REPORT

{INTRODUCTION}

One of the most trendy topic ,in this period, is the crypto coins.

In order to improve our know-how in this topic, we choose this for the Project.

The goal of this project is predict the price of the Bitcoin.

We choose Bitcoin because it is the most important and representative crypto coin in this moment.

Moreover, the information about Bitcoin ,that we used in the project, comes from Bitstamp Exchange, one of the most important Exchange in all the Crypto Sector.

Now, we do a small digression about exchanges in order to improve the understanding of the incoming chapters.

An Exchange is a Web Site that allows his users to buy Crypo coins using real money or others Crypto.

{DATA EXTRACTION}

For Data Extraction, we have used the CryptoWatch API that is ,practically, a REST Web Service.

This Web Service provides a long list of function which provides several informations about Crypto, Exchanges, Market Trades etc…..

In order to extract the right data for our purposes, we have used exclusively the **ohlc** function.

This function takes in input a coin, an exchange and optionally: a time interval from which start the query and some candles types (for types we intend the time-interval of the candle).

The output of ohlc function is a series of candles ,regarding the coin and the exchange chosen, divided in types starting from a date chosen in the function call.

Another important explanation to do is about candles.

In the Crypto Coins ambient, for candle is intended a variation of a Crypto Price in a time interval.

The reason of name “candle” is that usually this price variation is represented in a particular Graphic where every variation is drawed in an Object that looks like a candle.

Usually, a candle is synthesized in 6 parameters (this is ,also, the CryptoWatch Representation):

- **CloseTime**: it is the UNIX timestamp for the end of candle time interval.
- **OpenPrice:** it is the starting price of the candle.
- **HighPrice:** it is the highest price in the time interval.
- **LowPrice:** It is the lowest price in the time interval.
- **Volume:** it is an index calculated from the following formula:
  “ coin_value **x** number_of_trades_in_the_time_interval”.
  (a trade is a transaction of a coin in a specific exchange).
- **ClosePrice:** It is the coin price at the end of candle time interval.

For our project, we choosed the 12 hour candles, we choose these because it was the best trade-of between quality of data for prediction and numbers of samples into the Dataset.

Now we are going to explain the core of this chapter, the Data Extraction.

For Data Extraction, we built several classes starting from the **Dataset_Creator** class that allow you to download (in a JSON file) the result of a Query to the CryptoWatch REST API.

This class uses others classes ,built by us, in order to perform his goal like the **Extractor** class for retrieving the result of a Query and the **JSON_Saver** class to save the result of the query in a json file.

In essence, in this phase of the project we used **“Dataset_Creator”** class for downloading the 12-hour candles data starting from the year 2010.

Of course, the downloaded data regard Bitcoin coin and Bitstamp Exchange as previously said.

LIST ABOUT BUILT SOFTWARES IN THIS PHASE: Extractor, Time_Stamp_Converter, Periods_Maker, JSON_Saver, Dataset_Creator.

{DATASET BUILDING}

In order to make a Dataset starting from the previous downloaded data, we used ,again, the **“Dataset_Creator”** class.

In particular, this class has a method that allow you to build a Dataset ,starting from a JSON file, and saves its in a **pkl** file, accessible in future.

Entering in the specific, the building of Dataset consists in the creation of 2 Matrix:

- **Observations Matrix,** this matrix contains all the useful data in order to predict the price of the Bitcoin at the end of the candle.

- **Labels Matrix,** this matrix (Column Vector in the specific) contains all the “ClosePrice” values for every candle coming from the Dataset.

Another important thing to say is that we voluntarily remove a Feature into the Dataset because it wasn’t documented from the authors of the CryptoWatch REST API.

LIST ABOUT BUILT SOFTWARES IN THIS PHASE: Dataset_Creator, Dataset_Maker.

{DATA VISUALIZATION}

{LEARNING}

For Learning phase, we started with the split of the Dataset into Training-Set, Validation-Set and Test-Set, using the ,built by us, **“Dataset_Creator“** class.

In particular this split is divided as follows: 70% of Dataset for Training and Validation Set, and the remaining 30% for Test-Set.

We split the Dataset in Training-Set and Test-Set because the Training-Set is useful for fitting the Algorithm, instead Test-Set is useful for evaluate the performance of algorithm in case of data never seen before.

We introduce the Validation Test for Training reasons, infact in the Learning phase we compare a series of models.

Consequently, the Validation-Set need in order to choose the best Model.

In order to improve the performance of the Algorithm, we use a Variable Validation-Set using Cross-Validation techniques, in particular we used the K-Fold Validation (with k=5).

Infact, with a variable Validation-Set we decrease the probability of choosing a unfortunately configuration.

In addition to this, we applied the PCA trasformation to the Dataset.

We applied her because she orders features based on the importance (this is very useful when we will plot the Hypothesis plot, we obtain a more significant graphics because we’ll plot only the most important features), moreover PCA deletes the non-independent features.

The last refining of the Dataset has been the Normalization, infact we applied the Normalization to every feature of the Dataset in order to increase the algorithm execution speed-up

For this task we used the ,built by us, Cleaner class.

In order to predict the price, we choosed the fallowing Models:

[x_1 =CloseTime, x_2 =OpenPrice, x_3 =HighPrice, x_4 =LowPrice, x_5 =Volume]

MODEL 1: $h\theta(x) = \theta_0 + \theta_1 x_1$

MODEL 2: $h\theta(x) = \theta_0 + \theta_1 x_1^2$

MODEL 3: $h\theta(x) = \theta_0 + \theta_1 x_1^3$

MODEL 4: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

MODEL 5: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2$

MODEL 6: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2$

MODEL 7: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$

MODEL 8: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_5$

MODEL 9: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2 + \theta_3 x_5$

MODEL 10: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2 + \theta_3 x_5$

MODEL 11: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2 + \theta_3 x_5^2$

MODEL 12: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_{3+} + \theta_4 x_{4+} + \theta_5 x_5$

MODEL 13: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2 + \theta_3 x_{3+} + \theta_4 x_{4+} + \theta_5 x_5$

MODEL 14: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2 + \theta_3 x_{3+} + \theta_4 x_{4+} + \theta_5 x_5$

MODEL 15: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_{3+} + \theta_4 x_{4+} + \theta_5 x_5^2$

MODEL 16: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_{3+} + \theta_4 x_{4+} + \theta_5 x_5$

In the Model decision, we have given emphasis on the CloseTime and OpenPrice features.

Moreover, at the geometric level we have chosen straight lines, parables and Circumferences like geometric places designed to approximate the data.

The training of Models uses the ,built by us, **Trainer** class which ,in turn, uses the **sklearn LinearRegression** class and the ,built by us, **“Dataset_Projector”** class.

In substance, the training consists in the following phases:

- Every Model has been trained using the **“fit”** method of LinearRegression class, fit method was applied to the Training-Set using the k-cross validation strategy.
  Of course, the Training-Set needs to be restricted in order to have a perfect match with the model that we want to train.
  This service was performed by the **Dataset_Projetor** class.
- For every Model, we calculated the Training-Error and the Validation-Error (the error calculated on the Validation Set).
  The Error Function used was the MAE (Mean Absolute Error).
- Finally, we have chosen the Model with the lowest Validation-Errror.

With the goal of a future inference, we used the **“save_the_best_estimator”** method of Trainer class for saving ,in a pkl file, the best estimator (the corresponding LinearRegression object) and the best estimator features.

LIST ABOUT BUILT SOFTWARES IN THIS PHASE: Trainer, Dataset_Projector, Curve, Recorder_Degree, Model_Printer, learn.py.

{PERFORMANCE EVALUATION}

For the performance evaluation, we used the MAE error function.

In substance, we calculate the error using the **“mean_absolute_error”** on the Test-Set.

The result becomes the final Error of our Algorithm.

LIST ABOUT BUILT SOFTWARES IN THIS PHASE: learn.py

{INFERENCE}

This have been the final phase of the project, pratically we built a quick command line program which takes in input an opened candle (only with some specific parameters choosed in the Training phase) and returns in output a prediction of the ClosePrice of that candle.

LIST ABOUT BUILT SOFTWARES IN THIS PHASE: cryptowolf.py