Segunda quest?o

October 17, 2018

1 Questão 2 - Prova 2 de Inteligência Artificial

1.1 Lucas Nóbrega e Nathália de Vasconcelos

Enunciado: utilizando a base disponível no link, crie os datasets a seguir:

Dataset	% de instâncias
Treino	60%
Validação	20%
Teste	20%

Elabore uma rede neural de duas camadas para classificação do banco de dados. Ao fim do treinamento, avalie o desempenho da rede utilizando a matriz de confusão com o dataset de teste e mostre o valor de acurácia. Observações:

Utilize apenas o arquivo avila-tr.txt .

A camada de saída da rede deverá conter um neurônio para cada classe.

Utilize o dataset de validação para criar algum critério de parada no treinamento.

Bônus: defina uma arquitetura de rede neural ou modelos de deep learning que ultrapassem 75% de acurácia.

1.1.1 DATA SET DESCRIPTION

The Avila data set has been extracted from 800 images of the the "Avila Bible", a giant Latin copy of the whole Bible produced during the XII century between Italy and Spain.
The palaeographic analysis of the manuscript has individuated the presence of 12 copyists. The pages written by each copyist are not equally numerous. Each pattern contains 10 features and corresponds to a group of 4 consecutive rows.

The prediction task consists in associating each pattern to one of the 12 copyists (labeled as: A, B, C, D, E, F, G, H, I, W, X, Y). The data have has been normalized, by using the Z-normalization method, and divided in two data sets: a training set containing 10430 samples, and a test set containing the 10437 samples.

1.1.2 Class distribution (training set)

- A: 4286
- B: 5

- C: 103
- D: 352
- E: 1095
- F: 1961
- G: 446
- H: 519
- I: 831
- W: 44
- X: 522
- Y: 266

1.1.3 ATTRIBUTE DESCRIPTION

ID	Name
F1	intercolumnar distance
F2	upper margin
F3	lower margin
F4	exploitation
F5	row number
F6	modular ratio
F7	interlinear spacing
F8	weight
F9	peak number
F10	modular ratio/ interlinear spacing

Class: A, B, C, D, E, F, G, H, I, W, X, Y

CITATIONS If you want to refer to the Avila data set in a publication, please cite the following paper:

C. De Stefano, M. Maniaci, F. Fontanella, A. Scotto di Freca, Reliable writer identification in medieval manuscripts through page layout features: The "Avila" Bible case, Engineering Applications of Artificial Intelligence, Volume 72, 2018, pp. 99-110.

```
In [1]: import tensorflow as tf
```

```
/home/nath/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion
  from ._conv import register_converters as _register_converters
```

```
In [2]: import matplotlib.pyplot as plt

        %matplotlib inline

        import numpy as np
```

```
In [3]: import pandas as pd
```

```
In [4]: dataset = pd.read_csv('./avila/avila-tr.txt', header = None)
```

In [5]:
```
dataset.head(5)
```

Out[5]:
```
          0         1         2         3         4         5         6  \
0  0.266074 -0.165620  0.320980  0.483299  0.172340  0.273364  0.371178
1  0.130292  0.870736 -3.210528  0.062493  0.261718  1.436060  1.465940
2 -0.116585  0.069915  0.068476 -0.783147  0.261718  0.439463 -0.081827
3  0.031541  0.297600 -3.210528 -0.583590 -0.721442 -0.307984  0.710932
4  0.229043  0.807926 -0.052442  0.082634  0.261718  0.148790  0.635431

          7         8         9  10
0  0.929823  0.251173  0.159345  A
1  0.636203  0.282354  0.515587  A
2 -0.888236 -0.123005  0.582939  A
3  1.051693  0.594169 -0.533994  A
4  0.051062  0.032902 -0.086652  F
```

In [6]:
```
dataset.describe()
```

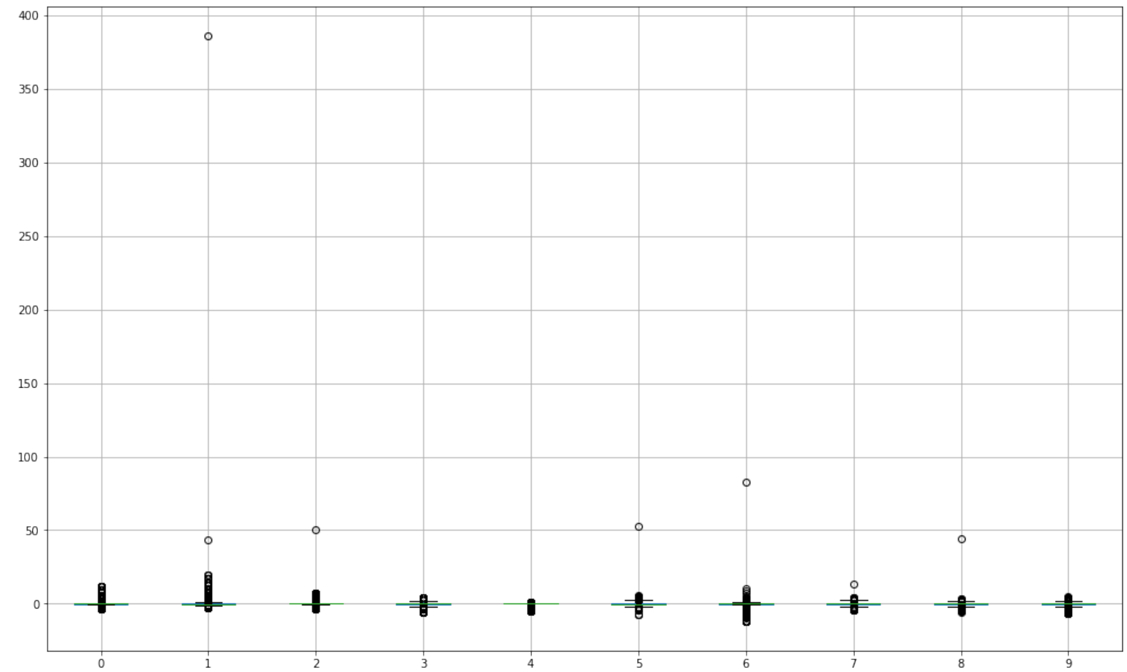
Out[6]:
```
                  0             1             2             3             4  \
count  10430.000000  10430.000000  10430.000000  10430.000000  10430.000000
mean       0.000852      0.033611     -0.000525     -0.002387      0.006370
std        0.991431      3.920868      1.120202      1.008527      0.992053
min       -3.498799     -2.426761     -3.210528     -5.440122     -4.922215
25%       -0.128929     -0.259834      0.064919     -0.528002      0.172340
50%        0.043885     -0.055704      0.217845      0.095763      0.261718
75%        0.204355      0.203385      0.352988      0.658210      0.261718
max       11.819916    386.000000     50.000000      3.987152      1.066121

                  5             6             7             8             9
count  10430.000000  10430.000000  10430.000000  10430.000000  10430.000000
mean       0.013973      0.005605      0.010323      0.012914      0.000818
std        1.126245      1.313754      1.003507      1.087665      1.007094
min       -7.450257    -11.935457     -4.247781     -5.486218     -6.719324
25%       -0.598658     -0.044076     -0.541992     -0.372457     -0.516097
50%       -0.058835      0.220177      0.111803      0.064084     -0.034513
75%        0.564038      0.446679      0.654944      0.500624      0.530855
max       53.000000     83.000000     13.173081     44.000000      4.671232
```

In [7]:
```
# Aumentar o tamanho do plot na proporção 8/13
x_size = 17
y_size = x_size * (8/13)
plt.figure(figsize=(x_size, y_size))
# Aumentar o tamanho do plot na proporção 8/13

dataset.boxplot()
```

Out[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b945df2e8>
```



2 Percebendo o *outlier* na coluna 1

In [8]:
```
a = dataset.loc[dataset[1] == 386]
```

In [9]:
```
a
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	10
6619	0.0	386.0	50.0	0.168104	0.0	53.0	83.0	0.275032	44.0	0.63802	A

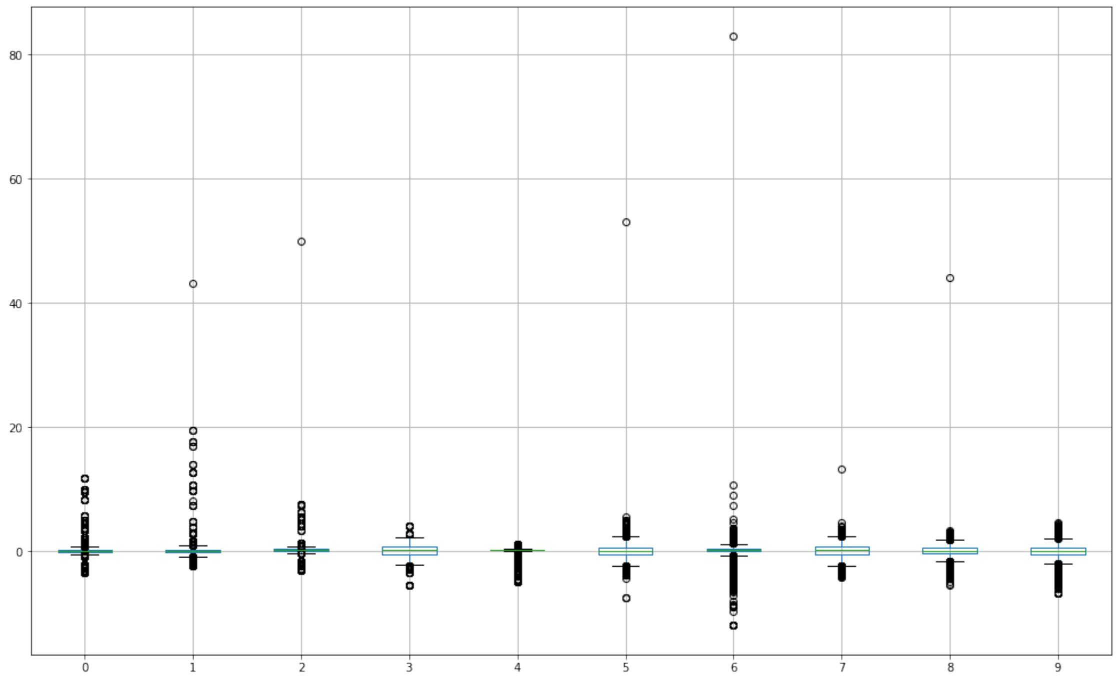
In [10]:
```
aux = np.array(dataset[1])
dataset[1] = dataset[1].replace(386, np.nanmedian(aux))
```

In [11]:
```
dataset.loc[dataset[1] == 386]
```

Out[11]:
```
Empty DataFrame
Columns: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Index: []
```

In [12]:
```
# Aumentar o tamanho do plot na proporção 8/13
x_size = 17
y_size = x_size * (8/13)
plt.figure(figsize=(x_size, y_size))
# Aumentar o tamanho do plot na proporção 8/13
dataset.boxplot()
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0b92768e48>
```



In [13]: dataset.describe()

Out[13]:

	0	1	2	3	4 \
count	10430.000000	10430.000000	10430.000000	10430.000000	10430.000000
mean	0.000852	-0.003403	-0.000525	-0.002387	0.006370
std	0.991431	1.042894	1.120202	1.008527	0.992053
min	-3.498799	-2.426761	-3.210528	-5.440122	-4.922215
25%	-0.128929	-0.259834	0.064919	-0.528002	0.172340
50%	0.043885	-0.055704	0.217845	0.095763	0.261718
75%	0.204355	0.203385	0.352988	0.658210	0.261718
max	11.819916	43.133656	50.000000	3.987152	1.066121

	5	6	7	8	9
count	10430.000000	10430.000000	10430.000000	10430.000000	10430.000000
mean	0.013973	0.005605	0.010323	0.012914	0.000818
std	1.126245	1.313754	1.003507	1.087665	1.007094
min	-7.450257	-11.935457	-4.247781	-5.486218	-6.719324
25%	-0.598658	-0.044076	-0.541992	-0.372457	-0.516097
50%	-0.058835	0.220177	0.111803	0.064084	-0.034513
75%	0.564038	0.446679	0.654944	0.500624	0.530855
max	53.000000	83.000000	13.173081	44.000000	4.671232

In [14]: b = np.array(dataset[6])

In [15]: b

Out[15]: array([0.371178, 1.46594 , -0.081827, ..., 0.295677, 0.069175,
        0.786433])

In [16]: np.max(b)

Out[16]: 83.0

In [17]:
```
for i in [1, 2, 5, 6, 7, 8]:
    aux = np.array(dataset[i])
    dataset[i] = dataset[i].replace(np.max(aux), np.nanmedian(aux))
```

In [18]:
```
dataset.describe()
```

Out[18]:

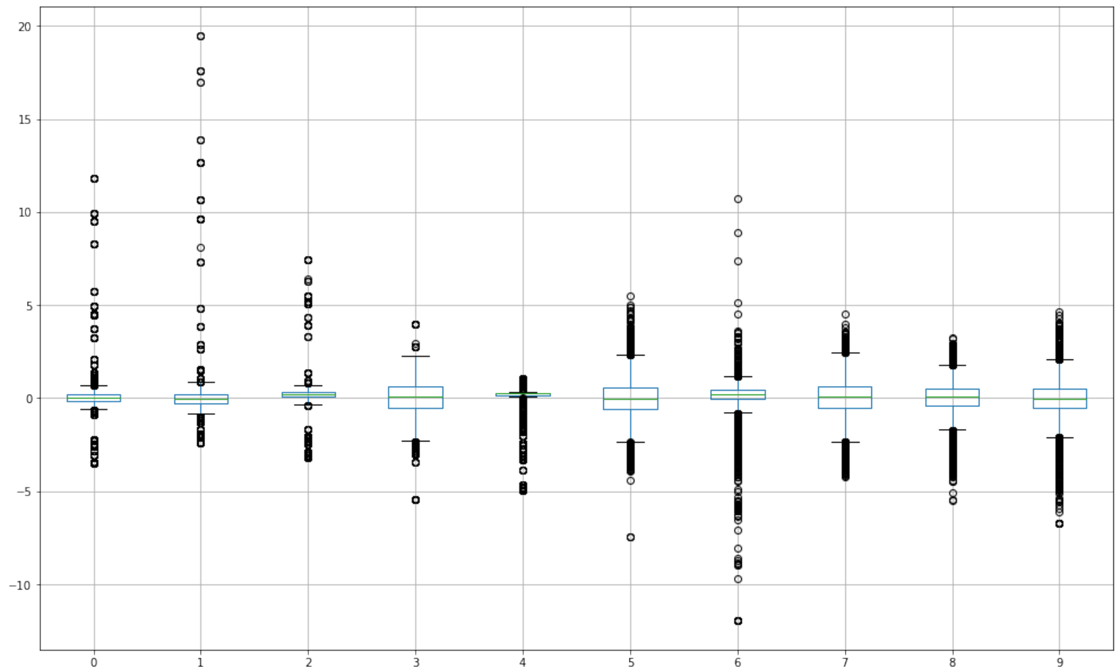
	0	1	2	3	4
count	10430.000000	10430.000000	10430.000000	10430.000000	10430.000000
mean	0.000852	-0.007544	-0.005298	-0.002387	0.006370
std	0.991431	0.953512	1.007528	1.008527	0.992053
min	-3.498799	-2.426761	-3.210528	-5.440122	-4.922215
25%	-0.128929	-0.259834	0.064919	-0.528002	0.172340
50%	0.043885	-0.055704	0.217845	0.095763	0.261718
75%	0.204355	0.203385	0.352988	0.658210	0.261718
max	11.819916	19.470188	7.458681	3.987152	1.066121

\

	5	6	7	8	9
count	10430.000000	10430.000000	10430.000000	10430.000000	10430.000000
mean	0.008886	-0.002331	0.009070	0.008702	0.000818
std	0.999600	1.032191	0.995195	0.998735	1.007094
min	-7.450257	-11.935457	-4.247781	-5.486218	-6.719324
25%	-0.598658	-0.044076	-0.541992	-0.372457	-0.516097
50%	-0.058835	0.220177	0.111778	0.064084	-0.034513
75%	0.564038	0.446679	0.654886	0.500624	0.530855
max	5.505495	10.714792	4.510897	3.244594	4.671232

In [19]:
```
# Aumentar o tamanho do plot na proporção 8/13
x_size = 17
y_size = x_size * (8/13)
plt.figure(figsize=(x_size, y_size))
# Aumentar o tamanho do plot na proporção 8/13
dataset.boxplot()
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0b91db00f0>



3 Mudando a classe de caractere para inteiro

In [20]:
```
classes_number = [i for i in range(12)]
classes_charcteres = ["A","B","C","D","E","F","G","H","I","W","X","Y"]
dictionary_classes = dict(zip(classes_charcteres, classes_number))
dictionary_classes
```

Out[20]:
```
{'A': 0,
 'B': 1,
 'C': 2,
 'D': 3,
 'E': 4,
 'F': 5,
 'G': 6,
 'H': 7,
 'I': 8,
 'W': 9,
 'X': 10,
 'Y': 11}
```

In [21]:
```
dataset[10].head(5)
```

Out[21]:
```
0    A
1    A
2    A
```

```
3    A
4    F
Name: 10, dtype: object
```

```
In [22]: for key in dictionary_classes:
             print(key, dictionary_classes[key])
             dataset[10] = dataset[10].replace(key, dictionary_classes[key])
```

```
A 0
B 1
C 2
D 3
E 4
F 5
G 6
H 7
I 8
W 9
X 10
Y 11
```

Testando se realmente funcionou

```
In [23]: dataset[10].head()
```

```
Out[23]: 0    0
         1    0
         2    0
         3    0
         4    5
         Name: 10, dtype: int64
```

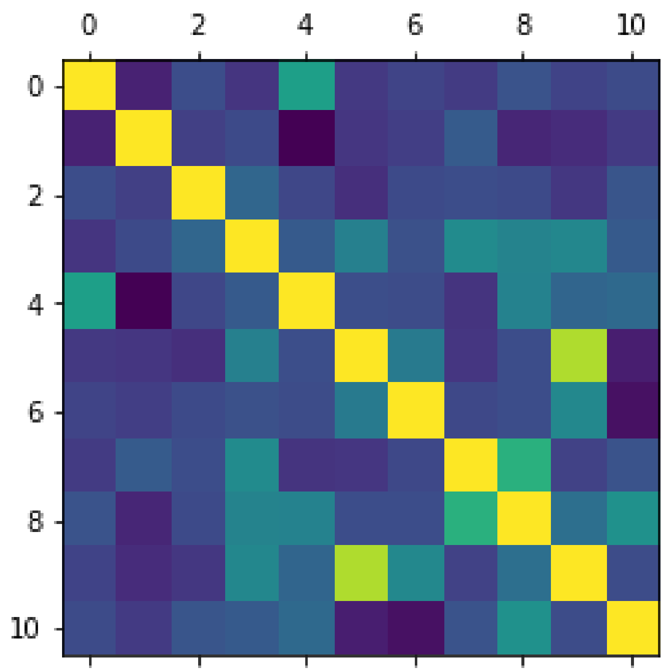
```
In [24]: dataset[10].tail()
```

```
Out[24]: 10425     5
         10426     5
         10427     0
         10428     4
         10429    10
         Name: 10, dtype: int64
```

4 Análise inicial de correlação entre os atributos

```
In [25]: plt.matshow(dataset.corr())
```

```
Out[25]: <matplotlib.image.AxesImage at 0x7f0b8df0b630>
```



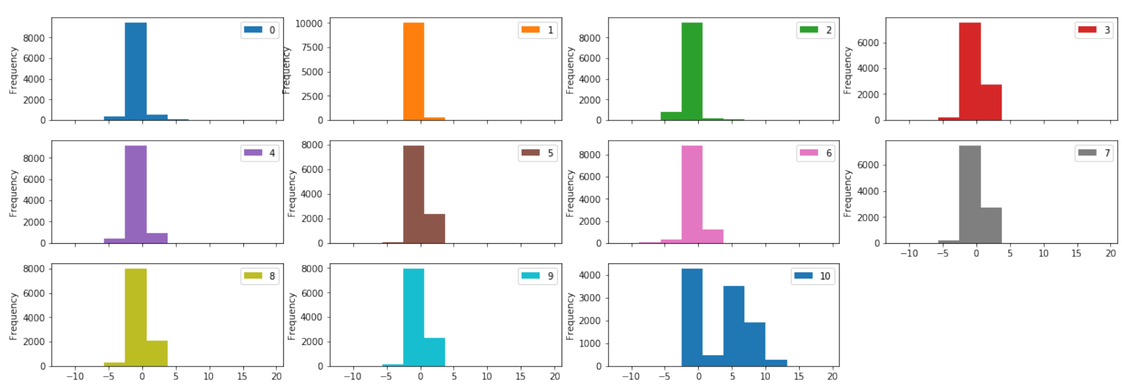
Percebemos que os atributos 5('row number') e 9(peak number) possuem alta correlação

```
In [26]: dataset[9].corr(dataset[5])
```

```
Out[26]: 0.8485534155934017
```

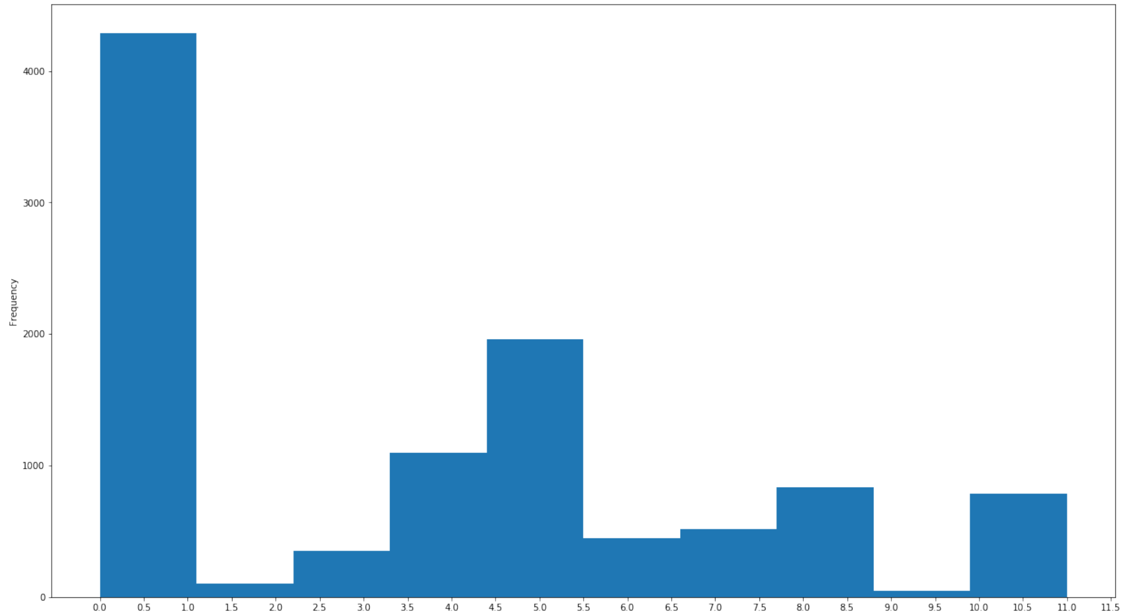
?dataset.plot('hist')

```
In [27]: a = dataset.plot(kind='hist', subplots=True, figsize=(21,12), layout=(5,4))
```




```
In [28]: dataset[10].plot(kind='hist',figsize=(21,12), xticks=[i/2 for i in range(24)])
```

```
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0b8c7832e8>
```



5 Normalizar os atributos

In [29]:
```
dataset.columns
```

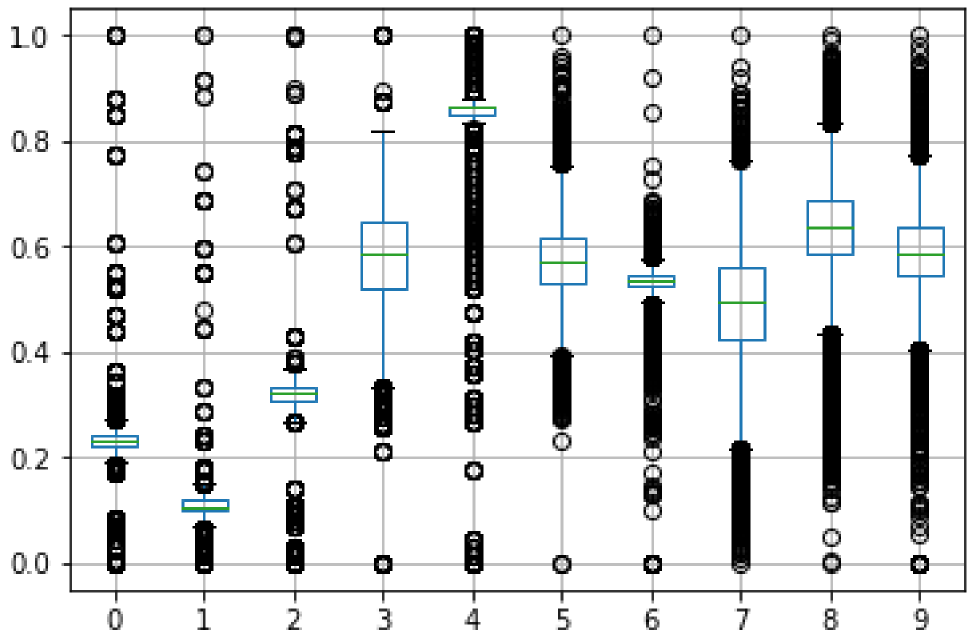
Out[29]: Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10], dtype='int64')

In [30]:
```
from sklearn import preprocessing

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 10].values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
df = pd.DataFrame(x_scaled)
```

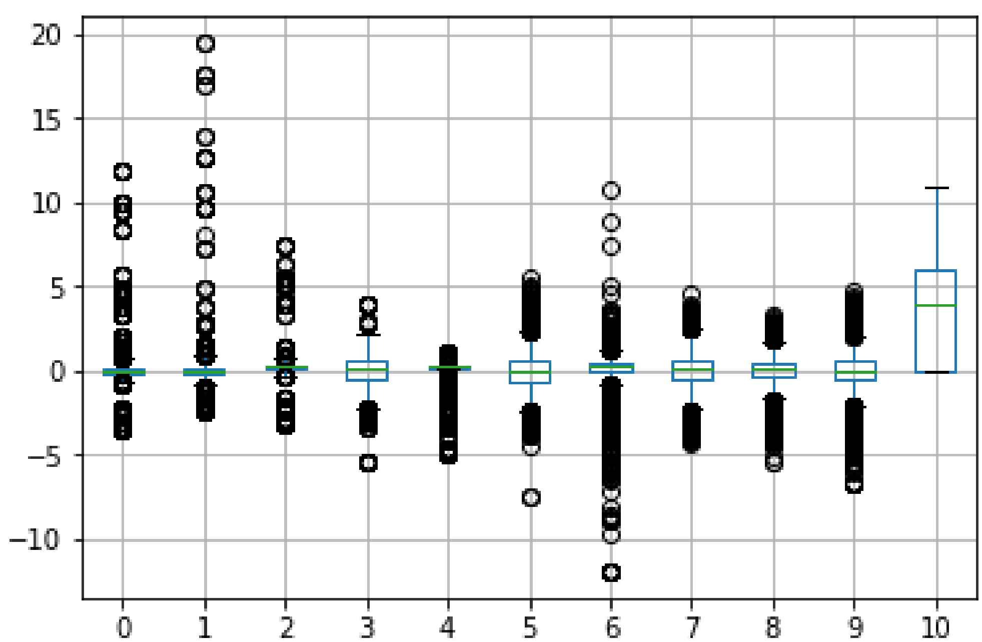
In [31]:
```
df.boxplot()
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0b8c4ac358>




```
In [32]: dataset.boxplot()

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0b7c200710>
```



```
In [33]: dataset

Out[33]:            0         1         2         3         4         5         6  \
         0      0.266074 -0.165620  0.320980  0.483299  0.172340  0.273364  0.371178
         1      0.130292  0.870736 -3.210528  0.062493  0.261718  1.436060  1.465940
         2     -0.116585  0.069915  0.068476 -0.783147  0.261718  0.439463 -0.081827
         3      0.031541  0.297600 -3.210528 -0.583590 -0.721442 -0.307984  0.710932
         4      0.229043  0.807926 -0.052442  0.082634  0.261718  0.148790  0.635431
         5      0.117948 -0.220579 -3.210528 -1.623238  0.261718 -0.349509  0.257927
         6      0.389513 -0.220579 -3.210528 -2.624155  0.261718 -0.764757  0.484429
         7      0.019197 -0.040001  0.288973 -0.042597  0.261718 -1.013906  0.069175
         8      0.500607  0.140576  0.388552 -0.637358  0.261718 -0.681707  0.295677
         9     -0.252367  0.069915  0.246296  0.523550  0.261718 -1.221530  0.899684
         10    -0.042522 -2.395356 -3.210528 -1.128463 -0.721442  0.397939  0.257927
         11    -3.498799 -0.566031  0.139604 -5.440122  0.976743 -0.847807 -1.176589
         12     0.043885 -0.173471  0.285416 -0.775121  0.172340 -0.391033  0.333428
         13     0.154980  0.336855  0.068476  0.021525  0.261718  0.024215  0.031425
         14     0.290762 -0.189174  0.079145 -0.085921  0.976743 -1.470679 -0.950086
         15     0.142636 -0.244132 -3.206971  0.165524  0.887365 -2.093552 -0.874585
         16    -0.005490 -0.322644  0.100483 -0.519439  0.261718 -0.307984  0.069175
         17    -0.301743 -0.314793  0.399221  0.770520  0.708609  0.564038 -1.327590
         18    -0.091897  0.297600  0.079145  0.196496  0.261718 -0.183409  0.220177
         19    -0.091897 -0.220579  0.274747  0.567174 -0.185173  0.730137  0.031425
         20    -0.005490  0.478177  0.029355 -0.247644  0.172340  1.062336  0.333428
         21     0.438888  0.195534  0.143160 -0.809435  0.261718 -1.138481 -0.232828
         22     0.364825 -0.047852 -0.038216  0.327813  0.261718  0.439463  0.484429
         23    -0.054866 -0.220579  0.466793 -0.216970  0.172340 -0.930856  0.446679
         24     0.290762  0.077766  0.118265  0.275805  0.261718 -0.432558  0.031425
         25     0.105604 -0.087108  0.367214  1.522618  0.261718  1.186911  0.635431
         26    -0.030178  0.022808  0.157386 -0.771451  0.261718  1.062336  0.182426
         27    -0.252367 -0.008597 -2.044028  0.342613  0.351096 -0.307984  0.220177
         28     0.290762 -0.189174  0.079145 -0.085921  0.976743 -0.930856 -1.101088
         29     0.253730  0.336855  0.086258 -0.979571  0.261718 -1.055431  0.144676
         ...         ...       ...       ...       ...       ...       ...       ...
         10400  0.142636  0.486028  0.100483 -0.448543  0.172340  1.519109  0.597681
         10401  0.192011 -0.369751  0.278304 -0.659618  0.261718  1.062336  0.182426
         10402 -0.375806 -0.291239 -3.210528 -0.835274  0.082961 -0.889332 -2.950858
         10403  0.043885 -0.079257  0.061363  0.565085  0.261718  1.145386  0.408929
         10404 -0.079554 -0.338346  0.409890  0.129748  0.172340 -0.058835  0.182426
         10405 -0.091897 -0.142067  0.324537  0.658210  0.172340 -0.307984  0.295677
         10406 -0.178304  0.014957  0.264078  0.582361  0.172340  0.439463  0.144676
         10407  0.229043  0.124874  0.182281  0.686496  0.261718  2.889429  0.748682
         10408 -0.314087 -0.024299 -1.681275  0.307284  0.261718  0.148790  0.182426
         10409 -2.523635 -0.346197 -2.349878 -0.775688  0.172340 -0.474083  0.371178
         10410 -0.079554 -0.338346  0.409890  0.129748  0.172340 -0.640182 -0.081827
         10411 -0.116585 -0.079257  0.381439  1.540879  0.172340  0.564038  0.069175
         10412 -0.289399  0.831480  0.157386  0.987562  0.529852 -0.432558  1.088436
         10413  0.241386 -0.110662  0.324537 -0.772226  0.261718  0.397939  0.257927
```

10414	0.130292	-0.071406	0.214288	0.915742	0.261718	0.730137	0.786433
10415	-0.264711	-0.126364	0.235627	0.176773	0.261718	0.564038	0.182426
10416	0.130292	0.870736	-3.210528	0.062493	0.261718	0.190314	0.257927
10417	0.204355	0.360409	0.139604	0.299019	-0.095795	2.141982	0.710932
10418	-0.042522	-0.189174	0.331650	-0.664392	0.261718	-1.512204	0.069175
10419	-0.474557	0.446772	-3.210528	-0.716161	0.351096	4.550423	0.522180
10420	-0.005490	0.478177	0.029355	-0.247644	0.172340	0.605563	0.673182
10421	0.241386	0.234790	0.121822	1.037988	0.261718	0.647088	0.182426
10422	-0.277055	-0.251983	-3.203415	1.957926	0.261718	1.892833	0.635431
10423	4.969080	-0.385453	0.143160	-2.600732	0.976743	-0.764757	-0.232828
10424	0.216699	0.321153	0.128935	0.491087	0.261718	0.439463	0.069175
10425	0.080916	0.588093	0.015130	0.002250	0.261718	-0.557133	0.371178
10426	0.253730	-0.338346	0.352988	-1.154243	0.172340	-0.557133	0.257927
10427	0.229043	-0.000745	0.171611	-0.002793	0.261718	0.688613	0.295677
10428	-0.301743	0.352558	0.288973	1.638181	0.261718	0.688613	0.069175
10429	-0.104241	-1.037102	0.388552	-1.099311	0.172340	-0.307984	0.786433

	7	8	9	10
0	0.929823	0.251173	0.159345	0
1	0.636203	0.282354	0.515587	0
2	-0.888236	-0.123005	0.582939	0
3	1.051693	0.594169	-0.533994	0
4	0.051062	0.032902	-0.086652	5
5	-0.385979	-0.247731	-0.331310	0
6	-0.597510	-0.372457	-0.810261	0
7	0.890701	0.095265	-0.842014	5
8	0.931046	0.500624	-0.642297	7
9	1.373076	0.625350	-1.400890	4
10	-1.167255	-0.060642	0.345537	0
11	-2.008078	0.438262	0.009512	8
12	-1.807711	-0.996086	-0.410850	5
13	0.079292	0.313536	0.125675	0
14	-0.273525	0.968347	-0.724959	8
15	-2.694490	-3.334697	-1.399407	8
16	-0.574114	0.282354	-0.182852	0
17	0.004193	0.750076	1.649058	11
18	0.305093	-0.278912	-0.163443	5
19	-0.396638	-0.403638	0.770625	0
20	-0.670454	0.095265	0.902376	0
21	-0.087894	-0.278912	-0.802015	5
22	-0.964306	-0.933723	0.232267	0
23	0.634922	0.313536	-0.942674	3
24	0.676859	-0.091823	-0.293988	5
25	0.574967	0.656532	0.817580	0
26	-1.217230	-0.185368	0.981553	7
27	1.240469	2.090879	-0.295362	10
28	0.495694	1.685520	-0.087857	8
29	-0.626478	0.064084	-0.896238	6

```
...          ...       ...       ...  ..
10400 -0.779685 -1.276719  1.123020   0
10401 -1.218279 -0.466001  0.978009   5
10402 -0.238674  0.001721 -2.199566   3
10403 -0.197785 -0.216549  0.914816   0
10404  0.283464  0.531806 -0.039143   0
10405 -0.168119 -0.840179 -0.324222   5
10406  1.025657  0.126447  0.442665   0
10407 -1.313899 -0.777816  2.230284   0
10408  0.524060  0.219991  0.142276   4
10409  0.574559 -0.029461 -0.505543   5
10410  0.634669  0.500624 -0.421122   0
10411  0.818883  1.654339  0.660936  10
10412 -0.421470 -0.123005 -0.075416  11
10413 -1.689549 -0.559546  0.334759   5
10414  0.006037 -0.185368  0.329010   0
10415 -0.183534 -0.933723  0.529293   0
10416  1.605517  0.750076  0.150569   0
10417 -1.136773 -0.653090  1.605636   0
10418  0.063663  1.030710 -1.295118   7
10419 -2.037065 -2.368071  3.871867   0
10420 -0.951919 -0.528364  0.286973   0
10421  0.684936  0.219991  0.628422   0
10422  1.898205  2.184424  1.427425  10
10423 -2.348488 -1.183175 -0.459372   8
10424  0.252846  0.188810  0.482857   0
10425  0.932346  0.282354 -0.580141   5
10426  0.348428  0.032902 -0.527134   5
10427 -1.088486 -0.590727  0.580142   0
10428  0.502761  0.625350  0.718969   4
10429 -1.337547  0.999528 -0.551063  10

[10430 rows x 11 columns]
```

6 Diminuir a classe A do sistema

Devido a um desbalanceamento das classes é necessário remover algumas instancias de A

In [34]:
```python
from sklearn.utils import resample
```

In [35]:
```python
dataset_b = dataset[dataset[10] == 1]
```

7 Criando fake data para a classe B

In [36]:
```python
dataset[10].describe()
```

Out[36]:
```
count    10430.000000
mean         3.542761
```

```
std           3.418046
min           0.000000
25%           0.000000
50%           4.000000
75%           6.000000
max          11.000000
Name: 10, dtype: float64
```

In [37]:
```
# Docstring:
# Return random integer in range [a, b], including both end points.
df = dataset
# array com indices de B
a = dataset.loc[dataset[10] == 1].index
a
```

Out[37]:
```
Int64Index([708, 4639, 7119, 7740, 9457], dtype='int64')
```

In [44]:
```
import random
import datetime
from datetime import datetime as datetime

for i in range(10):
    random.seed(datetime.now())
    parametro_alterado = random.randint(0,9)
    linha_alterada = random.randint(0,len(a)-1)
    valor_a_ser_alterado = (random.random())/10
    soma_ou_diminui = random.randint(0,1)

    print(parametro_alterado, linha_alterada, valor_a_ser_alterado, soma_ou_diminui)
    print(df[parametro_alterado][a[linha_alterada]])
    print(valor_a_ser_alterado + df[parametro_alterado][a[linha_alterada]])
    print(df[linha_alterada-1:linha_alterada])
    print(12*"#")



    if soma_ou_diminui == 1:
        pass
        #soma
        #df.append(pd.DataFrame(valor_a_ser_alterado + df[parametro_alterado][a[linha_

    #else:
    #    df = df.append(-valor_a_ser_alterado + df[parametro_alterado][a[linha_altera

    #a = df.loc[dataset[10] == 1].index
```

```
0 4 0.09190188811913438 1
-0.128929
-0.037027111880865604
```

```
         0        1          2         3          4          5         6   \
3  0.031541  0.2976 -3.210528 -0.58359 -0.721442 -0.307984  0.710932

         7         8         9   10
3  1.051693  0.594169 -0.533994   0
############
4 4 0.03995625978983738 1
-3.22403
-3.1840737402101627
         0        1          2         3          4          5         6   \
3  0.031541  0.2976 -3.210528 -0.58359 -0.721442 -0.307984  0.710932

         7         8         9   10
3  1.051693  0.594169 -0.533994   0
############
8 2 0.023568265700846015 0
1.404887
1.428455265700846
         0         1          2         3         4        5        6   \
1  0.130292  0.870736 -3.210528  0.062493  0.261718  1.43606  1.46594

         7         8         9   10
1  0.636203  0.282354  0.515587   0
############
7 0 0.08392281545422367 1
0.80704
0.8909628154542236
Empty DataFrame
Columns: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Index: []
############
8 1 0.05496091420794534 1
1.903791
1.9587519142079453
         0        1        2         3        4         5         6   \
0  0.266074 -0.16562  0.32098  0.483299  0.17234  0.273364  0.371178

         7         8         9   10
0  0.929823  0.251173  0.159345   0
############
1 2 0.03664369863667409 0
12.655362
12.692005698636674
         0         1          2         3         4        5        6   \
1  0.130292  0.870736 -3.210528  0.062493  0.261718  1.43606  1.46594

         7         8         9   10
1  0.636203  0.282354  0.515587   0
```

```
############
8 0 0.09870835870226205 0
1.186617
1.2853253587022622
Empty DataFrame
Columns: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Index: []
############
1 4 0.06968034335899059 1
12.655362
12.725042343358991
         0       1         2        3         4         5         6   \
3  0.031541  0.2976 -3.210528 -0.58359 -0.721442 -0.307984  0.710932

         7         8         9   10
3  1.051693  0.594169 -0.533994   0
############
1 3 0.09080262265097096 1
12.655362
12.746164622650971
         0         1         2         3         4         5         6   \
2 -0.116585  0.069915  0.068476 -0.783147  0.261718  0.439463 -0.081827

         7         8         9   10
2 -0.888236 -0.123005  0.582939   0
############
9 4 0.09427459555915263 0
0.9462200000000001
1.0404945955591527
         0       1         2        3         4         5         6   \
3  0.031541  0.2976 -3.210528 -0.58359 -0.721442 -0.307984  0.710932

         7         8         9   10
3  1.051693  0.594169 -0.533994   0
############
```

8 Usando Keras

Keras é uma API de alto nível para o tensorflow, que permite criar redes neurais complexas sem precisar entender as variáveis que o Tensorflow possui

Primeiro é preciso particionar a base de acordo com as porcentagens abaixo

Dataset	% de instâncias
Treino	60%
Validação	20%

Dataset	% de instâncias
Teste	20%

```
In [45]: X = dataset.iloc[:, :-1].values
         y = dataset.iloc[:, 10].values
```

```
In [46]: import math
```

Pega os valores do começo até o numero correspondente a 60% menos 1 (função floor)

```
In [47]: x_treinamento = X[:math.floor(len(X)*0.6)]
         y_treinamento = y[:math.floor(len(y)*0.6)]
```

Pega os valores do número correspondente a 60% mais 1 (função ceil) até 60% mais 1 (função ceil) mais 20% menos 1 (função floor)

```
In [48]: x_validacao = X[math.ceil(len(X)*0.6):(math.ceil(len(X)*0.6)+math.floor(len(X)*0.2))]
         y_validacao = y[math.ceil(len(y)*0.6):(math.ceil(len(y)*0.6)+math.floor(len(y)*0.2))]
```

Pega os valores do número correspondente a 60% mais 1 (função ceil) mais 20% mais 1 (função ceil) mais 20% menos 1 função floor

```
In [49]: x_teste = X[(math.ceil(len(X)*0.6)+math.ceil(len(X)*0.2)):(math.ceil(len(X)*0.6)+math
         y_teste = y[(math.ceil(len(y)*0.6)+math.ceil(len(y)*0.2)):(math.ceil(len(X)*0.6)+math
```

Pega os valores do número correspondente a 60% mais 1 (função ceil) até 60% mais 1 (função ceil) mais 20% mais 1 (função floor)

```
In [50]: model = tf.keras.models.Sequential([
           tf.keras.layers.Flatten(),
           tf.keras.layers.Dense(13, activation=tf.nn.relu),
           tf.keras.layers.Dropout(0.1),
           tf.keras.layers.Dense(12, activation=tf.nn.softmax)
         ])

         model.compile(optimizer=tf.train.GradientDescentOptimizer(0.1),
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
         '''
         model.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
         '''

         model.fit(x_treinamento, y_treinamento, epochs=20)

         print("Evaluate:")
         print("Returns the loss value & metrics values for the model in test mode.")
```

```
print(model.evaluate(x_teste, y_teste))

print("Returns the loss value & metrics values for the model in test mode. Test using
print(model.evaluate(x_validacao, y_validacao))
```

```
Epoch 1/20
6258/6258 [==============================] - 0s 56us/step - loss: 1.7228 - acc: 0.4578
Epoch 2/20
6258/6258 [==============================] - 0s 21us/step - loss: 1.3822 - acc: 0.5297
Epoch 3/20
6258/6258 [==============================] - 0s 20us/step - loss: 1.2863 - acc: 0.5641
Epoch 4/20
6258/6258 [==============================] - 0s 20us/step - loss: 1.2286 - acc: 0.5844
Epoch 5/20
6258/6258 [==============================] - 0s 21us/step - loss: 1.1793 - acc: 0.6056
Epoch 6/20
6258/6258 [==============================] - 0s 20us/step - loss: 1.1390 - acc: 0.6179
Epoch 7/20
6258/6258 [==============================] - 0s 20us/step - loss: 1.1062 - acc: 0.6261
Epoch 8/20
6258/6258 [==============================] - 0s 20us/step - loss: 1.0802 - acc: 0.6318
Epoch 9/20
6258/6258 [==============================] - 0s 19us/step - loss: 1.0571 - acc: 0.6384
Epoch 10/20
6258/6258 [==============================] - 0s 21us/step - loss: 1.0388 - acc: 0.6406
Epoch 11/20
6258/6258 [==============================] - 0s 22us/step - loss: 1.0233 - acc: 0.6483
Epoch 12/20
6258/6258 [==============================] - 0s 21us/step - loss: 1.0090 - acc: 0.6494
Epoch 13/20
6258/6258 [==============================] - 0s 22us/step - loss: 1.0029 - acc: 0.6505
Epoch 14/20
6258/6258 [==============================] - 0s 22us/step - loss: 0.9898 - acc: 0.6582
Epoch 15/20
6258/6258 [==============================] - 0s 20us/step - loss: 0.9827 - acc: 0.6572
Epoch 16/20
6258/6258 [==============================] - 0s 21us/step - loss: 0.9712 - acc: 0.6595
Epoch 17/20
6258/6258 [==============================] - 0s 23us/step - loss: 0.9649 - acc: 0.6641
Epoch 18/20
6258/6258 [==============================] - 0s 25us/step - loss: 0.9535 - acc: 0.6655
Epoch 19/20
6258/6258 [==============================] - 0s 19us/step - loss: 0.9529 - acc: 0.6681
Epoch 20/20
6258/6258 [==============================] - 0s 21us/step - loss: 0.9418 - acc: 0.6697
Evaluate:
Returns the loss value & metrics values for the model in test mode.
2086/2086 [==============================] - 0s 32us/step
```

```
[0.9191266548599287, 0.6706615531547415]
Returns the loss value & metrics values for the model in test mode. Test using validation
2086/2086 [==============================] - 0s 12us/step
[0.9578683657477023, 0.6596356664042231]
```

Errado, pois a separação está diferente da tabela acima, foi usada uma nova porcentagem para testes

Dataset	% de instâncias
Treino	60%
Teste	40%

```
In [51]: from sklearn.model_selection import train_test_split
         x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.4, train_size=0
```

```
In [52]: model = tf.keras.models.Sequential([
           tf.keras.layers.Flatten(),
           tf.keras.layers.Dense(13, activation=tf.nn.relu),
           tf.keras.layers.Dropout(0.1),
           tf.keras.layers.Dense(100, activation=tf.nn.relu),
           tf.keras.layers.Dropout(0.1),
           tf.keras.layers.Dense(100, activation=tf.nn.relu),
           tf.keras.layers.Dropout(0.1),
           tf.keras.layers.Dense(100, activation=tf.nn.relu),
           tf.keras.layers.Dropout(0.1),
           tf.keras.layers.Dense(12, activation=tf.nn.softmax)
         ])
         model.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])

         model.fit(x_train, y_train, epochs=20)

         print("Evaluate:")
         print("Returns the loss value & metrics values for the model in test mode.")
         model.evaluate(x_test, y_test)
```

```
Epoch 1/20
6258/6258 [==============================] - 1s 106us/step - loss: 1.6251 - acc: 0.4765
Epoch 2/20
6258/6258 [==============================] - 0s 38us/step - loss: 1.2016 - acc: 0.5719
Epoch 3/20
6258/6258 [==============================] - 0s 39us/step - loss: 1.0530 - acc: 0.6254
Epoch 4/20
6258/6258 [==============================] - 0s 38us/step - loss: 0.9627 - acc: 0.6548
Epoch 5/20
6258/6258 [==============================] - 0s 39us/step - loss: 0.8949 - acc: 0.6729
```

```
Epoch 6/20
6258/6258 [==============================] - 0s 38us/step - loss: 0.8473 - acc: 0.6833
Epoch 7/20
6258/6258 [==============================] - 0s 39us/step - loss: 0.8087 - acc: 0.6972
Epoch 8/20
6258/6258 [==============================] - 0s 38us/step - loss: 0.7795 - acc: 0.7133
Epoch 9/20
6258/6258 [==============================] - 0s 40us/step - loss: 0.7458 - acc: 0.7202
Epoch 10/20
6258/6258 [==============================] - 0s 40us/step - loss: 0.7226 - acc: 0.7263
Epoch 11/20
6258/6258 [==============================] - 0s 41us/step - loss: 0.6907 - acc: 0.7331
Epoch 12/20
6258/6258 [==============================] - 0s 40us/step - loss: 0.6761 - acc: 0.7408
Epoch 13/20
6258/6258 [==============================] - 0s 41us/step - loss: 0.6506 - acc: 0.7469
Epoch 14/20
6258/6258 [==============================] - 0s 40us/step - loss: 0.6417 - acc: 0.7502
Epoch 15/20
6258/6258 [==============================] - 0s 41us/step - loss: 0.6226 - acc: 0.7558
Epoch 16/20
6258/6258 [==============================] - 0s 40us/step - loss: 0.6003 - acc: 0.7685
Epoch 17/20
6258/6258 [==============================] - 0s 42us/step - loss: 0.5884 - acc: 0.7661
Epoch 18/20
6258/6258 [==============================] - 0s 42us/step - loss: 0.5691 - acc: 0.7702
Epoch 19/20
6258/6258 [==============================] - 0s 43us/step - loss: 0.5449 - acc: 0.7803
Epoch 20/20
6258/6258 [==============================] - 0s 42us/step - loss: 0.5330 - acc: 0.7903
Evaluate:
Returns the loss value & metrics values for the model in test mode.
4172/4172 [==============================] - 0s 44us/step
```

```
Out[52]: [0.631493254338792, 0.7615052731930924]
```

```
In [53]: model = tf.keras.models.Sequential([
           tf.keras.layers.Flatten(),
           tf.keras.layers.Dense(13, activation=tf.nn.relu),
           tf.keras.layers.Dropout(0.1),
           tf.keras.layers.Dense(100, activation=tf.nn.relu),
           tf.keras.layers.Dropout(0.1),
           tf.keras.layers.Dense(100, activation=tf.nn.relu),
           tf.keras.layers.Dropout(0.1),
           tf.keras.layers.Dense(100, activation=tf.nn.relu),
           tf.keras.layers.Dropout(0.1),
           tf.keras.layers.Dense(12, activation=tf.nn.softmax)
```

```
])
model.compile(optimizer=tf.train.GradientDescentOptimizer(0.1),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=20)

print("Evaluate")
print("Returns the loss value & metrics values for the model in test mode.")
model.evaluate(x_test, y_test)
```

```
Epoch 1/20
6258/6258 [==============================] - 0s 76us/step - loss: 1.5845 - acc: 0.4848
Epoch 2/20
6258/6258 [==============================] - 0s 31us/step - loss: 1.2148 - acc: 0.5722
Epoch 3/20
6258/6258 [==============================] - 0s 31us/step - loss: 1.1045 - acc: 0.6026
Epoch 4/20
6258/6258 [==============================] - 0s 32us/step - loss: 1.0166 - acc: 0.6331
Epoch 5/20
6258/6258 [==============================] - 0s 30us/step - loss: 0.9732 - acc: 0.6465
Epoch 6/20
6258/6258 [==============================] - 0s 30us/step - loss: 0.9185 - acc: 0.6657
Epoch 7/20
6258/6258 [==============================] - 0s 31us/step - loss: 0.8870 - acc: 0.6721
Epoch 8/20
6258/6258 [==============================] - 0s 29us/step - loss: 0.8614 - acc: 0.6831
Epoch 9/20
6258/6258 [==============================] - 0s 30us/step - loss: 0.8309 - acc: 0.6956
Epoch 10/20
6258/6258 [==============================] - 0s 30us/step - loss: 0.8052 - acc: 0.6985
Epoch 11/20
6258/6258 [==============================] - 0s 30us/step - loss: 0.7809 - acc: 0.7052
Epoch 12/20
6258/6258 [==============================] - 0s 30us/step - loss: 0.7575 - acc: 0.7183
Epoch 13/20
6258/6258 [==============================] - 0s 30us/step - loss: 0.7318 - acc: 0.7215
Epoch 14/20
6258/6258 [==============================] - 0s 30us/step - loss: 0.7188 - acc: 0.7202
Epoch 15/20
6258/6258 [==============================] - 0s 31us/step - loss: 0.6946 - acc: 0.7320
Epoch 16/20
6258/6258 [==============================] - 0s 29us/step - loss: 0.6698 - acc: 0.7367
Epoch 17/20
6258/6258 [==============================] - 0s 30us/step - loss: 0.6543 - acc: 0.7391
Epoch 18/20
6258/6258 [==============================] - 0s 30us/step - loss: 0.6320 - acc: 0.7459
Epoch 19/20
```

```
6258/6258 [==============================] - 0s 29us/step - loss: 0.6205 - acc: 0.7546
Epoch 20/20
6258/6258 [==============================] - 0s 31us/step - loss: 0.6090 - acc: 0.7584
Evaluate
Returns the loss value & metrics values for the model in test mode.
4172/4172 [==============================] - 0s 38us/step
```

Out[53]: [0.7204558540739241, 0.72315436235896]