já foram tratadas e centralizadas.

Recebemos os dados previamente separados em treino e teste. No conjunto de treino, existem 60 mil entradas, cada qual com 785 colunas, sendo uma delas, a classe. Desse modo, observamos que cada linha representa uma imagem de 28x28 pixels. No conjunto de teste, existem 10 mil linhas, com o mesmo número de colunas do conjunto de treino. Em ambos os conjuntos, o número de classes está distribuído de forma igual. Essas informações estão sumarizadas na tabela 1. Além disso, os valores de cada coluna de dados variam de 0 a 255.

Conjunto	**Linhas**	**Colunas**	**Classes**	**Linhas por classe**
Treino	60000	785	10	6000
Teste	10000	785	10	1000

Tabela 1 – Informações sobre o conjunto de dados

2 Solução proposta

2.1 Abordagem escolhida

O classificador escolhido para os experimentos foi o *Multi-layer Perceptron*, por ter sido estudado em aula e ser de fácil utilização a partir da biblioteca *sklearn*. Além disso, ainda utilizando-se desta biblioteca, foram aplicadas duas transformações no conjunto de dados: *one-hot encoding* nos valores das classes e *MinMaxScaling* nos valores dos atributos.

2.1.1 One-Hot Encoding

Como as classes do problema são simplesmente categóricas e não demonstram nenhuma relação com seus respectivos números, decidimos codificá-las como One-Hot. O resultado é mostrado na figura 1, enquanto o código deste processo é mostrado na listagem 1.

```
{0: array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.]),
 1: array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.]),
 2: array([0., 0., 1., 0., 0., 0., 0., 0., 0., 0.]),
 3: array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0.]),
 4: array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.]),
 5: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]),
 6: array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.]),
 7: array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.]),
 8: array([0., 0., 0., 0., 0., 0., 0., 0., 1., 0.]),
 9: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1.])}
```

Figura 1 – Resultado da codificação one-hot.

2.1.2 MinMaxScaling

Uma vez que as imagens do conjunto de dados estão em escala de cinza, é possível que existam muitos valores de pixels simplesmente brancos ou pretos que não ajudem na classificação da imagem, podendo, inclusive, piorá-lo. Deste modo, com o intuito de diminuir o impacto que os pixels de fundo possam ter na classificação, decidimos normalizar os atributos através da técnica *MinMax*: $X\_std * (max - min) + min$. A partir disso, os valores de treino ficarão no intervalo entre 0 e 1. O processo está descrito em 2. Note que primeiro realizamos o *fit* no conjunto de treino para, posteriormente, aplicá-lo no conjunto de testes.

2.2 Descrição dos experimentos

A biblioteca apresenta diversos parâmetros para customização do classificador (1), que serão explorados em 3.1. Demos ênfase para a variação dos seguintes parâmetros:

1. *hidden_layer_sizes*: entre uma a duas camadas, com número de neurônios entre 50 e 200;
2. *activation*: entre função logística e Relu;
3. *alpha*: entre 0.001, 0.0001 e 0.00001;

Além destes, mantivemos o *solver* como Adam e com *early_stopping* ligado para todos os experimentos. O critério para parada precoce consiste em 10 épocas sem melhorias relevantes (maiores que 0.0001).

2.3 Código

```
from sklearn.preprocessing import OneHotEncoder

one_hot_y = OneHotEncoder()
one_hot_y.fit(y_train.to_numpy().reshape(-1, 1))

y_train_transformed = one_hot_y.transform(y_train.to_numpy()
                                          .reshape(-1, 1)
                                          ).toarray()
y_test_transformed = one_hot_y.transform(y_test.to_numpy()
                                         .reshape(-1, 1)
                                         ).toarray()
```

Listing 1: Processo de codificação One-Hot.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler().fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Listing 2: Processo de normalização MinMax.

O restante do processo está descrito no notebook *assignment.ipynb*, enviado em conjunto com este relatório. As bibliotecas utilizadas foram *numpy*, *pandas* e *sklearn*. Além disso, criamos como um projeto *Poetry*, para facilitar a execução. O arquivo README contém detalhes adicionais sobre o processo.

3 Conclusão

3.1 Resultados Obtidos

Os resultados obtidos com os experimentos, assim como os valores dos parâmetros, estão descritos na tabela 2. Foram realizados, no total, 24 experimentos, variando os parâmetros de acordo com o que foi descrito em 2.2. Os maiores valores de acurácia estão destacados na tabela.

	Neurônios	**Activation**	**Alpha**	**Acc Treino**	**Acc Teste**	**Epochs**
1	(50)	Logistic	0.0001	0.88328	0.8513	80
2	(50)	Logistic	0.001	0.86348	0.8425	47
3	(50)	Logistic	0.00001	0.87068	0.8442	61
4	(50)	Relu	0.0001	0.87593	0.8519	45
5	(50)	Relu	0.001	0.88573	0.85	60
6	(50)	Relu	0.00001	0.88736	0.8505	57
7	(100)	Logistic	0.0001	0.89063	0.8557	58
8	(100)	Logistic	0.001	0.8887	0.855	58
9	(100)	Logistic	0.00001	0.89068	0.8557	58
10	(100)	Relu	0.0001	0.89081	0.8569	43
11	(100)	Relu	0.001	0.88181	0.8566	35
12	(100)	Relu	0.00001	0.8895	0.8556	43
13	(200)	Logistic	0.0001	0.9272	0.8673	79
14	(200)	Logistic	0.001	0.9031	0.8622	57
15	(200)	Logistic	0.00001	**0.92765**	0.868	79
16	(200)	Relu	0.0001	0.91745	0.8657	48
17	(200)	Relu	0.001	0.9311	0.8664	63
18	(200)	Relu	0.00001	0.91606	0.8658	48
19	(200, 200)	Logistic	0.0001	0.91785	0.8785	45
20	(200, 200)	Logistic	0.001	0.90546	0.8723	39
21	(200, 200)	Logistic	0.00001	0.90785	0.8731	39
22	(200, 200)	Relu	0.0001	0.91755	0.8792	30
23	(200, 200)	Relu	0.001	0.9248	0.8826	31
24	(200, 200)	Relu	0.00001	0.92636	**0.8856**	31

Tabela 2 – Parâmetros dos experimentos e resultados correspondentes.

3.2 Discussão

Para definir o melhor resultado, levamos em consideração apenas a acurácia no conjunto de teste. Deste modo, o melhor classificador consistiu no experimento 24, com uma acurácia

de 88.56%. Além do *solver* Adam, foi utilizado duas camadas, cada uma com 200 neurônios, função de ativação ReLu e alpha 0.00001. Nele, foram executadas apenas 31 épocas.

A figura 2 mostra a matriz de confusão gerada pelo classificador do experimento 24. A análise mais simples de performance dá-se pela visualização da diagonal principal. Sendo assim, podemos notar que a classe com pior perfomance foi a 6, *Shirt*, seguida pela 2, *Pullover* e pela 4, *Coat*. A classe com melhor performance foi a 1, *Trouser*.

A classe 6, que teve a pior performance, foi, majoritariamente, confundida com a classe 0. Em 179 vezes, enquanto eram visualizadas imagens da classe 6, foram classificadas erroneamente como classe 0. Por outro lado, 73 imagens da classe 0 foram classificadas erroneamente como classe 6. É importante destacar a categoria das classes 0 e 6: *T-shirt/top* e *Shirt*. Essa semelhança semântica entre as classes pode ajudar a explicar o porquê da confusão realizada pelo classificador. As outras classes com menor precisão, 2 e 4, também compartilham do mesmo problema de similidaridade, já que são, respectivamente, *Pullover* e *Coat*.

O restante das classes teve uma boa precisão, no geral. Isso se deu, principalmente, por serem itens mais diferentes entre si. Por exemplo, a classe 1, que teve a maior precisão, representa *Trousers*, item que não possui tanta similaridade com o restante da lista.

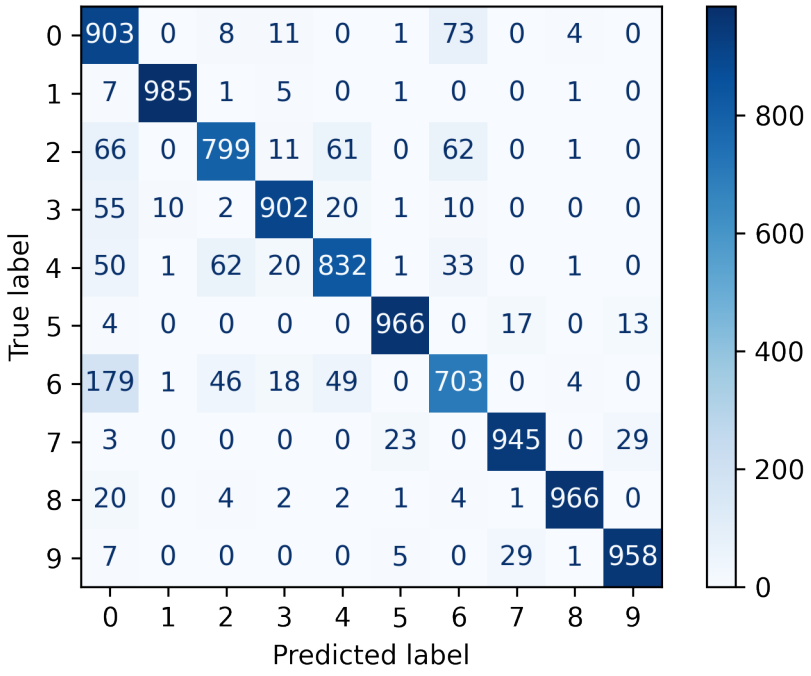

Figura 2 – Matriz de confusão para o classificador com melhor resultado.

Referências

1 sklearn.neural_network.MLPClassifier. *scikit-learn*. [Online; acessado em 30/11/2020]. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html>.