# Regression and Classification for Stock Market Prediction

**Mahdi Zarour** [*1] **Mehdi Ghaniabadi** [*2]

## Abstract

In this project, we intend to implement some probabilistic graphical models for regression and classification and compare their performance with other well-known machine learning models, on a real stock price dataset. For regression, HMM and Kalman Filtering models, and for classification, Bayesian Networks are implemented, which are then compared with parametric and non-parametric machine learning models. Important and interesting insights are then concluded from the results.

## 1. Introduction

This project is divided into two parts: Regression and Classification. For regression, we want to predict the future stock prices, where we use the historical stock prices of the Apple company extracted from the Yahoo Finance website. The training data contains the daily stock prices of Apple in the year 2018 (251 days in total) and the test data includes the daily stock prices in 2019 (252 days in total). In such a dataset, the target variable is the closing price of each day, while the covariate is the corresponding opening price of that day. For this task, two time series models are implemented: HMM and Kalman Filtering. Moreover, seven supervised learning models are also performed.

For classification, we transform the same dataset into a categorical dataset, where we want to predict to what extent the future stock price will change. Then we perform eight supervised learning models including non-parametric models, neural networks, and Bayesian Networks. Since the resulted best accuracy is relatively low, we also examine our models on another dataset related to a classical car classification task, which results in significantly higher accuracies.

The rest of this report is structured as follows. In sections 2 and 3, we present the efforts we made for performing regression, which include time series and supervised learning models, respectively, and their corresponding results. In section 4, we present our classification tasks and the results of supervised learning models performed on the tasks. We conclude our report in section 5. We also explain why we are a team of two, after the conclusions.

[*]Equal contribution [1]Department of Computer Science and Operations Research, University of Montreal, Montreal, Canada [2]Department of Logistics and Operations Management, HEC Montreal, Montreal, Canada. Correspondence to: Mehdi Ghaniabadi <mehdi.ghaniabadi@hec.ca>, Mahdi Zarour <mahdi.zarour@umontreal.ca>.

*IFT 6269 project report - Université de Montréal*

## 2. Regression: Time Series Models

We present two models, one using Hidden Markov Models architecture and the other implements Kalman filtering.

### 2.1. Hidden Markov Models

#### 2.1.1. Model

Implementing (Gupta & Dhingra, 2012), we consider the standard hidden markov models, the independence assumptions induced by the HMM are well known and as follows:

$$x_{t+1} \perp x_{1:t-1} | x_t \quad \forall t$$
$$y_t \perp x_{1:t-1}, y_{1:t-1} | x_t \quad \forall t$$

We denote by $x_t$ the latent variable at time step $t$ and $y_t$ the observation at time step $t$.

#### 2.1.2. Approach

The goal is to predict the daily closing price of a stock given the opening price. The model observations are constructed from known stock features as follows:

$$y_t = (\frac{close-open}{open}, \frac{high-open}{open}, \frac{open-low}{open})$$
$$y_t = (fracChange, fracHigh, fracLow)$$

where we take open, high, low and close prices at time $t$.
We then train the model using the hmmlearn library on python. The model is designed to run after the market opens because we need the open price to infer the close price, to predict the close price at time $d+1$ different possible values for the observation are generated, see Table 1. We will use the observation of the $d$ preceding time steps ($d$ is called the latency and can be changed) and choose the observation at time $d+1$ that maximizes the MAP estimate. Assuming $\lambda$ is the model parameter, the following formula summarizes the objective:

$$y_{d+1} = \underset{y_{d+1}}{\operatorname{argmax}}\, p(y_{d+1}|y_1, ..., y_d, \lambda)$$
$$y_{d+1} = \underset{y_{d+1}}{\operatorname{argmax}}\, \frac{p(y_{d+1}, y_1, ..., y_d, \lambda)}{p(y_1, ..., y_d, \lambda)}$$
$$y_{d+1} = \underset{y_{d+1}}{\operatorname{argmax}}\, p(y_{d+1}, y_1, ..., y_d|\lambda)$$

*Table 1.* Range values for predicted observation

| Observation | Min | Max | Number of points |
|---|---|---|---|
| fracChange | $-0.1$ | 0.1 | 50 |
| fracHigh | 0 | 0.1 | 10 |
| fracLow | 0 | 0.1 | 10 |

The intuition behind using HMM to predict the stock prices is that the latent variables $x_{1:T}$ determines the behaviour of the observations and the transition between the possible states is based on the company policy, decisions and economic conditions which are also hidden to the investor and that makes the HMM a prefect candidate to model this problem.

#### 2.1.3. Implementation

In the original paper the implementation was done in Matlab. Our implementation was done in python using the hmmlearn library (see https://github.com/hmmlearn/hmmlearn), to train the model and the other aspects were done from scratch. The HMM parameters are the following:

- The states are assumed to be multinomial and the number of hidden states is $n = 4$, the prior probabilities and the transition probabilities are initialized from a uniform distribution
- The emission probabilities are modeled as a mixture of Gaussians and the number of components is $m = 5$, they are initialized via K-means algorithm
- The value used for the latency $d$ is 10

### 2.2. Kalman Filter

#### 2.2.1. Model

Implementing (Yan & Guosheng, 2015/06), we consider the standard Kalman Filter, the graphical model is the same as HMM and hence induces the same independence properties. Here are the equations for the observations denoted by measurements here which are the opening price of a stock and the states which are the closing price:

$$x_t = Ax_{t-1} + W_t \qquad W_t \in \mathbb{R}^k \sim N(0, \Gamma)$$
$$y_t = Cx_t + V_t \qquad V_t \in \mathbb{R}^d \sim N(0, \Sigma)$$

With $x_t \in \mathbb{R}^k$ and $y_t \in \mathbb{R}^d$. Knowing the nature of the data we present an alternative definition:

$$x_t = a + bx_{t-1} + cw_t \qquad w_t \in \mathbb{R} \sim \text{iid } N(0, 1)$$
$$y_t = x_t + fv_t \qquad v_t \in \mathbb{R} \sim \text{iid } N(0, 1)$$

We further assume $x_1, v_1, ..., v_T, w_1, ..., w_T$ are jointly gaussian and independent. Using this assumptions we have $x_1, ...x_T, y_1, ..., y_T$ are jointly gaussian.

#### 2.2.2. Approach

The approach we used here is a mix between the paper model and our personal reasoning guided by the document in the link https://courses.maths.ox.ac.uk/node/view_material/1317 and it is justified. The values of the parameters used can be considered random values, did not take into account the data and did not follow any method. Instead by using the available graphical model we maximize the probability of the joint distribution over the states and the measurements. The following formula summarizes the objective:

$$\hat{a}, \hat{b}, \hat{c}, \hat{f} = \underset{a,b,c,f}{\operatorname{argmax}}\, p(x_1, ..., x_T, y_1, ..., y_T)$$
$$\hat{a}, \hat{b}, \hat{c}, \hat{f} = \underset{a,b,c,f}{\operatorname{argmax}}\, p(x_1) \prod_{t=2}^{t=T} p(x_t|x_{t-1}) \prod_{t=1}^{t=T} p(y_t|x_t)$$

Where $T$ is the sequence length, solving the following objective gives:

$$\hat{f}^2 = \frac{\sum_{t=1}^{T}(x_t - y_t)^2}{T}$$

$$\hat{b} = \frac{(T-1)\sum_{t=2}^{T}(x_t \times x_{t-1}) - (\sum_{t=2}^{T} x_t)(\sum_{t=2}^{T} x_{t-1})}{(T-1)\sum_{t=2}^{T} x_{t-1}^2 - (\sum_{t=2}^{T} x_{t-1})^2}$$

$$\hat{a} = \frac{\sum_{t=2}^{T} x_t - bx_{t-1}}{T-1}$$

$$\hat{c}^2 = \frac{\sum_{t=2}^{T}(x_t - a - bx_{t-1})^2}{T-1}$$

We keep $\hat{f}^2$ and $\hat{c}^2$ because we don't need the roots.
The next step consists on applying the Kalman filter, to do so we need to understand the nature of the random variable $x_t|y_{1:t-1}$, using our assumptions we have that the joint $y_{1:t-1}$ is normal plus using the fact that the conditional of two normal variables is also normal we hence need to compute the mean and variance of this random variable. The Kalman filter is divided into two parts, we start with the prediction equations:

$$\mu_{t|t-1} = E[x_t|y_{1:t-1}] = E[a + bx_{t-1} + cw_t|y_{1:t-1}]$$
$$\mu_{t|t-1} = a + bE[x_{t-1}|y_{1:t-1}] = a + b\mu_{t-1|t-1}$$

$$P_{t|t-1} = Var[x_t|y_{1:t-1}] = Var[a + bx_{t-1} + cw_t|y_{1:t-1}]$$
$$P_{t|t-1} = b^2 Var[x_{t-1}|y_{1:t-1}] + c^2 Var[w_t|y_{1:t-1}]$$
$$P_{t|t-1} = b^2 P_{t-1|t-1} + c^2$$

Next are the correction equations, we need to compute the Kalman gain $K_t$ before that which can be described as the trajectory taken (a high value means the model puts more weight on the prediction and a low value means the model puts more weight on the measurement):

$$K_t = \frac{P_{t|t-1}}{P_{t|t-1}+f^2}$$
$$\mu_{t|t} = \mu_{t|t-1} + K_t(y_t - \mu_{t|t-1})$$
$$P_{t|t} = f^2 K_t$$

The prediction of the next time step is just applying the filtering algorithm on the test data, hence we only need the training data to estimate the parameters $a, b, c$ and $f$.

#### 2.2.3. Implementation

The implementation in the original paper is done in Matlab, our implementation is done in python using numpy library, see (Harris et al., 2020). The initialization is as follows:

- $\mu_{1|1}$ is initialized to $x_1$ and $P_{1|1} = 10000$ is the result of trying many values and choosing the one that gives the best results

### 2.3. Results and Complexity

Here we present the results of applying the two methods on the daily Apple stock. We train the model on the data of 2018 (251 days) and test them on the data of 2019 (252 days) and will use the mean absolute percentage error MAPE to compare the results which is defined as:

$$MAPE = \sum_{t=1}^{T} \frac{|a_t - b_t|}{|b_t|} \times 100\%$$

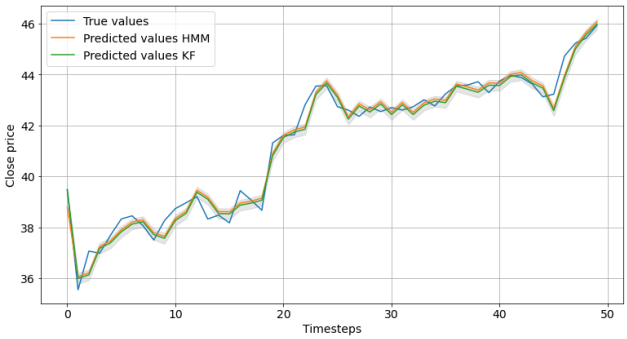
Where $a_t$ is the forecast of the price at time $t$ and $b_t$ is the true price at the same time. By applying it on the test set we get:

*Table 2.* MAPE for the two approaches

| | Kalman F | HMM |
|---|---|---|
| MAPE | 0.807% | 0.797% |

We see that the Kalman filter gives better results than Hidden Markov Models in Table 2. but it has been an easy task to improve the performance of the latter, by just changing the number of hidden states from 4 to taking the whole data as a unique sequence we get a MAPE of 0.797%, even though we still don't have a formal proof of why it works a simple intuition is by changing the number of hidden states we also change the seasonality of data which is simply speaking the repeated regular change the data follow, and the seasonality obviously changes by changing the data.
Now let's talk about the complexity of the two algorithms, we shall focus on the prediction of future values since it is where the difference is clear in practice. The HMM prediction uses the MAP estimate which complexity is dominated by the alpha recursion (Jordan, 2003). This latter runs in $O(m^2 d)$ for a single posterior estimation where $m$ is the number of components and $d$ the latency, we have $k = 5000$ different combination and $T$ test data implies the total complexity is $O(Tm^2dk)$. The kalman filter prediction consists on applying the filtering equations which results approximately in $O(T)$ due to the linearity of the computations (Montella, 2011). In practice the HMM runs in 295 seconds which is 5 minutes approximately while the Kalman filter takes less than 1 second.

*Figure 1.* Actual vs Forecast close price of Apple stock for the first 50 days of 2019




### 2.4. Comparison and efficiency

From the last section, we saw that the HMM gives a very slight improvement compared to the Kalman filter but is extremely slow compared to the latter. Are they efficient in real life application and which one is better. To answer this question we need to define a baseline to compare our results with, the most simple and naive one is the open price, obviously if your algorithm prediction is worse than the open price then it needs improvement. Using the open price of the test data to compute the MAPE we get 0.805%, this suggests that any prediction with a higher MAPE is worse. Using our improved HMM prediction for **2.3** and the Kalman filter prediction we see that the HMM beats slightly the baseline whereas the Kalman filter does not. This suggests those methods need further improvement to be applied in the real market.

## 3. Regression: Supervised Learning Models

Seven well-known supervised learning models are implemented for the regression problem, using the Scikit-Learn Python library (Pedregosa et al., 2011), and we do not follow any particular paper for these supervised learning models. Different values for the hyperparameters of each model are examined and optimized based on minimizing the test error. First, K-Nearest Neighbors (KNN) regrossor is performed, which provides a prediction by averaging the "K" points for which their features are closest to the features of the future period. Different values of K (number of neighbors) are considered where K=3 was the optimal one. Moreover, decision tree regressor under its default settings is used. Next, gradient boosting is performed which tries to improve the solution of a decision tree over various stages. For this model, the number of boosting levels (50) and the learning rate (0.05) are optimized. Random forest regressor, which takes the average of several decision trees, is also implemented. The number of trees (500) and the maximum depth of each tree (50) are optimized. These four models, form the set of our non-parametric models.

Furthermore, we have also implemented Neural Network regressor where the number of hidden layers (3), their size (5,10,15), activation function (relu), and the solver (lbfgs), are optimized. Using wider or deeper neural networks resulted in overfitting. Relu is the most popular activation function and it also worked better than "logistic" and "identity" activation functions. Adam is most popular solver for weight optimization, however, in our data, the solver "lbfgs" worked better than Adam and SGD solvers. We also performed linear regression and support vector regression (SVR). For SVR, the kernel (linear), and regularization parameter (0.087) are optimized. The kernels 'poly', 'rbf', and 'sigmoid' worked worse than the 'linear' one.

In regards to data preparation, since we only had one feature (the opening price of the day), we also added four other features: day, month, open and close price of the previous day, in order improve the accuracy of the supervised learning models. Surprisingly, adding these extra features did not help significantly (Neural network, linear regression and SVR got a little better, while non-parametric models got worse). This shows that additional information is not necessarily helpful in machine learning. Nevertheless, since SVR was our best model and it slightly improved, we included these extra features in the models.

The Table 3 shows the results of our regression models in terms of train and test MAPE. MAPE is an evaluation criteria for regression models and stands for Mean Absolute Percentage Error and is calculated as $\left|\frac{the\,true\,value - the\,predicted\,value}{the\,true\,value}\right| * 100$. Non-parametric models which consist of KNN, decision tree, gradient boosting, and random forest, do not perform well and have a high error. Random forest is the best non-parametric model which is typical. Interestingly, neural network, linear regression and SVR performed significantly better than non-parametric models. The high performance of Neural Network compared to non-parametric models, is surprising because we do not have a big data. SVR has the least MAPE which shows that it is even better than both HMM and Kalman Filtering which is surprising since we have time series data. This highlights the concept of "No free lunch in machine learning!" which states that there is no best machine learning model and that no model should be taken for granted. The Figure 2 presents the true and predicted values for the SVR model, for the first 50 rows of the test set.

*Table 3.* MAPE of the regression models for the stock price dataset.

| MODEL | TRAIN MAPE | TEST MAPE |
|---|---|---|
| KNN REGRESSOR | 0.828 | 5.282 |
| DECISION TREE REG. | 0.0 | 4.812 |
| GRADIENT BOOSTING REG. | 1.067 | 4.359 |
| RANDOM FOREST REG. | 0.412 | 4.026 |
| NEURAL NETWORK REG. | 1.015 | 0.813 |
| LINEAR REGRESSION | 1.016 | 0.808 |
| KALMAN FILTERING | - | 0.807 |
| HMM | - | 0.797 |
| SUPPORT VECTOR REG. | 1.012 | 0.790 |

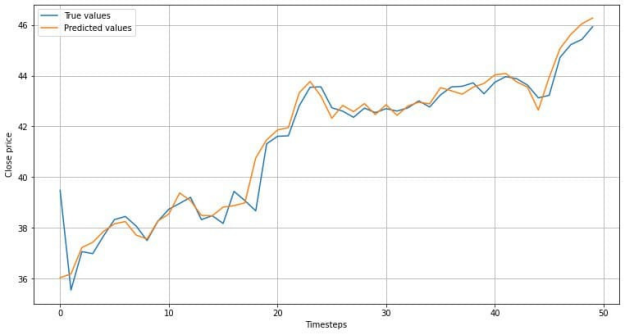



*Figure 2.* The predicted values of SVR vs the true values for the first 50 rows of the test set.

## 4. Classification

To do the classification task for the stock market prediction, we first transform the same stock data we used for regression, as follows. For each day, we obtain the value $\frac{Close\,Price - Open\,Price}{Close\,Price} * 100$, which demonstrates the percentage change of the stock price at each day. Then assign a category for that value based on the interval it resides in. We consider 8 intervals: less than $-2.5\%$, $(-2.5\%, -1\%)$,

*Table 4.* The accuracies of the classification models for the stock price dataset.

| MODEL | TRAIN ACC. | TEST ACC. |
|---|---|---|
| KNN CLASSIFIER | 47 | 18.6 |
| DECISION TREE CLASSIFIER | 100 | 17.4 |
| GRADIENT BOOSTING CL. | 59.7 | 25 |
| RANDOM FOREST CL. | 100 | 18.2 |
| NEURAL NETWORK CL. | 32.6 | 30.1 |
| SUPPORT VECTOR CL. | 32.6 | 34.9 |
| GAUSSIAN NAIVE BAYES | 34.2 | 35.3 |
| BAYESIAN NETWORK | 34.2 | 32.5 |

*Table 5.* The accuracies of the classification models for the car evaluation dataset.

| MODEL | TRAIN ACC. | TEST ACC. |
|---|---|---|
| KNN CLASSIFIER | 96.8 | 93.6 |
| DECISION TREE CLASSIFIER | 100 | 98.2 |
| GRADIENT BOOSTING CL. | 94.9 | 92.4 |
| RANDOM FOREST CL. | 100 | 97.1 |
| NEURAL NETWORK CL. | 89.1 | 86.1 |
| SUPPORT VECTOR CL. | 96.1 | 94.7 |
| GAUSSIAN NAIVE BAYES | 77.5 | 75.1 |
| BAYESIAN NETWORK | 81.0 | 64.7 |

$(-1\%, -0.25\%)$, $(-0.25\%, 0)$, $(0, 0.25\%)$, $(0.25\%, 1\%)$, $(1\%, 2.5\%)$, and more than $2.5\%$. Consequently, our target variable is the category of a future day. We also consider three features: day and month corresponding to the future day, and the category of the previous day.

We perform eight supervised learning models including KNN Classifier, Decision Tree Classifier, Gradient Boosting Classifier, Random Forest Classifier, Neural Network Classifier, Support Vector Classification, Gaussian Naive Bayes, and a discrete Bayesian Network. All of these models are implemented via the Scikit-Learn Library (Pedregosa et al., 2011), except for the Bayesian Network, which is implemented using the PGMPY Python library (Ankan & Panda, 2015). The Table 4 shows the corresponding train and test accuracies, where the accuracy score shows the percentage of the data which are classified correctly. Similar to the regression case, the four non-parametric models are have the lowest test accuracy. On the other hand, Gaussian Naive Bayes, Support Vector Classification, and Bayesian Network provide the highest test accuracy. The Naive Bayes assumes the independence of the features, as opposed to a Bayesian Network. Therefore, we would expect that Bayesian Network perform at least as good as Naive Bayes on real datasets. However, in this case the Gaussian Naive Bayes has a higher test accuracy. It seems to be due to the fact that our categories are ordinal and represent intervals of numerical data. Therefore, the gaussian distribution of the Gaussian Naive Bayes, better describes our ordinal categories, compared to our discrete Bayesian Network.

The Gaussian Naive Bayes achieves the best test accuracy which is 35.3% and is roughly three times higher than the random classification accuracy of 12.5%. Nevertheless, an accuracy of 35.3% is still relatively low which seems to be due to the fact that this is a hard classification task. To examine this argument, we also use our classification models on a classical Car evaluation dataset.

We will examine our models on this real dataset as follows. The high accuracy that we get on this task shows that our models are indeed effective classification models, however, the classification of our stock data is an inherently difficult machine learning task.

This data is a classical dataset which can be found in this link https://archive.ics.uci.edu/ml/datasets/Car+Evaluation, and have been used by many other researchers. For example, Cheng and Greiner (1999) use this data for classification via Bayesian Network. Nevertheless, our implementations of classification models for this dataset (and also for the stock classification dataset), do not follow that paper or any other paper. In this dataset, the target variable is the value of a given car which has four categories (unacceptable, acceptable, good and very good). The feature are six characteristics of the car including price, maintanance cost, number of doors and persons, size of luggage boot, and the safety of the car, where all considered as categorical variables in this data.

Since this data is not a time series, we split it into a train (90% of the original data) and a test set (10% of the original data), using sklearn Python package (Pedregosa et al., 2011), in a way that the distribution of each class in the train and test sets are roughly the same, in order to make sure that the results are not biased towards a specific class.

The results for the car classification task are presented in the Table 5. Again we see that Gaussian Naive Bayes has a higher test accuracy than the discrete Bayesian Network due to the fact that in the car evaluation dataset, we also have some ordinal variables. However, unlike the stock price classification, our test accuracies are much higher and we reach a best test accuracy of 98.2% via the Decision Tree Classifier. These results support the argument that our models are indeed powerful, however, the stock classification is a hard task under our stock price data.

## 5. Conclusions

In this project, we analyzed various machine learning models to perform stock price prediction, and achieved very interesting and at times surprising results. The results of all the regression and classification tasks that we performed have one common aspect which is the concept of no free lunch in machine learning and not to take for granted any model, regardless of the type of data you have.

Moreover, to our knowledge, the available Python packages such as PGMPY (Ankan & Panda, 2015) are not yet able to do prediction using a Bayesian Network, when the target variable is real valued. For our future research, we will use the "bnlearn" package (Scutari, 2010) of R instead, which has the capability of performing regression using Bayesian Network. It seems that for statistical learning tasks, R has more powerful packages than Python, which we did not expect before.

In regards to the time series models, we explored models to forecast the daily price of stock, even though the results do not show clear evidence of efficiency, we got better understanding of time series forecast and why the difficulty is higher when targeting the stocks, the lack of signal is the first striking reason because most of the classic and known methods rely on some redundancy of the signal called seasonality and a trend that explains the direction taken and those concepts are either not present here or are taking a different unknown shape if we are optimistic. An important approach to explore is to actually consider the original Kalman filter approach where we let the states be latent following the same intuition as the Hidden Markov Models and considering the measurements to be the open and closing prices. It also might be worth trying more modern models of deep learning that intuitively fit to the problem like LSTM, WaveNet and SeriesNet.

## The number of teammates

It is worth mentioning that we first were a group of three members, however, our teammate Daniel Dermont decided to drop the course on November 9th, 2020. Nevertheless, he did NOT have any contribution on this project. Hence, this whole project is the contribution of only two people.

## References


https://github.com/hmmlearn/hmmlearn.

https://courses.maths.ox.ac.uk/node/view_material/1317.

Ankan, A. and Panda, A. pgmpy: Probabilistic graphical models using python. In *Proceedings of the 14th Python in Science Conference (SCIPY 2015)*. Citeseer, 2015.

Cheng, J. and Greiner, R. Comparing bayesian network classifiers. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pp. 101–108. Morgan Kaufmann Publishers Inc., 1999.

Gupta, A. and Dhingra, B. Stock market prediction using hidden markov models. In *2012 Students Conference on Engineering and Systems*, pp. 1–4, 2012. doi: 10.1109/SCES.2012.6199099.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del R'ıo, J. F., Wiebe, M., Peterson, P., G'erard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Jordan, M. I. *An Introduction to Proabilistic Graphical Models*. University Of California, Berkeley, draft for ift-6269 edition, 2003.

Montella, C. The kalman filter and related algorithms: A literature review. *Research Gate*, 2011.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Scutari, M. Learning bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010. doi: 10.18637/jss.v035.i03.

Yan, X. and Guosheng, Z. Application of kalman filter in the prediction of stock price. In *Proceedings of the 5th International Symposium on Knowledge Acquisition and Modeling*, pp. 197–198. Atlantis Press, 2015/06. ISBN 978-94-62520-87-5. doi: https://doi.org/10.2991/kam-15.2015.53. URL https://doi.org/10.2991/kam-15.2015.53.