CS229 Project Report
Automated Stock Trading Using Machine Learning Algorithms

Tianxin Dai
tianxind@stanford.edu

Arpan Shah
ashah29@stanford.edu

Hongxia Zhong
hongxia.zhong@stanford.edu

1. Introduction

The use of algorithms to make trading decisions has become a prevalent practice in major stock exchanges of the world. Algorithmic trading, sometimes called high-frequency trading, is the use of automated systems to identify true signals among massive amounts of data that capture the underlying stock market dynamics. Machine Learning has therefore been central to the process of algorithmic trading because it provides powerful tools to extract patterns from the seemingly chaotic market trends. This project, in particular, learns models from Bloomberg data to predict stock price changes and aims to make profit over time.

Our algorithm tries to make a profit by predicting the price trend in the next minute based on the information seen earlier on that day. We make a buying decision if we predict an uptrend with a high confidence.

2. Data set

We use data obtained from Bloomberg terminal. Talk about how we bucketize it into minute-by-minute data (high/low/open/close) why we make this decision.

Also talk about universe selection.

Give an example with mean-reversion.

3. Approach

A brief paragraph about two approaches, pros/cons of each.

3.1. Logistic regression

We first used logistic regression to relate the percentage price change in the current minute to percentage price and volume change in the last minute. The intuition is quite simple: If the price went up in the last minute and volume also went up, that means more people are buying in the stock, and the uptrend should continue. Similarly, if the price went down in the last minute and volume went up, that means more people are selling the stock, and the downtrend should continue. On the other hand, if we are seeing a decrease in volume, the previous trend should revert. For our first attempt, we used six features to represent the trend in price and volume. Suppose we are trying to predict percentage change in closing price from $t-1$ to t, our regression variable y would be

$$\left(P_{close}^{(t)} - P_{close}^{(t-1)}\right)/P_{close}^{(t-1)} \tag{1}$$

Our six features would be:

1. $\left(P_{open}^{(t)} - P_{open}^{(t-1)}\right)/P_{open}^{(t-1)}$
2. $\left(P_{high}^{(t)} - P_{high}^{(t-1)}\right)/P_{high}^{(t-1)}$
3. $\left(P_{low}^{(t)} - P_{low}^{(t-1)}\right)/P_{low}^{(t-1)}$
4. $\left(V_{open}^{(t)} - V_{open}^{(t-1)}\right)/V_{open}^{(t-1)}$
5. $\left(V_{high}^{(t)} - V_{high}^{(t-1)}\right)/V_{high}^{(t-1)}$
6. $\left(V_{low}^{(t)} - V_{low}^{(t-1)}\right)/V_{low}^{(t-1)}$

However, we could not find a logistic model that fits this set of high-dimensional features well. We found out that our high-dimensional features are very noisy, so it is hard to fit a line through the data points.

In an attempt to reduce the dimensionality of our regression features, we selected the two most significant features:

1. $\left(P_{open}^{(t)} - P_{open}^{(t-1)}\right)/P_{open}^{(t-1)}$
2. $\left(V_{open}^{(t)} - V_{open}^{(t-1)}\right)/V_{open}^{(t-1)}$

$$\frac{\%\Delta P_{open}^{t}}{5MinHighLowSpread}$$

While logistic regression is able to fit a model with the two-dimensional feature set, the result is quite poor. The average profit made via trading on the selected 25 stocks is -? with a x% precision and y% recall. Then we tested our logistic regression model on the training set (9:30am to 11:00am) again and the results are also quite poor (give stats), this suggests that our model is suffering from high variance, and we must choose more descriptive features while keeping low dimensionality. Specifically, the percentage changes in price and volume of the one-minute timeframes are extremely volatile quantities, they are influenced by general trends of the market which normally lasts longer than one minute. For example, if the market enters a mode of frequent fluctuations in a certain time, then large 1-minute percentage changes in price or volume become less significant in comparison to the general trend. On the other hand, in a smoother market, even small percentage changes need to be considered with higher weights.

As a result, we decide to model the local general trend of the market from the high-low spread of the previous 5 minutes of any given time. Instead of using one-minute percentage changes, we choose to use percentage change over 5 minute high-low spread ratio of price and volume as our features. (maybe use math notations to express). Then we retrain the logistic regression model with the new features and we achieved quite promising results (x profit, y% precision, z% recall).

3.2. Conditional Random Fields and Graphcuts

Conditional random fields (CRFs) are the probabilistic model we use for segmention, which defines a conditional probability distribution of label sequences given the observation sequences. Suppose x is the features of the image, $y \in \{-1, +1\}$ is the segmentation label, with -1 denoting background and $+1$ denoting foreground. The CRF can be formulated as

$$P(y \mid x) = \frac{P(y,x)}{\sum_{y'} P(y',x)} = \frac{1}{Z(x)} \exp(-E(y,x)) \quad (2)$$

where $Z(x)$ is the normalizing function, and $E(y,x)$ is the energy function computed from various features. E consists of both node potentials and edge potentials. Intuitively, node potentials encodes the energy of a depth point taking on a particular label, and edge potential encodes the energy of a pair of depth points sharing the same label. In particular, E can be written as[1]

$$E(y,x) = \sum_{i\in\Phi_v} w_i \sum_{j\in v_i} \phi_j^{(i)}(y,x) + \sum_{i\in\Phi_{\mathcal{E}}} w_i \sum_{(j,k)\in\mathcal{N}_i} \phi_{jk}^{(i)}(y,x) \quad (3)$$

Here, Φ_v is the set of node potential indices (i.e. one for each type of node potential such as distance to previous foreground or bilateral filter), v_i is the set of all node indices for this potential type (one per depth point), and $\phi_j^{(i)}$ is the node potential of type i at depth point j. Similarly, $\Phi_{\mathcal{E}}$ is the set of edge potential indices, $\mathcal{N}_i$ is the edge set for edge potential type i. In our implementation, we draw an edge between depth point j and each depth point k within a hand-picked radius of point j. $\phi_{jk}^{(i)}$ is the edge potential between depth point j and k for edge potentials of type i. The weight w_i describes how important the ith potential is.

The goal of MAP inference in this model is to choose the most likely segmentation y given the features x by solving[1]

$$\underset{y}{\text{maximize}}\ P(y \mid x) = \underset{y}{\text{minimize}}\ E(y,x) \quad (4)$$

The solution of this problem can be efficiently computed using graph cuts. We construct graph G with all depth points as our node set, and edges between each depth point and its neighbors as our edge set. Then we assign node potential and edge potential using the definition of E above. We start with the seed frame, run graph cuts every frame in the sequence and use the current frame as ground truth for segmenting the next frame. Specifically,

Algorithm 1 Graphcuts segmentation.

```
1:  Let $D = \{d_1, d_2, \cdots, d_k\}$ be the $k$-frame-sequence we want to segment, where $d_i$ is the $i$th RGBD frame. Let $Y = \{y_1\}$ be the segmentation of this sequence. Initially, we only have $y_1$ which is the hand-segmented ground truth for the seed frame.
2:  for $i = 2, 3, \cdots, k$ do
3:      Construct graph $G$ with nodes $V = \{v_1, v_2, \cdots, v_n\}$ where $v_i$'s are the depth points of $d_i$.
4:      Use $y_{i-1}$ as ground truth labeling when computing node and edge potentials.
5:      for $v_j \in V$ do
6:          $\phi_{j_{source}} = \sum_{i\in\Phi_v} w_i \phi_{j_{source}}^{(i)}$ and $\phi_{j_{sink}} = \sum_{i\in\Phi_v} w_i \phi_{j_{sink}}^{(i)}$ where $\phi_{j_{source}}^{(i)}$ is the source potential of type $i$ at depth point $v_j$, $\phi_{j_{sink}}^{(i)}$ is the corresponding sink potential, and $w_i$ is the weight for node potential type $i$. A large positive source potential means that $v_j$ is more likely to be foreground and a large positive sink potential means that $v_j$ is more likely to be background.
7:          Add an edge $e_{jk}$ for every depth point $v_k$ that is within radius $r$ of $v_j$.
8:          for each $e_{jk}$ do
9:              $\phi_{jk} = \sum_{i\in\Phi_{\mathcal{E}}} w_i \phi_{jk}^{(i)}$ where $\phi_{jk}^{(i)}$ is the edge potential type $i$ for $e_{jk}$ and $w_i$ is the weight for edge potential type $i$. A large edge potential means that points $j$ and $k$ are more likely to share the same label.
10:         end for
11:     end for
12:     Run maxflow on $G$. Let $y_i$ be the source and sink labels produced by maxflow, where source denotes foreground and sink denotes background.
13:     $Y = Y \cup y_i$.
14: end for
```

3.3. Energy function terms

We now discuss different node and edge potentials we use in our graph cuts algorithm.

3.3.1 Node potentials

All node potentials have two parts, source potential and sink potential. We constrain them to $[0, 1]$. We visualize node potentials to give us an intuitive sense how they help the performance of our algorithm. Sample visualizations can be found in Figure 1.

Distance to previous foreground This potential is inversely related to the distance of a depth point to its nearest neighbor in the foreground of the previous frame. Formally,

$$\phi_{i_{source}} = \exp\left(-\frac{\|v_i - v'_k\|_2}{\sigma}\right) \tag{5}$$

and

$$\phi_{i_{sink}} = 1 - \phi_{i_{source}} \tag{6}$$

where v'_k is v_i's closest neighbor in the foreground of the previous frame, and σ is a hand-picked parameter to control how fast the potential falls off with distance.

Bilateral filter To compute bilateral potential of depth point v_i, we find all depth points within radius r of v_i in the previous frame. The bilateral potential is a sum of their labels weighted by their distance to v_i. Formally,

$$\psi_i = \sum_{v'_k \in N} y'_k \exp\left(-\frac{\|v_i - v'_k\|_2}{\sigma}\right) \tag{7}$$

where N is the set of all neighboring points of v_i in previous frame, $y'_k \in \{-1, +1\}$ is the label of v'_k in previous frame, σ is a hand-picked parameter to control how fast the weights fall off with distance. And we need to convert ψ_i to sink and source potential in range $[0, 1]$. We can do this by first converting it to $\phi_i \in [-1, +1]$

$$\phi_i = \frac{2}{1 + \exp(-\psi_i)} - 1 \tag{8}$$

Then we can convert it to range $[0, 1]$ by

$$\phi_{i_{source}} = \begin{cases} \phi_i & \text{if } \phi_i > 0 \\ 0 & \text{if } \phi_i < 0 \end{cases} \tag{9}$$

Similarly,

$$\phi_{i_{sink}} = \begin{cases} -\phi_i & \text{if } \phi_i < 0 \\ 0 & \text{if } \phi_i > 0 \end{cases} \tag{10}$$

3.4. Edge potentials

Edge potentials encodes the energy of two points sharing the same label, i.e. two points are more likely to share the same label if their edge potential is big and vice versa. Similar to node potentials, there is a σ term in all edge potentials which controls how fast the potential falls off.

3D distance This potential is inversely related to the 3D distance between 2 depth points. Formally,

$$\phi_{ij} = \exp\left(-\frac{\|v_i - v_j\|_2}{\sigma}\right) \tag{11}$$

Color distance This potential is inversely related to the distance between the RGB-color of the corresponding pixels of two depth points. Formally,

$$\phi_{ij} = \exp\left(-\frac{\|c_i - c_j\|_2}{\sigma}\right) \tag{12}$$

Surface Normal This potential measures how much the surface normal changes from one point to another. It is inversely related to the angle between surface normals at two points. Formally,

$$\phi_{ij} = \exp\left(-\frac{\arccos(n_i \cdot n_j)}{\sigma}\right) \tag{13}$$

where n_i and n_j are the unit-length surface normal vector at depth points v_i and v_j.

3.5. Learning

In our algorithm, we use Structural Support Vector Machine (SSVM) [4, 5] to choose weight w that maximizes the MAP. Each training example adds a constraint to our model. SSVM uses an iterative process to add more and more constraints and computes the weights that maximize the margin between the ground truth labels and the incorrect labels. To simplify the problem, we use the greedy approximation known as 1-slack formuation [3] . Thus we can formulate the approximation of the margin maximizing problem as follows:

$$\begin{array}{ll} \underset{w,\xi}{\text{minimize}} & \frac{1}{2}||w||^2 + C\xi \\ \text{subject to} & \xi \geq 0 \\ & \frac{1}{M}\sum_{m=1}^{M} E(\hat{y_m}, x_m) - E(y_m, x_m) \\ & \geq \frac{1}{M}\sum_{m=1}^{M} \Delta(y_m, \hat{y_m}) - \xi \\ & \forall(\hat{y_1}, ..., \hat{y_M}) \in \mathcal{Y} \end{array}$$

where Δ is a loss function and $\Delta(y_m, y)$ is 0-1 loss. C is a constant while ξ_m is a slack variable for training example m, and $\mathcal{Y}$ is the set of possible labelings of an image. Since graph cuts is based on non-negative edge weights, we constrain the edge weights to be non-negative.

Algorithm 2 Structural SVM for tracking.

```
1: $\mathcal{D}$ is a set of training examples $(y, x)$ created by human-labeling.
2: $C$ and $\varepsilon$ are constants, chosen by cross validation.
3:
4: $\mathcal{W} \leftarrow 0$
5: repeat
6:     Update the parameters $w$ to maximize the margin.
```

$$\begin{array}{ll} \underset{w,\xi}{\text{minimize}} & \frac{1}{2}||w||^2 + C\xi \\ \text{subject to} & w \geq 0, \xi \geq 0 \\ & \frac{1}{M}\sum_{m=1}^{M} E(\hat{y_m}, x_m) - E(y_m, x_m) \\ & \geq \frac{1}{M}\sum_{m=1}^{M} \Delta(y_m, \hat{y_m}) - \xi \\ & \forall(\hat{y_1}, ..., \hat{y_M}) \in \mathcal{Y} \end{array} \tag{14}$$

```
7:     for $(y_m, x_m) \in \mathcal{D}$ do
8:         Find the MAP assignment using graph cuts.
9:         $\hat{y_m} \leftarrow \arg\min_y E(y, x_m)$,
10:    end for
11:    $\mathcal{W} \leftarrow \mathcal{W} \cup \{\hat{y_1} ..., \hat{y_M}\}$
12: until
```

$$\begin{array}{l} \frac{1}{M}\sum_{m=1}^{M} \Delta(y_m, \hat{y_m}) - E(\hat{y_m}, x_m) \\ -E(y_m, x_m) \leq \xi + \varepsilon \end{array} \tag{15}$$

4. Experiment

We test our algorithm at different stages of implementation on XXX dataset and evaluated our results both qualitatively and quantitatively.

4.1. Data set

We use a dataset in the same format as Stanford Track Collection with 6 sequences and 227 frames extracted from natural, unstaged street scenes with a dense LIDAR system mounted on a car. Each sequence has 2 to 4 tracked objects. Data was recorded while driving and while parked at busy intersections with many people, bicyclists, and cars.

4.2. Evaluation Metrics

We evaluate the results both qualitatively and quantitatively. For our camshift baseline, we mainly evaluated it qualitatively by eyeballing the segmentation, while for our graphcuts algorithm, we mainly evaluated it quantitatively by calculating "Normalized Accuracy"[1] compared to our hand-segmented ground truth.

Normalized Accuracy is a measure to capture how many points we are classifying correctl normalized by the actual foreground size. Formally,

$$NA = 1 - \min\left(1, \frac{N_{fp} + N_{fn}}{N_{foreground}}\right) \tag{16}$$

where N_{fp} is the number of false positives (labeled as foreground by our algorithm but are actually background), N_{fn} is the number of false negatives (labeled as background by our algorithm but are actually foreground). And $N_{foreground}$ is the number of foreground points in ground-truth segmentation.

5. Experiment

5.1. Camshift baseline

Our baseline does a decent job on the first few frames, but has very high error after that. However, it has three major disadvantages. First, Camshift assumes that the camera does not shift position vastly. If the camera changes position and orientation a lot, Camshift has very high error tracking the object. Second, Camshift computes the position of the tracked object in the new frame by finding a window that has the highest back projection with regard to the color histogram of the object in the seed frame. It depends greatly on color, which is brittle to illumination and photometric variances, etc. Also when the object's color is close to background, Camshift cannot distinguish between them. Finally, Camshift is a bounding box tracker, while a lot of real world objects cannot be represented as a box. We cannot apply our Normalized Accuracy measure to Camshift results because the bounding box gives too many false positives. Figure 3 shows sample frames of Camshift segmentation on sequence 1.

5.2. Our algorithm

Compared to Camshift, our algorithm performs significantly better. The final normalized accuracy on 6 test sequences are shown in the following chart:

Sample frames extracted from 6 sequences are shown in Figure 4.

We tested our algorithm at different stages of implementation. Here is a summary of what features we added at each stage:

Stage 1 Used only distance to previous foreground as node potential and 3D distance as edge potential. Used coordinate system relative to camera on car.

Stage 2 Changed coordinate system to smooth coordinates. This is an absolute coordinate system based on GPS. Therefore, static objects, such as traffic signs, will have the same coordinates in every frame.

Stage 3 Added bilateral node potential.

Stage 4 Added color edge potential.

Stage 5 Added surface normal edge potential.

Stage 6 Implemented SSVM to learn weights of different potentials.

Table 1: Normalized accuracy on all frame at different stages of implementation

Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6
0.1393	0.3558	0.4036	0.4139	0.4156	0.5481
0	0.0733	0.0941	0.0778	0.076	0.2917
0	0.5770	0.5511	0.5193	0.5295	0.6264
0.1943	0.7512	0.7521	0.7573	0.7641	0.7685
0.1943	0.7015	0.7143	0.6917	0.7090	0.7300
0.1059	0.6200	0.6584	0.6184	0.6161	0.6867

We can see that [feature] and [feature] give us biggest improvement on performance. This corresponds with the weights we learned from SSVM. [feature1] and [feature2] have bigger weights while features such as color edge potential has a weight close to 0.

6. Conclusion

We have applied successfully a segmentation and tracking model for hand-held camera data to RGBD data captured by autonomous car. By trial and error we have gained a deep understanding of the model, the effects of different features, and SSVM setup. Due to time limit of a class project, we did not have time to experiment with more features that are more specific to the nature of autonomous car data. For example, we cannot calculate surface normal for a lot of far away depth points, because they do not have neighbors with which we can estimate the plane. If we had time, we could develop a new feature that computes the tangent through a few nearby points in the same depth ring to replace surface normals. Also, adding surface normal alone still does not get rid of all the points on the ground that touch the tracked objects. We may solve this problem by first applying ground subtraction, and then run our segmentation pipeline. However, the current result is satisfying both qualitatively and quantitatively. We believe the application of an improved version of the algorithm can help autonomous cars be more knowledgeable about the different kinds of obstacles and make finer-grained decisions in all kinds of traffic situations.

References

[1] Alex Teichman, Sebastian Thrun, Learning to Segment and Track in RGBD.

[2] Alex Teichman, Sebastian Thrun, Tracking-Based Semi-Supervised Learning.

[3] T. Joachims, T. Finley, and C.-N. Yu. Cutting-plane training of structural svms. Machine Learning, 77(1):2759, 2009.

[4] M. Szummer, P. Kohli, and D. Hoiem. Learning crfs using graph cuts. In ECCV, 2008.

[5] 21. B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: A large margin approach. In ICML, 2005.