ACTA ET COMMENTATIONES UNIVERSITATIS TARTUENSIS DE MATHEMATICA
Volume 21, Number 1, June 2017
Available online at http://acutm.math.ut.ee

Pattern recognition using hidden Markov models in financial time series

Sara Rebagliati and Emanuela Sasso

Abstract. Our aim consists in developing a software which can recognize M trading patterns in real time using Hidden Markov Models (HMMs). A trading pattern is a predefined figure indicating a specific behavior of prices. We trained $M + 1$ HMMs using Baum–Welch Algorithm combined with Genetic Algorithm. In particular, with HMMs we describe M trading patterns while the other one, called threshold model, can recognize all the not predefined patterns. The classification algorithm correctly recognizes 93% of the provided patterns. Thanks to the analysis of the false positive examples, we finally designed some more filters to reduce them.

1. Introduction

Automated trading systems have spread in the last years due to the advances in technologies. Before, traders usually took their investment decisions looking at the prices graphs and recognizing some trading patterns. Since our aim consists in replicating the behavior of a discretionary trader, we developed a software that can recognize trading patterns in real time using Hidden Markov Models (HMMs).

Trading patterns are predefined figures indicating a specific behavior of prices (see [7]). It is difficult to recognize them because they are defined by their shape and financial time series are strongly affected by noise. Hidden Markov Models are statistical models widely applied in speech recognition [8] and they can easily handle these patterns' characteristics.

HMMs are very good tools in simple classification applications, i.e., when we want to cluster some observations in well known classes. The big issue

Received September 29, 2016.
2010 *Mathematics Subject Classification.* 68T10.
Key words and phrases. Pattern recognition; hidden Markov models; financial time series; automated trading system.
http://dx.doi.org/10.12697/ACUTM.2017.21.02

in our application is that, in addition to the M classes referring to the interesting patterns, we need to define a class referring to all non reference patterns. It is also quite evident that the patterns we are looking for occur rarely comparing to the non reference ones. This is why it is so important to find a way to deal with them. The first paper dealing with this problem was published by Lee and Kim [6] in 1999. They suggested to use a HMM-based threshold model for gesture recognition and many researchers dealing with gesture recognition followed their idea.

Our aim is to apply a HMM-based threshold model in a different field. Extending the idea shown in [6] to continuous observations HMMs, we train a threshold model to recognize all the not predefined patterns. Then, we implement the classification algorithm to work in real time. Since a trader has to be fast to enter the market, we finally modify the algorithm to recognize forecasted scenario before they happen, as our brain does.

This paper is organized as follows. Section 2 gives a brief introduction on Hidden Markov Models and Genetic Algorithm (GA). We will explain how to estimate parameters of HMM combining the Expectation-Maximization Algorithm and GA. In Section 3, we will describe the data we are working with and the pre process needed to make our algorithm work in a general context. In Section 4, we will explain how to train HMMs for both reference and non reference patterns and we will present the recognition algorithms. Section 5 deals with online recognition. We will extend our algorithm to work in real time and to recognize trading pattern before they are completed. In Section 6, we will apply our software to recognize four trading patterns: double tops ("dtop"), double bottoms ("dbot"), head&shoulders ("has") and inverted head&shoulders ("invhas") and we will analyze the performance of our recognition algorithms. In Section 7, we will sum up the conclusions. We include two Appendices in this paper. Appendices A and B focus respectively on the Kullback–Liebler divergence and mixture reduction algorithm.

2. Introduction on Hidden Markov Model and Genetic Algorithm

In this section we will give a brief overview of Hidden Markov Model (HMM) [8] and explain how to estimate parameters of HMM combining the Expectation-Maximization Algorithm and Genetic Algorithm [5].

2.1. Hidden Markov Model.

A HMM is a statistical Markov Model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states.

Definition 1. A Hidden Markov Model (HMM) is a stochastic process $\{(X_t, Y_t)\}_{t=1,2,\ldots}$ satisfying the following properties:

$$p(X_t = x_t | X_{t-1} = x_{t-1}, \ldots, X_1 = x_1) = p(X_t = x_t | X_{t-1} = x_{t-1}),$$
$$p(Y_t = y_t | X_t = x_t, Y_{t-1} = y_{t-1}, \ldots, X_1 = x_1, Y_1 = y_1) = p(Y_t = y_t | X_t = x_t).$$

In particular, we assume that the process $\{X_t\}_{t=1,2,\ldots}$ is a not-observable homogeneous and stationary Markov Chain. On the other hand, the process $\{Y_t\}_{t=1,2,\ldots}$ is observable and thanks to it we can estimate the HMM.

In some applications, the hidden states have a clear meaning. For example, a hidden state may correspond to a phoneme in speech recognition. Sometimes, however, HMMs are simply used to model time series without defining the hidden states a priori. In this paper, we will firstly train HMMs and then we will investigate the interpretation of hidden states.
In general, a HMM is characterized by the following parameters.

- **N**: number of states in the model;
- **A**: $N \times N$ transition matrix of the hidden Markov Chain. By definition of Markov Chain, we have
$$a_{i,j} = \mathbb{P}(X_{t+1} = j | X_t = i) \quad i, j = 1, \ldots, N, \tag{1}$$
where X_t is the hidden process which describes the state visited at time t.
- **b**: a set of N emission probability density functions. Given the observation o_t at time t, we have
$$b_j(o_t) = p(Y_t = o_t | X_t = j) \quad j = 1, \ldots, N, \tag{2}$$
where Y_t is the observable process at time t;
- $\boldsymbol{\pi}$: initial state distribution, i.e.,
$$\pi_i = \mathbb{P}(X_1 = i) \quad i = 1, \ldots, N. \tag{3}$$

In this paper, we suppose that the hidden states are discrete while the observations are continuous and 2-dimensional. In particular, for each hidden state j the emission density function b_j is a Gaussian Mixture Model (GMM), i.e.,

$$b_j \sim \sum_{k=1}^{K_j} c_{j,k} \mathcal{N}(\mu_{j,k}, \sigma_{j,k}^2), \quad j = 1, \ldots, N, \tag{4}$$

where for all $j = 1, \ldots, N$ and for all $k = 1, \ldots, K_j$ the function $\mathcal{N}(\mu_{j,k}, \sigma_{j,k}^2)$ is a bivariate Gaussian distribution with mean vector $\mu_{j,k}$ and covariance matrix $\sigma_{j,k}^2$ and the positive weights $c_{j,k}$ satisfy the constraints $\sum_{k=1}^{K_j} c_{j,k} = 1$ for all $j = 1, \ldots, N$. Since the model is identified with its parameters, let us denote a HMM with $\lambda = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{b})$. Given an observation o_t at time t, we define the likelihood $p(o_t; \lambda)$ as the probability of o_t in the model λ.

There are different types of HMMs depending on the structure of the transition matrix of the hidden Markov Chain. The most common ones are ergodic and left-to-right models. In an ergodic HMM, you can move from one state to the others without restrictions. In a left-to-right HMM, you can move to the next states or stay in the same state but you cannot jump into a state you have already visited. We will consider only left-to-right HMMs.

2.2. Genetic Algorithm. Genetic Algorithm (GA) is a simulation of natural selection that can solve optimization problems. It is based on the Darwinian Theory of Evolution and it emulates the evolution of a population becoming fitter at each generation. Generally, a simple GA cycle consists of four operations: fitness evaluation, selection, genetic operations (crossover and mutation) and replacement. A simple GA cycle is shown in Algorithm 1.

Algorithm 1 A simple GA

```
Parents ← {randomly generated population}
while not (termination criteria) do
    calculate the fitness of each parent in the population
    Children ← $\emptyset$
    while |Children| < |Parents| do
        Use fitness to probabilistically select a pair of parents for mating
        Mate the parents to create children $c_1$ and $c_2$
        Children ← Children $\cup\{c_1, c_2\}$
    end while
    Randomly mutate some of the Children
    Parents ← Children
end while
```

In a simple GA cycle, there exists a population pool of chromosomes. The chromosomes are the encoded form of the potential solutions. Initially, the population is generated randomly and the fitness values of all the chromosomes are evaluated by calculating the objective function. After the initialization of the population pool, the GA evolution cycles begin. At the beginning of each iteration, the mating pool is formed by selecting some chromosomes from the population. This pool of chromosomes is used as the parents to generate an offspring via the processes of crossover and mutation. The fitness values of the offspring are also evaluated. At the end of each evolution cycle, some chromosomes of the population will be replaced by offsprings according to the replacement scheme. The above steps to create a new generation are repeated until the termination criterion is met. By emulating the natural selection and genetic operations, this process will hopefully lead to the best chromosomes, corresponding to the highly optimized solutions of the problem, in the final population.

The convergence to the global solution is not guaranteed but GA can easily reach very good solutions in few iterations.

2.3. HMM estimation through Baum–Welch Algorithm and GA. Let λ be a HMM. The standard way to set the parameters of λ is using the Baum–Welch algorithm (BWa), is an adaptation of the more general Expectation-Maximization algorithm [2]. Wu [13] proved that the BWa converges to a stationary point of the loglikelihood and not necessarily to the global maxima. Starting from different initial parameters, we can obtain different models.

In order to improve the performances of the BWa, we have combined it with a Genetic Algorithm to maximize the loglikelihood following the idea proposed by Kwong et al. [5].

The chromosomes encode the starting probability $\boldsymbol{\pi}$, the transition matrix $\mathbf{A}$ and the emission density functions $\mathbf{b}$ of some possible HMMs. The fitness function is the loglikelihood, which is the same function that the Baum–Welch Algorithm optimizes. Let us give some more details about the algorithm.

The initial population is obtained initializing the transition matrix randomly while the Gaussian Mixtures are estimated through the K-mean, a clustering algorithm [1]. Before starting the main loop, we perform some steps of Baum–Welch Algorithm on the initial population.

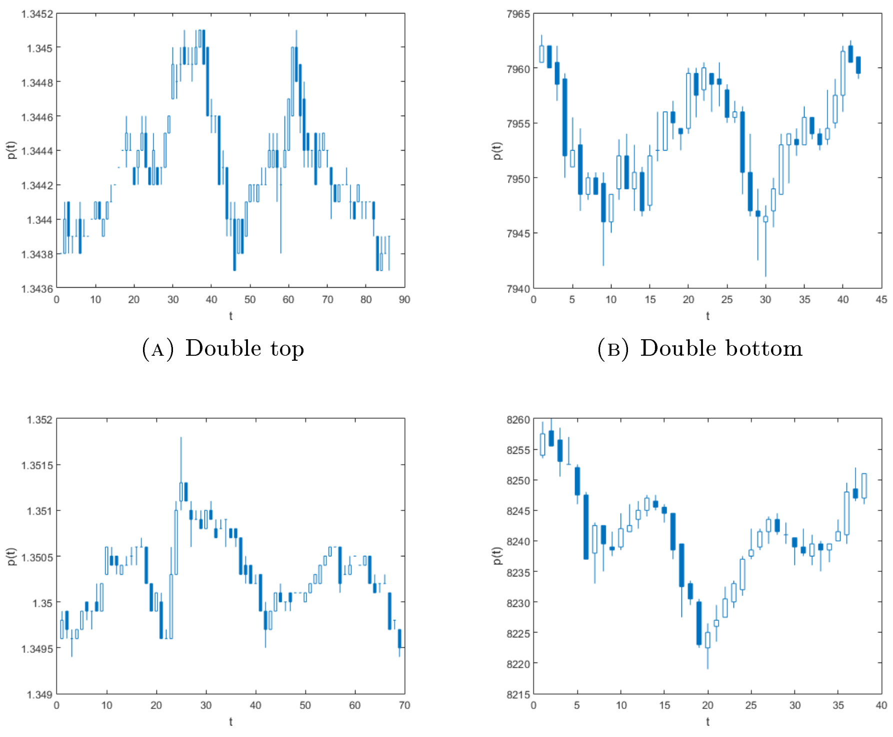
The main loop of the estimation algorithm consists of an alternation of a GA step and a predefined number of iterations of BWa: every time when we create a new population, we perform some iteration of BWa.

The last step consists in iterating BWa on the final population until the convergence is reached. Finally, we choose the model with the highest loglikelihood.

3. Data

In this section, we describe the data we are working with and the preprocess needed to make our algorithm work in a general context.

3.1. Data set. Our data consists of daily and intraday closing prices of some important financial assets such as DAX Future and EUR/USD Future. In this paper, we will show the application of the proposed algorithm on four trading patterns: double tops (“dtop”), double bottoms (“dbot”), head and shoulders (“has”) and inverted head and shoulders (“invhas”). We can see them in Figure 1. They are some well known patterns in technical analysis and they detect an inversion in the price trend.



(A) Double top (B) Double bottom

(C) Head&shoulder (D) Inv. head&shoulder

FIGURE 1. Trading patterns.

In order to create a database containing some examples of these patterns, we looked for them manually and we extracted them from the original time series. The random data set has been created extrapolating pieces of graph randomly and checking that they are not predefined patterns.

3.2. Pre-process. Trading patterns can occur at different time scales and have different amplitudes. In order to make the recognition algorithm independent from amplitude and time scale, we pre-process our pattern in the following way.

Let us consider a pattern $P = \{C_i\}_{i=1,\dots,T}$, where T is the length and C_i is the closing price at time index i.

The pre-process on data includes two steps.

(1) We normalize the closing prices such that

$$\tilde{C}_i = \frac{C_i - \min_s C_s}{\max_s C_s - \min_s C_s} \in [0,1], \quad i = 1, \dots, T.$$

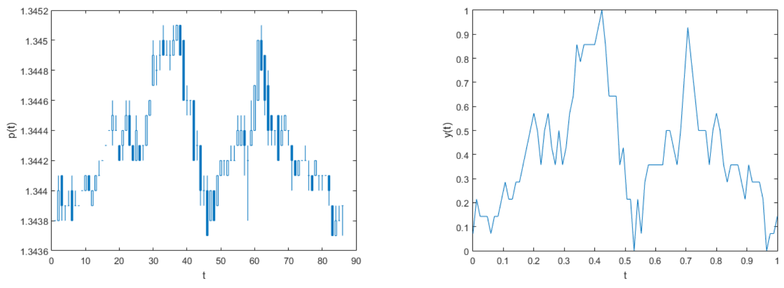
(2) We define

$$t_i = \frac{i-1}{T-1} \in [0,1], \quad i = 1, \dots, T.$$

Our new data set for each pattern consists of the 2-dimensional points

$$\tilde{P} = \{(t_i, \tilde{C}_i)\}_{i=1,\dots,T.}.$$

Now, $\tilde{P} \in [0,1] \times [0,1]$. Figure 2 shows the result of the pre-process algorithm applied to a double top.



(A) Double top

(B) Double top pre processed

FIGURE 2. Pre-processing algorithm.

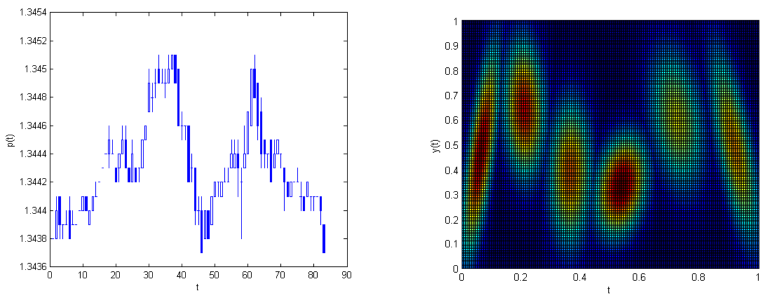
4. Models definition and static recognition

In this section, we explain how to train the HMMs for trading patterns and for non-reference patterns. Looking at the classification algorithm, we will focus on static recognition. We suppose to have some isolated pieces of financial time series and we want to answer the question if they represent an interesting pattern or not. In the first part of this section, we will assume that all the observations are trading patterns we are looking for. In the second part, we will include some non-reference patterns. We will explain how to define a threshold model and how the classification algorithm works in this case. In Section 5, we will show how to make this algorithm to work in real time.

4.1. Trading pattern. Let us suppose that we want to recognize M trading patterns and that we have a set of observations for each pattern.

4.1.1. *Model definition.* For each of the M patterns, we train some left-to-right models using different numbers of hidden states (from 6 to 15) and number of Gaussians in the mixture (from 1 to 3). The best HMMs are finally selected through cross validation.

We will denote $\lambda^i = (\boldsymbol{\pi}^i, \boldsymbol{A}^i, \boldsymbol{b}^i)$ the HMM describing the i^{th} pattern. Figure 3 shows a HMM dedicated to double tops. We can see that the shape given by the Gaussians is very similar to the one of double tops.



(A) Double top (B) HMM

FIGURE 3. A HMM dedicated to double tops.

4.1.2. *Classification algorithm.* Let $\lambda^i = (\boldsymbol{\pi}^i, \boldsymbol{A}^i, \boldsymbol{b}^i)$ be the HMM related to the i^{th} pattern for all $i = 1, \dots, M$ and let P be an example of the patterns we are looking for. We want to recognize which one it is. P belongs to the class $\hat{\lambda}(P)$ if

$$\hat{\lambda}(P) = \text{argmax}_{\lambda \in \{\lambda^1, \dots, \lambda^M\}} \log p(P|\lambda), \tag{5}$$

that is we associate the observation P to the model in which P has the highest probability.

4.2. Non-reference pattern. Let $\lambda^i = (\boldsymbol{\pi}^i, \boldsymbol{A}^i, \boldsymbol{b}^i)$ be the HMM related to the i^{th} pattern for all $i = 1, \dots, M$. In this section, we want to define the threshold model $\widetilde{\lambda} = (\widetilde{\boldsymbol{\pi}}, \widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{b}})$. We will show how to set the parameters of the threshold model and then we will focus on a reduction algorithm which is needed to reduce the computational cost.

4.2.1. *Threshold model definition.* A threshold model has been introduced to handle non-reference patterns [6]. It is a weak model for all trained patterns in the sense that its loglikelihood is smaller than that of the dedicated model for a given pattern. On the other hand, it should be a stronger model for random patterns which should be associated to it. Lee and Kim [6] defined a threshold model when both states and observations are discrete. We will extend their definition to the case when the emission density functions are Guassian Mixture Models.

In a left-to-right HMM, each state represents a subpattern of the complete one. The threshold model can be built putting together all the trained models.

In order to define a HMM, we have to set the number of hidden states $\widetilde{N}$, the starting probability $\widetilde{\boldsymbol{\pi}}$, the transition matrix $\widetilde{\boldsymbol{A}}$ and the emission probability density function $\widetilde{\boldsymbol{b}}$. The number of states of the threshold model is the sum of the states in the trained models, that is, if N^i is the number of states of the HMM λ^i, then the number of states $\widetilde{N}$ of HMM $\widetilde{\lambda}$ is

$$\widetilde{N} = \sum_{i=1}^{M} N^i.$$

The emission pdfs of $\widetilde{\lambda}$ are

$$\widetilde{\boldsymbol{b}} = \{\boldsymbol{b}^1, \dots, \boldsymbol{b}^M\}.$$

Previous literature suggests to connect all the states through an ergodic model. In our application, the HMMs are left-to-right. We decided to maintain the temporal evolution sorting the emission pdfs $\widetilde{\boldsymbol{b}}$ and defining $\widetilde{\boldsymbol{\pi}}$ and $\widetilde{\boldsymbol{A}}$ as follows.

We calculate the mean of $\widetilde{\boldsymbol{b}}_i$ for all $i = 1, \dots, \widetilde{N}$, and then we sort the states such that the time component of the means increases. We define a uniform tridiagonal transition matrix $\widetilde{\boldsymbol{A}}$, so we can move through the nearest states, i.e.,

$$\widetilde{\boldsymbol{A}} = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \dots & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & \dots & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}.$$

According to the definition of $\widetilde{\boldsymbol{A}}$, the starting probability are

$$\widetilde{\boldsymbol{\pi}} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, \dots, 0).$$

4.2.2. *Threshold model reduction.* Since the threshold model is a combination of models $\lambda^i = (\boldsymbol{\pi}^i, \boldsymbol{A}^i, \boldsymbol{b}^i)$ for all $i = 1, \dots, M$, we have already seen that the number of states of the threshold model is the sum of the number of states of the patterns' model. That is, the number of states can rapidly increase and so the time and space required to deal with this model. In order to avoid this problem, we propose an algorithm to reduce the number of states of the threshold model.

The reduction algorithm works in two steps. First, we select the state to merge choosing the ones that have the most similar emission probability density function. Then, we need to define the new emission pdf and modify the starting probability $\boldsymbol{\pi}$ and the transition matrix $\boldsymbol{A}$. We repeat these two steps until the target number of hidden states is reached.

Let us focus on the first step. In order to select the nearest state, we consider the Kullback–Leibler divergence (KL divergence) [4]. We can define it as follows.

Definition 2. Let f and g be two probability density functions over $\mathbb{R}^d$. The Kullback–Leibler (KL) divergence of g from f is defined as

$$d_{KL}(f, g) = \int_{\mathbb{R}^d} f(x) \log \frac{f(x)}{g(x)} dx.$$

The KL divergence is not a distance because it is not symmetric and it does not satisfy the triangular inequality. We can generalize it as follows.

Definition 3. Let f and g be two probability density functions over $\mathbb{R}^d$ and d_{KL} the KL divergence, we define the KL distance D_{KL} as follows

$$D_{KL}(f, g) = \frac{d_{KL}(f, g) + d_{KL}(g, f)}{2}.$$

For more details on the Kullback–Leibler divergence, see Appendix A.

Let $\lambda = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{b})$ be the HMM we want to reduce. The two states $(\bar{i}, \bar{j})$ to merge are

$$(\bar{i}, \bar{j}) = \operatorname{argmin}_{i=1,\dots,N, j=i+1\dots,N} D_{KL}(b_i, b_j).$$

Now, we need to define $\lambda^{\text{new}} = (\boldsymbol{\pi}^{\text{new}}, \boldsymbol{A}^{\text{new}}, \boldsymbol{b}^{\text{new}})$ removing the states $(\bar{i}, \bar{j})$ and creating a new one.

Looking at the starting probability, we simply remove the components $\boldsymbol{\pi}_{\bar{i}}$ and $\boldsymbol{\pi}_{\bar{j}}$ and we add a state with starting probability $\boldsymbol{\pi}_{\bar{i}} + \boldsymbol{\pi}_{\bar{j}}$, i.e.,

$$\boldsymbol{\pi}^{\text{new}} = (\pi_1, \dots, \pi_{\bar{i}-1}, \pi_{\bar{i}+1}, \dots, \pi_{\bar{j}-1}, \pi_{\bar{j}+1}, \dots, \pi_{\bar{i}} + \pi_{\bar{j}}).$$

It still holds $\sum_i \boldsymbol{\pi}_i^{\text{new}} = 1$.

The states $(\bar{i}, \bar{j})$ should also be deleted in the transition matrix, which means removing the $\bar{i}^{th}$ and $\bar{j}^{th}$ rows and columns from $\boldsymbol{A}$ and adding to it a column and a row related to the new state. The column to be added is obtained summing up the $\bar{i}^{th}$ and $\bar{j}^{th}$ columns of $\boldsymbol{A}$ while the row is equal to the mean of rows $\bar{i}^{th}$ and $\bar{j}^{th}$.

Given the two Gaussian Mixture Models $b_{\bar{i}}$ and $b_{\bar{j}}$, it is easy to merge them when creating a new GMM. The only operation consists of normalizing the weights. Let

$$b_{\bar{i}} = \sum_{k=1}^{K_{\bar{i}}} c_{\bar{i},k} \mathcal{N}(\mu_{\bar{i},k}, \Sigma^2_{\bar{i},k}) \text{ and } b_{\bar{j}} = \sum_{k=1}^{K_{\bar{j}}} c_{\bar{j},k} \mathcal{N}(\mu_{\bar{j},k}, \Sigma^2_{\bar{j},k}).$$

Then

$$b^{\text{temp}} = \sum_{k=1}^{K_{\bar{i}}} \frac{c_{\bar{i},k}}{2} \mathcal{N}(\mu_{\bar{i},k}, \Sigma^2_{\bar{i},k}) + \sum_{k=1}^{K_{\bar{j}}} \frac{c_{\bar{j},k}}{2} \mathcal{N}(\mu_{\bar{j},k}, \Sigma^2_{\bar{j},k})$$

The new emission probability density functions are

$$\boldsymbol{b}^{\text{new}} = (b_1, \dots, b_{\bar{i}-1}, b_{\bar{i}+1}, \dots, b_{\bar{j}-1}, b_{\bar{j}+1}, \dots, b_{\widetilde{N}}, b^{\text{temp}}).$$

However, repeating this procedure many times to reach the maximum required number of states, the number of Gaussians in a mixture can rapidly increase.

Salmond [10] proposed a mixture reduction algorithm in which the number of components is reduced repeatedly choosing the two components which appear to be most similar to each other, and merging them. His criterion of similarity is based on concepts from the statistical analysis of variance, and seeks to minimize the increase in within-components variance resulting from merging of the two chosen components. Once we selected the two Gaussians to merge, we use the moment-preserving merge algorithm. It is based on the idea that the new distribution must have the same zeroth, first and second-order moment as the original one. This mixture reduction algorithm is repeated until a target number of Gaussians per mixture is reached. For more details on the mixture reduction algorithm considered, see Appendix B.

The target number of states and the target number of Guassians per mixture are selected through a cross validation approach and they have to be similar to the number of states and Gaussians for mixture of the HMMs associated to the predefined patterns.

4.2.3. *Classification including threshold model.* Let $\lambda^i = (\boldsymbol{\pi}^i, \boldsymbol{A}^i, \boldsymbol{b}^i)$ be the HMM related to the i^{th} pattern for all $i = 1, \dots, M$, $\widetilde{\lambda} = (\widetilde{\boldsymbol{\pi}}, \widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{b}})$ the threshold model and let P be an observed pattern. P belongs to the class $\hat{\lambda}(P)$ if

$$\hat{\lambda} = \text{argmax}_{\lambda \in \{\lambda^1, \dots, \lambda^M, \lambda^T\}} \log p(P|\lambda). \tag{6}$$

In particular, if $\hat{\lambda}(p) = \widetilde{\lambda}$, P is not a pattern we are looking for.

5. Dynamic recognition

In order to replicate how a trader acts, the recognition algorithm should be able to work online in real time. At the end of a predefined time period (i.e., every minute), we ask the question: has a new interesting pattern appeared? The dynamic recognition gives an answer to this question.

5.1. Online recognition. The online recognition algorithm developed replicates how a trader looks for a pattern. Let us suppose we are at time T. We draw an imaginary horizontal line at the last price level and we look for the intersection between the line and the time series of prices. Once detected the intersection, we isolate the price in that window and we send this small time series to the static recognition algorithm.

In order to improve the performance, we do not look for the intersection using the price directly but we consider a moving average of prices. This is because the prices are very noisy and we prefer to use a smooth series to avoid false signals. Moreover, we fix a minimum temporal distance between the present time and intersection: we fix a minimum time amplitude for the patterns. Finally, once we recognize a pattern, we wait for a predefined time before attempting to recognize a new one.

5.2. Forecasted recognition. Since a trader needs to be fast to enter the market, we have finally modified the algorithm to recognize forecasted scenario before they happen, as our brain does.

Let $P = \{(t_i, C_i)\}_{i=1,\dots,T}$ be the price series available at time T. Before applying the online recognition explained in the previous section, we add a forecasted value to the time series. We repeat this procedure for a predefined tick variation. If we find a predefined pattern, we will pay attention if the price will reach the forecasted level in the following time interval before a new point is available.

6. Experimental Results

In this section we test the recognition algorithms explained in Section 4. We start investigating the performance of the classification algorithm applied to the four trading patterns shown in Figure 1. Then, we insert some non-reference patterns into our analysis and finally we look at the dynamic recognition working in real time.

6.1. Classification. Table 1 shows the results of a simple classification algorithm. Our data set is made up of 60 Double Tops, 41 Head&Shoulders, 59 Double Bottoms and 38 Inverse Head&Shoulders. For each of the provided observation, we classify it according to equation (5). In order to analyze the performance of our classification algorithm, we display the number of true positive, true negative and false positive values. We can see that the algorithm recognizes correctly 97% of the provided patterns.

6.2. Classification including non-reference pattern. In this section, we want to test a more realistic situation which includes non-reference patterns. We add 89 random patterns to our data set and we classify them

Table 1. Recognition results.

PATTERN	**N. TOT**	**TRUE POSITIVE**	**FALSE NEGATIVE**	**FALSE POSITIVE**
Double top	60	58	2	3
Head & shoulder	41	38	3	2
Double bottom	59	58	1	0
Inv. head & shoulder	38	38	0	1
Tot	198	192	6	–

through equation (6). Table 2 shows the number of true positive, true negative and false positive values. We can see that the algorithm recognizes correctly 93% of the provided patterns.

Table 2. Recognition results.

PATTERN	**N. TOT**	**TRUE POSITIVE**	**FALSE NEGATIVE**	**FALSE POSITIVE**
Double top	60	57	3	5
Head & shoulder	41	38	3	2
Double bottom	59	54	5	2
Inv. head & shoulder	38	37	1	2
Random	89	82	7	8
Tot	287	268	19	–

Remark 1. Thanks to the analysis of the false positive examples, we designed some more filters to reduce them. We will not explain in details the filters in this paper but we will use them in combination with the recognition algorithm previously exposed to obtain the results shown in the next section.

6.3. Dynamic recognition. In this section, we will show the performance we can reach with our recognition algorithm working in real time in combination with some filters designed to reduce the false positive patterns. We look for our four patterns in intraday data of DAX Future with timeframe 1 minute from 1^{st} January 2012 to 30^{th} June 2015. In particular, our algorithm correctly detects 428 double tops, 181 head&shoulders, 418 double bottoms and 217 inverse head&shoulders. Looking at the prices graphs, we can see that there are some patterns which occur and that we are not able detect but we preferred to eliminate the false positives to avoid investing money due to false signals.

7. Conclusions

In this paper we explained how to develop a software which can identify M trading patterns in financial time series in real time. In particular, we defined M Hidden Markov Models, one for each predefined pattern. Then, we explained how to define a HMM-based threshold model to handle non-reference pattern. Starting from the idea proposed in [6], we extended their algorithm to a more general one.

The simple classification algorithm can recognize properly 97% of the examples while the precision of the algorithm including non-reference patterns is 93%. Moreover, analyzing the false positive examples, we designed some filters to eliminate them. In this way, we increased the false negative cases but we preferred to eliminate the false positive cases to avoid investing money due to false signals. The algorithm proposed can actually work in real time and a recognition step takes less than 2 seconds.

In this paper we dealt with a specific application but the proposed algoritm can be easily applied to other research fields such as gesture recognition, for instance.

Appendix A. The Kullback–Leibler divergence

The Kullback–Leibler divergence [4], also known as relative entropy, is commonly used in statistics as a measure of similarity between two density functions.

A.1. Definition of KL divergence.

Definition 4. Let f and g be two probability density functions over $\mathbb{R}^d$. The Kullback–Leibler (KL) divergence of g from f is defined as

$$d_{KL}(f,g) = \int_{\mathbb{R}^d} f(x)\log\frac{f(x)}{g(x)}dx.$$

The Kullback–Leibler divergence satisfies three properties.

(1) Self similarity: $d_{KL}(f,f) = 0$.
(2) Self identification: $d_{KL}(f,g) = 0$ only if $f = g$.
(3) Positivity: $d_{KL}(f,g) \geq 0$ for all f, g.

The KL divergence is not a distance because it is not symmetric and it does not satisfy the triangular inequality. We can generalize it as follows.

Definition 5. Let f and g be two probability density functions over $\mathbb{R}^d$ and d_{KL} the KL divergence, we define the KL distance D_{KL} as follows

$$D_{KL}(f,g) = \frac{d_{KL}(f,g) + d_{KL}(g,f)}{2}.$$

The following proposition holds.

Proposition 1. *The KL distance D_{KL} is a distance.*

A.2. Calculation of KL divergence. We want to calculate the KL divergence for Gaussian Mixture Models. Unfortunately, there is a closed formed expression for Gaussian distributions but not for GMMs. In this section, we show the closed formula for a Gaussian distribution and an approximation algorithm for GMMs.

Proposition 2. *Let f and g be two d-dimensional Gaussian distributions, i.e., $f \sim \mathcal{N}(\mu_f, \Sigma_f)$ and $g \sim \mathcal{N}(\mu_g, \Sigma_g)$. The Kullback–Leibler divergence of g from f is*

$$d_{KL}(f,g) = \frac{1}{2}[\log \frac{|\Sigma_g|}{|\Sigma_f|} + Tr[\Sigma_g^{-1}\Sigma_f] - d + (\mu_f - \mu_g)^T \Sigma_g^{-1}(\mu_f - \mu_g)].$$

Now, we will focus on GMMs. We consider an approximation algorithm based on Monte Carlo sampling. This algorithm and other ones are described in [3].

Let f and g be two d-dimensional GMMs, i.e.,

$$f(x) = \sum_a \omega_a \mathcal{N}(x; \mu_a; \Sigma_a) = \sum_a \omega_a f_a(x),$$
$$g(x) = \sum_b \omega_b \mathcal{N}(x; \mu_b; \Sigma_b) = \sum_b \omega_b g_b(x).$$

The idea is to draw a sample x_i from the pdf f such that $\mathbb{E}_f[\log f(x_i)/g(x_i)] = d_{KL}(f,g)$. Using n i.i.d. samples $\{x_i\}_{i=1,\dots,n}$, we have

$$d_{MC}(f,g) = \frac{1}{n}\sum_{i=1}^{n} \log \frac{f(x_i)}{g(x_i)} \to d_{KL}(f,g), \qquad n \to +\infty.$$

The variance of the estimation error is $\frac{1}{n}\mathbb{V}_f[\log f/g]$.

To compute $d_{MC}(f,g)$, we need to generate the i.i.d. samples $\{x_i\}_{i=1,\dots,n}$ from f. We use the following algorithm.

For each $i = 1, \dots, n$,

- we select a Gaussian a_i randomly according to ω_a;
- we sample a point x_i from f_{a_i}.

Once we have a sample $\{x_i\}_{i=1,\dots,n}$, we can calculate $d_{MC}(f,g)$ which is a good approximation for $d_{KL}(f,g)$.

Appendix B. Mixture reduction algorithms

A common problem is to approximate a Gaussian Mixture by a model containing fewer components.

Salmond [10] proposed a mixture reduction algorithm in which the number of components is reduced repeatedly by choosing the two components which appear to be most similar to each other, and merging them. His criterion of similarity is based on concepts from the statistical analysis of variance, and seeks to minimize the increase in within-components variance resulting from merging the two chosen components. Once we selected the two Gaussians to merge, we use the moment-preserving merge algorithm. It is based on the idea that the new distribution must have the same zeroth, first and second-order moment of the original one.

B.1. Salmond's criterion. Let f be a GMM. We use notation

$$\{(\omega_1, \mu_1, \Sigma_1), (\omega_2, \mu_2, \Sigma_2), \dots, (\omega_n, \mu_n, \Sigma_n)\}$$

to denote a mixture of n such components.

Given the single components of the GMM we have

$$\begin{aligned} \mu &= \sum_{i=1}^{n} \omega_i \mu_i, \\ \Sigma &= \sum_{i=1}^{n} \omega_i \Sigma_i + \sum_{i=1}^{n} \omega_i (\mu_i - \mu)(\mu_i - \mu)^T \\ &= W + B. \end{aligned}$$

In the moment preserving merge, we keep Σ as constant, increasing W and decreasing B. Salmond's idea is to choose two components i and j such that W is minimized. The change in W when components i and j are merged is

$$\Delta W_{ij} = \frac{\omega_i \omega_j}{\omega_i + \omega_j} (\mu_i - \mu_j)(\mu_i - \mu_j)^T.$$

ΔW_{ij} is a matrix. Salmond proposed to use the dissimilarity measure

$$D_S^2(i, j) = Tr(\Sigma^{-1}) \Delta W_{ij}.$$

Other possible selection algorithms for mixture reduction are explained in [11], [12], and [9].

B.2. Moment-preserving merge. Suppose we have a GMM with just two Gaussian components and we want to merge them. The moment-preserving merge is based on the idea that the new distribution must have the same zeroth, first and second-order moment as the original one. In particular,

$$\begin{aligned} \mu &= \omega_1 \mu_1 + \omega_2 \mu_2, \\ \Sigma &= \omega_1 (\Sigma_1 + (\mu_1 - \mu)(\mu_1 - \mu)^T) + \omega_2 (\Sigma_2 + (\mu_2 - \mu)(\mu_2 - \mu)^T) \\ &= \omega_1 \Sigma_1 + \omega_2 \Sigma_2 + \omega_1 \omega_2 (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T. \end{aligned}$$

It has been proved that this is the Gaussian distribution whose Kullback–Leibler discrimination from the original distribution is minimal.

In general, given two weighted Gaussian components $(\omega_i, \mu_i, \Sigma_i)$ and $(\omega_j, \mu_j, \Sigma_j)$, their moment-preserving merge is the Gaussian component

$(\omega_{ij}, \mu_{ij}, \Sigma_{ij})$ as follows:

$$\begin{aligned} \omega_{ij} &= \omega_i + \omega_j, \\ \mu_{ij} &= \omega_{i|ij}\mu_i + \omega_{j|ij}\mu_j, \\ \Sigma_{ij} &= \omega_{i|ij}\Sigma_i + \omega_{j|ij}\Sigma_j + \omega_{i|ij}\omega_{j|ij}(\mu_i - \mu_j)(\mu_i - \mu_j)^T, \end{aligned}$$

where $\omega_{i|ij} = \frac{\omega_i}{\omega_i+\omega_j}$ and $\omega_{j|ij} = \frac{\omega_j}{\omega_i+\omega_j}$.

Acknowledgement. The authors wish to thank the referees for the remarks which helped to improve the version of the paper.

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via the em algorithm*, J. Roy. Statist. Soc. Ser. B **39**(1) (1977), 1–38.

[3] J. R. Hershey and P. A. Olsen, *Approximating the Kullback-Leibler divergence between Gaussian mixture models*, in: IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07, IEEE **4** (2007), pp. 317–320.

[4] S. Kullback, *Information Theory and Statistics*, Dover Publication Inc., Mineola, NY, 1968.

[5] S. Kwong, C. W. Chan, K. F. Man, and K. S. Tang, *Optimisation of hmm topology and its model parameters by genetic algorithm*, Pattern Recognition **34**(2) (2001), 509–522.

[6] H. K. Lee and J. H. Kim, *An hmm-based threshold model approach for gesture recognition*, IEEE Trans. Pattern Anal. Machine Intell. **21**(10) (1999), 961–973.

[7] M. J. Pring, *Technical Analysis Explained: The Successful Investor's Guide to Spotting Investment Trends and Turning Points*, McGraw-Hill Education, 2014.

[8] L. R. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, in: Proceedings of IEEE, IEEE **77** (1989), pp. 257–286.

[9] A. R. Runnals, *Kullback-Leibler approach to Gaussian mixture reduction*, IEEE Trans. Aerospace Electron. Syst. **43**(3) (2007), 989–999.

[10] D. J. Salmond, *Mixture reduction algorithms for target tracking in clutter*, in: Signal and Data Processing of Small Targets 1990, SPIE **1305** (1990), pp. 434–445.

[11] J. L. Williams, *Gaussian mixture reduction for tracking multiple maneuvering targets in clutter*, M.S.E.E. Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, OH, 2003.

[12] J. L. Williams and P. S. Maybeck, *Cost-function-based Gaussian mixture reduction*, in: Sixth International Conference on Information Fusion0, ISIF, 2003, pp. 1047–1054.

[13] C. F. Wu, *On the convergence properties of the em algoritm*, Ann. Statist. **11**(1) (1983), 95–103.

Università degli Studi di Genova, Dipartimento di Matematica, Via Dodecaneso 35, 16146 Genova, Italy

E-mail address: rebagliati@dima.unige.it

E-mail address: sasso@dima.unige.it