Data driven approach to predicting stock index movement and prices using S&P 500 Data (Category: Finance & Commerce)

Saksham Gakhar (06190199), sakshamg@stanford.edu
Lei Fang (06037833), lfang2@stanford.edu
Joseph Ballouz (06037006), jballouz@stanford.edu

1 Motivation

This is an application project. Our goal was to apply supervised and unsupervised machine learning methods to data available for the financial market (aiming companies and trusts) and see if we can make predictions about the direction of movement of stock prices and use that information for trading assets. As we went along, we realized that we could also extract useful information about the current state of the financial market (bull/sideways/breakout/panic zone differentiation) that, besides our trading model predictions, may also be of essence for estimating market returns. In section 2 we describe our dataset, in section 3 we visit the basics of the 4 main training models we tried on some of the companies and in section 4 we describe corresponding results.

2 Data and Features

Our time series data for all companies for each day is drawn from the S&P500 database which provides stock information for 500 companies tracked for 40+ years. The raw data features include: company ID, highest and lowest values of stocks that day, opening and closing values of the stocks that day, and the volume of stocks traded that day. All algorithms other than GMM-SVC hybrid method (section 3.2, 4.2) use these publicly available features. The GMM-SVC method uses some derived features that will be described in section 3.2. We recognize that these raw and derived features only form a small subset of the actual variables that affect the direction of index movement and its quantity and values. Other features characterizing social media, natural calamities or governmental policy influences (etc.) are not available. We intend to continue and refine this work with J.P. Morgan Chase & Co. as they are willing to provide us with more internal data.

3 Methods

3.1 Logistic Regression & Support Vector Classification (SVC) for Trading

Logistic regression and SVC are used for predicting whether stock prices will increase or decrease. The models are standard (scikit-learn is used) and objective functions are described in equations 1 and 2. For logistic reg.:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{m} log\big(1 + exp(-y^{(i)}(X_i^T w + c))\big) \tag{1}$$

For Support Vector Classification (SVC)

$$\begin{aligned} \min_{\gamma,w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{m} \xi_i \\ \text{s.t. } y^{(i)}(w^T y^{(i)} + b) \geq 1 - \xi_i, i = 1, 2, 3...m \\ \xi_i \geq 0, i = 1, 2, 3...m \end{aligned} \tag{2}$$

where symbols have usual meaning as defined in the lecture notes. For labelling the data, we take average of each raw feature to get the weekly averaged feature (weekly mean high price, weekly mean low price, weekly mean open price and weekly mean volume). If the closing price of the stock on following Friday is smaller than closing price at current Friday, then the label for current week is -1, else $+1$. We added L_2 penalty in the logistic regression. For Support Vector Classifier, we found that after fine tuning, C=1000 (regularization parameter).

Note on using weekly averaged features: Due to heavy fluctuations on a daily basis in the data, we hypothesized that the most useful data are the data from the previous k weeks. We assumed so because of very high degree of variability of the trends on a daily basis. We calculated the mean prediction errors based on different amount (k averaged weeks) of data points and concluded from figs. 1(a),(b) that $k = 20$ **weeks** gives **lowest prediction error (32%)** for logistic regression **and <50%** for Support vector classifier (rbf kernel).

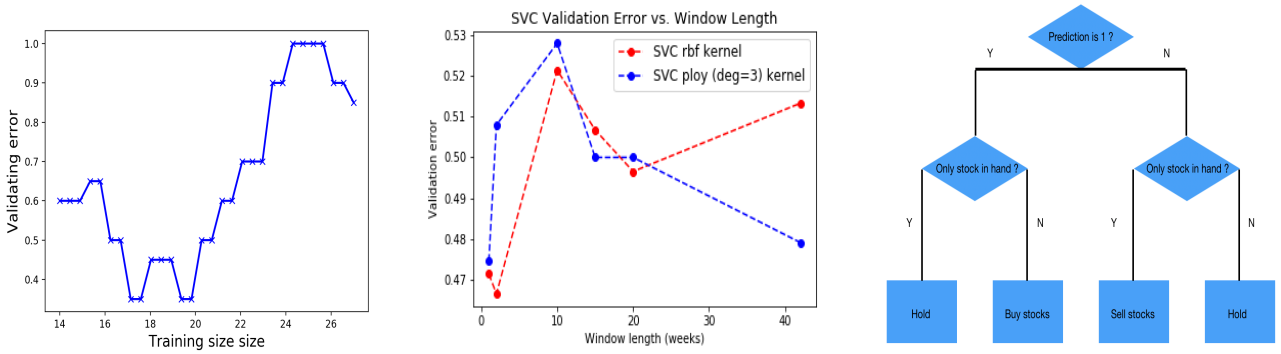


Figure 1: Error Analysis for (a) logistic regression; (b) SVC with 2 kernels (rbf is used finally);(c) Flow chart of trading decision.

3.2 Clustering (GMM) & SVC for Predicting Market Returns

Unsupervised (Gaussian Mixture Model): The aim in this section was to learn from the algorithm the inherent stage in market at different times such as bull, sideways, breakout and panic zones. This is achieved through an unsupervised Gaussian Mixture Model by assuming a mixture of 4 gaussians [1]. We used the inbuilt function in scikit-learn to achieve this. Preliminary result for the division of the market into 4 different zones is shown in fig.2. The y-axis is the cumulative log market returns. Log market returns ($log\frac{\text{opening price[i]}}{\text{opening price[i-1]}}$) are often used in financial literature in place of actual monetary returns because it can be shown [2] that the value of the cumulative log market return at a given time is the difference between the log of the initial investment and the log of the final value of investment at that time. Input features are opening, highest, lowest and closing prices.

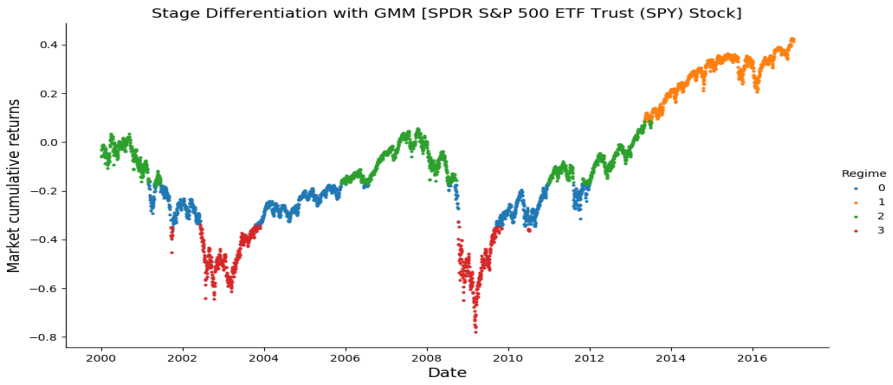


Figure 2: Stage differentiation into 4 market stages using GMM

The characteristics of these stages are listed in the table below.

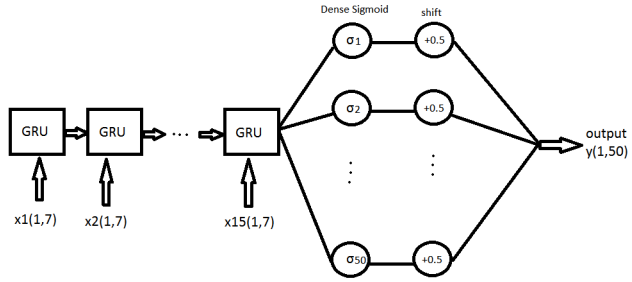
Stage	0	1	2	3
Color	Blue (Sideways or low volatility zone)	Orange (breakout zone)	Green	Red (panic zone or high volatility)
Characteristic	Moderate mean, Low Co-var	High mean, High Co-var.	High mean, Moderate Co-var	Low mean, Low Co-var
Mean ($)	115.44	197.24	138.70	92.01
Co-variance ($\$^2$)	29.91	251.15	75.02	57.47

Unsupervised (GMM) with Supervised (SVC): Similar stage differentiation (unsupervised GMM as above) was carried out with more derived features which include RSI (relative strength index), SMA (smooth moving average with window = 20 weeks), rolling window correlation (window = 20 weeks), parabolic SAR and

log-return-ratio as before of open to open price based on the data from previous days (i.e. for all features, only data till [i-1]th day was used for a given day[i]). The resulting stages from stage differentiation states {0, 1, 2, 3} were used as additional features (besides these derived ones listed above) for developing a Support Vector Classifier (supervised) which then gives a stage dependent prediction. The predicted variable was +1 for day [i] if the return of day [i] was > return of day [i-1] and −1 otherwise. We took an 80-20 split for train-test data for supervised SVC. The results are presented in section 4.2 (this idea is borrowed but modified for optimization from ref.[1]).

3.3 Recurrent Neural Network (RNN) – Architecture

Here we are predicting a time series (ratio for day[i] = closing value[i]/closing value[$i-1$]); the most suited Neural Network is a Recurrent Neural Network. The model will take in Tx consecutive time steps (x_1,..,x_{T_x}) each of length *chunk_size_x* and return one output of length *chunk_size_y*. This architecture is referred to as a many-to-one architecture since we are taking many x's and returning one output. after many trials and errors, we found that it is reasonable to take $T_x = 15$, chunk_size_x = 7, and *chunk_size_y* = 50, i.e the model takes in 15 weeks (1 week is 7 days in this case) and tries to predict the next 50 days. We tried different types of RNN cells using the Keras library: simple RNN cell, GRU (Gated Recurrent Unit), and LSTM (Long short-term memory unit). Both GRU and LSTM are designed to capture long term trends in the data, but we found that GRU performs better. In our model we use one layer of $T_x = 15$ GRU units (with standard tanh activation). We found that stacking additional GRU layers did not help the predictions. This GRU layer serves to capture any general upward or downward trend in the data. To capture the noisy texture of predictions, we pass the output of GRU layer to a densely connected sigmoid layer. Also, no dropout layers were added since they also have the effect of smoothing the data & eliminating short term variations. Since the average sigmoid value is 0.5 (sigmoid outputs are between 0 and 1), we pass our sigmoid predictions through a"shifting" layer which adds 0.5 to our results. Thus, if sigmoid layer predicts 0.6 > 0.5 (meaning the close value increased from the previous day) the final layer will predict 1.1 i.e a 10% increase from the previous day. Fig. 3 summarizes our model architecture.



Layer (type)	Output Shape	Param #
gru_53 (GRU)	(None, 14)	924
dense_213 (Dense)	(None, 50)	750
dense_214 (Dense)	(None, 50)	2550

Total params: 4,224
Trainable params: 1,674
Non-trainable params: 2,550

Figure 3: (a) RNN Architecture; (b)Keras model summary

3.4 Cluster-regression Hybrid Model

We speculate that if we use logistic regression to classify data, we are separating whole training data set with single hyper-plane, which may not work good for a complex feature space such as ours. We can see this in fig. 8(a), the data points are grouped as clusters in the feature spaces (the actual feature space are higher than this; this is just a illustration). So we run logistic regression in each cluster with ideas of prediction and trading similar to sec. 3.1, 4.1.

4 Results and Discussion

4.1 Logistic regression and Support Vector Classification for Trading

A trading model to guide investments

To show that the 20 weeks window method described in section 3.1 works, we start a simulation of stock trading (beginning in the week of July 8^{th} 2013) (the flow chart is in fig. 1(c)). We pretend we are in 2013 and we get new data points as time goes. When we trade, we trade with the current stock price. Each week we train a new model based on latest 20 weeks' data. We start with 5000 stocks in hand (this is the initial funds in hand). This is "buy everything, sell everything model". To simplify things, we make trading decision once a week and if we do in fact trade (either buy or sell), we trade at the end of Friday. Furthermore, the trading model assumes that if we buy, we spend all money to buy; if we sell, we sell all the stocks in hand. On Friday before close, we train the model over the past 20 weeks of data and have a prediction of whether stock price will increase or decrease next week (+1 or −1 respectively). Investment results for techniques described in section 3.1 are shown in red, green and blue markers in fig. 4.

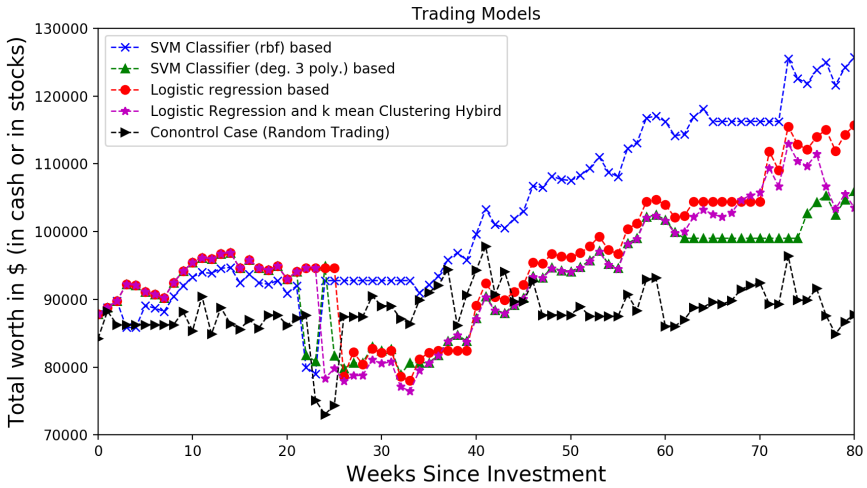


Figure 4: Figure representing the total in-hand value (in US dollars) after investments based on the trading models. The 4 models are in close agreement. There seems a bias as the time progresses. Random trading shows random trading based on a coin flip.

4.2 Clustering (GMM) and SVC for Predicting Market Returns

Here are the results for the discussion in section 3.2. We defined the *strategy return* for day [i] as prediction[i]$*log\frac{\text{opening price day [i-1]}}{\text{opening price day [i-2]}}$. These are then compared with actual *market returns* which is $log\frac{\text{opening price day [i]}}{\text{opening price day [i-1]}}$. The result for the test set is presented in fig. 5. Parameters were optimized over a wide range using the `GridSearchCV` function in scikit-learn using 3 different scoring methods – precision, recall and AUC. Finally radial basis function (gaussian) kernel with C=10 was used (although sensitivity to C is not much). It may seem that the results over the test set presented in fig. 5 are over-fitting for this SPY stock, which is true; but the goal for this exercise was to be able to make excellent predictions for 'this' particular stock (SPY). So if we don't worry about using the same set of optimized parameters for other companies, the agreement in fig. 5 between strategy and market returns for the test set is a very good practical result.

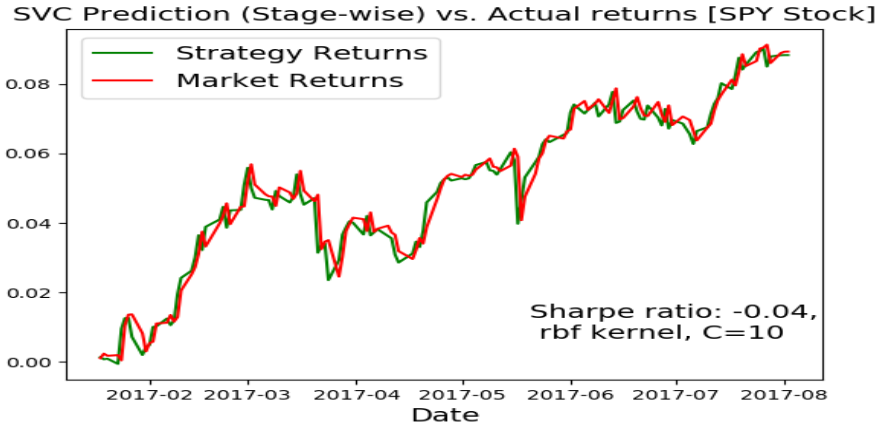


Figure 5: Result of stage specific SVC on test data for the SPY stock based on 80-20 train test data split.

4.3 Recurrent Neural Network

For IBM, our training data consists of daily closing values from 1962 to 2015. That amounts to 14,295 days which lead to 2020 training examples of inputs of size $(15, 7)$ and outputs of size $(50,)$. the input will be $X = (2020, 25, 7)$ and the output will be $Y = (2020, 50)$. We use a batch size of 202 for a total of 10 batches per training epoch. The loss function is the mean squared error between the model output and the actual output.

Hyper-parameter tuning: We performed hyper-parameter tuning for the learning rate and the optimizer used (we tried 3 that worked well) by training for 10 epochs. The table below summarizes our results, using loss and mean absolute percentage error ('mape' in Keras) as evaluation metrics. We trained our model using stochastic gradient descent (SGD, since it kept a healthy amount of variation in the predictions) for 20 epochs.

learning rate	1	0.1	0.01	0.001
Adam	na	loss: 2.4745e-04 mape: 1.1004	loss: 2.4697e-04 mape: 1.0991	loss: 7.8336e-04 mape: 1.9857
Adadelta	loss: 0.0011 mape: 2.6626	loss: 0.0091 mape: 7.9337	loss: 0.0066 mape: 6.31740	na
SGD	loss: 0.0010 mape: 2.4797	loss: 0.0045 mape: 5.55897	loss: 0.0049 mape: 5.6400	na

Predictions: For ratio predictions (refer beginning of sec. 3.3) in year 2016 for IBM (see fig. 7), the mean square error (mse) was 0.00033 and the mean absolute percentage error (mape) was 1.40. Given that our training data had similar mape of around 1, we see that the variance in our prediction for the year 2016 is low but there is always that bias of mape around 1, which we found to be unavoidable. For the close value predictions in year 2016, mse was 98.2 and mape was 5.65.

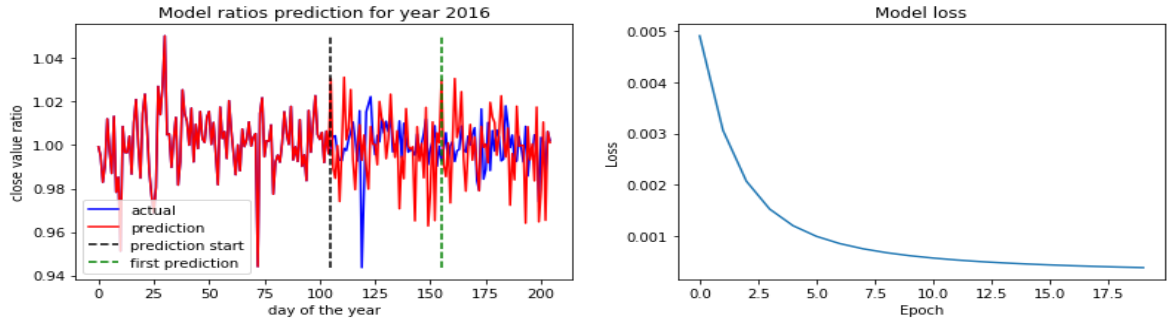


Figure 6: (a) RNN ratio predictions for IBM; (b) corresponding model loss for Adam optimizer

In the trading model, we buy 1 stock if the price (of the prediction) is increasing and sells it as soon as it starts decreasing. The black 'don't buy' line (fig. 7(b)) represents the profit of not buying the stock in the first place. The blue 'buy and keep' line represents the profit from buying the stock on day 1 of the year and selling it on the final day. IBM in the year 2016, amasses profits in the first prediction period (first 50 days) then tapers off, but still performs better that the two baseline trading schemes (profit of $6 for trading 1 stock of IBM).

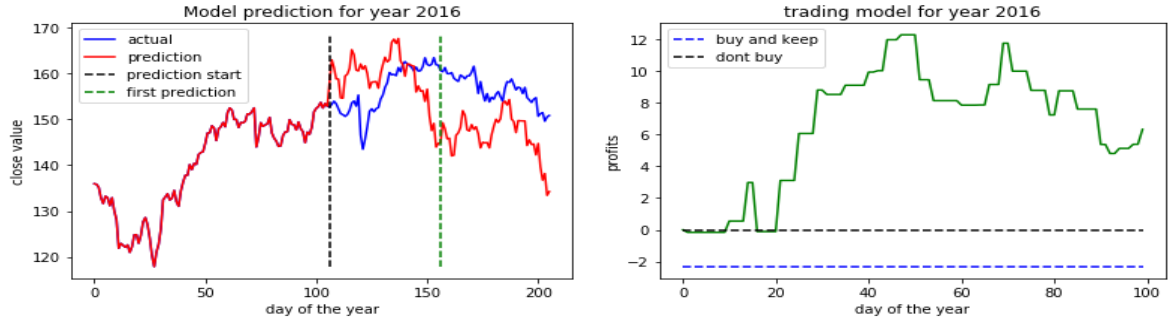
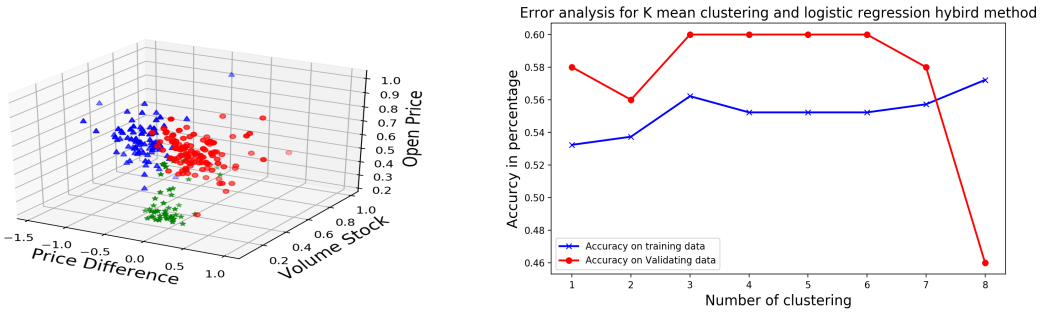


Figure 7: (a) RNN predicted actual closing values derived from ratios in fig. 6(a); (b) trading for IBM

4.4 Cluster-regression Hybrid Model

As we see in fig. 8 (b), if we choose the number of clusters to be $3 \sim 6$ clusters, the accuracy is the highest (80-20 split between training and test set). We also apply this model to our trading model described in sec. 4.1. The result of applying this hybrid model to the trading strategy can be seen in fig. 1(c) purple marker.



s

Figure 8: (a) One of the three dimensional feature space with 3 clusters.; (b) Searching best number-of-clusters

5 Contributions and Link to Code

The team worked together for the most part of the project. We collected the raw data together and Joseph Ballouz compiled it. Saksham Gakhar has worked on support vector classification tasks for the trading model. Saksham Gakhar also worked on clustering with GMM and SVC for predicton of market returns. Joseph Ballouz has worked on Neural networks and on linearly weighted regression (in milestone). Lei Fang has worked on logistic regression and clustering-regression hybrid model. Contributions to trading models and their performance were made by both by all 3 members. All three members worked on writing this document. Report compilation was done by Saksham Gakhar.
The codes used for the final work are available at the following link: `https://drive.google.com/drive/folders/1fcUY9RKNZAaLdfwFlq2CZW-v7SbUfUHF?usp=sharing`. Relevant comments are enclosed in the code. Anyone at Stanford University can view these files. Although we have checked, if there are any problems with the permissions, please contact sakshamg@stanford.edu

References

[1] "Stage differentiation and SVM." `https://www.quantinsti.com/blog/trading-using-machine-learning-python-svm-support-vector-machine`. Accessed: December 2018.

[2] "Why Log returns." `https://quantivity.wordpress.com/2011/02/21/why-log-returns/`. Accessed: December 2018.

[3] "Stage differentiation in financial markets." `http://www.nextbigtrade.com/stage-analysis/`. Accessed: December 2018.

[4] F. Pérez and B. E. Granger, "Ipython: A system for interactive scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 21–29, 2007.

[5] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[6] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference* (S. van der Walt and J. Millman, eds.), pp. 51 – 56, 2010.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[8] F. Chollet *et al.*, "Keras: The python deep learning library," *Astrophysics Source Code Library*, 2018.

[9] "Other python libraries." TALib, $\text{fix}_y ahoo_f inance, pandas_d atareader$.