دانشگاه شهید بهشتی

دانشکده مهندسی و علوم کامپیوتر

گزارش پروژه پایانی درس یادگیری ماشین

پیش‌بینی بازار سهام با استفاده از مدل‌های پنهان مارکوف

استاد درس: دکتر احمدعلی آبین

نام دانشجو: طراوت پارت

شماره دانشجویی: ۴۰۰۴۴۳۰۴۰

تابستان ۱۴۰۱

مقدمه:

هدف این پروژه پیش‌بینی بازارهای مالی با استفاده از الگوریتم‌های یادگیری ماشین است. با استفاده از مدل‌های پنهان مارکوف می‌توان اتفاقاتی که با گذر زمان تغییر می‌کنند را مدل کرد. در این پروژه بهترین مدل پنهان مارکوف پیدا می‌شود و پارامترها طوری تنظیم می‌شوند که منجر به بهترین نتیجه شود. سپس خروجی حاصل از مدل پنهان مارکوف با خروجی حاصل از یک شبکه عصبی عمیق متشکل از لایه‌های LSTM مقایسه می‌شود. در نهایت مشاهده می‌شود که در این مسئله HMM خروجی بهتری نسبت به LSTM تولید می‌کند.

تشریح الگوریتم HMM:

همانطور که در تمرین سری سوم بررسی شد، با استفاده از زنجیره‌های مارکوف داده‌هایی که با گذر زمان تغییر می‌کنند مدل می‌شوند. مدل پنهان مارکوف دقیقا مانند فرآیند مارکوف است، با این تفاوت که فضای حالت در آن نامعلوم است. اما هر حالت با یک خروجی همراه است. اجزای سازنده‌ی یک مدل پنهان مارکوف به شرح زیر است:

- **N:** تعداد حالت‌های پنهان.
- **M:** تعداد مشاهدات.
- **ماتریس احتمال گذر:** یک ماتریس N×N است که هر المان آن احتمال انتقال از یک حالت به حالت دیگر را نشان می‌دهد.
- **ماتریس احتمال مشاهده:** یک ماتریس N×M است که احتمال وقوع هر مشاهده در هر حالت را نشان می‌دهد.
- **بردار احتمال حالت اولیه:** یک بردار با N المان است که احتمال بودن در هر حالت در زمان اولیه را نشان می‌دهد.

```
class HMM:
    def __init__(self, num_states, observation_dim):

        self.num_states = num_states
        self.observation_dim = observation_dim
        self.A = np.random.uniform(0, 1, size=(num_states, num_states))
        self.B = np.random.uniform(0, 1, size=(num_states, observation_dim))
        self.pi = np.full((1, num_states), 1 / num_states)
```

مدل پنهان مارکوف در حل سه دسته کلی از مسائل کاربرد دارد: ارزیابی، رمزگشایی و آموزش. در ادامه هر کدام از مسائل شرح داده می‌شود.

ارزیابی(evaluation): احتمال وقوع یک دنباله از مشاهدات توسط ماشین چقدر است؟ برای حل این مسئله می‌توان از روش forward یا backward استفاده کرد. هر دو روش ما را به جواب یکسان می‌رسانند.

مراحل روش forward: در این روش جدول آلفا ساخته می‌شود و $\alpha_t(i)$ ها محاسبه می‌شوند. $\alpha_t(i)$ نشان دهنده احتمال این است که o_1 تا o_t را ببینیم و در لحظه‌ی t در حالت S_i باشیم. مراحل پر کردن جدول آلفا به شرح زیر است.

a. Initialization: در این مرحله ستون اول جدول آلفا پر می‌شود. احتمال اینکه در لحظه‌ی 1 o_1 را ببینیم و در حالت S_i باشیم محاسبه می‌شود:

$$\alpha_1(i) = \pi_i b_i(o_1)$$

b. Induction: در این مرحله بقیه ستون‌های جدول مطابق رابطه‌ی زیر پر می‌شوند.

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_i(o_{t+1})$$

این رابطه نشان می‌دهد احتمال اینکه مشاهدات را تا لحظه t دیده باشیم و در حالت S_i باشیم، از حالت S_i به حالت S_j برویم و در حالت S_j مشاهده‌ی O_{t+1} را ببینیم چند است.

c. Termination: جواب نهایی توسط رابطه زیر محاسبه می‌شود.

$$p(o|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

```
def forward(self, observation):
    num_rows = self.num_states  # number of states
    num_cols = self.observation_dim  # number of observations
    alpha_table = np.zeros((num_rows, num_cols))  # instantiating the alpha table
    alpha_table[:, 0] = self.pi * self.B[:, 0]  # filling the first column

    for col in range(1, num_cols):
        for row in range(num_rows):
            alpha_table[row, col] = np.dot(alpha_table[:, col - 1], self.A[:, row]) * self.B[row, observation[col]]

    return alpha_table
```

مراحل روش backward: در این روش جدول β ساخته می‌شود و $\beta_t(i)$ ها محاسبه می‌شوند. $\beta_t(i)$ نشان دهنده احتمال این است که در لحظه‌ی t در حالت S_i باشیم و سپس $o_{t+1} \dots o_T$ را ببینیم.

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

```
def backward(self, observation):
    num_rows = self.num_states  # number of states
    num_cols = self.observation_dim  # number of observations
    beta_table = np.zeros((num_rows, num_cols))  # instantiating the beta table
    beta_table[:, -1] = 1  # filling the last column

    for col in reversed(range(num_cols - 1)):
        for row in range(num_rows):
            beta_table[row, col] = np.sum(beta_table[:, col + 1] * self.A[row, :] * self.B[:, observation[col+1]])

    return beta_table
```

رمزگشایی (decoding): در این مسئله محاسبه می‌شود که چه دنباله‌ای از انتقالات، یک دنباله از مشاهدات را تولید کرده است. برای حل این مسئله از الگوریتم Viterbi استفاده می‌شود. الگوریتم Viterbi دقیقا مشابه forward است با این تفاوت که برای محاسبه مقدار هر خانه جدول به جای انجام عمل جمع، ماکزیمم مقادیر را بدست می‌آورد.

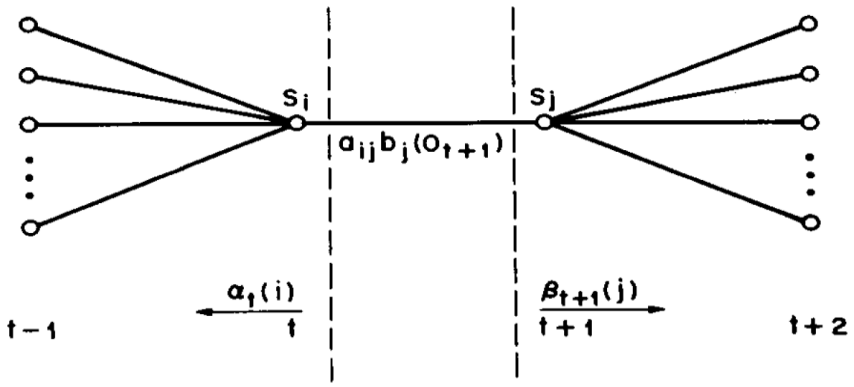
```python
def decode(self, observation):
    num_rows = self.num_states  # number of states
    num_cols = self.observation_dim  # number of observations
    viterbi_table = np.zeros((num_rows, num_cols))
    viterbi_table[:, 0] = self.pi * self.B[:, 0]  # filling the first column

    indexes = np.zeros((num_cols - 1, num_rows), dtype=int)

    for col in range(1, num_cols):
        for row in range(num_rows):
            probs = viterbi_table[:, col - 1] * self.A[:, row] * self.B[row, observation[col]]
            viterbi_table[row, col] = np.max(probs)  # storing the maximum value
            indexes[col - 1, row] = np.argmax(probs)  # storing the maximum index
    return viterbi_table, indexes
```

آموزش (learning): چطور می‌توان از یک دنباله از مشاهدات، یک مدل پنهان مارکوف ساخت؟ در واقع چطور می‌توان پارامترهای مدل را تنظیم کرد تا یک مجموعه‌ای از مشاهدات با احتمال زیاد توسط یک مدل پنهان مارکوف رخ بدهند؟ در این مسئله مقادیر ماتریس احتمال گذر و ماتریس احتمال مشاهدات ساخته می‌شود.

برای حل این مسئله ابتدا اعداد تصادفی روش ماشین قرار می‌دهیم. فرض می‌کنیم که تمام حالت‌ها به هم وصل هستند و در هر حالت تمام مشاهدات را می‌توانیم داشته باشیم. سپس احتمالات با استفاده از الگوریتم baum-welch تخمین زده می‌شوند. در این روش ابتدا مقادیر $\varepsilon_t(i,j)$ محاسبه می‌شوند. این عبارت نشان می‌دهد احتمال اینکه در زمان t در حالت S_i باشیم و در زمان t+1 در حالت S_j باشیم چند است.

$$\varepsilon_t(i,j) = P(q_t = S_i, q_{t+1} = s_j | O, \lambda)$$

تصویر زیر ترتیبی از اتفاقاتی که برای برقرار شدن عبارت بالا نیاز است را نشان می‌دهد:



بنابراین مقادیر $\varepsilon_t(i,j)$ را می‌توان توسط رابطه زیر محاسبه کرد:

$$\varepsilon_t(i,j) = \frac{\alpha_t(i) \times a_{ij} \times b_j(o_{t+1}) \times \beta_{t+1}(j)}{p(o|\lambda)}$$

با فرض اینکه $\gamma_t(i)$ احتمال قرار داشتن در حالت S_i در زمان t باشد، می‌توان ارتباط زیر را بین $\gamma_t(i)$ و $\varepsilon_t(i,j)$ برقرار کرد:

$$\gamma_t(i) = \sum_{j=1}^{N} \varepsilon_t(i,j)$$

عبارت زیر نشان دهنده میانگین تعداد دفعاتی است که حالت S_i ویزیت شده است یا به طور معادل تعداد گذرهایی است که از حالت S_i اتفاق افتاده است.

$$\sum_{t=1}^{T-1} \gamma_t(i) = expected\ number\ of\ transitions\ from\ S_i$$

به طور مشابه رابطه زیر نشان دهنده تعداد میانگین دفعاتی است که از S_i وارد S_j می‌شویم.

$$\sum_{t=1}^{T-1} \varepsilon_t(i,j) = expected\ number\ of\ transitions\ from\ S_i\ to\ S_j$$

در نهایت با استفاده از فرمول‌های زیر می‌توان پارامترهای مدل را تخمین زد.

$$\overline{a}_{ij} = \frac{expected\ number\ of\ transitions\ from\ S_i}{expected\ number\ of\ transitions\ from\ S_i\ to\ S_j} = \frac{\sum_{t=1}^{T-1} \varepsilon_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i,j)}$$

$$\overline{b}_j(k) = \frac{expected\ number\ of\ transitions\ from\ S_i\ and\ observing\ symbol\ v_k}{expected\ number\ of\ transitions\ from\ S_i}$$

اگر روابطه بالا را به دفعات تکرار کنیم، در نهایت به مدلی میرسیم که پارامترهای آن با احتمال زیادی مشاهدات مورد نظر ما را تولید کرده‌اند.

پیاده سازی الگوریتم یادگیری baum-weltch در کلاس HMM قابل مشاهده است.

توضیح HmmGMM:

در این پروژه برای پیاده‌سازی مدل پنهان مارکوف از کلاس HMMGMM موجود در کتابخانه hmmlearn استفاده شده است. دلیل استفاده از این مدل پیوسته بودن مقادیر مشاهدات(فیچرها) است. در HmmGMM احتمال وقوع مشاهدات با استفاده از تابع Gaussian mixture مدل می‌شوند. در واقع احتمال مشاهده‌ی O_t در S_j از رابطه‌ی زیر محاسبه می‌شود:

$$b_j(o_t) = \sum_{m=1}^{M} c_{jm} N(o_t, \mu_{jm}, \Sigma_{jm})$$

- m تعداد component ها در GMM است.
- c_{jm} وزن component شماره m در S_j است.
- μ_{jm} میانگین component شماره m در S_j است.
- Σ_{jm} ماتریس کوواریانس component شماره m در S_j است.
- $N(o_t, \mu_{jm}, \Sigma_{jm})$ احتمال مشاهده‌ی o_t توسط GMM شماره m در S_j است.

آماده سازی دیتاست:

بدست آوردن فیچرهای مناسب، از مهم‌ترین قدم‌های انجام پروژه یادگیری ماشین است. در این پروژه همانند آنچه در مقاله Stock Market Prediction Using Hidden Markov Models ذکر شده است، فیچرهای زیر ساخته شد:

$$o_t = (\frac{close - open}{open}, \frac{high - open}{open}, \frac{open - low}{open})$$

در این رابطه ***open*** ارزش باز شدن سهام در روز ***t***، ***close*** ارزش بسته شدن سهام در روز ***t***، ***high*** بیشترین ارزش و ***low*** کمترین ارزش روز ***t*** است. از این پس مشاهدات ما سه بعدی خواهد بود. با استفاده از فیچرهای سه بعدی مدل مخفی مارکوف آموزش می‌بینند.

بازه مقادیر موجود در مجموعه داده به شرح زیر است:

observation	*Range*	
	min	*max*
$\frac{close - open}{open}$	*-0.1*	*0.19*
$\frac{high - open}{open}$	*0*	*0.21*
$\frac{open - low}{open}$	*0*	*0.11*

در نهایت ۲۰ درصد از داده‌ها برای تست و ۸۰ درصد برای آموزش در نظر گرفته شده‌اند.

مراحل پیاده‌سازی HMM:

نحوه آموزش مدل:

در این مرحله ابتدا یک مدل از جنس hmm.GMMHMM ساخته میشود. هنگام ساخت مدل تعداد حالت‌ها برابر ۵ و تعداد component های تابع GMM برابر ۴ در نظر گرفته شده است. با استفاده از تابع fit الگوریتم baum-welch اجرا می‌شود و مدل آموزش می‌بینند.

```
model = hmm.GMMHMM(n_components = 6, n_mix=3)
model = model.fit(train_data_processed)
```

نحوه تولید خروجی مدل:

مرحله نهایی انجام ارزیابی روی داده‌های تست است. بدین منظور سعی شد از الگوریتم ***evaluation*** در ***HMM*** استفاده شده است. مراحل انجام کار به شرح زیر است:

۱. **پیدا کردن تمام خروجی های ممکن مدل:** همانطور که می‌دانیم هدف ما پیش‌بینی یک سطر داده است. هر سطر شامل سه عدد پیوسته است. برای انجام این کار ابتدا بازه مقادیری که فیچر اول در آن قرار می‌گیرند را پیدا می‌کنیم. با استفاده از توابع *np.max* و *np.min* این کار انجام می‌شود. سپس با استفاده از تابع *np.linspace* ۵۰ عدد در این بازه تولید می‌شود. همین کارها را برای فیچر دوم و سوم نیز انجام می‌دهیم. برای فیچر دوم ۱۰ عدد و برای فیچر سوم نیز ۱۰ عدد تولید شده است. پس تا اینجا مقادیر قابل پیش‌بینی کوانتیزه شده‌اند. با این وجود ما می‌توانیم ۵۰×۱۰×۱۰ خروجی ممکن داشته باشیم. (تعداد ترکیب‌های ممکن مقادیر تولید شده).

۲. **پیدا کردن بهترین خروجی:** در مرحله تست هر بار ۲۰ سطر متوالی را ورودی می‌گیریم(*20 observations*). سپس هر بار یکی از *possible outcome* ها را به ۲۰ سطر داده خود متصل می‌کنیم. در این حالت ۲۱ سطر خواهیم داشت. سپس احتمال تولید شدن هر کدام از این ۲۱ مشاهده توسط مدل محاسبه می‌شود. در واقع عمل *evaluation* انجام می‌شود. در نهایت هر کدام از *possible outcome* ها که منجر به بیشترین *score* شوند، به عنوان پیش‌بینی مدل در نظر گرفته می‌شود.

۳. **محاسبه خروجی نهایی:** تا به اینجای کار مدل مقادیر $(\frac{close-open}{open}, \frac{high-open}{open}, \frac{open-low}{open})$ را خروجی داده است. به سادگی با دانستن مقدار *open*، مقادیر *high* و *low* و *close* روز بعدی بدست می‌آیند.

Close_price_prediction = open_price × (1 + prediction[0])

High_price_prediction = open_price × (1 + prediction[1])

Low_price_prediction = open_price × (1 + prediction[2])

نحوه ارزیابی مدل:

برای ارزیابی مدل از تابع *MSE* استفاده شده است. یک بار مقدار *MSE* بین مقادیر $(\frac{close-open}{open}, \frac{high-open}{open}, \frac{open-low}{open})$ که توسط مدل پیش‌بینی شده است و مقادیر واقعی$(\frac{close-open}{open}, \frac{high-open}{open}, \frac{open-low}{open})$ بدست می‌آید. باردیگر از روی خروجی مدل مقادیر *(close_price,high_price,low_price)* ساخته می‌شود و *MSE* بین این مقادیر ساخته شده و مقادیر واقعی آن‌ها محاسبه می‌شود.

پیدا کردن بهترین پارامتر های hmmGMM:

در ادامه مدل را با پارامترهای مختلف تنظیم میکنیم و عملکرد مدل را روی داده‌های تست میسنجیم.

در جدول زیر خطای مدل به ازای تعداد *component* های متفاوت در *GMM* نشان داده شده است:

Number of components in GMM	*MSE Loss*
3	0.000706
5	0.000739
7	0.000739

همانطور که مشاهده می‌شود، تعداد *component=3* منجر به خطای کمتری شده است. همچنین زیاد کردن تعداد *component* ها از ۳ به ۵ و از ۵ به ۷ زمان تست را خیلی زیاد می‌کند(در حد ۳ دقیقه). بنابراین در ادامه با تعداد *component* های ۳ پیش میرویم تا هم به خطای کمتری برسیم و هم مدل سریع‌تری داشته باشیم.

نکته: دلیل کم شدن خطا با افزایش تعداد *component* ها میتواند به دلیل *overfitting* باشد. احتمالا با افزایش تعداد *component* ها مدل روی داده‌های آموزشی بهتر عمل می‌کند اما روی داده‌های تست به عملکرد نامناسب می‌رسد.

در جدول زیر خطای مدل به ازای تعداد حالت‌های پنهان متفاوت نشان داده شده است:

Number of states	*MSE Loss*
2	0.000446
4	0.000443
6	0.000441

همانطور که مشاهده می‌شود، با زیاد شدن تعداد حالت‌های مخفی خطای مدل کمتر می‌شود. اما زمان اجرا نیز بیشتر می‌شود. در ادامه از تعداد حالت‌های پنهان ۶ استفاده می‌شود تا به دقت و سرعت مناسب برسیم.

نکته: در صورت زیاد کردن تعداد استیت‌ها پیچیدگی مدل بیشتر می‌شود. در نتیجه ممکن است مدل روی دیتای آموزشی *overfit* کند. برای مثال فرض کنید تعداد استیت ها را انقدر زیاد کنیم که برابر با تعداد *observation* ها بشود(به هر سطر داده یک *observation* گفته می‌شود). در این حالت هر استیت نماینده یک *observation* می‌شود. مثلا استیت شماره*n* میگوید *observation* شماره *n* با احتمال یک می‌دهد و بقیه *observation* ها با احتمال صفر رخ می‌دهند. در این حالت مدل روی داده‌های تست عملکرد خیلی ضعیفی خواهد داشت.

در جدول زیر خطای مدل به ازای *time step* های متفاوت نشان داده شده است:

Timestep	*MSE Loss*
10	0.00067
20	0.00066
40	0.00044

مطابق آنچه مشاهده می‌شود، افزایش *time step* عمدتا باعث افزایش عملکرد مدل می‌شود. بنابراین از این پس از *timestep=40* استفاده خواهد شد.

نکته: اصلا در این مسئله خاص *time step* ربطی به عملکرد *HMM* نداره چون از *time step* در مرحله تست استفاده شده نه آموزش مدل! نمیدونم چرا *time step*های مختلف رو امتحان کردم حالا همینطوری تو گزارش گذاشتم☺

خروجی نهایی مدل hmmGMM:

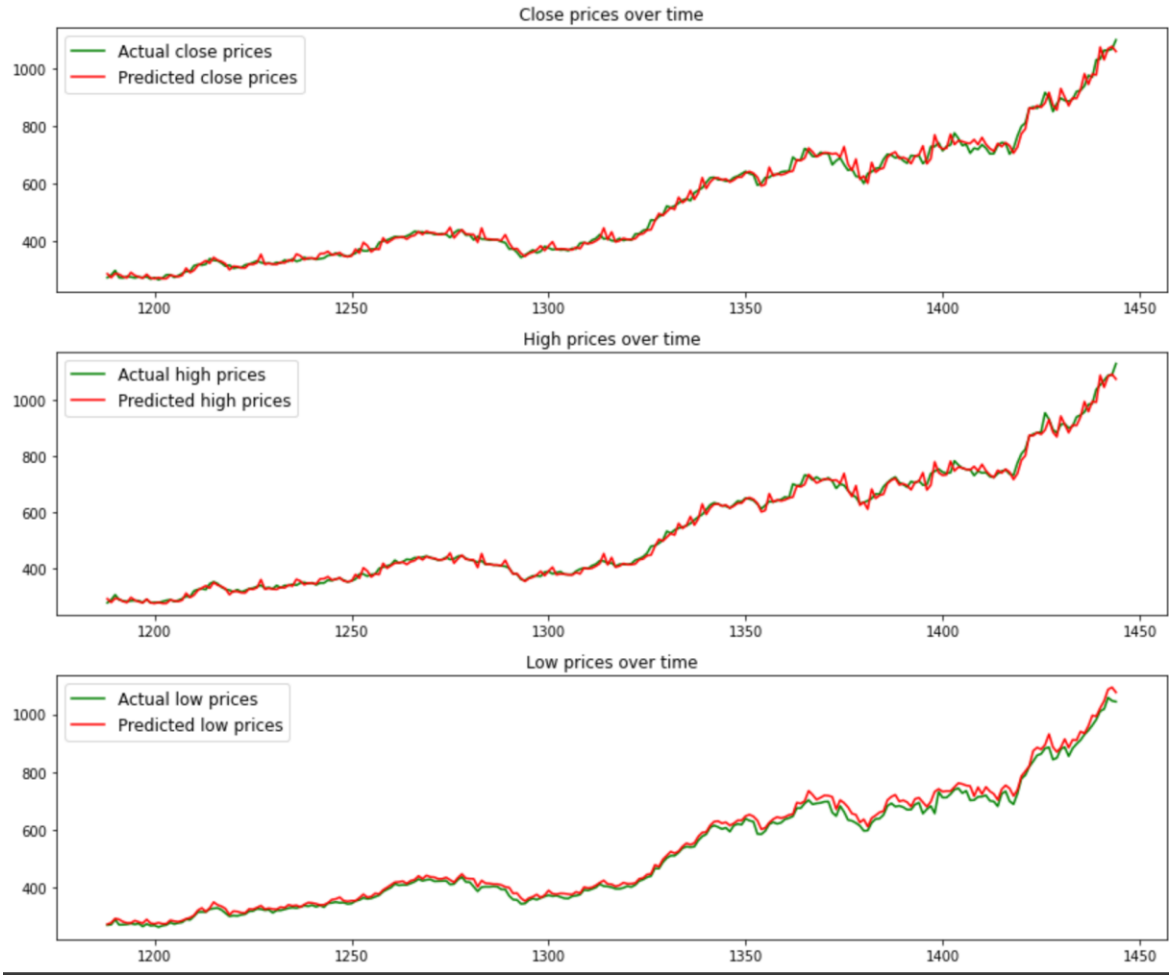
در نهایت پارامترهای مدل پنهان مارکوف مطابق زیر تنظیم شد:

- تعداد حالت‌های پنهان : ۶
- تعداد component ها در GMM: ۳

0.00044	زیان MSE مدل – استفاده از خروجی های مستقیم مدل

239.2	زیان MSE مدل- تبدیل خروجی‌های مدل به مقادیر high,low,close

نکته: دلیل متفاوت بودن مقدار زیان، متفاوت بودن scale پیش‌بینی‌ها در دو حالت ذکر شده است. صرفا از مقدار MSE برای مقایسه نتایج استفاده خواد شد. MSE کمتر به معنای عملکرد بهتر است.



مراحل پیاده‌سازی LSTM:

آماده سازی دیتاست: نحوه استخراج ویژگی‌ها در این قسمت دقیقا مشابه به مسئله قبلی است. با این تفاوت که در اینجا نیاز داریم داده‌های ترتیبی بسازیم. در اثر ساخت داده‌های ترتیبی، با فرض در نظر گرفتن time-step=40، ابعاد داده‌های train از 3×1188 به 3×40×1148 تبدیل می‌شود. یعنی 1148 داده برای ورود به شبکه داریم، که هر کدام شامل ۴۰ سطر هستند(انداز time step) و هرکدام 3 فیچر دارند. همچنین ترتیب‌های متوالی با هم همپوشانی دارند.

پیاده سازی مدل: یک شبکه عصبی بازگشتی با سه لایه LSTM و یک لایه Dense ساخته شده است. این لایه‌ها به ترتیب 50 ، 50 ، 100 و 3 نورون دارند. مدل با بهینه‌سازی Adam آموزش می‌بیند.

```
model=Sequential()
model.add(LSTM(50,return_sequences=True,input_shape=(time_series_trainX.shape[1],time_series_trainX.shape[2])))
model.add(LSTM(50,return_sequences=True))
model.add(LSTM(100))
model.add(Dense(3))
model.compile(loss='mean_squared_error',optimizer='adam')
```

پیدا کردن بهترین پارامتر های lstm:

در ادامه مدل را با پارامترهای مختلف تنظیم میکنیم و عملکرد مدل را روی داده‌های تست میسنجیم.

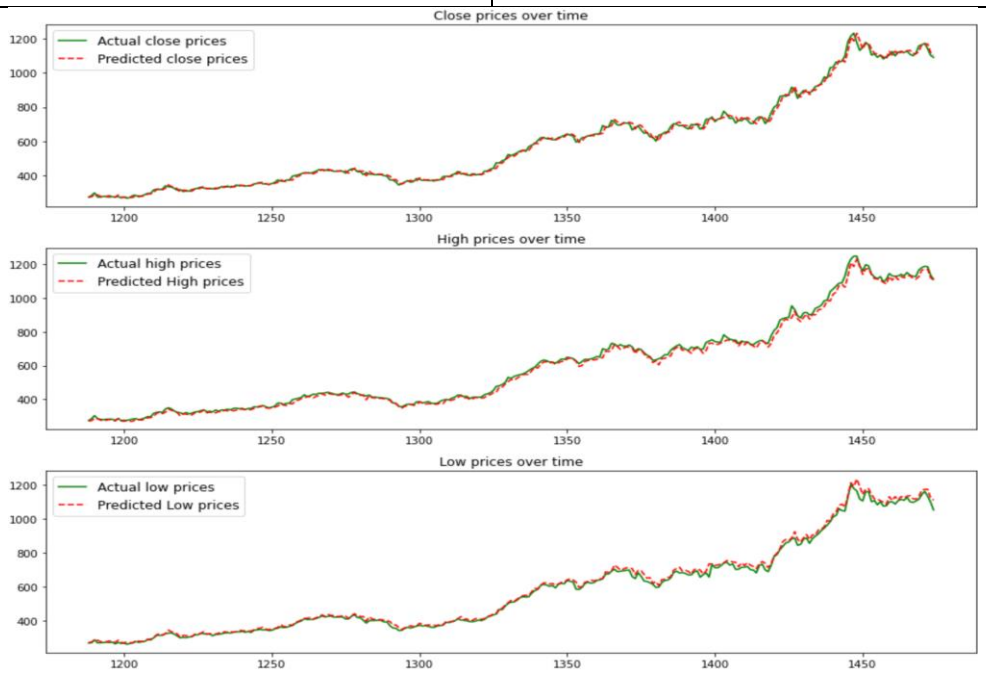
Timestep	*MSE Loss*
10	00059
20	0.00059
30	0.00065
40	0.0006

عوض کردن tme step تاثیر زیادی روی عملکرد مدل نمیگذارد!

نتایج LSTM:

همانند مسئله قبلی، از تابع زیان MSE برای ارزیابی مدل استفاده شده است.

زیان MSE مدل – استفاده از خروجی های مستقیم مدل	0.00059
زیان MSE مدل- تبدیل خروجی‌های مدل به مقادیر high,low,close	273.70



تحلیل نتایج:

مدل‌های پنهان مارکوف بسیار ساده‌تر از شبکه‌های عصبی بازگشتی هستند. مدل‌های مارکوف فرض می‌کنند که مقدار فعلی یک متغیر تصادفی فقط به مقدار قبلی آن وابسته است. این فرض بسته به نوع مسئله ممکن است همواره صحت نداشته باشد.

در صورت وجود مجموعه‌داده‌های حجیم، شبکه‌های عصبی بازگشتی بهتر کار می‌کنند. شبکه‌های عصبی بازگشتی ساختار پیچیده و تعداد پارامترهای زیاد دارند و می‌توانند به خوبی روی مجموعه داده‌های حجیم آموزش ببینند.

در ابتدا انتظار میرفت به دلیل اینکه در شبکه‌های عصبی LSTM حافظه وجود دارد اما HMM ها فقط به یک حالت قبلی برای رسیدن به حالت فعلی توجه می‌کنند، LSTM ها به خروجی بهتری برسند. اما در این مسئله خاص پس از آموزش هر دو مدل، به دقت‌های تقریبا مشابه رسیدیم. مسئله مدل پنهان مارکوف کمی بهتر عمل کرده‌است. بهترین خطای بدست آمده از HMM برابر 0.00044 و بهترین خطای بدست آمده از RNN برابر 0.0006 است. توجیه آن احتمالا کم برقرار بودن شرط مارکوف در این مسئله و کفایت کردن مدل HMM به دلیل کم بودن حجم دیتاست است.

نتیجه‌گیری:

در این پروژه ابتدا سعی شد که مدل‌های پنهان مارکوف بدون استفاده از کتابخانه‌های آماده پیاده‌سازی شود. الگوریتم‌های اصلی آن از جمله forward evaluation، backward evaluation، Viterbi و baum-weltch پیاده‌سازی شدند. در نهایت با توجه به مجموعه داده ما که شامل observation هایی با مقادیر پیوسته است، تصمیم گرفته شد از مدل‌های پنهان مارکوف گسسته یا به طور خاص از HmmGMM ها استفاده شود. در این نوع مدل پنهان مارکوف، احتمال وقع مشاهدات در هر استیت با یک GMM مدل می‌شود. در نهایت سعی شد این مدل با بهترین پارامترهای ممکن آموزش ببینند. در نهایت مشاهده شد که در این مسئله خاص هر دو مدل HmmGMM و LSTM به خوبی کار می‌کنند. شاید مدل GMMhMM در اینجا اندکی بهتر کار کند. دلیل آن ماهیت مجموعه داده است. احتمالا فرض مارکوف در این مجموعه داده صدق می‌کند. همچنین دیتاست ما بسیار کم حجم است که لزومی به استفاده از شبکه عصبی عمیق وجود ندارد.

به طور کل مدل های Hmm مولد هستند. آنها ورودی و خروجی مشخصی ندارند. از آن‌ها برای تولید یک توالی از observation ها میتوان استفاده کرد. یا می‌توان متوجه شده یک توالی از observation ها توسط چه ترتیبی از استیت‌ها تولید شده‌اند. اما شبکه‌های LSTM ورودی و خروجی دارند و discriminative هستند.