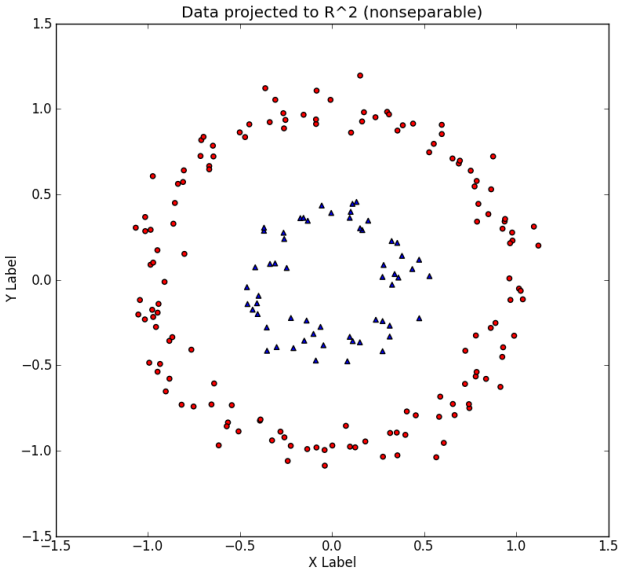
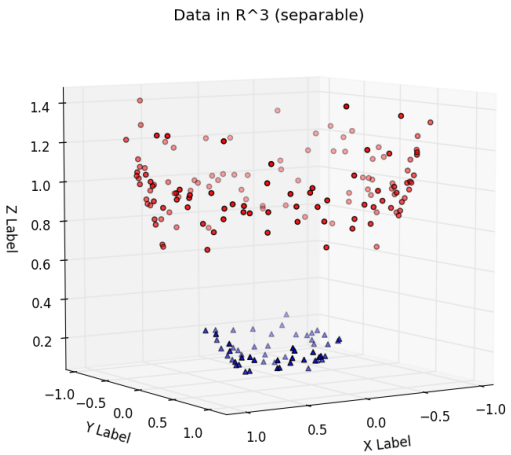
۱.

الگوریتم SVM با ماکزیموم کردن فاصله خط جدا کننده داده‌های دو کلاس، hyperplane خطی‌ای را بین داده‌ها پیدا می‌کند. حال اگر داده ما به صورت خطی قابل جداسازی نباشد، یک راهی که همچنان با SVM به نتیجه خوبی برسیم این است که داده‌هایمان را به فضای دیگری ببریم که در آن فضا به صورت خطی قابل جداسازی باشد. مانند مثال پایین که داده در دو بعد قابل جداسازی نیست اما اگر با تبدیلی آن را روی یک فضای با ابعاد بیشتر تصویر کنیم، احتمال اینکه در آن فضای با بعد بیشتر قابل جداسازی باشد را افزایش می‌دهیم.





از پر کاربردترین کرنل‌های SVM ، کرنل‌های polynomial و rbf هستند که هر دو سعی دارند با روش‌هایی تعداد فیچرها زیاد کنند تا ابعاد داده بزرگ‌تر شود.

کرنل polynomial :

این کرنل سعی می‌کند با ساختن ترکیب‌های چند جمله‌ای درجه n از فیچرها، تعداد فیچرها و درنتیجه ابعاد داده را افزایش دهد.

مثلاً برای فیچرهای a, b با اضافه کردن a ∧ 2, b ∧ 2, ab میتوانیم تعداد فیچرها را افزایش دهیم.

کرنل rbf:

این کرنل سعی می‌کند با اضافه کردن شباهت یک فیچر با یک نقطه ثابت (e به توان منفی فاصله این دو نقطه که بین صفر و یک است) فیچرهای جدیدی بسازد و بعد را افزایش دهد.

کرنل rbf شباهت را با فاصله اقلیدسی و کرنل polynomial شباهت را با ضرب داخلی محاسبه می‌کند. پس دو نقطه نزدیک مبدا که در دو سمت محور هستند در کرنل rbf عدد بالایی میگیرند اما در کرنل polynomial شباهت کمی دارند. می‌توان گفت در

مسائلی که شباهت بیشتر معنای فاصله می‌دهد بهتر است از rbf استفاده کنیم (در اکثر مسائل فاصله اقلیدسی متریک شباهت بهتری است). اما در مسائلی مانند پردازش متن و روش‌هایی مانند bag of words چون جهت بردارها بهتر شباهت را نشان می‌دهد بهتر است از کرنل‌های ضرب داخلی مثل polynomial استفاده کنیم.

۳.

در این بخش svm هایی با کرنل‌های linear, rbf, sigmoid, polynomial و پارامترهای مختلف مورد بررسی قرار گرفتند که به نظر میرسد داده‌های ما به صورت خطی قابل جداسازی می‌باشند و با کرنل linear میتوانیم به دقت 100 درصد برسیم. کرنل‌های rbfو polynomial 2 و polynomial3 و sigmoid نیز به ترتیب دقت‌های 99، 50، 79، 100 روی داده‌های تست رسیدند.
(البته لازم به ذکر است چون همه پارامترها نتایج خوبی میگرفتند و تفاوت قابل مشاهده نبود از ۲۰ درصد داده‌های برای training استفاده شده)

۴.

برای بررسی hard margin و soft margin دو مدل svm یکی با پارامتر C (میزان تاثیر missclassification) خیلی کوچک و یکی با C خیلی بزرگ روی داده‌های train آموزش دادیم. مدل با C خیلی کوچک در حقیقت soft margin است و عملکرد ضعیفی روی داده تست دارد (۲۴ درصد)،زیرا همانطور که گفته شد داده ما به صورت خطی قابل جداسازی است و اضافه کردن soft margin باعث می‌شود اجازه دهیم missclassification رخ دهد. اما مدل با C بزرگ که hard margin است همچنان به دقت ۱۰۰ درصد میرسد.

۵.

ب) چون با استفاده از روش‌هایی مثل label encoding که به هر متغیر categorical یک مقدار عددی نسبت می‌دهد، مدل را به اشتباه می‌اندازیم که بین این داده‌های categorical ترتیبی وجود دارد. مثلاً اگر متغیر ما رنگ باشد و ما رنگ آبی را ۱ و رنگ قرمز را ۲ و رنگ سبز را ۳ در نظر بگیریم، داریم به مدل القا می‌کنیم که رنگ آبی نزدیک تر به رنگ قرمز است تا رنگ سبز. به همین دلیل از one hot encoding استفاده می‌کنیم که این مشکل پیش نیاید.

ج) چون در اکثر مدل‌های ماشین لرنینگ فرض بر نرمال بودن داده‌ها است(با توجه به قضیه حد مرکزی)، اگر توزیع داده ما نرمال نباشد یا توزیع کشیدگی زیادی داشته باشد ممکن است مدل ما همگرا نشود. با استفاده از log transformation و تبدیل‌های دیگر

سعی داریم با توجه به نوع توزیع داده‌هایمان این کشیدگی توزیع را کم‌تر کنیم مثلا هنگامی که توزیع داده نمایی است می‌توانیم با log transformation توزیع داده‌ها را بهتر کنیم.

.۷

از مشهور ترین الگوریتم‌های ساخت درخت تصمیم می‌توان به id3, c4.5, cart اشاره کرد. تفاوت اصلی این الگوریتم‌ها در نحوه محاسبه خلوص داده است.

- الگوریتم ID3:

در این الگوریتم برای پیدا کردن فیچر روت درخت ابتدا آنتروپی کل داده را محاسبه می‌کنیم(که اگر تعداد داده‌های کلاس‌ها برابر باشد، آنتروپی ماکزیموم است و عدد ۱ را می‌گیرد و اگر داده‌ها از یک کلاس باشند آنتروپی مقدار مینیمم خود که صفر است را می‌گیرد). سپس برای هر فیچر و هر مقدار آن فیچر آنتروپی داده‌ها را حساب می‌کنیم و میانگین وزن دار (با وزن تعداد داده‌ها با آن مقدار) آنتروپی مقادیر مختلف را از آنتروپی کل کم می‌کنیم و به عنوان اطلاعاتی که آن فیچر به ما میدهد در نظر می‌گیریم. حال فیچری که بیشترین اطلاعات را به ما می‌دهد را به عنوان روت درخت قرار می‌دهیم. سپس در زیر درخت‌های که هنوز آنتروپی در آن‌ها زیاد است، دوباره همین الگوریتم را برای بقیه فیچرها اجرا می‌کنیم تا در نهایت همه برگ‌ها آنتروپی قابل قبولی داشته باشند.

- الگوریتم CART:

الگورتیم CART نیز بسیار شبیه الگوریتم ID3 عمل می‌کند با این تفاوت که به جای سنجیدن یکپارچکی داده با آنتروپی از مفهموم gini index استفاده می‌کند. در gini ما در حقیقت احتمال اینکه دو داده دلخواهی که انتخاب می‌کنیم از کلاس‌های متفاوتی باشند را اندازه می‌گیریم. پس اگر داده‌ها یکدست باشند gini صفر و اگر از هر کلاس تعداد برابر باشد یک می‌شود.

- الگوریتم C4.5:

الگوریتم C4.5 نیز شباهت بسیار زیادی به ID3 دارد و در حقیقت یک بهبود روی الگوریتم ID3 است. در این الگوریتم Gain محاسبه شده در ID3 را نورمالایز می‌کنیم و همچنین روی درخت نهایی prunning انجام میدهیم که در بخش بعدی توضیح داده می‌شود.

Features	*ID3*	*C4.5*	*CART*
Type of data	Categorical	Continuous and Categorical	continuous and nominal attributes data
Speed	Low	Faster than ID3	Average
Boosting	Not supported	Not supported	Supported
Pruning	No	Pre-pruning	Post pruning
Missing Values	Can't deal with	Can't deal with	Can deal with
Formula	Use information entropy and information Gain	Use split info and gain ratio	Use Gini diversity index

۹.

برای این سوال از grid search استفاده شده و مقادیر مختلفی برای عمق درخت و تعداد نمونه‌های موجود در هر گره امتحان شدند. که فقط در حالتی که ماکزیمم عمق درخت را به ۲ محدود می‌کنیم درخت قادر به کلاسبندی با ۱۰۰ درصد دقت نمی‌باشد.

۱۰.

انجام prunning روی درخت تصمیم باعث می‌شود مدل generalized تر شود و اگر مشکل overfitting داشته باشیم می‌توانیم این مشکل را کمتر کنیم. حرص کردن درخت تصمیم می‌تواند در حین درخت تصمیم انجام شود یا بعد از ساخته شدن درخت. برای حرص کردن در حین ساخت درخت می‌توان روش‌هایی مثل مشخص کردن ماکزیمم عمق درخت یا حداقل تعداد نمونه‌ها در هر گره نام برد. اما اگر درخت ما عمق زیادی دارد و روی داده‌های تست خوب جواب نمی‌دهد می‌توانیم از روش‌های post-prunning استفاده کنیم. معمولاً برای انجام post-prunning به validation set نیاز داریم. یکی از روش‌های معروف post-prunning با جایگذاری کلاسی که در برگ‌های یک زیر درخت بیشتر تکرار شده(یا در رگرسیون میانگین برگ‌ها) به جای خود زیر درخت و چک کردن عملکرد درخت جدید روی validation set سعی می‌کند درخت را کوچک‌تر کند. این عمل را می‌توان به صورت recursive روی درخت جدید ساخته شده نیز انجام داد.

۱۲.

چون در مسئله ما درخت تصمیم نیز نتیجه ۱۰۰ درصد می‌گیرد، مقایسه خوبی از دقت این دو مدل نمی‌توان داشت. اما احتمالاً نتایج random forest در خیلی از مسائل بهتر است زیرا از قدرت generalization بیشتری برخوردار است که این قدرت را با استفاده از ensemble learning بدست می‌آورد. با آموزش دادن درخت‌های تصمیم کوچک (pre-prunned با max_depth) و همچنین نشان دادن زیرمجموعه‌های مختلف از داده به هرکدام و در نهایت تجمیع نظرات این درخت‌ها، به generalization خیلی بهتری نسبت به فقط یک درخت تصمیم می‌رسد.

۱۳.

در خیلی از مسائل دنیای واقعی تعداد داده‌های در دسترس ما به اندازه‌ای بزرگ نیست که بتوانیم به راحتی از شبکه‌های عمیق استفاده کنیم. در نتیجه یکی از الگوریتم‌های خوب ماشین لرنینگ کلاسیک که درخت‌های تصمیم هستند(انواع مختلف آن‌ها مثل random forest و boosted trees) استفاده می‌کنیم که معمولاً نتایج خیلی خوبی در خیلی از مسائل می‌دهند.

یکی دیگر از دلایل استفاده زیاد از این الگوریتم‌ها تفسیرپذیری راحت این الگوریتم است. در خیلی از مسائل دنیای واقعی لازم داریم بدانیم به چه دلیل و در چه مواردی الگوریتم ما یک تصمیم را می‌گیرد. مثلاً اگر موردی را به اشتباه تشخیص دادیم لازم داریم بدانیم

چرا الگوریتم چنین تصمیمی گرفته است. این امر در شبکه‌های عصبی عمیق امری بسیار دشوار است در حالی که تفسیر الگوریتمی مانند درخت تصمیم، حتی برای افرادی بدون پیش زمینه ماشین لرنینگ نیز بسیار ساده است.

۱۵.

برای حل مسائل سری زمانی نیز می‌توان از درخت‌های تصمیم استفاده کرد. تنها تفاوت این است که داده‌ای که به آن ورودی می‌دهیم متفاوت خواهد بود. در هر لحظه از سری زمانی که بخواهیم پیش‌بینی انجام بدهیم می‌توانیم داده‌های k استپ زمانی قبل را به عنوان k فیچر ورودی درخت تصمیم بدهیم. تنها مشکل این روش این است که تقدم و تاخر یا در حقیقت تاثیر مستقیم زمان در فیچرها در این روش نادیده گرفته می‌شود.

۲۱.

برای پیش‌بینی داده‌های بیت کوین روش‌های مختلفی تست شد. که در ادامه هر کدام توضیح داده می‌شوند. در تمامی تست‌ها تعداد استپ زمانی که از گذشته درنظر گرفته می‌شود ۷ است(اعداد مختلفی تست شدند اما به نظر ۷ هایپرپارامتر بهتری است)

- الگوریتم Random Forest

در این قسمت با استفاده از grid search، حالت‌های مختلف random forest با تعداد درخت 10, 50, 100, 200, 500 و مینیمم عضو گره 10, 20, 50 و ماکزیمم عمق 3, 5, 7 را بررسی شدند که بهترین مدل با هایپرپارامترهای max_depth=7, min_samples_split=10, n_estimators=10 بود که نتایج زیر را گرفت.

```
RMSE :  16329.027051500707
-------------------------------------------------
accuracy with 5% error :  0.5397489539748954
```

- الگوریتم MLPRegressor

این الگوریتم در حقیقت یک پیاده سازی از شبکه عصبی است که در کتابخانه sklearn پیاده سازی شده. این مدل روی داده‌های تست نتایج زیر را گرفت.

```
RMSE :  4637.311646596395
-------------------------------------------------
accuracy with 5% error :  0.5585774058577406
```

- الگوریتم Gradient Boosting

روش gradient boosting خیلی شبیه به روش xgboost است که از مفهوم boosting روی درخت‌ها استفاده می‌کند اما اینطور که در داکیومنتیشن xgboost مطرح شده در جزئیات این دو مدل تفاوت‌هایی دارند. این مدل روی داده‌های تست نتایج زیر را گرفت.

```
RMSE :  16112.93608767365
-----------------------------------------
accuracy with 5% error :  0.5878661087866108
```

- الگوریتم XgBoost

این الگوریتم از معروف‌ترین الگوریتم‌های درخت تصمیم است که در خیلی از مسائل دنیای واقعی نیز استفاده می‌شود(مثلاً اوبر در مقاله‌ای نوشته است که تعداد زیادی از مدل‌های در حال استفاده در اوبر همین xgboost است). برای استفاده از این الگوریتم از کتابخانه xgboost استفاده شده است. نتایج بدست آمده از این مدل به صورت زیر است.

```
RMSE :  16443.322527470613
-----------------------------------------
accuracy with 5% error :  0.5732217573221757
```

- الگوریتم AdaBoost

این الگوریتم نیز یکی دیگر از الگوریتم‌هایی است که از ایده boosting استفاده می‌کند. با استفاده از grid search، مدل‌های ممکن برای مجموعه هایپرپارامترهای تعداد درخت 10, 50, 100, 200, 500 و لرنینگ ریت 1, 0.1, 0.001 و لاس‌های linear , square ساخته شده و تست شدند که مدل با هایپرپارامترهای loss=square, learning_rate=0.01, n_estimators=50 بهترین نتیجه را گرفت.

```
RMSE :  15761.331683344924
-----------------------------------------
accuracy with 5% error :  0.5460251046025104
```

- الگوریتم SVM

برای انجام دادن رگرسیون با SVM از الگوریتم SVR پیاده سازی شده در کتابخانه sklearn استفاده شد که نتایج این الگوریتم روی داده‌های تست به صورت زیر بوده.

```
RMSE :  19797.625679009638
-----------------------------------------
accuracy with 5% error :  0.5711297071129707
```

- الگوریتم‌های Vanilla RNN, LSTM, GRU

در مرحله اول بیشترین امید را به الگورتیم‌های از جنس RNN داشتم تا بهترین نتایج را روی مسئله پیش‌بینی سری زمانی بگیرند. اما با پیاده سازی مدل‌های مختلف از جمله vanilla RNN, LSTM, GRU متاسفانه نتایج به خوبی که تصور میشد نبود. نتایج بدست آمده از هر مدل به صورت زیر است.

مدل VanillaRNN:

```
{'accuracy': 0.34309623430962344,
 'mae': 8749.804,
 'r2': 0.04992557369663353,
 'rmse': 16770.633381002637}
```

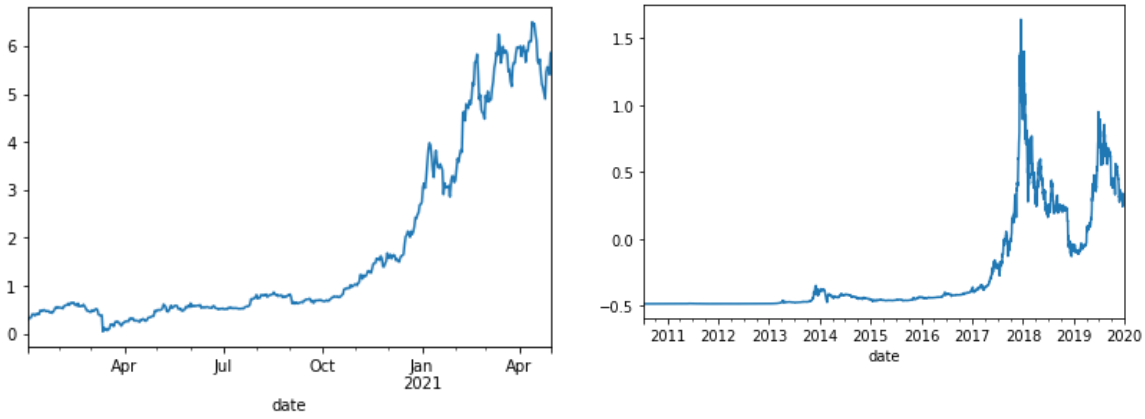
مدل LSTM:

```
{'accuracy': 0.401673640167364,
 'mae': 0.91858786,
 'r2': 0.09154286159127223,
 'rmse': 1.8018581124690987}
```

مدل GRU:

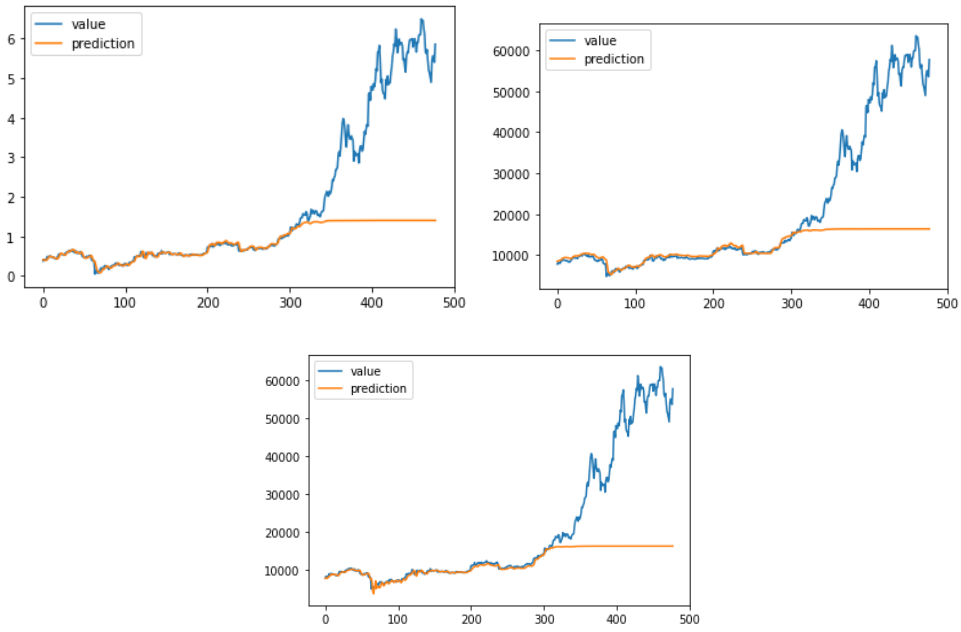
```
{'accuracy': 0.45397489539748953,
 'mae': 8759.393,
 'r2': 0.03436745996878088,
 'rmse': 16907.391046521636}
```

به همین دلیل به دنبال دلیل این عملکرد بد رفتم و حدسی که میزدم تفاوت فاحش رنج داده‌های تست و ترین بود. همانطور که در نمودار زیر می‌بینید نمودار سمت راست که داده ترین است ماکزیموم تا مقدار ۱.۵ را دارد(واحد محور y متریک z-score میباشد) و نمودار سمت چپ که داده‌های تست هستند اکثرا بالای ۱.۵ هستند. در حقیقت این مدل‌ها نمی‌توانند حتی تابع خطی را برای مقادیری که ندیده‌اند پیش‌بینی کنند(عمل شمارش اعداد را نمی‌توانند یاد بگیرند).



Test set **Training set**

از دیگر شواهدی که می‌تواند نشان داد این عملکرد بد به خاطر این مشکل است می‌توان به نتایج به دست آمده از هر یک از مدل‌ها روی داده‌های تست است. همانطور که در نمودارهای زیر می‌بینید هر سه مدل از قسمتی که داده‌ها از رنج داده‌های training بیشتر می‌شوند توانایی پیشبینی ندارند.



برای رفع این مشکل دو ایده تست شد که به شرح زیر است:

- استفاده از min max scaling در هر ورودی RNN. یعنی مثلاً اگر ۷ استپ زمانی گذشته را میبینیم ابتدا آن‌ها را minmax scale کنیم تا همه داده‌ها بین صفر و یک باشند و از مدل بخواهیم این مقادیر scale شده را پیش‌بینی کند. با این روش مشکل رنج‌های متفاوت تست و ترین حل می‌شود. اما متاسفانه این روش هم مشکلات خود را داشت و مدل نمی‌توانست از این داده‌های scale شده چیز زیادی بیاموزد.
- استفاده از diff به جای داده‌های واقعی. یعنی به جای عدد واقعی قیمت تفاوت قیمت را روز گذشته را به مدل بدهیم و از مدل بخواهیم تفاوت با آخرین روز را به ما بدهد. این ایده نیز متاسفانه به جایی نرسید و مدل همیشه مقدار میانگین را پیش‌بینی می‌کرد.

متاسفانه علیرغم وقت زیادی که برای حل این مشکل گذاشته شد در نهایت این مشکل برجا ماند و موفق نشدم مدل‌های RNN خوبی با این داده‌های test , train آموزش بدهم.

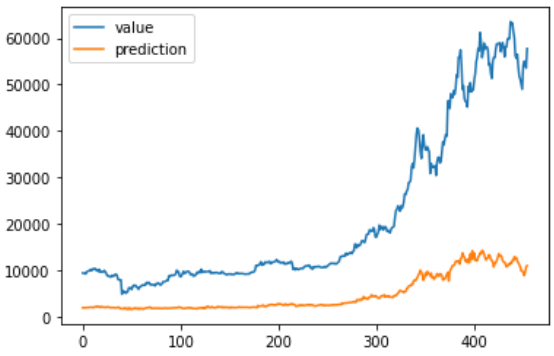
- مدل Torch NN

در این مدل پیاده سازی یک مدل شبکه عصبی با pytorch انجام شده است که نتایج حاصل به صورت زیر است.

```
{'accuracy': 0.401673640167364,
 'mae': 1209.265,
 'r2': 0.988426158422957,
 'rmse': 1851.0127633271468}
```

- مدل CNN

در این مدل با استفاده از کرنل‌های یک بعدی با طول ۳ سعی شده فیچرهایی از تایم سری استخراج شود و درنهایت با فلت کردن آخرین لایه از فیلترها و استفاده از یک شبکه fully connected پیش‌بینی قیمت انجام شود. که متاسفانه مدل نتایج خوبی را نگرفت و با تحلیل‌های مختلف و دیباگ کردن مدل نیز موفق به پیدا کردن مشکل مدل نشدم. در نحوه پیش‌بینی مدل روی داده‌های تست می‌توان دید مدل کلیت روند را درست تشخیص میدهد اما با مقدار واقعی فاصله فاحشی دارد.




```
{'accuracy': 0.01098901098901099,
 'mae': 12332.331,
 'r2': -0.04866431265727944,
 'rmse': 17842.40611576813}
```

- مدل ensemble شده

در این مرحله ۵ مدل از مدل‌های random forest, xgboost, gradient boosting, adaboost , svr را با دو روش مختلف ensemble شده‌اند. یک روش stacking است. در این نوع از ensemble learning نتایج بدست آمده از مدل‌های مختلف را به هم میچسبانیم و به عنوان فیچر به یک رگرسور دیگر می‌دهیم تا نتیجه نهایی را به ما بدهد. در این جا از MLPRegressor به عنوان مدل دیگری که نتایج را تجمیع کند استفاده شده‌است. نتایج به دست آمده با این روش به شرح زیر است.

```
RMSE :  4594.954621296016
--------------------------------
accuracy with 5% error :  0.5
```

روش دیگری که برای ensemble learning استفاده شده، استفاده از ایده bagging و stacking با هم است. در این روش به جای اینکه همه مدل‌ها روی یک داده آموزش داده شوند و سپس نتایجشان stack شود، هر مدل روی بخشی از دیتای bootstrap شده آموزش داده می‌شوند. نتایج به دست آمده از این روش با همان مدل‌های قبلی به این صورت بود.

```
RMSE :  16043.237039714808
--------------------------------------------------
accuracy with 5% error :  0.600418410041841
```

در این مرحله از همان LSTM استفاده شده در بخش پیش استفاده شده است. با این تفاوت که متغیر target به جای قیمت یک متغیر باینری شده که نشان میدهد قیمت افزایش داشته یا خیر و لاس استفاده شده از mse به bce تغییر کرده. با استفاده از این دیتا مدل عملکرد خوبی نداشته و همیشه یک عدد خاص مثل .47 را پیش‌بینی می‌کند. راهی که می‌تواند مدل را بهبود دهد این است که به جای داده‌های قیمت صرفاً افزایش قیمت را به مدل بدهیم یعنی اگر قیمت افزایش داشته یک و اگر نداشته صفر به عنوان ورودی بدهیم با این روش مدل بهتر یاد می‌گیرد و پیش‌بینی‌های متفاوتی روی تست انجام میدهد.

اما مدل با دقت حدود ۵۴ درصد روی داده تست درست پیش‌بینی می‌کند که دقت خوبی نیست چون در داده‌های تست حدود ۵۴ درصد داده‌ها یک هستند و اگر همه را یک پیش‌بینی می‌کردیم نیز به همین دقت می‌رسیدیم.

.۲۶

در این بخش روی داده‌های سهام شرکت داده شده سه مدل XgBoost, MLPRegressor, SVM بررسی شدند که نتایج آن‌ها به صورت زیر بود.

مدل xgboost:

```
RMSE :  0.22312069615156965
-------------------------------------------------
accuracy with 5% error :  0.47514995715509856
```

مدل MLPRegressor:

```
RMSE :  0.223550149930654 67
-------------------------------------------------
accuracy with 5% error :  0.472160777983699
```

مدل SVR: