Exercises - Week 6

Exercise 1

In last weeks exercise 6, you were asked to „Use a multilayer perceptron to classify petal length and width for the Iris data set".

The techniques you used last week can obviously rather easily be used on other datasets as well. And that is exact where we will begin this weeks exercises…

Starting from the Moons dataset:

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons;
from pandas import DataFrame
# generate 2d classification dataset
X, y = make_moons(n_samples=1000, noise=0.1)
# scatter plot, dots colored by class value
df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
colors = {0:'red', 1:'blue'}
fig, ax = plt.subplots()
grouped = df.groupby('label')
for key, group in grouped:
    group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])

plt.show()
```

You are here asked to construct a neural net, similarly to the one you used last week – and train it. So that it can be used to predict data points, according to the Moons dataset.

a) What is the minimum number of hiddens layers needed?

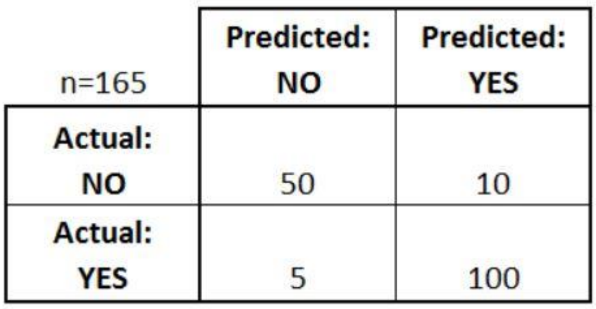
b) What is the minimum number of neurons per layer needed?

c) Try out other values for the (make_moons) samplesize & noise. And try out other ratios for testsize.

    And then redo a) and b)

Exercise 2 – Confusion matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.



n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

How often (in percent) is this model correct?

True Positive Rate: When it's actually yes, how often does it predict yes?

False Positive Rate: When it's actually no, how often does it predict yes?

Exercise 3 - prediction accuracy

Let us say you have a binary predictor where the TN = 12, TP = 23, FP = 5, FN = 10.
What percentage of the total predictions did this predictor classify correctly?
You will need to understand this in order to calculate the prediction accuracy
for the mini-titanic exercise in order to understand what is going on the following exercise.
Don’t rush it, but take your time to fully understand what is going on here (HINT: The percentage should be a nice round number…..!)

Exercise 4 – Recall and Confusion matrix, continued.

Lets imagine that we have a zip code reader – and we want to identify a number as either a 5 or not a 5.

We make ***a very simple algorithm that just always predicts 5*** – no matter what the number is (0,1,2..9).

Be very careful…. (Insert numbers in confusion matrix)….

- What is recall …is that a good value…?!
- What is precision… is that a good value?

Exercise 5 - train on new data - titanic limited dataset - real data

In groups of 2 or 3.

Note: For this exercise we will have a small „competition“ (compete with yourself, and other groups.. …☺) - it’s simple – try to get the highest possible prediction accuracy (as calculated the same way as in exercise 2).

So, be systematic in your experiments today. And keep track of the hyperparameter settings that gives you the best result!

In this exercise you will get some datasets that are a limited version of the „Titanic Dataset“ from ship of Titanic. You will try to build a network using scikit learn and then train on this data and then predict on the test data and see how good you can make the prediction. You should be able to achieve better accuracy than just a random prediction!

On Canvas you can find a .csv file with data from the Titanic shipwreck. Download this file and open it (You might want to use openoffice for working with .csv files, which many find to be the best, but you can also just use a texteditor, although openoffice formats it very nicely to view).

As you can see there are 500 data rows with passenger information.
Each passenger has an id number, a number (0 = died, 1 = survived) to indicate if the passenger survived or died. Then there is a column named Pclass which indicates the class ticket the passenger had (1 = first class, 2 = second class, 3 = third class), finally there is an age field indicating the age of the person (Note: here there are some missing data).

Roughly 67% of the passengers on Titanic died. In this specific data set (in the test set - the last 100 columns) we have 64 % who died.

So if you made a predictor that simply always output “died”, then this predictor would in fact be right in 64 % of the cases you tried it on in this data set.
Of course we want to be better than that if possible.

You should divide the dataset into a training set and a test set in the ratio of 80:20, so the first 400 rows will be for training and the last 100 rows will be for evaluating how good the model is. For all rows in both the training and the test set we do know the true value, but obviously when evaluating the model on the last 100 rows you can only use the passenger class and the age as features to predict whether the person survived or not and then compare your predicted value with the true value from the dataset.
Then calculate how many of the 100 predictions on the test set you got correct by comparing to the true value. You should try to aim to get more than 64% correct!
Obviously many ML methods can be used on this problem and it’s difficult to say beforehand which method would be best, but in this exercise you will try to use a **neural network** with scikit learn to train on the dataset (but of course using a NN on a problem with only 2 features might be a bit overkill, but it is for learning to use a neural network).

A good way to handle data in python from .csv files is by using Pandas.

Here are some material you should look at to get started using Pandas if you did not already do it:
https://www.youtube.com/watch?v=XDAnFZqJDvI&list=WL&index=11
A very good introductory video.

There is also some interactive material here:
https://www.learnpython.org/en/Pandas_Basics
Below you can also see some code to get you started - but you will (probably) still need to look at some tutorials on Pandas to understand what is going on.

```
#import panda library and a few others we will need.
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import
classification_report,confusion_matrix
# skipping the header
data =pd.read_csv( ' titanic_train_500_age_passengerclass.csv ' , sep = ',' ,
header = 0 )

# Make sure to place the csv file correctly …

# show the data
print ( data .describe( include = 'all' ))
#the describe is a great way to get an overview of the data
print ( data .values)
yvalues = pd.DataFrame( dict ( Survived =[]), dtype = int )
yvalues[ "Survived" ] = data [ "Survived" ].copy()
#now the yvalues should contain just the survived column
```

A good idea is also to display the data graphically, so you get an idea of how the age and PClass looks like:

```
x = data[ "Age" ]
y = data[ "Pclass" ]
plt.figure()
plt.scatter(x.values, y.values, color = 'black' , s = 20 )
plt.show()
```

This will show you the distribution of the age and the pclass values.

Notice, that a python program will stop and wait for you to press enter when you have plt.show() to proceed with executing the rest of the program, so it can sometimes make sense to place this command at the very end of your python script.
A good idea is also to plot the age against the survived value (so you change the y value) and then also to do a plot with the Pclass against the survived value.
Then you can see if there are some relations between these parameters also.
So in total you will have 3 plots each plotting a 2D plot between these 3 parameters.
Now we can clean the data a bit also:

```
#now we can delete the survived column from the data (because
we have copied that already into the yvalues.
data.drop( 'Survived' , axis = 1 , inplace = True )
```

In the data there is also a column named ‘PassengerId’ as you can see. Now that column is of course not in the yvalues (either train or test).
But should you keep this column or delete it for the x data? (Does it have any impact on whether a person survives or not?)
Notice that the Titanic passengers have a mean age of 29.

Of course you need to split the data into two sets : xtrain (with the first 400 values)

and xtest (with the last 100 values) and the split for the yvalues into two sets: ytrain and ytest:
You can split pandas easily with the head and tail commands:

```
xtrain = data.head( 400 )
xtest = data.tail( 100 )
```

Do the same for the yvalues to setup two arrays - ytrain and ytest arrays - print them out to make sure it looks what you expected - they should just hold a single column and the ytrain should have 400 entries while the ytest should have 100 entries in the array.

You also need to consider how you will handle the missing age data for some of the rows? There are several different strategies you can choose - for instance setting all missing data to 0 (there is a convenient python command for that you can use on any Panda columns: fillna (0.0) - you can look up fillna and pandas to see more) , setting all missing data to the average of all the other ages, or simply deleting the rows with the missing data (of course this will give you less training samples to train the network) - choose one method and then you can try to change later to see if it improves the prediction accuracy?

You can then train on the xtrain,ytrain data and afterwards test the performance of your model on the xtest,ytest data set.
But first we need to scale our data - there is a normal requirement for working with neural networks. Our yvalues are either 0 or 1, but for instance the age has a pretty big span, so usually you want to scale all input values into [0;1].
This can be easily done in python:

```
scaler = StandardScaler()
scaler .fit(xtrain)
xtrain = scaler .transform(xtrain)
xtest= scaler .transform(xtest)
```

Note that we ONLY scale the xtrain and xtest arrays, NOT the y arrays.

After building your network (and you have to think about the number of hidden layers and how many neurons do you want in those layers? Also what should the batch size be? How many epochs/iterations do you want to train?). Think about these questions as your proceed with the exercise:

Check the documentation to see what the default values are for the Neural Network:
http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
Also to get deterministic results I will recommend that you set the random state to some integer (I just choose 0).
Here you can see an example for defining with 1000 epochs for a neural network with 2 hidden layers, where each hidden layer has 8 neurons:

```
mlp =
MLPClassifier( hidden_layer_sizes =( 8 , 8 ), max_iter = 1000 ,
random_state = 0 )
```

Scikit-learn will automatically add the correct number of input neurons and the

correct number of neurons in the output layer based on the xtrain and ytrain data.
You will then need to train the model to the data:

```
mlp.fit(xtrain,ytrain.values.ravel())
#the reason for the values.ravel is that these data has not
#been scaled and they need to be converted to the correct
#input format for the mlp.fit. Data that is scaled already has
#this done to them.
```

You can now try to predict the values for the TEST set and then evaluate the performance of the predictions on the test set against the correct y values for the test set.

```
#to predict on our xtest set
predictions = mlp.predict(xtest)
```

Now is the time to evaluate how good our predictor is:

```
matrix = confusion_matrix(ytest,predictions)
print (matrix)
print (classification_report(ytest,predictions))
```

This will give you two reports - the first is the [TP,TN,FP,FN] matrix which is very useful for seeing exactly how many of the 100 passengers were correctly classified as dead or survived.

You can extract the values from the matrix like this:

```
tn, fp, fn, tp = matrix.ravel()
```

You can then print the values out with the print command in python.

What accuracy (percentage of correct predictions out of the 100 predictions) can you achieve? (use the TP,TN,FP,FN to calculate this (get python to calculate it of course.. - see exercise 2-3 and make that into a general formula you can use))

Experiments below - be systematic and record your result of each experiment with both the value of all important parameters used and the achieved accuracy.

Now it is time for some experiments (remember to record, write down, your results - each time. Be systematic in your experiments):

Try (at least) the following (in a systematic way, record your findings):

a) Try to experiment with the network topology (i.e. number of nodes and layers) to see if you can achieve a higher accuracy?

b) Try to experiment with the way you handle the missing input age data to see how this reflect the accuracy?

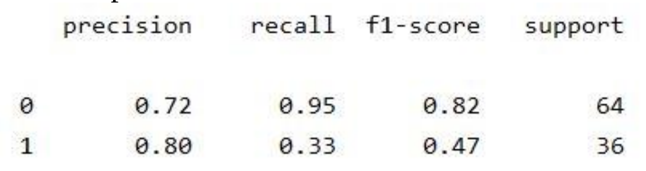
c) Can we do without the scaler?

d) What about the batch size? Experiment with that also and see if it has an effect on the accuracy.

e) What about the number of epochs? Does that have an effect on accuracy?

f) What about changing the activation function? (again see the MLP scikit documentation for that - link earlier)

First attempt:



```
      precision    recall  f1-score   support

0          0.72      0.95      0.82        64
1          0.80      0.33      0.47        36
```

Can you reproduce, or do better?

Hints.
# You will probably need to use the standard scaler…
scaler = StandardScaler()
etc…

Use the MLPClassifier… with appropriate number of hidden layers and nodes..
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes =(2 , 2), max_iter = 1000 , random_state = 1)

When training… well, use this trick:
mlp.fit(xtrain, ytrain.values.ravel())
#the reason for the values.ravel is that these data has not
#been scaled and they need to be converted to the correct
#input format for the mlp.fit. Data that is scaled already has
#this done to them.

Make your predictions…
#predictions
predictions = mlp.predict(xtest)

And show you results..
matrix = confusion_matrix(ytest,predictions)
print (matrix)
print (classification_report(ytest,predictions))

Exercise 6 (a)

See the file Exercise_DataSetSpine.pdf.
And improve the Mlpclassifier so that you get a better accuracy.

Exercise 6 (b) – Work on you own dataset.

In groups of 2 or 3.

See the Exercise ‘‘Create your own dataset‘‘, on Canvas.