**kNN**

1. **Using the iris data set implement the KNN algorithm. Take different values for Test and training data set. Also use different values for k. Also find the accuracy level.**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv("iris.csv")

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, 4].values

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20)

from sklearn.neighbors import KNeighborsClassifier

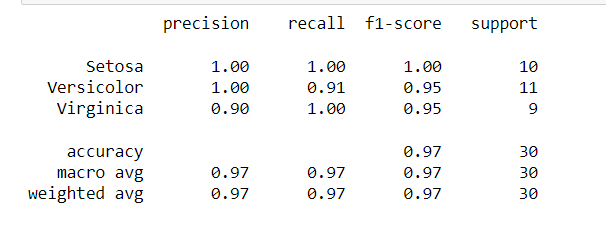
classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import classification\_report, confusion\_matrix

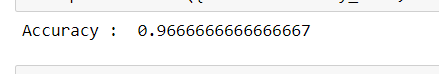
print(classification\_report(y\_test, y\_pred))



from sklearn.metrics import accuracy\_score

print ("Accuracy : ", accuracy\_score(y\_test, y\_pred))

df = pd.DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})



Reference: <https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-learn/>

1. **Download another data set suitable for the KNN and implement the KNN algorithm. Take different values for Test and training data set. Also use different values for k.**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv("cancer.csv")

dataset.head()

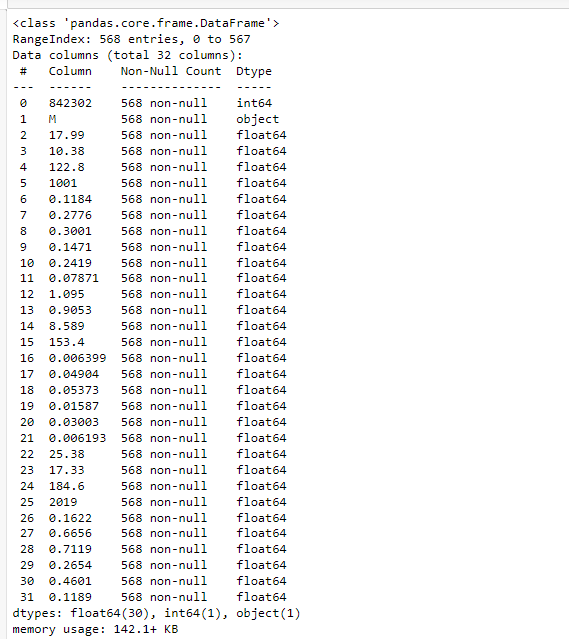
dataset.info()

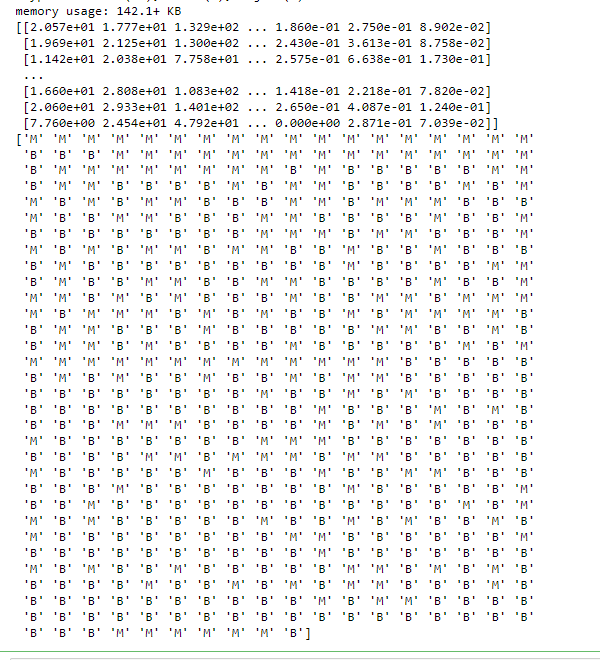
X = dataset.iloc[:, 2:35].values

print(X)

y = dataset.iloc[:, 1].values

print(y)





from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20)

from sklearn.neighbors import KNeighborsClassifier

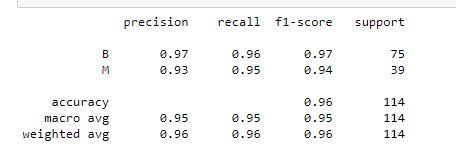
classifier = KNeighborsClassifier(n\_neighbors=5)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import classification\_report, confusion\_matrix

print(classification\_report(y\_test, y\_pred))



from sklearn.metrics import accuracy\_score

print ("Accuracy : ", accuracy\_score(y\_test, y\_pred))

df = pd.DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})



**Naive Bayes**

1. **Using iris data set, implement naive bayes classification for different naive Bayes classification algorithms. ((i) gaussian (ii) bernoulli etc)**

* **Find out the accuracy level w.r.t to each algorithm**
* **Display the no:of mislabeled classification from test data set**
* **List out the class labels of the mismatching records**

import numpy as np

import matplotlib.pyplot as plt

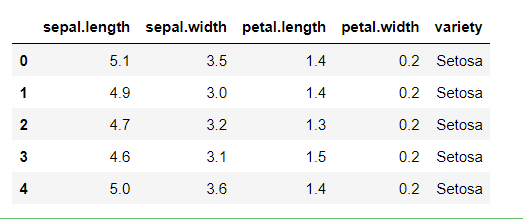
import pandas as pd

dataset = pd.read\_csv('iris.csv')

X = dataset.iloc[:,:4].values

y = dataset['variety'].values

dataset.head(5)



from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2)

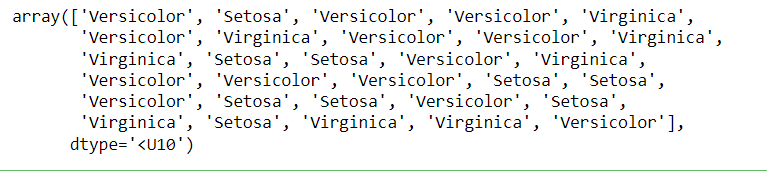
from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

y\_pred



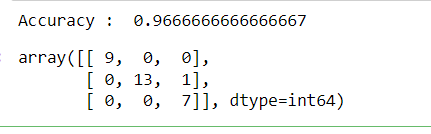
from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

from sklearn.metrics import accuracy\_score

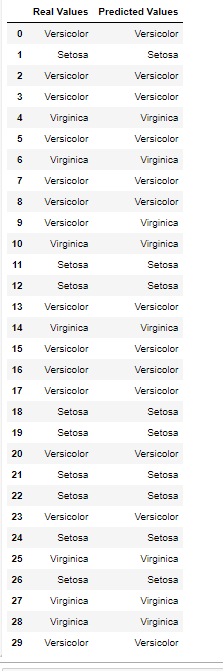
print ("Accuracy : ", accuracy\_score(y\_test, y\_pred))

cm



df = pd.DataFrame({'Real Values':y\_test, 'Predicted Values':y\_pred})

df



References:

* <https://towardsdatascience.com/machine-learning-basics-naive-bayes-classification-964af6f2a965>
* <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive_bayes>

**Decision Tree**

1. **Use car details CSV file and implement decision tree algorithm**

* **Find out the accuracy level.**
* **Display the no: of mislabelled classification from test data set**
* **List out the class labels of the mismatching records**

import os

import numpy as np

import pandas as pd

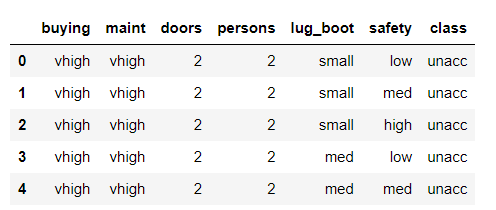
import numpy as np, pandas as pd

import matplotlib.pyplot as plt

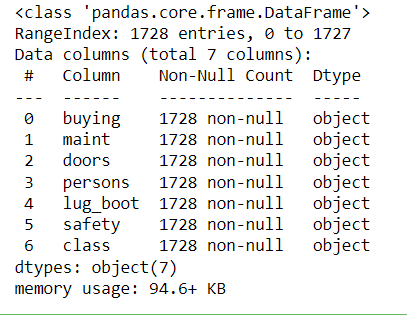
from sklearn import tree, metrics, model\_selection

data = pd.read\_csv('car.csv',names=['buying','maint','doors','persons','lug\_boot','safety','class'])

data.head()



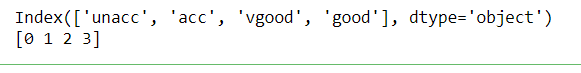
data.info()



data['class'],class\_names = pd.factorize(data['class'])

print(class\_names)

print(data['class'].unique())



data['buying'],\_ = pd.factorize(data['buying'])

data['maint'],\_ = pd.factorize(data['maint'])

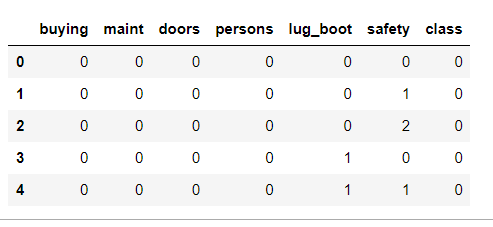
data['doors'],\_ = pd.factorize(data['doors'])

data['persons'],\_ = pd.factorize(data['persons'])

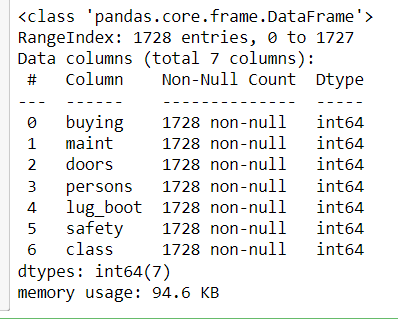
data['lug\_boot'],\_ = pd.factorize(data['lug\_boot'])

data['safety'],\_ = pd.factorize(data['safety'])

data.head()



data.info()



X = data.iloc[:,:-1]

y = data.iloc[:,-1]

# split data randomly into 70% training and 30% test

X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, y, test\_size=0.3, random\_state=0)

# train the decision tree

dtree = tree.DecisionTreeClassifier(criterion='entropy', max\_depth=3, random\_state=0)

dtree.fit(X\_train, y\_train)



# use the model to make predictions with the test data

y\_pred = dtree.predict(X\_test)

# how did our model perform?

accuracy = metrics.accuracy\_score(y\_test, y\_pred)

print('Accuracy: {:.2f}'.format(accuracy))



count\_misclassified = (y\_test != y\_pred).sum()

print('Misclassified samples: {}'.format(count\_misclassified))



**Single Linear Regression**

import numpy as np

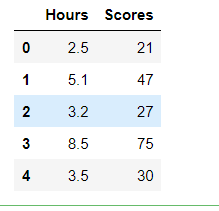
import pandas as pd

import matplotlib.pyplot as plt

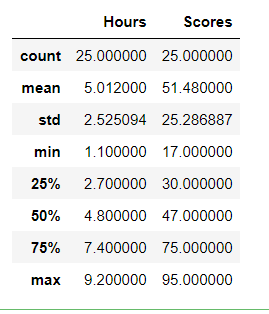
#data set contains details of no.of hours spend by students for studt and their marks

student = pd.read\_csv('student\_scores.csv')

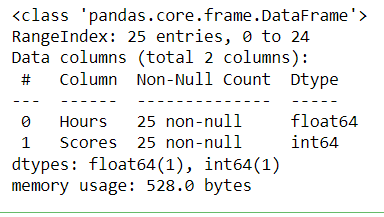
student.head()



student.describe()



student.info()



import matplotlib.pyplot as plt

Xax=student.iloc[:,0]

Yax=student.iloc[:,1]

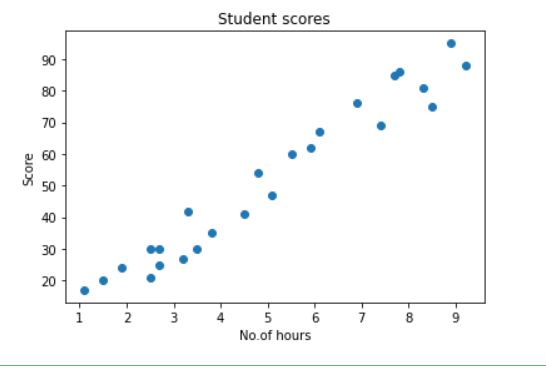
plt.scatter(Xax,Yax)

plt.xlabel("No.of hours")

plt.ylabel("Score")

plt.title("Student scores")

plt.show()



#Perform the simple linear regression model

#Equation: Y=w0+w1.x

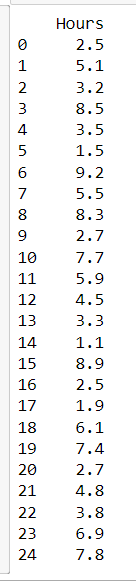
#Here Y(marks)=w0+w1.x

#Create x as hours and Y as marks

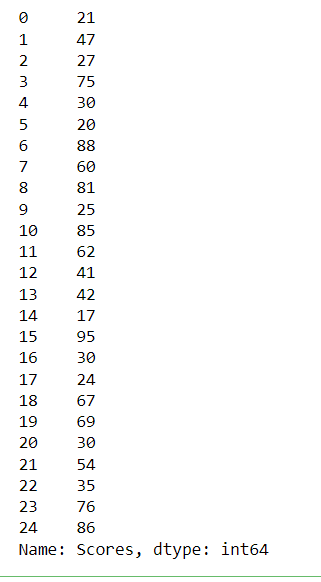
X = student.iloc[:, :-1]

y = student.iloc[:, 1]

print(X)



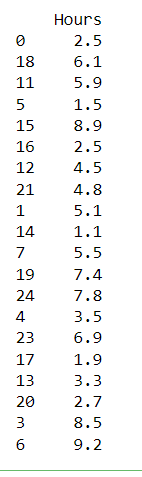
print(y)



from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

print(X\_train)



from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)



print(regressor.intercept\_)



print(regressor.coef\_)



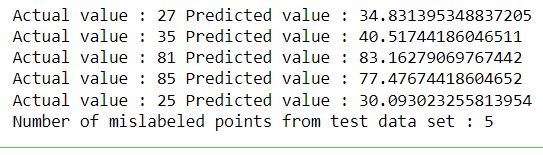
y\_pred = regressor.predict(X\_test)

for(i,j) in zip(y\_test,y\_pred):

if i!=j:

print("Actual value :",i,"Predicted value :",j)

print("Number of mislabeled points from test data set :", (y\_test != y\_pred).sum())

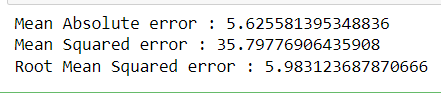


from sklearn import metrics

print("Mean Absolute error :", metrics.mean\_absolute\_error(y\_test,y\_pred))

print("Mean Squared error :", metrics.mean\_squared\_error(y\_test,y\_pred))

print("Root Mean Squared error :", np.sqrt(metrics.mean\_squared\_error(y\_test,y\_pred)))



import matplotlib.pyplot as plt

c=X\_test['Hours'].count()

xax=np.arange(c)

print(xax)

X\_axis = np.arange(len(xax))

plt.bar(X\_axis-0.2, y\_test, 0.6, label='Actual')

plt.bar(X\_axis+0.2, y\_pred, 0.6, label='Predicted')

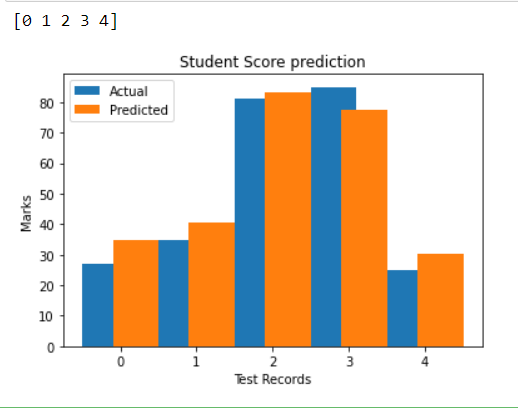
plt.xlabel("Test Records")

plt.ylabel("Marks")

plt.title("Student Score prediction")

plt.legend()

plt.show()



**Multiple Linear Regression**

Abcd

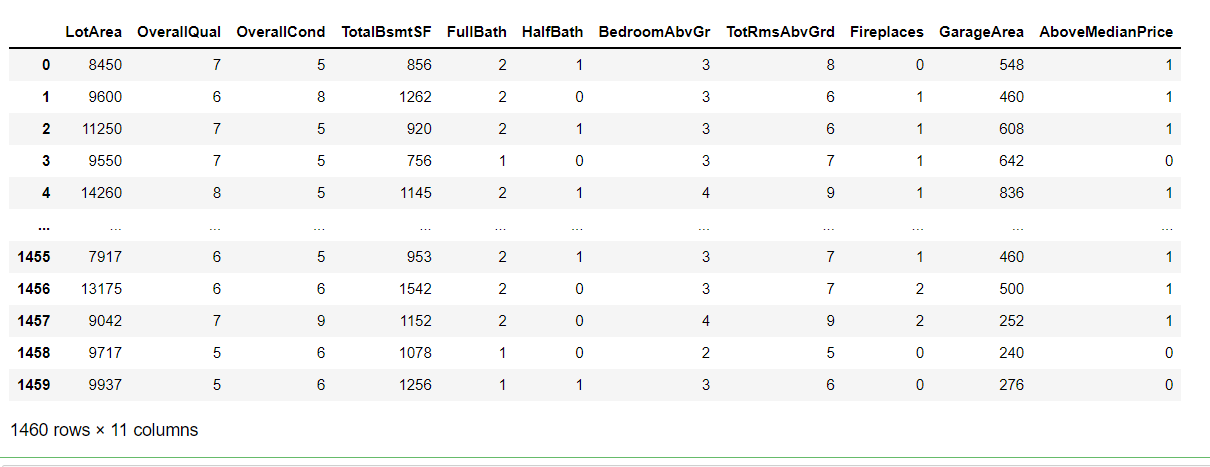
**Neural Networks**

1. **Create a neural network for the given ‘houseprice.csv’ to predict the whether price of the house is above or below median value or not**

import pandas as pd

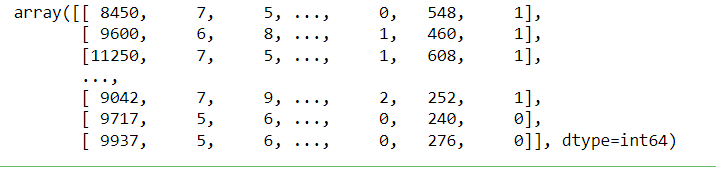
df = pd.read\_csv('housepricedata.csv')

df



dataset = df.values

dataset



X = dataset[:,0:10]

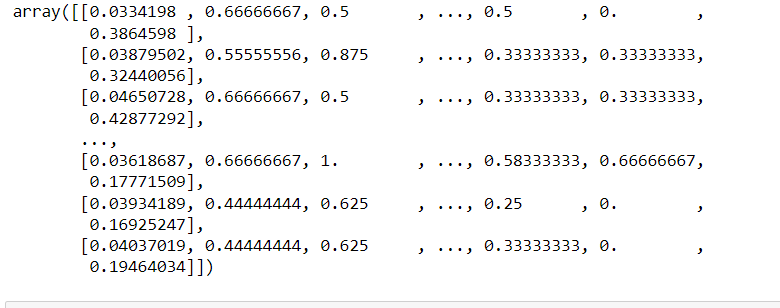
Y = dataset[:,10]

from sklearn import preprocessing

min\_max\_scaler = preprocessing.MinMaxScaler()

X\_scale = min\_max\_scaler.fit\_transform(X)

X\_scale



from sklearn.model\_selection import train\_test\_split

X\_train, X\_val\_and\_test, Y\_train, Y\_val\_and\_test = train\_test\_split(X\_scale, Y, test\_size=0.3)

X\_val, X\_test, Y\_val, Y\_test = train\_test\_split(X\_val\_and\_test, Y\_val\_and\_test, test\_size=0.5)

print(X\_train.shape, X\_val.shape, X\_test.shape, Y\_train.shape, Y\_val.shape, Y\_test.shape)



from keras.models import Sequential

from keras.layers import Dense

model = Sequential([

Dense(32, activation='relu', input\_shape=(10,)),

Dense(32, activation='relu'),

Dense(1, activation='sigmoid'),

])

model.compile(optimizer='sgd',

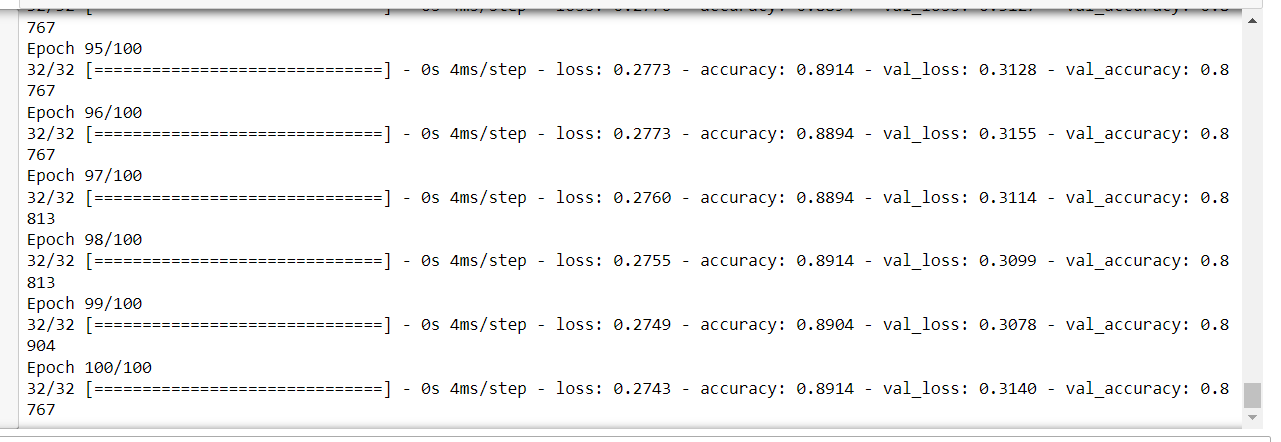
loss='binary\_crossentropy',

metrics=['accuracy'])

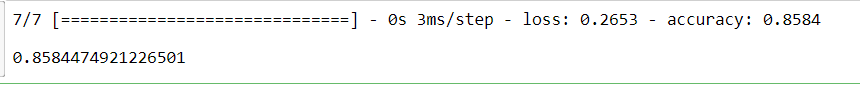
hist = model.fit(X\_train, Y\_train,

batch\_size=32, epochs=100,

validation\_data=(X\_val, Y\_val))



model.evaluate(X\_test, Y\_test)[1]



import matplotlib.pyplot as plt

plt.plot(hist.history['loss'])

plt.plot(hist.history['val\_loss'])

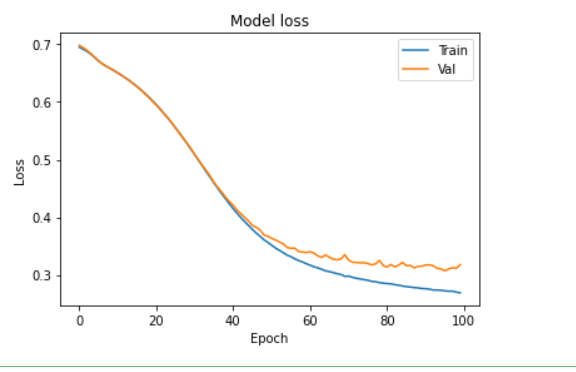
plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper right')

plt.show()



plt.plot(hist.history['accuracy'])

plt.plot(hist.history['val\_accuracy'])

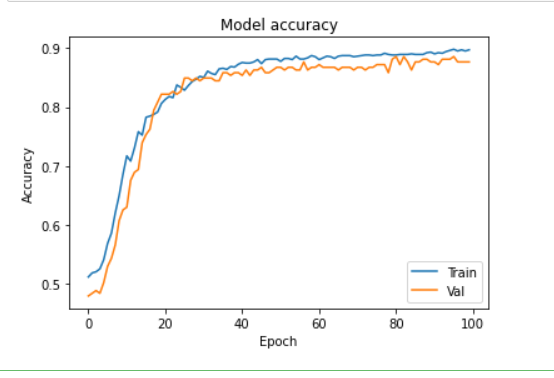
plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='lower right')

plt.show()



model\_2 = Sequential([

Dense(1000, activation='relu', input\_shape=(10,)),

Dense(1000, activation='relu'),

Dense(1000, activation='relu'),

Dense(1000, activation='relu'),

Dense(1, activation='sigmoid'),

])

model\_2.compile(optimizer='adam',

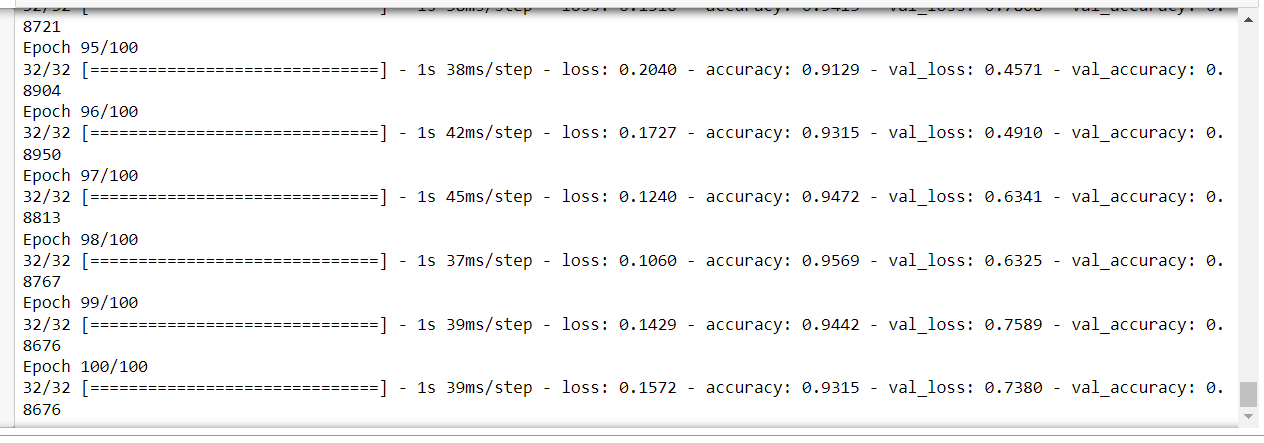
loss='binary\_crossentropy',

metrics=['accuracy'])

hist\_2 = model\_2.fit(X\_train, Y\_train,

batch\_size=32, epochs=100,

validation\_data=(X\_val, Y\_val))



plt.plot(hist\_2.history['loss'])

plt.plot(hist\_2.history['val\_loss'])

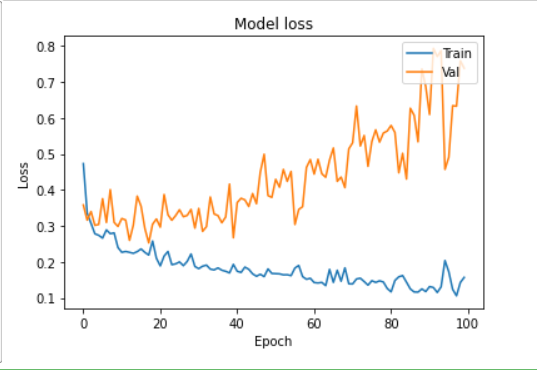
plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper right')

plt.show()



plt.plot(hist\_2.history['accuracy'])

plt.plot(hist\_2.history['val\_accuracy'])

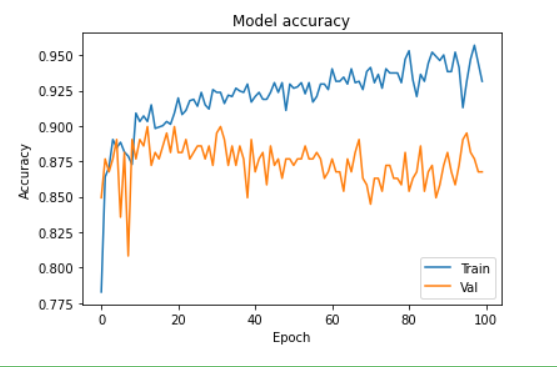
plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='lower right')

plt.show()



from keras.layers import Dropout

from keras import regularizers

model\_3 = Sequential([

Dense(1000, activation='relu', kernel\_regularizer=regularizers.l2(0.01), input\_shape=(10,)),

Dropout(0.3),

Dense(1000, activation='relu', kernel\_regularizer=regularizers.l2(0.01)),

Dropout(0.3),

Dense(1000, activation='relu', kernel\_regularizer=regularizers.l2(0.01)),

Dropout(0.3),

Dense(1000, activation='relu', kernel\_regularizer=regularizers.l2(0.01)),

Dropout(0.3),

Dense(1, activation='sigmoid', kernel\_regularizer=regularizers.l2(0.01)),

])

model\_3.compile(optimizer='adam',

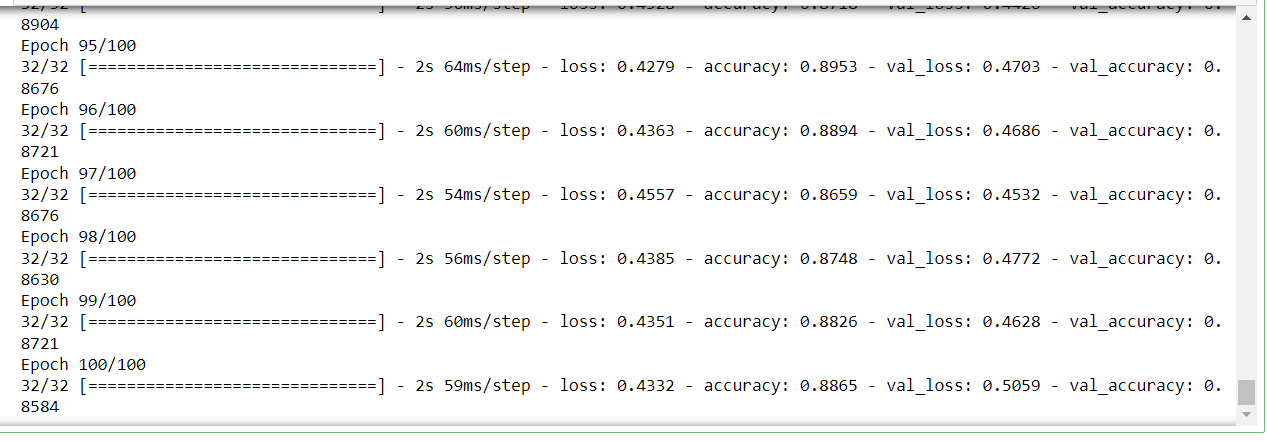
loss='binary\_crossentropy',

metrics=['accuracy'])

hist\_3 = model\_3.fit(X\_train, Y\_train,

batch\_size=32, epochs=100,

validation\_data=(X\_val, Y\_val))



plt.plot(hist\_3.history['loss'])

plt.plot(hist\_3.history['val\_loss'])

plt.title('Model loss')

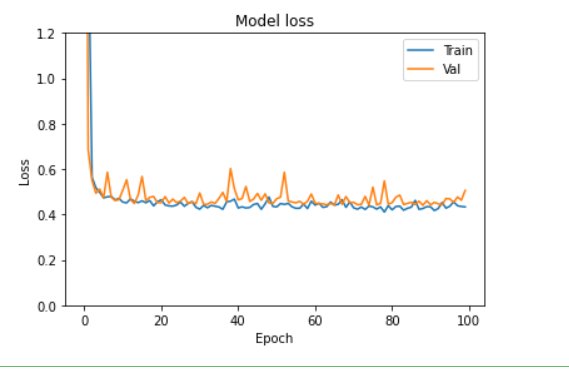
plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='upper right')

plt.ylim(top=1.2, bottom=0)

plt.show()



plt.plot(hist\_3.history['accuracy'])

plt.plot(hist\_3.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Val'], loc='lower right')

plt.show()

