

Case study 6

Artificial Neural Networks

Balaji Avvaru, Apurv Mittal, Ravi Sivaraman

Quantifying The World

# Introduction

The goal of this project is to maximize accuracy and minimize loss using a dense neural network. The dataset contains 7 million records (big dataset), with 29 features. The features are just numbered from f0 to f26, label. “# label” is the target variable in the dataset.

# Data Analysis

The dataset contains 7 million records (big dataset), with 29 features. The features are numbered from *f0* to *f26,* label. *“# label*” is the target variable in the dataset.

There are no missing values. This dataset doesn’t require imputation for the features.

The correlation matrix below shows the collinearity among the variables. Data shows there is high collinearity for a few of the attributes.

This study considered all the features with lower than 99% correlation for this analysis. There are features with up to 83% of correlation. Since none of the features showed 99% or above collinearity, all features are considered for modeling, and none are dropped.

### Correlation Analysis

A picture containing text, scoreboard

Description automatically generated

Figure 1: Collinearity heatmap of Attributes

### Target

Target is a binary variable called *‘# label’*, which has value either *0* or *1.*

|  |  |
| --- | --- |
| Figure 2: Target variable distribution plot |  |

# 2. Methods

f9, f13, f17, f21 are categorical variables, which are converted using one-hot encoding.

All the features are normalized using the Standard Scaler technique. After normalization, the data was split into train and test datasets by a factor of 80/20.

The training data was further split into training and validation set by 80/20. The validation set is used to validate the model performance during training.

The test data is unseen data and is used to calculate the final accuracy of the best model and ROC curve, and confusion matrix.

### ANN

Artificial neural networks (ANNs) are models inspired by humans (or biological) neural networks.

An ANN is a collection of neurons (nodes) loosely based on the biological brain interconnected in a dense layer. Each neural is a linear regression created from each feature. The neurons send a signal once it crosses a certain threshold. Usually, they are aggregated into several layers (sequential or dense). The final layer depends on the type of classification required. For binary classification, sigmoid functions are used. The last layer uses a sigmoid function for this study, as the target is a binary classification.

### Layers of Neural Network

In this study, the authors created two sets of neural networks, one with 3 layers and another with 4 layers.

First neural network with the following layers:

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Activation Function | Number of Neurons | Input to Layer |
| 1 | gelu | 512 | Features |
| 2 | gelu | 256 | Layer1 |
| 3 | gelu | 128 | Layer2 |
| 4 | sigmoid | 2 | Layer3 |

Table 1: Layers of first Neural Network

Second neural network with the following layers:

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Activation Function | Number of Neurons | Input to Layer |
| 1 | gelu | 1024 | Features |
| 2 | gelu | 512 | Layer1 |
| 3 | gelu | 256 | Layer2 |
| 4 | gelu | 128 | Layer3 |
| 5 | sigmoid | 2 | Layer4 |

Table 2: Layers of second Neural Network

The hyper-parameters (tunable parameters) are:

Third neural network with the following layers:

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Activation Function | Number of Neurons | Input to Layer |
| 1 | swish | 512 | Features |
| 2 | swish | 256 | Layer1 |
| 3 | swish | 128 | Layer2 |
| 4 | sigmoid | 2 | Layer3 |

Table 3:Layers of third Neural Network

#### Parameters:

*batchsize:* Constant that multiplies the regularization term; the higher the value, the stronger the regularization. Also used to compute the learning rate when *learning*\_*rate* is set to ‘*optimal.’*

*Optimizer:* Adam(lr=1e-2).

*loss*: Sparse Categorical Cross-Entropy

*safety:* Early Stopping (using *val\_loss*)

*patience*: 3

*min\_delta*: 2e-4

The models are compiled to run for 50 epochs. Each epoch runs the entire dataset once. With 50 epochs, the model runs each record in the dataset 50 times or unless early stopping condition is met (when loss stops reducing further). If the accuracy or val\_loss doesn't improve, then the model doesn’t run the entire 50 epochs and stops early. The model is configured to stop if 3 (hyper-param patience) consecutive epochs did not improve val\_loss. This prevents running neural networks when the metrics don’t improve.

# 3. Results

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Batch Size | Dense Neurons | Validation Accuracy |
| gelu/4 Layers | 2048 | 1094 à 512 à 128 | 87.99% |
| gelu/3 Layers | 2048 | 512 à 256 | 88.20% |
| gelu/3 Layers | 1024 | 512 à 256 | 88.07% |
| swish/3 Layers | 2048 | 512 à 256 | 88.12% |
| gelu/3 Layers | 100 | 512 à 256 | 87.63% |

Table 4: Results of various models

All the models are giving similar results. The *gelu* model with 3layers and batch size of *2048* gives the highest accuracy of *88*.*20*%.

When batch size decreases, training time increases (when all others are kept same):

|  |  |
| --- | --- |
| Batch Size | Training Time (in seconds) |
| 2048 | 16 |
| 1024 | 31 |
| 100 | 250 |

The detailed results from the best of the models (gelu/3layers/2048 batch) are listed below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 (target) | 0.9 | 0.85 | 0.88 | 699445 |
| 1 (target) | 0.86 | 0.91 | 0.88 | 700555 |
| accuracy |  |  | 0.88 | 1400000 |
| macro avg | 0.88 | 0.88 | 0.88 | 1400000 |
| weighted avg | 0.88 | 0.88 | 0.88 | 1400000 |

Table 5: Confusion Matrix

Chart, treemap chart

Description automatically generated

Figure 3: Confusion Matrix

# 4. Conclusion

All neural network models produced similar results. In terms of accuracy, there is no significant difference in the model output. Changing the activation function between gelu or swish doesn’t change the final accuracy significantly. Similarly, the addition of the 4th layer doesn’t produce higher accuracy results. The change in batch size between 1024 and 2048 also doesn’t produce a significantly different output.

Due to early stopping criteria, the model stops much earlier than running all 50 epochs as it stops improving further. Overall model accuracy is around 88.20%.

The neural network model was able to accurately predict the existence of new particles with 88% accuracy, we can advise our clients to use neural network model to detect a new particle.

# Appendix – Code

NB Viewer Link:

<https://nbviewer.org/github/ravisiv/CaseStudy6_NN/blob/main/Case%20Study%206.ipynb>

[CaseStudy6\_NN (/github/ravisiv/CaseStudy6\_NN/tree/main)](https://nbviewer.org/github/ravisiv/CaseStudy6_NN/tree/main)

/  [Case Study 6.ipynb (/github/ravisiv/CaseStudy6\_NN/tree/main/Case Study 6.ipynb)](https://nbviewer.org/github/ravisiv/CaseStudy6_NN/tree/main/Case%20Study%206.ipynb)

In [1]: **import** **os** **import** **pandas** **as** **pd** **import** **re** **import** **datetime** **as** **dt** **import** **numpy** **as** **np**

**from** **IPython.display** **import** display **import** **warnings** **import** **seaborn** **as** **sns**

**from** **sklearn.preprocessing** **import** StandardScaler **from** **sklearn** **import** metrics **as** mt

**from** **sklearn.metrics** **import** classification\_report **from** **sklearn.metrics** **import** f1\_score **import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.linear\_model** **import** SGDClassifier

**from** **sklearn.metrics** **import** precision\_recall\_curve, plot\_precision\_reca **from** **sklearn.preprocessing** **import** label\_binarize, StandardScaler **from** **sklearn** **import** metrics **as** mt

**from** **sklearn.model\_selection** **import** cross\_validate, cross\_val\_predict, warnings.filterwarnings('ignore')

In [2]: *# load data*

*#df = pd.read\_csv('all\_train.csv') # read in the csv file*

df = pd.read\_csv('/Users/ravis/Downloads/data/all\_train.csv')

In [3]: df.mass

Out[3]: 0 1000.000000 1 750.000000

1. 750.000000
2. 1250.000000
3. 750.000000 ...
4. 750.000000
5. 1250.000000
6. 1500.000000
7. 1500.000000
8. 499.999969

Name: mass, Length: 7000000, dtype: float64

In [4]:

df

.

info

()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 7000000 entries, 0 to 6999999

Data columns (total 29 columns):

# Column Dtype

--- ------ -----

1. # label float64
2. f0 float64
3. f1 float64
4. f2 float64
5. f3 float64
6. f4 float64
7. f5 float64
8. f6 float64
9. f7 float64
10. f8 float64
11. f9 float64
12. f10 float64
13. f11 float64
14. f12 float64
15. f13 float64
16. f14 float64
17. f15 float64
18. f16 float64
19. f17 float64
20. f18 float64
21. f19 float64
22. f20 float64
23. f21 float64
24. f22 float64
25. f23 float64
26. f24 float64
27. f25 float64
28. f26 float64 28 mass float64 dtypes: float64(29) memory usage: 1.5 GB In [5]:

df

.

describe

()

Out[5]:

**# label f0 f1 f2 f3 f4**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **count** | 7.000000e+06 | 7.000000e+06 | 7.000000e+06 | 7.000000e+06 | 7.000000e+06 | 7.000000e+06 |
| **mean** | 5.001256e-01 | 1.612528e-02 | 4.770022e-04 | 2.686578e-05 | 1.056081e-02 | -1.050026e-04 |
| **std** | 5.000000e-01 | 1.004417e+00 | 9.974864e-01 | 1.000080e+00 | 9.956003e-01 | 9.998670e-01 |
| **min** | 0.000000e+00 | -1.960549e+00 | -2.365355e+00 | -1.732165e+00 | -9.980274e+00 | -1.732137e+00 |
| **25%** | 0.000000e+00 | -7.288206e-01 | -7.332548e-01 | -8.656704e-01 | -6.092291e-01 | -8.658025e-01 |
| **50%** | 1.000000e+00 | -3.930319e-02 | 8.523957e-04 | 3.199154e-04 | 1.963316e-02 | -5.070131e-04 |
| **75%** | 1.000000e+00 | 6.900799e-01 | 7.347832e-01 | 8.659464e-01 | 6.798818e-01 | 8.657646e-01 |
| **max** | 1.000000e+00 | 4.378282e+00 | 2.365287e+00 | 1.732370e+00 | 4.148023e+00 | 1.731978e+00 |
| 8 rows × 29 columns | |

# *Missing value analysis*

|  |  |
| --- | --- |
| Out[6]: In [7]: | 0 **Target** |
| Out[7]: | 1.0 3500879  0.0 3499121  Name: # label, dtype: int64 |

In [6]:

df

[

'# label'

]

.

value\_counts

()

*# Validate null values in the csv file*

df

.

isnull

()

.

sum

()

.

sum

()

In [8]:

sns

.

countplot

(

x

=

"# label"

,

data

=

df

)

plt

.

title

(

"Distribution of Target Values"

)

plt

.

show

()



In [9]:

*# Pie chart*

df

[

'# label'

]

.

value\_counts

()

.

plot

.

pie

(

autopct

=

"

**%.1f%%**

"

)

plt

.

title

(

"Proportion of Target Value"

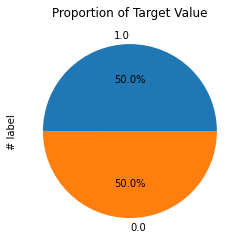
)

plt

.

show

()



# *Independent Variable analysis p y*

In [10]:

*#Visualizing the hist of data to check normality of independent variabl*

df\_X

=

df

.

drop

([

'# label'

]

,

axis

=

1

)

df\_X

.

hist

(

bins

=

50

,

figsize

=

(

25

,

30

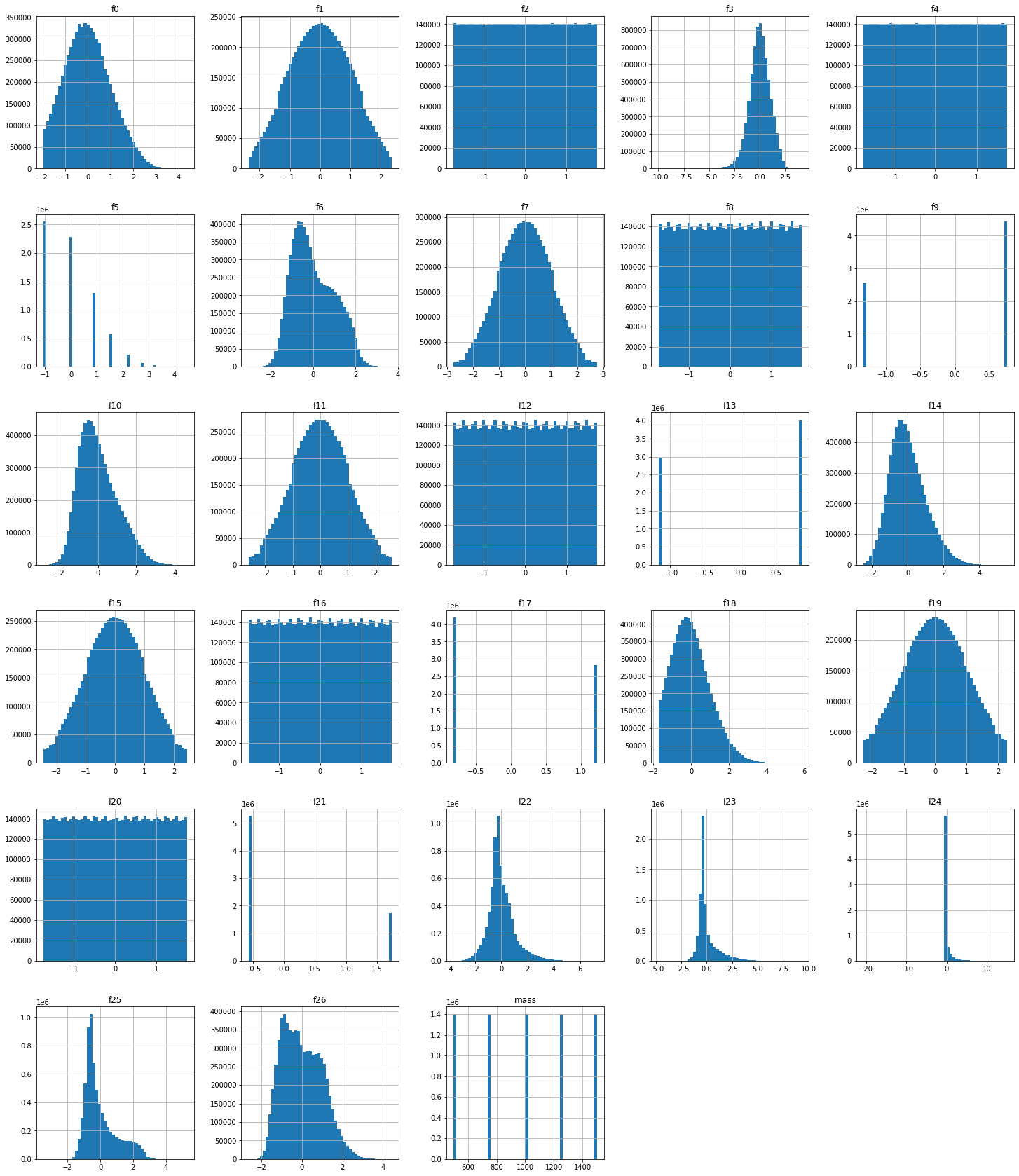
))

plt

.

show

()



In [11]: print(df['f9'].value\_counts()) print(df['f13'].value\_counts()) print(df['f17'].value\_counts()) print(df['f21'].value\_counts())

|  |  |  |  |
| --- | --- | --- | --- |
| In [12]: Out[12]: | 0.754261 4438579 -1.325801 2561421  Name: f9, dtype: int64  0.860649 4027351 -1.161915 2972649  Name: f13, dtype: int64  -0.815440 4187343  1.226331 2812657 Name: f17, dtype: int64  -0.573682 5265796  1.743123 1734204 Name: f21, dtype: int64  df  .  drop  ([  '# label'  ]  ,  axis  =  1  )  .  describe  () |  |  |
|  | **f0 f1 f2 f3** | **f4** | **f5** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **count** | 7.000000e+06 | 7.000000e+06 | 7.000000e+06 | 7.000000e+06 | 7.000000e+06 | 7.000000e+0 |
| **mean** | 1.612528e-02 | 4.770022e-04 | 2.686578e-05 | 1.056081e-02 | -1.050026e-04 | 2.765919e-0 |
| **std** | 1.004417e+00 | 9.974864e-01 | 1.000080e+00 | 9.956003e-01 | 9.998670e-01 | 1.000957e+0 |
| **min** | -1.960549e+00 | -2.365355e+00 | -1.732165e+00 | -9.980274e+00 | -1.732137e+00 | -1.054221e+0 |
| **25%** | -7.288206e-01 | -7.332548e-01 | -8.656704e-01 | -6.092291e-01 | -8.658025e-01 | -1.054221e+0 |
| **50%** | -3.930319e-02 | 8.523957e-04 | 3.199154e-04 | 1.963316e-02 | -5.070131e-04 | -5.983562e-03 |
| **75%** | 6.900799e-01 | 7.347832e-01 | 8.659464e-01 | 6.798818e-01 | 8.657646e-01 | 8.504885e-0 |
| **max** | 4.378282e+00 | 2.365287e+00 | 1.732370e+00 | 4.148023e+00 | 1.731978e+00 | 4.482618e+0 |
| 8 rows × 28 columns | |

In [13]:

*#heatmap - correlation matrix*

plt

.

figure

(

figsize

=

(

60

,

60

))

*#code reference (5-1)*

plt

.

xticks

(

rotation

=

90

,

fontsize

=

35

)

plt

.

yticks

(

rotation

=

180

,

fontsize

=

35

)

ax

=

sns

.

heatmap

(

df\_X

.

corr

,

()

annot

=

**False**

,

cbar

=

**True**

,

annot\_kws

=

{

"size"

:

2

cbar

=

ax

.

collections

[

0

]

.

colorbar

cbar

.

ax

.

tick\_params

(

labelsize

=

30

)

plt

.

title

(

'HeatMap-Correlation Matrix'

,

fontsize

=

45

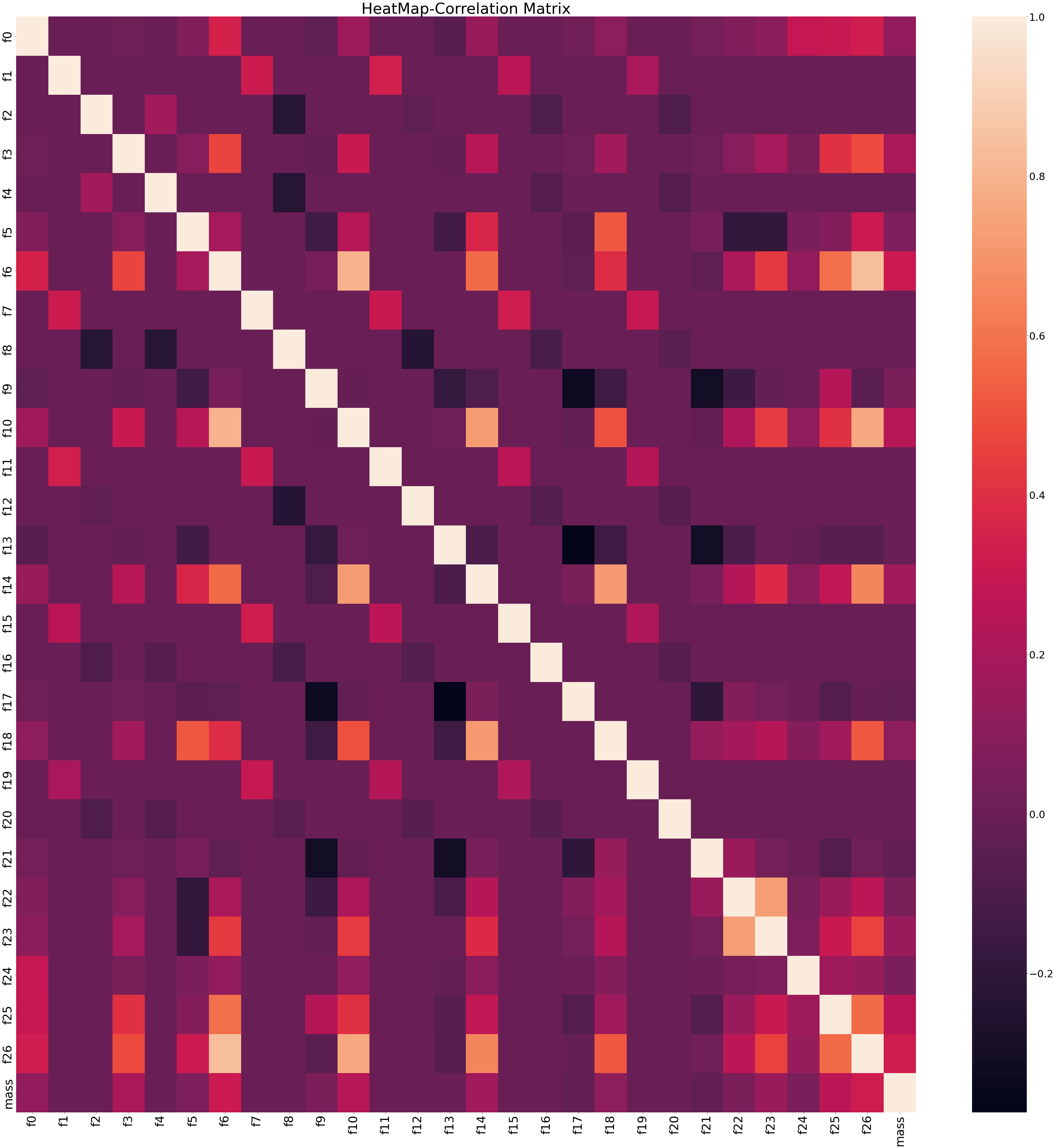
)

plt

.

show

()



# *Check for Multicolliniarity*

In [14]: *#https://www.projectpro.io/recipes/drop-out-highly-correlated-features-*

*# to drop features with colliniarity more than 95%* pd.set\_option('display.max\_rows', 100)

corr\_df = pd.DataFrame(df\_X.corr().abs()) corr\_df.head(100)

Out[14]:

**f0 f1 f2 f3 f4 f5 f6 f7 f**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **f0** | 1.000000 | 0.000556 | 0.000321 | 0.012037 | 0.000464 | 0.078401 | 0.349973 | 0.000026 | 0.00092 |
| **f1** | 0.000556 | 1.000000 | 0.000200 | 0.000706 | 0.000131 | 0.000511 | 0.000454 | 0.315357 | 0.00058 |
| **f2** | 0.000321 | 0.000200 | 1.000000 | 0.000074 | 0.174967 | 0.000162 | 0.000436 | 0.000024 | 0.23231 |
| **f3** | 0.012037 | 0.000706 | 0.000074 | 1.000000 | 0.000385 | 0.092129 | 0.468157 | 0.000091 | 0.00018 |
| **f4** | 0.000464 | 0.000131 | 0.174967 | 0.000385 | 1.000000 | 0.000496 | 0.000307 | 0.000435 | 0.23300 |
| **f5** | 0.078401 | 0.000511 | 0.000162 | 0.092129 | 0.000496 | 1.000000 | 0.191900 | 0.000740 | 0.00067 |
| **f6** | 0.349973 | 0.000454 | 0.000436 | 0.468157 | 0.000307 | 0.191900 | 1.000000 | 0.000491 | 0.00069 |
| **f7** | 0.000026 | 0.315357 | 0.000024 | 0.000091 | 0.000435 | 0.000740 | 0.000491 | 1.000000 | 0.00088 |
| **f8** | 0.000924 | 0.000580 | 0.232319 | 0.000188 | 0.233003 | 0.000671 | 0.000699 | 0.000887 | 1.00000 |
| **f9** | 0.039924 | 0.000141 | 0.000426 | 0.021872 | 0.000084 | 0.143509 | 0.038961 | 0.000328 | 0.00054 |
| **f10** | 0.171641 | 0.000726 | 0.000291 | 0.306169 | 0.000229 | 0.243807 | 0.799686 | 0.000724 | 0.00048 |
| **f11** | 0.000564 | 0.336327 | 0.000268 | 0.000073 | 0.000174 | 0.000271 | 0.000290 | 0.306276 | 0.00046 |
| **f12** | 0.000268 | 0.000218 | 0.036190 | 0.000460 | 0.001119 | 0.000454 | 0.000685 | 0.000132 | 0.23824 |
| **f13** | 0.057995 | 0.000088 | 0.000378 | 0.024117 | 0.000092 | 0.137170 | 0.000264 | 0.000083 | 0.00012 |
| **f14** | 0.147686 | 0.000375 | 0.000017 | 0.244698 | 0.000189 | 0.363165 | 0.575503 | 0.000471 | 0.00040 |
| **f15** | 0.000105 | 0.256024 | 0.000185 | 0.001113 | 0.000011 | 0.000005 | 0.000920 | 0.324219 | 0.00046 |
| **f16** | 0.000684 | 0.000249 | 0.091000 | 0.000181 | 0.065579 | 0.000300 | 0.000205 | 0.000238 | 0.10618 |
| **f17** | 0.027194 | 0.000136 | 0.000912 | 0.008179 | 0.000023 | 0.042581 | 0.038356 | 0.000435 | 0.00047 |
| **f18** | 0.119041 | 0.000079 | 0.000026 | 0.174068 | 0.000338 | 0.517752 | 0.390014 | 0.000251 | 0.00023 |
| **f19** | 0.000152 | 0.195312 | 0.000399 | 0.000328 | 0.000400 | 0.000337 | 0.000226 | 0.299012 | 0.00012 |
| **f20** | 0.000617 | 0.000254 | 0.092043 | 0.000047 | 0.068271 | 0.000305 | 0.000303 | 0.000276 | 0.05167 |
| **f21** | 0.034794 | 0.000191 | 0.000312 | 0.010787 | 0.000485 | 0.044609 | 0.034123 | 0.000131 | 0.00004 |
| **f22** | 0.080169 | 0.000100 | 0.000171 | 0.096627 | 0.000136 | 0.195117 | 0.209496 | 0.000023 | 0.00009 |
| **f23** | 0.109947 | 0.000384 | 0.000075 | 0.190014 | 0.000061 | 0.188721 | 0.433089 | 0.000133 | 0.00035 |
| **f24** | 0.289670 | 0.000178 | 0.000499 | 0.056976 | 0.000009 | 0.050882 | 0.126159 | 0.000182 | 0.00065 |
| **f25** | 0.296976 | 0.000563 | 0.000186 | 0.406295 | 0.000346 | 0.082941 | 0.581645 | 0.000020 | 0.00037 |
| **f26** | 0.327533 | 0.000782 | 0.000315 | 0.482503 | 0.000414 | 0.308311 | 0.835995 | 0.000621 | 0.00029 |
| **mass** | 0.126717 | 0.000050 | 0.000169 | 0.203470 | 0.000002 | 0.064497 | 0.310024 | 0.000069 | 0.00075 |
| 28 rows × 28 columns | | |

In [15]: *# Multi Colliniarity analysis on Independent variables*  upper\_tri = corr\_df.where(np.triu(np.ones(corr\_df.shape),k=1).astype(np print(upper\_tri)

f0 f1 f2 f3 f4 f5 f6 \ f0 NaN 0.000556 0.000321 0.012037 0.000464 0.078401 0.349973 f1 NaN NaN 0.000200 0.000706 0.000131 0.000511 0.000454 f2 NaN NaN NaN 0.000074 0.174967 0.000162 0.000436 f3 NaN NaN NaN NaN 0.000385 0.092129 0.468157 f4 NaN NaN NaN NaN NaN 0.000496 0.000307 f5 NaN NaN NaN NaN NaN NaN 0.191900 f6 NaN NaN NaN NaN NaN NaN NaN f7 NaN NaN NaN NaN NaN NaN NaN f8 NaN NaN NaN NaN NaN NaN NaN f9 NaN NaN NaN NaN NaN NaN NaN f10 NaN NaN NaN NaN NaN NaN NaN f11 NaN NaN NaN NaN NaN NaN NaN f12 NaN NaN NaN NaN NaN NaN NaN f13 NaN NaN NaN NaN NaN NaN NaN f14 NaN NaN NaN NaN NaN NaN NaN f15 NaN NaN NaN NaN NaN NaN NaN f16 NaN NaN NaN NaN NaN NaN NaN f17 NaN NaN NaN NaN NaN NaN NaN f18 NaN NaN NaN NaN NaN NaN NaN f19 NaN NaN NaN NaN NaN NaN NaN f20 NaN NaN NaN NaN NaN NaN NaN f21 NaN NaN NaN NaN NaN NaN NaN f22 NaN NaN NaN NaN NaN NaN NaN f23 NaN NaN NaN NaN NaN NaN NaN f24 NaN NaN NaN NaN NaN NaN NaN f25 NaN NaN NaN NaN NaN NaN NaN f26 NaN NaN NaN NaN NaN NaN NaN mass NaN NaN NaN NaN NaN NaN NaN

f7 f8 f9 ... f18 f19 f20 f0 0.000026 0.000924 0.039924 ... 0.119041 0.000152 0.000617 f1 0.315357 0.000580 0.000141 ... 0.000079 0.195312 0.000254 f2 0.000024 0.232319 0.000426 ... 0.000026 0.000399 0.092043 f3 0.000091 0.000188 0.021872 ... 0.174068 0.000328 0.000047 f4 0.000435 0.233003 0.000084 ... 0.000338 0.000400 0.068271 f5 0.000740 0.000671 0.143509 ... 0.517752 0.000337 0.000305 f6 0.000491 0.000699 0.038961 ... 0.390014 0.000226 0.000303 f7 NaN 0.000887 0.000328 ... 0.000251 0.299012 0.000276 f8 NaN NaN 0.000548 ... 0.000235 0.000124 0.051674 f9 NaN NaN NaN ... 0.146027 0.000023 0.000175 f10 NaN NaN NaN ... 0.497812 0.000294 0.000334 f11 NaN NaN NaN ... 0.000089 0.239606 0.000590 f12 NaN NaN NaN ... 0.000126 0.000859 0.054172 f13 NaN NaN NaN ... 0.148374 0.000217 0.000031 f14 NaN NaN NaN ... 0.714613 0.000295 0.000070 f15 NaN NaN NaN ... 0.000265 0.221685 0.000307 f16 NaN NaN NaN ... 0.000242 0.000065 0.055865 f17 NaN NaN NaN ... 0.005241 0.000265 0.000239 f18 NaN NaN NaN ... NaN 0.000117 0.000113 f19 NaN NaN NaN ... NaN NaN 0.000138 f20 NaN NaN NaN ... NaN NaN NaN f21 NaN NaN NaN ... NaN NaN NaN f22 NaN NaN NaN ... NaN NaN NaN f23 NaN NaN NaN ... NaN NaN NaN f24 NaN NaN NaN ... NaN NaN NaN f25 NaN NaN NaN ... NaN NaN NaN f26 NaN NaN NaN ... NaN NaN NaN mass NaN NaN NaN ... NaN NaN NaN

f21 f22 f23 f24 f25 f26 m f0 0.034794 0.080169 0.109947 0.289670 0.296976 0.327533 0.126 f1 0.000191 0.000100 0.000384 0.000178 0.000563 0.000782 0.000 f2 0.000312 0.000171 0.000075 0.000499 0.000186 0.000315 0.000 f3 0.010787 0.096627 0.190014 0.056976 0.406295 0.482503 0.203 f4 0.000485 0.000136 0.000061 0.000009 0.000346 0.000414 0.000 f5 0.044609 0.195117 0.188721 0.050882 0.082941 0.308311 0.064 f6 0.034123 0.209496 0.433089 0.126159 0.581645 0.835995 0.310 f7 0.000131 0.000023 0.000133 0.000182 0.000020 0.000621 0.000 f8 0.000047 0.000099 0.000354 0.000657 0.000375 0.000296 0.000 f9 0.307213 0.155551 0.015394 0.000711 0.235398 0.049122 0.055 f10 0.024445 0.211504 0.441256 0.123966 0.407570 0.760416 0.245 f11 0.000412 0.000509 0.000661 0.000559 0.001009 0.000406 0.000 f12 0.000263 0.000215 0.000447 0.000077 0.000146 0.000387 0.000 f13 0.306429 0.103286 0.000119 0.018679 0.055174 0.066454 0.000 f14 0.039548 0.234758 0.380652 0.104323 0.279477 0.656016 0.174 f15 0.000801 0.000806 0.001035 0.000278 0.000507 0.000914 0.000 f16 0.000275 0.000115 0.000236 0.000256 0.000441 0.000048 0.000 f17 0.205639 0.076665 0.033371 0.001520 0.076047 0.009786 0.028 f18 0.144685 0.184386 0.235264 0.082366 0.176284 0.521591 0.117 f19 0.000133 0.000185 0.000421 0.000060 0.000246 0.000005 0.000 f20 0.000442 0.000109 0.000493 0.000050 0.000220 0.000070 0.000 f21 NaN 0.153676 0.037039 0.003852 0.077928 0.021282 0.028 f22 NaN NaN 0.727996 0.039866 0.155997 0.261498 0.057 f23 NaN NaN NaN 0.064041 0.302942 0.461259 0.155 f24 NaN NaN NaN NaN 0.169888 0.141599 0.051 f25 NaN NaN NaN NaN NaN 0.572478 0.253 f26 NaN NaN NaN NaN NaN NaN 0.323 mass NaN NaN NaN NaN NaN NaN

[28 rows x 28 columns]

In [16]: to\_drop = [column **for** column **in** upper\_tri.columns **if** any(upper\_tri[colu print((to\_drop))

[]

In [17]: df = pd.get\_dummies(df, columns=['f9', 'f13', 'f17', 'f21'], prefix=['f

In [18]: X = df.drop(['# label'],axis=1) ind\_columns = df.drop('# label',axis=1).columns y = df['# label']

We did normalize the attributes using StandardScaler() to scale them between 0 and 1 before running models.

In [19]:

*# Normalize the data*

scaler

=

StandardScaler

()

X\_scaled

=

scaler

.

fit\_transform

(

X

)

**from**

**sklearn.model\_selection**

**import**

train\_test\_split

*#Direct train/test split*

X\_train

,

X\_test

,

Y\_train

,

Y\_test

=

train\_test\_split

(

X\_scaled

,

y

,

test\_size

=

0.20

,

random\_state

=

1234

)

In [20]:

**Neural Network Model**

In [21]:

**import**

**tensorflow**

**as**

**tf**

**from**

**tensorflow.keras.callbacks**

**import**

EarlyStopping

**from**

**tensorflow.keras.optimizers**

**import**

Adam

my\_model

=

tf

.

keras

.

Model

()

In [22]:

Metal device set to: Apple M1 Pro

2022-03-25 16:26:02.223468: I tensorflow/core/common\_runtime/pluggable\_

2022-03-25 16:26:02.223589: I tensorflow/core/common\_runtime/pluggable\_

# Model 1 gelu - Four layers

In [23]: layer\_zero = tf.keras.Input(shape=(32,)) layer1 = tf.keras.layers.Dense(1094, activation='gelu')(layer\_zero) layer2 = tf.keras.layers.Dense(512, activation='gelu')(layer1) layer3 = tf.keras.layers.Dense(128, activation='gelu')(layer2) layer4 = tf.keras.layers.Dense(2, activation='sigmoid')(layer2)

my\_model = tf.keras.Model(inputs=layer\_zero, outputs=layer4)

In [24]: my\_model.compile(optimizer=Adam(lr=1e-2), loss=tf.keras.losses.SparseCa safety = EarlyStopping(monitor='val\_loss', patience=3, min\_delta=2e-4)

x\_train, x\_val, y\_train, y\_val = train\_test\_split(X\_train, Y\_train, tes

In [25]: nn\_history = my\_model.fit(x\_train, y\_train, validation\_data=(x\_val, y\_val), callbacks=[safety], epochs=50, batch\_size=100)

Epoch 1/50

2022-03-25 16:26:04.870768: W tensorflow/core/platform/profile\_utils/cp

2022-03-25 16:26:05.102127: I tensorflow/core/grappler/optimizers/custo

2188/2188 [==============================] - ETA: 0s - loss: 0.2856 - a

2022-03-25 16:26:34.969380: I tensorflow/core/grappler/optimizers/custo

2188/2188 [==============================] - 33s 15ms/step - loss: 0.28 Epoch 2/50

2188/2188 [==============================] - 33s 15ms/step - loss: 0.27 Epoch 3/50

2188/2188 [==============================] - 31s 14ms/step - loss: 0.26 Epoch 4/50

2188/2188 [==============================] - 30s 14ms/step - loss: 0.26 Epoch 5/50

2188/2188 [==============================] - 30s 14ms/step - loss: 0.26 Epoch 6/50

2188/2188 [==============================] - 31s 14ms/step - loss: 0.26 Epoch 7/50

2188/2188 [==============================] - 31s 14ms/step - loss: 0.26 Epoch 8/50

In [26]:

my\_model

.

summary

()

2188/2188 [==============================] - 31s 14ms/step -

loss

: 0.26

Model: "model\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # ================================================================= input\_1 (InputLayer) [(None, 32)] 0 dense (Dense) (None, 1094) 36102 dense\_1 (Dense) (None, 512) 560640 dense\_3 (Dense) (None, 2) 1026

================================================================= Total params: 597,768

Trainable params: 597,768

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [27]: history\_df = pd.DataFrame(nn\_history.history) history\_df[['loss', 'val\_loss']].plot()

history\_df = pd.DataFrame(nn\_history.history) history\_df[['accuracy', 'val\_accuracy']].plot()

Out[27]: <AxesSubplot:>

Chart, line chart

Description automatically generated

Line chart

Description automatically generated

# Model 1 gelu - 3 layers

In [53]: layer\_zero = tf.keras.Input(shape=(32,)) layer1 = tf.keras.layers.Dense(512, activation='gelu')(layer\_zero) layer2 = tf.keras.layers.Dense(256, activation='gelu')(layer1) *#layer3 = tf.keras.layers.Dense(128, activation='gelu')(layer2)* layer4 = tf.keras.layers.Dense(2, activation='sigmoid')(layer2)

my\_model = tf.keras.Model(inputs=layer\_zero, outputs=layer4)

In [54]: my\_model.compile(optimizer=Adam(lr=1e-2), loss=tf.keras.losses.SparseCa safety = EarlyStopping(monitor='val\_loss', patience=3, min\_delta=2e-4)

x\_train, x\_val, y\_train, y\_val = train\_test\_split(X\_train, Y\_train, tes

In [55]: nn\_history = my\_model.fit(x\_train, y\_train, validation\_data=(x\_val, y\_val), callbacks=[safety], epochs=50, batch\_size=100)

Epoch 1/50

2022-03-26 10:27:51.674809: I tensorflow/core/grappler/optimizers/custo

44800/44800 [==============================] - ETA: 0s - loss: 0.2929 -

2022-03-26 10:31:34.644074: I tensorflow/core/grappler/optimizers/custo

44800/44800 [==============================] - 255s 6ms/step - loss: 0 Epoch 2/50

44800/44800 [==============================] - 249s 6ms/step - loss: 0 Epoch 3/50

44800/44800 [==============================] - 251s 6ms/step - loss: 0 Epoch 4/50

44800/44800 [==============================] - 253s 6ms/step - loss: 0 Epoch 5/50

44800/44800 [==============================] - 245s 5ms/step - loss: 0 Epoch 6/50

44800/44800 [==============================] - 244s 5ms/step - loss: 0

In [56]: y\_pred=my\_model.predict(X\_test)

cm = confusion\_matrix(Y\_test, np.argmax(y\_pred, axis=1))

cm\_plot\_label =['No', 'Yes'] plot\_confusion\_matrix(cm, cm\_plot\_label, title ='Confusion Matrix')

2022-03-26 10:52:50.264679: I tensorflow/core/grappler/optimizers/custo

[[601100 98345]

[ 74978 625577]]

Chart, treemap chart

Description automatically generated with medium confidence

In [57]:

my\_model

.

summary

()

Model: "model\_5"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # ================================================================= input\_5 (InputLayer) [(None, 32)] 0 dense\_13 (Dense) (None, 512) 16896 dense\_14 (Dense) (None, 256) 131328 dense\_15 (Dense) (None, 2) 514 ================================================================= Total params: 148,738

Trainable params: 148,738

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [32]: history\_df = pd.DataFrame(nn\_history.history) history\_df[['loss', 'val\_loss']].plot()

history\_df = pd.DataFrame(nn\_history.history) history\_df[['accuracy', 'val\_accuracy']].plot()

Out[32]: <AxesSubplot:>

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

# Batch size 1024

In [33]: nn\_history = my\_model.fit(x\_train, y\_train, validation\_data=(x\_val, y\_val), callbacks=[safety], epochs=50, batch\_size=1024)

Epoch 1/50

2022-03-25 16:35:50.010830: I tensorflow/core/grappler/optimizers/custo

4375/4375 [==============================] - 29s 7ms/step - loss: 0.267 Epoch 2/50

4375/4375 [==============================] - 31s 7ms/step - loss: 0.266 Epoch 3/50

4375/4375 [==============================] - 29s 7ms/step - loss: 0.266 Epoch 4/50

4375/4375 [==============================] - 29s 7ms/step - loss: 0.266 Epoch 5/50

4375/4375 [==============================] - 29s 7ms/step - loss: 0.266 Epoch 6/50

4375/4375 [==============================] - 27s 6ms/step - loss: 0.266 Epoch 7/50

In [34]:

my\_model

.

summary

()

4375/4375 [==============================] - 28s 6ms/step -

loss

: 0.265

Model: "model\_2"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # ================================================================= input\_2 (InputLayer) [(None, 32)] 0 dense\_4 (Dense) (None, 512) 16896 dense\_5 (Dense) (None, 256) 131328 dense\_6 (Dense) (None, 2) 514 ================================================================= Total params: 148,738

Trainable params: 148,738

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [ ]:

In [35]: history\_df = pd.DataFrame(nn\_history.history) history\_df[['loss', 'val\_loss']].plot()

history\_df = pd.DataFrame(nn\_history.history) history\_df[['accuracy', 'val\_accuracy']].plot()

Out[35]: <AxesSubplot:>

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

# Model 2 Swish - 3 layers

In [36]: layer\_zero = tf.keras.Input(shape=(32,)) layer1 = tf.keras.layers.Dense(512, activation='swish')(layer\_zero) layer2 = tf.keras.layers.Dense(256, activation='swish')(layer1) *#layer3 = tf.keras.layers.Dense(128, activation='gelu')(layer2)* layer4 = tf.keras.layers.Dense(2, activation='sigmoid')(layer2)

my\_model = tf.keras.Model(inputs=layer\_zero, outputs=layer4)

In [37]: my\_model.compile(optimizer=Adam(lr=1e-2), loss=tf.keras.losses.SparseCa safety = EarlyStopping(monitor='val\_loss', patience=3, min\_delta=2e-4)

x\_train, x\_val, y\_train, y\_val = train\_test\_split(X\_train, Y\_train, tes

In [38]: nn\_history = my\_model.fit(x\_train, y\_train, validation\_data=(x\_val, y\_val), callbacks=[safety], epochs=50, batch\_size=2048)

Epoch 1/50

2022-03-25 16:39:18.261506: I tensorflow/core/grappler/optimizers/custo

2188/2188 [==============================] - ETA: 0s - loss: 0.2841 - a

2022-03-25 16:39:33.405411: I tensorflow/core/grappler/optimizers/custo

2188/2188 [==============================] - 17s 8ms/step - loss: 0.284 Epoch 2/50

2188/2188 [==============================] - 16s 7ms/step - loss: 0.271 Epoch 3/50

2188/2188 [==============================] - 16s 7ms/step - loss: 0.269 Epoch 4/50

2188/2188 [==============================] - 16s 7ms/step - loss: 0.268 Epoch 5/50

2188/2188 [==============================] - 16s 8ms/step - loss: 0.267 Epoch 6/50

2188/2188 [==============================] - 16s 7ms/step - loss: 0.266 Epoch 7/50

2188/2188 [==============================] - 17s 8ms/step - loss: 0.265 Epoch 8/50

2188/2188 [==============================] - 16s 7ms/step - loss: 0.264 Epoch 9/50

2188/2188 [==============================] - 16s 7ms/step - loss: 0.265 Epoch 10/50

2188/2188 [==============================] - 16s 7ms/step - loss: 0.264 Epoch 11/50

2188/2188 [==============================] - 16s 7ms/step - loss: 0.264

In [39]:

my\_model

.

summary

()

Model: "model\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # ================================================================= input\_3 (InputLayer) [(None, 32)] 0 dense\_7 (Dense) (None, 512) 16896 dense\_8 (Dense) (None, 256) 131328 dense\_9 (Dense) (None, 2) 514 =================================================================

Total params: 148,738

Trainable params: 148,738

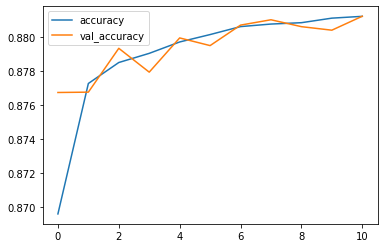
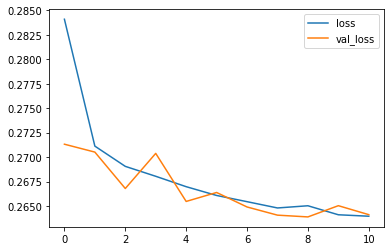
Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [40]: history\_df = pd.DataFrame(nn\_history.history) history\_df[['loss', 'val\_loss']].plot()

history\_df = pd.DataFrame(nn\_history.history) history\_df[['accuracy', 'val\_accuracy']].plot()

Out[40]: <AxesSubplot:>



In [41]:

my\_model

.

summary

()

Model: "model\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # ================================================================= input\_3 (InputLayer) [(None, 32)] 0 dense\_7 (Dense) (None, 512) 16896 dense\_8 (Dense) (None, 256) 131328 dense\_9 (Dense) (None, 2) 514 =================================================================

Total params: 148,738

Trainable params: 148,738

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Model Evaluation**

In [42]:

y\_pred

=

my\_model

.

predict

(

X\_test

)

y\_pred

2022-03-25 16:42:18.676823:

I tensorflow/core/grappler/optimizers/custo

array([[0.48715827, 0.46217152],

[0.08716998, 0.96522045],

In [43]:

Out[43]:

[0.5221664 , 0.43013135], ...,

[0.23122704, 0.9607484 ],

[0.17528357, 0.8504923 ],

[0.24064562, 0.7891484 ]], dtype=float32)

In [44]: **from** **sklearn.metrics** **import** classification\_report print(classification\_report(Y\_test, np.argmax(y\_pred, axis=1)))

precision recall f1-score support

0.0 0.90 0.85 0.88 699445

1.0 0.86 0.91 0.88 700555

accuracy 0.88 1400000 macro avg 0.88 0.88 0.88 1400000 weighted avg 0.88 0.88 0.88 1400000

In [47]: **from** **sklearn.metrics** **import** confusion\_matrix **import** **itertools**

**def** plot\_confusion\_matrix(cm, classes,

title='Confusion matrix', cmap=plt.cm.Blues):

print(cm)

plt.imshow(cm, interpolation='nearest', cmap=cmap) plt.title(title) plt.colorbar()

tick\_marks = np.arange(len(classes)) plt.xticks(tick\_marks, classes, rotation=55) plt.yticks(tick\_marks, classes)

*#fmt = '.2f''d'* thresh = cm.max() / 2.

**for** i, j **in** itertools.product(range(cm.shape[0]), range(cm.shape[1] plt.text(j, i, format(cm[i, j]), horizontalalignment="center",

color="white" **if** cm[i, j] > thresh **else** "black")

plt.ylabel('True label') plt.xlabel('Predicted label') plt.tight\_layout()

cm = confusion\_matrix(Y\_test, np.argmax(y\_pred, axis=1))

cm\_plot\_label =['No', 'Yes']

plot\_confusion\_matrix(cm, cm\_plot\_label, title ='Confusion Matrix')

[[597416 102029]

[ 64132 636423]]

Chart, treemap chart

Description automatically generated

In [46]:

**from**

**sklearn.metrics**

**import**

roc\_auc\_score

,

auc

**from**

**sklearn.metrics**

**import**

roc\_curve

roc\_log

=

roc\_auc\_score

(

Y\_test

,

np

.

argmax

(

y\_pred

,

axis

=

1

))

false\_positive\_rate

,

true\_positive\_rate

,

threshold

=

roc\_curve

(

Y\_test

,

area\_under\_curve

=

auc

(

false\_positive\_rate

,

true\_positive\_rate

)

plt

.

plot

([

0

,

1

]

,

[

0

,

1

]

,

'r--'

)

plt

.

plot

(

false\_positive\_rate

,

true\_positive\_rate

,

label

=

'AUC =

**{:.3f}**

'

.

plt

.

xlabel

(

'False positive rate'

)

plt

.

ylabel

(

'True positive rate'

)

plt

.

title

(

'ROC curve'

)

plt

.

legend

(

loc

=

'best'

)

plt

.

show

()

plt

.

close

()

In [ ]:

In [ ]:

In [ ]:

