COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

IEOR 4742 - Deep Learning Final Project

Market Jump Prediction

Guillaume Delorme GGD2113

Naman Goyal NG2746

Guande Wu GW2403

Xinchen Zhai XZ2791

Columbia University

May, 2020

Abstract

Using Neural Networks and 6-months data of stock moves for several companies, we try to predict jumps in the stock market. To perform this prediction we use Convolutional Neural Networks and Long-Short Term Memory Networks. We also discuss the difference between predicting jumps on a second basis or on a minute basis.

We present some resampling techniques and how they may be applied to highly imbalanced data. Our model achieved a AUC score more than 0.7 among different stocks. We also generalized the model with multi-stock combined training data, so the same model can be applied on different stocks.

Table of Contents

Introduction

"Jumps" are rare, flagrant discontinuities one can notice on financial markets. They have a significant impact on a wide range of actions such as risk management, bond and options pricing or hedging.

Being able to predict such anomalies has become an important part of recent studies and paper because being able to forecast precisely these Jumps would allow banks and hedge funds to diminish their unavoidable risk.

A lot of studies have been conducted in order to find a link between Financial Market jumps and nationwide or even worldwide news but Marcel Prokopczuk and Chardin Wese Simen show in their paper that close to 50% of jumps are not explained by fundamental news and are "excess jumps".

We are going to use Deep Neural Networks to try to predict the market jumps, without looking at the news but using features such as "Ask Price", "Bid Price" and the volumes of stock exchanged per second.

Our work is pursuing the work of another group of students from Columbia that use CNN to predict minute level jumps and we will discuss their methods and results.

Chapter 1

Data Engineering

In this section we will present the datasets we collected and used in order to predict stock market jumps, but also all the preprocessing that was necessary.

1.1 The Dataset Used

We gathered level two consolidated trade data from big companies such as Apple (AAPL), Bank of America Corporation (BAC) or JP Morgan (JPM) containing information such as bid price, ask price or volume traded per millisecond.
The got for each company two dataset, the dataset Trade gives us the value of the option and the amount exchanged at every time unit. The dataset quote gives us more precise information about the bid and ask sizes and prices.
From there we will define a jump as a brutal variation of more than twice the standard deviation of the stock price (calculated over the last day).
We understand that a jump is a very rare event and that our dataset risks to be highly imbalanced. In fact for the JPM data, over 287849 timestamps we have 29383 jumps (around 10 %).

DATE	TIME_M	SYM_ROOT	SYM_SUFFIX	SIZE	PRICE
20200102	30:00.1	JPM		2	139.74
20200102	30:00.6	JPM		148620	139.79
20200102	30:00.6	JPM		148620	139.79
20200102	30:00.7	JPM		15	139.85
20200102	30:00.7	JPM		130	139.9
20200102	30:00.7	JPM		25	139.94
20200102	30:00.7	JPM		77	139.9
20200102	30:00.7	JPM		23	139.92

Figure 1.1: Dataset Trade

DATE	TIME_M	BID	BIDSIZ	ASK	ASKSIZ	SYM_ROOT
20200102	30:00.0	139.66	5	140.01	1	JPM
20200102	30:00.1	139.73	1	145.33	2	JPM
20200102	30:00.1	139.73	1	145.33	2	JPM
20200102	30:00.6	139.5	3	140.25	1	JPM
20200102	30:00.6	139.74	1	141.45	3	JPM
20200102	30:00.6	134.07	1	145.33	1	JPM
20200102	30:00.6	139.5	3	140.08	2	JPM
20200102	30:00.6	139.79	4	139.9	2	JPM
20200102	30:00.6	134.07	2	145.33	2	JPM
20200102	30:00.6	139.79	4	139.9	2	JPM
20200102	30:00.6	139.74	1	140.25	2	JPM
20200102	30:00.7	139.79	4	139.92	3	JPM
20200102	30:00.7	139.74	1	141.45	3	JPM
20200102	30:00.7	139.74	1	140.25	2	JPM
20200102	30:00.7	139.8	1	146.25	1	JPM
20200102	30:00.7	139.8	1	140.25	2	JPM
20200102	30:00.7	133.62	1	146.25	1	JPM
20200102	30:00.7	139.74	1	141.45	3	JPM
20200102	30:00.7	139.8	1	140.25	2	JPM

Figure 1.2: Dataset Quote

1.2 Data Cleaning

We noticed a lot of ”anomalies” in these datasets, for example really low or really high bid or ask prices, meant by the traders to buy or sell stock as soon as possible, which are not representative of the market actual price.

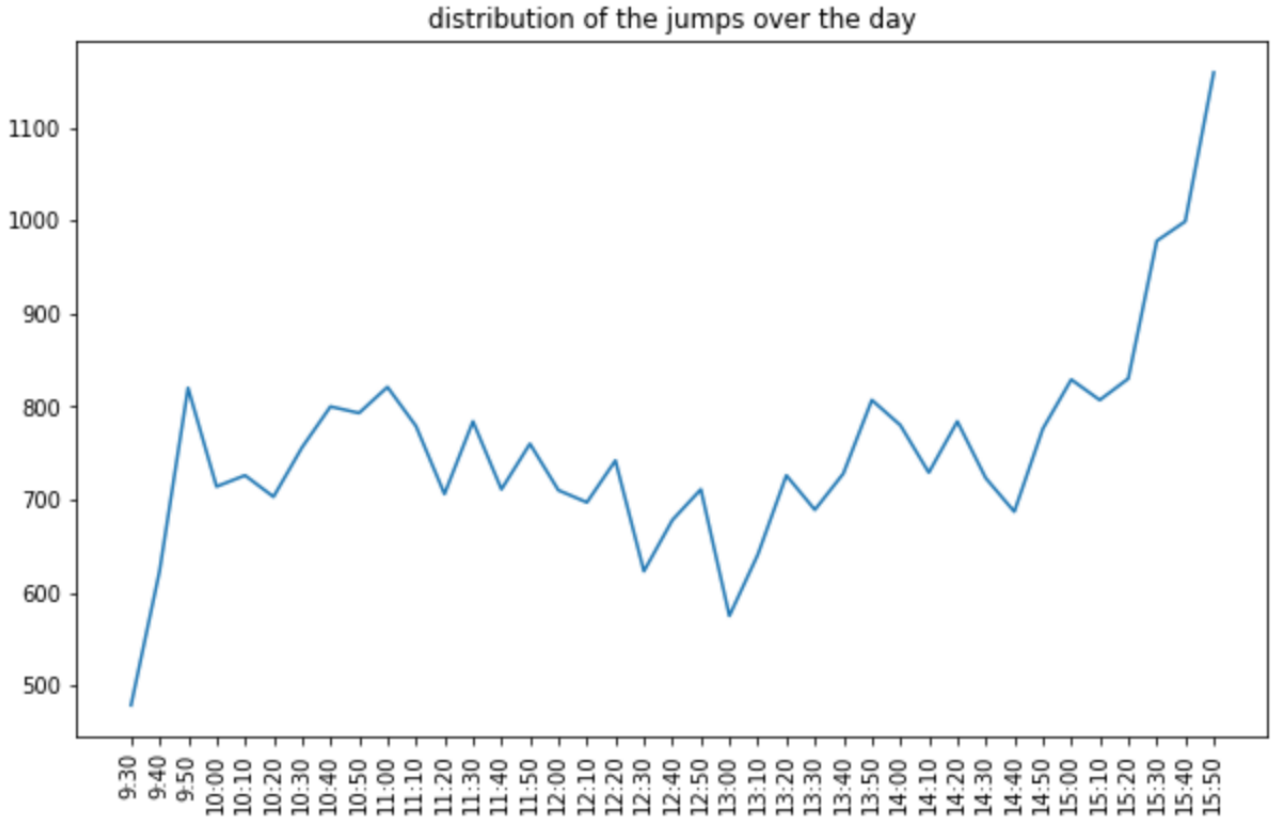

Figure 1.3: Distribution of the number of the jumps over the day

There are also time steps where the number of stock traded is too low to be representative of the market behaviour.

In the data cleaning phase, we group per second the different features we have, and we get rid of the times with too few trades.

Chapter 2

Feature Engineering

To divide the raw data into seconds-level data, we first group the raw data by DATE and TIME. Then we further group the orders by their limit prices, so we can get a list of limit prices and order volumes. As there is always some extreme outliers with extreme prices, such as a limit price of 0 or over 1000, in order to filter out these anomalies, in practice we choose to ignore all price levels with order volumes less than 10. The first two features we choose is "ask-bid-spread" and "ask-bid-ratio", as we know the order book tends to be will be highly-imbalanced just before the jump happens (Milla Makinen Et al. 2018). Then we also measured the skewness of the distribution of both ask and bid. "Ask-skewness" is measured by the difference between mean ask price and the volume weighted ask price, same for the "bid-skewness".

Then, all of the previous four features are taken the absolute value, resulting a total of 8 features. In experiment, we have tested that the absolute values of features do contribute extra information and improve the model performance.

One more feature we added is the time of the day. According to previous group's report, jumps are more likely to happen at the beginning or the end of the trading day, so we map the hour of the day to a continuous variable between 0 and 1. We suppose jumps will more likely happen when this variable is close to 0 or 1.

DATE	TIME_M	ask_bid_ratio	ask_bid_spread	time	ask_skew	bid_skew	jump	jump_lag_1
20200102	10:00:00	-0.128755	0.134638	0.076923	-0.000002	0.108434	0	0.0
	10:00:01	0.022901	0.134638	0.076966	-0.000006	0.088216	0	0.0
	10:00:02	-0.292308	0.134638	0.077009	0.000011	0.041348	0	0.0
	10:00:03	-0.503448	0.134597	0.077051	0.000000	0.050966	0	0.0
	10:00:04	-0.397938	0.097669	0.077094	-0.000014	0.085395	0	0.0
...	...	...	...	...	...	...	...	...
20200131	9:59:55	0.033195	0.010947	0.076709	-0.002951	0.003368	0	0.0
	9:59:56	-0.228916	0.000381	0.076752	0.000001	0.000081	0	0.0
	9:59:57	0.400000	0.000448	0.076795	0.000000	0.000000	0	0.0
	9:59:58	-0.051724	0.000537	0.076838	0.000005	-0.000043	0	0.0
	9:59:59	0.000000	0.000336	0.076880	-0.000010	-0.000016	0	0.0

287849 rows × 7 columns

Figure 2.1: Dataset clean with the new features

After examining this time series, we also find that jumps tend to last more than one seconds, ususally a price jump continues for 2 or three seconds, so the last feature we used is the lag_one indicator of the jump, that is weather there is a jump in the previous second.

In conclusion, we used 10 features, this is less than what the previous group did, due to the fact that seconds-level model requires 60 times more computing power, that is why we have limited the dimension at each time step. Nevertheless we believe these 10 features contain most of the information we need for the purpose of this project

Chapter 3

The models used

In order to perform the Jump detection algorithm, we used mostly two models, one Convolutional Neural Network (CNN), and one Long Short Term Memory Network (LSTM). One crucial fact is that our data is a time serie and that our goal would be to predict a jump before it arrives, in order to make money out of this knowledge. Therefore the information about the moment, or the few moments before the jump is the most important.

3.1 The CNN

We added to our dataset a column "Jump Lag" which is basically the jump column shifted. In fact the CNN will not take in account the time series so we have to predict the jump based on the moment just before the jump occurs.
We use a Network with 2 convolutional layers as shown on the summary figure 3.1.
The performance of this algorithm are shown in figure 3.2

3.2 The LSTM

We also tested the LSTM model as it is known to be a robust model for time series data. For our LSTM model, we used 2 hidden layers with 20 steps, layers are passed as se-

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_3 (Conv1D)            (None, 18, 16)            496
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 9, 16)             0
_________________________________________________________________
dropout_5 (Dropout)          (None, 9, 16)             0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 7, 16)             784
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 3, 16)             0
_________________________________________________________________
dropout_6 (Dropout)          (None, 3, 16)             0
_________________________________________________________________
flatten_2 (Flatten)          (None, 48)                0
_________________________________________________________________
dense_4 (Dense)              (None, 40)                1960
_________________________________________________________________
batch_normalization_3 (Batch (None, 40)                160
_________________________________________________________________
activation_3 (Activation)    (None, 40)                0
_________________________________________________________________
dropout_7 (Dropout)          (None, 40)                0
_________________________________________________________________
dense_5 (Dense)              (None, 20)                820
_________________________________________________________________
batch_normalization_4 (Batch (None, 20)                80
_________________________________________________________________
activation_4 (Activation)    (None, 20)                0
_________________________________________________________________
dropout_8 (Dropout)          (None, 20)                0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 21
=================================================================
Total params: 4,321
Trainable params: 4,201
Non-trainable params: 120
```

Figure 3.1: Summary of the Neural Network used

quences. In our experience, unlike the CNN model, the LSTM is not very sensitive to the hyperparameters used, a different set of hyperparameters can also result a promising performance as we can see in figure 3.3

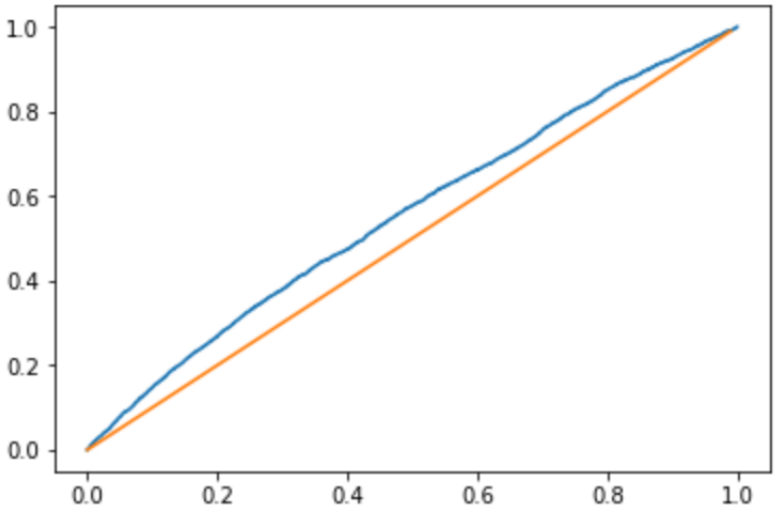

Figure 3.2: ROC curve of the CNN Model

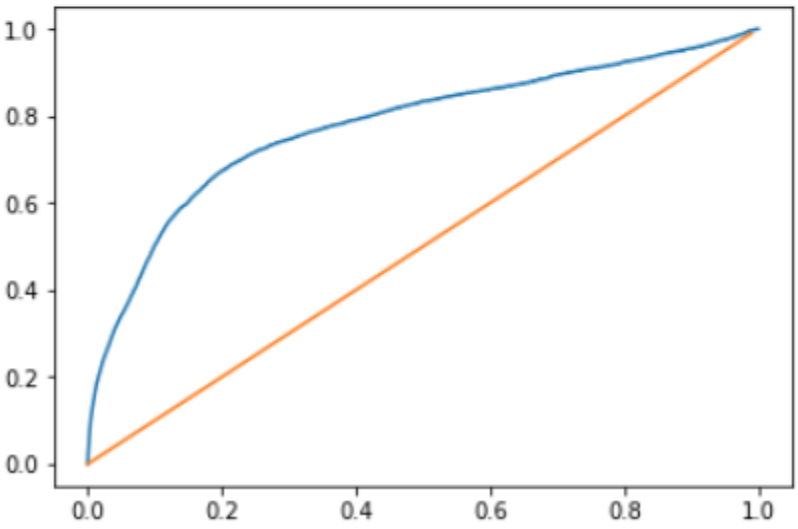

Figure 3.3: ROC curve of the LSTM Model

Chapter 4

Resampling techniques

As we mentioned before, the major issue we encounter with our data is that it is imbalanced (around 10% jumps). The issue with it is that a model predicting constantly the lack of jump will achieve a accuracy of 90% which may look good, but is absolutely useless in the goal we are trying to reach.
To overcome this problem we performed two main actions

- We changed our metrics from accuracy to AUC or ROC which give more importance to true positives and less to false negatives
- We researched and used several resampling techniques to be able to get rid of this imbalance in the dataset.

4.1 Upsampling

The easiest way to get rid of the imbalance in the data is to duplicate enough samples from the minority class (in our case the jump class) in order to have a balanced dataset, with around 50% of each class.
We tested the performances of this technique with two simple models which are Logistic Regression and Random Forest, the results can be see on figure 4.2.

```
[ ] rfc.score(X_train_up,y_train_up)

0.5991495460003209

[ ] rfc.score(X_val,y_val)

0.8342539517109606

[ ] y_pred=rfc.predict(X_val)

[ ] precision_recall_fscore_support(y_val,y_pred)

(array([0.92066377, 0.24026625]),
 array([0.89282085, 0.30582637]),
 array([0.90652857, 0.26911096]),
 array([77739,  8616]))
```

Figure 4.1: Score of the Random Forest Classifier with upsampled majority class

```
[ ] clf.score(X_train_up,y_train_up)

0.5330830479120441

[ ] y_pred=clf.predict(X_val)

[ ] precision_recall_fscore_support(y_val,y_pred)

(array([0.90311323, 0.10322138]),
 array([0.54592933, 0.47156453]),
 array([0.68049899, 0.16936929]),
 array([77739,  8616]))
```

Figure 4.2: Score of the Logistic Regression with upsampled majority class

4.2 Downsampling

Another solution we found is to downsample the majority class, i.e. for each sample of the minority class, we will take a sample from the majority class, without replacement. The major drawback of this model is that it causes the loss of a lot of information that may be contained in the remaining samples from the majority class.

The results using Random Forest and Logistic Regression can be see on figure 4.4.

```
[ ] rfc.score(X_train_down,y_train_down)

0.5959624184543538

[ ] rfc.score(X_val,y_val)

0.8805512130160384

[ ] y_pred=rfc.predict(X_val)

[ ] precision_recall_fscore_support(y_val,y_pred)

(array([0.92028624, 0.36171252]),
 array([0.949562  , 0.25789229]),
 array([0.93469494, 0.30110441]),
 array([77739,  8616]))
```

Figure 4.3: Score of the Random Forest Classifier with downsampled majority class

```
clf.score(X_train_down,y_train_down)

0.5330830479120441

y_pred=clf.predict(X_val)

precision_recall_fscore_support(y_val,y_pred)

(array([0.90311323, 0.10322138]),
 array([0.54592933, 0.47156453]),
 array([0.68049899, 0.16936929]),
 array([77739,  8616]))
```

Figure 4.4: Score of the Logistic Regression with downsampled majority class

4.3 Synthetic Minority Upsampling Technique (SMOT)

The last solution we found to reduce the imbalance between the classes is a hybrid solution called Sy thetic Minority Upsampling Technique (SMOT) which consist in creating new (synthetic) samples from the minority class by using methods such as nearest neighbors.

The results using Random Forest and Logistic Regression can be see on figure 4.6.

```
rfc.score(X_train_SMOT,y_train_SMOT)
```

```
0.9032848009963975
```

```
rfc.score(X_val,y_val)
```

```
0.8983382548781194
```

```
y_pred=rfc.predict(X_val)
```

```
precision_recall_fscore_support(y_val,y_pred)
```

```
(array([0.90284139, 0.3931848 ]),
 array([0.99404417, 0.03481894]),
 array([0.94625025, 0.0639727 ]),
 array([77739,  8616]))
```

Figure 4.5: Score of the Random Forest Classifier with SMOT

```
clf.score(X_train_SMOT,y_train_SMO
```

```
0.8152644316040897
```

```
y_pred=clf.predict(X_val)
```

```
precision_recall_fscore_support(y_
```

```
/usr/local/lib/python3.6/dist-pack
  _warn_prf(average, modifier, msg
(array([0.90022581, 0.        ]),
 array([1., 0.]),
 array([0.94749351, 0.        ]),
 array([77739,  8616]))
```

Figure 4.6: Score of the Logistic Regression with SMOT

Chapter 5

Generalization

During model tuning and testing, eventually we start to realize that the pattern of intra-day jumps is shared among different stocks. One model trained with a specific stock can also be applied on another stock. Furthermore, stocks with higher beta and/or volatility have more Jumps happening every day and thus constitute a better source of training data. Our experiments showed that models trained with high-beta stocks have an even better performance on low-beta stocks testing set, than the models trained with the same low- beta stocks.

So, in order to create a 'mega-dataset', we selected 7 high-beta, volatile stocks, that is 'BAC', 'BA', 'C', 'COF', 'COP', 'MS', 'OXY'. we downloaded the order book history data of these stocks for three month (09/19.10/19,11/19). Then we trained our LSTM model with this 'mega-dataset', and tested it on three out-sample stocks('T','MMM,'KO'), with a different time period (12/19). This time the model still achieved a very high AUC score of 78%, which proved our hypothesis. Note that the model has never been trained with data from 'AAPL' or 'JPM', but still recognizes the intra-day jump patterns. We suppose, given enough computing power, using all high-beta stocks and most up-to-date data may constitute the best source of training set for the objective of this project.

Conclusion

In order to perform the jump prediction algorithm, we gathered millisecond level data and after characterizing each stock movement as a jump or not, we applied two well known deep learning models : namely a CNN and a LSTM model.

Looking for solutions to increase our prediction we trained this models on ”improved” data we got by using resampling techniques such as upsampling and downsampling.

Next improvements could be done by:

- Trying new models to improve the results
- Trying different resampling methods
- Use more features with a computer powerfull enough

References

- Prokopczuk, Marcel and Wese Simen, Chardin, What Makes the Market Jump? (June 13, 2014) Available Here
- Mäkinen, Ymir, et al. "Forecasting Jump Arrivals in Stock Prices: New Attention-Based Network Architecture Using Limit Order Book Data." Quantitative Finance
- Medium article on SMOT available Here