TELECOM PARIS

PROJET PRIM

Social Media Posts on Cryptocurrencies: bullish or bearish ?

author:
Maha Meihemid

Supervised by:
M. Mauro Sozio

February 15, 2022

Contents

Liste des figures

1 Abstract

Bitcoin is the first decentralized cryptocurrency, which was first released as open-source software in 2009. It has the potential to be a native currency of the internet and to revolutionize the world banking system. Researchers have observed a correlation between cryptocurrency price trends and posts in social media, which motivates our project.
In this project, we try to build a powerful classifier for classifying posts from social media such as Twitter and Reddit. The project includes all the main steps of machine learning, from data collection to building an appropriate architecture and selecting the model.We try in first, simple models and tools such as TF-IDF and Word2vec for word embedding, SVM, MLP, LSTM, K nearest neighbors and Random forest for classification, then we move to the state of the art models such as BERT [Dev+19], [Vas+17] and RoBERTa [Liu+19] for embedding and deep neural networks to fine tune the BERT and RoBERTa model for our down-stream task.

2 Introduction

The aim of this part is to give an overview of this project. It presents motivations, the objectives and finally, we present the structure of the project.

2.1 Motivations

Natural language processing NLP as a sub field of machine learning is a very promising approach for bridging the gap between natural language and computers. Recently many platforms felt the need to automate tasks which deal with natural language. For example topic identification, document classification etc. NLP algorithms represent a paradigm shift from traditional methods toward powerful data-driven approaches. For topic identification there are mainly two stages, first is designing a word embedding model that maps words in Vector space and second is a construction of classifier that maps vectors into a specific class. Deep learning is used in both stages. Deep learning seems to be an appealing approach both for learning numerical representation as done by Word2vec, BERT [Dev+19], ELECTRA [Cla+20], and for classification. One of the state of the art tools for NLP and computer vision is what we call attention mechanism. It allows the model to learn where to focus on a sequence of words to produce an output. In this project, we are mainly concerned with sentiment analysis and in particular the task consists of labeling whether a given social media post on a cryptocurrency

(e.g. Bitcoin, Ethereum etc.) is “bullish” or “bearish”: “bullish” means that the author of the post believes that the price of Bitcoin will rise, while “bearish” means that he/she believes that the price will drop. Observe that this is slightly different than identifying whether a post contains a positive or negative sentiment about Bitcoin. In particular a post could be positive but not necessarily bullish, e.g. ”Bitcoin performed great last year!”, while a post of the kind “Bitcoin to 100k US” is clearly bullish, as the current value is much smaller than “100k US”.
So to summarize the project is done in two main steps: collecting data on cryptocurrencies from social media such as Reddit (using API pushift) and Twitter and classifying posts as bullish or bearish using state of the art text classifiers such as BERT and its variants as well as manual labelling.

2.2 Objective

Improve the results as much as possible, ideally obtaining an accuracy of 0.9 or larger. In order to respect the Occam’s razor principle, we will try to first, test simple models (SVM, MLP etc). In case of poor capacity of generalization upon unseen data we will jump to more sophisticated models and tools such as BERT, ELECTRA, LSTM combined with attention mechanism.

2.3 Project structure

First we try to give an overview and details about the main used tools for our task. We will introduce different word embedding and model architecture in a form of literature review, then we will describe the dataset we have, and finally we will present how we combine the previous tools to construct a powerful classification system.

3 Background and Literature Review

In this section we try to go in details in the different tools that are used in our project, we will analyse them from different points of view. It will cover from traditional Machine Learning models to large pre-trained Deep Learning models. We will give our intuition about the capacity of generalization for some classifiers.

3.1 Word2vec

In their paper [Mik+13] Tomas Mikolov and his team created a set of word embedding models. They trained their models on trillions of words and quickly became popular for numerous NLP tasks. In the previous cited paper there are mainly two architecture, which are CBOW and Skip-gram.

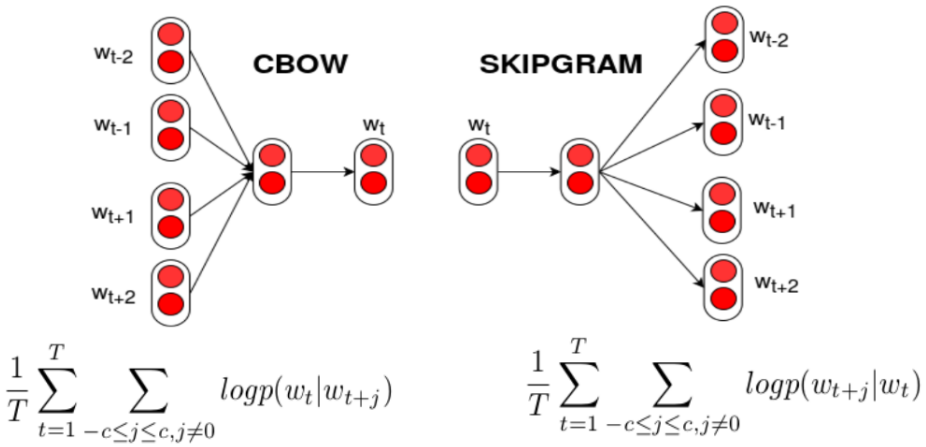

Figure 1: Word2vec architectures, from Wikipedia.

The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. The input is typically a one hot encoding of the different words in a given sentence, the model tries to output a masked word in the input based only on the projection representation in the latent space. This latent space will be the Word2vec representation. The latent space has a smaller dimension and capture features in the target word. Skip-gram is quite the inverse process of the CBOW architecture. It takes the one hot encoding of an input word, then project it in a latent space and from this representation we impose on the model to generate the context of the given word. This architecture is based on the intuition : if we are able to reconstruct the context from a projection of the one-hot encoding of a word, then this projection contains a considerable amount of information about the starting vector and then we can use it as a vector representation. CBOW and Skip-gram in their time made paradigm shift in the word embedding world.

3.2 BERT

3.2.1 The architecture

BERT was first proposed by Google AI Language, Jacob Devlin and his co-author published [Dev+19]. They introduced a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. They used transformers model as described in [Vas+17] as a building block for the BERT architecture. Indeed BERT is a stack of encoders. Each encoder contains mainly a multi head self attention and a set of feed forward neural network. The attention mechanism used in BERT takes in input a sequence of word representation $x_1, ...x_T$ and output a sequence of a new representation $z_1, ..., z_T$ each z_i is the weighted sum of a linear transformation of the x_i. The weights in the sum are a learnable parameters using a pretext task. This gives the model the capacity to learn where to focus in a given sequence. Here is the equation that describes the relation between z_i and x_i :

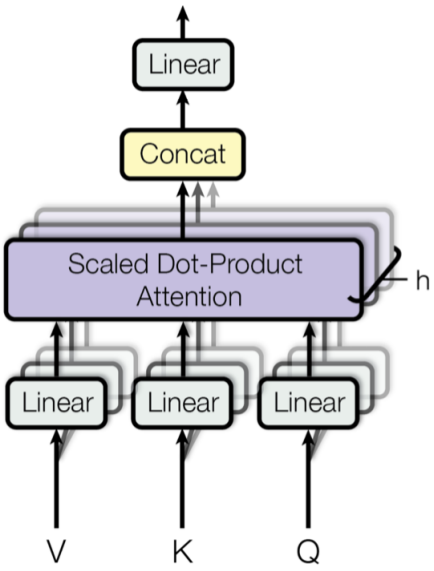

Figure 2: Multi Head Attention Block.

$$\text{MultiHead}\,(Q,K,V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)\,W^o \quad (1)$$

where $\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$, and $\text{Attention}\,(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$.

And Q, K, V are linear transformation of $X = [x_1, ..., x_T]$. So the attention mechanism is just a fancy name for the weighted sum of the columns of V.

3.2.2 The pre-training

In the previous subsection we described the architecture of the BERT model. Given this architecture BERT use a two pretext task in order to learn a dynamic embedding.
The first task is the **Masked Language Model (MLM)**, it consists of replacing 15% of the input sequence's tokens by a special token [MASK]. Of the selected tokens, BERT masks 80%, leaves 10% unchanged and replaces the remaining 10% by a randomly selected vocabulary token. Then the model learns to predict the masked tokens by minimizing a cross entropy loss.
The second task is the **Next Sentence Prediction (NSP)** which is a binary classification loss to detect if two sequences follow each other in the original text. Positive labels are constructed simply by taking a true consecutive sentences from the original text corpus. Negative examples are made putting together segments from different texts. Through minimizing the loss, the model tries to learn which pairs are successive or not.

3.2.3 Data

BERT is trained on both BookCorpus [Liu+19] and English Wikipedia, which contains 16GB of text. For optimization BERT uses the Adam algorithm.

3.2.4 BERT performance

BERT took advantage from the transformers architecture, it is conceptually simple and empirically powerful. It obtains new state of the art results on eleven natural language processing tasks, It did achieve 80.5% in the GLUE score. In the following figure we present in details its performance published in [Dev+19].

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

Figure 3: Bert performance.

These performances in the GLUE test make us having confidence in it. BERT is easy to fine tune for a specific task and in particular for our problem of classification.

3.3 RoBERTa

3.3.1 The architecture

In [Liu+19] Yinhan Liu and his coleagues wrote that BERT was significantly undertrained and propose an improved recipe for training BERT models, which they call RoBERTa, it can match or exceed the performance of all of the post-BERT methods. Their modifications are simple, they include: (1) training the model longer, with larger batches, over more data; (2) removing the next sentence prediction objective; (3) training on longer sequences; and (4) dynamically changing the masking pattern applied to the training data. They also collected a large new dataset (CC-NEWS) of comparable size to other privately used datasets, to better control for training set size effects. Indeed BERT and RoBERTa are very similar. There is no differences in the architecture, the only difference is in the pre-training phase.

3.3.2 Performance

In the following table we present the performance of the RoBERTa model.

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Figure 4: RoBERTa performance.

3.4 Feedforward neural network and LSTM

Firstly we will describe the feedforward architecture, then show its limits and finally we will move to LSTM.
A feedforward neural network (or MLP with L layers and m neurons in each layer), is a composition of L+1 matrix projection plus an addition of a bias vector followed by a non linear function. The goal of an MLP is to find the matrices and biases in order to approximate a given function $f*$. They helped a lot the community to build a very accurate model. They were able to learn the XOR function and combined with regularization, they reached high accuracy and capacity of generalization in many tasks. Unfortunately they are not very optimized to deal with some specific data structure. For example, sequential data, as sentences may be a challenge for such architecture. If the data is formed by a set of sequence of the form $x^{(1)}, .., x^{(T)}$ then an architecture like MLP must be connected to all of the sequences (no weight sharing) which is computationally expensive. This limitation is completely overcome by the LSTM architecture introduced in [HS97]. LSTM make a recursion computation on the terms of the sequence in a way to propagate a memory of the previous context.

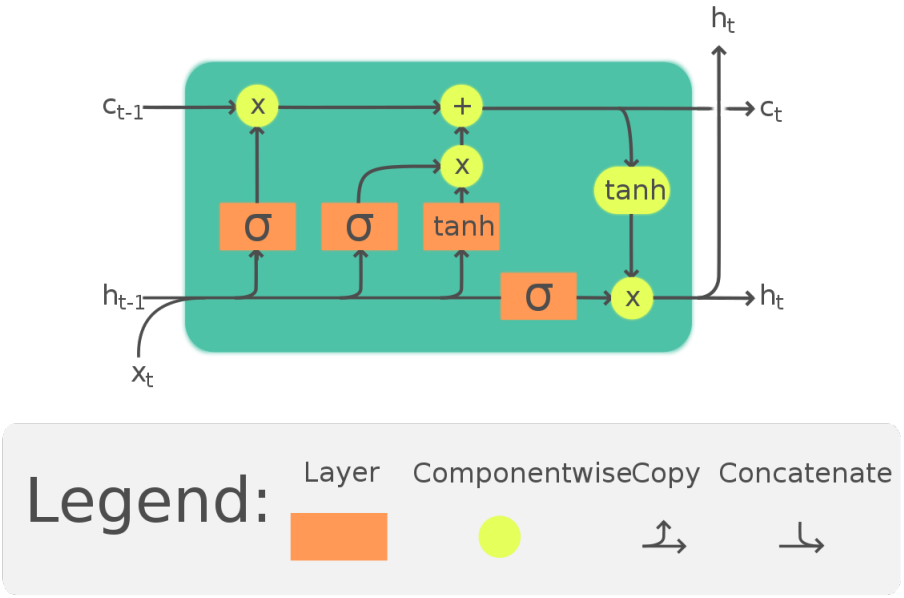

Figure 5: LSTM CELL.

More formally :

$$\mathbf{i}_t = \sigma\left(\mathbf{x}_t\mathbf{U}^i + \mathbf{h}_{t-1}\mathbf{W}^i + \mathbf{b}_i\right) \tag{2}$$
$$\mathbf{f}_t = \sigma\left(\mathbf{x}_t\mathbf{U}^f + \mathbf{h}_{t-1}\mathbf{W}^f + \mathbf{b}_f\right) \tag{3}$$
$$\mathbf{o}_t = \sigma\left(\mathbf{x}_t\mathbf{U}^o + \mathbf{h}_{t-1}\mathbf{W}^o + \mathbf{b}_o\right) \tag{4}$$
$$\mathbf{q}_t = \tanh\left(\mathbf{x}_t\mathbf{U}^q + \mathbf{h}_{t-1}\mathbf{W}^q + \mathbf{b}_q\right) \tag{5}$$
$$\mathbf{p}_t = \mathbf{f}_t * \mathbf{p}_{t-1} + \mathbf{i}_t * \mathbf{q}_t \tag{6}$$
$$\mathbf{h}_t = \mathbf{o}_t * \tanh\left(\mathbf{p}_t\right) \tag{7}$$

With this architecture, the weight sharing is satisfied and in the same time thanks to p_t and q_t the information is propagated through each term of the sequence. Before the coming of transformers, LSTM in the encoder-decoder mode had reached the state of the art in machine translation.

3.5 K-nearest neighbors

One of the major principles of machine learning is **the empirical risk minimization (ERM)** principle, for more details we recommend the Vapnik's paper where he is the main author [Vap92]. Some algorithms use other approaches. The plug-in approach is one of them, it consists of constructing an

estimator to the a posteriori probability defined by $\mu(x) = P(Y = 1|X = x)$ then plug it in the classifier formula.
In the book entitled *A Probabilistic Theory of Pattern Recognition* by Luc Devroye, László Györfi and Gábor Lugosi we find that the K-nearest neighbors is strongly consistent with relatively weak hypothesis on the data generation. After reading the mathematical proof we were motivated to use KNN and we were doubtless about the capacity of generation on KNN on unseen data.

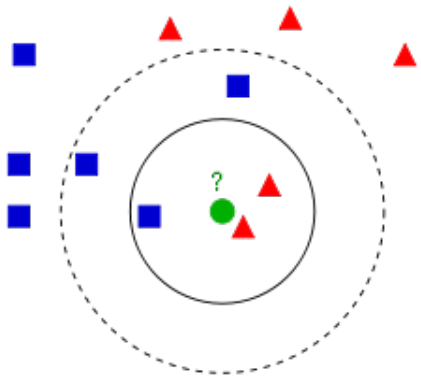

Figure 6: KNN with 2 labels.

3.6 Random forest

Introduced by Leo Breiman, random forest is based on the bagging principle. In bagging we are given a specific model, we train it on bootstrap data, let's say we have n batch. Every batch of the bootstrap data give birth to a model, then we average all the models to end up with a single robust model. We can show mathematically that the variance of the model given by the bagging model is decreased by a factor of $\frac{1}{n}$, however the bias remains the same. So it is very reasonable to use bagging when we have a model with low bias and high variance.
In random forest the bagging principle is used with randomized decision trees. In [SBV15] we find proof of consistency of random forest.

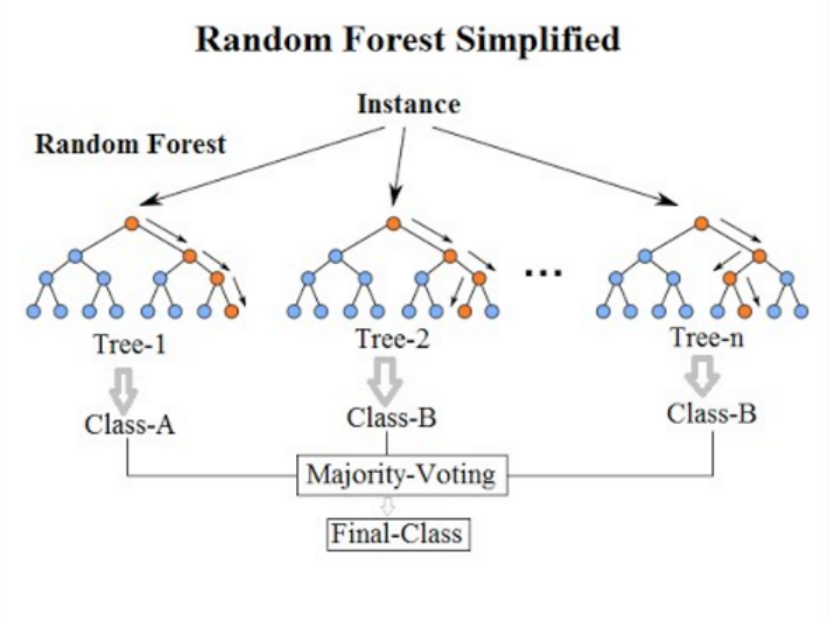

Figure 7: Random Forest.

4 Data, model selection and experiments

In this section we describe the data we have, the way of collection and the model we decide to use for solving our classification problem.

4.1 Dataset

In order to construct a classification system in the two classes "bullish", "bearish" we collected the data from Twitter and Reddit posts. We used the APIs provided by Twitter and Reddit. All data were collected between 01-01-2021 and 01-01-2022 (both dates are included).

4.1.1 Data Collection

- **Twitter** : Tweets are available and are easily retrieved making use of Twitter Application Programming Interface (API). Composing the hashtag #Bitcoin or #bit-coin, we are able to gather all tweets that mentioned the analyzed subject. We briefly describe the different components of our system.;

- **Reddit** : We collected data from Reddit by using Pushshift API. We obtained posts from r/Bitcoin, r/CryptoCurrency, r/CryptoCurrencyTrading and r/CryptoMarkets subreddits containing the keyword "Bitcoin".;

Then we filtered the data according to the following price related keywords: 'buy', 'bought', 'sell', 'sold', 'sodl', 'hold', 'hodl', 'fork', 'invest', 'long', 'short', 'purchase'..etc.

```
df = pd.read_csv('data.csv')
print(df.shape)
df.head(10)
# label 0 == bearish
# label 1 == bullish
```

```
(2900, 2)
```

	Post	label
0	Yes, 2x growth (100% only) long-term estimate ...	1
1	Bitcoin is risen. Amen.	1
2	Bitcoin still to double from here	1
3	Just got a "Crypto markers are highly volatile...	1
4	Absolument. Bitcoin will soon be back on top.	1
5	bitcoin is the hardest money humanity has ever...	1
6	You're only going to get more stressed if you ...	0
7	Bitcoin will go beyond 40k by Monday. I know t...	0
8	maybe we will get some bullish action toward t...	1
9	Personally I think we consolidate for another ...	1

Figure 8: First 10 elements of our DataSet.

Figure 9: Wordcloud created from our dataset.

Here is the distribution of the collected posts composing our dataset :

- **A train dataset** of 2320 sentences : 1553 labelled as "bullish" and 767 labelled as "bearish";
- **A test dataset** of 580 sentences : 388 labelled as "bullish" and 192 labelled as "bearish".;

Note that the sentiment value is a Boolean value equal to 0 in case of a bearish sentiment and 1 otherwise.

```
FOR TRAIN SET

Average word length of Post in train set is 14.
Max word length of Post in train set is 76.
Average character length of Post in train set is 83.
Label Distribution in train set is
1    1553
0     767
Name: label, dtype: int64.


FOR VALIDATION SET

Average word length of Post in valid set is 15.
Max word length of Post in valid set is 59.
Average character length of Post in valid set is 85.
Label Distribution in validation set is
1    388
0    192
Name: label, dtype: int64.
```

Figure 10: Data split information.

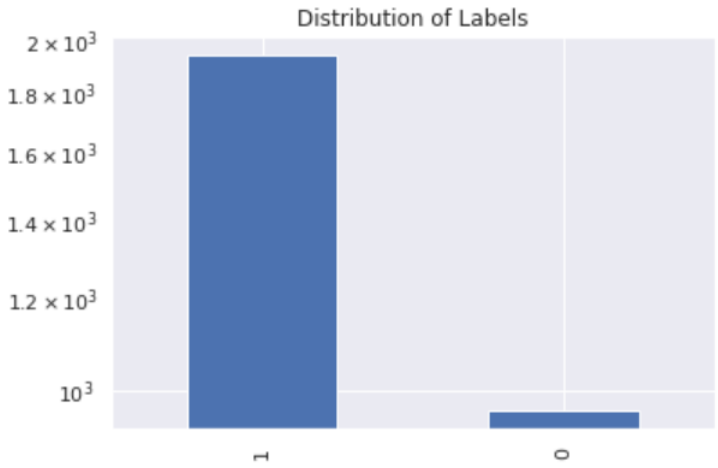

Figure 11: Distribution of labels.

The data is a little bit unbalanced, due to people being more optimistic about cryptocurrencies as they are looking for profit. We are going to deal with that by using the re-sampling technique.
The BERT model we are using from Hugging Face platform can only map a vector of length 512. The post we have do not exceed this number, as they are mostly tweets limited by 280 characters.
It was a very challenging task to collect data by ourselves. The labeling procedure took a long time. We tried some tools to make the labeling easier like applying clustering before labeling.

4.1.2 Data Cleaning

After using our tweet scraper and reddit scraper, we had to drop some columns that we deemed irrelevant for our analysis, this includes for example the tweet locations, tweet id, the list of mentions, hashtags, permalinks...etc. Before we are able to start doing any form of sentiment analysis, the tweets collected have to be cleaned.
We have to clean emojis from sentences, remove punctuation, links, mentions, \r\n new line characters, clean hashtags at the end of the sentence and keep those in the middle of the sentence by removing just the # symbol, filter special characters such as & present in some words, remove multiple spaces etc.

	Post	label	Post_clean
0	Yes, 2x growth (100% only) long-term estimate ...	1	yes 2x growth 100 only longterm estimate for b...
1	Bitcoin is risen. Amen.	1	bitcoin is risen amen
2	Bitcoin still to double from here	1	bitcoin still to double from here
3	Just got a “Crypto markers are highly volatile...	1	just got a crypto markers are highly volatile ...
4	Absolument. Bitcoin will soon be back on top.	1	absolument bitcoin will soon be back on top
5	bitcoin is the hardest money humanity has ever...	1	bitcoin is the hardest money humanity has ever...
6	You're only going to get more stressed if you ...	0	youre only going to get more stressed if you s...
7	Bitcoin will go beyond 40k by Monday. I know t...	0	bitcoin will go beyond 40k by monday i know th...
8	maybe we will get some bullish action toward t...	1	maybe we will get some bullish action toward t...
9	Personally I think we consolidate for another ...	1	personally i think we consolidate for another ...
10	I am still bullish on crypto for 2021	1	i am still bullish on crypto for 2021
11	I don't understand why. I feel bullish for the...	1	i dont understand why i feel bullish for the f...
12	Lot of bullish sentiment here. It’s gonna 💣up...	1	lot of bullish sentiment here its gonna up to ...
13	Bullish On Bitcoin, Bloomberg Predicts $400,00...	1	bullish on bitcoin bloomberg predicts 400000 p...

Figure 12: Data after cleaning.

4.2 Model selection and results

Our final system uses BERT as an embedding system followed by feed forward neural network in order to fine tune the BERT model.

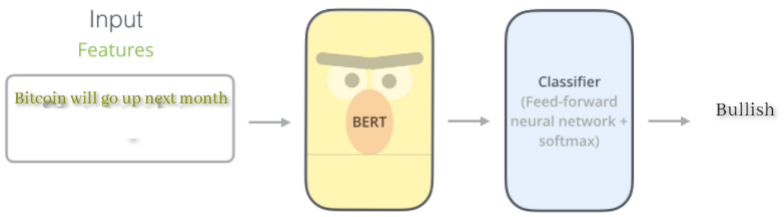

Figure 13: The system summary.

The weights of the BERT model are frozen, only the parameters of the feed forward neural network are learnable. The only changeable part of our system is the fine tuning part. We can play with the number of layers and neurons in each layer. With one layer of 768 neurons we reached 0.90 accuracy on test data. We tried to make the fine tuning deeper and we reached 0.89. We constantly tried to fine tune BERT by different architectures. We added $dropout = 0.5$ and $leaning\_rate = 2e^{-5}$ we then got 0.92. We tried also RoBERTa fine tuned with one layer and we got 0.93. In the experiments

section we will display the different epochs and the learning dynamic of each model we previously mentioned.

4.3 Challenges

In the beginning we did not know much about BERT and RoBERTa we were limited to use TF-IDF and Word2vec for embedding combined with classical classifiers as Support Vector Machine, random forest and K-nearest neighbors. We were not able to reach good accuracy. In fact we were not able to go beyond 0.8 on the testing data. With the guidance of our supervisor Mr. Mauro Sozio, we started to read papers about transformers, BERT, and other state of the art architectures. In the beginning we did not know about the Hugging Face platform, so we tried to use the implementation we found on Git-hub repositories. We found many problems in the dependencies we have to install and the whole procedure were tough. But the Hugging Face and TensorFlow hub platforms facilitates a lot the use of pre-trained models.

4.4 Experiments

Here are the different results we got from the different architectures we used. We display only the architectures that showed good capacity of generalization. We show also some graphs that help to visualize the data.

- **Report classification for KNN+Random Forest+MLP :**

Classification Report for KNN:

	precision	recall	f1-score	support
Negative	0.70	0.55	0.62	245
Positive	0.79	0.88	0.83	480
accuracy			0.77	725
macro avg	0.75	0.72	0.73	725
weighted avg	0.76	0.77	0.76	725

Classification Report for MLP:

	precision	recall	f1-score	support
Negative	0.71	0.75	0.73	245
Positive	0.87	0.84	0.85	480
accuracy			0.81	725
macro avg	0.79	0.79	0.79	725
weighted avg	0.81	0.81	0.81	725

Figure 14: Report for KNN+MLP.

Classification Report for Random Forest:

	precision	recall	f1-score	support
Negative	0.00	0.00	0.00	245
Positive	0.66	1.00	0.80	480
accuracy			0.66	725
macro avg	0.33	0.50	0.40	725
weighted avg	0.44	0.66	0.53	725

Figure 15: Report for Random Forest.

- **BERT fine tuned with one layer :**

```
Epoch 1/7
73/73 [==============================] - 75s 840ms/step - loss: 0.6195 - accuracy: 0.6552 - val_loss: 0.5120 - val_accuracy: 0.7603
Epoch 2/7
73/73 [==============================] - 62s 851ms/step - loss: 0.4837 - accuracy: 0.7841 - val_loss: 0.3774 - val_accuracy: 0.8534
Epoch 3/7
73/73 [==============================] - 62s 845ms/step - loss: 0.2888 - accuracy: 0.8927 - val_loss: 0.3113 - val_accuracy: 0.8862
Epoch 4/7
73/73 [==============================] - 62s 848ms/step - loss: 0.1730 - accuracy: 0.9405 - val_loss: 0.3078 - val_accuracy: 0.8828
Epoch 5/7
73/73 [==============================] - 62s 846ms/step - loss: 0.0961 - accuracy: 0.9750 - val_loss: 0.3384 - val_accuracy: 0.9017
Epoch 6/7
73/73 [==============================] - 62s 847ms/step - loss: 0.0516 - accuracy: 0.9866 - val_loss: 0.3532 - val_accuracy: 0.8914
Epoch 7/7
73/73 [==============================] - 62s 848ms/step - loss: 0.0346 - accuracy: 0.9892 - val_loss: 0.3840 - val_accuracy: 0.9017
```

	precision	recall	f1-score	support
Negative	0.86	0.84	0.85	192
Positive	0.92	0.93	0.93	388
accuracy			0.90	580
macro avg	0.89	0.89	0.89	580
weighted avg	0.90	0.90	0.90	580

Figure 16: Classification report for BERT_1.

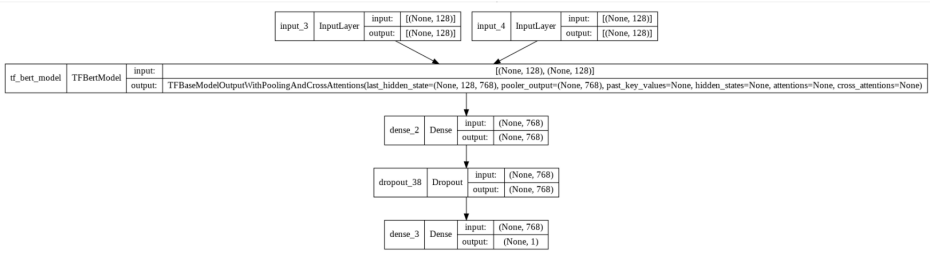

Figure 17: Model architecture.

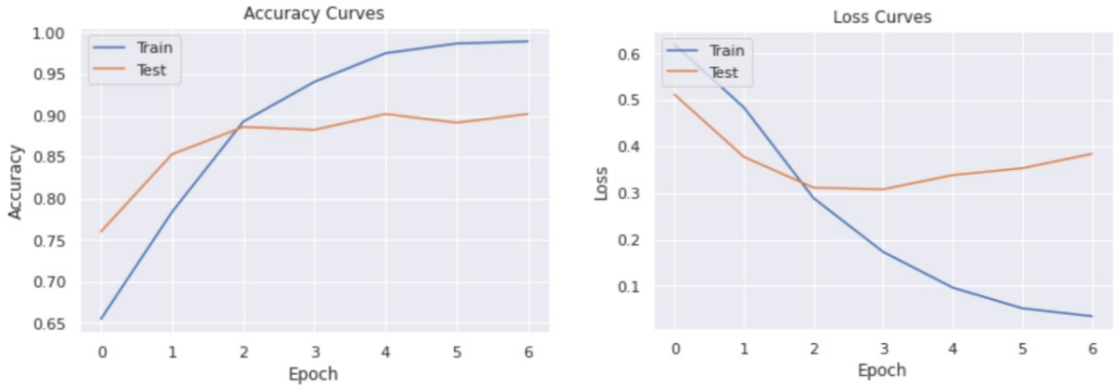

Figure 18: Accuracy Curves and Loss Curves for BERT_1.

- **BERT fine tuned with one layer, dropout=0.5 :**

```
Epoch 1/7
73/73 [==============================] - 79s 885ms/step - loss: 0.1396 - accuracy: 0.9603 - val_loss: 0.3194 - val_accuracy: 0.9034
Epoch 2/7
73/73 [==============================] - 62s 846ms/step - loss: 0.0384 - accuracy: 0.9897 - val_loss: 0.3817 - val_accuracy: 0.9103
Epoch 3/7
73/73 [==============================] - 62s 847ms/step - loss: 0.0300 - accuracy: 0.9909 - val_loss: 0.4402 - val_accuracy: 0.9052
Epoch 4/7
73/73 [==============================] - 62s 846ms/step - loss: 0.0341 - accuracy: 0.9892 - val_loss: 0.4004 - val_accuracy: 0.9155
Epoch 5/7
73/73 [==============================] - 62s 847ms/step - loss: 0.0096 - accuracy: 0.9987 - val_loss: 0.4238 - val_accuracy: 0.9241
Epoch 6/7
73/73 [==============================] - 62s 848ms/step - loss: 0.0055 - accuracy: 0.9987 - val_loss: 0.5395 - val_accuracy: 0.9052
Epoch 7/7
73/73 [==============================] - 62s 848ms/step - loss: 0.0056 - accuracy: 0.9987 - val_loss: 0.4407 - val_accuracy: 0.9241
```

	precision	recall	f1-score	support
Negative	0.87	0.91	0.89	192
Positive	0.95	0.93	0.94	388
accuracy			0.92	580
macro avg	0.91	0.92	0.92	580
weighted avg	0.93	0.92	0.92	580

Figure 19: Classification report for BERT_2.

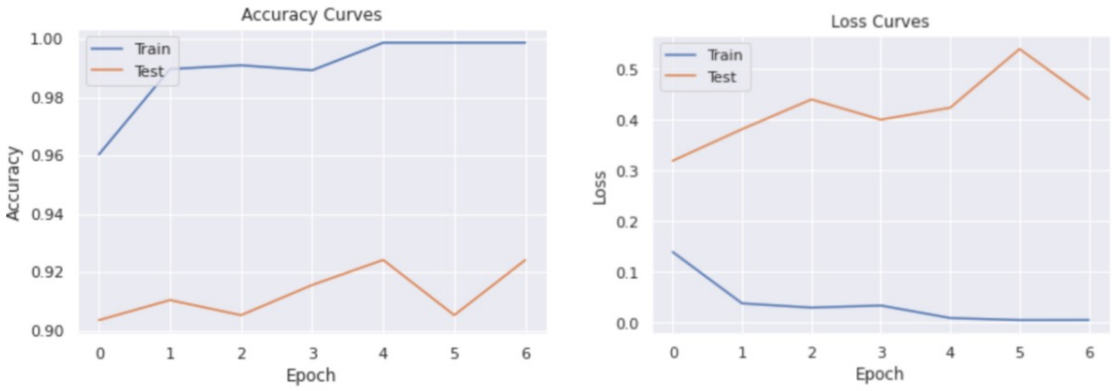

Figure 20: Accuracy Curves and Loss Curves for BERT_2.

- **BERT fine tuned with four layers :**

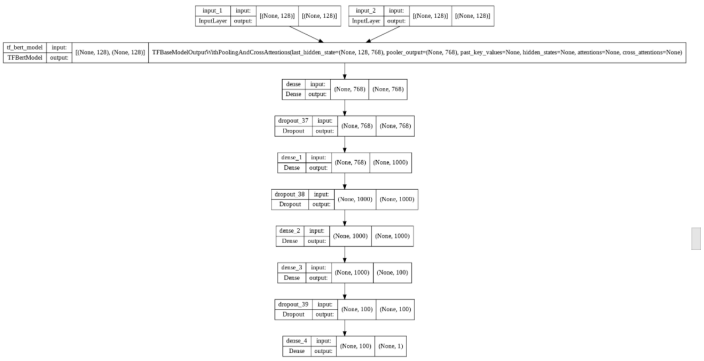

Figure 21: Model architecture.

```
Epoch 1/7
73/73 [==============================] - 147s 2s/step - loss: 0.6552 - accuracy: 0.6198 - val_loss: 0.5687 - val_accuracy: 0.6983
Epoch 2/7
73/73 [==============================] - 120s 2s/step - loss: 0.5393 - accuracy: 0.7478 - val_loss: 0.5244 - val_accuracy: 0.7690
Epoch 3/7
73/73 [==============================] - 120s 2s/step - loss: 0.3889 - accuracy: 0.8328 - val_loss: 0.3548 - val_accuracy: 0.8621
Epoch 4/7
73/73 [==============================] - 120s 2s/step - loss: 0.2564 - accuracy: 0.9056 - val_loss: 0.3566 - val_accuracy: 0.8793
Epoch 5/7
73/73 [==============================] - 120s 2s/step - loss: 0.1569 - accuracy: 0.9522 - val_loss: 0.3944 - val_accuracy: 0.8862
Epoch 6/7
73/73 [==============================] - 120s 2s/step - loss: 0.1049 - accuracy: 0.9672 - val_loss: 0.4573 - val_accuracy: 0.8707
Epoch 7/7
73/73 [==============================] - 120s 2s/step - loss: 0.0658 - accuracy: 0.9832 - val_loss: 0.4474 - val_accuracy: 0.8897
```

	precision	recall	f1-score	support
Negative	0.84	0.82	0.83	192
Positive	0.91	0.93	0.92	388
accuracy			0.89	580
macro avg	0.88	0.87	0.87	580
weighted avg	0.89	0.89	0.89	580

Figure 22: Classification report for BERT_3.

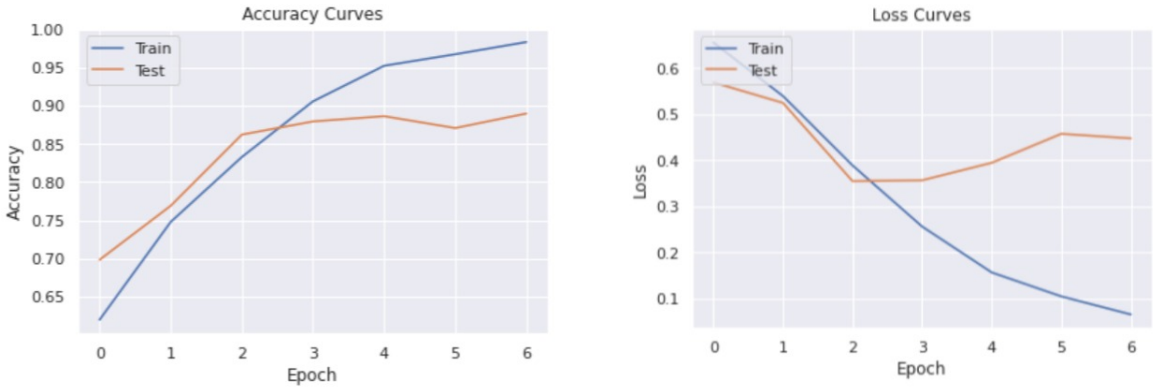

Figure 23: Accuracy Curves and Loss Curves for BERT_3.

- **RoBERTa fine tuned with one layer :**

```
Epoch 1/7
73/73 [==============================] - 75s 823ms/step - loss: 0.5788 - accuracy: 0.7030 - val_loss: 0.3992 - val_accuracy: 0.8431
Epoch 2/7
73/73 [==============================] - 59s 807ms/step - loss: 0.3756 - accuracy: 0.8427 - val_loss: 0.3044 - val_accuracy: 0.8845
Epoch 3/7
73/73 [==============================] - 60s 821ms/step - loss: 0.2021 - accuracy: 0.9207 - val_loss: 0.2571 - val_accuracy: 0.9138
Epoch 4/7
73/73 [==============================] - 60s 829ms/step - loss: 0.1340 - accuracy: 0.9539 - val_loss: 0.2755 - val_accuracy: 0.9207
Epoch 5/7
73/73 [==============================] - 61s 833ms/step - loss: 0.0862 - accuracy: 0.9716 - val_loss: 0.2842 - val_accuracy: 0.9207
Epoch 6/7
73/73 [==============================] - 61s 835ms/step - loss: 0.0322 - accuracy: 0.9901 - val_loss: 0.4145 - val_accuracy: 0.9155
Epoch 7/7
73/73 [==============================] - 61s 836ms/step - loss: 0.0506 - accuracy: 0.9853 - val_loss: 0.3334 - val_accuracy: 0.9276
```

	precision	recall	f1-score	support
Negative	0.89	0.90	0.89	192
Positive	0.95	0.94	0.95	388
accuracy			0.93	580
macro avg	0.92	0.92	0.92	580
weighted avg	0.93	0.93	0.93	580

Figure 24: Classification report for BERT_1.

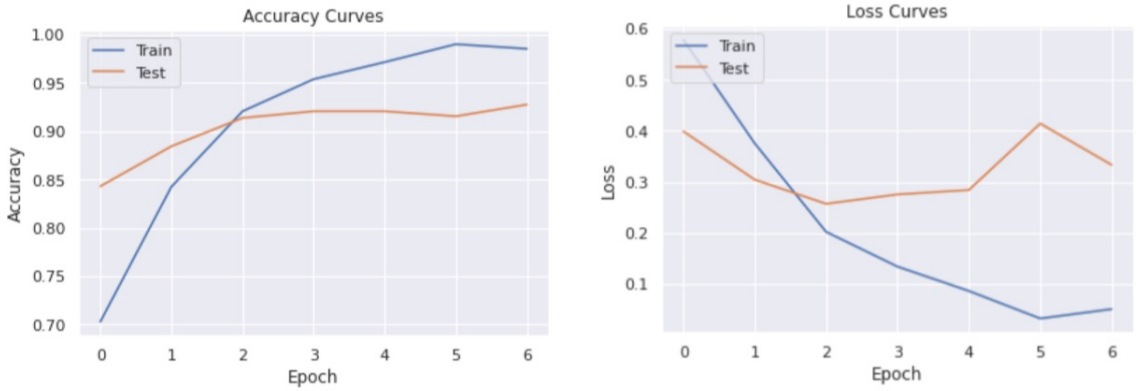

Figure 25: Accuracy Curves and Loss Curves for RoBERTa.

5 Future work and Conclusions

5.1 Future work

We estimate that there are still many things to explore in order to improve our classification system. We will describe here some tools without implementing them.

5.1.1 Dealing with unbalanced data

This method gives higher weights to the local loss with less data. That way the optimizer focus more on the class that have less samples. More formally, let's note $(\lambda_1, ..., \lambda_n)$ the weights, $(l_1, ..., l_n)$ the local losses. Then the total loss is given by

$$L(X, Y, \lambda) = \frac{1}{n} \sum_{i=1}^{n} \lambda_i l_i(x_i, y_i(W)) \tag{8}$$

5.1.2 Oversampling

This method creates literally new samples based on the observed ones. Let's say we have observed $x_1, ..., x_p$ then we define what we call the empirical mesure defined by :

$$\forall H \in B\left(R^d\right) \quad P(H) = \frac{1}{p} \sum_{i=1}^{p} 1_{x_i}(H) 1_{y_i=0}(H) \tag{9}$$

Then we use that distribution to sample from it new artificial observation for class 0.

5.2 Conclusion

We have seen how much the process of collecting the data is a serious phase in machine learning. It is very critical to construct powerful AI systems. We have seen how much the attention mechanism is helping BERT and RoBERTa to achieve state of the art in NLP tasks. It helps us to achieve an accuracy of 0.9241 without need to huge capacity of computing.

References

[Vap92] V. Vapnik. "Principles of Risk Minimization for Learning Theory". In: *Advances in Neural Information Processing Systems*. Ed. by J. Moody, S. Hanson, and R. P. Lippmann. Vol. 4. Morgan-Kaufmann, 1992. URL: https://proceedings.neurips.cc/paper/1991/file/ff4d5fbbafdf976cfdc032e3bde78de5-Paper.pdf.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[Mik+13] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[SBV15] Erwan Scornet, Gérard Biau, and Jean-Philippe Vert. "Consistency of random forests". In: *The Annals of Statistics* 43.4 (Aug. 2015). ISSN: 0090-5364. DOI: 10.1214/15-aos1321. URL: http://dx.doi.org/10.1214/15-AOS1321.

[Vas+17] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[Dev+19] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[Liu+19] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[Cla+20] Kevin Clark et al. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. 2020. arXiv: 2003.10555 [cs.CL].