به نام خدا

پروژه پنجم درس یادگیری عمیق

آشنایی با کاربرد شبکه های عصبی بازگشتی در تحلیل سری های زمانی

استاد درس: دکتر صفابخش

نگارش: زهرا اخلاقی

شماره دانشجویی: ۴۰۱۱۳۱۰۶۴

زمستان ۱۴۰۲

فهرست مطالب

بخش‌اول) دسته‌بندی سری‌های زمانی

از کد زیر برای استخراج سابقه روزانه شاخص کل بورس، ،شاخص هم وزن و نماد فولاد استفاده شده است، دیتاست نهایی شامل ستون close در هر جدول است. در دیتاست فولاد فیلد close با داده دیروز مقایسه می‌شود و در صورتی که بیشتر بود مقدار ۱ و در غیر اینصورت مقدار -۱ ب عنوان label در نظر گرفته می‌شود.

```
tickers = tse.download(symbols="فولاد", write_to_csv=True)
df = pd.read_csv('./tickers_data/فولاد.csv')
df['Label']=np.where(df['close']>df['yesterday'],1,-1)
total_index_data=tse.FinancialIndex(symbol="شاخص‌کل").history[['date','clo
se']]
homogenous_index_data=tse.FinancialIndex(symbol="هوزن)شاخص‌کل(").history[[
'date','close']]
```

دیتاست نهایی حاصل جوین جداول فوق می‌باشد:

	date	close_total_index_data	close_homogenous_index_data	close_steel_symbol_data	Label
0	2015-02-23	64678	9939	1910.0	-1
1	2015-02-24	64526	9893	1896.0	-1
2	2015-02-25	64052	9815	1880.0	-1
3	2015-02-28	63951	9794	1844.0	-1
4	2015-03-01	63860	9811	1867.0	1
...	...	...	...	...	...
1998	2023-12-24	2200110	761234	6280.0	-1
1999	2023-12-25	2194500	760623	6240.0	-1
2000	2023-12-26	2178752	755254	6200.0	-1
2001	2023-12-27	2177410	755296	6220.0	1
2002	2023-12-30	2182718	759645	6280.0	1

2003 rows × 5 columns

الف)

شبکه‌های عصبی بازگشتی (RNN) از اطلاعات داده‌های گذشته برای پیش‌بینی داده‌های آینده استفاده می‌کند ولی برای پردازش داده‌های گذشته طولانی طراحی نشده‌است. عمل Data windowing به حل این مشکل کمک می‌کند و برای آموزش هر داده فقط داده های درون پنجره پردازش میشود.

Data windowing یک مرحله پیش پردازش ضروری برای آموزش شبکه های عصبی بازگشتی (RNN) بر روی داده های سری زمانی است. اندازه پنجره باید به اندازه کافی بزرگ باشد تا الگوهای مربوطه را در داده ها را ثبت کند، اما به اندازه کافی کوچک باشد تا از برازش بیش از حد جلوگیری شود.

دلایل کلیدی برای استفاده از پنجره داده:

- **گرفتن وابستگی های زمانی:** پنجره سازی داده سری زمانی را به پنجره های کوچکتر تقسیم می کند و به RNN اجازه می دهد تا اطلاعات گذشته را به اندازه پنجره تحلیل کند.
- **مدیریت مراحل زمانی متغیر:** داده های سری زمانی اغلب فواصل زمانی متغیری را بین نقاط داده نشان می دهند. پنجره‌دهی داده می‌تواند این مشکل را با ترکیب پنجره‌های با طول متغیر برطرف کند، و به RNN اجازه می‌دهد تا با مقیاس‌های زمانی مختلف سازگار شود.
- **کاهش پیچیدگی محاسباتی:** پردازش کل داده های سری زمانی به طور همزمان می تواند از نظر محاسباتی گران باشد، به خصوص برای سری های زمانی طولانی. پنجره دهی داده ها را به تکه های کوچکتر تقسیم می کند و پردازش آن را برای RNN ها قابل مدیریت تر می کند.
- **ساده سازی آموزش مدل:** پنجره سازی داده ها با ارائه یک قالب استاندارد برای داده های ورودی، فرآیند آموزش را ساده می کند.
- **افزایش قابلیت تفسیر:** پنجره‌سازی داده‌ها می‌تواند تفسیرپذیری مدل‌های RNN را با آشکار کردن الگوها و روابط آموخته‌شده از بخش‌های داده بهبود بخشد. این می تواند به درک فرآیند تصمیم گیری مدل کمک کند.

تابع زیر عمل Data windowing را انجام می‌دهد، برای پیاده سازی مدل مقدار ws=10 در نظر گرفته شده است، برای پیش بینی هر داده، ۱۰ داده گذشته در نظر گرفته می‌شود.

```python
import numpy as np


def data_windowing(seq,ws):
    x = []
    y = []
    L = len(seq)

    for i in range(L-ws):

        x.append(data.iloc[i:i+ws, :-1])  # Exclude the label column
        y.append(data.iloc[i+ws, -1])

    return np.array(x) , np.array(y)
```

ب)

۸۰ درصد داده‌ها برای آموزش و ۲۰ درصد برای تست استفاده شده‌اند. داده ستون label که مقادیر قبلی آن -۱ و ۱ بود در مرحله پیش پردازش به ۰و۱ تغییر کرده است.

RNN Model:

کد زیر یک مدل شبکه عصبی بازگشتی (RNN) برای پیش بینی سری زمانی تعریف می کند. مدل از دو لایه LSTM تشکیل شده است، هر کدام با اندازه مخفی 128 واحد. لایه های LSTM با یک لایه dropout برای کاهش overfitting دنبال می شوند و یک لایه کاملاً متصل (FC) برای نقشه برداری از حالت مخفی به اندازه خروجی دلخواه.

```
class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers=2, dropout=0.2):
        super(RNNModel, self).__init__()
        self.lstm1 = nn.LSTM(input_size, hidden_size, num_layers=num_layers, batch_first=True, dropout=dropout)
        self.lstm2 = nn.LSTM(hidden_size, hidden_size, num_layers=num_layers, batch_first=True, dropout=dropout)
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.lstm1(x)
        out = self.dropout(out)
        out, _ = self.lstm2(out)
        out = self.dropout(out)
        out = self.fc(out[:, -1, :])
        return out
```

نتیجه پس از ۱۰۰ ایپاک اجرا به صورت زیر میباشد:

```
Epoch [10/100], Loss: 0.7063
Epoch [20/100], Loss: 0.6677
Epoch [30/100], Loss: 0.6858
Epoch [40/100], Loss: 0.6906
Epoch [50/100], Loss: 0.6612
Epoch [60/100], Loss: 0.6915
Epoch [70/100], Loss: 0.6726
Epoch [80/100], Loss: 0.6924
Epoch [90/100], Loss: 0.7070
Epoch [100/100], Loss: 0.8601
```

داده به دو دسته train , test تقسیم شده اند و پس از پیش‌بینی داده تست روی آن تابع sigmoid اعمال میشود و اگر خروجی بزرگتر از ۰.۵ بود برچسب ۱ و در غیر اینصورت برچسب ۰ میگیرد.

Loss: 0.6680, Accuracy: 0.6164

مدل آموزش دیده با اینکه بررسی شده لیبل توزیع مناسبی دارد تنها مقدار ۰ را برای داده‌ها پیش بینی میکند.

CNN Model:

کد زیر برای یک مدل شبکه عصبی کانولوشنی (CNN) برای طبقه بندی تصاویر ارائه شده است. این مدل از یک لایه کانولوشن، یک لایه فعال‌سازی ReLU، یک لایه pooling، یک لایه کانولوشن دیگر، یک لایه فعال‌سازی ReLU، یک لایه pooling، یک لایه fully connected و یک لایه خروجی تشکیل شده است.

```python
class CNNModel(nn.Module):
    def __init__(self, input_size, num_channels, output_size):
        super(CNNModel, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=input_size, out_channels=num_channels, kernel_size=3)
        self.relu = nn.ReLU()
        self.fc = nn.Linear(num_channels, output_size)

    def forward(self, x):
        x = self.conv1(x)
        x = self.relu(x)
        x = x.mean(dim=2)
        x = self.fc(x)
        return x
```

نتیجه پس از ۱۰۰ ایپاک اجرا به صورت زیر میباشد:

```
Epoch [10/100], Loss: 0.6897
Epoch [20/100], Loss: 0.7001
Epoch [30/100], Loss: 0.7018
Epoch [40/100], Loss: 0.6954
Epoch [50/100], Loss: 0.7263
Epoch [60/100], Loss: 0.6481
Epoch [70/100], Loss: 0.7134
Epoch [80/100], Loss: 0.6569
Epoch [90/100], Loss: 0.7303
Epoch [100/100], Loss: 0.6557
```

نتیجه روی داده تست به صورت زیر است:

Loss: 0.6701, Accuracy: 0.6164

مدل آموزش دیده با اینکه بررسی شده لیبل توزیع مناسبی دارد تنها مقدار ۰ را برای داده‌ها پیش بینی میکند.

تحلیل:

با توجه به اینکه در هر دو مدل داده تست و آموزش برابر است، نتیجه دقت روی داده تست برای هر دو مدل یکسان میباشد (مدل به خوبی آموزش نمی‌بیند و تنها مقدار -۱ را پیش‌بینی می‌کند.) و در حین آموزش مقدار loss در حین فرآیند آموزش گاهی اوقات کوچک و گاهی اوقات بزرگ می‌شوند و با زیاد کردن تعداد ایپاک این مشکل برطرف نشد.

ج)

اضافه کردن اندیکاتور به داده‌ها قبل از آموزش شبکه می‌تواند به بهبود عملکرد شبکه کمک کند. اندیکاتورها متغیرهایی هستند که از داده‌های گذشته برای پیش‌بینی رفتار آینده استفاده می‌کنند. وقتی اندیکاتورها به داده‌ها اضافه می‌شوند، شبکه عصبی می‌تواند از اطلاعات بیشتری برای یادگیری استفاده کند و می‌تواند به شبکه کمک کند تا الگوهای پیچیده‌تری را در داده‌ها شناسایی کند و پیش‌بینی‌های دقیق‌تری انجام دهد.

در جداول زیر عملکرد مدل‌ها با افزودن اندیکیتور sma (میانگین متحرک ساده) و ema (میانگین متحرک نمایی، یک اندیکاتور تکنیکال است که برای شناسایی روند قیمت) ,rsi به ویژگی های جدول نشان می‌دهد.

accuracy:

	شاخص کل	شاخص هم‌وزن	هر دو
RNN	0.7	0.67	0.63
CNN	0.62	0.63	0.62

Loss:

	شاخص کل	شاخص هم‌وزن	هر دو
RNN	0.53	0.61	0.56
CNN	0.63	0.64	0.63

با توجه به جدول بالا شبکه RNN در کل عملکرد بهتری نسبت به شبکه CNN دارد (شبکه‌های RNN برای تشخیص داده‌های بورس بهتر هستند زیرا می‌توانند وابستگی‌های زمانی طولانی‌مدت را در داده‌ها شناسایی کنند. این امر به ویژه برای داده‌های بورس مهم است زیرا قیمت سهام اغلب تحت تاثیر عوامل مختلفی قرار می‌گیرد. شبکه‌های CNN برای شناسایی الگوهای فضایی در داده‌ها طراحی شده‌اند. داده‌های بورس اغلب دارای الگوهای فضایی واضحی نیستند، بنابراین شبکه‌های CNN نمی‌توانند عملکرد خوبی در تشخیص آنها داشته باشند)

افزودن اندیکاتور به شاخص کل برای پیش بینی نماد فولاد دقتی حدود ۰.۷ دارد که دقت آن از افزودن اندیکاتور به شاخص هم‌وزن بیشتر است و شاخص کل تاثیر بیشتری برای تشخیص نماد فولاد دارد.

بخش‌دوم) ناهنجاری در سری‌های زمانی

د)

شبکه خودکدگذار با یادگیری الگوهای داده‌های ورودی، می‌توانند داده‌ها را به یک فرم فشرده‌تر تبدیل کنند.یک خودرمزگذار از دو قسمت تشکیل شده است :

1. **رمزگذار** : این بخشی از شبکه است که ورودی را به تعداد کمتری از بیت ها فشرده می کند. فضای نشان داده شده توسط این تعداد بیت کمتر "فضای پنهان" و نقطه حداکثر فشرده سازی "گلوگاه" نامیده می شود. این بیت های فشرده شده که نشانگر ورودی اصلی هستند ، "رمزگذاری" ورودی خوانده می شوند.
2. **رمزگشا** : این بخشی از شبکه است که ورودی را با استفاده از رمزگذاری ، بازسازی می کند. به صورت ایده آل خروجی رمزگشا باید همان داده ورودی باشد که نویز از روی آن حذف شده است.

تشخیص ناهنجاری، فرآیندی است که در آن داده‌های غیرعادی یا غیرمنتظره از داده‌های عادی جدا می‌شوند. شبکه‌های خودکدگذار یک روش برای تشخیص ناهنجاری در سری‌های زمانی هستند. این شبکه‌ها با یادگیری الگوهای عادی در داده‌ها، می‌توانند داده‌های غیرعادی را شناسایی کنند. فرآیند تشخیص ناهنجاری در سریهای زمانی با استفاده از شبکه خودکدگذار به شرح زیر است:

1. ابتدا، داده‌های سری زمانی به عنوان ورودی به شبکه خودکدگذار داده می‌شوند.
2. شبکه خودکدگذار سعی می‌کند داده‌های ورودی را به یک فرم فشرده‌تر تبدیل کند.
3. داده‌های فشرده‌شده سپس به عنوان خروجی از شبکه خودکدگذار خارج می‌شوند (با عبور از لایه رمزگشا).
4. داده‌های استخراج شده سپس با یک الگوریتم تشخیص ناهنجاری تجزیه و تحلیل می‌شوند (میتوان فاصله داده ورودی تا مقدار استخراج شده مورد استفاده قرار بگیرد).

اگر داده‌های فشرده‌شده از یک الگوی عادی پیروی نکنند، به عنوان داده‌های غیرعادی شناسایی می‌شوند. در واقع شبکه خودکدگذار سعی می‌کند داده‌های ورودی را به یک فرم فشرده‌تر تبدیل کند. داده‌های فشرده‌شده سپس با یک الگوریتم تشخیص ناهنجاری تجزیه و تحلیل می‌شوند. اگر داده‌های فشرده‌شده از یک الگوی عادی پیروی نکنند، به عنوان داده‌های غیرعادی شناسایی می‌شوند.

ه)

داده ورودی به سه قسمت train,test, val تقسیم میشود که به صورت زیر است:

```
x_train.shape, x_test.shape, x_val.shape
```

```
((2153, 1, 4), (126, 1, 4), (255, 1, 4))
```

مدل در نظر گرفته شده به صورت زیر است:

```
class AutoEncoder(nn.Module):
    def __init__(self):
        super(AutoEncoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(4, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 16),
            nn.ReLU(),
            nn.Linear(16, 8),
            nn.ReLU()
        )
        self.decoder = nn.Sequential(
            nn.Linear(8, 16),
            nn.ReLU(),
            nn.Linear(16, 32),
            nn.ReLU(),
            nn.Linear(32, 64),
            nn.ReLU(),
            nn.Linear(64, 4)
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded
```

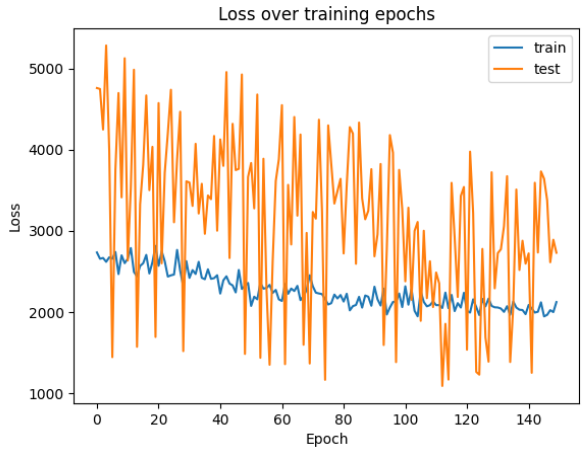
مدل خودکدگذار یک شبکه عصبی مصنوعی است که از دو جزء اصلی تشکیل شده است:

- رمزگذار: یک دنباله از لایه‌ها که داده‌های ورودی را به یک نمایش با ابعاد کمتر (کد مخفی) فشرده می‌کند.
    - ورودی: داده‌های ورودی با ابعاد 4 را می‌پذیرد.
    - لایه‌ها:
        - لایه خطی (4 به 64) با فعال‌سازی ReLU
        - لایه خطی (64 به 32) با فعال‌سازی ReLU
        - لایه خطی (32 به 16) با فعال‌سازی ReLU
        - لایه خطی (16 به 8) با فعال‌سازی ReLU
        - خروجی: یک کد مخفی 8 بعدی که داده‌های ورودی فشرده شده را نشان می‌دهد.
- رمزگشا: یک دنباله از لایه‌ها که سعی می‌کند داده‌های اصلی را از کد مخفی بازسازی کند.
    - ورودی: کد مخفی 8 بعدی از رمزگذار را می‌گیرد.

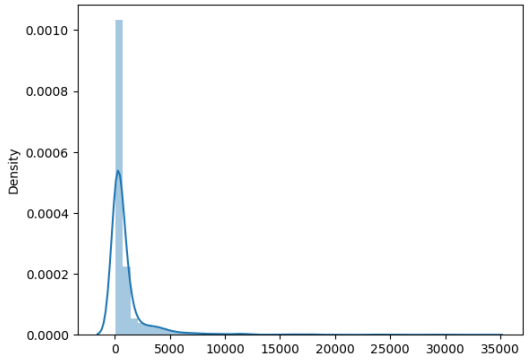
- لایه‌ها:
  - لایه خطی (8 به 16) با فعال‌سازی ReLU
  - لایه خطی (16 به 32) با فعال‌سازی ReLU
  - لایه خطی (32 به 64) با فعال‌سازی ReLU
  - لایه خطی (64 به 4) بدون فعال‌سازی (سعی می‌کند ابعاد داده‌های اصلی را خروجی دهد)
  - خروجی: بازسازی 4 بعدی از داده‌های ورودی.
- بهینه‌ساز: از بهینه‌ساز Adam برای به‌روزرسانی پارامترهای مدل در طول آموزش استفاده می‌شود.
- نرخ یادگیری: مقدار 1e-3 تنظیم شده است که کنترل می‌کند مدل هر مرحله آموزش چقدر پارامترها را تنظیم می‌کند.
- تابع loss: از تابع L1Loss (خطای مطلق میانگین) برای اندازه‌گیری تفاوت بین خروجی بازسازی شده و داده‌های اصلی استفاده می‌شود.

برای آموزش مدل از تابع train_model استفاده شده است، که آرگمان‌های (model, train_dataset, val_dataset, n_epochs) را برای ورودی می‌گیرد (در هر ایپاک ، فرایند آموزش مدل با تمام نمونه های آموزشی یاد می‌گیرد و عملکرد را در مجموعه اعتبار سنجی ارزیابی می کند.) و نتیجه بعد از ۱۵۰ ایپاک به صورت زیر است:

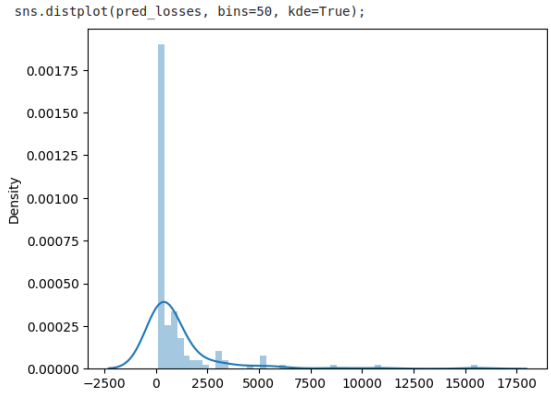
در شکل زیر منظور از test داده validation میباشد (مدل ما به خوبی همگرا شد. به نظر می رسد ممکن است برای هموارسازی نتایج به مجموعه اعتبارسنجی بزرگتری نیاز داشته باشیم، اما در حال حاضر این کار انجام خواهد شد.):



تابع predict برای نشان دادن خطای در مجموعه داده ورودی آن میباشد، در مجموعه آموزشی خطای بازسازی به صورت زیر است:



در مجموعه تست خطای بازسازی به صورت زیر است:

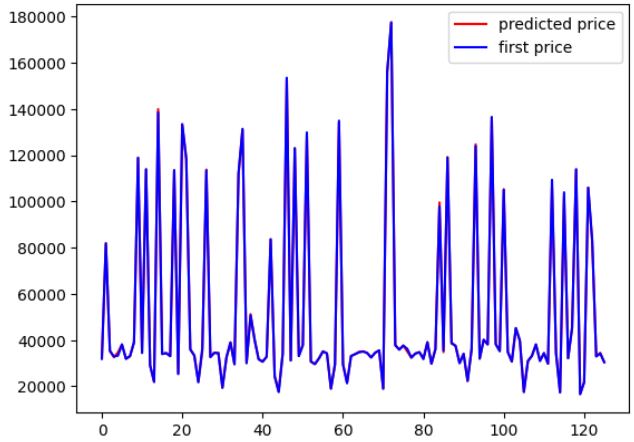


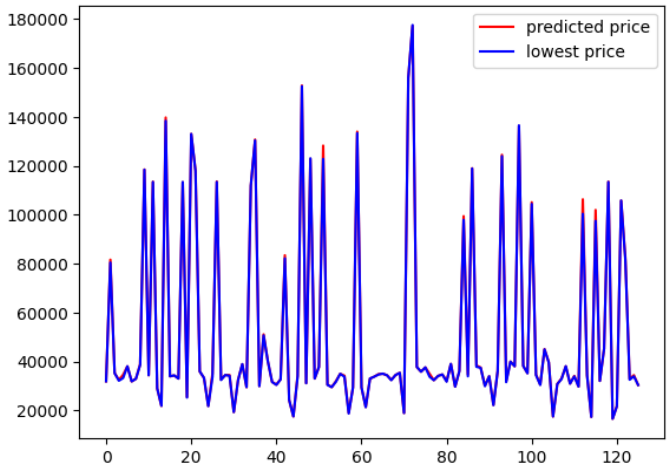
با استفاده از دو شکل بالا حد آستانه ۱۰۰۰۰ برای مجموعه داده در نظر گرفته می‌شود، به این معنا که اگر داده مقدار پیش‌بینی شده و واقعی بیشتر از این مقدار با یکدیگر اختلاف داشتند به عنوان ناهنجاری شناخته میشوند، روی مجموعه تست ناهنجاری‌ها به صورت زیر است:

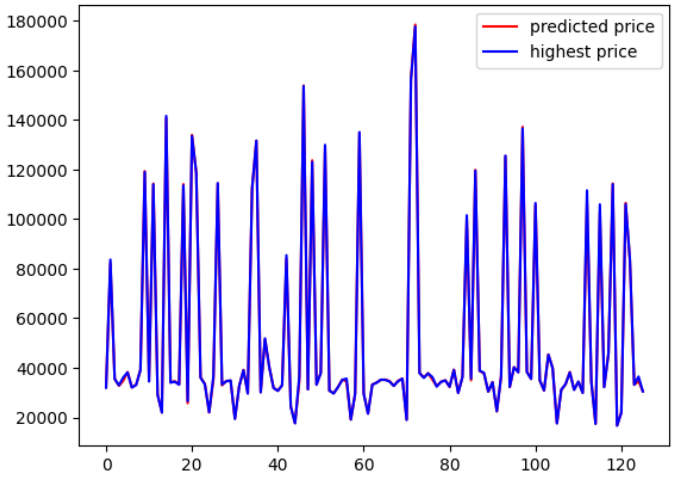
```
correct = sum(l <= THRESHOLD for l in pred_losses)
print(f'Correct normal predictions: {correct}/{len(x_test_tensor)}')
```

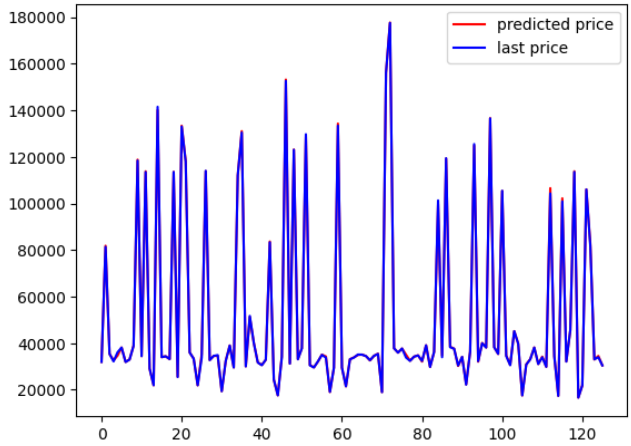
```
Correct normal predictions: 124/126
```

مقدار پیش بینی شده و مقدار واقعی روی مجموعه تست به صورت زیر است:









در نهایت با حد آستانه در نظر گرفته شده روی کل مجموعه داده ۳۸ به عنوان ناهنجاری شناسایی می شوندکه اطلاعات آنها در جدول زیر درج شده است:

	Timestamp	first	lowest	highest	last	first_pred	lowest_pred	highest_pred	last_pred
0	2011-11-27	81000.0	79620.0	90960.0	87050.0	84420.070312	84253.335938	84832.468750	84457.898438
1	2011-11-27	87000.0	78760.0	89430.0	81550.0	84218.125000	84051.804688	84629.484375	84255.835938
2	2011-11-27	88990.0	88730.0	95340.0	93700.0	91557.929688	91376.132812	92006.687500	91599.726562
3	2011-11-27	103780.0	101000.0	118260.0	116500.0	109789.562500	109569.304688	110331.210938	109841.531250
4	2011-11-27	116550.0	107810.0	117610.0	114700.0	114518.601562	114288.375000	115084.320312	114573.195312
5	2011-11-27	114710.0	95090.0	114890.0	105410.0	108149.695312	107932.914062	108682.976562	108200.750000
6	2011-11-27	105390.0	101010.0	111930.0	105980.0	105947.257812	105735.117188	106469.320312	105997.078125
7	2011-11-27	105990.0	95600.0	113120.0	112960.0	107432.195312	107216.914062	107961.820312	107482.851562
8	2011-11-27	112970.0	102100.0	113370.0	105770.0	108779.460938	108561.335938	109315.945312	108830.851562
9	2011-11-27	105760.0	99000.0	113460.0	101480.0	104590.429688	104381.148438	105105.585938	104639.507812
10	2011-11-27	95970.0	82310.0	98050.0	89760.0	91846.593750	91664.187500	92296.820312	91888.554688
11	2011-11-27	89780.0	89250.0	107860.0	102540.0	96895.429688	96702.375000	97371.375000	96940.210938
12	2011-11-27	102520.0	91740.0	111550.0	103360.0	102372.664062	102168.046875	102876.515625	102420.492188
13	2011-11-27	103350.0	103170.0	109670.0	109430.0	106346.468750	106133.476562	106870.562500	106396.507812
14	2011-11-27	109370.0	100370.0	111550.0	104280.0	106508.320312	106294.992188	107033.242188	106558.460938
15	2011-11-27	103970.0	97470.0	105910.0	100850.0	102145.273438	101941.132812	102647.960938	102192.976562
16	2011-11-27	100880.0	99470.0	106590.0	106260.0	103313.601562	103107.007812	103822.250000	103361.953125

17	2011-11-27	112060.0	104000.0	121980.0	119770.0	114690.726562	114460.132812	115257.328125	114745.429688
18	2011-11-27	119790.0	119370.0	135880.0	127950.0	125168.609375	124915.921875	125788.601562	125229.156250
19	2011-11-27	137140.0	136400.0	150670.0	136890.0	139434.296875	139151.515625	140126.968750	139502.781250
20	2011-11-27	136430.0	134680.0	144700.0	135110.0	137238.031250	136959.875000	137919.484375	137305.281250
21	2011-11-27	128710.0	120970.0	130320.0	129630.0	127816.531250	127558.257812	128450.000000	127878.546875
22	2011-11-27	129940.0	129550.0	138980.0	138160.0	134049.828125	133778.390625	134715.062500	134115.312500
23	2011-11-27	138190.0	135190.0	146060.0	135330.0	138215.250000	137935.046875	138901.718750	138283.062500
24	2011-11-27	113570.0	113570.0	121100.0	121100.0	117265.648438	117029.617188	117845.367188	117321.773438
25	2011-11-27	120240.0	113660.0	120290.0	114300.0	117198.554688	116962.679688	117777.937500	117254.656250
26	2011-11-27	108230.0	86900.0	109940.0	97100.0	101062.937500	100861.085938	101560.117188	101110.039062
27	2011-11-27	109000.0	109000.0	118000.0	112000.0	111588.484375	111364.429688	112139.273438	111641.445312
28	2011-11-27	125950.0	125900.0	132550.0	131540.0	128877.820312	128617.304688	129516.695312	128940.421875
29	2011-11-27	130840.0	129800.0	139560.0	129850.0	131967.250000	131700.218750	132621.890625	132031.578125
30	2011-11-27	144980.0	137340.0	145000.0	137500.0	141264.218750	140977.562500	141966.203125	141333.718750
31	2011-11-27	129350.0	129340.0	135000.0	134940.0	132111.687500	131844.343750	132767.046875	132176.078125
32	2011-11-27	119000.0	111890.0	119000.0	113890.0	116098.914062	115865.359375	116672.687500	116154.398438
33	2011-11-27	129540.0	129400.0	135550.0	135530.0	132464.187500	132196.109375	133121.359375	132528.812500
34	2011-11-27	147940.0	147910.0	156010.0	155910.0	151868.265625	151559.250000	152624.281250	151943.687500

35	2011-11-27	156970.0	148910.0	157010.0	148980.0	153023.687500	152712.234375	153785.578125	153099.750000
36	2011-11-27	147970.0	147950.0	156550.0	156550.0	152179.218750	151869.546875	152936.796875	152254.812500
37	2011-11-27	157540.0	148950.0	157550.0	149030.0	153327.796875	153015.703125	154091.234375	153404.031250
38	2011-11-27	149050.0	148950.0	159050.0	157950.0	153601.875000	153289.203125	154366.703125	153678.250000