Karlsruher Institut für Technologie

Department of Economics and Management

Institute of Economics (ECON)

Chair of Econometrics

Prof. Dr. Melanie Schienle

Seminararbeit

Stock-Forecasting using HMM - A Model Averaging Approach

von

Jonas Falkner

1717559

Wirtschaftsingenieurwesen (Master)

Abgabe am

10. April 2017

Inhaltsverzeichnis

1 Einführung **1**

2 Hidden Markov Models **2**

2.1 HMM 2

2.2 Gauß HMM 3

3 Beschreibung des Modells **4**

3.1 Model Averaging 5

3.1.1 Naive Model Averaging (NMA) 5

3.1.2 Total Model Averaging (TMA) 5

3.1.3 Selective Model Averaging (SMA) 7

4 Analyse **8**

4.1 Versuchsaufbau 8

4.2 Analyse der einzelnen Methoden 8

5 Fazit **16**

Literaturverzeichnis **17**

A Python-Code **19**

1 Einführung

Aktienkurse spiegeln häufig zumindest mittelbar recht gut den allgemeinen Zustand eines Wirtschaftssystems wider, der durch Aufschwungs- und Rezessionsphasen geprägt ist. Diese Phasen sind zwar nicht direkt beobachtbar, lassen sich aber zum Beispiel u.a. durch unterschiedliche Volatilität charakterisieren [Hamilton and Lin (1996); Lee and Shin (2009)]. Aus diesem Grund bietet sich zur Prognose von Aktienkursen die Verwendung sogenannter Hidden Markov Models (HMM) an.

Die vorliegende Seminararbeit widmet sich der Vorhersage von täglichen Schlusskursen, mit Hilfe von HMM. Das Modell verwendet dazu verschiedene Ansätze des *Model Averaging*, um dynamisch die Vorhersagen verschiedener HMM mit unterschiedlich vielen Zuständen zu verbinden und verwendet den ermittelten Wert zur Vorhersage des Schlusskurses am Folgetag. Zur beispielhaften Demonstration des Modells werden die bei der NASDAQ notierten Aktienkurse von vier bekannten Unternehmen, namentlich IBM, Apple, Facebook und Google herangezogen. Die Daten stammen von Yahoo! Finance [Yahoo! Inc. (2017)].

2 Hidden Markov Models

Hidden Markov Models werden seit Ende der 1960er Jahre eingesetzt und erfreuen sich seitdem überaus großer Beliebtheit. Sie wurden und werden auch aktuell in vielen verschiedenen Themengebieten und wissenschaftlichen Disziplinen wie Automatische Spracherkennung, Finanzanalyse, Erkennung von Handschrift und Bio-Chemischen Analysen, z.B. zur Entschlüsselung von DNA, eingesetzt [Rabiner (1989); Bishop (2009)]. Dies liegt an ihren starken mathematischen Grundlagen und den schnellen und effizienten Algorithmen, die aufgrund ihrer langjährigen kontinuierlichen Erforschung und Weiterentwicklung vorhanden sind [Hassan and Nath (2005)].

2.1 HMM

Für ein HMM nimmt man meist eine Beobachtungsreihe O der Länge T an, deren Beobachtungen stochastisch unabhängig sind. Grundsätzlich besteht ein HMM aus einer Anzahl N versteckter („hidden“) Zustände $Z_i, i \in 1,...,N$, die auch als diskrete latente Variablen interpretiert werden können [Bishop (2009)]. Darüber hinaus wird abhängig von den Zeitpunkten t die Sequenz $Q = q_t$, $t = 1,...,T$ der Zustandsübergänge bestimmt. Der Übergang von einem Zustand in den anderen wird im Normalfall als eine Markovkette erster Ordnung modelliert [Rabiner (1989)]. Es existiert demzufolge eine Übergangsmatrix A, welche die Übergangswahrscheinlichkeiten $a_{ij} = P(q_t = Z_j | q_{t-1} = Z_i)$ mit $i, j = 1,...,N$ darstellt. Des Weiteren gibt es auch eine entsprechende Startverteilung π mit den Startwahrscheinlichkeiten $\pi_i = P(q_0 = Z_i)$, $i = 1,...,N$.

Zusätzlich wird eine Anzahl M an Zustandsausprägungen $v_k, k = 1,...,M$ definiert, welche die einzelnen Zustände annehmen können. Abschließend gibt es eine Emissionsmatrix B, welche die Emissionswahrscheinlichkeiten $b_i(k) = P(O_t = v_k | q_t = Z_i)$, $i = 1,...,N, k = 1,...,M$ beschreibt [Rabiner (1989); Hassan and Nath (2005)]. Die Parametermenge des HMM sei mit λ gekennzeichnet. Dabei gilt zunächst $\lambda = \{A, B, \pi\}$.

Im Allgemeinen hat man es im Umfeld von HMM mit drei grundlegenden Problemen zu tun [Rabiner (1989); Nguyen (2016)]:

1. Wie kann die Wahrscheinlichkeit von O unter dem Modell gegeben die Modellparameter λ und die Beobachtungsreihe O, berechnet werden?

2. Wie kann die Sequenz Q von Zustandsübergängen, die am wahrscheinlichsten bzw. passendsten ist, gefunden werden, sofern die Beobachtungsreihe O und Modellparameter λ gegeben sind?

3. Wie sollen die Modellparameter λ für ein HMM gegeben die Beobachtungsreihe O gewählt werden, sodass $P(O|\lambda)$ maximiert wird?

Wie bereits erwähnt, existieren zur Lösung dieser Probleme bereits etablierte Algorithmen. So kann für Problem 1 der so genannte Forward-Backward-Algorithmus verwendet werden, Problem 2 lässt sich mit Hilfe des Viterbi-Algorithmus lösen, wie der Max-Sum-Algorithmus im Umfeld von HMM auch genannt wird [Bishop (2009)]. Die Lösung für Problem 3 wird schließlich durch Anwendung des Baum-Welch-Algorithmus (bzw. EM-Algorithmus), erreicht [Hassan and Nath (2005); Nguyen (2016)].

2.2 Gauß HMM

Bei einem Gauß HMM wird als Wahrscheinlichkeitsverteilung für die Beobachtungen eine Normalverteilung angenommen. Das Modell ändert sich dadurch wie folgt:

Die Emissionswahrscheinlichkeiten $b_i(k)$, zusammengefasst in der Matrix B sind nun durch entsprechende Gaußverteilungen bestimmt. Damit gilt für Emissionswahrscheinlichkeiten $b_i(k) = b_i(O_i = v_k) = \mathcal{N}(v_k, \mu_i, \sigma_i)$. Hierbei ist μ_i der Erwartungswert und σ_i die Standardabweichung der entsprechenden Normalverteilung, die zu dem betreffenden Zustand Z_i gehört. Die Parametermenge des Modells ändert sich entsprechend zu $\lambda = \{A, \mu, \sigma, \pi\}$ [Bishop (2009); Nguyen (2016)].

3 Beschreibung des Modells

Das grundlegende Modell ist dem von Nguyen [Nguyen (2016)] nachempfunden. Als Datenbasis dienen die Schluss- und Öffnungspreise sowie der Höchst- und Niedrigstpreis einer Aktie zu jedem Handelstag. Diese Beobachtungen seien als Observationsmenge zum Zeitpunkt t mit $O_t = \{O_t^{(1)}, O_t^{(2)}, O_t^{(3)}, O_t^{(4)}\}$ bezeichnet. Zur Vorhersage der Schlusskurse wird zunächst ein Lauf-Intervall der Länge T festgelegt. Dann werden die Parameter für das Modell auf Trainingsdaten dieser Länge trainiert. Dazu werden die Daten von T Handelstagen verwendet, deren letzter Datenpunkt den Zeitpunkt t_0 hat. Zusätzlich wird die (Log-)Likelihood $L(t_0)$ des Modells unter Berücksichtigung der Beobachtungen berechnet. Im nächsten Schritt wird versucht einen Tag in der Zeitreihe zu finden, für den das Modell eine ähnliche Likelihood hat. Zu diesem Zweck wird das Intervall um einen Handelstag in die Vergangenheit verschoben und die entsprechende Likelihood berechnet. Der letzte Tag des Intervalls entspricht nun also dem Zeitpunkt $t_0 - 1$. Dies wird kontinuierlich fortgesetzt, bis eine gewisse Anzahl Intervalle untersucht wurde. Nguyen verwendet das Intervall $I(t^*)$, deren Likelihood $L(t^*)$ der Likelihood des Ausgangs-Intervalls $I(t_0)$ am nächsten kommt, um die Vorhersage zu berechnen. Dazu wird die Differenz der Schlusspreise des letzten Tages des Intervalls und seines Folgetages berechnet, mit dem Vorzeichen der Differenz der Likelihoods für die beiden Intervalle multipliziert und zum Schlusspreis des Zieltages t_0 addiert. Die Vorhersage des Schlusspreises zum Folgetag ergibt sich also gemäß:

$$O_{t_0+1}^{(4)} = O_{t_0}^{(4)} + (O_{t^*+1}^{(4)} - O_{t^*}^{(4)}) * sign(L(t^*) - L(t_0)). \tag{1}$$

Zur Berechnung der Vorhersage des Schlusspreises für den darauffolgenden Tag wird das Ausgangs-Intervall um einen Tag nach vorne verschoben und der gesamte Vorgang wiederholt.

Da es sich bei den finalen Vorhersagewerten immer nur um die Schlusspreise handeln wird, wird im Folgenden zur besseren Lesbarkeit auf die Indexierung der Beobachtungen mit $O_t^{(4)}$ verzichtet.

Die Modellselektion, genauer gesagt die Auswahl der Anzahl N von Zuständen des HMM Ω_i für den entsprechenden Datensatz wird von Nguyen bereits vor dem Training der Modelle durch eine Voruntersuchung anhand des Akaike Informationskriteriums (AIC) und des Bayes Informationskriteriums (BIC) durchgeführt. Die Autoren testen hierbei für $I = \{1,2,3\}$ HMM Ω_i mit zwei bis vier Zuständen und wählen dann das HMM für die gesamte Vorhersage aus, das den geringsten BIC bzw. AIC für den betreffenden Datensatz verfügt. Die Ergebnisse werden durch den Vergleich des MAPE (mean absolute percentage error) für die Vorhersagen der verschiedenen Modelle bestätigt [Nguyen (2016)].

Der Hauptunterschied des in dieser Arbeit entwickelten Modells zu dem von Nguyen liegt in der Art der Modellselektion und der damit verbundenen Berechnung des Vorhersagewertes.

3.1 Model Averaging

Im Gegensatz zu Nguyen wird bei dem in dieser Arbeit verwendeten Verfahren keine strikte Modellselektion für die gesamte Vorhersage durchgeführt. Vielmehr wird angenommen, dass HMM mit einer unterschiedlichen Anzahl von Zuständen in der Lage sind verschiedene Charakteristiken und Muster in den Daten zu finden und für die Vorhersage nutzbar zu machen.

Um diesen Umstand auszunutzen, werden zunächst für jeden vorherzusagenden Datenpunkt jeweils für $I = \{1,...,4\}$ HMM Ω_i mit zwei bis fünf Zuständen trainiert. In einem weiteren Schritt werden dann drei verschiedene Methoden getestet, um auf dieser Basis eine Vorhersage zu treffen:

3.1.1 Naive Model Averaging (NMA)

Bei der ersten Methode zum Mitteln der Vorhersageergebnisse werden für jeden Datenpunkt jeweils das Modell mit dem besten AIC und das Modell mit dem besten BIC ausgewählt. Deren Vorhersagen für die Differenz der Schlusspreise werden dann jeweils genauso wie oben mit dem Vorzeichen der Differenz der Likelihoods der entsprechenden Modelle multipliziert. Schließlich werden diese Werte 50:50 gemittelt und zum Wert des Ausgangs-Tages addiert:

$$\hat{O}_{t_0+1} = O_{t_0} + 0.5 * (\Delta_{AIC} + \Delta_{BIC}) \tag{2}$$

mit

$$\Delta_{AIC} = (O_{t^*_{AIC}+1} - O_{t^*_{AIC}}) * sign(L(t^*_{AIC}) - L(t_0)); \tag{3}$$

$$\Delta_{BIC} = (O_{t^*_{BIC}+1} - O_{t^*_{BIC}}) * sign(L(t^*_{BIC}) - L(t_0)); \tag{4}$$

wobei t^*_{AIC} (t^*_{BIC}) der Zeitpunkt des bezeichnenden Datums für das Intervall $I(t^*_{AIC})$ ($I(t^*_{BIC})$) ist, für welches das Modell Ω^*_i den geringsten Wert für das AIC (BIC) hat.

3.1.2 Total Model Averaging (TMA)

Im Vergleich zum naiven Ansatz der 1. Methode, wird beim Total Model Averaging nicht nur die Information der Informationskriterien, also wie gut das Modell die Daten beschreibt, verwendet.

Stattdessen wird zusätzlich die Information benutzt, wie nah sich die Likelihood des ausgewählten Intervalls tatsächlich an der Likelihood des Ausgangs-Intervalls befindet. Dazu werden mit Hilfe des AIC, BIC und dem Absolut-Wert der Likelihood-Differenz $\Theta = abs(L(t^*) - L(t_0))$ Gewichte berechnet, mit denen dann die Vorhersagen der einzelnen Modelle Ω_i, $i = 1, ..., 4$ bewertet und anschließend verrechnet werden. Um im Folgenden eine zu große Instabilität des Modells zu vermeiden, die aus der Multiplikation mit sehr kleinen Θ resultieren kann, werden alle Θ mit einem Wert kleiner als 1 auf den Wert 1 normiert. Zunächst wird dann jeweils die Gesamtsumme der Werte des AIC, BIC und von Θ gebildet. Da für alle drei Größen gilt, dass ein kleiner Wert besser ist als ein großer, werden die Gesamtsummen (mit $K = I$ zur übersichtlicheren Indexierung) wieder jeweils durch die einzelnen Werte geteilt. Dies führt dazu, dass kleine Werte nun verhältnismäßig ein größeres Gewicht erhalten, als große Werte. Die entstehenden Zwischenergebnisse $\overline{AIC}_i$ und $\overline{BIC}_i$ des Modells Ω_i erhält man dann durch Multiplikation mit dem entsprechenden $\overline{\Theta}_i$, wobei

$$\overline{\Theta}_i = \frac{\sum_K \Theta_k}{\Theta_i}; \tag{5}$$

$$\overline{AIC}_i = (\frac{\sum_K AIC_k}{AIC_i}) * \overline{\Theta}_i; \tag{6}$$

$$\overline{BIC}_i = (\frac{\sum_K BIC_k}{BIC_i}) * \overline{\Theta}_i. \tag{7}$$

Die Gewichte ω_i erhält man im nächsten Schritt indem die Einzelwerte $\overline{AIC}_i$ und $\overline{BIC}_i$ normalisiert werden und ihr Mittelwert berechnet wird. Dadurch wird zudem gewährleistet, dass $\sum_I \omega_i = 1$ gilt.

$$\omega_i = 0.5 * (\frac{\overline{AIC}_i}{\sum_K \overline{AIC}_k} + \frac{\overline{BIC}_i}{\sum_K \overline{BIC}_k}) \tag{8}$$

Zur Berechnung der finalen Vorhersage werden schließlich die vorhergesagten Differenzen jedes Modells, die analog zu Modell 1 mit dem Vorzeichen der Differenz der Likelihoods der entsprechenden Modelle multipliziert wurden, entsprechend gewichtet und aufsummiert:

$$\hat{O}_{t_0+1} = O_{t_0} + \sum_I \omega_i \Delta_i \tag{9}$$

wobei

$$\Delta_i = (O_{t_i^*+1} - O_{t_i^*}) * sign(L(t_i^*) - L(t_0)). \tag{10}$$

3.1.3 Selective Model Averaging (SMA)

Die 3. Methode basiert auf dem Verfahren der zweiten Methode, verzichtet jedoch auf den unmittelbaren Einfluss von Θ auf die Gewichte durch direkte Multiplikation. Stattdessen wird in einem vorgelagerten Schritt anhand von Θ eine Modellselektion durchgeführt. Es werden nur Modelle Ω_i verwendet, deren Likelihood „nah genug“ an der Likelihood des Ausgangs-Intervalls liegen, deren Θ_i also kleiner als eine Schranke τ ist. Um den Umstand „nah genug“ verwertbar zu machen, wird ein τ gewählt, das gewisse Anforderungen erfüllt: Genauer soll für eine solche Schranke gelten, dass in dem Fall, in dem alle Modelle ähnliche Werte haben, egal ob diese alle recht groß oder recht klein sind, alle Werte verwendet werden, um die Vorhersage zu verbessern. Besteht dagegen eine große Differenz zwischen den einzelnen Werten, so sollen nur die kleineren davon verwendet werden. Dieses Verhalten erreicht man zum Beispiel, indem man $\tau = 1.25 * mean(\Theta)$ setzt. Für die ausgewählten Modelle Ω_j, $j \in J, J \subseteq I$ (mit entsprechend hier $K = J$), werden dann analog zu Methode 2 Gewichte ω_j berechnet, wobei sich die Gleichungen (6) und (7) auf $\overline{AIC}_j = (\sum_K AIC_k)/AIC_j$ und $\overline{BIC}_j = (\sum_K BIC_k)/BIC_j$ reduzieren.

Ein weiterer Unterschied zum Modell von Nguyen ist zuletzt der, dass zum Training der HMM nicht die absoluten Werte für Schluss- und Öffnungspreise sowie den Höchst- und Niedrigstpreis einer Aktie zu jedem Handelstag verwendet werden, sondern deren erste Differenzen. Es werden also anstatt der statischen Preise deren tägliche Veränderungen als Information genutzt. Für die finale Vorhersage der Schlusskurse werden dann jedoch trotzdem die unveränderten Schlusspreise der entsprechenden Tage verwendet.

4 Analyse

Um die Performance der verschiedenen Methoden zu vergleichen, wird der sogenannte RMSE (Root Mean Squared Error) verwendet. Dabei gilt für die Vorhersage $\hat{O}_t$ und den beobachteten Wert O_t die Beziehung:

$$RMSE(\hat{O}) = \sqrt{\frac{\sum_{t=1}^{n}(\hat{O}_t - O_t)^2}{n}}. \tag{11}$$

4.1 Versuchsaufbau

Zur Evaluation der drei verschiedenen Methoden wurde das Verfahren auf die Zeitreihen der Aktienkurse von vier großen bekannten Unternehmen angewendet. Genauer handelt es sich hierbei um die Aktienkurse von IBM, Apple, Facebook und Google. Die Daten stammen von Yahoo! Finance [Yahoo! Inc. (2017)].

Die Zeitreihen gehen dabei jeweils vom 01.08.2014 bis zum 30.11.2016. Als Trainingsdaten wurde das Intervall 01.08.2014 - 01.08.2016 ausgewählt. Dann wurden mit den trainierten Modellen die Schlusskurse der jeweiligen Aktien für die Handelstage im September bis November 2016 vorhergesagt.

4.2 Analyse der einzelnen Methoden

Die Untersuchung des RMSE für die Ergebnisse der einzelnen Methoden sind in Tabelle 1 dargestellt.

Methode	IBM	Apple	Facebook	Google
Naive Model Averaging	2.4141	1.9277	2.5259	10.9277
Total Model Averaging	1.9154	1.6207	1.9013	10.7346
Selective Model Averaging	1.9658	1.5978	1.8794	11.0118

Tabelle 1: RMSE der einzelnen Methoden für die jeweiligen Datensätze

Zudem sind die realen Zeitreihen (blau) und die entsprechenden Vorhersagen (rot) der Methoden in den folgenden Graphen gegeneinander abgetragen.

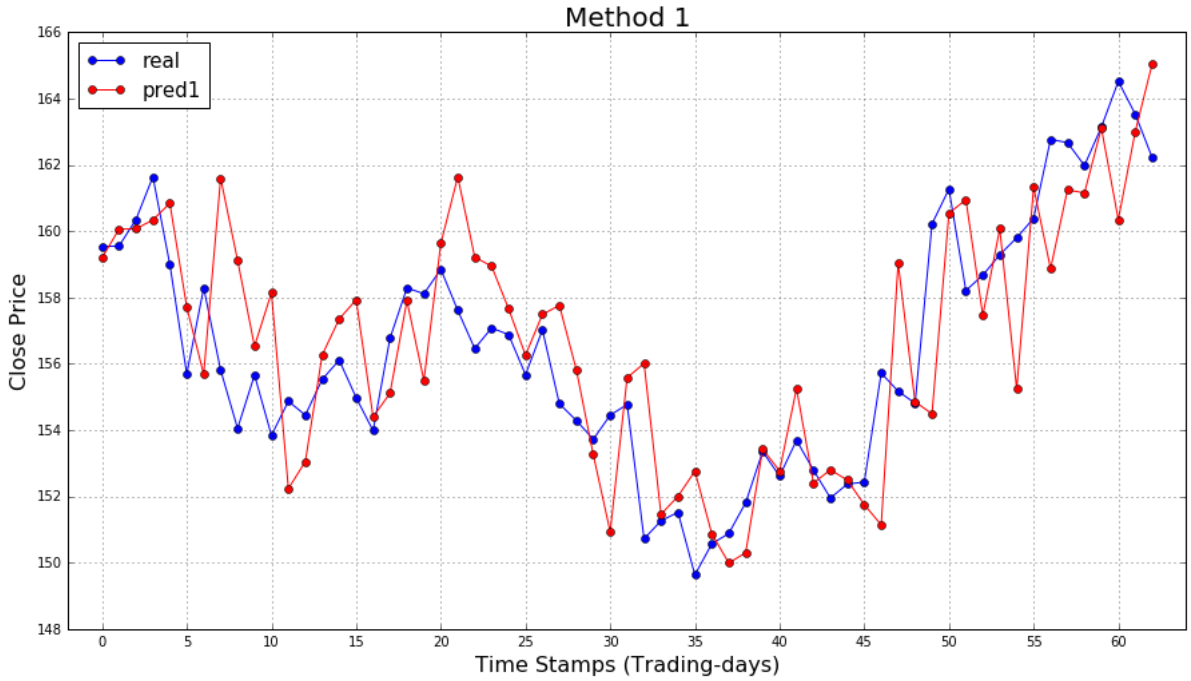

Abbildung 1: Zeitreihen der realen und vorhergesagten Werte (*NMA*) für **IBM**

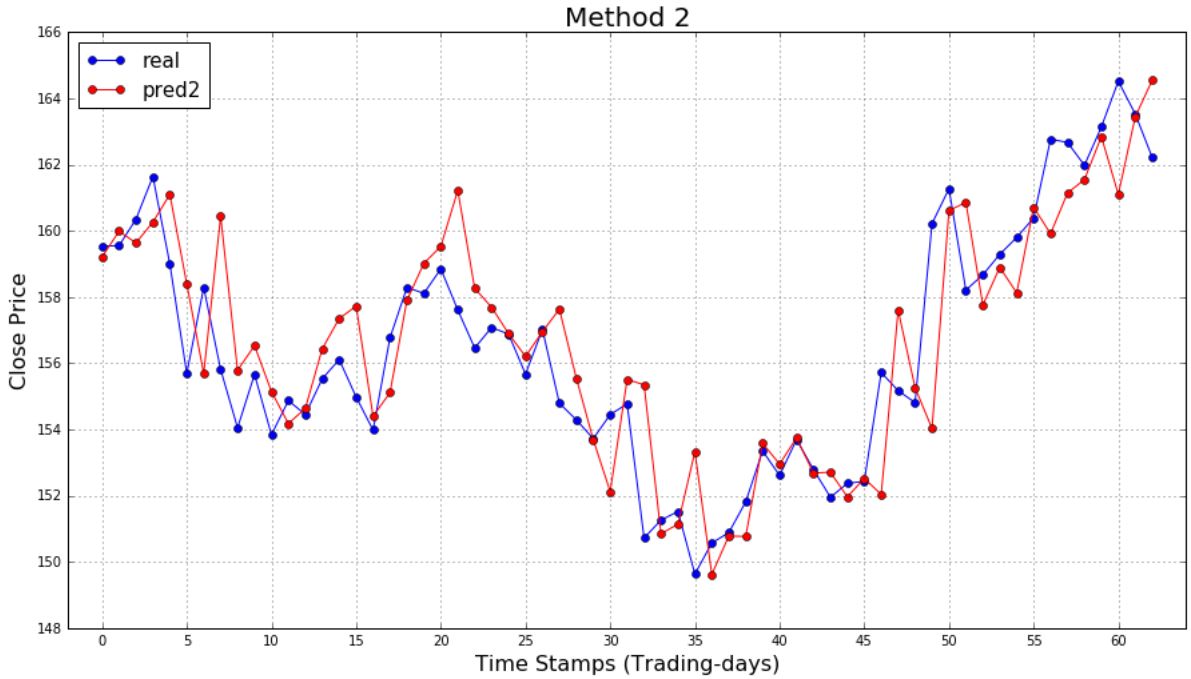

Abbildung 2: Zeitreihen der realen und vorhergesagten Werte (*TMA*) für **IBM**

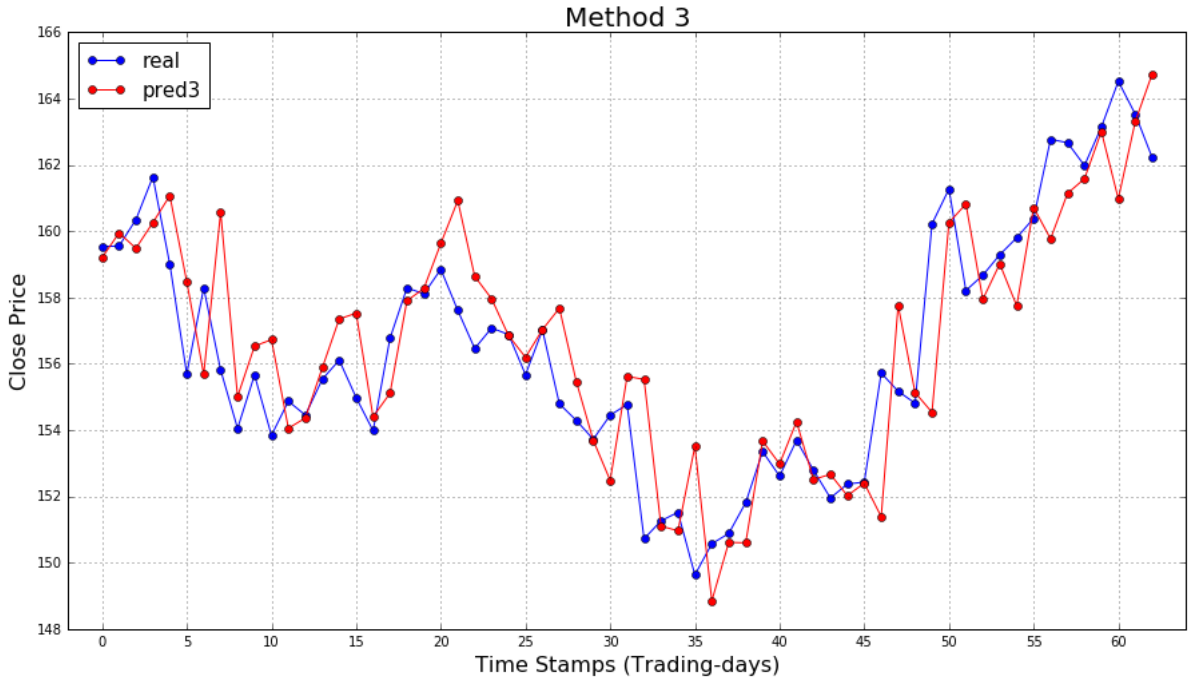

Abbildung 3: Zeitreihen der realen und vorhergesagten Werte (*SMA*) für **IBM**

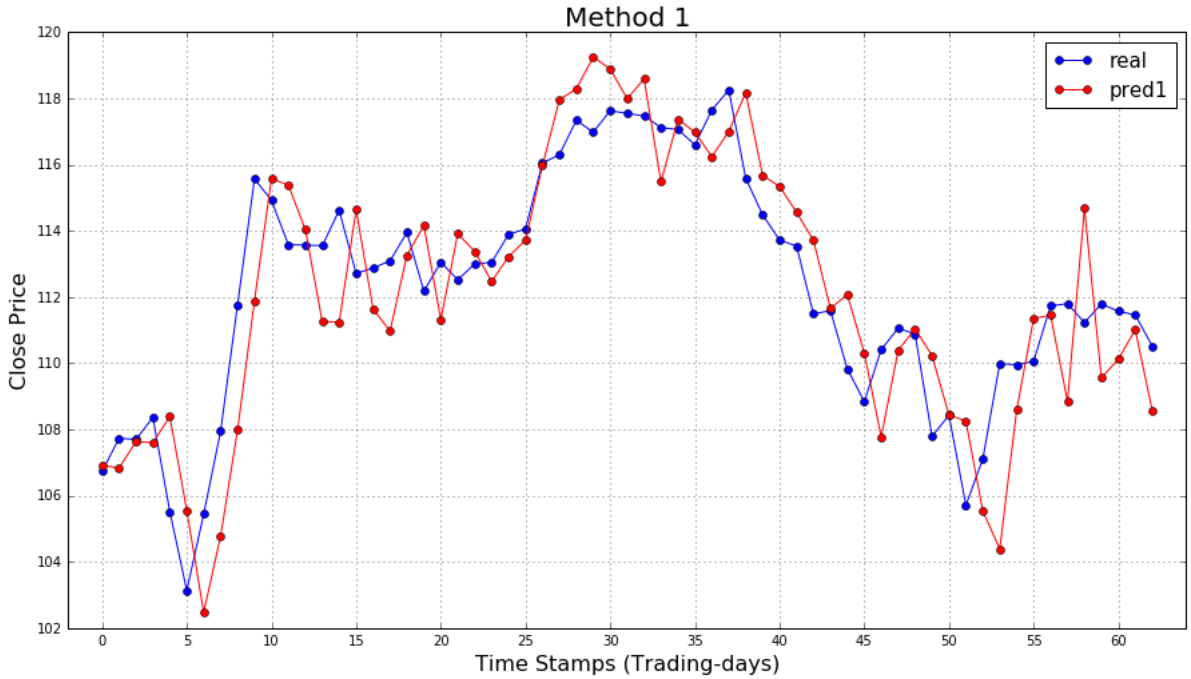

Abbildung 4: Zeitreihen der realen und vorhergesagten Werte (*NMA*) für **Apple**

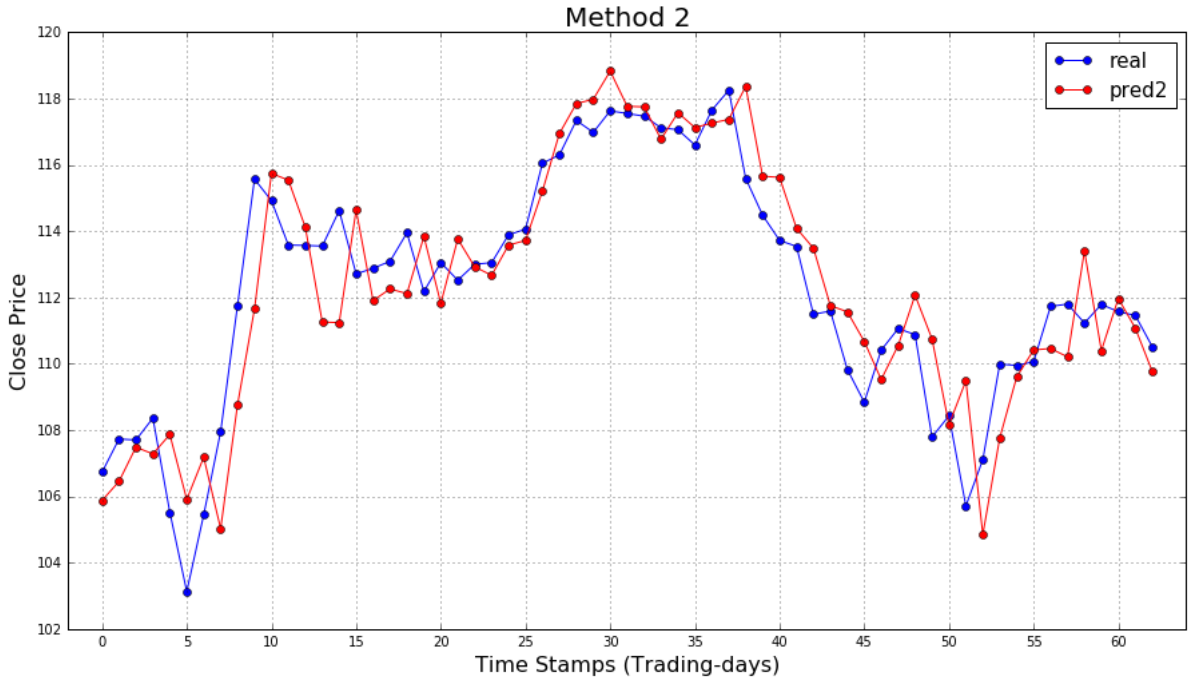

Abbildung 5: Zeitreihen der realen und vorhergesagten Werte (*TMA*) für **Apple**

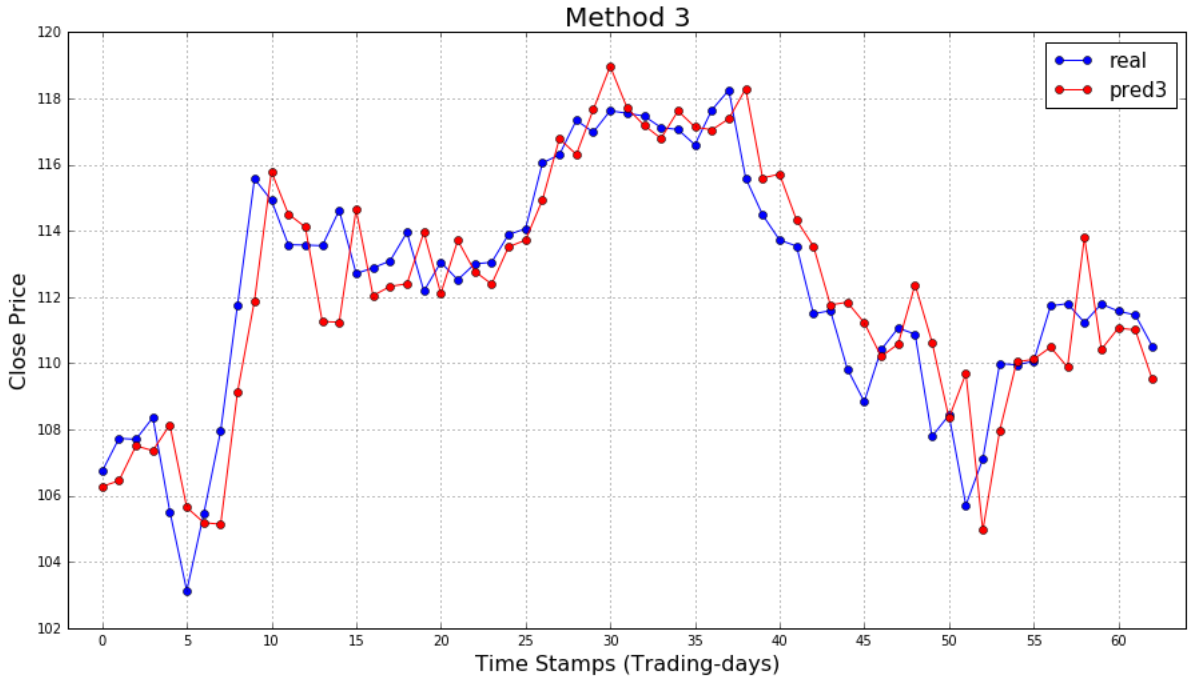

Abbildung 6: Zeitreihen der realen und vorhergesagten Werte (*SMA*) für **Apple**

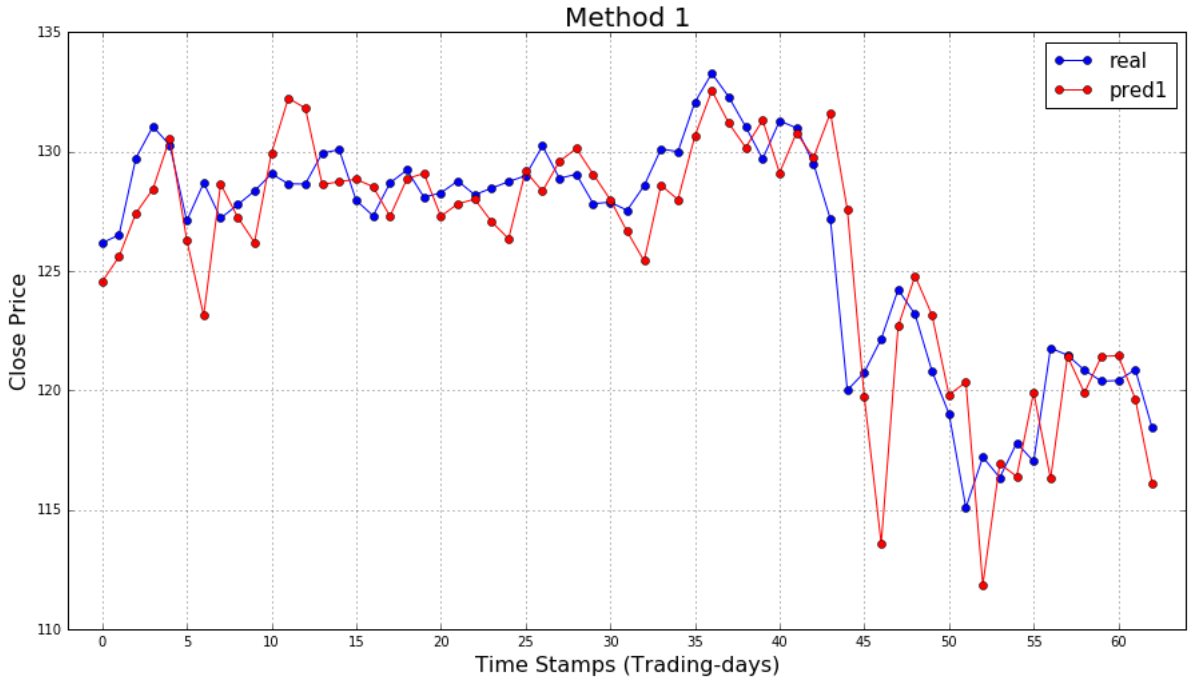

Abbildung 7: Zeitreihen der realen und vorhergesagten Werte (*NMA*) für **Facebook**

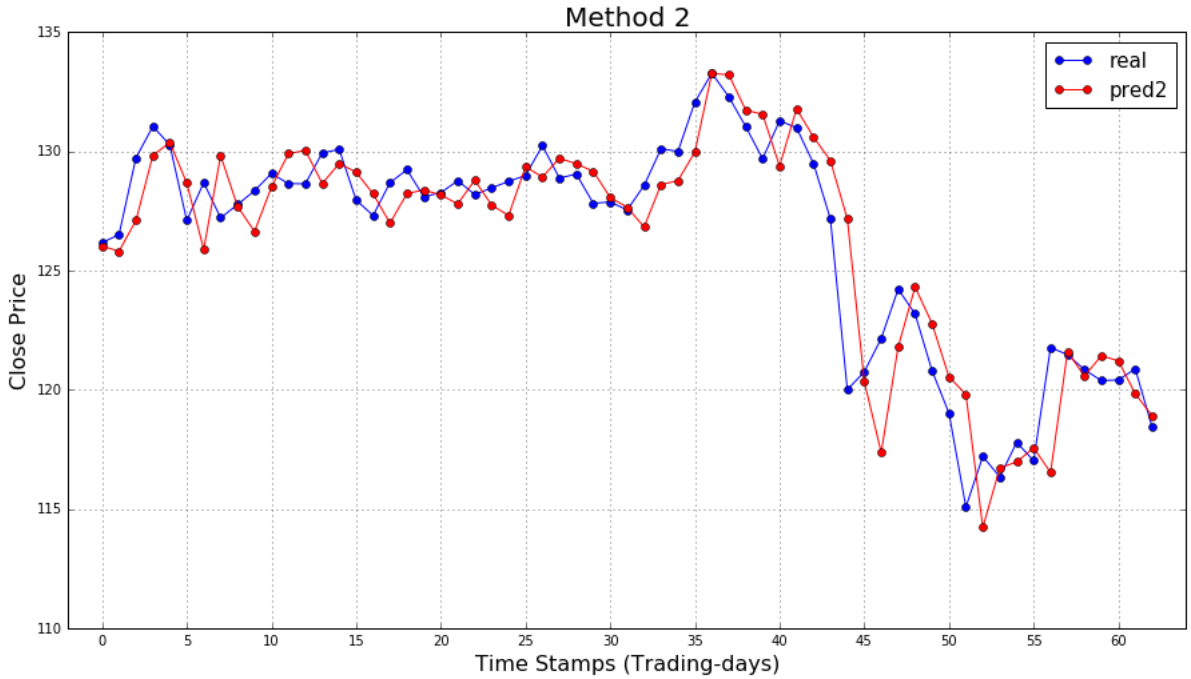

Abbildung 8: Zeitreihen der realen und vorhergesagten Werte (*TMA*) für **Facebook**

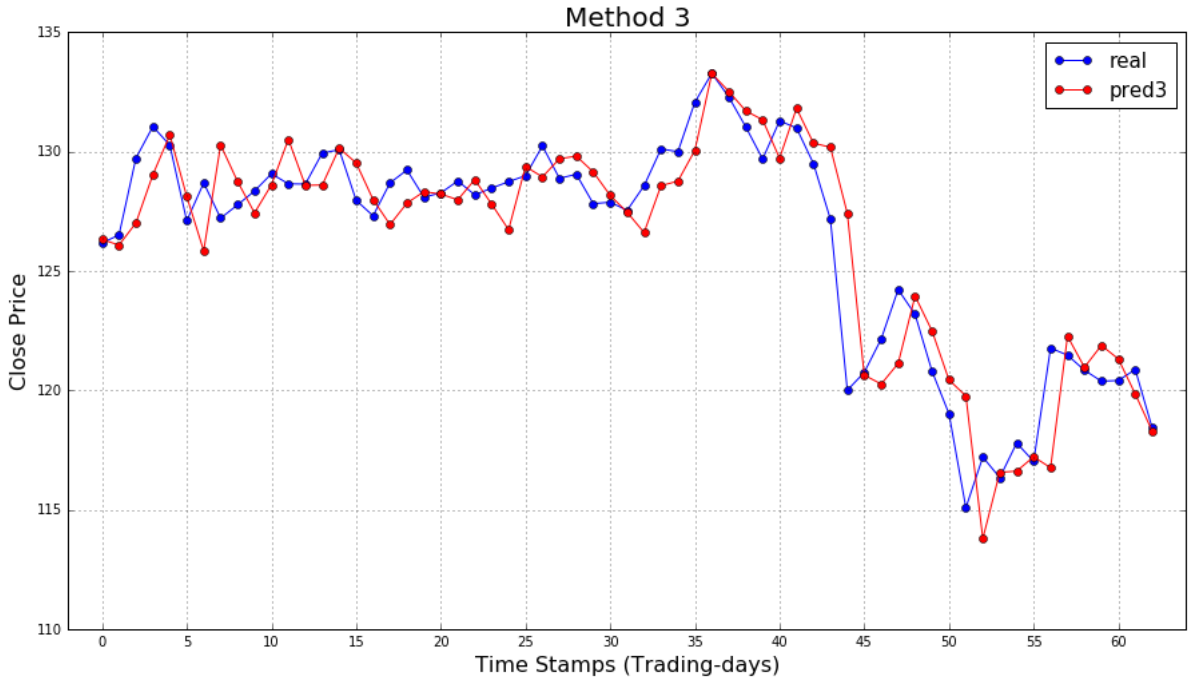

Abbildung 9: Zeitreihen der realen und vorhergesagten Werte (*SMA*) für **Facebook**

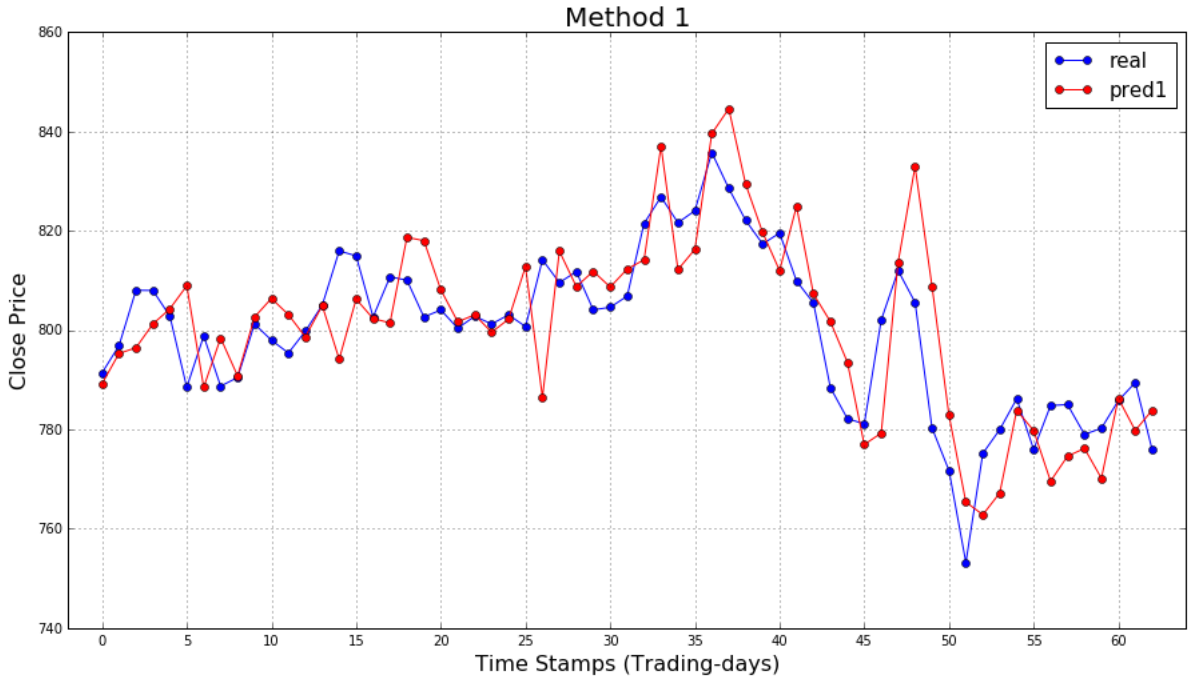

Abbildung 10: Zeitreihen der realen und vorhergesagten Werte (*NMA*) für **Google**

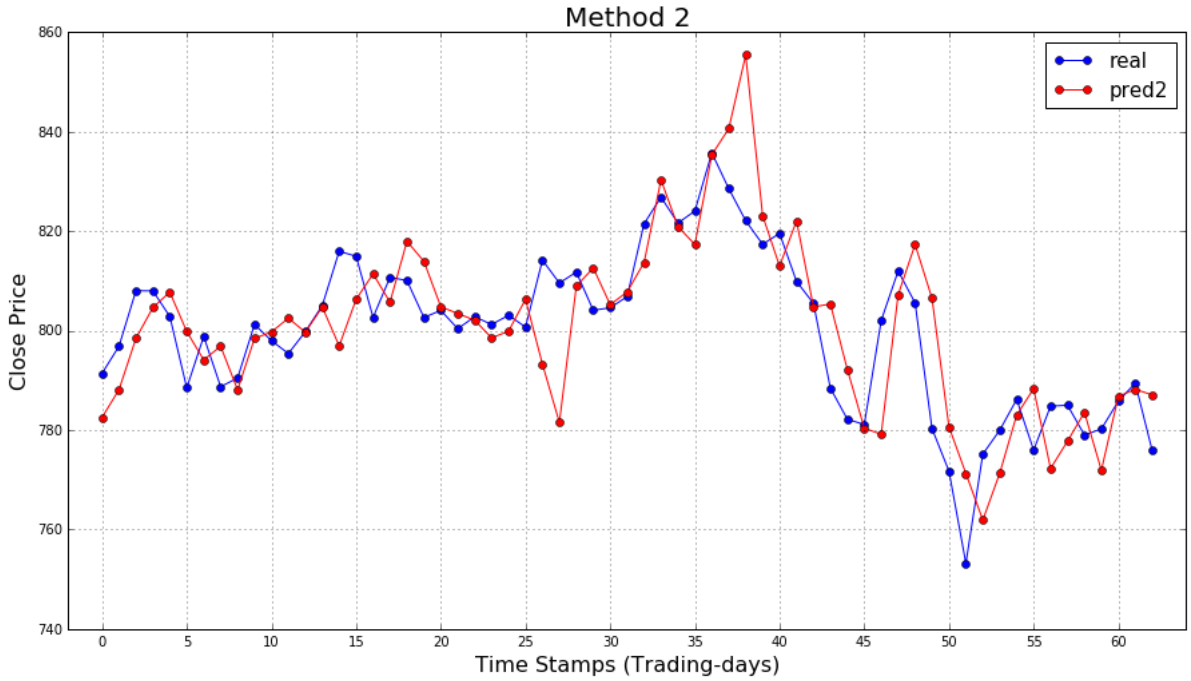

Abbildung 11: Zeitreihen der realen und vorhergesagten Werte (*TMA*) für **Google**

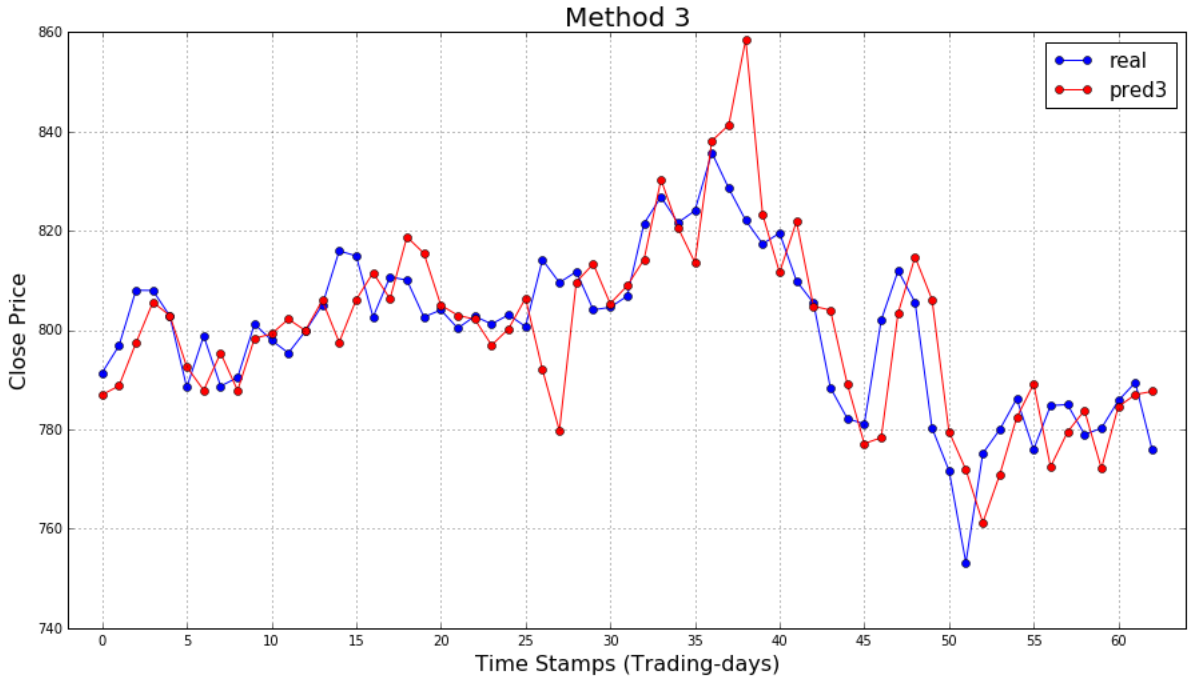

Abbildung 12: Zeitreihen der realen und vorhergesagten Werte (*SMA*) für **Google**

Betrachtet man den RMSE der einzelnen Methoden in Tabelle 1, kann man erkennen, dass im Fall der Aktienkurse von IBM, Apple und Facebook die beiden fortgeschrittenen Methoden *TMA* und *SMA* dem einfacheren *NMA* durchgehend deutlich überlegen sind. So unterscheidet sich der RMSE im Fall von IBM verglichen mit *NMA* um 0.5 (*TMA*) bzw. 0.45 (*SMA*). Noch höher ist die Differenz für die Facebook Daten, diese beträgt 0.62 für *TMA* sowie 0.65 für *SMA*. Auch für die Apple Zeitreihe ergibt sich schließlich ein Unterschied von immerhin noch 0.31 (*TMA*) und 0.33 (*SMA*). Diese Ergebnisse stützen die Annahme, dass HMM mit unterschiedlich vielen Zuständen verschiedene Aspekte der Zeitreihen abbilden und damit eine geeignete Gewichtung und Mittelung der Ergebnisse zu besseren Vorhersagen führt. Dies liegt u.a. an dem Umstand, dass durch die gewichtete Mittelwertbildung die einseitige Abweichung um große Beträge abgeschwächt wird, wenn nicht alle Modelle einen solchen Ausschlag prognostizieren. Genau dieser Effekt ist auch in den Abbildungen 1 bis 9 zu erkennen. Hier sind die geplotteten Zeitreihen der Vorhersagen durch *NMA* teilweise deutlich höheren Schwankungen unterworfen.

Die einzige Zeitreihe, für die diese Beziehung nicht gilt, ist der Aktienkurs von Google. Hier performt *NMA* zumindest um 0.08 besser als *SMA*, während *TMA* dennoch um weitere 0.19 darunter liegt. Dies lässt sich aber u.a. damit erklären, dass die Varianz für die Google Zeitreihe verglichen mit den anderen Daten signifikant höher ist (vgl. Tabelle 2).

Zeitreihe	Varianz
IBM:	13.5711
Apple:	12.6693
Facebook:	21.4236
Google:	255.4551

Tabelle 2: Varianz der Zeitreihendaten für Sep.-Nov. 2016

Vergleicht man die Ergebnisse von *TMA* und *SMA* in Tabelle 1, so lassen sich keine eindeutigen Schlüsse daraus ziehen, welches Verfahren besser abschneidet. Beide Methoden erzielen ähnlich gute Ergebnisse, deren RMSE sich meist nur um 0.02 bis 0.05 unterscheidet. Die einzige Ausnahme zu diesem Verhalten ist die Zeitreihe von Google, wo der Unterschied 0.28 beträgt, was aber erneut der weit größeren Varianz geschuldet ist. Allgemein lässt sich sagen, dass sowohl *TMA* als auch *SMA* ähnlich gut performen. Dabei schneidet *TMA* bei den Zeitreihen für IBM und Google besser ab, während *SMA* für Apple und Facebook die besseren Vorhersagen liefert.

5 Fazit

Zusammenfassend lässt sich sagen, dass die fortgeschrittenen Methoden *TMA* und *SMA* dem einfacheren *NMA*, dessen Verfahren dem Modell von Nguyen am ähnlichsten ist, in den meisten Fällen überlegen sind. Diese Schlussfolgerung ergibt sich aus der Analyse der Vorhersageergebnisse der einzelnen Methoden für vier Zeitreihen-Datensätze von Aktienkursen, bei denen jeweils der Schlusskurs für den Folgetag vorhergesagt wurde.

Der beobachtete Effekt entsteht vor allem dadurch, dass mit Hilfe der gewichteten Mittelwertbildung die einseitige Abweichung um große Beträge abgeschwächt wird, wenn nicht alle Modelle einen solchen Ausschlag prognostizieren, was dann im Allgemeinen zu stabileren Vorhersagen führt. Es bestätig sich also die Annahme, dass durch das Mitteln verschiedener HMM die Vorhersagequalität signifikant verbessert werden kann.

In einer zukünftigen Untersuchung wäre es nun interessant herauszufinden, wie die in dieser Arbeit entwickelten Verfahren im direkten Vergleich zu anderen state-of-the-art Algorithmen abschneiden, in welchen Fällen sie viel versprechende Ergebnisse liefern und in welchen Szenarien sie mit Problemen konfrontiert sind.

Anmerkung Grundlegend muss außerdem darauf hingewiesen werden, dass durch die Verwendung des Baum-Welch- bzw. EM-Algorithmus beim Training der HMM eine gewisse Variabilität der Modelle entsteht. Da der Algorithmus nur lokale Maxima approximiert [Nguyen (2016)], ist es möglich, dass sich die Parameter und dementsprechend auch die Modelle in verschiedenen Läufen durchaus unterscheiden. Dadurch ändern sich auch die Werte für die Likelihoods der verschiedenen Datenpunkte und Modelle, was schließlich keinen unerheblichen Einfluss auf die finalen Vorhersagen hat. Da die Likelihood durch das AIC und BIC bei *TMA* und *SMA* außerdem direkt als numerischer Wert bei den Gewichten in die Berechnung der Vorhersage eingeht, ist hier der Einfluss umso größer. Die in dieser Arbeit präsentierten Ergebnisse können daher nur als exemplarisch angesehen werden. Zur fundierten empirischen Untersuchung und Evaluation der Verfahren wäre eine umfassende Simulationsstudie notwendig.

Literatur

BISHOP, C. M. (2009): *Pattern Recognition and Machine Learning*, Information Science and Statistics, New York [u.a.]: Springer, 11. (corrected 8th printing) ed.

HAMILTON, J. D. AND G. LIN (1996): "Stock market volatility and the business cycle," *Journal of Applied Econometrics*, 11, 573–593.

HASSAN, M. R. AND B. NATH (2005): "Stock market forecasting using hidden Markov model: A new approach," in *5th International Conference on Intelligent Systems Design and Applications, 2005. ISDA'05. Proceedings.*, IEEE, 192–196.

LEE, J. AND M. SHIN (2009): "Stock Forecasting using Hidden Markov Processes," `http://cs229.stanford.edu/proj2009/ShinLee.pdf`.

NGUYEN, N. (2016): "Stock Price Prediction using Hidden Markov Model," `https://editorialexpress.com/cgi-bin/conference/download.cgi?db_name=SILC2016&paper_id=38`.

RABINER, L. R. (1989): "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, 77, 257–286.

YAHOO! INC. (2017): "Yahoo! Finance Services," `https://finance.yahoo.com/`.

Declaration of Authorship

Ich bestätige hiermit, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Alle Passagen die sowohl wörtlich als auch im Allgemeinen von anderen Quellen übernommen wurden, sind als solche gekennzeichnet.

Karlsruhe, 28. Februar 2017

Jonas Falkner

A Python-Code

```
##################################################
#download data from yahoo! finance
##################################################

# -*- coding: utf-8 -*-
"""
Created on Wed Jan 11 22:11:23 2017

@author: Jonas

data source: http://finance.yahoo.com/
"""



from pandas_datareader import data as pdr
from datetime import datetime



#download data
ibm = pdr.DataReader('IBM', 'yahoo', start=datetime(2014, 8, 1),
                     end=datetime(2016, 11, 30))
aapl = pdr.DataReader('AAPL', 'yahoo', start=datetime(2014, 8, 1),
                      end=datetime(2016, 11, 30))
fb = pdr.DataReader('FB', 'yahoo', start=datetime(2014, 8, 1),
                    end=datetime(2016, 11, 30))
googl = pdr.DataReader('GOOGL', 'yahoo', start=datetime(2014, 8, 1),
                       end=datetime(2016, 11, 30))

#print first few lines of data
print(ibm.head())
print(aapl.head())
print(fb.head())
print(googl.head())



#export and save as csv files
ibm.to_csv('IBM_stock.csv', sep=',')
```

```
aapl.to_csv('Apple_stock.csv', sep=',')
fb.to_csv('Facebook_stock.csv', sep=',')
googl.to_csv('Google_stock.csv', sep=',')
```

ML2_data_download.py

```
################################################
#data exploration & viz
################################################

# -*- coding: utf-8 -*-
"""
Created on Fri Jan 13 20:17:39 2017

@author: Jonas
"""









import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt




#load data
ibm = pd.read_csv('IBM_stock.csv', index_col=0)
aapl = pd.read_csv('Apple_stock.csv', index_col=0)
fb = pd.read_csv('Facebook_stock.csv', index_col=0)
googl = pd.read_csv('Google_stock.csv', index_col=0)




#show basic summaries

print(ibm.describe())
print(aapl.describe())
```

```
print(fb.describe())
print(googl.describe())



#data visualization
#-----------------------------------------------------------------------

#timeseries plots

fig, (ax1,ax2,ax3,ax4) = plt.subplots(ncols=4,figsize=(32, 8))
sns.tsplot(ibm.Open, ax=ax1)
sns.tsplot(aapl.Open, ax=ax2)
sns.tsplot(fb.Open, ax=ax3)
sns.tsplot(googl.Open, ax=ax4)

ax1.set_title('IBM')
ax2.set_title('Apple')
ax3.set_title('Facebook')
ax4.set_title('Google')



#distributions

fig, (ax1,ax2,ax3,ax4) = plt.subplots(ncols=4,figsize=(32, 8))
sns.distplot(ibm.Open, ax=ax1)
sns.distplot(aapl.Open, ax=ax2)
sns.distplot(fb.Open, ax=ax3)
sns.distplot(googl.Open, ax=ax4)

ax1.set_title('IBM')
ax2.set_title('Apple')
ax3.set_title('Facebook')
ax4.set_title('Google')



#variance and range
```

```
print('--IBM--')
print(ibm.var())
print('mean diff. of High/Low   ',ibm.High.mean()-ibm.Low.mean())
print('--Apple--')
print(aapl.var())
print('mean diff. of High/Low   ',aapl.High.mean()-aapl.Low.mean())
print('--Facebook--')
print(fb.var())
print('mean diff. of High/Low   ',fb.High.mean()-fb.Low.mean())
print('--Google--')
print(googl.var())
print('mean diff. of High/Low   ',googl.High.mean()-googl.Low.mean())
```

ML2_data_expl_viz.py

```
################################################
#Model
################################################


# -*- coding: utf-8 -*-
"""
Created on Sat Jan 14 11:21:26 2017

@author: Jonas
"""


import pandas as pd
import numpy as np
from hmmlearn.hmm import GaussianHMM as ghmm
import math
import copy

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)


```

```
25
#FUNCTIONS
27 ##############################################

29 #create functions for calculation of information criteria
class model_fit_measure:
31
    def AIC(L,k):
33      return(-2*math.log(abs(L))+2*k)

35  def BIC(L,k,M):
        return(-2*math.log(abs(L))+k*math.log(M))
37



39


41 #basic hmm training function
#-------------------------------------------------------------
43 def build_hmm_model(data, states, start_prob=None, mu=None, sigma=None,
                     ex = [], max_iterations=1000, EM_threshold=0.01,
45                   conv=False, show=True):

47  if show == True:
        print('Fitting and decoding... ', end='')
49
    #create basic model
51  model = ghmm(n_components = states,
                 covariance_type='diag',
53               n_iter = max_iterations,
                 tol = EM_threshold)
55


57  if 'sp' in ex:
        model.startprob_ = start_prob
59
    if 'm' in ex:
61      model.means_prior = mu

63  if 's' in ex:
        model.covars_prior = sigma
65
```

```
67      #fit model to data
        model.fit(data)
69
        #convergence information
71      if conv == True:
            print('\n--Convergence--')
73          print(model.monitor_.converged)
            print(model.monitor_)
75
        if show == True:
77          print('done')


79
        #calculate likelihood of observations given model
81      mod_likelihood = model.score(data)

83      #calculate information criteria
        k = states*8  ##mu and sigma for 4 dim
85      n = len(data.index)  ##number of observations

87      aic = model_fit_measure.AIC(mod_likelihood,k)
        bic = model_fit_measure.BIC(mod_likelihood,k,n)
89


91      #format output as python data type "dictionary"
        out = {'model' : model,
93             'likelihood' : mod_likelihood,
               'aic' : aic,
95             'bic' : bic
               }
97


99      return(out)

101


103


105 #calculates parameters for future iterations by one initial run of hmm
    #-----------------------------------------------------------------
```

```
107 def hmm_init(data):
109     out = []
111     print('Initializing...')
113     for i in range(2,6):
115         #build model
            model = build_hmm_model(data=data, states=i, show=False)
117
            #get parameters
119         mod_0 = model['model']
            start_prob_0 = mod_0.startprob_
121         mu_0 = mod_0.means_
            sigma_0 = mod_0.covars_
123
            #format output as list
125         lambda_0 = [start_prob_0, mu_0, sigma_0]
            out.append( lambda_0)
127
    return(out)
129
131
133
    #calculates the predicted difference of todays value and
135 #the next-day value of the Close Price
    #-----------------------------------------------------------
137 def predict_diff(T, data_train, data_predict, states,
                      start_prob, mu, sigma, conv=False):
139
141     t_init = len(data_train.index)
        ##index of the last entry of the training data
143
        t0 = t_init-1
145     ##index of the 2nd last entry of the training data --> point to start
        prediction
```

```
147
    data_t = data_train.iloc[(t_init-T):t_init]
149
151 #build hmm model with initialization parameters
    model = build_hmm_model(data=data_t, states=states,
153                         start_prob=start_prob, mu=mu,
                            sigma=sigma, ex=['sp','m','s'], show=True)
155
157 mod = model['model']
    L_target = model['likelihood']
159
    #convergence information
161 if conv == True:
        conv = mod.monitor_.converged
163     print('converged: ', end='')
        print(conv)
165
167 val = []
    cond = []
169
    #use interval of length T and move it one datapoint back in each run
171 for i in range(1,(t0-T)):
        t_b = (t0-i)  ##define borders of interval T
173     t_a = (t_b-T)
        seq = data_train.iloc[t_a:t_b]
175
        likelihood = mod.score(seq)
177     ##predict likelihood of the model for the "new" data

179     l_diff = (likelihood-L_target)
        ##calculate diff of likelihood of "new" data and actual data (t0)
181
        cond.append( abs(l_diff))
183
        sign = np.sign(l_diff)  ##get the sign of the likelihood diff
185
        #calculate diff of Close Price of the current day and its successor
187     val.append((data_predict.Close.iloc[(t_b+1)] - data_predict.Close.iloc[(
```

```
    t_b)])*sign)


    start_prob = mod.startprob_
    mu = mod.means_
    sigma = mod.covars_

    #format output
    out = {'res' : pd.DataFrame({'cond':cond,
                    'val': val}),
           'aic' : model['aic'],
           'bic' : model['bic'],
           'sp' : start_prob,
           'mu' : mu,
           'sigma' : sigma}

    return(out)









#DATA
##########################################################



#load data  (seperate runs for each import data file)
#-------------------------------------------------------
data = pd.read_csv('IBM_stock.csv', index_col=0)
#data = pd.read_csv('Apple_stock.csv', index_col=0)
#data = pd.read_csv('Facebook_stock.csv', index_col=0)
#data = pd.read_csv('Google_stock.csv', index_col=0)


''' --> use August 2014 - 2016 for training, then predict Sep.-Nov. 2016 '''

#data used for prediction
#----------------------------------------
#only use Open, Close, High, Low -price
```

```
data_predict = data.iloc[1:,0:4]


#data used for training: use 1st differences
#-------------------------------------------------------
data_train = data

#calculate differences
data_train.Open = data.Open.diff()
data_train.High = data.High.diff()
data_train.Low = data.Low.diff()
data_train.Close = data.Close.diff()

#only use Open, Close, High, Low-price
data_train = data_train.iloc[1:,0:4]









#CALCULATIONS
############################################################


#walking interval length
T = 100

#get complete initial training data (August 2014 - 2016)
data_init = data_train[data_train.index < '2016-09-01']



#walking index
t_walk = len(data_init.index)-1


#train initial hmm on full initial training set to get
#good initial par for algorithm
res = hmm_init(data_init)
```

```
269
#set initial start probabilities to uniform dist
271 for p in range(0,4):
        res[p][0] = [(1/(p+2))]*(p+2)
273
275 #create df
    final_diff = pd.DataFrame(columns=('diff1', 'diff2', 'diff3'))
277
279 #main calculation loop
    #-------------------------------------------------------------------------
281 for q in range(0,(len(data_predict.index)-len(data_init.index))):

283     print()
        print('LOOP {}'.format((q+1)))
285     #print(data_predict.index[t_walk+1])

287     #data for training model and predict data points
        data_t = data_train.iloc[0:t_walk]
289     data_pred = data_predict.iloc[0:t_walk]

291
        pred = []
293
        #train 4 hmms with 2-5 states and predict with each
295     for i in range(0,4):

297         #get initial hmm parameters
            sp = res[i][0]
299         mu = res[i][1]
            s = res[i][2]
301
            #format covariance values for input
303         sigma = []
            for j in range(0,len(s)):
305             sigma.append(np.diag(s[j]))

307         #number of states
            st = i+2
309
```

```
        #calculate differences
311     diff_obj = predict_diff(T=T, data_train=data_t,
                                data_predict=data_pred,
313                             states=st, start_prob=sp,
                                mu=mu, sigma=sigma, conv=True)
315
        #save parameters of this model as init par for
317     #successing model (successive update of init par)
        res[i][0] = diff_obj['sp']
319     res[i][1] = diff_obj['mu']
        res[i][2] = diff_obj['sigma']
321
        #get values of information criteria and the list of
323     #predicted diff with corresponding likelihood diff
        aic = diff_obj['aic']
325     bic = diff_obj['bic']
        res_list = diff_obj['res']
327
        #get diff that is most likely
329     diff_pred_idx = res_list.idxmin(0)
        diff_pred = res_list.iloc[diff_pred_idx['cond']]
331
        #save result in multi dimensional list
333     pred.append([aic,bic,diff_pred])

335 #update walking index
    t_walk = (t_walk+1)
337
339
341
    #final results for predicted differences (different averaging methods)
343 #-------------------------------------------------------------------------

345
    #METHOD 1
347 #average value of model with best AIC and model with best BIC 50/50

349 #read out results from list and create df
    x_aic = []
```

```
    x_bic = []
    x_cond = []
    x_val = []

    for k in range(0,4):
        x_aic.append(pred[k][0])
        x_bic.append(pred[k][1])
        x_cond.append(pred[k][2]['cond'])
        x_val.append(pred[k][2]['val'])

    cal = pd.DataFrame({'aic':x_aic,
                        'bic':x_bic,
                        'cond':x_cond,
                        'val':x_val})

    #calculate prediction as average of value with
    #best AIC and value with best BIC
    final1 = ((cal.iloc[cal.idxmin(0)['aic']]['val']+
                        cal.iloc[cal.idxmin(0)['bic']]['val'])*0.5)









    #METHOD 2
    #total model averaging
    '''
    #calculate weights by BIC and AIC "inversely" normalized by their totals
    #(since small aic or bic ist better than high),
    #each multiplied by cond (=likelihood) and normalized again for all models
    #and then calculate weighted average of predicted differences
    #finally average the BIC and AIC result values 50/50
    '''

    pred2 = copy.deepcopy(pred)


    #if the abs difference of the likelihoods is smaller than 1,
    #it will be set to 1 to avoid to much instability when multiplying
    for k in range(0,4):
        if pred2[k][2]['cond'] < 1:
```

```
            pred2[k][2]['cond'] = 1
        else:
            pred2[k][2]['cond'] = pred2[k][2]['cond']


    #calculate 1st totals
    aic_tot1 = 0
    bic_tot1 = 0
    cond1 = 0

    for k in range(0,4):
        aic_tot1 = (aic_tot1 + pred2[k][0])
        bic_tot1 = (bic_tot1 + pred2[k][1])
        cond1 = (cond1 + pred2[k][2]['cond'])

    for k in range(0,4):
        pred2[k][0] = (aic_tot1/pred2[k][0])
        pred2[k][1] = (bic_tot1/pred2[k][1])
        pred2[k][2]['cond'] = (cond1/pred2[k][2]['cond'])

    #multiply by likelihood difference
    for k in range(0,4):
        pred2[k][0] = (pred2[k][0]*pred2[k][2]['cond'])
        pred2[k][1] = (pred2[k][1]*pred2[k][2]['cond'])

    #calculate 2nd totals
    aic_tot2 = 0
    bic_tot2 = 0

    for k in range(0,4):
        aic_tot2 = (aic_tot2 + pred2[k][0])
        bic_tot2 = (bic_tot2 + pred2[k][1])

    for k in range(0,4):
        pred2[k][0] = (pred2[k][0]/aic_tot2)
        pred2[k][1] = (pred2[k][1]/bic_tot2)


    #calculate final value
    final2 = 0
```

```
433     for k in range(0,4):
            final2 = final2+((pred2[k][0]+pred2[k][1])*0.5*pred2[k][2]['val'])
435
437
        #METHOD 3
439     #selected average
        '''
441     #choose only values that are "close enough" to the
        #value the predicition is for
443     #--> small enough cond (=diff of likelihood under the model)
        #calculate weights by BIC and AIC "inversely" normalized by their totals
445     #(since small aic or bic ist better than high),
        #and then calculate weighted average of predicted differences
447     #finally average the BIC and AIC result values 50/50
        '''
449
        pred3 = copy.deepcopy(pred)
451
        #select values that are "close enough"
453     #--> within 1.25 times the mean of cond
        cond_max = 0
455
        for k in range(0,4):
457         cond_max = (cond_max + pred3[k][2]['cond'])
        cond_max = 0.3125*cond_max ## mean*1.25
459
        condit = len(pred3)
461     k = 0
        while k < condit:
463         if pred3[k][2]['cond'] > cond_max:
                del pred3[k]
465             condit = (condit-1)
            else:
467             k += 1

469     n = len(pred3)

471     #calculate 1st totals
        aic_tot1 = 0
473     bic_tot1 = 0
```

```
475
    for k in range(0,n):
477         aic_tot1 = (aic_tot1 + pred3[k][0])
            bic_tot1 = (bic_tot1 + pred3[k][1])
479

481     for k in range(0,n):
            pred3[k][0] = (aic_tot1/pred3[k][0])
483         pred3[k][1] = (bic_tot1/pred3[k][1])

485
    #calculate 2nd totals
487 aic_tot2 = 0
    bic_tot2 = 0
489
    for k in range(0,n):
491         aic_tot2 = (aic_tot2 + pred3[k][0])
            bic_tot2 = (bic_tot2 + pred3[k][1])
493
    for k in range(0,n):
495         pred3[k][0] = (pred3[k][0]/aic_tot2)
            pred3[k][1] = (pred3[k][1]/bic_tot2)
497
    #calculate final value
499 final3 = 0

501 for k in range(0,n):
        final3 = final3+((pred3[k][0]+pred3[k][1])*0.5*pred3[k][2]['val'])
503

505 #save all final values of the 3 methods in a list
    final = [final1,final2,final3]
507
    #and add it to a df
509 final_diff.loc[q] = final

511 print('FINISHED')

513
```

```
515
517 #calculate the predicted close price for the next day by
    #adding the predicted diff to the close price of the current day
519 final_df = pd.DataFrame(columns=('real','pred1', 'pred2', 'pred3'))
    n = len(data_predict.index) - 64
521
    for j in range(0,63):
523
        r = data_predict.iloc[(n+j+1)].Close
525     f1 = data_predict.iloc[(n+j)].Close + final_diff.diff1.iloc[j]
        f2 = data_predict.iloc[(n+j)].Close + final_diff.diff2.iloc[j]
527     f3 = data_predict.iloc[(n+j)].Close + final_diff.diff3.iloc[j]

529     add = [r,f1,f2,f3]
        final_df.loc[j] = add
531
533
535
    #export resulting df as .csv file for further work
537 #corresponding to import data file

539 final_df.to_csv('IBM_pred.csv', sep=',')
    #final_df.to_csv('Apple_pred.csv', sep=',')
541 #final_df.to_csv('Facebook_pred.csv', sep=',')
    #final_df.to_csv('Google_pred.csv', sep=',')
```

ML2_mod.py

```
    ################################################
2   #model evaluation & result viz
    ################################################
4
    # -*- coding: utf-8 -*-
6   """
    Created on Sat Jan 21 13:21:14 2017
8
```

```
@author: Jonas
"""


import pandas as pd
import numpy as np
#import seaborn as sns
import matplotlib.pyplot as plt



#load data
data = pd.read_csv('IBM_pred.csv', index_col=0)
#data = pd.read_csv('Apple_pred.csv', index_col=0)
#data = pd.read_csv('Facebook_pred.csv', index_col=0)
#data = pd.read_csv('Google_pred.csv', index_col=0)




#define function to calculate RMSE
def rmse(prediction, real):
    return np.sqrt(((prediction-real)**2).mean())


#format data for calculation
tar = np.array(data.real[:])
p1 = np.array(data.pred1[:])
p2 = np.array(data.pred2[:])
p3 = np.array(data.pred3[:])


#calculate RMSE, rounded to 4 digits
r1 = round(rmse(p1,tar), 4)
r2 = round(rmse(p2,tar), 4)
r3 = round(rmse(p3,tar), 4)


print()
print('RMSE results:')
print('Method 1: {}'.format(r1))
```

```
print('Method 2: {}'.format(r2))
print('Method 3: {}'.format(r3))



#plot time series of real values and the predicted values


#method 1
fig1, ax1 = plt.subplots(ncols=1,figsize=(15, 8))
data.loc[:,['real','pred1']].plot(colormap='bwr', ax=ax1,
xticks=pd.Series(range(0,66,5)), xlim=[-2,64], marker='o')
ax1.set_title('Method 1', fontsize=20)
ax1.set_xlabel('Time Stamps (Trading-days)', fontsize=16)
ax1.set_ylabel('Close Price', fontsize=16)
ax1.legend(fontsize=15, loc=0)
ax1.grid()


#method 2
fig2, ax2 = plt.subplots(ncols=1,figsize=(15, 8))
data.loc[:,['real','pred2']].plot(colormap='bwr', ax=ax2,
xticks=pd.Series(range(0,66,5)), xlim=[-2,64], marker='o')
ax2.set_title('Method 2', fontsize=20)
ax2.set_xlabel('Time Stamps (Trading-days)', fontsize=16)
ax2.set_ylabel('Close Price', fontsize=16)
ax2.legend(fontsize=15, loc=0)
ax2.grid()


#method 3
fig3, ax3 = plt.subplots(ncols=1,figsize=(15, 8))
data.loc[:,['real','pred3']].plot(colormap='bwr', ax=ax3,
xticks=pd.Series(range(0,66,5)), xlim=[-2,64], marker='o')
ax3.set_title('Method 3', fontsize=20)
ax3.set_xlabel('Time Stamps (Trading-days)', fontsize=16)
ax3.set_ylabel('Close Price', fontsize=16)
ax3.legend(fontsize=15, loc=0)
ax3.grid()


```

```
92
#variance of real data
94 print()
print('Variance of real data: ')
96 print(round(data.loc[:,'real'].var(), 4))
```

ML2_mod_eval.py