FINANCIAL TIME SERIES PREDICTION USING KALMAN FILTERS AND HIDDEN MARKOV MODELS

by

Burak Bayramlı

B.E., Computer Engineering, Stevens Institute of Technology, 1996

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Systems and Control Engineering
Boğaziçi University

2008

FINANCIAL TIME SERIES PREDICTION USING KALMAN FILTERS AND HIDDEN MARKOV MODELS

APPROVED BY:

Prof. Fikret Gürgen .....
(Thesis Supervisor)

Prof. Ethem Alpaydın .....

Prof. Levent Akın .....

DATE OF APPROVAL: .....

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Dr. Gürgen for his help, direction and patience during my work on this thesis.

Thanks to Dr. Ethem Alpaydın for first suggesting Systems and Control Engineering department as a good master's goal, and second for all the knowledge I gained in his *Pattern Recognition* class. Once in Systems and Control Engineering department, I was able to take immensely interesting classes such as *Nonlinear Dynamics* and *Simulation* both of which helped greatly in researching this thesis. I am also grateful to Dr. Goldsman for the all knowledge he was willing to part with in his *Simulation* class - I think his lectures on pseudorandom generation will stay with me forever. It was also in his class I first saw Brownian Motion which is very important in financial time series analysis.

I extend warmest thanks to Dr. Levent Akın for his excellent *Autonomous Robotics* class and his suggestion that I look into *Probabilistic Robotics*, a book and term popularized by Sebastian Thrun - it was while flipping through the pages of this book that I got the idea of using Kalman Filters for stock price prediction.

I would also like to thank my parents Ercan and Füsun Bayramlı for their support without whom none of this would have been possible.

ABSTRACT

FINANCIAL TIME SERIES PREDICTION USING KALMAN FILTERS AND HIDDEN MARKOV MODELS

Financial markets are challenging targets for analysis and prediction. The existence of many factors that contribute to a final price of security at time t, makes the task of predicting a future price a very hard task indeed. In the past, many statistical and non-statistical models have been utilized that attempted to perform this daunting task - the aim of this thesis is going to be trying to demonstrate Hidden Markov Models and Kalman Filter methods can be used for predicting a future price. We also devised a mixture predictor using HMM and KF which we called KMM.

For this purpose, we hypothized HMM, KF and KMM based models that are trained on historical data can generate future data, thereby predicting this securities' price in the future. For data generation, we used Monte Carlo simulation to smooth over the irregular patterns. "Carrying the model forward in time" is achieved by a combination of Viterbi algorithm and rolling the dice on hidden state transitions, in KF case, we follow the time transition equation.

Another goal of this thesis was to determine how HMM, KF and KMM methods stood in comparison to other conventional methods. We picked polynomial regression, Neural Networks (ANN) and plain "random guess" as our comparison criteria. Random guess was expected to be the lowest performer in our tests, every method mentioned should have surpassed random guess results by wide margin. We were glad to see that this was indeed the case.

We used Istanbul Stock Exchange ISE-100 and Dow Jones Industrial (DJI) Index historical data as our test data batch.

ÖZET

HMM VE KALMAN FİLTRELERİ KULLANARAK FİNANSAL ZAMAN SERİSİ TAHMİNİ

Finans piyasaları, analiz ve fiyat tahmini yapmak açısından zor bir alandır. Tahmin yapılmasını zorlaştıran faktör, herhangi bir gündeki fiyatın ortaya çıkmasında çok fazla etkinin olmasıdır. Tahmin ve analiz için şimdiye kadar pek çok istatistiki ve diğer yöntemler bu amaç için seferber edilmiştir - biz de bu yolda bir adım atmaya uğraştık.

Bu bağlamda Gizli Markov Modelleri ve Kalman Filtreleri adlı iki kuvvetli metotun, geçmiş senet verisini modelleyebilerek geleceğe dönük tahmin yapabileceğini varsaydık. Fiyat tahmin sayılarını üretirken Monte Carlo metotu kullanarak uç noktalarda olan tahminleri ortalama içinde düzeltmeye ve böylece daha iyi bir tahmine ulaşmaya çabaladık. Model eğitildikten sonra onu ileriye dönük tahminlerde kullanmak için ise, GMM durumunda Viterbi algoritması ile geçiş olasılıklarını rasgele sayı üreterek takip eden bir algoritma, Kalman Filtreleri durumunda ise en son gelinen zaman noktasından bir sonrakine geçiş için bizim değiştirdiğimiz zaman güncelleme denklemleri, ve yine rasgele sayı üreten bir algoritma kullandık.

Bu tezin bir diğer amacı tarif ettiğimiz metotların diğer klasik metotlara kıyasla nasıl başarı göstereceğini ortaya koymaktı. Bunun için Yapay Sinir Ağları, polinom regresyon ve en son pür rasgele sayı üreterek tahmin yapmaya uğraşan üç diğer metotu test senaryolarımıza dahil ettik. Bu metotların arasında rasgele üretim en alt seviyede performans göstermesini beklenilen metot idi, diğer tüm metotlar farklı derecelerde bu metotu geçmeliydi, ki deneyler sonucunda durumun böyle olduğunu gözlemledik.

Test verisi olarak Istanbul Menkul Değerler Borsası (IMKB) IMKB-100 ve Dow

Jones Endustriyel endekslerinin tarihi verisini kullanıldı.

TABLE OF CONTENTS

1. INTRODUCTION 1
1.1. Time Series 2
1.2. Financial Time Series 3
1.3. Return of Financial Time Series 4
1.4. ISE and DJI Data 5
1.5. Classic Methods: ARCH, GARCH, EGARCH models 8
1.6. ANN 10
1.7. Polynomial Regression 12
1.8. KMM 13
1.9. Previous Work 14
2. HIDDEN MARKOV MODELS 16
2.1. Structure 17
2.2. Forward-Backward Recursion 19
2.3. Training 21
2.4. Algorithm 22
2.5. Prediction with HMM 23
3. KALMAN FILTERS 26
3.1. Filtering 28
3.2. Smoothing 32
3.3. Training 34
3.4. Usage 36
3.5. Algorithm 37
3.6. Prediction with Kalman Filters 38
4. RESULTS 41
4.1. Monte Carlo Simulation 41
4.2. Results 42
4.2.1. Metrics 43
4.2.2. Plots 44
4.2.3. Conclusion 45

REFERENCES 52

LIST OF FIGURES

Figure 1.1. Some Values from DJI and ISE-100 4

Figure 1.2. IMKB Volatility 5

Figure 1.3. Example of Polynomial Fitting 13

Figure 4.1. One Day ISE-100 Predictions Using HMM, KF and ANN 45

Figure 4.2. One Day ISE-100 Predictions Using polynomial, random and KMM 46

Figure 4.3. One Day Predictions Using HMM method by Hassan & Nath [18] 47

LIST OF TABLES

Table 1.1. Istanbul Stock Exchange 6

Table 1.2. Dow Jones Industrial 6

Table 4.1. ISE Prediction - 1 day 47

Table 4.2. ISE Prediction - 5 days 48

Table 4.3. ISE Prediction - 10 days 48

Table 4.4. Dow Jones Prediction - 1 day 49

Table 4.5. Dow Jones Prediction - 5 days 49

Table 4.6. Dow Jones Prediction - 10 days 50

LIST OF SYMBOLS/ABBREVIATIONS

a_{ij}	HMM transition probability $p(q_t^j = 1 \mid q_t^i = 1)$ of hidden states
A	HMM hidden transitions matrix. a_{ij} accesses elements of A
A	Linear multiplier for KF time transition from time t to $t+1$
$\alpha(q_t)$	Probability of emitting partial sequence of outputs $y_0, ..., y_t$ and ending up in state q_t [8] for HMM
$\beta(q_t)$	Probability of emitting partial sequence of outputs $y_{t+1}, .., y_T$ given that system starts in state q_t for HMM
C	A vector holding all means of emission Gaussians for HMM
C	Linear multiplier for KF that translates x_t to y_t
η	Distribution representing output emissions for HMM
ϵ	Error
γ_t	HMM posterior probability $p(q_t \mid y)$ at time t
h_t	Sum of errors going back p days for ARCH / GARCH / EGARCH modeling.
$\hat{x}_t^t$	Conditional expectation of x at time t given output emissions $y_0, ..., y_t$
K_t	Kalman gain matrix at time t
x_t	Random variable representing hidden state at time t for KF (Gaussian distribution).
v_t	White noise - Gaussian distribution with zero mean and covariance Q
w_t	White noise - Gaussian distribution with zero mean and covariance R
π	HMM initial probability distribution
P_t	Price of a security at time t
R_t	Rate of return for financial time series data
R	Variance of the emission Gaussian for HMM
P_t^t	Conditional variance of x at time t given output emissions $y_0, ..., y_t$
q_t	HMM's hidden state symbol at time t

T	Number of training data points
$P_{t,t-1}$	Joint covariance necessary for KF lag-one covariance smoothing calculations
ξ_t	HMM cooccurence probabilities of q_t and q_{t+1}
y	All of the measured/observed data for HMM, KF and KMM
ANN	Artificial Neural Network
ARCH	Autoregressive conditional heteroskedasticity
DJI	Dow Jones Industrial
EM	Expectation Maximization
GARCH	General ARCH
HMM	Hidden Markov Models
HME	Hidden Markov Experts
iid	Independent Identically Distributed
ISE	Istanbul Stock Exchange
KF	Kalman Filters
MC	Markov Chains
MLP	Multilayer perceptron
RTS	Rauch-Tung-Striebel
SSM	State space model

1. INTRODUCTION

The aim of this thesis is to explain and apply Hidden Markov Model (HMM) and Kalman Filter (KF) methods in context of time series prediction problem. Istanbul Stock Exchange (ISE) and Dow Jones Industrial (DJI) index historical prices were used as a testbed for both models and algorithms. In order to show the capability of HMM, KF and KMM methods, various other non-statistical methods were also used on the same dataset among which are ANNs, polynomial fitting, and finally a simple random number generator based predictor.

We also combined HMM, KF and KMM methods in a hybrid mechanism called KMM. This method tries to merge both HMM, KF and KMM results through predefined weights. It was hoped that the total would be better than the sum of its parts, that is, HMM and KF combined could yield better and different results than each method taken seperately.

Our test data sets, ISE and DJI consisted of daily closing prices for over 21 years for ISE and 30 years for DJI. DJI data was obtained from `http://finance.yahoo.com` site. ISE-100 historical data is available from Turkish Central Bank site at `http:/¬ /evds.tcmb.gov.tr`. We attempted to predict future price for 1, 5, and 10 days for each method mentioned above. All methods were given a limited "time window" on which training took place, after which, prediction for future prices could be performed. We kept the training window the same size for all algorithms with the exception of ANN because too much data was causing overfitting and effecting results negatively.

A natural benchmark for success of HMM, KF, KMM algorithms, or any predictive algorithm, should be it's relative success against a random predictor. This idea was inspired by Fama [14] where he states "if [an] analyst can make meaningful judgments concerning the purchase and sale of individual securities, his choices should consistently outperform *randomly* selected securities of the same general riskiness". One can also say the worst-case performance, lower bound of any financial predictive algorithm

should be random prediction. Any algorithm which can beat random prediction can claim that it is calculating "some" trend in the market that "pure chance" does not.

Before further delving into the details of each algorithm, we will briefly introduce time series analysis in general, and financial time series analysis in particular. We will provide a motivation for the methods we used, the details of HMM, KF and KMM methods, the results of all algorithms mentioned and finally a detailed analysis of the outcome.

1.1. Time Series

Time series data analysis is a natural by-product in many fields as diverse as seismology, biology, physics, speech processing, medicine and more. Such fields produce experimental data that needs to be analyzed using specific time series methods and in certain cases, this analysis needs to be conducted through probabilistic, statistical models.

Historically time series analysis was divided into two schools: *Frequency domain* and *time domain* approaches. Time domain approaches stated current data was a function of earlier data and concocted an autocorrelation function ρ_l of the data, frequency approaches used spectral analysis where Fourier transform of the autocorrelation function became important [15]. The sharp divisions that seemed to exist between the two approaches seem to be settled, and at the same time the use of state space models and Kalman filtering in 1980s caused an explosion of interest in related models for time series analysis.

Statistical models try to explain sample data using a collection of random variables, in order, that fluctuate over time. For example a time series sample is represented by a collection of 10 random variables $x_1, .., x_{10}$ where random variable x_1 represents the value of the series at the first time point, x_2 at the second time point, so on. A collection of random variables x_t indexed by t is referred to as a stochastic process [7].

An excellent review of time series analysis methods can be found in [15].

1.2. Financial Time Series

Financial time series data brings forth certain issues with it from a statistical analysis standpoint, most important of which is the interdependency of one data point with previous data points. As a result of this interdependency, the modeler of the data is unable to assume each point to be independent and identically distributed (i.i.d.) which complicates the modeling process.

The collapse of Long-Term Capital Management in 1998 proved to modelers of financial time series that one could not see fluctuations of stock prices as though coming from i.i.d. normal density functions. The said company based its investment on such an model, and when it collapsed the situation threatened to open a trillion-dollar hole in the banking industry. Since Bachelier [16], constructing finance models on Brownian Motion had been popular until scientists started to poke holes in the theory and real-life catastrophes such as the one above proved otherwise. Brownian Motion based modeling requires that at each time step the price difference was a result of an indepedent normal distribution. Fama [13] offered a detailed explanation of the problem with that approach:

> "If the population of price changes is strictly normal, on the average for any stock ... an observation more than five standard deviations from the mean should be observed about once every 7,000 years. In fact such observations seem to occur about once every three to four years."

In other words Fama's findings indicated that independence assumption was incorrect. HMM, KF and KMM methods construct a Bayesian graphical model that tries to address this problem.

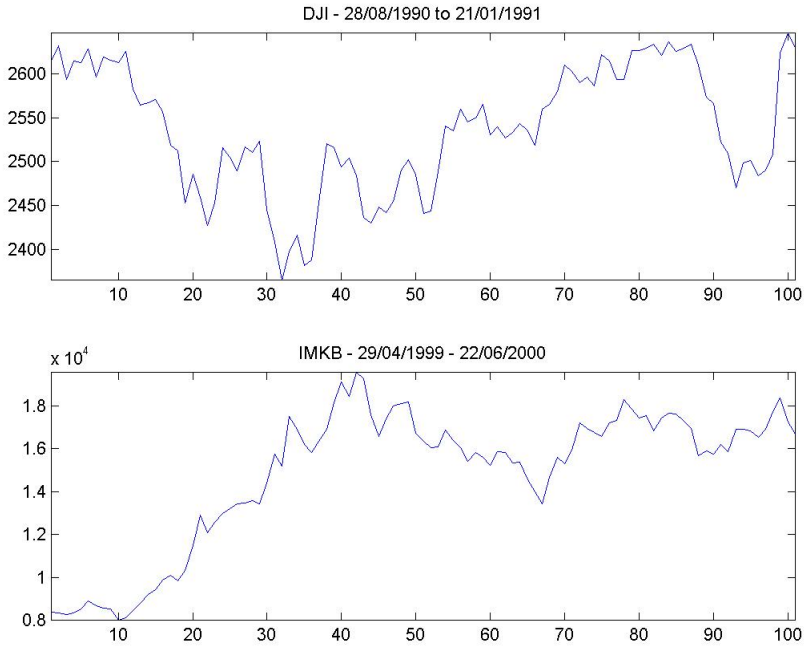


Figure 1.1. Some Values from DJI and ISE-100

1.3. Return of Financial Time Series

We wanted to limit variations in price changes to a respectable range of values, therefore chose to utilize rate of return of ISE and DJI price changes, instead of the absolute values themselves. For time t, the return R_t can be calculated as

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

where P_t is the price of the financial instrument at time t.

In order for HMM, KF and KMM models to learn from the data, we first pre-processed it converting regular prices into rate of returns. This way after training and prediction, generated values from both models would automatically be daily returns themselves. However if an application required predictions in the form of "absolute stock values", one needs go from rate of returns back to absolute values, in this case one can simply take last absolute value for known price of that security, and start multiplying with the predicted rate of returns values. For example, if training was

performed on $P_1, ..., P_T$ where T is the amount of training points, then HMM or KF would predict returns for $R_{T+1}, ..., R_{T+30}$, and using the last known price point P_T, we would start multiplying our generated returns and obtain $P_{T+1}, ..., P_{T+30}$.

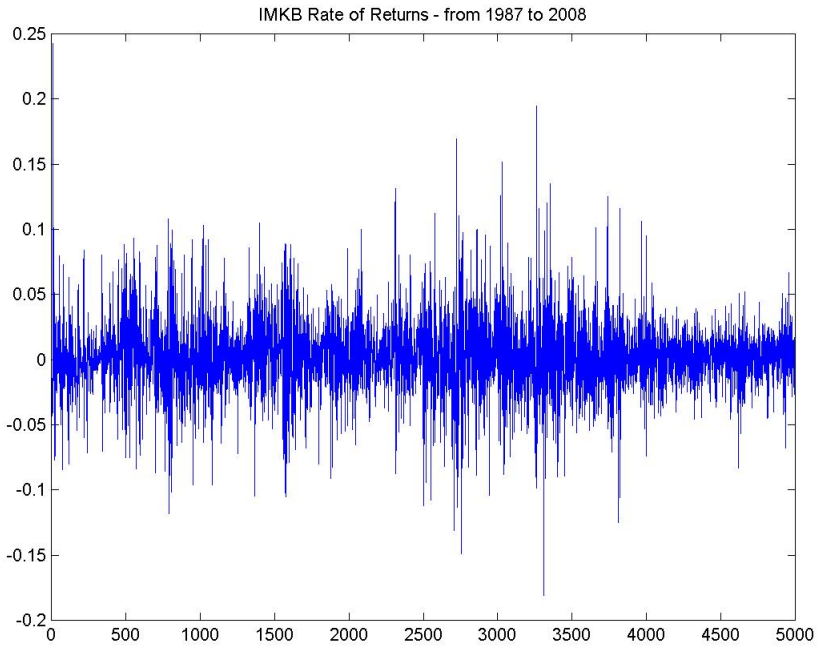


Figure 1.2. IMKB Volatility

1.4. ISE and DJI Data

Now let us look at the data more closely. ISE National-100 index and Dow Jones Industrial index are not individual stock prices themselves, instead they are *summaries* over certain stock prices. This summarization capability makes them good indicators, as economists like to say, especially an index such as ISE-100 can signal good or bad times in the nation's stock market because of the many number of companies' stocks it choses to reflect.

There are many methods to calculate an index. Some methods calculate a simple average, e.g. Dow Jones Industrial index is such an average. 30 industrial companies with high enough valuation in New York Stock Exchange were picked to form this index, and the stock price for all 30 companies were simply averaged to obtain a final DJI value.

ISE National-100 index on the other hand, is an index that is a weighted average - it takes more than one parameter to calculate a final result and it assigns different weights to each financial instrument. The formula is

$$E_t = \frac{\sum_{i=1}^{n} F_{it} N_{it} H_{it}}{B_t}$$

where

E_t: The value of index at time t

n: The number of companies that contribute to the index' calculation

F_{it}: The value of i^{th} stock at time t

N_{it}: The total number of shares outstanding of the stock i at time t.

H_{it}: The flotation weight (publicly-held portion, i.e. the ratio of stocks kept in custody at Takasbank, except those kept in non-fungible accounts) of the stock i at period t.

B_t: The value of divisor at period t (Adjusted base market value).

Here are some statistics on the DJI and ISE data:

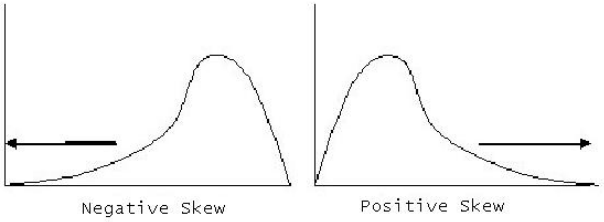
Size	Mean	Var	Kurtosis	Skew	J.Bera Null	Min	Max
5045	0.00214	0.0008	6.9154	0.2745	1	-0.1811	0.242

Table 1.1. Istanbul Stock Exchange

Size	Mean	Var	Kurtosis	Skew	J.Bera Null	Min	Max
9341	0.00035	0.0001	34.40	-1.15	1	-0.226	0.101

Table 1.2. Dow Jones Industrial

Notable statistics presented above are kurtosis, skew and Jarque-Bera test. Skewness is a measure of asymmetry, visually a distribution with positive skewness would have a so-called tail much longer on its right side than on its left. The reverse statement is true for negative skewness - such distributions would have a longer tail on the left side of its graphic. Longer tail on one side means more data would come from that side of that distribution.



Mathematically, skewness is the third standardized moment γ_1 calculated as

$$\gamma_1 = \frac{\mu_3}{\sigma^3}$$

where μ_3 is the third moment and σ is the well-known standard deviation. Skewness can signal us if one can assume a distribution is Gaussian. Gaussian distribution has a skewness of zero, hence if sample skewness of a distribution is non-zero, we can conclude that this distribution is not Normal.

Kurtosis, another important metric, tells us something about the presence of extremely high values in the data. Standard normal distribution has a kurtosis of zero, higher kurtosis values signal the presence of sharper peak in the distribution, lower kursosis distributions tend to have flatter peaks. In other words, kurtosis metric can tell us if extreme values are found more commonly in a distribution or not. The formula for kurtosis is

$$\frac{\sum_{i=1} T(X_i - \bar{X})^4}{(T-1)\sigma^4}$$

Taken together skewness and kurtosis metrics can indicate if a distribution is normal. There is also a goodness-of-fit measure called Jarque-Bera test that can indicate normalty of a distribution. This test combines skewness and kurtosis measures to accept or reject the null hypothesis that the distribution in question is normal.

Looking at skewness and kurtosis for ISE and DJI data sets, we can see kurtosis value is very high for both of the sets. This means there are a lot of extreme values present in the sample data. It can also be observed that ISE-100 is skewed positively,

though not by much, and DJI data is skewed negatively. It is known in finance literature that stock markets are skewed negatively because decreases are much sharper than increases. An hypothesis as to why claims that the reason is related to basic human psychology: At time of crises, people panic and group mentality tempts a big number of people to sell their stocks at the same time, hence causing sharper decreases than increases in the stock market data. The claim seems to be justified by the presence of certain "brakes" in the market in almost all markets in the world with the help of which, if daily market decrease passes a certain predefined threshold, all activity in the market is stopped until the next day. The goal here is giving traders time to think and calm down, trying to offset the selling histeria that might be caused in a moment of confusion.

We mentioned almost all markets are skewed negatively, then we need to explain why ISE-100 skewed positively. The reason here could be that ISE is a relatively young market and the composition of ISE-100 index has changed over the years, hence the sample skewness for 20 years worth of data points (as we have done) might not offer the best results. We hypothized taking later years' data points would offer negative skewness. This turned out to be correct,, a skewness calculation for around last 8 years did indeed offer a negative result.

1.5. Classic Methods: ARCH, GARCH, EGARCH models

Least squares models have been used on financial time series data for many years. One of the important assumptions that least squares based analysis makes is the variance of error terms stays the same throughout the time series. This assumption, which is called homoskedasticity, does not seem to hold for financial time series data consistently. It has been shown that the reverse condition is true, called heteroskedasticity, that posits the variance of the error changes. ARCH/GARCH methods attempt to model this variance [20].

ARCH stands for **a**uto**r**egressive **c**onditional **h**eteroskedasticity, and GARCH is **g**eneral ARCH. First proposed by Engle [22] then Bollershev [23], these models try

to put forth a correlation between variance of error at time t and variance on days preceding t. The reason for such relationship is the fact that volatility seems to have "regions" as observed by Mandelbrot in [25] - he stated that in rate of return data "large changes tend to be followed by large changes, of either sign, and small changes tend to be followed by small changes". This and similar observations led to an array of research on volatility clustering using methods such as ANN and many other statistical tools. (G)ARCH methods are one of these methods which try to forecast volatility based on historical rate of return data. As of 1992, a survey showed over 200 papers had used (G)ARCH and related models for financial time series [21].

As shown in [21], let r_t be the rate of return of a stock from time $t-1$ to t, and F_{t-1} be "information" that is relevant to make a decision about buying or selling a security at time $t-1$. Then the expected rate of return at time t can be shown as $E(r_t|F_{t-1})$ and the variance can be shown as $Var(r_t|F_{t-1})$. Let us call the former m_t and the latter h_t. Engle [22] suggests that h_t can be modeled as a function of ϵ_t which is the error calculated as $\epsilon_t = r_t - m_t$. In a ARCH(p) model this can be shown as

$$h_t = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2$$

where ω, $\alpha_1, .., \alpha_p$ and p are all constants which (except p) are to be estimated from the past historical data. According to this ARCH(p) model, past errors (news) older than p periods have no effect on the current day.

GARCH(p,q) model invented by Bollershev [23] generalizes ARCH(p) model as

$$h_t = \omega + \sum_{i=1}^{p} \alpha_i \omega_{t-1}^2 + \sum_{i=1}^{q} \beta_i h_{t-i}$$

where $\alpha_1, .., \alpha_p, \beta_1, ..., \beta_p$ and ω are constants.

EGARCH model proposed by Nelson [24] attempts to correct problems in (G)ARCH modeling that stem from the asymetric nature of the stock price data: Statistical analysis of volatility showed that drops in markets are sharper and deeper than increases. Some try to explain this phenomena through psychology, bad news cause hysteria and momentarily cause everyone to panic, which, it is claimed, makes drops much worse than increases. In comparison good news in the market is taken in much more rationally and causes more moderate increases than decreases. (G)ARCH models do not take this asymetry into account which EGARCH models attempt to correct. EGARCH is formulized as

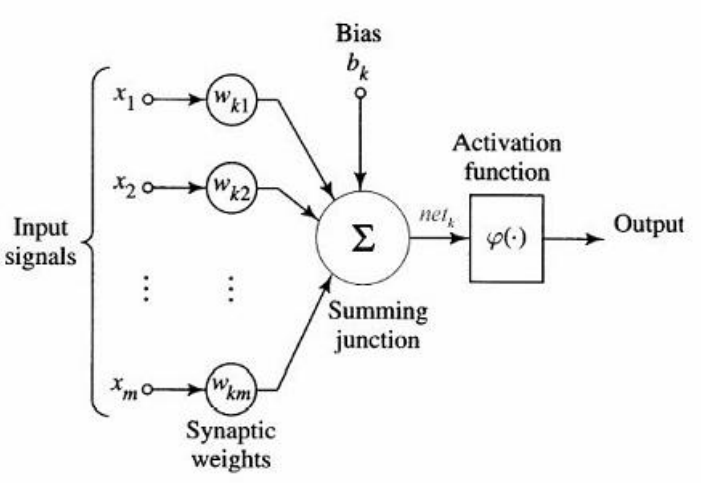
$$log(h_t) = \omega + \beta \cdot log(h_{t-1}) + \gamma \frac{\epsilon_{t-1}}{\sqrt{h_{t-1}}} + \alpha \left[\frac{|\epsilon_{t-1}|}{\sqrt{h_{t-1}}} - \sqrt{2/\pi} \right]$$

where ω, β, γ and α are constants. As a result of this formulation positive ϵ values generate less volatility than negative values - or good news have less effect than bad news. Mathematically, out of two ϵ values that have the same absolute value but different signs, the negative one would cause more increase. Notice how γ in $\gamma \cdot \epsilon_{t-1}/\sqrt{h_{t-1}}$ can now capture the sign of earlier ϵ values, whereas in ARCH and GARCH models this term was squared causing only the amplitude of ϵ to contribute to the model, whether it was negative or positive. Also, EGARCH model assures that "bigger" news have "greater" impact on volatility which GARCH model does not take into account [21].

All ARCH variants with preference being on GARCH and EGARCH have been found quite successful in predicting market volatility. We did not use them in our comparison of prediction methods.

1.6. ANN

ANNs are biologically inspired constructs which are used heavily in robotics, time series analysis, signal filtration, regression and classification. ANNs are built to mimic neurons in human brain which can receive signals through dendrites, and emits them through axons to snyapses, then the signal is transmitted to the next neuron in the chain.



A network with one layer of neurons is called a perceptron. If there is more than one layer available, this network is called a multilayer perceptron (MLP). For the purposes of this thesis, we used an MLP for our time series analysis tasks. It has been proven that an MLP with single hidden layer and sufficient number of neurons can approximate any function. This is an improvement over single layer regular perceptron which has problems with the infamous XOR function.

ANNs are trained with using data sets whose contents include a set of input points $x_1, .., x_N$ along with output data point(s) $y_1, .., y_K$. At each training step, the current state (weights of the neurons) of ANN is used to calculate an error between predicted and real output values, then, the weights of the hidden neurons are modified in order to decrease the error value. This is performed for all data points in the training data set. The error measure J is

$$J = \frac{1}{2N} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Gradient descent with back propagation is a popular method to minimize the error function at each iteration.

For stock price prediction purposes, we utilized an MLP network with 30 neurons in the hidden layer, input size as wide as our training window, one output and linear output function. We used a Matlab® package called Netlab as our ANN package

written by Christopher Bishop and Ian Nabney.

Time series data had to be tailored so it could be fed into a neural network. In applications such as face recognition, flattening out the input matrix into one continous vector $\mathbf{x}$ and mapping this flattened structure to a single classification code is quite natural. With time series, data the mapping of input to outputs is somewhat less intuitive. The trick is using autoregression - having the time series contain both the input and the output at the same time where one data input set $x_1, ..., x_N$ can act as input and x_{N+1} can act as output, next input would be $x_2, ..., x_{N+1}$ as input and x_{N+2} as output and by shifting by one, so on. With this method, at each iteration we essentially are creating a new dataset which contains a set of input/output matchings.

To make predictions, we feed "current data" starting from today back as much as input allows, as a new input collection, and the single output value is then the prediction for tomorrow's stock price. If we need to make prediction for more than one day, we treat each newly generated predicted value as another input, folding it into the same time series sequence as the rest of the input data. Naturally this would cause predictions to worsen as part of the input data starts consisting of not actual data but predicted data, but if the modeler keeps the projected days to a minimum, the algorithm does produce acceptable results.

1.7. Polynomial Regression

Another simple yet powerful method used for comparion was a method that falls under linear regression umbrella, called polynomial regression. With this model, given data for y and x pairs, the relation between them is assumed to be

$$y = \beta_0 + \beta_1 x + ... + \beta_1 x^k + \epsilon$$

In order to fit the model to the data, the modeler first decides the order of the polynomial equation (we used 10). Then, a least squares fit is performed which aims to minimize the error ϵ. At the end this process outputs all β coeffecients to define the equation for y. Matlab's `polyfit` accomplished this task.

During modeling process, we first tried "time" variable as x and obtained less than satisfactory results. Next we decided to divide up the historical stock price data itself in two parts, and have i^{th} element of first part to be x, and i^{th} element of second part to be its matching y, then do this for all elements of first and second parts. On this data set, we could fit the polynomial equation to the data giving us the β values.

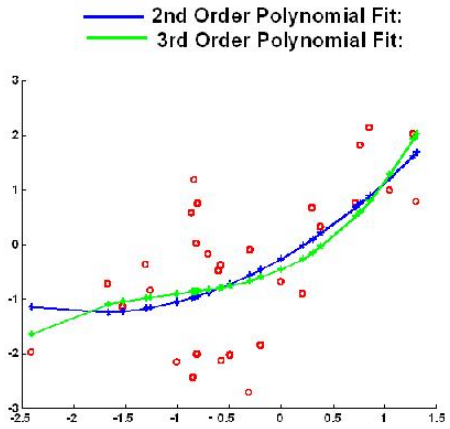


Figure 1.3. Example of Polynomial Fitting

For prediction, if we need to estimate N days in the future, then we go back N days starting from today, hence the historical N points became our new x's, plugging each x into our "learned" polynomial equation would give us future y's.

1.8. KMM

In order to obtain better prediction, we hypothesized a combination of HMM and KF methods could yield different, hopefully better results. We called this method KMM, a combination of words **K**F and H**MM**. There are various methods in the literature to combine more than one predictor, or "experts", for a certain data set. One method is assigning a different expert for each seperate region, training it, then

combining alllocal experts to arrive at a better global predictor.

Another method is combining the output of each expert for each data point. This way, every expert is contributing something to the overall results, although at different weights. Mathematically every data point y_i is a combination of

$$y_i = \sum_j w_j o_{ij}$$

where o_{ij} is prediction from expert j at time i, and w_j is the weight assigned to expert j. The sum of weights for all experts must equal to one, that is, $\sum_j w_j = 1$.

1.9. Previous Work

Using HMM and KF for financial time series prediction is a well known method. As stated in [6] and confirmed by a quick look in to the literature shows that most probabilistic time series models of use today are descendants of either HMMs or state space models (SSMs). There is one approach that combines both HMM and KF methods proposed by Ghahramani and Hinton. They divide the data into multiple regions detected by HMM, then for each segment, a SSM is trained as an "expert" for that region. In other words, for any given data point, SSMs are competing among each other to describe that data point. Once SSMs are trained, the overall flow over those "multiple regions" -multiple SSMs- can be handled by the HMM. There is a similar method called Hidden Markov Experts (HME) proposed by Shi and Weigend [17]. HME also utilizes seperation of regions using HMMs but connects them by using ANNs [19].

Gurgen and Yumlu [1], employed neural mixture of experts (MOE) structure for ISE prediction and compared those results with an ANN, a polynomial and a radial basis function (RBF) nets.

Hassan and Nath [18] use HMMs in a way similar that is used in speech recog-

nition. They decided to train one HMM *per trading day* and used four values of each day in sequence as a mini-time series. Those values were opening price, high price, low price and closing price. The modeler trains one HMM per trading day using the four values and also remembers the next day's closing price. This will become useful for prediction, during which, four values of yesterday are passed to every trained HMM one by one. Then, whichever HMM reports the highest likelihood for those four mini time series points, that HMM is picked as the correct HMM to predict tomorrow. Of course even though the match gives a change pattern that might be similar to values from yesterday, the absolute values of both sides can be different - in which case prediction program simply "adds" the average difference of values from matched HMM to tomorrows predicted stock price. This method [18] predicts only one day ahead.

2. HIDDEN MARKOV MODELS

Hidden Markov Models are based on over hundred years old mathematical construct called Markov Chains (MC). An MC can be in one of distinct N states and the transition between those states occur according to predefined transition probabilities at discrete intervals. For the simplest, first-order Markov Chain, these probabilities can be captured in a `NxN` matrix. The fact that present state is enough to represent a system and is the only factor that determines the future (through the transition table) is also called as the system having the "Markov property". This also means that given present state, the past and future are independent of eachother. Mathematically, a Markov Chain is a sequence of random variables $X_1, X_2, X_3, ...$, the state of the system at time t being x_t, then

$$P(X_{t+1} = x | X_t = x_t, ..., X_1 = x_1) = P(X_{t+1} = x | X_t = x_t)$$

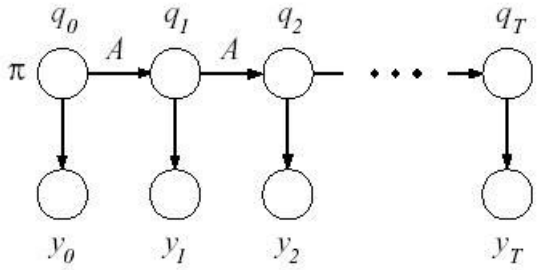
A Markov Chain could also be second order or third order and so on - for example for second order Markov Chains, the transition table is `2N x N` which necessitates both present and *previous* step to obtain a transition probability distribution for the next state.

Markov Chains have no hidden states. The emissions are the states themselves, the transition probability matrix captures all the dynamics of the system. HMMs improve on this model by adding a second layer of probability distribution that represents observation values.

2.1. Structure

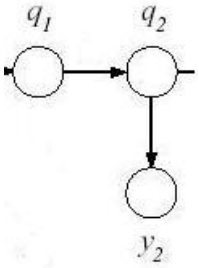
We assume a two level structure, with one level for hidden states, another for observed states. Observed states model the "emissions", the output of the model. In a typical setting where HMMs are used for prediction, modeler needs to supply the number hidden states - an optimal value for this is usually determined by using seperate training sets.

Hidden and observed states have different distributions.



Following the style and derivations by Bishop and Jordan [8], hidden states sequence which correspond to an observation sequence are represented as $q_1, ..., q_T$. The output sequence is represented by random variables $y_1, ..., y_T$ where T represents the number of available data points. It will be assumed that q_t and y_t are i.i.d $\forall t$, that is, as time progresses, the underlying distribution will not change for q_t and y_t. There is a conditional dependence between

- y_t and q_t
- q_t and q_{t-1}



We let random variable q_t to have a multinomial distribution. There are M finite states and for q_t at state i, the"output" of q_t is a vector of size M carrying a "1"

at index i, "0" otherwise. Another representation that can be used is $q_t^i = 1$ where $i \in \{1, .., M\}$ and $p(q_t)$ being a vector of size M where each cell carries the probability value for a possible value of q_t.

There is conditional dependency between y_t and q_t, and q_t and q_{t-1}. Since q_t is multinomial, and there are M states, joint distribution between q_t and q_{t-1} should be represented by a table of size M x M. Let's call this M x M table the transition matrix A. a_{ij} gives transition probability $p(q_t^j = 1|q_t^i = 1)$. Notice A is independent of t.

Data output is captured in y_t. Output probability is $p(y_t|q_t)$. We make no assumptions on this distribution for now.

One of our goals with HMM will be training it with "given" emissions (output) $y_1, .., y_T$ to come up with matrix A which can also be represented as $p(q_t^j = 1|q_t^i = 1)$. Another goal of HMM use is being able to compute a likelihood - given a matrix A and observation sequence y we should be able to compute the likelihood of a "hidden" state sequence q, more succintly $p(q|y)$.

It's easier to compute $p(q_t|y)$ instead of entire q. We can use $p(y|q_t)$ to get to $p(q_t|y)$.

$$p(q_t|y) = \frac{p(y|q_t)p(q_t)}{p(y)}$$

Then, we need to calculate $p(y|q_t)$ and $p(y)$. Trick: Conditioning rule states given q_t,

- q_{t-1} becomes independent of q_{t+1}
- y_{t-1} becomes independent of y_{t+1}.

$$\begin{aligned} p(q_t|y) &= \frac{p(y|q_t)p(q_t)}{p(y)} \\ &= \frac{p(y_0...y_t|q_t)p(y_{t+1}...y_T|q_t)p(q_t)}{p(y)} \\ &= \frac{p(y_0...y_t, q_t)p(y_{t+1}...y_T|q_t)}{p(y)} \\ p(q_t|y) &= \frac{\alpha(q_t)\beta(q_t)}{p(y)} \end{aligned}$$

where

$$\begin{aligned} \alpha(q_t) &\triangleq p(y_0...y_t, q_t) \\ \beta(q_t) &\triangleq p(y_{t+1}...y_T|q_t) \end{aligned}$$

Obtain $p(y)$ by applying $\sum_{q_t}$ on both sides

$$\begin{aligned} \sum_{q_t} p(q_t|y) &= \sum_{q_t} \frac{\alpha(q_t)\beta(q_t)}{p(y)} \\ 1 &= \sum_{q_t} \frac{\alpha(q_t)\beta(q_t)}{p(y)} \\ p(y) &= \sum_{q_t} \alpha(q_t)\beta(q_t) \end{aligned} \tag{2.1}$$

2.2. Forward-Backward Recursion

We are finished with $p(y)$. We can move on to $\alpha(q_t)$ and $\beta(q_t)$. We will compute these recursively. That is, from $\alpha(q_t)$ we will compute $\alpha(q_{t+1})$.

$$
\begin{aligned}
\alpha(q_{t+1}) &= p(y_0, ..., y_{t+1}, q_{t+1}) \\
&= p(y_0, ..., y_{t+1}|q_{t+1})p(q_{t+1}) \\
&= p(y_0, ..., y_t|q_{t+1})p(y_{t+1}|q_{t+1})p(q_{t+1}) \\
&= p(y_0, ..., y_t, q_{t+1})p(y_{t+1}|q_{t+1}) \\
&= \sum_{q_t} p(y_0, ..., y_t, q_t, q_{t+1})p(y_{t+1}|q_{t+1}) \\
&= \sum_{q_t} p(y_0, ..., y_t, q_{t+1}|q_t)p(q_t)p(y_{t+1}|q_{t+1}) \\
&= \sum_{q_t} p(y_0, ..., y_t|q_t)p(q_{t+1}|q_t)p(q_t)p(y_{t+1}|q_{t+1}) \\
&= \sum_{q_t} p(y_0, ..., y_t, q_t)p(q_{t+1}|q_t)p(y_{t+1}|q_{t+1}) \\
&= \sum_{q_t} \alpha(q_t)a_{q_t,q_{t+1}}p(y_{t+1}|q_{t+1})
\end{aligned}
$$

Similarly we can derive the equation for $\beta(q_t)$.

$$
\begin{aligned}
\beta(q_t) &= p(y_{t+1}, ..., y_T|q_t) \\
&= \sum_{q_{t+1}} p(y_{t+1}, ..., y_T, q_{t+1}|q_t) \\
&= \sum_{q_{t+1}} \frac{p(y_{t+1}, ..., y_T, q_{t+1}, q_t)}{p(q_t)} \\
&= \sum_{q_{t+1}} \frac{p(y_{t+1}, ..., y_T, q_t|q_{t+1})p(q_{t+1})}{p(q_t)} \\
&= \sum_{q_{t+1}} p(y_{t+1}, ..., y_T, q_t|q_{t+1})p(q_{t+1}|q_t) \\
&= \sum_{q_{t+1}} p(y_{t+2}, ..., y_T, q_t|q_{t+1})p(q_{t+1}|q_t)p(y_{t+1}|q_{t+1}) \\
&= \sum_{q_{t+1}} \beta(q_{t+1})a_{q_t,q_{t+1}}p(y_{t+1}|q_{t+1})
\end{aligned}
$$

We see that

- α recursion is a forward recursion (we plug in q_t to get q_{t+1})
- β recursion is a backward recursion (we plug in q_{t+1} to get q_t)

The start-up equation can be calculated as follows:

$$\begin{aligned}\alpha(q_0) &= p(y_0, q_0) \\ &= p(y_0|q_0)p(q_0) \\ &= p(y_0|q_0)\pi_0\end{aligned}$$

There is a little complication for the start-up formula for β recursion. In this case it would not be possible to start with $\beta(q_{T+1})$ because that state does not exist in our system. In this case, we can simply start with vector of ones. For the proof the reader can refer to [8].

2.3. Training

We need to find optimal θ that maximizes $log(p(y, q|\theta))$ where $\theta = (\pi, A, \eta)$. We know π and A. η represents parameters necessary for $p(y_t|q_t)$, the output distribution.

Let's write the full $p(y, q)$ distribution;

$$p(q, y) = \pi_{q_0} \prod_{t=0}^{T-1} a_{q_t, q_{t+1}} \prod_{t=0}^{'} p(y_t|q_t, \eta)$$

We take log, use Lagrange multipliers, take derivative set to zero and solve:

$$\hat{a}_{ij} = \frac{\sum_{t=0}^{T-1} q_t^i q_{t+1}^j}{\sum_{j=1}^{M} \sum_{t=0}^{T-1} q_t^i q_{t+1}^j}$$

Similarly for η_{ij} and π_i,

$$\begin{aligned}\hat{\eta}_{ij} &= \frac{\sum_{t=0}^{T-1} q_t^i y_t^j}{\sum_{k=1}^{N} \sum_{t=0}^{T-1} q_t^i y_t^k} \\ \hat{\pi}_i &= q_0^i\end{aligned}$$

$\hat{\eta}_{ij}$ assumes a multinomial output distribution y_t. For stock prediction purposes, we will be using a Gaussian distribution to represent each stock price at time t. Therefore, we can simply use y_t itself as the output instead of the y_t^j, hence $\hat{\eta}$ changes to being the mean of the estimated Gaussian.

$$\hat{C}_i^{(p+1)} = \frac{\sum_{t=0}^{T} \gamma_t^i y_t}{\sum_{t=0}^{T} \gamma_t^i}$$

where γ_t is the posterior probability $p(q_t|y)$ at time t.

In each iteration

- We calculate $\hat{a}$, $\hat{C}$, $\hat{R}$, $\hat{\pi}$ using given q and y.
- We calculate $p(q_t|y)$ for each q_t using A, η and π using forward backward recursion.

2.4. Algorithm

The algorithm for HMM forward-backward inference and training through EM (Baum-Welch) as presented by Roweis and Ghahramani [3] can be found below. The operator $\odot$ is defined to be element-wise product of two equal size vectors or matrices. The algorithm assumes the emission distribution is a multidimensional Gaussian.

`HMMInference`(Y,A,π,C,R)
`for` t=1 to T % forward pass
    `for` i=1 to k
        $b_{t,i} = \texttt{exp}\{-\frac{1}{2}(y_t - C_i)'R^{-1}(y_t - C_i)\}|R|^{-1/2}(2\pi)^{-p/2}$
    `end`
    if (t=1) $\alpha_t = \pi \odot b_1$ else $\alpha_t = [A'\alpha_{t-1}] \odot b_t$ end
    $\rho_t = \sum_i \alpha_{t,i}$
    $\alpha_t = \alpha_t/\rho_t$ % normalization

`end`

$\beta_T = 1/\rho_t$

`for` t=T-1 to 1 % backward pass

$\beta_t = A[\beta_{t+1} \odot b_{t+1}]/\rho_t$

`end`

$\gamma_t = \alpha_t \odot \beta_t/(\alpha_t'\beta_t)$

$\xi_t = \alpha_t[\beta_{t+1} \odot b_{t+1}]'$

$\xi_t = \xi_t/(\sum_{ij} \xi_{ij})$

`return` γ_t, ξ_t, ρ_t for all t

`HMMLearn`(Y,k,ϵ)

`initialize` A, π, C, R

`while change in log likelihood` $> \epsilon$

`HMMInference`(Y, A, π, C, R); %E step

% M Step

$\pi = \gamma_1$

$A = \sum_t \xi_t$

$A_{ij} = A_{ij}/\sum_l A_{il}$ `for all` i, j

$C_j = \sum_t \gamma_{t,j} y_t / \sum_t \gamma_{t,j}$ `for all` j

$R = \sum_{t,j} \gamma_{t,j}(y_t - C_j)(y_t - C_j)'/T$

`return` A, π, C, R

2.5. Prediction with HMM

For HMMs, if we start generating using π, this would give us a replay of the training data starting with day 1, not future data starting at $T + 1$. Then, we should not start generating starting with π, instead, we need to determine *the last hidden state of our HMM* on day T, and use transition table A to get to hidden state at day $T + 1$, *then* we can start generating starting

In order to find the hidden state at day T, we can simply run the Viterbi algorithm

that gives us the most likely hidden path for the given data and given HMM. Then, we simply pick the last element from this list, throw the dice and apply transition once, and this gives us our first prediction for day $T+1$. The rest will be the same as generating data out of a regular HMM - we keep following the transition table. The pseucode for the function `observationGeneration` looks as follows:

```
observationGeneration(A, π, C, R, N)
    states = viterbi(A, π, C, R);
    lastState = states[T];
     for i = 1 to  N
        currentState = nextState(lastState, A, π, C, R);
        mean = C[currentState];
        variance = R[currentState];
        o[i] = gaussianRandomGeneration(mean, variance);
        lastState = currentState;
    end
    return o;


nextState(state, A, π, C, R);
    randVal =generateUniform(0,1);
    total = 0;
    done = false;
    total = A[currentState][0];
    for  j = 1 to  numStates
       if done = true break;
       if (randVal > total and
            randVal ⩽ (total + A[currentState][j]))
            currentState = j;
            done = true;
       end
       total = total + A[currentState][j];
```

```
end
return currentState;
```

3. KALMAN FILTERS

Compiling an exhaustive list of fields where Kalman Filters (KF) are utilized would be a formiddable task: The list could start with navigation, radar tracking, satellite orbit computation [11], continue with sonar ranging, stock price prediction (which hopefully this thesis will prove), and many other fields where there is a dynamical system present. When the *Eagle* module of the Apollo 11 mission landed on the moon, it is widely known that it did so with the help of a Kalman Filter. KF was first proposed in a seminal paper by [9], later Kalman and Bucy [10] - the method's first application setting was space tracking.

Kalman Filters, or in general, SSM methods try to model dynamic data using two-level graphic model. In the first level, there are hidden states, one per time step, which are distributed as Gaussians. An outside observer (modeler) does not see these states directly, instead sees them through an "observed" variable whose outputs are assumed to be distorted by another Gaussian distribution, refered to as white noise. In each time step, it is assumed the hidden variables evolve linearly, in linear algebra this means the hidden state is multiplied by a constant matrix A. Mathematically,

$$x_{t+1} \quad = \quad Ax_t + v_t$$

where

- $v_0, v_1, ...$ is white Gaussian noise with $\mathbf{E}v_t = 0$ and $cov\ v_t = Q$
- $x_t \sim N(\mu_t, \Sigma_t)$.

In order to model the output variables, we multiply hidden states with another constant matrix and add another Gaussian white noise. All together our SSM looks as follows:

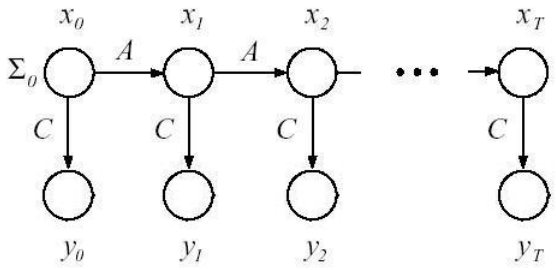
$$x_{t+1} = Ax_t + v_t \tag{3.1}$$

$$y_t = Cx_t + w_t \tag{3.2}$$

where $w_0, w_1, ...$ are white Gaussian noise with $\mathbf{E}w_t = 0$ and $cov\ w_t = R$. Random variables w_t and v_t are independent of x_t and y_t, and from eachother, this assumption is put forth for simplicity. The dimensions of each variable are:

- $x(t) \in \mathbb{R}^n$ is the state
- $y(t) \in \mathbb{R}^p$ is the observed output
- $v(t) \in \mathbb{R}^n$ is process noise
- $w(t) \in \mathbb{R}^p$ is measurement noise

Kalman Filters carry the same conditional independence properties that characterized HMMs - given a state for time t, the future is conditionally independent from the past, hence KF also carries the first-order Markovian property like HMMs. Structurally there are many similarities between two models and its methods for inference, as we shall soon see, the main difference being that HMM uses multinomial distribution for hidden states whereas KF uses Gaussians.



Inference with Kalman Filters means estimating the posterior probability given an observed output sequence. There are two tasks here, filtering and smoothing. Filtering is formulized by $p(x_t|y_0, ..., y_t)$ which tries to calculate posterior probability for hidden states using measurements up to time t. In smoothing, we will calculate the same probability using measurements taken after time t, which gives us $p(x_t|y_{t+1}, ..., y_T)$. We also hope to formulize recursive formulas for both filtering and smoothing.

Both filtering and smoothing have direct consequences and use in terms of real-life applications. KFs' natural roots are embedded in space tracking, the method started as an "online" algorithm trying to track down targets using noisy measurements, as

each new measurement is received. In this sense, KF can be seen as an estimator trying to guess a "hidden" position at time t using measurements up to time t. Smoothing, on the other hand, can be thought of a global "corrector" - once we have better knowledge on hand, through smoothing we can go back to old estimations and correct previous online estimates using backward recursion.

3.1. Filtering

Following the notation and the derivation steps used by Bishop and Jordan [8], let's first denote

- $\hat{x}_t^t \triangleq \mathbf{E}[x_t|y_0, ..., y_t]$
- $P_t^t \triangleq \mathbf{E}[(x_t - \hat{x}_{t|t})(x_t - \hat{x}_{t|t})'|y_0, ..., y_t]$

Our first goal is going to be trying to calculate $\hat{x}_{t+1}^{t+1}$ and P_{t+1}^{t+1} using a new measurement y_{t+1}. For this, we need to "reverse the arrow" so to speak, in other words formulize the distribution where x_t has conditional dependence on y_t instead of other way around, which is what the base equations state right now.

If we take expectation of both sides of (3.1)

$$\mathbf{E}x_{t+1} = \hat{x}_{t+1} = A\mu_t = A\hat{x}_t$$

Taking covariance of both sides of (3.1) and denoting P_t as $cov\ x(t)$

$$P_{t+1} = AP_tA' + Q$$

This transition is called a "time update". It will allow us to propagate the normal distribution at time t to time t_1. The formulas that contain given y's use the same

technique;

$$\hat{x}_{t+1}^{t} = Ax_t^t$$
$$P_{t+1}^{t} = AP_t^t A' + Q$$

$$\begin{aligned} y_{t+1} &= Cx_{t+1} + w_t \\ \mathbf{E}[y_{t+1}|y_0, ..., y_t] &= \mathbf{E}[Cx_{t+1} + w_t|y_0, ..., y_t] \\ \hat{y}_{t+1}^{t} &= C\hat{x}_{t+1} \end{aligned}$$

Similarly for covariance

$$E[(y_{t+1} - \hat{y}_{t+1}^{t})(y_{t+1} - \hat{y}_{t+1}^{t})'|y_0, ..., y_t] = CP_{t+1}^{t}C' + R$$

Now the harder task of reversing the arrow: If our goal is formulizing $p(x_t|y_t)$, then we need to derive the joint distribution between both variables. Addition of Gaussians is another Gaussian, we can surmize the joint probability $p(x_t, y_t)$ will be one huge Gaussian.

In order to define the joint normal distribution of x_t and y_t, we need to write the joint mean and covariance of this big Gaussian. Writing down the mean will be easy, covariance will be little harder to derive. We can use a trick [12] where we write $y_t = Cx_t + w_t$ as

$$z = Hu$$

where

$$z = \begin{bmatrix} x_t \\ y_t \end{bmatrix}, H = \begin{bmatrix} I & 0 \\ C & I \end{bmatrix} u = \begin{bmatrix} x_t \\ w_t \end{bmatrix},$$

Now we need to take the covariance of the simpler equation:

$$\begin{aligned} cov(z) &= Hcov(u)H' \\ cov(u) &= \begin{bmatrix} P_t & 0 \\ 0 & R \end{bmatrix} \end{aligned}$$

The full product is:

$$\begin{bmatrix} I & 0 \\ C & I \end{bmatrix} \begin{bmatrix} P_t & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} I & C' \\ 0 & I \end{bmatrix}$$

which results in

$$\begin{bmatrix} P_t & P_t C' \\ CP_t & CP_t C' + R \end{bmatrix}$$

We can also re-write this for conditional and with the mean;

$$\begin{bmatrix} \hat{x}_t^t \\ C\hat{x}_t^t \end{bmatrix} and \begin{bmatrix} P_t^t & P_t^t C' \\ CP_t^t & CP_t^t C' + R \end{bmatrix} \tag{3.3}$$

Same for the joint distribution for x_{t+1}, y_{t+1}.

$$\begin{bmatrix} \hat{x}_{t+1}^t \\ C\hat{x}_{t+1}^t \end{bmatrix} and \begin{bmatrix} P_{t+1}^t & P_{t+1}^t C' \\ CP_{t+1}^t & CP_{t+1}^t C' + R \end{bmatrix} \tag{3.4}$$

Now in order to get statements for mean and variance for x_{t+1}^{t+1}, we need to understand partitioned Gaussians. This will guide us when we use portions of (3.4) and

deriving the final equation [8].

An n dimensional Gaussian distribution can be partitioned into p and q dimensional sub-distributions where $n = p + q$. Hence we can say,

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \tag{3.5}$$

$$\begin{aligned} p(x|\mu, \Sigma) &= \frac{1}{(2\pi)^{(p+q)/2}|\Sigma|^{1/2}} \\ & exp\Bigg\{ -\frac{1}{2} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix}' \Bigg\} \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}^{-1} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix} \Bigg\} \end{aligned}$$

After much algebra, we can obtain equation for $p(x_1|x_2)$. From this, we can get *conditioned* μ and Σ, as

$$\begin{aligned} \mu_{1|2} &= \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2) \\ \Sigma_{1|2} &= \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} \end{aligned} \tag{3.6}$$

Now plugging elements from (3.6) into (3.4) according to placement in (3.5) we can formulize $\hat{x}_{t+1}^{t+1}$ and P_{t+1}^{t+1}.

$$\begin{aligned} \hat{x}_{t+1}^{t+1} &= \hat{x}_{t+1}^{t} + P_{t+1}^{t}C'(CP_{t+1}^{t}C' + \Sigma_w)^{-1} \\ & \quad (y_{t+1} - C\hat{x}_{t+1}^{t}) \\ P_{t+1}^{t+1} &= P_{t+1}^{t} - P_{t+1}^{t}C'(CP_{t+1}^{t}C' + R)^{-1}CP_{t+1}^{t} \end{aligned}$$

If we declare $K_t \triangleq P_{t+1}^t C'(CP_{t+1}^t C' + \Sigma_w)^{-1}$ then,

$$\hat{x}_{t+1}^{t+1} = \hat{x}_{t+1}^t + K_t(y_{t+1} - C\hat{x}_{t+1}^t) \tag{3.7}$$

$$P_{t+1}^{t+1} = P_{t+1}^t - K_t C P_{t+1}^t \tag{3.8}$$

3.2. Smoothing

Filtering involves applying a new measurement y_{t+1} to calculate $\hat{x}_{t+1}^{t+1}$ and P_{t+1}^{t+1} which is an excellent tool for an online algorithm. However for stock price prediction, we will have all data we need already at hand. As opposed to a tracking application where data cannot be known in advance, we will have our data available, as measurements $y_1, ..., y_T$ in the form of stock prices, to train a Kalman filter based solution, all at once. Hence we need an equation for x_t'. For this we need RTS (Rauch-Tung-Striebel) Smoother.

Based on (3.3), we can write the covariance of joint distribution (x_t, x_{t+1}). In (3.3) the multiplication was with C, here it's A. Hence,

$$\begin{bmatrix} P_t^t & P_t^t A' \\ AP_t^t & AP_t^t A' + R \end{bmatrix}$$

Combining the previous equations together, we need to reach to a statement for x_t'. We can start with $\mathbf{E}[x_t | x_{t+1}, y_0, ..., y_t)]$.

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$$

which becomes

$$\mathbf{E}[x_t | x_{t+1}, y_0, ..., y_t)] = \hat{x}_t^t + P_t^t A'(P_{t+1}^t)^{-1}(x_{t+1} - \hat{x}_{t+1}^t)$$

Define $L_t \triangleq P_t^t A'(P_{t+1}^t)^{-1}$. Then,

$$\mathbf{E}[x_t|x_{t+1}, y_0, ..., y_t)] = \hat{x}_t^t + L_t(x_{t+1} - \hat{x}_{t+1}^t)$$

How about $Var[x_t|x_{t+1}, y_0, ..., y_t]$? If we remember the joint covariance for (x_t, x_{t+1})

$$\begin{bmatrix} P_t^t & P_t^t A' \\ AP_t^t & AP_t^t A' + R \end{bmatrix}$$

Then

$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

becomes

$$Var[x_t|x_{t+1}, y_0, ..., y_t] = P_t^t - P_t^t A'(P_{t+1}^t)^{-1} AP_t^t$$

Why?

$$\begin{bmatrix} P_t^t & P_t^t A' \\ AP_t^t & AP_t^t A' + R \end{bmatrix} = \begin{bmatrix} P_t^t & P_t^t A' \\ AP_t^t & P_{t+1}^t \end{bmatrix}$$

If we try to substitute $L_t \triangleq P_t^t A'(P_{t+1}^t)^{-1}$ in to the equation:

$$\begin{aligned} Var[x_t|x_{t+1}, y_0, ..., y_t] &= P_t^t - P_t^t A'(P_{t+1}^t)^{-1} AP_t^t \\ &= P_t^t - P_t^t A'(P_{t+1}^t)^{-1}(P_{t+1}^t(P_{t+1}^t)^{-1}) AP_t^t \\ &= P_t^t - (P_t^t A'(P_{t+1}^t)^{-1}) P_{t+1}^t((P_{t+1}^t)^{-1} AP_t^t) \\ &= P_t^t - L_t P_{t+1}^t L_t' \end{aligned}$$

Due to conditional independence, when we condition on x_{t+1}, we make x_t independent

of future observations $y_{t+1}, ..., y_T$. Hence we can simply say;

$$\begin{aligned} E[x_t|x_{t+1}, y_0, ..., y_T] &= E[x_t|x_{t+1}, y_0, ..., y_t] \\ &= \hat{x}_t^t + L_t(x_{t+1} - \hat{x}_{t+1}^t) \\ Var[x_t|x_{t+1}, y_0, ..., y_T] &= Var[x_t|x_{t+1}, y_0, ..., y_t] \\ &= P_t^t - L_t P_{t+1}^t L_t' \end{aligned}$$

We are almost done. If we can drop the x_{t+1} from LHS of both equations, we can have our smoothing equations. Due to the so-called Tower Property:

$$\begin{aligned} \mathbf{E}[X|Z] &= \mathbf{E}[\mathbf{E}[X|Y, Z]|Z] \\ Var[X|Z] &= Var[\mathbf{E}[X|Y, Z]|Z] + \mathbf{E}[Var[X|Y, Z]|Z] \end{aligned}$$

we can drop x_{t+1} from LHS.

$$\begin{aligned} \hat{x}_t' &\triangleq \mathbf{E}[x_t|y_0, ..., y_T] \\ &= \mathbf{E}[\mathbf{E}[x_t|x_{t+1}, y_0, ..., y_T]|y_0, ..., y_T] \\ &= \mathbf{E}[\hat{x}_t^t + L_t(x_{t+1} - \hat{x}_{t+1}^t)] \\ &= \hat{x}_t^t + L_t(\hat{x}_{t+1}' - \hat{x}_{t+1}^t) \end{aligned} \tag{3.9}$$

Using similar algebra, we get

$$P_t' = P_t^t + L_t(P_{t+1}' - P_{t+1}^t)L_t'$$

3.3. Training

In order to train a Kalman Filter based predictor, we need to estimate parameters A, C, Q, R, π. These parameters define a KF.

Making use of Markov property, we write our joint probability as;

$$p(x_0,..,x_t,y_0,..,y_t) = p(x_0)\prod_{t=1}^{T-1} p(x_{t+1}|x_t)\prod_{t=0}^{T-1} p(y_t|x_t)$$

$$\begin{aligned}
p(y_t|x_t) &= exp\Big\{ -\frac{1}{2}(y_t - Cx_t)'R^{-1}(y_t - Cx_t)\Big\}(2\pi)^{-\frac{p}{2}}|R|^{-\frac{1}{2}} \\
p(x_{t+1}|x_t) &= exp\Big\{ -\frac{1}{2}(x_{t+1} - Ax_t)'Q^{-1}(x_{t+1} - Ax_t)\Big\} \\
&\quad (2\pi)^{-\frac{k}{2}}|Q|^{-\frac{1}{2}} \\
p(x_1) &= exp\Big\{ -\frac{1}{2}(x_1 - \pi)'\Sigma_\pi^{-1}(x_1 - \pi)\Big\}(2\pi)^{-\frac{k}{2}}|\Sigma_\pi|^{-\frac{1}{2}}
\end{aligned}$$

We take log

$$\begin{aligned}
\log p(x,y) = \; & -\sum_{t=0}^{T-1}(-\frac{1}{2}(y_t - Cx_t)'R^{-1}(y_t - Cx_t)) - \frac{T}{2}\log|R| \\
& -\sum_{t=1}^{T-1}(-\frac{1}{2}(x_{t+1} - Ax_t)'Q^{-1}(x_{t+1} - Ax_t)) \\
& -\frac{T-1}{2}\log|Q| \\
& -\frac{1}{2}(x_1 - \pi)'\Sigma_\pi^{-1}(x_1 - \pi) - \frac{T(p+k)}{2}\log|\Sigma_\pi|
\end{aligned}$$

In order to estimate parameters, we take derivative of expected log likelihood, set to zero and solve

$$\begin{aligned}
Q &= \mathbf{E}[\log p(x,y|y)] \\
\frac{\partial Q}{\partial C} &= -\sum_{t=1}^{T} R^{-1}y_t\hat{x}_t' + \sum_{t=1}^{'} R^{-1}CP_t = 0
\end{aligned}$$

$$C^{new} = \left(\sum_{t=1}^{'} y_t \hat{x}_t'\right)\left(P_t'\right)^{-1}$$

Using similar technique, we can calculate other constants as;

$$\begin{aligned} R^{new} &= \frac{1}{T}\sum_{t=1}^{T}\left(y_t y_t' - C^{new}\hat{x}_t y_t'\right) \\ A^{new} &= \left(\sum_{t=2}^{T} P_{t,t-1}\right)\left(\sum_{t=2}^{T} P_{t-1}\right)^{-1} \\ Q^{new} &= \frac{1}{T-1}\left(\sum_{t=2}^{T} P_t - A^{new}\sum_{t=2}^{T} P_{t-1,t}\right) \\ \pi_1^{new} &= \hat{x}_1 \end{aligned}$$

3.4. Usage

While using Kalman Filters for analysis and prediction purposes, one needs to be careful about starting condition of the training algorithm. Because of the nature of the training algorithm, the parameters $\mu_i, \sigma_i, A, C, Q, R, P$ are all used for likelihood calculation/estimation and are *themselves* targets for estimation. The "chicken and egg" nature of the EM training process required us to start somewhere, and that somewhere is picking appropiate parameters for the parameters mentioned above.

We had to determine said values by trial and error, as part of the process where we were trying to determine meta parameters such as number of iterations required for MC process, the lowest likelihood value threshold (which was used to determine when to stop in the training process - if likelihood difference fell below this threshold value, we would conclude that training was complete).

3.5. Algorithm

The algorithm that applies forward-backward algorithm and learning through EM is described below [3].

LDSInference(Y, A, C, Q, R, x, P)

for $t = 1$ to T % forward recursion

$x_t^{t-1} = Ax_{t-1}^{t-1}$

$P_t^{t-1} = AP_{t-1}^{t-1}A' + Q$ if $t > 1$

$K_t = P_t^{t-1}C'(CP_t^{t-1}C' + R)^{-1}$

$x_t^t = x_t^{t-1} + K_t(y_t - Cx_t^{t-1})$

$P_t^t = P_t^{t-1} - K_tCP_t^{t-1}$

end

initialize $\hat{P}_{T,T-1} = (I - K_TC)AP_{T-1}^{T-1}$

for $t = T$ to 2 % backward recursion

$J_{t-1} = P_{t-1}^{t-1}A'(P_t^{t-1})^{-1}$

$\hat{x}_{t-1} = x_{t-1}^{t-1} + J_{t-1}(\hat{x}_t - Ax_{t-1}^{t-1})$

$\hat{P}_{t-1} = P_{t-1}^{t-1} + J_{t-1}(\hat{P}_t - P_t^{t-1})J_{t-1}'$

$\hat{P}_{t,t-1} = P_t^tJ_{t-1}' + J_t(\hat{P}_{t+1,t} - AV_t^t)J_{t-1}'$ if $t < T$

end

return $\hat{x}_t, \hat{P}_t, \hat{P}_{t,t-1}$ for all t

LDSLearn(Y, k, ϵ)

initialize A, C, Q, R, x_1^0, P_1^0

$\alpha = \sum_t y_t y_t'$

while change in likelihood > ϵ

LDSInference$(Y, A, C, Q, R, x_1^0, P_1^0)$ % E-step

initialize $\delta = 0, \gamma = 0, \sigma = 0$

for $t = 1$ to T

$\delta = \delta + y_t\hat{x}_t$

$\gamma = \gamma + \hat{x}_t\hat{x}_t' + \hat{P}_t$

$\beta = \beta + \hat{x}_t \hat{x}'_{t-1} + P_{t,t-1}$ `if` $t > 1$

`end`

$\gamma_1 = \gamma - \hat{x}_T \hat{x}'_T - \hat{P}_T$

$\gamma_2 = \gamma - \hat{x}_1 \hat{x}'_1 - \hat{P}_1$

% M-step

$C = \delta \gamma^{-1}$

$R = (\gamma - C\delta')/T$

$A = \beta \gamma_1^{-1}$

$Q = (\gamma_2 - A\beta')/(T-1)$

$x_1^0 = \hat{x}_1$

$P_1^0 = \hat{P}_1$

`end`

`return` A, C, Q, R, x_1^0, P_1^0

3.6. Prediction with Kalman Filters

Generating future values with a KF model is somewhat easier than generating random values from an HMM. We know state space model assigns one hidden Gaussian distribution per data point, which itself is modeled by another Gaussian. In comparison, the number of hidden states of an HMM is specified by the modeler and is usually less than the number of data points used for training.

The fact that we have as many hidden/observed random variables as data points makes the following more intuitive: After the training phase of the KF which includes forward recursion and smoothing, we have estimates for the mean and variance for all Gaussian distributions. Another knowledge we have at hand is the basic time transition equation for KFs - if we have an estimate at t for hidden state, we know how to propagate those estimates to the future at $t+1$. Therefore, once training is complete, we can simply fast forward to the emission Gaussian at time T, and using propagation equation generate a new observation Gaussian at time $T+1$ which we use to generate a random number that represents "generatively" an output at time $T+1$.

One point we have to be careful with is propagating the emission Gaussian, not simply the hidden Gaussian. In the literature the basic time transition equation is for the latter not the former, hence we need to expand the equation for y_t by taking time transition into account:

Given,

$$x_{t+1} = Ax_t + v_t \tag{3.10}$$

$$y_t = Cx_t + w_t \tag{3.11}$$

where $v_0, v_1, ...$ is white Gaussian noise with $\mathbf{E}v_t = 0$ and $cov\, v_t = Q$ and $w_0, w_1, ...$ are also white Gaussian noise with $\mathbf{E}w_t = 0$ and $cov\, w_t = R$, we can determine y_{t+1} as

$$\begin{aligned} y_{t+1} &= Cx_{t+1} + w_{t+1} && (3.12)\\ &= C(Ax_t + v_t) + w_{t+1} && (3.13)\\ &= CAx_t + Cv_t + w_{t+1} && (3.14) \end{aligned}$$

which gives us the distribution of the emission Gaussian y_{t+1}. The conditional expectation and variance of this distribution given $y_{1,..,t}$ is

$$\begin{aligned} y_{t+1} &= Cx_{t+1} + w_{t+1} && (3.15)\\ &= C(Ax_t + v_t) + w_{t+1} && (3.16)\\ &= CAx_t + Cv_t + w_{t+1} && (3.17)\\ \hat{y}_{t+1|t} &= CA\hat{x}_{t|t} && (3.18)\\ cov(y_{t+1}|y_{0,...,t}) &= CAP_{t|t}A'C' + CQC' + R && (3.19) \end{aligned}$$

We can apply this recursive formula to have our KF generate M many random observations starting at T up to $T + N$. The algorithm is

`observationGeneration`$(A, C, R, Q, \mu, \Sigma, T, N)$

$mu_x = mu[T-1];$

$sigma_x = sigma[T-1];$

`for` $i = 1$ `to` N

$mu_y = CAmu_x;$

$sigma_y = CAsigma_xA'C' + CQC' + R;$

$o[i] =$ `gaussianNumberGeneration`$(mu_y, sigma_y);$

$mu_x = Amu_x$

$sigma_x = Asigma_xA + Q$

`end`

`return` $o;$

4. RESULTS

For both HMM and KF methods, all training points are taken in one continuous sequence, and training is performed on a window of size T. Once training is completed, predictions can then be made for the next N days starting right after the training window finished. The important question at this point is "what is the exact prediction mechanism for both HMM and KF"?

When we train an HMM or KF over a continuous batch of time series data, what we actually achieve is compressing that data into a shorter, more compact form. In a sense the data shrinks and the resulting model gives us a summary of what the data contains. Once this is completed and let us say we wanted to re-generate data that was used to train the learner, theoretically this should be possible - for instance for HMMs, starting with the initial probabilities π, one can "roll the dice" based on π and transition matrix A, we can choose a transition, follow that to the next hidden state, then we throw dice again which lets us follow a transition probability contained in A, so on. This lets us to run a simulation of the data we used to train our HMM. The real data could have contained 1000 points, however with a trained HMM, all we really need is π, A, η. The rest is random number generation using the HMM. However for generating *future* values, we need to modify this method a little further.

4.1. Monte Carlo Simulation

We found that executing HMM or KF forward prediction "once" did not give optimal results. The reason for the bad performance was hypothized to be related to pseudorandom number generation - our method, using HMM or KF relies on pseudo-random number generation over gaussian and uniform distributions. Especially when a prediction is performed for more than 5 days, it was observed that the results would degenerate very rapidly. In order to offset this issue, we decided that using Monte Carlo (MC) method could alleviate these problems.

Monte Carlo method involves the use of pseudorandom numbers to generate results, and each experiment is repeated many times to obtain better accuracy and approximation. For HMM and KF pseudorandom number generation, we chose to follow this method, then, in each case, the forward prediction which utilizes pseudorandom number generation was repeated for over 500 times. After each run the predicted values of each iteration were summed and at the end, averaged for a final list of predicted values. The iteration numbers were determined during training phase where many different parameters were tested for optimal results. We observed that the results improved considerably by this method.

For the generation of pseudorandom numbers, we employ Java ® language's internal random number generator. However, an important rule of MC simulation is that the experiments need to be replicable - that is, each time the same experiment is executed, the pseudorandom generation should be exactly the same as the previous run. The default mode of Java pseudorandom number generation takes the current time as its seed, we had to modify this by setting the seed ourselves manually. A hex value large enough such as `0x5DAE6EE6DL` yielded good results.

4.2. Results

For our testing we used ISE-100 Index historical data between dates 30/10/1987 and 29/1/2008, for DJI between dates 02/02/1970 and 02/02/2007. Two thirds of the data was used for training (to adjust model parameters) then the rest of the data was used for testing final results.

We picked single Gaussian as our HMM's emission distribution and determined 7 to be the number of hidden states. All HMM, KF and KMM methods used a single dimension to model their input data. Important parameters used for all methods are presented below:

Method	**Parameter**	**Value**
HMM	# of hidden states	7
ANN - ISE	# of inputs	1700
ANN - DJI	# of inputs	3000
ANN	# of output	1
ANN	# of Neurons in Hidden Layer	30
Polynomial	Degree	10
ANN - ISE and DJI	Size of Training Window	300
All (except ANN) - ISE	Size of Training Window	1700
All (except ANN) - DJI	Size of Training Window	3000

Kalman/HMM/KMM testing involved a runner program that was able to divide the data into appropiate pieces, forming one test run, executing it and collecting the results. In order to increase the number of test runs, we decided to form a "shifting window" in the data, running HMM/Kalman/KMM training and prediction on this window. The training window was shifted 30 times for each run. The runner starts with a section of data W wide, trains the model, predicts the next 1,5 and 10 days and compares the results against actual stock price data. While comparing predicted vs. actual data, the runner program collects various metrics. A short description of these metrics can be found below;

4.2.1. Metrics

MSE Ror: This metric's long name is mean square error for rate of returns. In general we can say in our application we collect three types of metrics: Prediction error on rate of returns, prediction error on absolute prices and hit rates for increase and decrease predictions. MSE Ror falls in the "rate of returns" category. The formula for mse ror is below; R_i is real rate of returns, S_i is the real stock price, X_i is estimates for rate of returns or prices on day i and N is the number of predictions, MSE Ror is:

$$\sqrt{\frac{\sum_i (R_i - X_i)^2}{N}}. \tag{4.1}$$

MAE Ror: The long name for this metric is "mean absolute error for rate of returns". It calculates the absolute error between predicted rate of returns and real values. The formula for it is

$$\frac{\sum_i(|R_i - X_i|)}{N} \tag{4.2}$$

MSE Price statistic is very similar to MSE Ror except that instead of rate of returns we are using the stock prices themselves. This metric is formulized as

$$\sqrt{\frac{\sum_i(S_i - X_i)^2}{N}} \tag{4.3}$$

MAE Price is similar to MAE Ror except that instead of rate of returns we are using the stock prices themselves which is formulized as

$$\frac{\sum_i(|S_i - X_i|)}{N} \tag{4.4}$$

HR whose long name is "hit rate" calculates the number of times (days) the model succeeded in making a correct increase or decrease prediction for stock price data. This metric does not look at actual amount of increase in price, it simply looks at the rate of returns metric and compares the "signs" of predicted and actual values. Since rate of returns inherently reveal whether stock price increased or decreased, the comparison of said values suffices for hit rate calculations. HR+ captures the number of times an increase was predicted correctly, HR- captures the number of times a decrease was predicted correctly. HR is a sum of HR+ and HR-.

4.2.2. Plots

To produce the plots 4.1 and 4.2 of "one day ahead" predictions (inspired by [18]), we used a shifting training window of size 1700. HMM, KF and KMM were trained using data points $t, ..., t + W$ and using the trained model, we generated a prediction for the next day, $t + W + 1$. Then the training window is shifted to the right by one

day and the same procedure is repeated. This procedure was performed for 100 days. The predictions, along with the actual index price for each day are plotted side by side in the same graph. This graph was inspired by Hassan and Nath (covered in Section 1.9) who used it to display their method's results. The comparison of our graph and theirs can give a good sense of both methods in terms of prediction success.

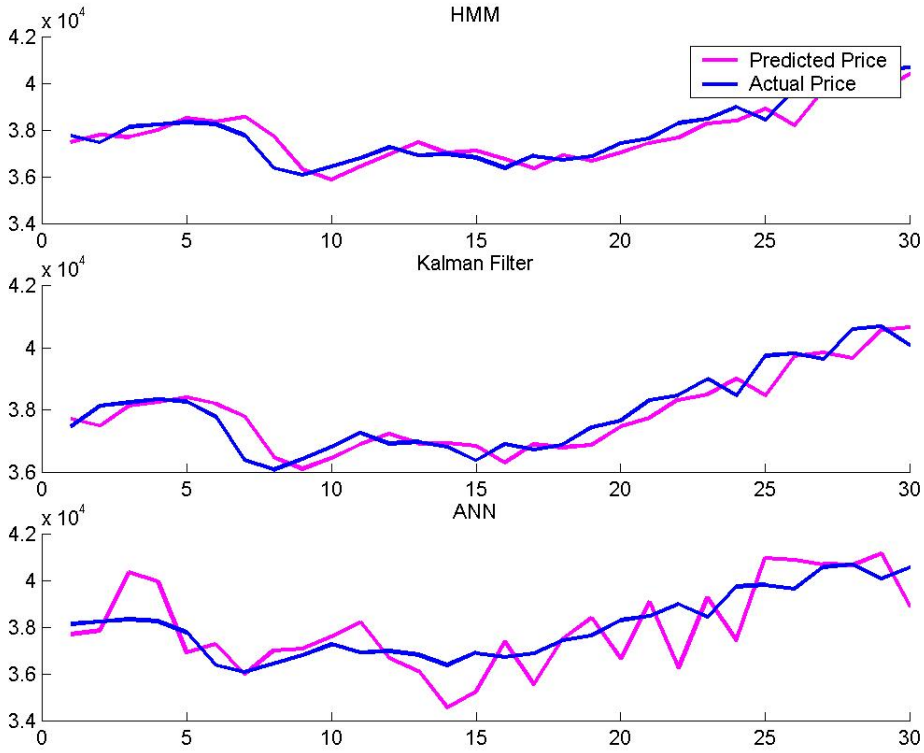


Figure 4.1. One Day ISE-100 Predictions Using HMM, KF and ANN

4.2.3. Conclusion

As "one day ahead" predictions (seen in Table 4.1) show , HMM, KF and KMM methods offer results that are better than random and polynomial based predictions. Kalman filters offer the best prediction in all categories, but the differences are especially obvious in absolute price categories. The only contender here was the ANN method, it was able to surpass KF prediction in terms of MAE metric. Kalman predictions in these categories are almost one third of HMMs. Most promising result was the *HR* result, for KF at 70%. ANN and polynomial HR results were better than HMM. The winner here are both ANN and KF methods. For trading purposes, I would use

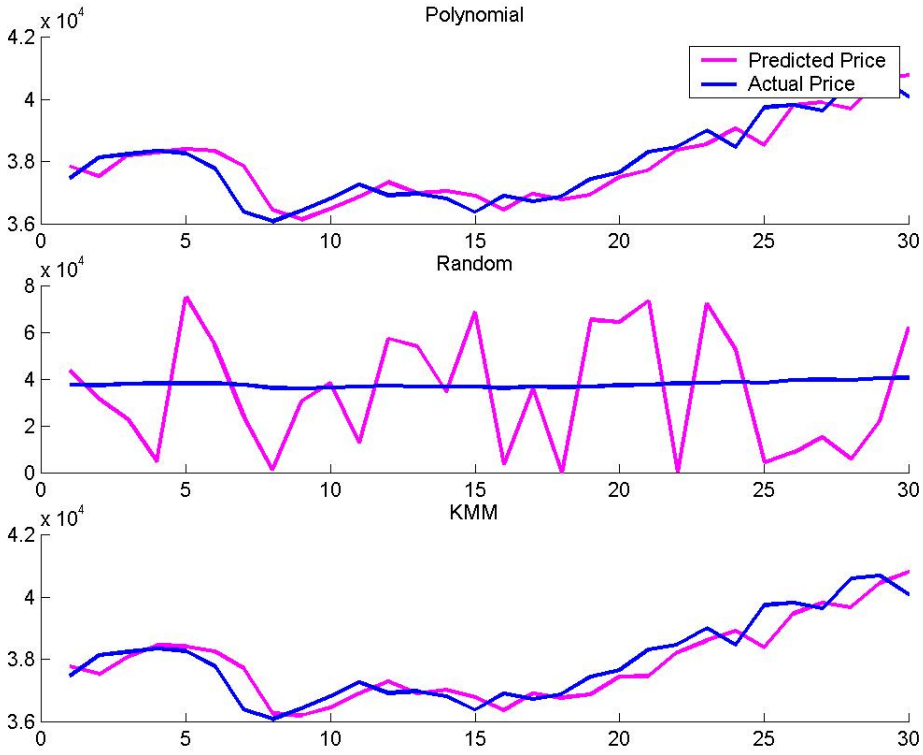


Figure 4.2. One Day ISE-100 Predictions Using polynomial, random and KMM KF.

As the number of days being predicted increases (seen in Table 4.2), we notice two things: The error rates in terms of absolute prices start increasing and second, in terms of *HR* metric, the difference between HMM and KF and their closest contender polynomial method start decreasing. In fact, all methods start approaching random prediction which in the long run must approach -%50 success rate. In terms of *MAE* metrics ANN method was the top performer, closely follwed by HMM, KF and KMM methods. One surprise here was ANN's *HR* metric, it was at %58 which for 5 day prediction is still a very good result. However we did not witness the same level of success for DJI 5-day prediction, hence we contributed this success to being "lucky", perhaps due to random number generation outputting just the right values, or to precision somehow working in the method's favor. The top performer in terms of *MSE* results is again ANN at around %0.006, so the method's overall performance is still very good. The difference between the top performers and random prediction is almost

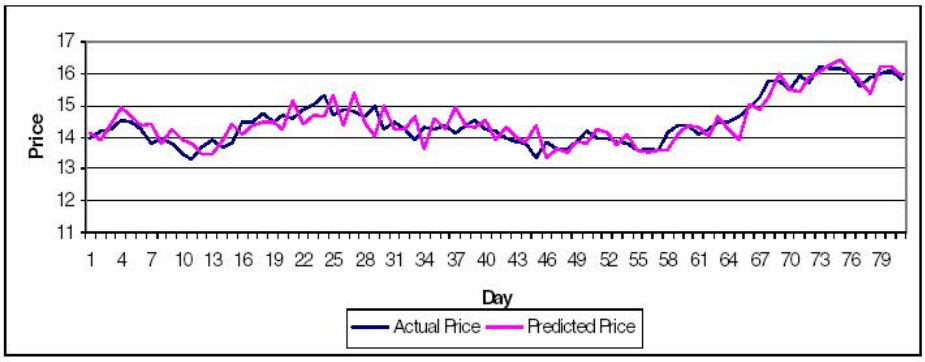


Figure 4.3. One Day Predictions Using HMM method by Hassan & Nath [18]

	HMM	Kalman	Random	ANN	Polynomial	KMM
MSE Ror	0.011	0.011	0.049	0.01154460	0.015	0.012
MSE Price	114.715	36.446	2176.878	1.24473817	518.535	156.740
MAE ROR	0.011	0.011	0.049	0.01154460	0.020	0.012
MAE Price	114.715	36.446	2176.878	1.24473817	518.535	156.740
HR	0.567	0.700	0.467	0.54838710	0.581	0.600
H+	0.333	0.467	0.267	0.32258065	0.581	0.400
H-	0.233	0.233	0.200	0.22580645	0.000	0.200

Table 4.1. ISE Prediction - 1 day

tenfold, random prediction is doing very badly, which of course, it should.

For 10 day predictions (seen in Table 4.3), we see that all methods with the exception of random prediction started approaching the same performance numbers in terms of *HR*. This is an indication that prediction 10 days ahead is a very hard task, and also, that each method is losing its advantage. Here, again, polynomial method was consistent in its success, however it also has a problem that its *H-* metric which is at a nearly zero. In comparison its *H+* result is at %50, this means the method has a bias toward increases. This bias would cause problems if one wanted to use this method for any form of trading activity.

The closeness of the results can be witnessed not only for *HR* metric, but also for absolute price and *HR* statistics. ANN method reported the best prediction in terms

	HMM	Kalman	Random	ANN	Polynomial	KMM
MSE Ror	0.007	0.007	0.027	0.00656345	0.007	0.007
MSE Price	382.314	375.512	1727.495	273.23645268	428.181	414.425
MAE ROR	0.012	0.012	0.053	0.01199135	0.015	0.013
MAE Price	690.210	690.491	3415.059	444.12065452	827.508	765.393
HR	0.527	0.493	0.440	0.58709677	0.490	0.467
H+	0.273	0.240	0.240	0.28387097	0.490	0.213
H-	0.253	0.253	0.200	0.30322581	0.000	0.253

Table 4.2. ISE Prediction - 5 days

	HMM	Kalman	Random	ANN	Polynomial	KMM
MSE Ror	0.005	0.005	0.019	0.00514647	0.005	0.005
MSE Price	407.231	424.893	1514.800	229.06783797	456.783	437.967
MAE ROR	0.013	0.013	0.053	0.01301490	0.014	0.014
MAE Price	1064.323	1107.195	4116.571	547.09327032	1239.209	1146.643
HR	0.500	0.497	0.437	0.52580645	0.510	0.483
H+	0.270	0.240	0.223	0.25806452	0.506	0.230
H-	0.230	0.257	0.213	0.26774194	0.003	0.253

Table 4.3. ISE Prediction - 10 days

of absolute stock prices.

It is clear looking at all *HR* numbers that for 10 days, we are at a level where random prediction is as good as any other method. Since the probability of guessing an increase and decrease are both 0.25 (in total 0.5) in the long run, we can see all methods are in the neighborhood of those numbers, in other words, predicting the increase and decrease 10 days ahead is as good as flipping a coin.

For "one day ahead" predictions using DJI values (seen in Table 4.4), we see similar performances as we saw with ISE-100 predictions. This actually is a relief because it proves that financial time series data behaves in similar ways to ISE time series values, and the trends are the same no matter which stock, index or country the

	HMM	Kalman	Random	ANN	Polynomial	KMM
MSE Ror	0.005	0.005	0.058	0.00580727	0.009	0.005
MSE Price	10.305	3.694	654.055	0.08456436	57.121	18.566
MAE ROR	0.005	0.005	0.058	0.00580727	0.011	0.005
MAE Price	10.305	3.694	654.055	0.08456436	57.121	18.566
HR	0.600	0.633	0.500	0.57142857	0.548	0.600
H+	0.367	0.367	0.233	0.33333333	0.290	0.367
H-	0.233	0.267	0.267	0.23809524	0.258	0.233

Table 4.4. Dow Jones Prediction - 1 day

prices are based on. More similarities can be seen in all metrics - for example *MSE Ror* difference between HMM/KF/KMM and polynomial prediction are still within 40%, *HR* success of KF method is the best, closely followed by ANN, HMM and KMM and then polynomial methods. There is almost a tenfold difference between *MSE Ror* for KF and random prediction. In terms of rate of return prediction, HMM/KF/KMM reported the best results even though in terms of absolute prices, ANN still performed better than all of the other methods.

	HMM	Kalman	Random	ANN	Polynomial	KMM
MSE Ror	0.003	0.003	0.025	0.00247295	0.003	0.003
MSE Price	45.048	43.990	403.774	21.49305772	51.288	47.909
MAE ROR	0.005	0.005	0.050	0.00449691	0.005	0.005
MAE Price	83.589	82.309	797.036	35.36699410	103.298	89.661
HR	0.460	0.553	0.473	0.51428571	0.587	0.460
H+	0.320	0.347	0.307	0.30476190	0.329	0.293
H-	0.140	0.207	0.167	0.20952381	0.258	0.167

Table 4.5. Dow Jones Prediction - 5 days

And just as was the case with ISE predictions, HMM/KF and polynomial results start approaching eachother as number of predicted days start to increase (seen in Table 4.5). However, in terms of DJI prediction, this approach happen much earlier (at 5 days) than was the case with ISE predictions. With ISE prediction, it was not until 10 days that the results came close to each other.

	HMM	Kalman	Random	ANN	Polynomial	KMM
MSE Ror	0.002	0.002	0.018	0.00187652	0.002	0.002
MSE Price	41.802	43.171	376.496	18.27777881	44.407	41.473
h MAE ROR	0.005	0.005	0.051	0.00477930	0.005	0.005
MAE Price	111.533	117.051	1012.789	44.48823690	123.517	110.753
HR	0.510	0.557	0.537	0.47619048	0.484	0.527
H+	0.320	0.283	0.287	0.28095238	0.235	0.290
H-	0.190	0.273	0.250	0.19523810	0.248	0.237

Table 4.6. Dow Jones Prediction - 10 days

As one can see from the data tables 4.1,4.2,4.3, 4.4,4.5 and 4.6 all methods were able to consistenly outperform random number based prediction. We expected to see those results. In respect to *MSE* and *MAE* metrics, ANN, HMM, KF and KMM methods are found to be superior to other methods. For "one day ahead" prediction, KF based prediction was best in terms of *HR* metric. The downside with polynomial fitting was its *H-* metric always reported values close to zero, which means this method was not able to guess market decreases meaning it is biased. HMM, KF, KMM and ANN methods all have a healthy allocation between *H+* and *H-* values.

The "one day ahead" plots make the success of the guesses quite visible as well. One point that needs mentioning here is that it could be incorrect to compare our graph directly with Hassan and Nath's, because they have consistently used the four values mentioned in order to "find" an HMM whereas we used one single HMM to "generate" a rate of return which we use to multiply with the last days' "known" price.

In terms of trading, various strategies can be devised in order to maximize return on investment. A simple trading strategy that takes a single stock or index as a basis for trade could decide to "buy" a stock if it predicts increase, or a"sell" if it predicts a decrease for the next day. In case of "buy" and "sell" sequence, the difference would be profit. In case of "sell" and "buy" sequence. In other words we could still make money, by extending this strategy in order to make money from "decreases" in the market. Most markets offer a trading activity called "short selling" which allows a trader to sell

a security he does not own, in hope of repurchasing them at a lower price. If I know the price of a security decrease to $100 tomorrow, and the price is $120 today, I can shortsell at $120, with a commitment to buy the actual stock tomorrow. Then if the price falls, I will be buying at a lower price selling it higher, pocketing the difference as profit.

REFERENCES

1. Gürgen, S., F., Yümlü, S., Okay, N., *A comparison of global, recurrent and smoothed-piecewise neural models for Istanbul stock exchange (ISE) prediction*, Pattern Recognition Letters, Volume 26, Issue 13, Pages 2093-2103, 1 October 2005,

2. L. R. Rabiner, *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of IEEE vol. 77, no. 2, pp. 257-286, 1989

3. Roweis, S. and Ghahramani, Z. *A Unifying Review of Linear Gaussian Models* , Neural Computation 11(2):305–345, 1999

4. Ghahramani, Z., Hinton, H.,E., *Parameter Estimation for Linear Dynamical Systems* , Technical Report CRG-TR-96-2 [ftp://ftp.cs.toronto.edu/pub/¬ zoubin/tr96-2.ps.gz], Department of Computer Science, University of Toronto, 1996

5. Alpaydın E. , *Introduction to Machine Learning*, MIT Press, 2004.

6. Ghahramani, Z., Hinton, H.,E., *Switching State Space Models*, Technical Report CRG-TR-96-3, Dept. Comp. Sci., Univ. Toronto, 1996

7. R.H.Shumway, S. Stoffer *Time series analysis and its applications 2nd Edition*, New York, Springer, (Springer texts in statistics), 2000

8. M. I. Jordan, C. Bishop *An Introduction to Graphical Models*, Not yet published, 2000

9. R., E., Kalman, *A New Approach to Linear Filtering and Prediction Problems*, Transactions of the ASME-Journal of Basic Engineering, 82 (Series D): 35-45.

10. Kalman, R.E. and R.S. Bucy, *New results in filtering and prediction theory*, Trans. ASME J. Basic Eng., 83, 95-108.

11. Welling, M., *The Kalman Filter - Lecture Tutorial*, California Institute of Technology, 2008

12. Lall, S., *Modern Control 2 Lecture Notes*, Stanford University, 2006

13. Fama, E.,F., *The behavior of stock-market prices,*, The Journal of Business of the University of Chicago, 38, no. 1. 1965

14. Fama, E.,F., *Random walks in stock market prices,*, Financial Analysts Journal, 21 (5): 55-59., 1965

15. Tsay, R.,S., *Time Series and Forecasting: Brief History and Future Research,*, Journal of the American Statistical Association, Vol. 95, No. 450. (Jun., 2000), pp. 638-643. 2000

16. Bachelier, L., *Thorie de la speculation. Annales Scientiques de l'Ecole Normale Superieure, III-17:21-86* Thesis for the Doctorate in Mathematical Science., 1900 Reprinted by Editions Jacques Gabay, Paris 1995. English translation in The Random Character of Stock Market Prices, Ed. P. Cootner, pp 1778, Cambridge, MIT Press, 1964.

17. S. Shi and A. S. Weigend., *Taking Time Seriously: Hidden Markov Experts Applied to Financial Engineering*. In Proceedings of the IEEE/IAFE 1997 Conference on Computational Intelligence for Financial Engineering , pages 244-252. IEEE, 1997.

18. Md. Rafiul Hassan, Baikunth Nath, *Stock Market Forecasting Using Hidden Markov Model: A New Approach*, ISDA 2005: 192-196

19. Mehdi Azzouzi and Ian T. Nabney, *Modelling Financial Time Series with Switching State Space Models*, `citeseer.ist.psu.edu/267552.html`

20. Robert F. Engle *GARCH 101: The Use of ARCH/GARCH Models in Applied Econometrics*, Journal of Economic Perspectives-Volume 15, Number 4-Fall 2001-Pages 157-168

21. Robert F. Engle, Victor K. Ng, *Measuring and Testing the Impact of News on Volatility*, The Journal of Finance, Volume XLVIII, No. 5, December 1993

22. Robert F. Engle, *Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation*, Econometrica. 50:4, pp. 987-1007. University of California, San Diego. Revision

23. Bollershev, T., *Generalized autoregressive conditional heteroskadasdicity*, Journal of Econometrics 31, 307-327

24. Nelson, D., *Conditional heteroskedasticity in asset returns: A new approach*, Econometrica, 59, 347-370, 1990

25. B. B. Mandelbrot, *The variation of certain speculative prices, Journal of Business*, XXXVI (1963), pp. 392-417, 1963