# Machine Learning Models In Stock Market

Prediction of Stock Market Movement
and Discovery of Different
States in the Stock Market

Bill Huang Charles Cui Jingtian Wang

May 2017

**Abstract**

In this paper we will introduce binary classification methods (KNN, Naive Bayes, and Gaussian Process Classification) in predicting stock index movement (up or down) by using the headline of daily top 25 news from Reddit WorldNews Channel. We will also investigate the number of different states of the stock market (bull, bear, etc.) by using the Hierarchical Dirichlet Process Hidden Markov Model (HDP-HMM).

## 1 Introduction

Short-term stock market movement is difficult to predict yet meaningful for many individual investors and financial institutions. Investors' thoughts and decisions are influenced by their emotions, which to some extent is related to what is happening in the world meanwhile. Therefore, daily news could be a reasonable indicator of stock market movement. On Kaggle, Aaron7sun started a competition on "Daily News for Stock Market Prediction", which he gathered the headline of the top 25 daily news (in English) from Reddit WorldNews Channel to predict Dow Jones Industrial Average (DJIA) index's movement. There are two possible movements: up if the adjusted close is higher than the opening price, down if otherwise. The headline data are sentences, so before doing classification, we had to break the sentences into words and use whether the word occurs in the headline as the feature in our classification algorithm. We will discuss in more details about the data preprocessing in Section 2. The algorithm we chose for the binary classification are K-nearest neighbors (KNN), Naive Bayes, and Gaussian Process Classification. It turned out that all algorithms were only slightly better than random guessing, so we decided to apply ensemble learning with the three algorithms mentioned above to

improve our classification performance. Details and results of the binary classification will also be presented in Section 2.

Investors are not only concerned with the daily movement, but are also interested in the market trend for a longer period (a week for short-term investors, possibly years or 10 years for long-term investors). Bull market, a period in which the stock index consistently grow, and the Bear market, a period in which the stock index consistently fall, are often discussed among the investors. Including the normal fluctuation stage, one question might be interested to ask: are there only three stages (Bull, Bear, and Normal) in the stock market? Daily index movement is the only thing we can observe, but we can think the stages as hidden, which makes the Hidden Markov Model a perfect algorithm for learning the stages. Moreover, instead of fixing the hidden state to be 3, we decided to also learn about the number of possible hidden states. It was possible that there were also multiple stages for the Bull market due to the differences in magnitude. To do so, we implemented the Hierarchical Dirichlet Processes Hidden Markov Model [3], and the data we used here are the DJIA index movement from 1987 up to May 11, 2017 collected from Yahoo. More details and the results will be presented in Section 3.

Last, we will summarize and draw conclusion in Section 4.

# 2 Binary Classification

### Data Preprocessing

The daily news headline data set consists of string of sentences. One common way to handle the data is to break up all the sentences into words and then count the occurrences of each word for every day in our data set. This is known as the Bag of Words algorithm. For our data set, this left us around 5,000 distinct words. Since we had data on 1989 days of data in total, then our feature matrix will be 1989 by 5000+, and this will be computational expensive when using for classification. Also, for common English stopwords, such as "a" and "the", occurs in almost every Enligsh sentence but do not have a specific meaning. Including them is not useful in our classification algorithm. Moreover, this breaking procedure does not take the two-word phrase that might be important to the financial world, such as "JP Morgan", "Goldman Sachs", "Donald Trump" and many others, into account. To deal with the above issue, we also did the following in our data preprocessing step:

- First, we replaced the a set of two-words or three-words phrase that might be important to the financial market by a one-word phrase before breaking up the sentence into words. For example, we will replace "JP Morgan" to "jpmorgan".
- Second, we broke up the sentences into words and saved all the unique words. Then,

we removed all the stopwords provided by Python NLTK package from the set of unique words.

- Third, we stemmed the words left over from the second step so we can further reduce the number of words in the bags. For example, by stemming, "believe", "believes", and "believed" will all reduce to "believ".
- Last, we counted the number of occurrences for each word after stemming over the 1989 days and we removed all the words that occur less than 20 times, leaving us 3630 words in the bag.

By using the 3630 words, we can construct our feature matrix $X$, where $x_{i,j} = 1$ if word $j$ occurs in the headline of daily news for day $i$ and $x_{i,j} = 0$ if word $j$ does not occur in the headline of daily news for day $i$. Hence, our feature matrix is 1989 by 3630 with each entry being either 0 or 1. Our label vector $t$ is a vector with 1989 elements, where $t_i = 1$ if the adjusted closed price is higher than opening price for day $i$ and $t_i = 0$ if the opposite. With the feature matrix and label vector, we can divide them into two sets, with two third of the data being the training set and the rest one third being the test set. Moreover, we let the proportion of label "0" (stock price moves down) days be the same for both the training set and test set, which is 46.5%. This means that if we classify everything as label 1, our missclassification rate will be 46.5%, and this will be the value we compare to in evaluating our classification algorithm.

**Cosine Distance Function**

For both KNN and Gaussian Process Classification, we need a way to compute the distance between any two feature vectors. Since each entry of the feature vector is binary, Manhattan distance might be useful, but this will give us the distance range from 0 to 3630, which is difficult for choosing good parameters for kernels in KNN and Gaussian Process Classification. Here, we were concerning the similarity between any two feature vectors, and the cosine similarity (distance) function will be useful in this case. By the geometric definition of the dot product, for any two vectors $a$ and $b$, their dot product can also computed by

$$a \cdot b = \|a\|\|b\| \cos\theta$$

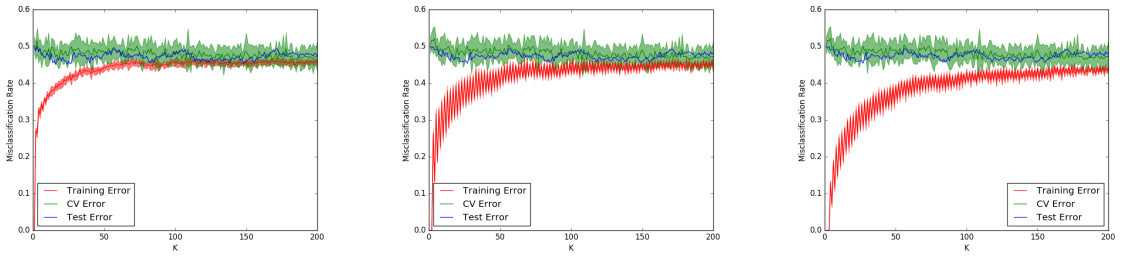
where $\theta$ is the angle between the two vectors $a$ and $b$. Thus, the cosine distance between any two vectors $a$ and $b$ is defined as

$$\theta = \arccos\left(\frac{a \cdot b}{\|a\|\|b\|}\right)$$

which is to measure the angle between the two vectors. If two vectors are similar, especially in the case of binary entry vectors, the angle between them should be small. Moreover, $\theta$ only ranges from 0 to $\pi/2$ for the distances between any two feature vectors

| Label | Distance | No Kernel | Exponential Kernel | Cauchy Kernel |
|---|---|---|---|---|
| 1 | 1 | 1 | 0.368 | 0.5 |
| 0 | 4 | 1 | 0.018 | 0.25 |
| 0 | 9 | 1 | 0.0001 | 0.1 |

Table 1: Weight (on the right hand side) calculated by applying kernel to distance for classifying a data point by its 3 nearest neighbors.




(a) No Kernel (b) Exponential Kernel (c) Cauchy Kernel

Figure 1: Training, validation, and test errors for various value of $k$ and different kernels.

in our data set, so we could better pick the parameters for the kernels.

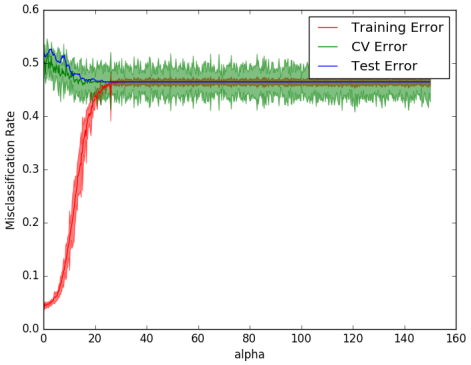
**KNN**

The first binary classification algorithm we used is the k-nearest neighbor (KNN) algorithm. The standard KNN gives each nearest neighbors equal weight in the classifying step, as shown in the "No Kernel" column in Table 1. However, assigning equal weight to the nearest neighbor regardless of its distance to the neighbors might not be efficient in many cases. We generally considered the closet neighbor should have bigger influence in the classifying step. We could achieve the above property by introducing kernel into the KNN algorithm. One is the exponential kernel, which is defined as

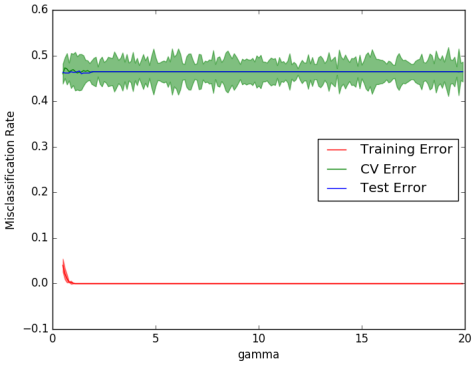
$$w_{i,j} = e^{-d_{i,j}}$$

where $w_{i,j}$ is the weight of nearest neighbor $j$ in classifying the data point $i$ and $d_{i,j}$ is the distance of their feature vector computed by cosine distance function introduced above. Column "Exponential Kernel" in Table 1 shows the weight of the 3 nearest neighbor computed by the exponential kernel. One drawback of the exponential kernel is that the weight for nearest neighbor that is far away is very small, which gives great weight for neighbor that is nearby. However, our distance function gives us distance between 0 to $\pi/2$, so the smallest weight will still be 0.16 ($e^{\pi/2}$) and it still have some significant weight on the classifying step. Another kernel we used is the Cauchy kernel, which is defined by

$$w_{i,j} = \frac{1}{1 + d_{i,j}}$$




(a) Training, validation, and test errors for various value of $\alpha$ in Naive Bayes classification.




(b) Training, validation, and test errors for various value of $\gamma$ in Gaussian Process Classification.

Figure 2: Results for Naive Bayes and Gaussian Process Classification.

Column "Cauchy Kernel" in Table 1 shows the weight of the 3 nearest neighbor computed by the Cauchy kernel. Cauchy kernel provides smoother decay on the weight as the distance increases. From Table 1, we will classify the data point as 0 if we are not using any kernel (weight of label 1, $1 < 2$, weight of label 2). However, for both exponential and Cauchy kernel, we will classify the point as label 1 after we add up the weight.

Figure 1 shows the training, validation, and test errors for different kernels. For exponential kernel, the cross validation errors are above 0.465 for all $k$, which is worse by guessing all 1s. For Cauchy kernel, the cross validation errors are below 0.465 for only a few value of $k$, and this is the same case for the standard KNN algorithm (no Kernel). Different kernels do not have a advantage in improving our classification process and KNN in general perform very poorly for our classification problem.

### Naive Bayes

The second classification algorithm we used is the Naive Bayes algorithm. Different from the KNN, Naive Bayes is a probabilistic classifier, which we will classify the data point as label $t$ by computing the following

$$p(t|x_n) \propto p(t)p(x_n|t) = p(t)\prod_{d=1}^{D} p(x_{nd}|t)$$

where $x_n$ is the feature vector for test point $n$, $p(t)$ and $p(x_{nd}|t)$ can be calculated from the training set by using MLE and Bayesian method. Figure 2a shows the training, validation, and test errors for different value of $\alpha$ (the prior weight to ensure that the probabilities of assigning to a certain label will not be zero if some words are not observed). The MLE is the case when $\alpha = 0$. We can see that as $\alpha$ gets large, all training, validation, and

test errors converges to 0.465 since the prior $p(t)$ had the major impact on classifying the points when $\alpha$ is large. Since there are more 1s than 0s in the train set label, the Naive Bayes classified all test points as label 1, resulted in the misclassification rate of 0.465 as discussed above. For $\alpha$ in range between 15 and 16, the validation error is around 0.460, which is slightly better than guessing all 1s. However, Naive Bayes is still weak for our classification problem.

**Gaussian Process Classification**

Different from Naive Bayes that is a parametric model, in our third classification algorithm, we used a non-parametric model, Gaussian Process Classification. Here, we used a logistic function to convert the value we got from Gaussian Process into a value of between 0 and 1 so we can interpret it as a probability, which

$$p(t_n = 1|f_n) = \frac{1}{1 + \exp(-f_n)}$$

where $f_n$ is computed from Gaussian Process. Here, we used a point estimate approximation, so we need to first find the MAP value for $f$ of the train set data points, where is to the find the $\widehat{f}$ that maximize $p(f|X, t)$ where $X$ is the feature matrix and $t$ is label vector for the train set. Recall that in Gaussian Process, $f \sim \mathcal{N}(0, C(X))$ where $C$ is covariance matrix. This leaves us to solve the following problem [2],

$$\begin{aligned}
\widehat{f} &= \arg\max_f \left\{ \log p(t|f) + \log p(f|X) \right\} \\
&= \arg\max \left\{ \sum_n \log \left( P_n^{t_n} (1 - P_n)^{1-t_n} \right) - \frac{1}{2} f^T C^{-1} f \right\} \\
&= \arg\max g(f)
\end{aligned}$$

where $P_n = p(t_n = 1|f_n)$, the probability computed by the logistic regression. It is difficult to get the analytical solution, so we went with the Newton-Raphson method, which we used the following update rule until convergence:

$$f' = f - \left( \frac{\partial^2 g(f)}{\partial f \partial f^T} \right)^{-1} \frac{\partial g(f)}{\partial f}$$

The gradient can be computed by differentiate $g(f)$ with respect to $f$, which

$$\frac{\partial g(f)}{\partial f} = (t - P) - C^{-1} f$$

and the Hessian is

$$\frac{\partial^2 g(f)}{\partial f \partial f^T} = -P - C^{-1}$$

| | Data 1 | Data 2 | Data 3 | Error |
|---|---|---|---|---|
| Truth Label | 1 | 1 | 0 | |
| Model 1 | 1 | 1 | 1 | 33.3% |
| Model 2 | 0 | 1 | 0 | 33.3% |
| Model 3 | 1 | 0 | 0 | 33.3% |
| Boosting | 1 | 1 | 0 | 0% |

| | Data 1 | Data 2 | Data 3 | Error |
|---|---|---|---|---|
| Truth Label | 1 | 1 | 0 | |
| Model 1 | 1 | 0 | 1 | 33.3% |
| Model 2 | 0 | 1 | 1 | 33.3% |
| Model 3 | 0 | 0 | 0 | 33.3% |
| Boosting | 0 | 0 | 1 | 100% |

Table 2: Ensemble learning performance for binary classification. Upper table: Ensemble learning has positive impact; Lower table: Ensemble learning has negative impact.

With the Newton-Raphson approach, we got the $\widehat{f}$, so we can apply Gaussian Process to get $f^*$ (the $f$ value for the test set) by

$$f^*|\widehat{f} \sim \mathcal{N}(R^T C^{-1}\widehat{f},\ C^* - R^T C^{-1} R)$$

For Gaussian density, the MAP estimate is given by $f^* = R^T C^{-1}\widehat{f}$, the mean of the Gaussian distribution, then we can apply the logistic regression to $f^*$ to get $P_n^* = p(t_n = 1|f^*)$. If $P_n^* \geq 0.5$, then $t_n^* = 1$ (classify as label 1); if otherwise, then classify the test point $n$ as label 0.

In our classification problem, we used the cosine distance and the radial basis kernel with $\alpha = 1$ but various $\gamma$ (from 0.5 to 20) in getting the covariance matrix $C$. Figure 2b shows the training, validation, and test errors for various value of $\gamma$. We can see that as we increase $\gamma$ to a certain value, the training error is always 0, showing the over-fitting property of Gaussian Process. Moreover, the validation error and test error again converges to 0.465. However, for $\gamma = 0.5$, the validation error is 0.460, which is slightly better than 0.465.

**Ensemble Learning**

Among the three classifiers we chose for our problem, none of them performed well in predicting the daily stock price movement. They are only slightly better than guessing the label to be all 1s. One natural way for improving the performance of our classifier is to use Ensemble Learning. Ensemble learning is the process that combines multiple models together to solve a particular problem, and the advantage is more obvious when each of the combining models are weak learners, which are the models that only perform slightly better than random guessing. One common way to use Ensemble is by majority

vote. In the case of combining three models, we classify a data point as label 1 if at least two of the models classify it as label 1, which is shown in Table 2. Moreover, Table 2 also shows that it is possible for the Ensemble learning to have a negative impact.

For our classification problem, we combined KNN, Naive Bayes, and Gaussian Process Classification classifier for Ensemble Learning. We try various combination of $k$, $\alpha$, $\gamma$ (the parameters for each classifier), and the best validation error is 0.458 for $k = 17$ (KNN), $\alpha = 15.5$ (Naive Bayes), and $\gamma = 0.5$ (Gaussian Process Classification), and the corresponding test error is 0.459. However, this is still only a slightly better than 0.460 validation error from the Naive Bayes classifier. Moreover, we also get the probability of the data points assigning to each label, and we can take advantage of it when doing Ensemble learning. For binary classification, let $P_n^i$ be the probability of assigning data $n$ to label 1 computed by model $i$. We will classify the data point $n$ as label 1 if

$$\frac{1}{3}\left(P_n^1 + P_n^2 + P_n^3\right) \geq 0.5$$

The best validation error is 0.459 for this ensemble learning with $k = 17$, $\alpha = 15.25$ and $\gamma = 1.2$. Clearly, Ensemble Learning does not improve our classification.

**Summary**

All three binary classification, KNN, Naive Bayes, and Gaussian Process Classification, and their combining Ensemble Learning all perform poorly on predicting daily stock price movement by using the headline of daily news. The failure might come with many reasons. One is that Bag of words we had might still include too many words that might not be related to stock market at all, and their occurrence in the feature space might affect the predicting power of algorithm. We need a better way to further remove the irrelevant word from our bag. Another possible explanation is that the news today might not be able to influence the market immediately. The effect might come days later. Using today's news to predict today's stock price movement might not be accurate. Or it is possible that the daily stock market movement itself is random and unpredictable in its nature.

# 3 Learning the Hidden Stages

Since our attempt to predict daily stock price movements by using news headline fails, we shifted our focus to learn about some longer period pattern in the stock market in this Section. In this section we applied the HDP-HMM to learn about the hidden stages (Bull, Bear periods) in the stock market.

### Data Preprocessing

For this section, we used the Dow Jones Industrial Average (DJIA) index data from 1987 to May 11, 2017 collected from Yahoo. In general, the stock index is assumed to follow a lognormal distribution. Let $S_t$ denotes the stock price at time $t$, then $(\ln S_{t+1} - \ln S_t)$ follows a normal distribution. Thus, we first took the natural log for all the index data and then calculated the first differences. Moreover, we normalized the first differences by taking the $z$-scores so we can use uninformative prior for $\mu$ and $\sigma$ in the emission model (univariate normal).

### HDP-HMM

HDP-HMM, also known as the Infinite Hidden Markov Model (iHMM) is a non-parametric Bayesian extension of the HMM with an infinite number of hidden states. As introduced by Teh et al. [3], the hierarchical Dirichlet process is a set of Dirichlet Processes that shared a random base measure that is drawn from another Dirichlet Process with the top level base measure. More specifically, the process defines a set of probability measures $(G_j)_{j=1}^J$,

$$G_j|\alpha_0, G_0 \sim \mathrm{DP}(\alpha_0, G_0)$$

where $G_0$ is the shared global base measure and $\alpha$ is the concentration parameter. The global base measure $G_0$ is also distributed as a Dirichlet Process,

$$G_0|\gamma, H \sim \mathrm{DP}(\gamma, H)$$

where $\gamma$ is the top level concentration parameter and $H$ is the top level base probability measure. Let $y_t$ denotes the observation we have at time $t$ and let $z_t$ denotes the corresponding hidden states. Let $\phi_{z_t}$ denotes the parameters of the emission model (in the case of normal, $\phi_{z_t} = \mu_{z_t}, \sigma_{z_t}$), which $y_t|z_t \sim F(\phi_{z_t})$. Then, the HDP-HMM can be defined as the follows [4]:

$$\begin{aligned} &\beta \sim \mathrm{GEM}(\gamma), \qquad \pi_k|\beta \sim \mathrm{DP}(\alpha_0, \beta), \qquad \phi_k \sim H \\ &z_t|z_{t-1} \sim \mathrm{Multinomial}(\pi_{s_{t-1}}), \qquad y_t|z_t \sim F(\phi_{z_t}) \end{aligned}$$

where GEM is the stick-breaking construction for Dirichlet Process [3]. Many approaches have been introduced to make inference on the above model, one approach is to integrate out $\pi$ introduced by Teh et al. [3], and another approach is the Beam Sampling introduced by Van Gael et al. [4]. Both approach requires a lot of book-keeping in implementation, so we decided to use the weak limit approach described in Fox et al.'s Sticky HMM paper [1]. Instead of allowing for the "true" infinite number state, we will truncate the possible maximum number of state at some number $L$ that we think is much larger than

what we need, then

$$\beta|\gamma \sim \text{Dir}(\gamma/L, \ldots, \gamma/L) \text{ and } \pi_k|\alpha_0, \beta \sim \text{Dir}(\alpha_0\beta)$$

The rest of the model is the same as above. With the weak limit approach, we can use the forward-filtering, backward sampling procedure to sample our hidden states. The transition parameters $\pi$, $\beta$ and concentration parameters $\gamma$, $\alpha_0$ can be sampled as described in Teh et al.'s paper [3]. For our problem, the emission distribution is one-dimensional normal, which we used the following priors:

$$\mu_k \sim \mathcal{N}(0, 1) \text{ and } h_k \sim \text{Gamma}(1, 1)$$

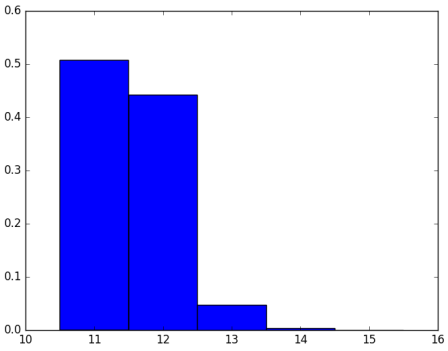
The update is the same as the one we use in Gaussian mixture model for the above two parameters. Moreover, for the inference on the hidden state of stock market, the prior for the concentration parameters are

$$\gamma \sim \text{Gamma}(1, 1) \text{ and } \alpha_0 \sim \text{Gamma}(1, 1)$$

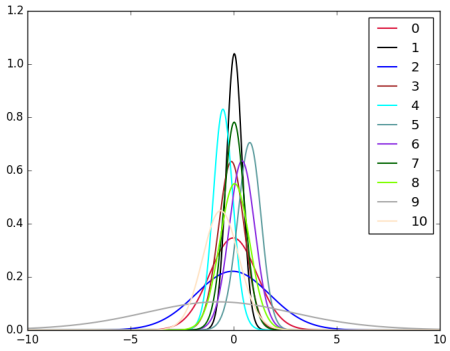
We believed that the possible hidden stage in the stock market should not exceed 7, so we picked $L = 15$ as the truncation for the maximum number of states.

**Results**

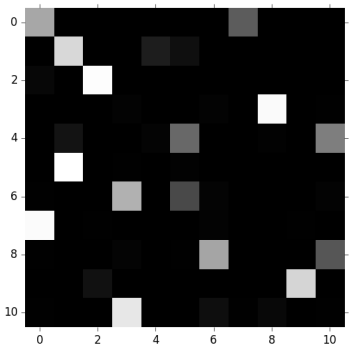
We implemented the HDP-HMM on the DIJA index from 1987 to 2017 for 5000 iterations and Figure 3 shows the results for samples after the post burn-in periods (burn-in is the first 1000 iterations). Figure 3a shows the distribution for the number of hidden states for post burn-in samples. Since our data consist 8000 days of data, a hidden state that occurred at least 10 times in the sample sequence would be considered as a state in that iteration. The result showed that there could be 11 or 12 states in the stock market, which is more than we expected. Then we averaged the occurrences for all the fifteen states for the post burn-in iterations and graphed the marginal distributions for the eleven hidden states that with average occurrences that is more than 10 times, which is shown in Figure 3b. We can see that hidden state 0, 1, 2, 7, 8 seemed to share the same mean but with different volatility. State 4 and 10 are the two states with the lowest mean, which represent for the strong bear market but with different volatility. State 3 has a negative mean but in a small magnitude, which might represent for the weak bear market. On the other hand, state 6 represents for the weak bull market and state 5 represents for the strong bull market. State 10 is the most interesting one with the lowest mean but very large volatility. It might represents the dramatic up or down in the market, and most of the time, dramatic downturn in the stock price happened more frequently than the



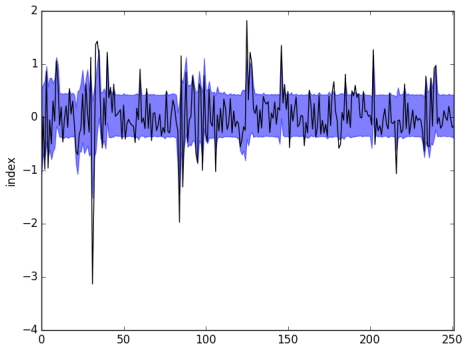

(a) Distribution for the number of hidden states for post burn-in samples.




(b) Marginal distributions of the hidden states for post burn-in samples.




(c) Transition matrix averaged over post burn-in sample.




(d) 67% HDI for past year predictions computed from post burn-in samples.

Figure 3: Results from the HDP-HMM on DJIA index.

dramatic rise. Figure 3c, and the state label is the same state label in Figure 3b. Self transition is favorable in some of the states, which is expected. State 0 and state 7 are favorable to move to each other, showing that volatility are frequently changing in the market as we can see from the marginal distribution of the two state in Figure 3b. The transition of state 4 (strong bear) is interesting: it prefers moving to state 5, strong bull, or state 10, another strong bear state but with more fluctuation. The movement from state 4 to 10 reflects market's strong adjustment. If price get dropped unexpectedly, people might purchase more the following days, bringing the index up. On the other hand, this also showed that the result from HMM did not capture the long term hidden stage that we wanted to capture. The model only captures some day to day movements: a bad day is generally from by a good day (the transition from 4 to 5) or by another bad day (the transition from 4 to 10). Last, we also computed the 67% highest density interval (HDI) for the marginal distribution for each trading day for the post burn-in samples, and the 67% HDI for the last year (252 trading days) index is shown in Figure 3d. From the plot, one can see that the HMM captures the normal fluctuation quite well. However, bear and bull market are considered as a short term large fluctuation in the market rather than some consistent pattern in the market. A hidden semi-markov model might capture this structure better than the ordinary HMM.

# 4 Conclusion

In the first part of the paper, we implemented the K-nearest neighbor, Bayesian Naive Bayes, and Gaussian process classification in predicting the rise or fall of the stock market (Dow Jones Industrial Average Index) based on the daily news headline from Reddit WorldNews Channel. It turned out that none of the above algorithms perform significantly better than guessing the label of all 1s. Then we implemented the Ensemble learning algorithm that combine the three classification algorithm mentioned above, and it still did not significantly improve our performance on making predictions. The weak performance might largely due to the bags of words we are using, which might contain many words that is not relevant but largely affect our classifiers. One future direction is to find a better way to construct our bags. The word2vec package Peter mentioned in the presentation will be our next step, since it can potentially further reduced our feature space. However, it is also possible that the market itself is unpredictable in nature, especially from a day to day basis, and our results substantiated the claim.

In the second part of the paper we shift our focus to unsupervised learning, which we try to investigate the number of hidden stages in the stock market by looking into the Dow Jones Industrial Average Index from 1987 to 2017. Our results showed that there are possible 11 hidden stages as shown in Figure 3b, but this is more than what we expected.

Moreover, the clusters we ended up reflect more about the short term fluctuations rather than some long term pattern, and this is largely due to the fact that HMM treat transition to each other state as independent while we expect there should be strong self-transition in a day to day basis. Hence, our future direction is to apply the Hidden Semi-Markov model on the same data set and see whether we can better learn about the long term pattern in the stock market.

Another future approach we are considering is to include the hidden states while evaluating news headline data in predicting the stock price movements. In bull market, since the market is generally performing well, it might overcome the effect of some bad news in the market. On the other hand, the effect of good news might not be able to reverse the downturn trend in the bear market. Thus, including the hidden stage, we can better learn about what news might contribute positively to the market, and this would be beneficial when the market are perform normally (with mean 0 movement).

# References


[1] Emily B Fox et al. "The sticky HDP-HMM: Bayesian nonparametric hidden Markov models with persistent states". In: *Arxiv preprint* (2007).

[2] Simon Rogers and Mark Girolami. *A first course in machine learning*. CRC Press, 2016.

[3] Yee Whye Teh et al. "Hierarchial Dirichlet Processes". In: *Journal of the American Statistical Association* (2006).

[4] Jurgen Van Gael et al. "Beam sampling for the infinite hidden Markov model". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1088–1095.