Group Project Neural Network

Adam Threlfall, Ulrich Pogson

October 29, 2021

1 Loading Data

For this project, since the data that needs to be processed is stored as a `.csv` file, the data can be loaded with `pandas`. Using `pandas` allows for the data to maintain the format of a table with columns and tuples.

[ ]:
```
import pandas as pd
data = pd.read_csv('project_data.csv')
```

The labled columns represent each atribute being considered in the classification and the tupels contain the data of a specific client that is being assesed. The `pandas` table of data can be seen below:

[ ]:
```
data
```

[ ]:
```
     age  surgery  docvisit allergy  med      disease  bmi class
0     20        0         2      no   no  cholesterol   28   low
1     21        0         4      no   no           no   23   low
2     22        0         3      no   no           no   23   low
3     23        0         3      no   no           no   23   low
4     24        0         3      no   no           no   21   low
..   ...      ...       ...     ...  ...          ...  ...   ...
112   88        0         2     yes  yes           no   21   low
113   88        0         2      no   no           no   22   low
114   88        2        18     yes  yes           no   28  high
115   88        1         5      no  yes     diabetes   33  high
116   88        2        17      no  yes     diabetes   32  high

[117 rows x 8 columns]
```

2 Preprocessing

In order to be able to train a neural network with this data some preprocessing needs to be done. The data contains strings that need to be replaced with an 'equivalent' integers.

[ ]:
```
data = data.replace({'allergy': {'no': 0, 'yes': 1}})
data = data.replace({'med': {'no': 0, 'yes':1}})
data = data.replace({'disease': {'no': 0, 'cholesterol':1, 'heart':2,␣
 ↪'diabetes':3}})
```

```
data = data.replace({'class': {'low': 0, 'medium':1, 'high':2}})
data
```

[ ]:
```
     age  surgery  docvisit  allergy  med  disease  bmi  class
0     20        0         2        0    0        1   28      0
1     21        0         4        0    0        0   23      0
2     22        0         3        0    0        0   23      0
3     23        0         3        0    0        0   23      0
4     24        0         3        0    0        0   21      0
..   ...      ...       ...      ...  ...      ...  ...    ...
112   88        0         2        1    1        0   21      0
113   88        0         2        0    0        0   22      0
114   88        2        18        1    1        0   28      2
115   88        1         5        0    1        3   33      2
116   88        2        17        0    1        3   32      2

[117 rows x 8 columns]
```

As can be seen below, `data` is now homogenous with dtype `int`.

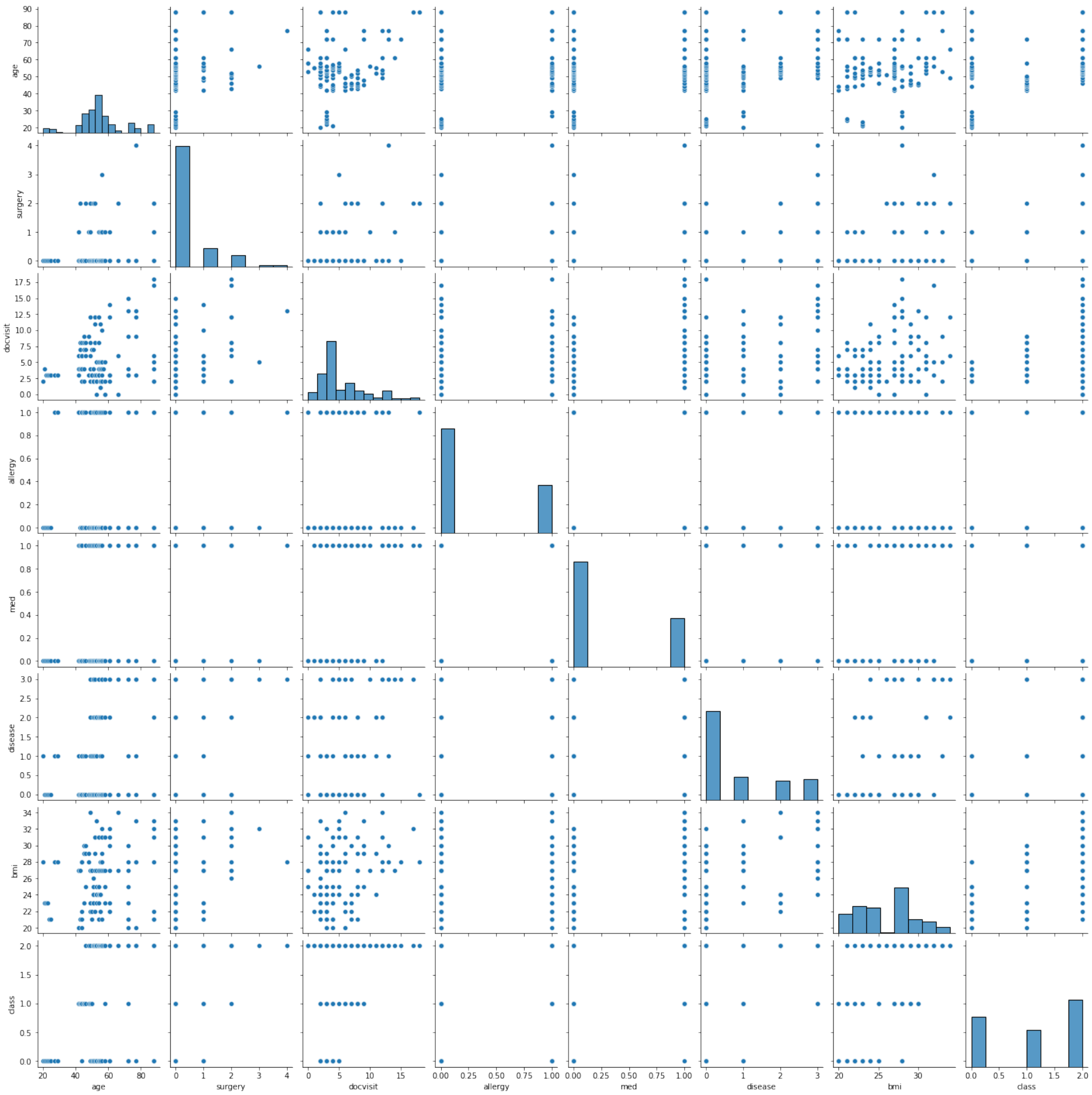
[ ]:
```
data.dtypes
```

[ ]:
```
age         int64
surgery     int64
docvisit    int64
allergy     int64
med         int64
disease     int64
bmi         int64
class       int64
dtype: object
```

The validity of the data can be tested using `seaborn.pairplot`.

[ ]:
```
import seaborn as sns
sns.pairplot(data)
```

[ ]: <seaborn.axisgrid.PairGrid at 0x7fa09fdd0910>



The `data` table needs to be split so that the class category is contained within its own `target` table.

[ ]:
```
target = data.pop('class')
```

The result is the `data` table no longer contains the class category and a new `target` table is created.

[ ]:
```
data
```

[ ]:

	age	surgery	docvisit	allergy	med	disease	bmi
0	20	0	2	0	0	1	28
1	21	0	4	0	0	0	23
2	22	0	3	0	0	0	23
3	23	0	3	0	0	0	23
4	24	0	3	0	0	0	21

```
..   ...     ...      ...      ...  ...      ...  ...
112   88       0        2        1    1        0   21
113   88       0        2        0    0        0   22
114   88       2       18        1    1        0   28
115   88       1        5        0    1        3   33
116   88       2       17        0    1        3   32

[117 rows x 7 columns]
```

[ ]:
```
target
```

[ ]:
```
0      0
1      0
2      0
3      0
4      0
      ..
112    0
113    0
114    2
115    2
116    2
Name: class, Length: 117, dtype: int64
```

2.1 Neural Network Training

The neural network can now be trained. The neural network must have an output layer with three nodes, where each one represents one of the possible classifications; 'low', 'medium', or 'high'.

[ ]:
```
from keras.models import Sequential
from keras.layers import Dense,Flatten,Activation,Dropout

model = Sequential()
model.add(Flatten())
model.add(Dense(100, input_dim=7, activation='relu'))
model.add(Dense(3, activation='softmax'))

model.
 ↪compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])

history = model.fit(data, target, validation_split=0.2, epochs=50,␣
 ↪batch_size=16, verbose = 2)
```

```
Epoch 1/50
6/6 - 1s - loss: 1.7776 - accuracy: 0.3226 - val_loss: 2.7411 - val_accuracy:
0.4583
Epoch 2/50
6/6 - 0s - loss: 1.5499 - accuracy: 0.3333 - val_loss: 2.4250 - val_accuracy:
0.4167
```

```
Epoch 3/50
6/6 - 0s - loss: 1.3363 - accuracy: 0.3011 - val_loss: 1.3468 - val_accuracy:
0.4583
Epoch 4/50
6/6 - 0s - loss: 1.2309 - accuracy: 0.3441 - val_loss: 1.4546 - val_accuracy:
0.5417
Epoch 5/50
6/6 - 0s - loss: 1.1132 - accuracy: 0.4624 - val_loss: 1.3112 - val_accuracy:
0.5417
Epoch 6/50
6/6 - 0s - loss: 1.0427 - accuracy: 0.4624 - val_loss: 1.0399 - val_accuracy:
0.6250
Epoch 7/50
6/6 - 0s - loss: 0.9762 - accuracy: 0.4946 - val_loss: 0.9499 - val_accuracy:
0.6250
Epoch 8/50
6/6 - 0s - loss: 0.9445 - accuracy: 0.5054 - val_loss: 0.8167 - val_accuracy:
0.6250
Epoch 9/50
6/6 - 0s - loss: 0.9027 - accuracy: 0.6559 - val_loss: 0.7100 - val_accuracy:
0.6667
Epoch 10/50
6/6 - 0s - loss: 0.8682 - accuracy: 0.6667 - val_loss: 0.6309 - val_accuracy:
0.6667
Epoch 11/50
6/6 - 0s - loss: 0.8792 - accuracy: 0.5914 - val_loss: 0.6143 - val_accuracy:
0.7500
Epoch 12/50
6/6 - 0s - loss: 0.8597 - accuracy: 0.7204 - val_loss: 0.5373 - val_accuracy:
0.8750
Epoch 13/50
6/6 - 0s - loss: 0.8359 - accuracy: 0.6022 - val_loss: 0.4813 - val_accuracy:
0.9167
Epoch 14/50
6/6 - 0s - loss: 0.7995 - accuracy: 0.6022 - val_loss: 0.5033 - val_accuracy:
0.8333
Epoch 15/50
6/6 - 0s - loss: 0.8139 - accuracy: 0.7957 - val_loss: 0.5074 - val_accuracy:
0.8333
Epoch 16/50
6/6 - 0s - loss: 0.7742 - accuracy: 0.6344 - val_loss: 0.4067 - val_accuracy:
0.8750
Epoch 17/50
6/6 - 0s - loss: 0.7700 - accuracy: 0.6882 - val_loss: 0.4973 - val_accuracy:
0.7917
Epoch 18/50
6/6 - 0s - loss: 0.7730 - accuracy: 0.7634 - val_loss: 0.4194 - val_accuracy:
0.8750
```

```
Epoch 19/50
6/6 - 0s - loss: 0.7365 - accuracy: 0.7097 - val_loss: 0.3736 - val_accuracy:
0.8750
Epoch 20/50
6/6 - 0s - loss: 0.7349 - accuracy: 0.7204 - val_loss: 0.4611 - val_accuracy:
0.8333
Epoch 21/50
6/6 - 0s - loss: 0.7321 - accuracy: 0.8065 - val_loss: 0.3946 - val_accuracy:
0.8750
Epoch 22/50
6/6 - 0s - loss: 0.7238 - accuracy: 0.7634 - val_loss: 0.3586 - val_accuracy:
0.8750
Epoch 23/50
6/6 - 0s - loss: 0.7098 - accuracy: 0.7527 - val_loss: 0.3759 - val_accuracy:
0.8750
Epoch 24/50
6/6 - 0s - loss: 0.7031 - accuracy: 0.7957 - val_loss: 0.3835 - val_accuracy:
0.8750
Epoch 25/50
6/6 - 0s - loss: 0.7073 - accuracy: 0.7957 - val_loss: 0.3633 - val_accuracy:
0.8750
Epoch 26/50
6/6 - 0s - loss: 0.7375 - accuracy: 0.6667 - val_loss: 0.3850 - val_accuracy:
0.8750
Epoch 27/50
6/6 - 0s - loss: 0.7201 - accuracy: 0.7849 - val_loss: 0.4132 - val_accuracy:
0.8333
Epoch 28/50
6/6 - 0s - loss: 0.6826 - accuracy: 0.7634 - val_loss: 0.3539 - val_accuracy:
0.8750
Epoch 29/50
6/6 - 0s - loss: 0.6696 - accuracy: 0.7849 - val_loss: 0.3905 - val_accuracy:
0.8750
Epoch 30/50
6/6 - 0s - loss: 0.7144 - accuracy: 0.6882 - val_loss: 0.3206 - val_accuracy:
0.8750
Epoch 31/50
6/6 - 0s - loss: 0.7073 - accuracy: 0.6989 - val_loss: 0.4227 - val_accuracy:
0.8333
Epoch 32/50
6/6 - 0s - loss: 0.6841 - accuracy: 0.7742 - val_loss: 0.3765 - val_accuracy:
0.8333
Epoch 33/50
6/6 - 0s - loss: 0.6543 - accuracy: 0.8172 - val_loss: 0.3126 - val_accuracy:
0.8750
Epoch 34/50
6/6 - 0s - loss: 0.6511 - accuracy: 0.7634 - val_loss: 0.4017 - val_accuracy:
0.8333
```

```
Epoch 35/50
6/6 - 0s - loss: 0.6312 - accuracy: 0.7849 - val_loss: 0.3474 - val_accuracy:
0.8750
Epoch 36/50
6/6 - 0s - loss: 0.6328 - accuracy: 0.8280 - val_loss: 0.3411 - val_accuracy:
0.8750
Epoch 37/50
6/6 - 0s - loss: 0.6271 - accuracy: 0.7957 - val_loss: 0.3506 - val_accuracy:
0.8750
Epoch 38/50
6/6 - 0s - loss: 0.6175 - accuracy: 0.7849 - val_loss: 0.4067 - val_accuracy:
0.8333
Epoch 39/50
6/6 - 0s - loss: 0.6350 - accuracy: 0.8065 - val_loss: 0.3307 - val_accuracy:
0.8750
Epoch 40/50
6/6 - 0s - loss: 0.6108 - accuracy: 0.7742 - val_loss: 0.3791 - val_accuracy:
0.8750
Epoch 41/50
6/6 - 0s - loss: 0.6061 - accuracy: 0.7957 - val_loss: 0.3622 - val_accuracy:
0.8750
Epoch 42/50
6/6 - 0s - loss: 0.6019 - accuracy: 0.7957 - val_loss: 0.3502 - val_accuracy:
0.8750
Epoch 43/50
6/6 - 0s - loss: 0.5957 - accuracy: 0.8280 - val_loss: 0.4065 - val_accuracy:
0.7917
Epoch 44/50
6/6 - 0s - loss: 0.6002 - accuracy: 0.7849 - val_loss: 0.3634 - val_accuracy:
0.8750
Epoch 45/50
6/6 - 0s - loss: 0.5953 - accuracy: 0.8065 - val_loss: 0.3927 - val_accuracy:
0.8750
Epoch 46/50
6/6 - 0s - loss: 0.5991 - accuracy: 0.8172 - val_loss: 0.3587 - val_accuracy:
0.8750
Epoch 47/50
6/6 - 0s - loss: 0.5841 - accuracy: 0.8065 - val_loss: 0.3441 - val_accuracy:
0.8750
Epoch 48/50
6/6 - 0s - loss: 0.6219 - accuracy: 0.7634 - val_loss: 0.3607 - val_accuracy:
0.8750
Epoch 49/50
6/6 - 0s - loss: 0.6468 - accuracy: 0.7527 - val_loss: 0.5156 - val_accuracy:
0.7917
Epoch 50/50
6/6 - 0s - loss: 0.6205 - accuracy: 0.7527 - val_loss: 0.3146 - val_accuracy:
0.8750
```

2.2 Evaluation of Training

The newly trained neural network can be evaluated for accuracy and loss using `model.evaluate`.

[ ]:
```
model.evaluate(data,target, batch_size=16, verbose = 3)
```

[ ]:
```
[0.5567188262939453, 0.8034188151359558]
```

The neural network can also now be used to predict the classification of any appropriate sets of input data.

[ ]:
```
import numpy as np
predict = model.predict(data)
classes = np.argmax(predict,axis=1)

classes = np.where(classes == 0,'low', classes)
classes = np.where(classes == '1','medium', classes)
classes = np.where(classes == '2','high', classes)

print(classes)
```

```
['low' 'low' 'low' 'low' 'low' 'low' 'low' 'low' 'medium' 'medium' 'high'
 'medium' 'high' 'high' 'medium' 'low' 'medium' 'high' 'high' 'medium'
 'high' 'medium' 'medium' 'high' 'medium' 'medium' 'medium' 'high' 'high'
 'high' 'high' 'high' 'medium' 'low' 'high' 'high' 'high' 'high' 'low'
 'high' 'low' 'high' 'high' 'high' 'low' 'high' 'high' 'high' 'low' 'high'
 'medium' 'low' 'high' 'low' 'high' 'high' 'low' 'high' 'high' 'high'
 'low' 'low' 'low' 'low' 'high' 'low' 'high' 'high' 'high' 'low' 'high'
 'high' 'low' 'low' 'high' 'high' 'low' 'high' 'high' 'low' 'high' 'high'
 'high' 'high' 'high' 'low' 'high' 'low' 'high' 'low' 'high' 'low' 'low'
 'high' 'high' 'high' 'low' 'high' 'low' 'low' 'low' 'low' 'low' 'high'
 'low' 'high' 'low' 'high' 'high' 'high' 'high' 'low' 'low' 'low' 'high'
 'high' 'high']
```

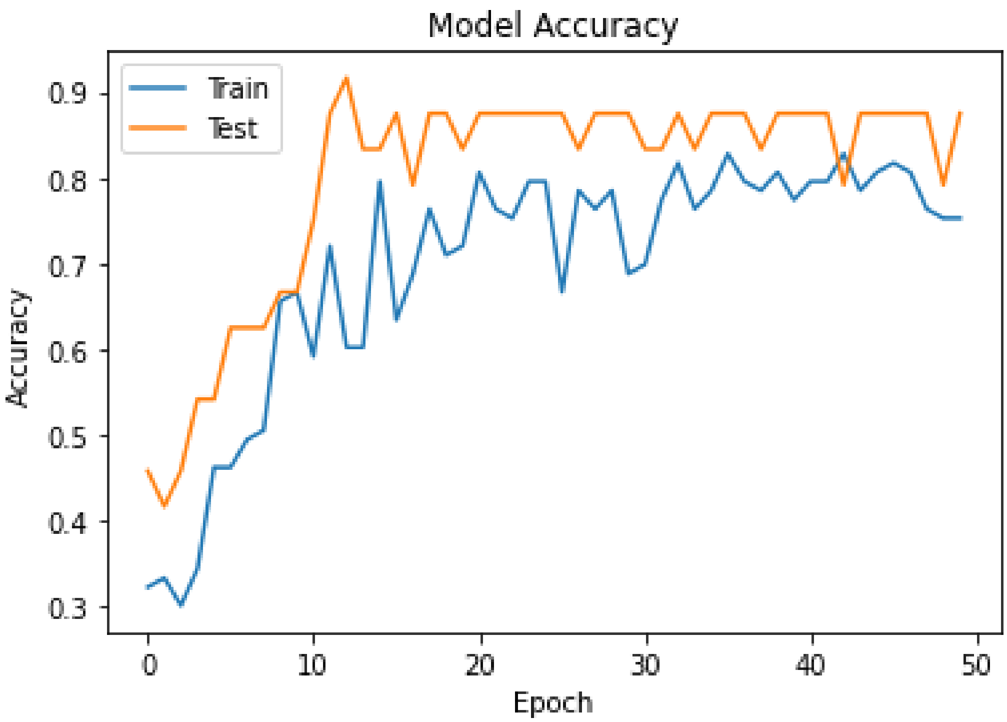
Graphs can be plotted in order to make the accuracy and loss easier to visualise. The first graph shows 'Acuraccy' against 'Time' for the the neural network model. The second graph shows 'Loss' against 'Time' for the neural network model. Both graphs compare the 'Training' results with the 'Test' resluts.

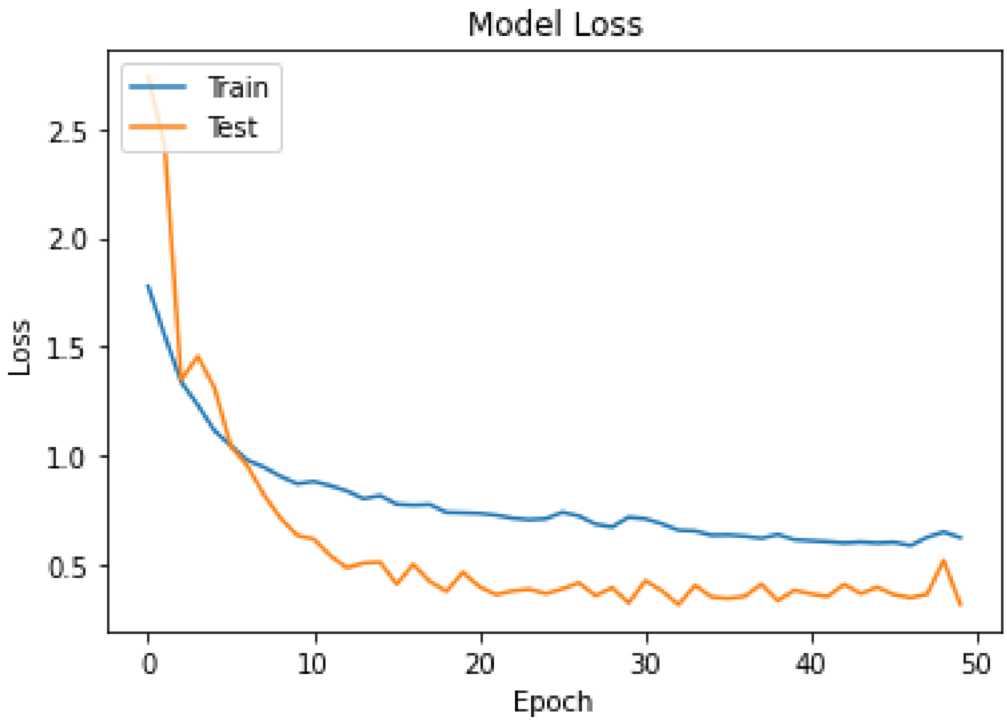
[ ]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Test'],loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
```

```
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Test'],loc='upper left')
plt.show()
```




Model Loss
Train
Test
2.5
2.0
1.5
1.0
0.5
Loss
0
10
20
30
40
50
Epoch