# Prepare Notebook for Regression

* In the collateral directory from the github project copy the DowHistory.csv to the notebooks folder
* In a browser open the Juypiter app, this is likely at <http://localhost:8888/>
* Enter the notebook directory
* Click new “Python 3” to create a new notebook and save it as “RegressionModel”
* Add the following imports

#imports

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

from tensorflow import keras

# Read in the Data

* Add the following code. This load in data from csv file and loads it into two columns, each with a type of float32

#Read in Data

filename = "DowHistory.csv"

csvData = tf.data.experimental.CsvDataset(

filename,

[tf.float32,tf.float32],

header=True

)

# Split the data

* We now need to split the data into two sets, one for training our model and the other for testing its accuracy
* The is\_test function returns 1 out 5 records as testing data, is\_train says the rest are for testing

Notice that the information we are supplying to the training, the X variable contains the year as well as an integer that is the year – 1915 (the first year for our stock market data). Even though this is derived data, it gives our regression enough information to figure out the quadradic nature of the line.

#Split the data

def is\_test(x, y):

return x % 5 == 0

def is\_train(x, y):

return not is\_test(x, y)

def convert\_to\_Array(data, colMapping):

returnValue = []

for item in (data.enumerate().map(colMapping)).\_\_iter\_\_():

returnValue.append(item.numpy())

return tf.constant(returnValue, dtype=tf.float32)

def convert\_to\_2Array(data):

returnValue = []

for item1, item2 in (data.enumerate()).\_\_iter\_\_():

returnValue.append([item2[0].numpy(), item2[0].numpy() - 1915])

return tf.constant(returnValue, dtype=tf.float32)

recover = lambda x,y: y

test\_dataset = csvData.enumerate().filter(is\_test).map(recover)

train\_dataset = csvData.enumerate().filter(is\_train).map(recover)

train\_x = convert\_to\_2Array(train\_dataset)

train\_y = convert\_to\_Array(train\_dataset, lambda x, y: y[1])

test\_x = convert\_to\_2Array(test\_dataset)

test\_y = convert\_to\_Array(test\_dataset, lambda x, y: y[1])

# Train the model

* Next we are going to use the Keras neural network library.
* Our training will go through three layers
* “Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.”

We need to specify the number of dimensions for the first layer, it needs to be 2, the same as our train\_x tensor. No other layers need to specify an input size, The first and second layer output 100 dimensions. The third and final layer outputs a single result.

#Train the model

model = keras.Sequential()

model.add(keras.layers.Dense(100, activation='relu', input\_dim=2))

model.add(keras.layers.Dense(100, activation='relu'))

model.add(keras.layers.Dense(1, activation='linear'))

model.compile(loss='mean\_squared\_error', optimizer='adam')

model.fit(train\_x, train\_y,

epochs=5000,

batch\_size=100,

validation\_data=(test\_x,test\_y)

Note that we supply the testing data when calling fit, which trains our model. This data is used to calculate the loss at the end of each iteration (epoch). We have set up our model training to go through 5000 iterations.

# View the Results

* Earlier we imported matplotlib.pyplot. This can be used to visually show us how accurate our trained model is by comparing the actual validation results with the results predicted by our model.
* Additionally we can try a new point and see what that looks like.

#view the results

te\_years, te\_avgs = tf.split(test\_x, num\_or\_size\_splits=2, axis=1)

plt.scatter(te\_years, test\_y, label="actual")

plt.scatter(te\_years, model.predict(test\_x), label="test")

plt.scatter([2022], model.predict([[2022., 2022. - 1915]]), label="future")

plt.legend()

The results of the run of this program should be:

A screenshot of a cell phone

Description automatically generated

# Save the model

* Add and run the following command. Make sure all the commands we already did were run first. This saves our model to a directory we can then use in a service.

#Save the model

tf.saved\_model.save(model, "dow\_regression/1/")

Note, the /1/ in the path means this is version one. We can use this scheme to denote what is the newest model and hosting services like the TensorFlow Serving package will understand it.

* Save your notebook so you can modify the model later

This was obviously more difficult than the Azure method. What are some of the advantages using a tool like TensorFlow? What are some of the disadvantages?