Project 2 Data Programming with Python (Pytorch and Plotly)

Aniket Shah

Student Id: 18200042

- **PyTorch** - is an open source machine learning library for Python, based on Torch, used for applications such as natural language processing. It is primarily developed by Facebook's artificial-intelligence research group, and Uber's "Pyro" software for probabilistic programming is built on it. PyTorch provides few high-level features:
  - Tensor computation (like NumPy) with strong GPU acceleration
  - Deep Neural Networks built on a tape-based autodiff system
  - in PyTorch, you can define/manipulate your graph on-the-go. This is particularly helpful while using variable length inputs in RNNs.
  - It is more *pythonic* and building ML models feel more intuitive.
  - **PyTorch** is an open-source machine learning library for Python, based on Torch, used for applications such as natural language processing. It is primarily developed by Facebook's artificial-intelligence research group, and Uber's "Pyro" software for probabilistic programming is built on it.
    PyTorch provides two high-level features:
  - Tensor computation (like NumPy) with strong GPU acceleration
  - Deep Neural Networks built on a tape-based autodiff system.

- **Plotly** - also known by its URL, Plot.ly, is a technical computing company headquartered in Montreal, Quebec, that develops online data analytics and visualization tools. Plotly provides online graphing, analytics, and statistics tools for individuals and collaboration, as well as scientific graphing libraries for Python, R, MATLAB, Perl, Julia, Arduino, and RESTBokeh allows you to build complex statistical plots quickly and through simple commands.
  - Plotly was built using Python and the Django framework, with a front end using JavaScript and the visualization library D3.js, HTML and CSS. Files are hosted on Amazon S3.
  - Plot.ly is differentiated by being an online tool for doing analytics and visualization.
  - It has a robust API and includes one for python.
  - Browsing the website, one can see that there are lots of very rich, interactive graphs. You'll need to follow the docs to get your API key set up. Once we do, it all seems to work pretty seamlessly.

- The one caveat is that everything you are doing is posted on the web so make sure you are ok with it. There is an option to keep plots private so you do have control over that aspect.

- Plotly integrates pretty seamlessly with pandas.

PROBLEM STATEMENT (CLIENT REQUIREMENTS):

Using the Pytorch for deep learning and Plotly for visualization on the Google Stock Dataset which contains the stocks data of year 2011-2017 daily data, provided by the Google for data prediction. We have to suggest the client, about the stock price that will trend in future on daily basis.

Why am I using Google Stock Price Dataset:

1. This is a good project because it is so well understood.
2. Attributes are numeric and categorical so you have to figure out how to load and handle data.
3. It is a Classification problem, allowing you to practice with perhaps an easier type of supervised learning algorithm using deep learning library Pytorch
4. Using Plotly for visualization .
5. Creative feature engineering

Data preprocessing refers to the transformations applied to our data before feeding it to the algorithm. Data Preprocessing is a technique that is used to convert the raw data into a clean data set

Each row represents a customer, each column contains that customer's attributes:
Open: Opening rate of the share
High: Highest price reached by the data
Low: Low price reached by the data
Close: Opening rate of the share
Volume: Total sales of that day
Loading the csv file in the numpy and assigning the input features with the desired columns and output feature with last "Volume" column. X is the list of predictors whereas Y is the output variable

```
#Loading Dataset
trainData = np.loadtxt("Google_Stock_Price_Train.csv", delimiter=",")

#Setting Predictors and Target Values
features_data = trainData[:,0:4]
target_data = trainData[:,4]
```

Looking the dataset:

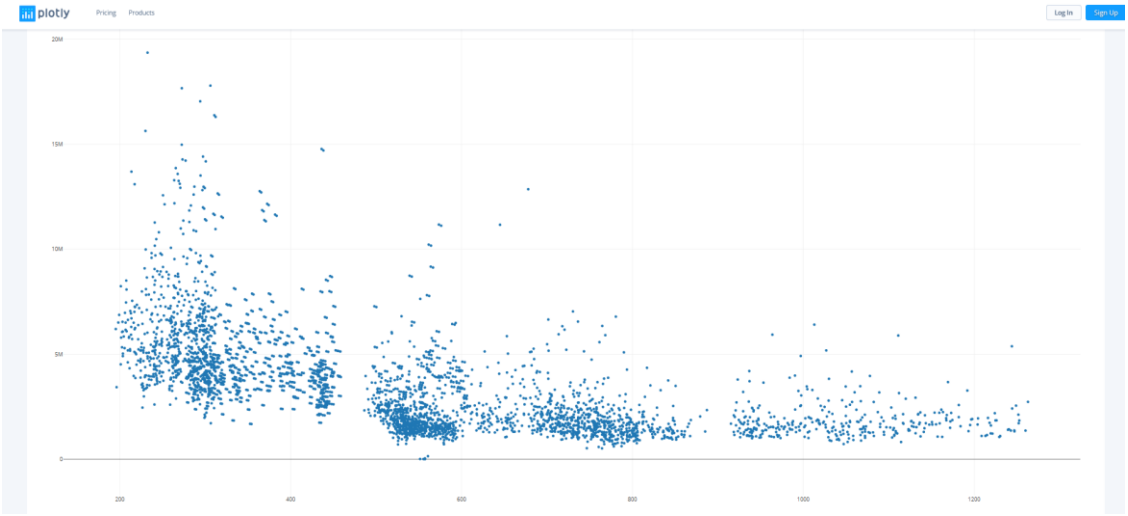
	0	1	2	3	4
0	325.25	332.83	324.97	663.59	7.3805e+06
1	331.27	333.87	329.08	666.45	5.7494e+06
2	329.83	330.75	326.89	657.21	6.5903e+06
3	328.34	328.77	323.68	648.24	5.4059e+06
4	322.04	322.29	309.46	620.76	1.16888e+07
5	313.7	315.72	307.3	621.43	8.824e+06
6	310.59	313.52	309.4	624.25	4.8178e+06
7	314.43	315.26	312.08	627.92	3.7644e+06
8	311.96	312.3	309.37	623.28	4.6318e+06
9	314.81	314.81	311.67	626.86	3.8328e+06
10	312.14	315.82	309.9	631.18	5.544e+06

Exploratory data analysis and feature selection:

EDA helps us in the better understanding of data and only using that we derive out trends and relationship among variables. That ultimately results in generation and selection of useful features that directly impact the model performance.

Data visualization is the presentation of data in a pictorial or graphical format. It enables decision makers to see analytics presented visually, so they can grasp difficult concepts or identify new patterns. Plotting scatter plot using plotly library to show the open values vs volume of the share sold that day. X axis contains the open values for each day and Y axis contains the values for Volume.

```
# Plotting the graph
xd=trainData[:,2]
yd=trainData[:,4]
layout = dict(title = 'Average Open and Volumes of Shares',
              xaxis = dict(title = 'Open'),
              yaxis = dict(title = 'Volume'),
              )
trace = go.Scatter(
    x = xd,
    y = yd,
    mode = 'markers'
)
data = [trace]
py.plot(data, layout)
```



REGRESSION USING PYTORCH:

Regression is a technique used to model and analyze the relationships between variables and often times how they contribute and are related to producing a particular outcome together. A linear regression refers to a regression model that is completely made up of linear variables. Beginning with the simple case, Single Variable Linear Regression is a technique used to model the relationship between a single input independent variable (feature variable) and an output dependent variable using a linear model i.e a line.

The more general case is Multi Variable Linear Regression where a model is created for the relationship between multiple independent input variables (feature variables) and an output dependent variable. The model remains linear in that the output is a linear combination of the input variables. We can model a multi-variable linear regression as the following:

$$Y = \beta_{1X_1} + \beta_{2X_2} + \beta_{3X_3} \ldots + \beta_{mX_m} + b$$

Where β_m are the coefficients, $\mathrm{X_m}$ are the variables and b is the bias. As we can see, this function does not include any non-linearities and so is only suited for modelling linearly separable data. It is quite easy to understand as we are simply weighting the importance of each feature variable $\mathrm{X_m}$ using the coefficient weights β_m. We determine these weights β_m and the bias b using a Stochastic Gradient Descent (SGD).

Splitting the data in training and testing parts so that we can make the model learn on training dataset and test the accuracy on testing set.

After this, we select the optimiser and the loss criteria. Here, we will use the mean squared error (MSE) as our loss function and stochastic gradient descent (SGD) as our optimiser. Also, we arbitrarily fix a learning rate of 0.01.

We now arrive at our training step. We perform the following tasks for 1000 times during training:

1. Perform a forward pass by passing our data and finding out the predicted value of y.
2. Compute the loss using MSE.
3. Reset all the gradients to 0, peform a backpropagation and then, update the weights.

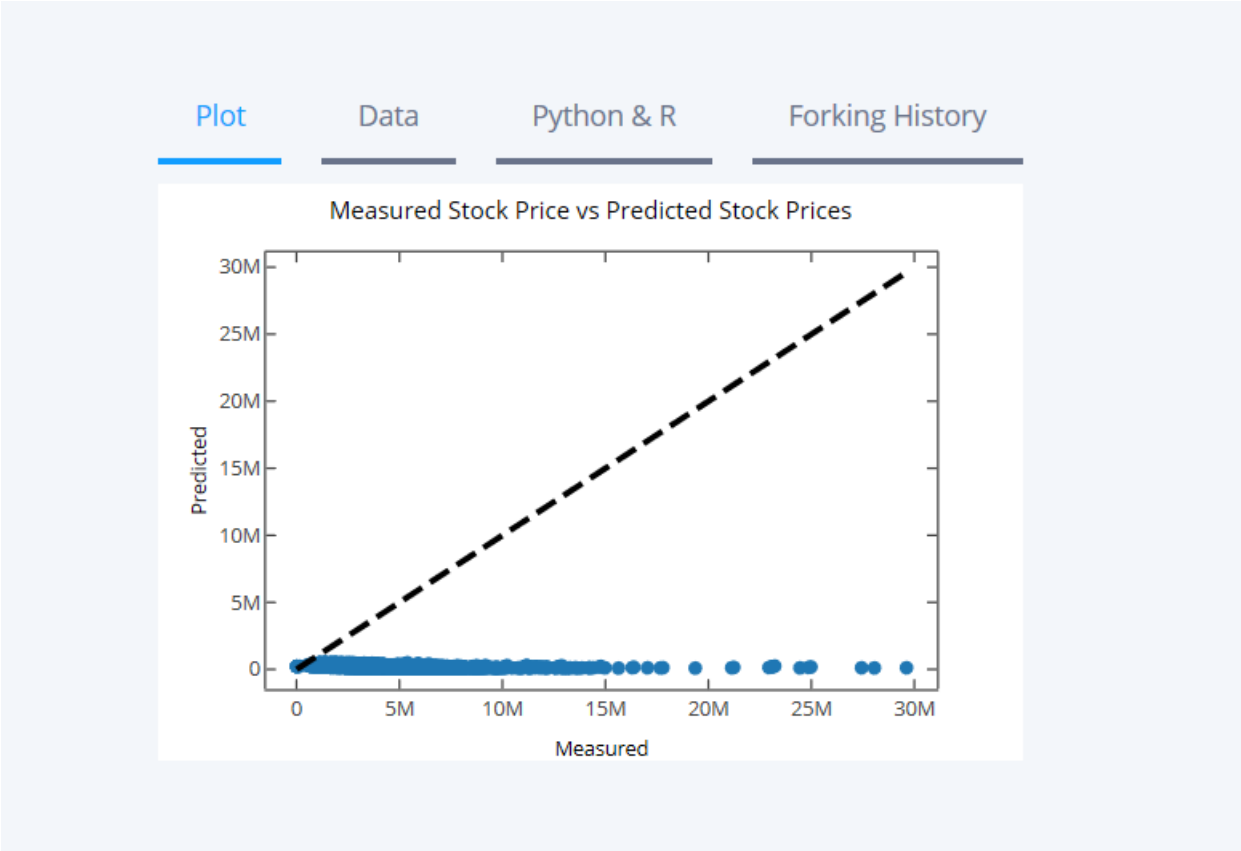
```
#Tuning Predictors with the Pytorch datatype
predictors = torch.from_numpy(np.array(features_data)).float()
predictors = Variable(predictors)

outputData = torch.from_numpy(np.array(target_data)).float()
outputData = Variable(outputData)

#Perfomring Linear Regression on the training dataset
linearRegression = nn.Linear(4, 1)
```

TRAINING MODEL:

```
#Running the loop for 1000 times to findout the best accuracy result
for ep in range(EPOCHS_COUNT):
    linearRegression.zero_grad()
    data_output = linearRegression(predictors)
    loss = criterion(data_output.squeeze(0), outputData.unsqueeze(1))
    loss.backward()
    optimizer.step()
    print('epoch {}'.format(ep))
```



Here we have three plots which represent the fit of the regression line. We have the actual prices on one side which is represented by y and we have the predicted price (y ̂)on the other side. We fit the line using the RMSE function given above. Here we could have used the absolute values or the L1 norm but we don't use it since it's not differentiable. We determine the weights βs and bias b by using SGD. There are two hyperparameters which we have to tune. One is the number of epochs and the other is the learning rate. In Figure 1 we have set learning rate = 0.01 and number of epochs =10000. In Figure 2 we have set learning rate = 0.1 and number of epochs =1000.

CONCLUSION:

So we can handover this model to the client. Where client can insert the new or targeted day details and get the prediction of the Google stock price. Also, client can use this model on the specific targeted month to increase there overall volume of the shares.

COMPARING PART 1 (KERAS) WITH PART 2 (PYTORCH):

Keras and PyTorch differ in terms of the level of abstraction they operate on.

Keras is a higher-level framework wrapping commonly used deep learning layers and operations into neat, lego-sized building blocks, abstracting the deep learning complexities away from the precious eyes of a data scientist.

PyTorch offers a comparatively lower-level environment for experimentation, giving the user more freedom to write custom layers and look under the hood of numerical optimization tasks. Development of more complex architectures is more straightforward when you can use the full power of Python and access the guts of all functions used. This, naturally, comes at the price of verbosity.

After performing deep learning using Keras and Pytorch on Churn data it has been found that Keras library performs faster. So, we observe that Part 1 that is Keras and Bokeh is better for predicting the churn modelling dataset than Part 2 that is Pytorch and Plotly.

Summing up the conclusion we can say:

- Keras – more concise, simpler API
- PyTorch – more flexible, encouraging deeper understanding of deep learning concepts