دانشکده علوم ریاضی
گروه علوم کامپیوتر

گزارش تمرین ۳

نگارش

ریحانه داورزنی

استاد

دکتر هادی فراهانی

تیر ۱۴۰۰

فهرست مطالب

۱ مقدمه **۳**

۲ تمرین **۵**
۱.۲ یک ۵
۲.۲ دو ۷
۳.۲ سه ۸
۴.۲ چهار ۱۱
۵.۲ پنج ۱۴
۶.۲ شش ۱۶
۷.۲ هفت ۱۸
۸.۲ هشت ۲۰
۹.۲ نه ۲۰
۱۰.۲ ده ۲۰
۱۱.۲ یازده ۲۱
۱۲.۲ دوازده ۲۱
۱۳.۲ سیزده ۲۲
۱۴.۲ چهارده ۲۳
۱۵.۲ پانزده ۲۴
۱۶.۲ شانزده ۲۴
۱۷.۲ هفده ۲۵
۱۸.۲ هجده ۳۴
۱۹.۲ نوزده ۳۵
۲۰.۲ بیست ۳۷
۲۱.۲ بیست و دو ۳۸
۲۲.۲ بیست و سه ۳۹
۲۳.۲ بیست و شش ۳۹

۱ مقدمه

- «ماشین بردار پشتیبان» (SVM) یک الگوریتم نظارت‌شده یادگیری ماشین است که هم برای مسائل طبقه‌بندی و هم مسائل رگرسیون قابل استفاده است؛ با این حال از آن بیشتر در مسائل طبقه‌بندی استفاده می‌شود. در الگوریتم SVM، هر نمونه داده را به عنوان یک نقطه در فضای n بعدی روی نمودار پراکندگی داده‌ها ترسیم کرده (n تعداد ویژگی‌هایی است که یک نمونه داده دارد) و مقدار هر ویژگی مربوط به داده‌ها، یکی از مؤلفه‌های مختصات نقطه روی نمودار را مشخص می‌کند. سپس، با ترسیم یک خط راست، داده‌های مختلف و متمایز از یکدیگر را دسته‌بندی می‌کند.بردارهای پشتیبان در واقع مختصات یک مشاهده منفرد هستند. ماشین بردار پشتیبان مرزی است که به بهترین شکل دسته‌های داده‌ها را از یکدیگر جدا می‌کند.

- درخت تصمیم که هدف اصلی آن، دسته‌بندی داده‌هاست، مدلی در داده‌کاوی است که مشابه فلوچارت، ساختاری درخت‌مانند را جهت اخذ تصمیم و تعیین کلاس و دسته یک داده خاص به ما ارائه می‌کند. این درخت از تعدادی گره و شاخه تشکیل‌شده است به‌گونه‌ای که‌برگ‌ها کلاس‌ها یا دسته‌بندی‌ها را نشان می‌دهند و گره‌های میانی هم برای تصمیم‌گیری با توجه به یک یا چند صفت خاصه به‌کارمی‌روند.

- Random Forest ، یک الگوریتم یادگیری ماشین با قابلیت استفاده آسان است که اغلب اوقات نتایج بسیار خوبی را حتی بدون تنظیم فراپارامترهای آن، فراهم می‌کند. این الگوریتم به دلیل سادگی و قابلیت استفاده، هم برای «دسته‌بندی» و هم «رگرسیون»، یکی از پر کاربردترین الگوریتم‌های یادگیری ماشین محسوب می‌شود. جنگل تصادفی یک الگوریتم یادگیری نظارت شده محسوب می‌شود. این الگوریتم جنگلی را به طور تصادفی می‌سازد. «جنگل» ساخته شده، در واقع گروهی از «درخت‌های تصمیم»است. کار ساخت جنگل با استفاده از درخت‌ها اغلب اوقات به روش Bagging انجام می‌شود. ایده اصلی روش Bagging آن است که ترکیبی از مدل‌های یادگیری، نتایج کلی مدل را افزایش می‌دهد. به بیان ساده، جنگل تصادفی چندین درخت تصمیم ساخته و آن‌ها را با یکدیگر ادغام

می‌کند تا پیش‌بینی‌های صحیح‌تر و پایدارتری حاصل شوند.

- یک مسئله پیش بینی سری زمانی که در آن می خواهد یک یا چند مقدار عددی آینده را پیش بینی کند ، یک مسئله مدل سازی پیش بینی نوع رگرسیون است.

۲ تمرین

۱.۲ یک

تابع کرنل روشی است که برای گرفتن داده به عنوان ورودی و تبدیل به فرم مورد نیاز پردازش داده، استفاده می شود. کرنل به دلیل مجموعه ای از توابع ریاضی مورد استفاده در SVM پنجره‌ای را برای دستکاری داده‌ها به کار می‌برد. بنابراین ، عملکرد هسته به طور کلی training set را به گونه ای تغییر می دهد که سطح تصمیم گیری غیر خطی قادر به تبدیل شدن به یک معادله خطی در تعداد بیشتری از فضاهای بعد باشد. اساساً ، ضرب داخلی[۱] را در یک بعد ویژگی استاندارد بین دو نقطه برمی گرداند.
معادله تابع کرنل استاندارد به صورت زیر می‌باشد:

$$K(\bar{x}) = 1, if||\bar{x}|| <= 1$$
$$K(\bar{x}) = 0, Otherwise$$

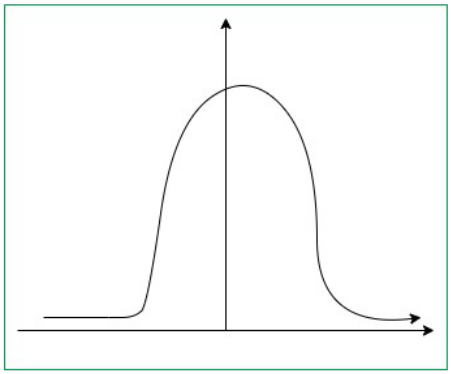
- کرنل گاوسی [۲] : این یک کرنل برای اهداف عمومی است. و هنگامی که هیچ دانش پیشینی در مورد داده‌ها وجود ندارد استفاده می‌شود. معادله آن به صورت زیر است :

$$K(x, y) = e^{-(\frac{||x-y||^2}{2\sigma^2})}$$

- کرنل تابع پایه شعاعی گاوسی [۳] : همان عملکرد کرنل فوق می‌باشد یعنی برای اهداف عمومی کاربرد دارد. و هنگامی که هیچ دانش پیشینی در مورد داده ها وجود نداشته باشد، مورد استفاده قرار می گیرد.با این تفاوت که برای بهبود انتقالات روش پایه شعاعی را می‌افزاید. معادله آن به صورت زیر است :

[1] inner product
[2] Gaussian Kernel
[3] Gaussian Kernel Radial Basis Function (RBF)

$$K(x,y) = e^{-}(\gamma||x-y||^2)$$
$$K(x,x1) + K(x,x2)(Simplified - Formula)$$
$$K(x,x1) + K(x,x2) > 0(Green)$$
$$K(x,x1) + K(x,x2) = 0(Red)$$



- کرنل سیگموئید [۴] : این عملکرد معادل یک مدل دو لایه ، پرسپترون شبکه عصبی است که به عنوان تابع activation برای نورونهای مصنوعی استفاده می شود. معادله آن به صورت زیر است :

$$K(x,y) = tanh(\gamma.x^T y + r)$$

- کرنل چند جمله‌ای [۵] : این کرنل نشان دهنده شباهت بردارها در مجموعه داده‌های آموزشی در یک feature space نسبت به چند جمله‌ای متغیرهای اصلی مورد استفاده در کرنل است. همچنین این کرنل در پردازش تصویر پرکاربرد است. معادله آن به صورت زیر است :

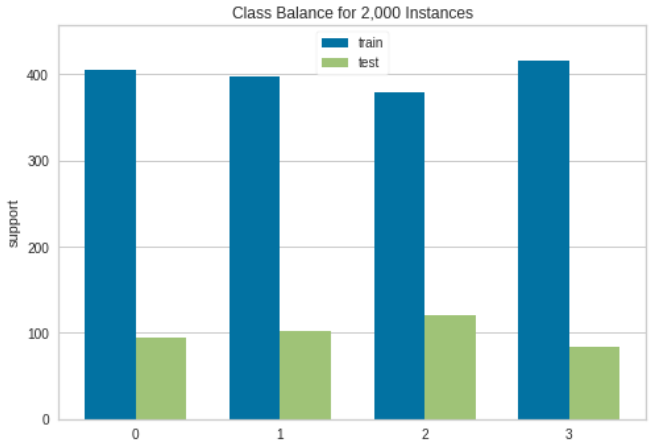
$$K(x,y) = tanh(\gamma.x^T y + r)^d, \gamma > 0$$

- کرنل خطی [۶] : هنگامی که داده ها به صورت خطی قابل تفکیک هستند استفاده می‌شود.

[4] Sigmoid Kernel
[5] Polynomial Kernel
[6] Linear Kernel

۲.۲ دو

در این قسمت SVM را روی دیتاست کلاس بندی قیمت موبایل اجرا می‌کنیم. در ابتدا می‌بینیم که هر کلاس به چه صورت تقسیم بندی شده است که به شکل زیر می‌باشد ، تقریبا تعداد اعضای هر کلاس برابر است.



SVM را به صورت زیر پیاده‌سازی می‌کنیم:

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(x_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(x_test)
```

از متریک‌های رایج‌ای که برای تسک‌های classification استفاده می‌شود می‌توان به Confusion matrix ، recall ، precision و معیار F۱ اشاره کرد. نتایج ارزیابی به شرح زیر است:

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

print("Precision:",metrics.precision_score(y_test, y_pred,average='macro'))

print("Recall:",metrics.recall_score(y_test, y_pred,average='macro'))

print("F1:",metrics.f1_score(y_test, y_pred,average='macro'))
```

```
Accuracy: 0.935
Precision: 0.935322004507089
Recall: 0.9410569014839979
F1: 0.9368334089109775
```

و برای confusion matrix به شرح زیر می‌باشد:

```
[[ 92   2   0   0]
 [  4  97   1   0]
 [  0  10 103   7]
 [  0   0   2  82]]
```

همان طورکه مشاهده می‌کنید ، تمام معیارها حاصلی نزدیک یک را به دست آورده‌اند که نتیجه مناسبی می‌باشد . همچنین تعداد misclassification که از ماتریس بالا نیز قابل مشاهده است نیز تعداد زیادی نمی‌باشد. برای کلاس یک ، ۲ مورد ، کلاس دو ، ۵ مورد ، کلاس سه ، ۱۷ مورد و کلاس چهار ۲ مورد می‌باشد.

۳.۲ سه

- در این بخش ، ابتدا از کرنل چندجمله‌ای به صورت زیر استفاده می‌کنیم :

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='poly', degree=8)
svclassifier.fit(x_train, y_train)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=8, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
y_pred = svclassifier.predict(x_test)
```

```
Accuracy: 0.25
Precision: 0.1999223602484472
Recall: 0.2925531914893617
F1: 0.15849412447410416

[[ 16   0   0  78]
 [ 10   0   0  92]
 [  2   4   0 114]
 [  0   0   0  84]]
```

نتایج را در بالا مشاهده می‌کنید همان طور که واضح است نتایج معیارها اصلا قابل قبول نمی‌باشد و تعداد misclassification به شدت بالا است. پس می‌توان گفت کرنل چندجمله‌ای مورد مناسبی برای استفاده نمی‌باشد.

- در این قسمت از کرنل RBF به صورت زیر استفاده می‌کنیم:

```
1 svclassifier = SVC(kernel='rbf')
2 svclassifier.fit(x_train, y_train)
3 y_pred = svclassifier.predict(x_test)
```

و نتایج به دست آمده به شرح زیر می‌باشد:

همان طور که مشاهده می‌کنید در اینجا هم مانند قسمت قبل نتایج قابل قبولی حاصل نشده است و کلاس‌بندی فقط در دو کلاس است که باعث misclassi-fication بالایی شده است.

```
Accuracy: 0.2225
Precision: 0.11118321297319525
Recall: 0.24841691995947315
F1: 0.15321588264678293
      [[52  0  0 42]
       [64  0  0 38]
       [70  0  0 50]
       [47  0  0 37]]
```

- در این قسمت از کرنل sigmoid استفاده شده است که نتایج آن به صورت زیر می‌باشد:

```
Accuracy: 0.2075
Precision: 0.12817619404706967
Recall: 0.21230034567018297
F1: 0.14542025166007802
     [[40 51  0  3]
      [55 42  0  5]
      [60 58  0  2]
      [42 41  0  1]]
```

همان طور که واضح است در اینجا هم دوباره نتیجه مطلوبی حاصل نشده است و طبقه بندی به طور کلی در سه کلاس انجام شده است که باعث misclassifi-cation بالایی شده است.

- در این بخش بازهم از کرنل RBF استفاده شده است با این تفاوت که مقدار پارامتر gamma را از مقدار پیش فرض scale به auto تغییر داده شده است ، که gamma ضریب کرنل است که زمانی که مقدار پیش فرض‌اش را دارد برابر $1/(n_{features} * X.var())$ و زمانی که مقدار auto را میگیرد برابر $1/n_{features}$ است. که به صورت زیر می‌باشد:

```
svclassifier = SVC(kernel='rbf',gamma='auto')
svclassifier.fit(x_train, y_train)
y_pred = svclassifier.predict(x_test)
```

و نتایج این حالت به صورت زیر می‌باشد ، که همان طورکه مشاهده می‌کنید نتایج خوبی نیست ولی از حالت قبلی که مقدار gamma مقدار پیش فرض خودش بود ، بهتر عمل کرد.

```
[[30 26 16 22]
 [27 32 12 31]
 [22 33 32 33]
 [20 15 14 35]]
Accuracy: 0.3225
Precision: 0.3316514315656683
Recall: 0.32905193992490617
F1: 0.32248336463962324
```

- در این بخش از کرنل چندجمله‌ای استفاده می‌شود ولی مثل حالت قبل مقدار gamma را برابر auto قرار داده شده است. نتایج به صورت زیر می‌باشد:

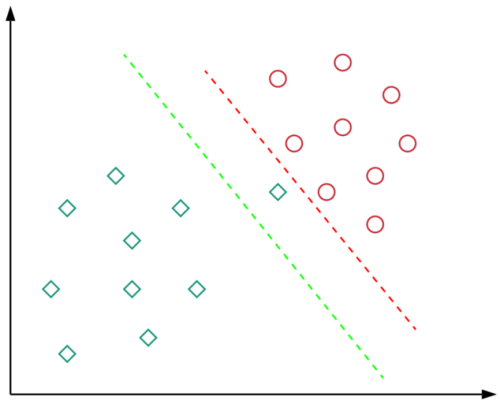
```
[[  0   0   0  94]
 [  0   0   0 102]
 [  0   0   0 120]
 [  0   0   0  84]]
Accuracy: 0.21
Precision: 0.0525
Recall: 0.25
```

همان طورکه مشخص است نتایج نسبت به حالت قبلی خود به شدت بدتر شده است و مورد مناسبی نمی‌باشد.

۴.۲ چهار

در Soft Margin به SVM اجازه داده می‌شود تا تعداد معینی از اشتباهات را مرتکب شود و حاشیه را تا حد ممکن گسترده نگه دارد تا نقاط دیگر همچنان به درستی طبقه

بندی شوند. این کار به سادگی با اصلاح هدف SVM قابل انجام است. برای مثال در شکل زیر با اینکه خط قرمز ، داده‌ها را به طور کامل تفکیک کرده است ولی خط سبز حاشیه وسیع تری دارد که اجازه می دهد تا به خوبی در داده های دیده نشده تعمیم یابد. از این نظر ، فرمولاسیون soft margin نیز در جلوگیری از مشکل overfitting کمک می کند.



اما در hard margin به این صورت نیست و تمام داده‌ها باید توسط خط جداکننده به طور کامل از هم تفکیک شوند.

برای اینکه مقدار margin را تعیین کنیم که به چه صورت باشد ، حالت soft باشد یا hard از پارامتر C برای تنظیم آن استفاده می‌کنیم. در ابتدا C را برابر 0.001 قرار می‌دهیم به صورت زیر:

```
clf = svm.SVC(kernel='linear',C=0.001)

clf.fit(x_train, y_train)

y_pred = clf.predict(x_test)
```

حال نتایج به شرح زیر است:

```
[[94  0  0  0]
 [73 26  2  1]
 [14 53  2 51]
 [ 0  0  0 84]]
Accuracy: 0.515
Precision: 0.491524499862187
Recall: 0.5678921568627451
F1: 0.44170590236710355
```

با $C = 0.01$ نتایج به صورت زیر است:

```
[[ 89   5   0   0]
 [  2  97   3   0]
 [  0  14 100   6]
 [  0   0   3  81]]
Accuracy: 0.9175
Precision: 0.9221648959368542
Recall: 0.9238519876035521
F1: 0.9210986480769794
```

با $C = 0.1$ نتایج به صورت زیر است:

```
[[ 90   4   0   0]
 [  2  98   2   0]
 [  0   8 103   9]
 [  0   0   1  83]]
Accuracy: 0.935
Precision: 0.9357604966813335
Recall: 0.941164923416175
F1: 0.9367391200828107
```

با $C = 10$ نتایج به صورت زیر است:

```
[[ 93   1   0   0]
 [  2 100   0   0]
 [  0  15 100   5]
 [  0   0   2  82]]
Accuracy: 0.9375
Precision: 0.9409843066083058
Recall: 0.9448194171285536
F1: 0.9403808512934538
```

با $C = 50$ نتایج به صورت زیر است:

```
[[ 91   3   0   0]
 [  1 100   1   0]
 [  0   9 106   5]
 [  0   0   1  83]]
Accuracy: 0.95
Precision: 0.9516627193257629
Recall: 0.9549764586685738
F1: 0.9520037258453816
```

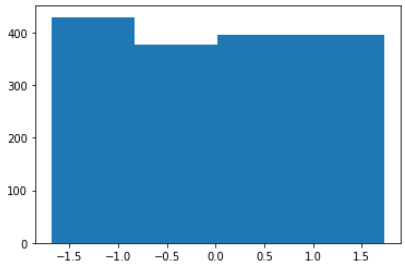
مقدار پیش فرض C برابر یک می‌باشد. همان طورکه مشاهده شد،با افزایش مقدار C به نتایج بهتری رسیدیم و داده‌ها به طور بهتری از یکدیگر تفکیک شده‌اند و تعداد

misclassification کمتر می‌شود.یعنی هرچه C کمتر باشد به سمت soft margin می‌رویم و هرچه C بزرگتر باشد به سمت hard margin می‌رویم.

۵.۲ پنج

- binning

بر روی فیچر battery power عملیات binning را اعمال می‌کنیم. به ۴ بخش دیتاهای موجود را تقسیم بندی می‌کنیم.توزیع داده‌ها در هر بخش به شکل زیر است:



با استفاده از تابع زیر دیتاها را به ۴ دسته که نام‌هایشان نیز در شکل مشخص است با اندازه یکسان بین‌ها تقسیم می‌کنیم:

```
def make_bins(df):
    label_names = ["low battery", "mid1 battery","mid2 battery","high battery" ]
    cut_points = [-2, -0.8269101, 0.02499725, 0.8769045, 1.72881194]
    df["battery_power_effeciency"] = pd.cut(df["battery_power"], cut_points, labels=label_names)
    return df
```

و در نهایت دیتاها پس از one hot کردن به صورت زیر می‌شود:

low battery	mid1 battery	mid2 battery	high battery
0	0	0	1
0	1	0	0
0	1	0	0
0	0	0	1
1	0	0	0

حال اگر سایز بین ها مساوی نباشد به صورت زیر عمل کردیم:

```
def make_bins2(df):
    label_names = ["low battery", "mid1 battery","mid2 battery","high battery" ]
    cut_points = [-2, -1, 0, 0.75, 1.72881194]
    df["battery_power_effeciency"] = pd.cut(df["battery_power"], cut_points, labels=label_names)
    return df
```

- one hot encoding

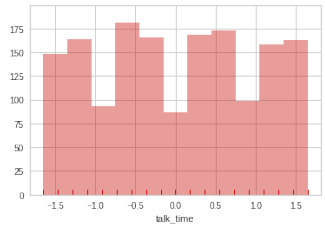
با استفاده از کد زیر ،بر فیجرهای کتگوریکال که در شکل نیز مشخص است ، one hot encoding اعمال می‌کنیم.

```
categorical = ['bluetooth', 'dual_sim', 'four_g','three_g','touch_screen', 'wifi']
new_x_train = x_train
df = pd.get_dummies(new_x_train, columns = categorical)
dummies = pd.get_dummies(new_x_train, columns = categorical)
new_x_train = pd.concat([new_x_train, dummies], axis=1)
```

- transform

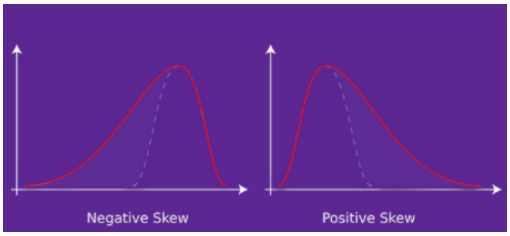
برخی از مدل‌های یادگیری ماشین ، مانند رگرسیون خطی و لجستیکی ، فرض می کنند که متغیرها از یک توزیع نرمال پیروی می کنند. معمولا ، متغیرها در مجموعه داده‌های واقعی بیشتر توزیع کج [۷] را دنبال می کنند. با اعمال تعدادی تبدیل در این متغیرها و map کردن توزیع کج آنها بر توزیع نرمال ، می توانیم عملکرد مدل های خود را افزایش دهیم.

ابتدا باید ببینیم که آیا متغیرهای ما از توزیع نرمال پیروی می کنند یا خیر. ما می توانیم نرمال بودن را با plot های histograms و Q-Q تخمین بزنیم. به طور مثال توزیع فیچر $talktime$ را در زیر مشاهده می‌کنید که نرمال نمی‌باشد.



[7] skewed distribution

Logarithmic transformation ساده ترین و محبوب ترین روش در میان انواع مختلف تبدیل است و شامل یک تحول اساسی است که به طور قابل توجهی بر شکل توزیع تأثیر می گذارد. ما می توانیم از آن (لگاریتم بر پایه ۱۰ یاln) استفاده کنیم تا توزیع های بسیار کج ، بخصوص برای توزیع های کج راست ، کمتر کج شوند. باید توجه داشته باشید که این تابع فقط برای اعداد مثبت تعریف شده است. ولی در اینجا چون بر دیتاها StandardScaler در مرحله preprocces اعمال شده است و مقدار منفی نیز دارد پس از این روش نمی‌توان استفاده کرد. و همچنین زمانی از تبدیلات استفاده می‌کنیم که plot آن‌ها شبیه توزیع نرمال باشد با این تفاوت که به سمت چپ یا راست خم شده‌اند و از حالت نرمال خارج شده‌اند ولی به طور مثال در شکل قبل توزیع به صورت پراکنده است .



- new feature

طول و عرض گوشی را در هم ضرب کرده و یک فیچر جدید به نام area می‌سازیم.به صورت زیر:

```
x_train["area"] = x_train["sc_h"] * x_train["sc_w"]
```

۶.۲ شش

- bining

در این بخش دیتاهایی که binning رویشان اعمال شده است را SVM بر آن‌ها اجرا می‌کنیم که نتایج آن به شرح زیر است:

```
[[ 90   4   0   0]
 [  0 100   2   0]
 [  0   6 108   6]
 [  0   0   0  84]]
Accuracy: 0.955
Precision: 0.956060606060606
Recall: 0.9594597413433459
F1: 0.9565761930355577
```

همان طورکه واضح است اعمال binning باعث بهبود نتایج شده است و خروجی بسیار خوبی دریافت شده است پس binning تاثیر مثبتی در SVM داشته است.

در حالتی که اندازه بین‌ها باهم برابر نباشد نتایج به صورت زیر است:

```
[[ 90   4   0   0]
 [  4  97   1   0]
 [  0   9 106   5]
 [  0   0   2  82]]
Accuracy: 0.9375
Precision: 0.9385676975452969
Recall: 0.9419877525478276
F1: 0.9393424169395994
```

با توجه به نتایج مقدار کمی بهتر از حالت بدون binning است ولی با این حال مقدار کمی بهبود نسبی پیدا کرده است.

- one hot encoding

در این بخش نیز SVM را بر روی دیتاهایی که فیچرهای کتگوریکال آن hot one شده است اعمال می‌کنیم که نتایج آن به صورت زیر است و نتایج آن تفاوت چندانی با حالت قبل‌اش یعنی بدون تغییر فیجرهای کتگوریکال ندارد.

```
[[ 91   3   0   0]
 [  2  98   2   0]
 [  0  10 102   8]
 [  0   0   2  82]]
Accuracy: 0.9325
Precision: 0.933688192148326
Recall: 0.9387649740747364
F1: 0.9346583571004031
```

- transform

- new feature

در این بخش SVM را روی دیتاستی که فیچر area به آن اضافه شده‌است ، اعمال می‌کنیم. نتایج به شرح زیر است که تاثیر قابل توجهی بر نتایج در جهت بهبود نداشته است:

```
[[ 91   3   0   0]
 [  2  95   5   0]
 [  0   6 106   8]
 [  0   0   1  83]]
Accuracy: 0.9375
Precision: 0.9376181614084839
Recall: 0.9427215567077896
F1: 0.9394891652981612
```

- All

در این بخش تمام حالت‌هایی را که در سوال ۵ ذکر شده است را با هم اعمال می‌کنیم و SVM را می‌سازیم . نتایج آن به صورت زیر می‌باشد:

```
[[ 86   8   0   0]
 [  2  94   6   0]
 [  0   8 105   7]
 [  0   0   0  84]]
Accuracy: 0.9225
Precision: 0.9252102627102627
Recall: 0.9278655611180642
F1: 0.9252345767440107
```

همان طور که واضح است ، با اعمال تمامی تغییرات تاثیر قابل توجهی بر نتایج نداشته است ، حتی به طور جزئی افت در نتایج مشاهده می‌شود.

۷.۲ هفت

درخت تصمیم یک مدل پیشبینی کننده است به طوری که می‌تواند برای هر دو مدل رگرسیون و طبقه‌ای مورد استفاده قرار گیرد. زمانی که درخت برای کارهای طبقه‌بندی استفاده می‌شود،به عنوان درخت طبقه‌بندی [۸] شناخته می‌شود و هنگامی که برای

[8] Classification Tree

فعالیت‌های رگرسیونی به کار می‌رود درخت رگرسیون [9] نامیده میشود.در ساختار درخت تصمیم، پیشبینی به دست آمده از درخت در قالب یک سری قواعد توضیح داده می‌شود. هر مسیر از ریشه تا یک برگ درخت تصمیم، یک قانون را بیان می‌کند و در نهایت برگ با کلاسی که بیشترین مقدار رکورد در آن تعلق گرفته برچسب می‌خورد.

$ID3$: یکی از الگوریتم‌های بسیار ساده درخت تصمیم که در سال ۱۹۸۶ توسط Quinlan مطرح شده است. اطلاعات به دست آمده به عنوان معیار تفکیک به کار می‌رود. این الگوریتم هیچ فرایند هرس کردن را به کار نمی‌برد و مقادیر اسمی و مفقوده را مورد توجه قرار نمی‌دهد.

$C4.5$: این الگوریتم درخت تصمیم، تکامل یافته ID۳ است که در سال ۱۹۹۳ توسط Quinlan مطرح شده است. Ratio Gain به عنوان معیار تفکیک در نظر گرفته می‌شود. عمل تفکیک زمانی که تمامی نمونه‌ها پایین آستانه مشخصی واقع می‌شوند، متوقف می‌شود. پس از فاز رشد درخت عمل هرس کردن بر اساس خطا اعمال می‌شود. این الگوریتم مشخصه‌های اسمی را نیز در نظر می‌گیرد

$CART$: برای برقراری درخت‌های رگرسیون و دسته‌بندی از این الگوریتم استفاده می‌شود. در سال ۱۹۸۴توسط Breiman و همکارانش ارائه شده است. نکته حائز اهمیت این است که این الگوریتم درخت‌های باینری ایجاد می‌کند به طوری که از هر گره داخلی دو لبه از آن خارج می‌شود و درخت‌های بدست آمده توسط روش اثربخشی هزینه، هرس می‌شوند. یکی از ویژگی‌های این الگوریتم، توانایی در تولید درخت‌های رگرسیون است. در این نوع از درخت‌ها برگ‌ها به جای کلاس مقدار واقعی را پیشبینی می‌کنند. الگوریتم برای تفکیک کننده‌ها، میزان مینیمم مربع خطا را جستجو می‌کند. در هر برگ، مقدار پیشبینی بر اساس میانگین خطای گره‌ها می‌باشد.

$CHID$: این الگوریتم درخت تصمیم به جهت در نظرگرفتن مشخصه‌های اسمی در سال ۱۹۸۱ توسط Kass طراحی شده است. الگوریتم برای هر مشخصه ورودی یک جفت مقدار که حداقل تفاوت را با مشخصه هدف داشته باشد، پیدا می‌کند.

[9](Regression Decision Tree

۸.۲ هشت

درخت تصمیم را به صورت زیر روی دیتاست با پارامترهای که در تصویر مشاهده می‌شود ، اعمال می‌کنیم:

```
clf_entropy = DecisionTreeClassifier(criterion="entropy",random_state=100,max_depth=3,min_samples_leaf=5)
clf_entropy.fit(x_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=100, splitter='best')
```

و با توجه به پارامترهای ذکرشده accuracy مدل برابر 74.0 می‌شود.

۹.۲ نه

در ابتدا برای درخت تصمیم سوال قبل ، پارامتر maxDepth را تغییر می‌دهیم. با عمق 6 به accuracy ی 84.5 و با عمق 9 به accuracy ی 87.0 دست پیدا می‌کنیم. پس واضح است با افزایش عمق ، دقت افزایش می‌یابد. ولی از یک جایی به بعد افزایش عمق دیگر باعث افزایش دقت نمی‌شود.پس در نهایت می‌توان نتیجه گرفت که با افزایش عمق ، دقت بالا می‌رود ولی از جایی به بعد دیگر تاثیری در دقت ندارد.

با تغییر minSamplesLeaf از 5 به 21 accuracy به 73.5 تغییر می‌کند پس تغییر minSamplesLeaf به مقدار قابل توجهی ، دقت را به مقدار کمی تغییر داده است. پس با افزایش مقدار minSamplesLeaf ، دقت کاهش می‌یابد.

۱۰.۲ ده

هرس روشی در یادگیری ماشین است که باعث کاهش اندازه درخت‌های تصمیم‌گیری با از بین بردن بخشهایی از درخت است که قدرت کمی برای طبقه‌بندی نمونه‌ها دارند. هرس باعث کاهش پیچیدگی طبقه‌بندی کنندهٔ نهایی می‌شود و از این رو باعث بهبود دقت پیش بینی و کاهش overfitting می‌شود.

اگر عمق درخت زیاد و از انشعاب های متعدد استفاده شود باعث پیچیدگی زمانی می‌شود . یک درخت کوچکتر با تعداد انشعاب های کمتر ممکن است مقدار کمی

بایاس داشته باشد، اما به شدت واریانس کمتری بر روی داده های تست خواهد داشت. بنابراین یکی از راهکارها، هرس کردن درخت است.

۱۱.۲ یازده

در این بخش هرس را روی درخت تصمیم اجرا می‌کنیم. در ابتدا درخت را بدون هرس تست می‌کنیم . نتایج accuracy به صورت زیر است.

```
Train score 1.0
Test score 0.8275
```

همان طور که مشاهده می‌کنید ، دقت train برابر یک می‌باشد یعنی مدل overfit شده‌است. حال برای مقابله با این موضوع از هرس به شکل زیر استفاده می‌کنیم:

```
params = {'max_depth': [2,4,6,8,10,12],
          'min_samples_split': [2,3,4],
          'min_samples_leaf': [1,2]}

clf = tree.DecisionTreeClassifier()
gcv = GridSearchCV(estimator=clf,param_grid=params)
gcv.fit(x_train,y_train)
model = gcv.best_estimator_
model.fit(x_train,y_train)
```

نتایج accuracy به صورت زیر است. همان طور که مشاهده می‌کنید در این حالت دیگر مدل overfit نشده است و در زمان تست مدل‌های دیگر می‌تواند بهتر از مدل قبل عمل کند و خروجی مطلوب نیز دریافت شده است.

```
Train score 0.926875
Test score 0.8275
```

۱۲.۲ دوازده

بر روی دیتاست ، random forest را به صورت زیر اجرا کردیم . همان طور که واضح است ، دقت برابر 0.8875 است که نسبت به حالتی که در سوال ۸ درخت تصمیم را روی

```
clf=RandomForestClassifier(n_estimators=100)
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8875
```

دیتاست اعمال کردیم و دقت 0.74 را گرفتیم ، بهبودی خیلی خوبی به دست آورده‌ایم و نشان می‌دهد که random forest بسیار بهتر از درخت تصمیم عمل می‌کند. از عواملی که می‌توان برای علت بهبود در random forest اشاره کرد این است که درrandom forest مشکل overfitting با ایجاد درخت از زیرمجموعه‌های تصادفی حل می‌شود در حالی که در درخت تصمیم با عمق زیاد با مشکل overfitting روبرو هستیم.

۱۳.۲ سیزده

از جمله مزایای درخت تصمیم می‌توان به موارد زیر اشاره کرد:

خواندن و تفسیر آسان است. یکی از مزایای درخت تصمیم این است که خواندن و تفسیر خروجی آنها آسان است ، حتی بدون اینکه به دانش آماری احتیاج داشته باشید. به عنوان مثال ، هنگام استفاده از درخت تصمیم برای ارائه اطلاعات دموگرافیک در مورد مشتریان ، کارکنان بخش بازاریابی می توانند نمایش گرافیکی داده ها را بدون نیاز به دانش آماری بخوانند و تفسیر کنند.

آماده سازی آسان. در مقایسه با سایر تکنیک های تصمیم گیری ، درخت تصمیم برای تهیه اطلاعات ، مراحل کمتری انجام می‌دهند. با این حال ، کاربران برای ایجاد متغیرهای جدید با قدرت پیش بینی متغیر هدف ، باید اطلاعات آماده داشته باشند. آنها همچنین می توانند طبقه بندی داده ها را بدون محاسبه محاسبات پیچیده ایجاد کنند. برای شرایط پیچیده ، کاربران می توانند درخت تصمیم را با روش های دیگر ترکیب کنند.

data cleaning کمتری مورد نیاز است. یکی دیگر از مزایای درخت تصمیم گیری این است که ، پس از ایجاد متغیرها ، data cleaning کمتری مورد نیاز است. مواردی که مقادیر از دست رفته و دور از انتظار هستند ، در داده های درخت تصمیم اهمیت کمتری دارند.

۱۴.۲ چهارده

الگوریتم Ripper یک الگوریتم طبقه بندی مبتنی بر Rule است. مجموعه ای از قوانین را از مجموعه آموزش استخراج می کند. الگوریتم استخراج قوانین ای است که به طور گسترده مورد استفاده قرار می گیرد.

موارد استفاده از الگوریتم Ripper : در مجموعه های داده با توزیع های نامتوازن کلاس به خوبی کار می کند. با مجموعه داده های نویزی به خوبی کار می کند زیرا از مجموعه اعتبار سنجی برای جلوگیری از overfitting مدل استفاده می کند.

حالت اول RIPPER : Training records فقط به دو کلاس تعلق دارد

در میان رکوردهای داده شده ، کلاس اکثریت را مشخص می کند (که بیشترین ظاهر را داشته است) و این کلاس را به عنوان کلاس پیش فرض می گیرد. به عنوان مثال: اگر ۱۰۰ رکورد وجود داشته باشد و ۸۰ ردیف متعلق به کلاس A و ۲۰ طبقه کلاس B باشد ، کلاس A کلاس پیش فرض خواهد بود. برای کلاس دیگر ، سعی در یادگیری / استخراج قوانین مختلف برای تشخیص آن کلاس است.

حالت دوم RIPPER : Training records به بیش از دو کلاس تعلق دارد (کلاسهای چندگانه)

تمام کلاسهای موجود را در نظر بگیرید و سپس آنها را بر اساس فراوانی آنها به ترتیب خاص مرتب کنید (مثلاً در حال افزایش است). کلاس با حداکثر frequency به عنوان کلاس پیش فرض در نظر گرفته می شود.

چگونه قاعده استخراج می شود:

در مرحله اول ، سعی می شود قوانینی برای آن دسته از سوابق مربوط به کلاس C۱ (کلاس با کمترین frequency)استخراج شود. سوابق متعلق به C۱ به عنوان مثالهای مثبت (+ve) و سایر طبقات به عنوان مثالهای منفی (-ve) در نظر گرفته می شوند. الگوریتم پوششی متوالی برای تولید قوانینی استفاده می شود که بین مثالهای مثبت و منفی تبعیض قائل می شوند.بعد ، در این اتصال Ripper سعی می کند قوانینی را برای C۲ استخراج کند که آن را از کلاسهای دیگر متمایز کند. این فرآیند تا رسیدن به معیارهای توقف تکرار می شود ، یعنی وقتی که ما با Cn (کلاس پیش فرض) باقی می مانیم. Ripper قوانینی را از کلاس اقلیت به کلاس اکثریت استخراج می کند.

۱۵.۲ پانزده

سری زمانی مجموعه ای از نقاط داده است که به ترتیب زمانی نمایه می شوند. یک سری زمانی می تواند به سه قسمت تقسیم شود: روند (جهت طولانی مدت) فصلی (حرکتهای مربوط به تقویم) و خطا (نوسانات غیر سیستماتیک). غالباً ویژگیهای خارجی دیگری نیز وجود دارد که بر روی سریهای زمانی تأثیر می گذارد. به عنوان مثال ، اگر به فروش لباس ها نگاهی بیندازیم ، اگر تبلیغاتی در حال انجام باشد ، می توانید جهش حجم را مشاهده کنیم. پیش بینی مقادیر آینده یک سری زمانی می تواند یکی از مهمترین مراحل برای یک تجارت باشد. به عنوان مثال ، پیش بینی دقیق تقاضا برای یک محصول می تواند به یک شرکت تجاری کمک کند تا زنجیره تأمین خود را بهتر برنامه ریزی کند.

همه برای تصمیم گیری در برخی موارد از درخت تصمیم استفاده کرده‌اند. یک ساختار درخت مانند به این صورت است که در هر گره ای تصمیم می گیریم و انجام آن را تا رسیدن به نتیجه ادامه می دهیم.

random forest مجموعه ای از الگوریتم های درخت تصمیم گیری است که به طور گسترده برای طبقه بندی و رگرسیون مسائل مدل سازی پیش بینی با ساختار داده (جدول) مجموعه داده ها استفاده می شود ، به عنوان مثال. داده ها همانطور که در صفحه گسترده یا جدول پایگاه داده به نظر می رسند. از Random Forest می توان برای پیش بینی سری های زمانی نیز استفاده کرد ، اگرچه این امر مستلزم آن است که ابتدا مجموعه داده های سری زمانی به یک مسئله یادگیری تحت نظارت تبدیل شود. همچنین به استفاده از یک تکنیک ویژه برای ارزیابی مدل موسوم walk-forward validation نیاز دارد ، زیرا ارزیابی مدل با استفاده از k-fold cross validation می تواند منجر به نتایج مغرضانه شود.

۱۶.۲ شانزده

ستون Date داده‌های قیمت بیت‌کویین را به صورت زیر تغییر می‌دهیم:

```
def mdy_to_ymd(d):
    return datetime.strptime(d, '%b %d, %Y').strftime('%Y-%m-%d')
```

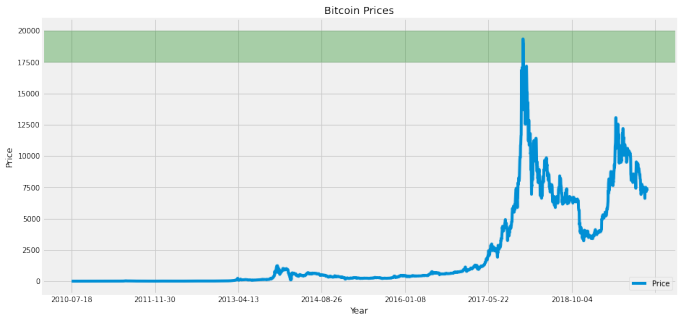
```
df['newDate'] = df['Date'].apply(mdy_to_ymd)
dfT['newDate'] = dfT['Date'].apply(mdy_to_ymd)
```

```
df = df.drop('Date', axis=1)
dfT = dfT.drop('Date', axis=1)
```

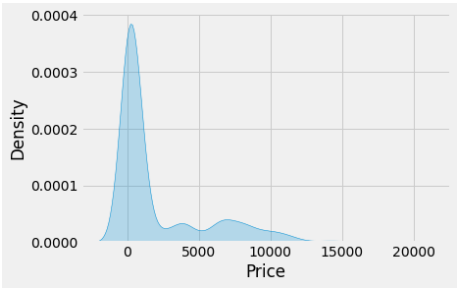
۱۷.۲ هفده

در ابتدا پیش پردازش‌های لازم را روی دیتاست انجام می‌دهیم.ازجمله تغییر فرمت تاریخ، تغییر دیتاتایپ وتبدیل به float، حذف کردن ضریب K ، M در اعداد و ضرب آن اعداد در به ترتیب ۱۰۰۰ و ۱۰۰۰۰۰۰. همچنین index ها را با توجه به تاریخ مرتب می‌کنیم.

می‌توان میزان قیمت را در بازه‌های زمانی دیتای train تصوبرسازی کنیم که به صورت زیر می‌باشد:



می‌توان داده ها با نمودارهای Density برای دیدن محل قرارگیری جرم داده ها خلاصه کرد که نمودار آن به صورت زیر می‌باشد:

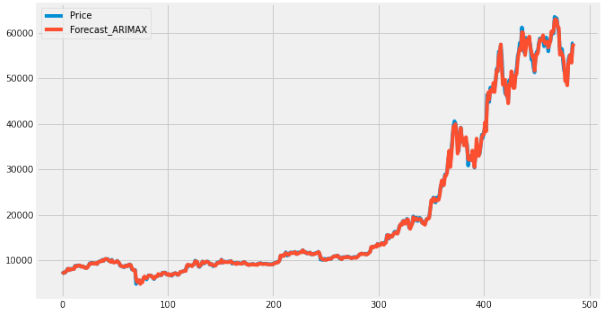


- ARIMA

مدل ARIMA را به صورت زیر روی دیتاست اعمال می‌کنیم که df مجموعه داده train و dfT مجموعه داده test می‌باشدexogenousFeatures فیچرهای استفاده شده در مدل می‌باشد:

```
model = pm.auto_arima(df.Price, exogenous=df[exogenous_features], trace=True, error_action="ignore", suppress_warnings=True)
model.fit(df.Price, exogenous=df[exogenous_features])
forecast = model.predict(n_periods=len(dfT), exogenous=dfT[exogenous_features])
dfT["Forecast_ARIMAX"] = forecast
```

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
print("RMSE of Auto ARIMAX:", np.sqrt(mean_squared_error(dfT.Price, dfT.Forecast_ARIMAX)))

print("\nMAE of Auto ARIMAX:", mean_absolute_error(dfT.Price, dfT.Forecast_ARIMAX))
```

```
RMSE of Auto ARIMAX: 487.49448654455387

MAE of Auto ARIMAX: 255.2042191030597
```

و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم نتیجه به شرح زیر می‌باشد:

```
temp['upper_range'] = temp['test'] * 1.05
temp['lower_range'] = temp['test'] * 0.95

temp[(temp['upper_range'] >=temp['pred']) & (temp['pred'] >= temp['lower_range'])].shape[0] * 100/temp.shape[0]
```

```
97.94238683127573
```

همان طور که واضح است نتایج مقادیر مناسبی دارند.

- XGBoost

مدل XGBoost را به صورت زیر روی دیتاست اعمال می‌کنیم:

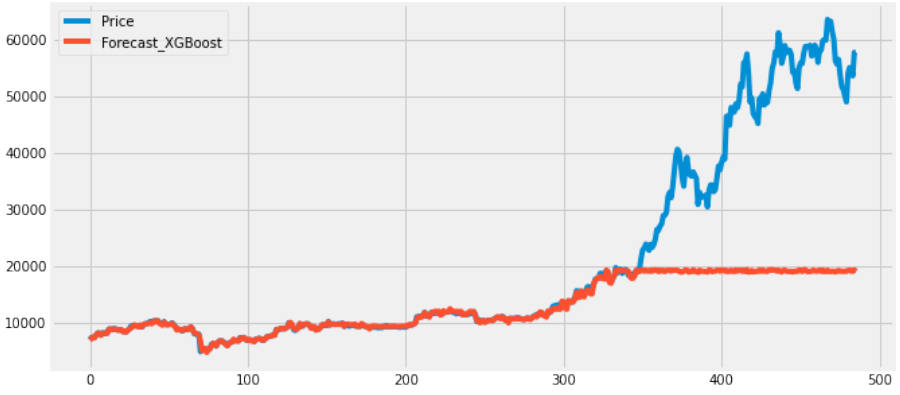
```
reg = xgb.XGBRegressor()
```

```
params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
 "max_depth"        : [1, 3, 4, 5, 6, 7],
 "n_estimators"     : [int(x) for x in np.linspace(start=100, stop=2000, num=10)]
 "min_child_weight" : [int(x) for x in np.arange(3, 15, 1)],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
 "subsample"        : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
 "colsample_bytree" : [0.5, 0.6, 0.7, 0.8, 0.9, 1],
 "colsample_bylevel": [0.5, 0.6, 0.7, 0.8, 0.9, 1],

}
```

```
model  = RandomizedSearchCV(
                reg,
                param_distributions=params,
                n_iter=10,
                n_jobs=-1,
                cv=5,
                verbose=3,
)
```

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
test MAE : 7570.41918020994
test RMSE : 15411.236181561702
test R2 : 0.1987453922554392
```

و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم ، برابر 69.341563786 درصد خواهد بود. همان طور که واضح است ، نتایج حاصل از این مدل مقدار مناسبی نمی‌باشد.

- FaceBook Prophet

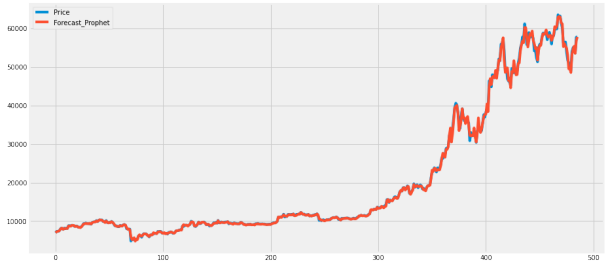
مدل FaceBook Prophet را به صورت زیر روی دیتاست اعمال می‌کنیم:

```
model_fbp = Prophet()
for feature in exogenous_features:
    model_fbp.add_regressor(feature)

model_fbp.fit(df[["newDate", "Price"] + exogenous_features].rename(columns={"newDate": "ds", "Price": "y"}))

forecast = model_fbp.predict(dfT[["newDate", "Price"] + exogenous_features].rename(columns={"newDate": "ds"}))
```

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
Prophet's MAE  : 253.7444872870993
Prophet's RMSE : 486.6399007321618
```

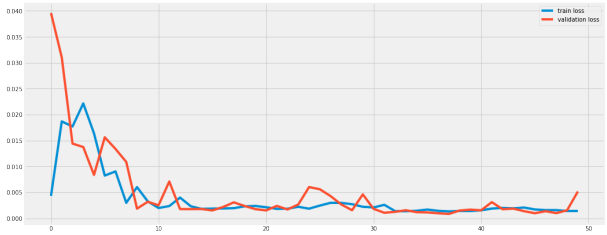
و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم ، برابر 97.94238683درصد خواهد بود.

همان طور که واضح است نتایج حاصل از این مدل نیز همانند مدل اول به میزان خوبی قابل قبول است.

- LSTM

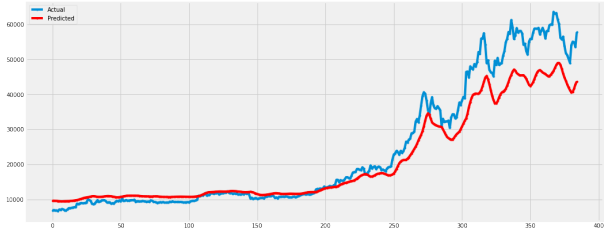
مدل LSTM را به صورت زیر روی دیتاست اعمال می‌کنیم:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100, 50)           10400
_________________________________________________________________
dropout (Dropout)            (None, 100, 50)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
dropout_1 (Dropout)          (None, 100, 50)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
dropout_2 (Dropout)          (None, 100, 50)           0
_________________________________________________________________
lstm_3 (LSTM)                (None, 50)                20200
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 71,051
Trainable params: 71,051
Non-trainable params: 0
```



میزان loss این شبکه در حالت train و Test به صورت شکل بالا می‌باشد.

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
Test RMSE: 0.09822954645186623
Test MAE: 0.25635289238335535
```

و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم ، برابر 14.805194 درصد خواهد بود.

همان طورکه مشاهده می‌کنید روند پیش‌بینی قیمت تقریبا درست است ولی دقت تخمین آن اصلا مناسب نمی‌باشد.

- Gradient Boosting Regressor

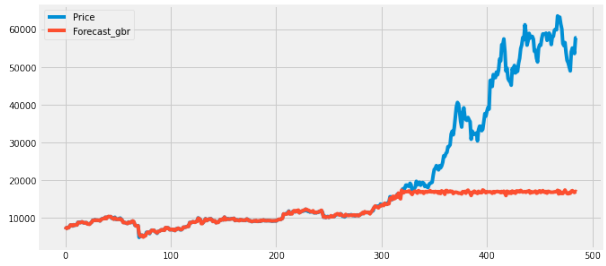
مدل Gradient Boosting Regressor را به صورت زیر روی دیتاست اعمال می‌کنیم:

```
gbr = GradientBoostingRegressor(n_estimators=6000,learning_rate=0.01,
                                max_depth=4,max_features='sqrt',
                                min_samples_leaf=15,
                                min_samples_split=10,
                                loss='huber',random_state=42)
```

```
gbr.fit(X_train, y_train)
```

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.01, loss='huber',
                          max_depth=4, max_features='sqrt', max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=15, min_samples_split=10,
                          min_weight_fraction_leaf=0.0, n_estimators=6000,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=42, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
test MAE : 8257.483634128172
test RMSE : 16518.67765050611
test R2 : 0.0794526241689617
```

و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم ، برابر 65.843621درصد خواهد بود.

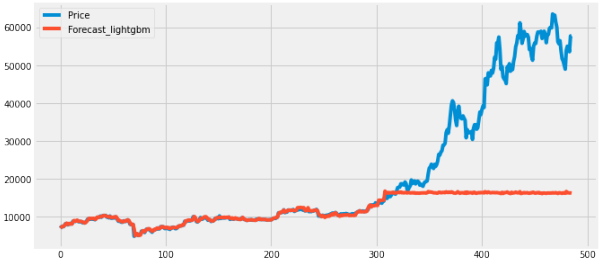
- LGBMRegressor

مدل LGBMRegressor را به صورت زیر روی دیتاست اعمال می‌کنیم:

```
lightgbm = LGBMRegressor(objective='regression',
                         num_leaves=6,
                         learning_rate=0.01,
                         n_estimators=7000,
                         max_bin=200,
                         bagging_fraction=0.8,
                         bagging_freq=4,
                         bagging_seed=8,
                         feature_fraction=0.2,
                         feature_fraction_seed=8,
                         min_sum_hessian_in_leaf = 11,
                         verbose=-1,
                         random_state=42)
```

```
lightgbm.fit(X_train, y_train)
```

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
test MAE : 8505.200412552247
test RMSE : 16787.063764372262
test R2 : 0.04929655467421046
```

و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم ، برابر 62.345679درصد خواهد بود.

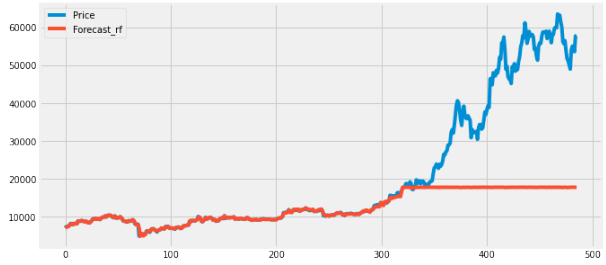
- Random Forest Regressor

مدل Random Forest Regressor را به صورت زیر روی دیتاست اعمال می‌کنیم:

```
rf = RandomForestRegressor(n_estimators=1200,
                           max_depth=15,
                           min_samples_split=5,
                           min_samples_leaf=5,
                           max_features=None,
                           oob_score=True,
                           random_state=42)
```

```
rf.fit(X_train, y_train)
```

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
test MAE : 7981.102444786862
test RMSE : 16089.58932901156
test R2 : 0.12665566754054935
```

و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم ، برابر 67.48971درصد خواهد بود.

- Ridge Regressor

مدل Ridge Regressor را به صورت زیر روی دیتاست اعمال می‌کنیم:

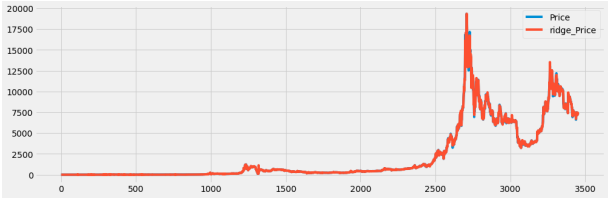
```
kf = KFold(n_splits=12, random_state=42, shuffle=True)

ridge_alphas = [1e-15, 1e-10, 1e-8, 9e-4, 7e-4, 5e-4, 3e-4, 1e-4, 1e-3, 5e-2, 1e-2, 0.1, 0.3, 1, 3, 5, 10, 15, 18, 20, 30, 50, 75, 100]

ridge = make_pipeline(RobustScaler(), RidgeCV(alphas=ridge_alphas, cv=kf))

ridge.fit(X_train, y_train)
```

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
test MAE : 254.6629090613454
test RMSE : 487.20125816024023
test R2 : 0.9991992215264411
```

و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم ، برابر 97.942386 درصد خواهد بود.

همان طور که مشاهده می‌شود ، نتایج حاصل از این مدل بسیار خوب و قابل قبول می‌باشد.

۱۸.۲ هجده

- voting

روی سه مدل RandomForestRegressor و GradientBoostingRegres-sor و XGBRegressor ،عملیات voting را اعمال می‌کنیم که به صورت زیر می‌باشد:

```
votingreg = VotingRegressor([('gbr',gbr),('rf',rf), ('xgboost', xgboost)])

test_pred = votingreg.fit(x_train, y_train).predict(x_test)
```

و محک های زیر حاصل می‌شود و میزان دقت را با ۵ درصد خطا برابر 67.078189 است. همان طور که مشاهده می‌شود حاصل جواب‌های به دست آمده میانگینی از جواب‌های هریک از مدل‌ها می‌باشد.

```
MAE: 8048.743442341143
MSE: 262677822.12227932
RMSE: 16207.338526799498
r2_score: 0.11382601903783651
```

- bagging

تکنیک bagging را روی XGBRegressor به صورت زیر اعمال می‌کنیم :

```
1 regr = BaggingRegressor(base_estimator=XGBRegressor(), n_estimators=10, random_state=0).fit(x_train, y_train)

1 test_pred = regr.predict(x_test)
```

و نتایج زیر حاصل می‌شود و میزان دقت را با ۵ درصد خطا برابر 70.164609 است.

```
MAE: 7608.423234752765
MSE: 240737200.88964447
RMSE: 15515.70819813406
r2_score: 0.18784523963825683
```

همان طور که مشاهده می‌کنید نسبت به حالتی که bagging را اعمال نکردیم ، نتایج کمی بهتر شده است.

همین روند را برای RandomForest هم اعمال می‌کنیم و نتایج زیر حاصل می‌شود میزان دقت را با ۵ درصد خطا برابر 70.781893 است.

```
MAE: 7693.062752057614
MSE: 245294132.12009287
RMSE: 15661.868730138587
r2_score: 0.17247190565508796
```

در این حالت نیز نتایج نسبت به حالتی که bagging را اعمال نکردیم ، نتایج بهتر شده است. و این پیشرفت نسبت به نمونه قبل در این مدل بیشتر بوده است.

برای مدل GradientBoosting نتایج به شرح زیر است و میزان دقت را با ۵ درصد خطا برابر 71.39917 است.

```
MAE: 7586.439196338683
MSE: 240380671.2748307
RMSE: 15504.214629410633
r2_score: 0.1890480334848701
```

در این حالت نیز نتایج نسبت به حالتی که bagging را اعمال نکردیم ،بهتر شده است. و این پیشرفت نسبت به دو نمونه قبل بسیار بیشتر بوده است.

- boosting

با اعمال boosting روی دیتاست ، نتایج زیر حاصل می‌شود و میزان دقت را با ۵ درصد خطا برابر 68.31275 است.

```
est = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1,max_depth=10, random_state=0, loss='ls').fit(x_train, y_train)
```

```
test_pred = est.predict(x_test)
```

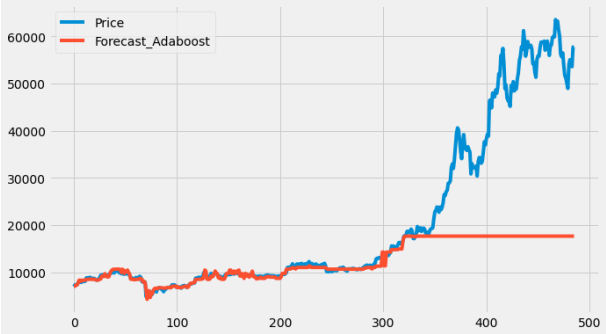
```
MAE: 7569.879954363816
MSE: 237604492.93382037
RMSE: 15414.42483305233
r2_score: 0.1984137918592821
```

۱۹.۲ نوزده

الگوریتم AdaBoost را به صورت زیر اجرا و روی دیتاست اعمال می‌کنیم:

```
from sklearn.ensemble import AdaBoostRegressor
ada=AdaBoostRegressor(n_estimators=50, learning_rate=0.2,loss='exponential').fit(x_train, y_train)
pred=ada.predict(x_test)
adab=ada.score(x_test,y_test)
predict = ada.predict(x_test)
```

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
test MAE : 8183.429362780755
test RMSE : 16122.868929872582
test R2 : 0.12303909179501626
```

و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم ، برابر 51.4403292 درصد خواهد بود.
اگر مقدار پارامترهای آن را تغییر دهیم و برابر مقادیر زیر قرار دهیم:

```
(n_estimators=100, learning_rate=0.2,loss='square'
```

نتایج به صورت زیر خواهد بود:

```
test MAE : 7721.196891317671
test RMSE : 15543.877154959839
test R2 : 0.18489360902942065
```

و میزان دقت را با ۵ درصد خطا ، برابر 66.04938271 درصد خواهد بود.
همان طور که مشاهده می‌کنید با تغییر پارامتر ، نتایج به مقدار قابل توجهی بهبود پیدا کرد.

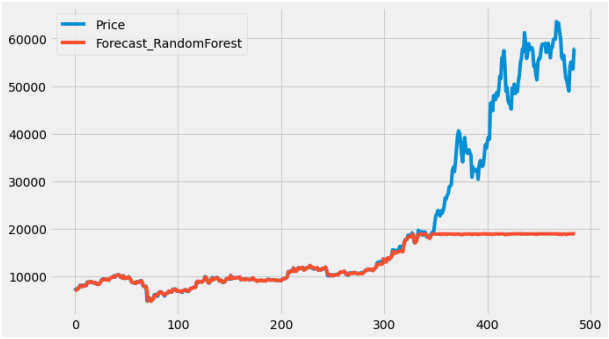
۲۰.۲ بیست

الگوریتم RandomForest را به صورت زیر اجرا و روی دیتاست اعمال می‌کنیم:

```
1 from sklearn.ensemble import RandomForestRegressor
2 rf_model = RandomForestRegressor(random_state=1)
```

```
1 rf_model.fit(x_train, y_train)
```

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
test MAE : 7594.654465020577
test RMSE : 15499.614641062464
test R2 : 0.1895291692662311
```

و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم ، برابر 70.5761316 درصد خواهد بود.
اگر مقدار پارامترهای آن را تغییر دهیم و برابر مقادیر زیر قرار دهیم:

```
random_state=1,max_depth=20,max_features="sqrt"
```

نتایج به صورت زیر خواهد بود:
و میزان دقت را با ۵ درصد خطا ، برابر 70.9876543 درصد خواهد بود.

```
test MAE : 7672.565469135803
test RMSE : 15644.073241696864
test R2 : 0.17435136214615843
```

همان طور که مشاهده می‌کنید با تغییر پارامتر تفاوت چندانی در میزان خروجی حاصل نشده است.

۲۱.۲ بیست و دو

در ابتدا یک ستون جدید به نام target ایجاد می‌کنیم که دو حالت صفر و یک را می‌گیرد ، صفر برای زمانی که کاهش قیمت داشته‌ایم و یک برای زمانی که افزایش قیمت. که این ستون را به عنوان ستون هدف درنظر می‌گیریم. و به صورت زیر اعمال می‌شود :

```
def target_column(x):
    if x > 0:
        return 1
    else:
       return 0
```

```
df['target'] = df['Change'].apply(target_column)
```

سپس بر روی مدل LSTM زیر دیتاست را اعمال می‌کنیم:

```
LSTM_model = Sequential()

LSTM_model.add(LSTM(128, input_shape=(x_train.shape[1],1),return_sequences=True))
LSTM_model.add(Dropout(0.2))
LSTM_model.add(BatchNormalization())

LSTM_model.add(LSTM(128,return_sequences=True))
LSTM_model.add(Dropout(0.1))
LSTM_model.add(BatchNormalization())

LSTM_model.add(LSTM(128))
LSTM_model.add(Dropout(0.2))
LSTM_model.add(BatchNormalization())

LSTM_model.add(Dense(32, activation='relu'))
LSTM_model.add(Dropout(0.2))

LSTM_model.add(Dense(2, activation='softmax'))
```

و نتایج آن به صورت زیر می‌باشد:

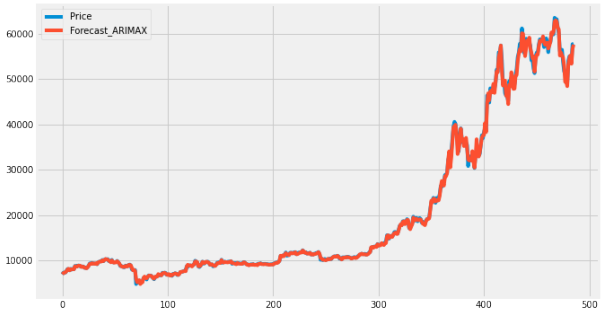
```
Test loss: 0.011201261542737484
Test accuracy: 0.9938271641731262
```

همان طور که از نتایج مشخص است مدل به خوبی کار می‌کند و دقت بالایی دارد.

۲۲.۲ بیست و سه

فیچری که در سوال قبل ایجاد شد را به عنوان ورودی در نظر می‌گیریم و قیمت را با مدل ARIMA تخمین می‌زنیم. نتایج به شرح زیر است:

اگر مقدار قیمت پیش‌بینی شده را برای داده تست plot کنیم به صورت شکل زیر خواهدشد:



و میزان معیار RMSE برابر زیر است:

```
RMSE of Auto ARIMAX: 488.5439920809565

MAE of Auto ARIMAX: 258.1158347133519
```

و اگر بخواهیم میزان دقت را با ۵ درصد خطا محاسبه کنیم برابر 98.3539 است. که اضافه کردن فیچر target باعث بهبود نتایج شده است.

۲۳.۲ بیست و شش

بعد از اینکه دیتاها را لود کردیم، ستون‌هایی که نامشان دارای feature است را انتخاب می‌کنیم . به صورت زیر:

```
training_data = pd.read_csv("/content/files/training_data.csv").set_index("id")
tournament_data = pd.read_csv('/content/files/tournament_data.csv').set_index ('id')

feature_names = [ f for f in training_data.columns if "feature" in f ]
```

سپس مدل XGBRegressor را روی دیتاست اعمال می‌کنیم:

```
model = XGBRegressor ( max_depth =5 , learning_rate= 0.01 , n_estimators=2000, colsample_bytree = 0.1 )
model.fit(training_data[feature_names],training_data["target"])

predictions = model.predict(tournament_data[feature_names])
```

که تخمین‌های به دست آمده برابر زیر می‌باشد و میزان دقت را با ۵ درصد خطا محاسبه کنیم برابر 28.177 است.

```
array([0.46547246, 0.4772104 , 0.55718875, ..., 0.45788038, 0.5258958 ,
       0.482519  ], dtype=float32)
```

اگر مدل GradientBoostingRegressor را روی دیتاست اعمال کنیم تخمین‌های زیر به دست می‌آید میزان دقت را با ۵ درصد خطا محاسبه کنیم برابر 28.093 است.:

```
array([0.45725866, 0.4468723 , 0.55534376, ..., 0.46394901, 0.51614985,
       0.47250196])
```

اگر مدل LGBMRegressor را روی دیتاست اعمال کنیم تخمین‌های زیر به دست می‌آید میزان دقت را با ۵ درصد خطا محاسبه کنیم برابر 28.15 است.:

```
array([0.47464881, 0.46274834, 0.56126726, ..., 0.471091  , 0.51286592,
       0.47264152])
```