

گزارش تمرین سوم

علی محمد بیگی

شماره دانشجویی: 99422039

لینک های کد تمرین: برای جلوگیری از به هم ریختگی کدها و بزرگ شدن فایل colab کدها را به دو دسته تقسیم کردیم. Colab اول مربوط به کدهای سوالات کدینگ ۱ تا ۲۱ میباشد و colab دوم مربوط به کدهای سوالات کدینگ ۲۲ تا ۲۷ میباشد.

- Colab for Q1-Q21:

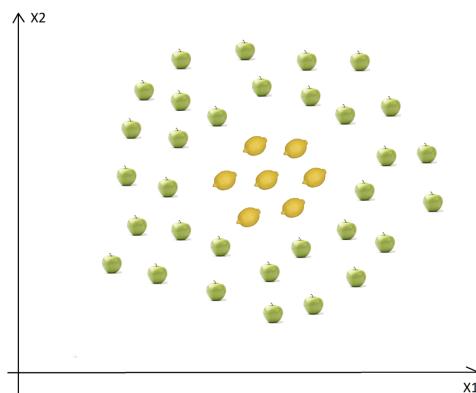
https://colab.research.google.com/drive/190aRzS0ULWFs_leintfBwr8DdVSsWkaO?usp=sharing

- Colab for Q22-Q27:

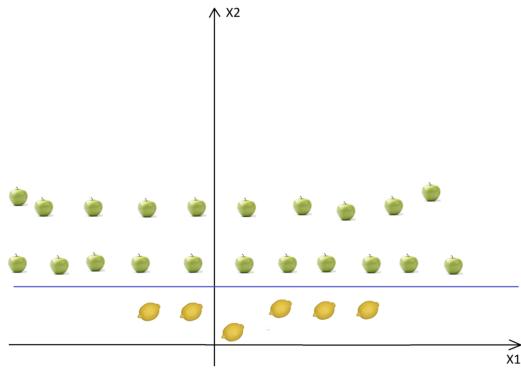
https://colab.research.google.com/drive/1CWlbIOJabOWeNYUa4OD7Jf_-JUKk4zn2?usp=sharing

سوال ۱

به طور کلی زمانی که نتوان داده ها را توسط یک خط (linearly separable) از هم جدا کرد نیاز به استفاده از کرنل بوجود می آید. مثل زیر این مساله را در دو بعد نشان می دهد. همانطور که دیده می شود نمی توان سیب و لیمو را با یک خط از هم جدا کرد. در چنین موقعیتی از کرنل استفاده می کنیم.



ایده کلی این است که اگر نمی توان در بعد فعلی (در مثال بالا ۲ بعد) کلاس ها را از هم جدا کرد یک بعد به مساله اضافه کرده (در مثال بالا ۳ بعد) و سپس مساله را در بعد جدید حل می کنیم. به عنوان نمونه ممکن است در مثال بالا در ۳ بعد سیب ها در یک سطح بالاتر از لیموها قرار گرفته و به راحتی و توسط یک صفحه (hyperplane) قابل تفکیک باشند. البته این کار به راحتی اضافه کردن یک بعد نیست. بلکه نیاز به تبدیل فضا (transform) می باشد. یک نمونه برای تغییر بعد برای مثال بالا مانند مثال زیر بوده که در آن میوه ها به راحتی از هم قابل تفکیک هستند.



این تغییر فضا کرنل نامیده می شود. کرنل ها انواع مختلفی دارند که می توان به Radial Basis Function (RBF), Laplace RBF Kernel, Sigmoid Kernel, Anova RBF Kernel نکته ای مهم در اینجا این است که همیشه نمی توان نظر قطعی و کلی برای استفاده از کرنل داد. شاید بتوانیم از کرنل خطی برای جدا کردن داده های خطی و از کرنل RBF برای تفکیک داده های غیرخطی استفاده کنیم اما به طور کلی برای استفاده از کرنل های پیچیده تر نیاز به بررسی بیشتر مساله داریم. باید توجه کرد که پیچیدگی کرنل RBF به علت داشتن پارامتر های بیشتر و این که باید ماتریس کرنل را حفظ کرد همیشه از کرنل خطی (نتها support vector ها کافی هستند) بیشتر بوده و بنابراین باید نهایت دقت را در انتخاب کرنل مورد نظر کرد. به طور کلی و بر اساس نظریه Occam's Razor پیشنهاد شده است که همیشه از ساده ترین کرنل یعنی کرنل خطی شروع کرده و تنها در صورت نیاز به کرنل های پیچیده تر پناه برد.

سوال ۲

برای این سوال دیتابست موبایل را از کگل دانلود کردیم و قبل از هرگونه عملیاتی آن را پیش پردازش کردیم. برای پیش پردازش:

- ستون هایی را که بیشتر از ۵۰ درصد مقدار null داشتند حذف کردیم
- اگر مقدار داده ای برای یک فیچر مشخص numerical برابر با null بود آن را با مقدار میانگین دیگر داده ها برای ان فیچر جایگزین کردیم
- اگر مقدار داده ای برای یک فیچر مشخص categorical برابر با null بود آن را با مقداری که بیشترین تکرار برای آن فیچر را داشته است جایگزین کردیم
- اگر یک فیچر categorical بیشتر از ۵۰٪ مقدار ممکن داشت، آن فیچر را حذف کردیم
- مقادیر فیچرهای numerical را normalize کردیم به این ترتیب که برای ان فیچر میانگین صفر و واریانس یک شود

نهاینتا بعد از پیش پردازش هیچ فیچری حذف نشد ولی داده هایی که مقادیر null داشتند پیش پردازش شدند. در این دیتابست ۲۰۰۰ داده و ۲۱ فیچر داریم. کلاس price_range نیز ۴ مقدار دارد.

در این سوال ۸۰ درصد داده را برای train و ۲۰ درصد را برای test امتحان میکنیم. و عملکرد مدل را با توجه به معیارهای زیر ارزیابی کردیم:

- Accuracy
- Precision, recall and F1 for each class, and also averaged over all 4 classes

برای پیاده سازی از پکیج `sklearn.svm` از تابع `svc` استفاده کردیم.

سوال ۳

برای کلسیفایر `svm` پارامترهای مختلفی را امتحان کردیم که عبارتند از:

- **Kernel:** ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’
- **Gamma:** if kernel is ‘poly’, ‘rbf’, ‘sigmoid’, then examine two possible values for gamma:
 - Auto, scale
- **multi -label classification setting:** one-versus-one (ovo) , one-vs-rest (ovr)
- **C (which is the regularizer):** default value 1 - مقادیر مختلف این پارامتر در سوال های بعدی امتحان شده است

به طور کلی ۱۴ حالت مختلف را بررسی کردیم. نتایج برای multi-label classification setting برای هر دو حالت ovo و ovr یکسان بود. بنابراین نتایج را فقط برای یک حالت گزارش میکنیم.

- بهترین عملکرد از نظر accuracy برای حالتی است که از linear kernel استفاده میکنیم و $accuracy = 0.9725$
- بهترین عملکرد از نظر میانگین ۴ کلاس برای F1 برای حالتی است که از linear kernel استفاده میکنیم و $F1 = 0.9721$
- بهترین عملکرد از نظر میانگین ۴ کلاس برای precision برای حالتی است که از linear kernel استفاده میکنیم و $precision = 0.9724$
- بهترین عملکرد از نظر میانگین ۴ کلاس برای recall برای حالتی است که از linear kernel استفاده میکنیم و $recall = 0.9719$

```

SVM results for kernel linear-- accuracy: 0.9725
Precision: 0.9724 [0.99038462 0.95192308 0.9673913 0.98]
Recall: 0.9719 [0.99038462 0.97058824 0.93684211 0.98989899]
Fscore: 0.9721 [0.99038462 0.96116505 0.95187166 0.98492462]
=====
SVM results for kernel poly and Kernel coeff auto-- accuracy: 0.8025
Precision: 0.8272 [0.975 0.68852459 0.69298246 0.95238095]
Recall: 0.8033 [0.75 0.82352941 0.83157895 0.80808081]
Fscore: 0.807 [0.84782609 0.75 0.75598086 0.87431694]
=====
SVM results for kernel poly and Kernel coeff scale-- accuracy: 0.8175
Precision: 0.8331 [0.94318182 0.72566372 0.71052632 0.95294118]
Recall: 0.8182 [0.79807692 0.80392157 0.85263158 0.81818182]
Fscore: 0.8207 [0.86458333 0.7627907 0.77511962 0.88043478]
=====
SVM results for kernel rbf and Kernel coeff auto-- accuracy: 0.895
Precision: 0.8975 [0.96875 0.81981982 0.85263158 0.94897959]
Recall: 0.8946 [0.89423077 0.89215686 0.85263158 0.93939394]
Fscore: 0.8953 [0.93 0.85446009 0.85263158 0.94416244]
=====
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.8875
Precision: 0.891 [0.96808511 0.81081081 0.83673469 0.94845361]
Recall: 0.8875 [0.875 0.88235294 0.86315789 0.92929293]
Fscore: 0.8882 [0.91919192 0.84507042 0.84974093 0.93877551]
=====
SVM results for kernel sigmoid and Kernel coeff auto-- accuracy: 0.9225
Precision: 0.9232 [0.95238095 0.86915888 0.89247312 0.97894737]
Recall: 0.9216 [0.96153846 0.91176471 0.87368421 0.93939394]
Fscore: 0.9222 [0.9569378 0.88995215 0.88297872 0.95876289]
=====
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy: 0.905
Precision: 0.906 [0.96116505 0.84684685 0.85869565 0.95744681]
Recall: 0.9035 [0.95192308 0.92156863 0.83157895 0.90909091]
Fscore: 0.9042 [0.95652174 0.88262911 0.84491979 0.93264249]
=====
```

سوال ۴

پارامتر C مسیول کنترل soft margin و یا Hard margin است. هرچه مقدار این پارامتر کمتر باشد به این معنی است که regularization کمتر است و تنبیه کمتری برای missclassification در نظر گرفته میشود و در نتیجه به عبارتی softer margin خواهیم داشت. و هرچه مقدار C بیشتر باشد harder margin خواهیم داشت. برای پاسخ به این سوال مقادیر مختلفی از C را ازمایش کردیم: $\{0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1e10\}$, برای اینکه مطمئن باشیم hard margin را در نظر گرفتیم و مدل عملاً hard margin است مقدار $1e10$ را نیز امتحان کردیم و آن را در نتایج حالت margin نامیدیم. در زیر نتایج soft and hard margin برای کرنش های مختلف را میبینیم و حالت کلسیفیکشن ovr:

- Linear kernel

```

Soft Margin, C = 0.0001
SVM results for kernel linear-- accuracy: 0.2375
Precision: 0.0594 [0. 0. 0.2375 0. ]
Recall: 0.25 [0. 0. 1. 0. ]
Fscore: 0.096 [0. 0. 0.38383838 0. ]
Soft Margin, C = 0.001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision
    _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel linear-- accuracy: 0.51
Precision: 0.5964 [0.73684211 0.24444444 0.40434783 1. ]
Recall: 0.5157 [0.67307692 0.10784314 0.97894737 0.3030303 ]
Fscore: 0.4727 [0.70351759 0.14965986 0.57230769 0.46511628]
Soft Margin, C = 0.1
SVM results for kernel linear-- accuracy: 0.945
Precision: 0.9461 [0.98019802 0.88990826 0.93478261 0.97959184]
Recall: 0.9445 [0.95192308 0.95098039 0.90526316 0.96969697]
Fscore: 0.9449 [0.96585366 0.91943128 0.9197861 0.97461929]
Soft Margin, C = 1
SVM results for kernel linear-- accuracy: 0.9725
Precision: 0.9724 [0.99038462 0.95192308 0.9673913 0.98 ]
Recall: 0.9719 [0.99038462 0.97058824 0.93684211 0.98989899]
Fscore: 0.9721 [0.99038462 0.96116505 0.95187166 0.98492462]
Soft Margin, C = 10
SVM results for kernel linear-- accuracy: 0.97
Precision: 0.9701 [1. 0.94339623 0.94736842 0.98969072]
Recall: 0.9696 [0.98076923 0.98039216 0.94736842 0.96969697]
Fscore: 0.9697 [0.99029126 0.96153846 0.94736842 0.97959184]
Soft Margin, C = 100
SVM results for kernel linear-- accuracy: 0.97
Precision: 0.9709 [0.99029126 0.92592593 0.9673913 1. ]
Recall: 0.9695 [0.98076923 0.98039216 0.93684211 0.97979798]
Fscore: 0.9699 [0.98550725 0.95238095 0.95187166 0.98979592]
Hard Margin
SVM results for kernel linear-- accuracy: 0.9675
Precision: 0.9683 [0.98095238 0.92523364 0.96703297 1. ]
Recall: 0.9668 [0.99038462 0.97058824 0.92631579 0.97979798]
Fscore: 0.9673 [0.98564593 0.94736842 0.94623656 0.98979592]

```

- همانطور که میبینیم اگر مدل خیلی soft margin باشد (C=0.0001) نتایج بد است ولی اگر کاملاً هم soft margin (C=1e10) باشد باز نتایج خوب نیست. بهترین حالت زمانی است که بین این دو حالت است کمی soft و hard باشد که دقت برابر با ۰.۹۷۲۵ است. F1 نیز بیشترین مقدار را در این حالت دارد.

● Poly Kernel

```

=====
Soft Margin, C = 0.0001
SVM results for kernel poly and Kernel coeff scale-- accuracy: 0.2375
Precision: 0.0594 [0. 0. 0.2375 0. ]
Recall: 0.25 [0. 0. 1. 0. ]
Fscore: 0.096 [0. 0. 0.38383838 0. ]
Soft Margin, C = 0.001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: P
    _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: P
    _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel poly and Kernel coeff scale-- accuracy: 0.2375
Precision: 0.0594 [0. 0. 0.2375 0. ]
Recall: 0.25 [0. 0. 1. 0. ]
Fscore: 0.096 [0. 0. 0.38383838 0. ]
Soft Margin, C = 0.1
SVM results for kernel poly and Kernel coeff scale-- accuracy: 0.5725
Precision: 0.687 [0.94285714 0.38983051 0.41517857 1. ]
Recall: 0.5785 [0.63461538 0.2254902 0.97894737 0.47474747]
Fscore: 0.5678 [0.75862069 0.28571429 0.5830721 0.64383562]
Soft Margin, C = 1
SVM results for kernel poly and Kernel coeff scale-- accuracy: 0.8175
Precision: 0.8331 [0.94318182 0.72566372 0.71052632 0.95294118]
Recall: 0.8182 [0.79807692 0.80392157 0.85263158 0.81818182]
Fscore: 0.8207 [0.86458333 0.7627907 0.77511962 0.88043478]
Soft Margin, C = 10
SVM results for kernel poly and Kernel coeff scale-- accuracy: 0.8075
Precision: 0.811 [0.90625 0.71171171 0.72916667 0.89690722]
Recall: 0.8067 [0.83653846 0.7745098 0.73684211 0.87878788]
Fscore: 0.8081 [0.87 0.74178404 0.73298429 0.8877551 ]
Soft Margin, C = 100
SVM results for kernel poly and Kernel coeff scale-- accuracy: 0.81
Precision: 0.8143 [0.91489362 0.71818182 0.72727273 0.89690722]
Recall: 0.8095 [0.82692308 0.7745098 0.75789474 0.87878788]
Fscore: 0.811 [0.86868687 0.74528302 0.74226804 0.8877551 ]
Hard Margin
SVM results for kernel poly and Kernel coeff scale-- accuracy: 0.81
Precision: 0.8143 [0.91489362 0.71818182 0.72727273 0.89690722]
Recall: 0.8095 [0.82692308 0.7745098 0.75789474 0.87878788]
Fscore: 0.811 [0.86868687 0.74528302 0.74226804 0.8877551 ]

```

- همانطور که میبینیم اگر مدل خیلی soft margin باشد (C=0.0001) نتایج بد است ولی اگر کاملاً هم soft margin (C=1e10) باشد باز نتایج خوب نیست. بهترین حالت زمانی است که بین این دو حالت است کمی soft و hard باشد که دقت برابر با ۰.۸۱۷۵ است. F1 نیز بیشترین مقدار را در این حالت دارد.

● RBF Kernel

```
=====
Soft Margin, C = 0.0001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:
    _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.2375
Precision: 0.0594 [0. 0. 0.2375 0. ]
Recall: 0.25 [0. 0. 1. 0. ]
Fscore: 0.096 [0. 0. 0.38383838 0. ]
Soft Margin, C = 0.001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:
    _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.2375
Precision: 0.0594 [0. 0. 0.2375 0. ]
Recall: 0.25 [0. 0. 1. 0. ]
Fscore: 0.096 [0. 0. 0.38383838 0. ]
Soft Margin, C = 0.1
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.74
Precision: 0.7741 [0.93975904 0.60714286 0.58730159 0.96202532]
Recall: 0.7408 [0.75 0.66666667 0.77894737 0.76767677]
Fscore: 0.7483 [0.8342246 0.63551402 0.66968326 0.85393258]
Soft Margin, C = 1
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.8875
Precision: 0.891 [0.96808511 0.81081081 0.83673469 0.94845361]
Recall: 0.8875 [0.875 0.88235294 0.86315789 0.92929293]
Fscore: 0.8882 [0.91919192 0.84507042 0.84974093 0.93877551]
Soft Margin, C = 10
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.885
Precision: 0.885 [0.93069307 0.82857143 0.875 0.90566038]
Recall: 0.8843 [0.90384615 0.85294118 0.81052632 0.96969697]
Fscore: 0.8839 [0.91707317 0.84057971 0.84153005 0.93658537]
Soft Margin, C = 100
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.8925
Precision: 0.8938 [0.94897959 0.82568807 0.88636364 0.91428571]
Recall: 0.8918 [0.89423077 0.88235294 0.82105263 0.96969697]
Fscore: 0.8919 [0.92079208 0.85308057 0.85245902 0.94117647]
Hard Margin
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.8925
Precision: 0.8938 [0.94897959 0.82568807 0.88636364 0.91428571]
Recall: 0.8918 [0.89423077 0.88235294 0.82105263 0.96969697]
Fscore: 0.8919 [0.92079208 0.85308057 0.85245902 0.94117647]
```

- همانطور که میبینیم اگر مدل خیلی hard margin باشد (C=0.0001) نتایج بد است ولی اگر کاملاً soft margin باشد باز نتایج خوب نیست. بهترین حالت زمانی است که بین این دو حالت است کمی soft و کمی hard باشد. F1 نیز بیشترین مقدار را در این حالت دارد. در این کرنل مدل کمی hard margin از کرنلهای قبلی است تا نتایج بهتری بگیریم.

● Sigmoid kernel

```
=====
Soft Margin, C = 0.0001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:
    _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy: 0.2375
Precision: 0.0594 [0. 0. 0.2375 0. ]
Recall: 0.25 [0. 0. 1. 0. ]
Fscore: 0.096 [0. 0. 0.38383838 0. ]
Soft Margin, C = 0.001
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning:
    _warn_prf(average, modifier, msg_start, len(result))
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy: 0.2375
Precision: 0.0594 [0. 0. 0.2375 0. ]
Recall: 0.25 [0. 0. 1. 0. ]
Fscore: 0.096 [0. 0. 0.38383838 0. ]
Soft Margin, C = 0.1
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy: 0.8725
Precision: 0.883 [0.96907216 0.79130435 0.77142857 1. ]
Recall: 0.8718 [0.90384615 0.89215686 0.85263158 0.83838384]
Fscore: 0.874 [0.93532338 0.83870968 0.81 0.91208791]
Soft Margin, C = 1
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy: 0.905
Precision: 0.906 [0.96116505 0.84684685 0.85869565 0.95744681]
Recall: 0.9035 [0.95192308 0.92156863 0.83157895 0.90909091]
Fscore: 0.9042 [0.95652174 0.88262911 0.84491979 0.93264249]
Soft Margin, C = 10
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy: 0.845
Precision: 0.8458 [0.9047619 0.78899083 0.78723404 0.90217391]
Recall: 0.8435 [0.91346154 0.84313725 0.77894737 0.83838384]
Fscore: 0.8441 [0.90909091 0.81516588 0.78306878 0.86910995]
Soft Margin, C = 100
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy: 0.8375
Precision: 0.839 [0.89719626 0.76146789 0.76344086 0.93406593]
Recall: 0.8357 [0.92307692 0.81372549 0.74736842 0.85858586]
Fscore: 0.8367 [0.90995261 0.78672986 0.75531915 0.89473684]
Hard Margin
SVM results for kernel sigmoid and Kernel coeff scale-- accuracy: 0.815
Precision: 0.8195 [0.88349515 0.71304348 0.75824176 0.92307692]
Recall: 0.8134 [0.875 0.80392157 0.72631579 0.84848485]
Fscore: 0.8153 [0.87922705 0.75576037 0.74193548 0.88421053]
```

- همانطور که میبینیم اگر مدل خیلی hard margin باشد (C=0.0001) نتایج بد است ولی اگر کاملاً soft margin باشد (C=1e10) باز نتایج خوب نیست. بهترین حالت زمانی است که بین این دو حالت است کمی soft و کمی hard باشد. F1 نیز بیشترین مقدار را در این حالت دارد.

سوال ۵ و ۶

برای تمامی حالت ها در این دو سوال از کرنل RBF و با گاما scale استفاده کردیم و مدل svm را ترین کردیم. قابل ذکر است که عملکرد مدل با این پارامترها و با تمامی فیچرها به طور معمولی به شرح زیر است:

```
=====
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.8875
Precision: 0.891 [0.96808511 0.81081081 0.83673469 0.94845361]
Recall: 0.8875 [0.875 0.88235294 0.86315789 0.92929293]
Fscore: 0.8882 [0.91919192 0.84507042 0.84974093 0.93877551]
```

Case 1) Binning ‘battery power’ feature

سه حالت مختلف برای سایز بین ها در نظر گرفتیم. ۵ بین، ۱۰ بین، و ۵ بین با سایز مختلف:

- 5 equal bins
- 10 equal bins
- 5 unequal bins: (-inf, -1.67939025], (-1.67939025,-1.2], (-1.2,-0.3), (-0.3,0], (0, 1.6], (1.6, 1.73016828]

نتایج را برای هر سه حالت در زیر میبینیم. دقت کنین که در این حالت باقی فیچر ها به شکل قبل هستند و تنها فیچر battery power را پردازش کردیم(در کل به مدل تمامی فیچرها داده شده است به جز فیچر battery power که نسخه پردازش شده داده شده است)

```
===== 5 equal bins =====
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.9025
Precision: 0.9034 [0.94059406 0.85294118 0.85148515 0.96875 ]
Recall: 0.9028 [0.91346154 0.85294118 0.90526316 0.93939394]
Fscore: 0.9028 [0.92682927 0.85294118 0.87755102 0.95384615]

===== 10 equal bins =====
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.895
Precision: 0.8982 [0.96842105 0.85046729 0.82692308 0.94680851]
Recall: 0.8953 [0.88461538 0.89215686 0.90526316 0.8989899 ]
Fscore: 0.8955 [0.92462312 0.8708134 0.86432161 0.92227979]

===== 5 unequal bins =====
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.8825
Precision: 0.8835 [0.92156863 0.81730769 0.83673469 0.95833333]
Recall: 0.8824 [0.90384615 0.83333333 0.86315789 0.92929293]
Fscore: 0.8828 [0.91262136 0.82524272 0.84974093 0.94358974]
```

همانطور که میبینیم

بهترین عملکرد برای حالتی است که ۵ بین با سایز یکسان داشته باشیم و عملکرد نسبت معمولی که هیچ Binning نداریم بهتر است (دقت ۰.۹۰۲۵ در مقایسه با ۰.۸۸۷۵)

Case 2) One hot encoding for categorical features

بسیاری از الگوریتم های یادگیری ماشین نیاز به فیچرهای با مقدار عددی (numeric) دارند. این در حالیست که فیچرهای کتگوریکال مقادیر عددی نداشته و به جای آن برچسبی (label) یا کتگوری (category) هستند (به عنوان نمونه فیچر رنگ دارای مقادیر سفید سیاه قرمز و ... می باشد). بنابراین نیاز به روش های کدگذاری نظیر integer و one-hot است. در روش کدگذاری one-hot از متغیر های باینری برای تبدیل فیچر استفاده می کنیم. به عنوان نمونه در مثال رنگ می توان از کدگذاری زیر استفاده کرد:

قرمز	سیاه	سفید
0	0	1
0	1	0
1	0	0

در این قسمت فیچرهای categorical را با روش One Hot encoding پردازش کردیم. در مجموع ۶ فیچر categorical داشتیم:

categorical_features = ['blue', 'dual_sim', 'four_g', 'three_g', 'touch_screen', 'wifi']

نتایج را در زیر برای حالتی که این فیچرهای جدید را استفاده کردیم به شرح زیر است: همانطور که میبینیم عملکرد در این حالت نسبت به عملکرد با حالت معمولی که هیچ one hot encoding نداریم کمی بدتر شده است (دفت ۰.۸۸۲۵ در مقایسه با ۰.۸۸۷۵)

```
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.8825
Precision: 0.8843 [0.93877551 0.80555556 0.84375    0.94897959]
Recall: 0.8824 [0.88461538 0.85294118 0.85263158 0.93939394]
Fscore: 0.8829 [0.91089109 0.82857143 0.84816754 0.94416244]
```

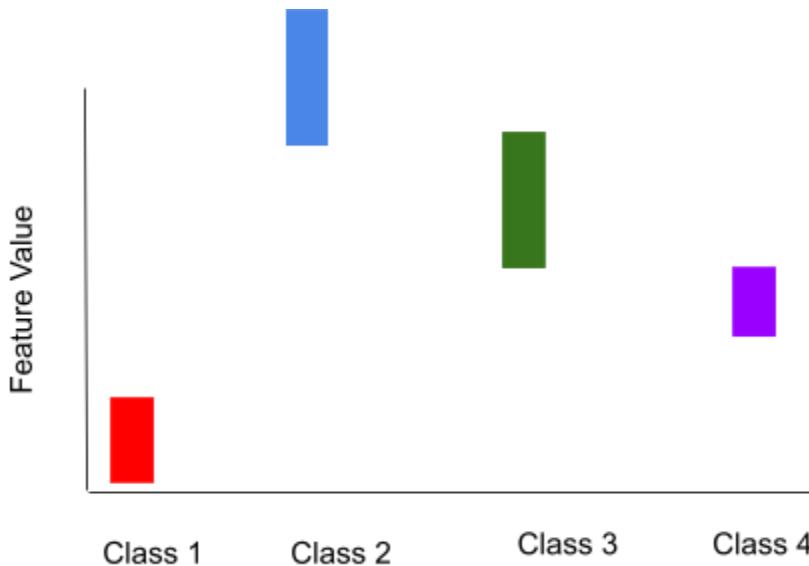
Case 3) Log and Exponential Transformation

برای این سوال transformation های مختلفی را برای هر فیچر جداگانه بررسی کردیم:

- Log transformation
- Exponential transformation with base 'e'
- Power 2 transformation
- Square root transformation

برای هر کدام از این تغییرات،تابع مورد نظر را روی فیچر انتخابی اعمال کرده و عملکرد مدل svm را با فیچر جدید (در حال یکه باقی فیچرها مقدار طبیعی خود را دارند) مقایسه کردیم. هم چنین نموداری از رنج مقادیر این فیچر تبدیل شده برای هر کلاس رسم کردیم. هر چه مقادیر فیچر برای هر کلاس از کلاس دیگر قابل تمیز تر باشد، میتوان گفت آن تبدیل مناسب بوده است چرا که به جداولی داده های کلاسها مخفتف کمک کرده است. به طور مثال به عکس زیر نگاه کنید که رنج مقدار فیچر در هر کلاس با کلاس

دیگر متفاوت است و با داشتن مقدار فیچر میتوان با احتمال خوبی حس زد که آن داده متعلق به کدام کلاس است. هرچه این جداسازی بهتر باشد آن فیچر برای مساله ما بهتر است و اثر بهتری دارد:

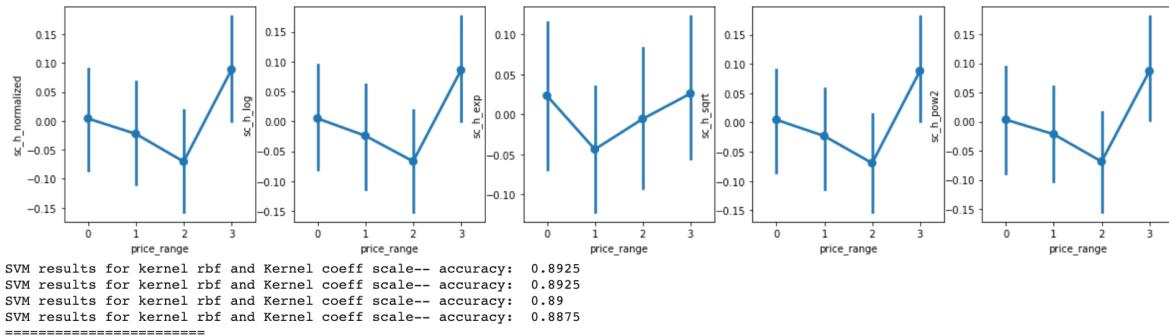


اگر مقدار جداسازی آن فیچر نسبت به کلاس های مختلف در قبل از تبدیل با بعد از تبدیل تفاوتی نکند، و هم چنین عملکرد مدل در classification بهبود قابل توجهی نیابد، میتوان گفت که تبدیل تاثیر خاصی نداشته است. لزوماً تبدیل ها عملکرد مدل را بهبود نمیدهند. بنابراین این مساله برای هر فیچر باید برسی شود. گاهی تبدیل ها جداسازی داده ها را بدتر هم میکنند چرا که قدرت جداسازی فیچر در فضای جدید بعد از تبدیل کمتر میشود. اما در برخی موارد تبدیل ها باعث میشوند که فیچرها به فضای جدید بروند که جداسازی داده های کلاس های مختلف در ان فضا بهتر باشد و در نتیجه دقت و عملکرد مدل بهبود یابد. به طور مثال فیچر n_cores بعضی تبدیل ها مانند تبدیل اکسپوننشیال باعث میشوند که جداسازی کلاسها با توجه به مقدار این فیچر بهتر و راحت‌تر شود و در نتیجه عملکرد و دقت مدل بهبود یابد. برای مثال این فیچر تعداد کورهای موبایل را نشان میدهد و هرچه بیشتر باشد قاعده‌تا قیمت موبایل باید بالاتر باشد. تبدیل اکسپوننشیال باعث میشود که کورهای بالاتر مقدار بالاتری برای این فیچر داشته باشند و بهتر از داده های با کور پایین تر قابل جداسازی باشند. بنابراین این تبدیل باعث میشود که کلاس قیمت بالاتر جداسازی بهتری داشته باشد.

برای سنجیدن عملکرد مدل از svm with RBF Kernel and scale gamma استفاده کردیم و عملکرد مدل را با accuracy گزارش دادیم. در زیر نتایج تبدیل برای هر یک از فیچرها را میبینیم. دقت کنین که بدون هیچ گونه تبدیلی روی فیچر ها دقت عملکرد مدل 8875% میباشد.

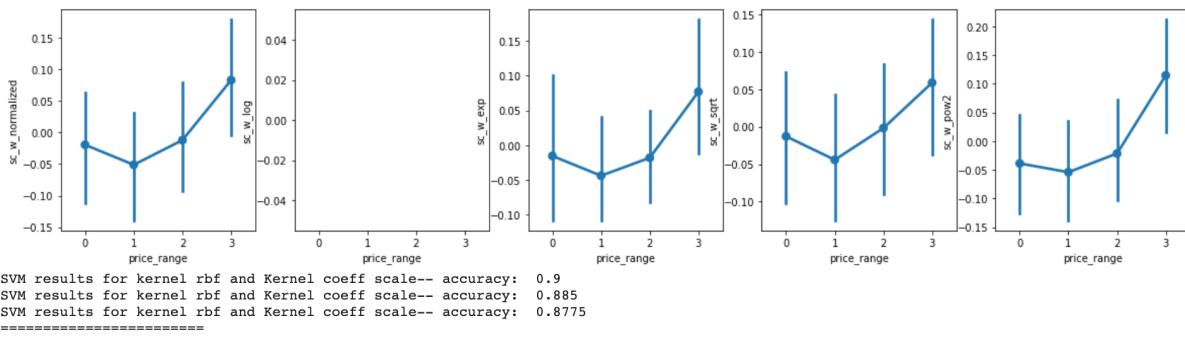
Sc_h Feature:

دقت مدل برای هر کدام از تبدیل ها از چپ به راست (از بالا به پایین) نوشته شده است. به این معنی که اگر تبدیلی لگاریتمی بکنیم عملکرد 0.8925 و اگر تبدیل 2^{power} را اعمال کنیم دقت 8875% خواهد بود. چپ هر یک از نمودار ها رنچ مقدار فیچر را برای هر یک از کلاس ها را نشان میدهد. هر نمودار مربوط به یکتابع تبدیل است. چپ ترین نمودار هیچ تبدیلی نداشته است و مقدار فیچر تنها normalized شده است. همانطور که مشخص است تبدیل وسطی تاثیری در بهتر جداسازی فیچر در کلاس‌های مختلف نداشته است و مقادیر فیچر در کلاس های 0 و 2 و 3 همچنانی زیادی دارند ولی نمودار سمت چپ کلاس 3 مقادیر متفاوت تری از باقی کلاس ها دارد هم چنین کلاس 2 مقادیر متفاوتی از سه کلاس دیگر دارد. در کل 4 تابع تبدیل چندان و چشم گیری در بهتر جداسازی کلاس ها با توجه به این فیچر نداشته اند. هم چنین اگر به عملکرد مدل با توجه به هر کدام از این تبدیل ها نگاه کنیم، دقت مدل غیر چشم گیری نداشته است و دقت تنها 5% افزایش داشته است (8875%). بدون تابع تبدیل به نسبت 8925% با تابع تبدیل لگاریتم و یا توانی اکسپوننشیال)



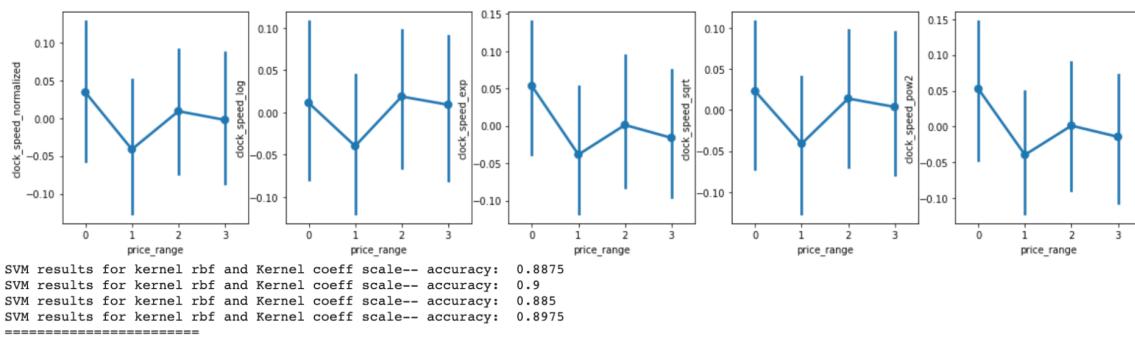
sc_w Feature:

همانطور که مشخص است تبدیل اکسپوننشیال دقت مدل را نسبت به باقی تبدیل ها بیشتر بهبود داده است (۹). با تبدیل اکسپوننشیان به نسبت ۰.۸۸۷۵ بدون تبدیل). هم چنین اگر مقدار رنج فیچرها با این تبدیل را با حالت های دیگر مقایسه کنیم مشاهده میکنیم که داده های کلاس صفر اندکی بهتر از حالت های دیگر قابل تمیز از داده های کلاسهای ۲ و ۳ هستند (در نمودارهای دیگر میبینیم این سه کلاس هم پوشانی بیشتری دارند). بعد از تبدیل داده های کلاس صفر تارنج ۱.۰ قرار گرفتند و در حالت بدون تبدیل (چپ ترین نمودار) تارنج کمتر از ۰.۵ بودند که با داده های کلاس ۲ و ۳ هم پوشانی بیشتری داشت.



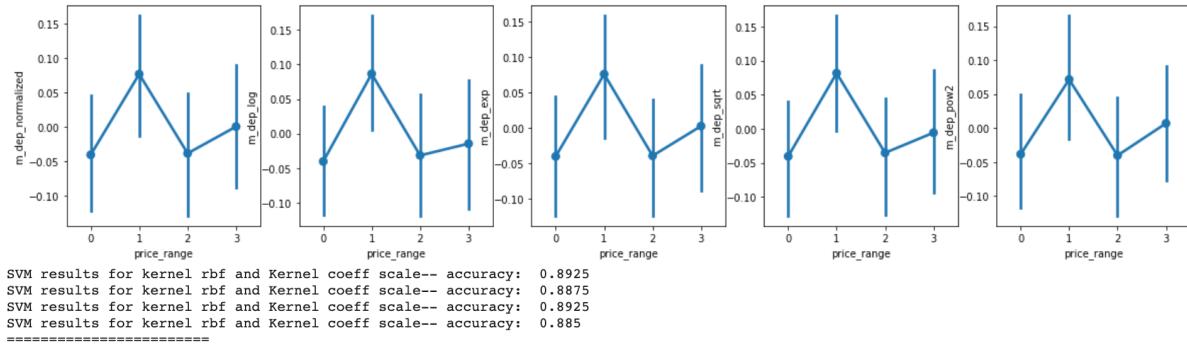
Clock speed Feature:

همانطور که مشخص است تبدیل اکسپوننشیال دقت مدل را نسبت به باقی تبدیل ها بیشتر بهبود داده است (۹). با تبدیل اکسپوننشیان به نسبت ۰.۸۸۷۵ بدون تبدیل). هم چنین اگر مقدار رنج فیچرها با این تبدیل را با حالت های دیگر مقایسه کنیم مشاهده میکنیم که داده های کلاس صفر اندکی بهتر از حالت های دیگر قابل تمیز از داده های کلاسهای ۲ و ۳ هستند (در نمودارهای دیگر میبینیم این سه کلاس هم پوشانی بیشتری دارند). بعد از تبدیل داده های کلاس صفر در رنج [۰.۱, ۰.۱۵] قرار گرفتند و در حالت بدون تبدیل (چپ ترین نمودار) در رنج کمتر از ۰.۱ بودند که با داده های کلاس ۲ و ۳ هم پوشانی بیشتری داشت.



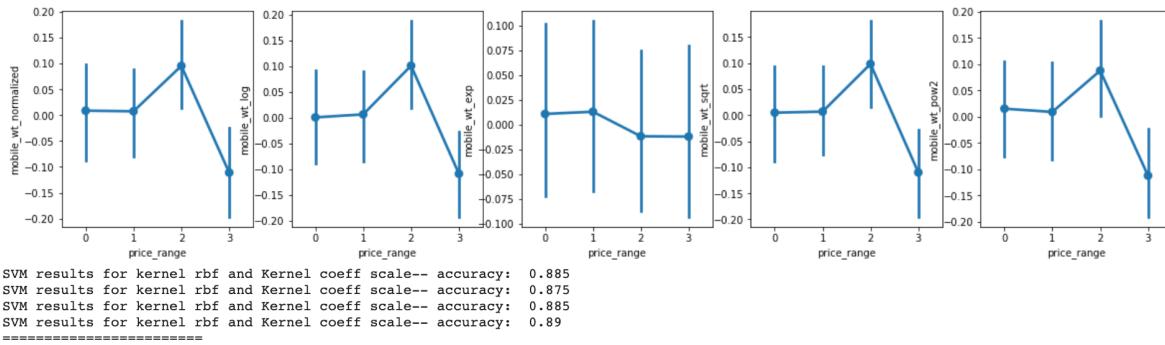
m_dep Feature:

با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل‌ها فرق چشم‌گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده‌های کلاس‌های مختلف تغییری نکرده است. هم‌چنان دقت مدل بهبود چشمگیری نداشته است (۰.۸۹۲۵ در بهترین حالت تبدیل‌ها به نسبت ۰.۸۸۷۵، بدون هیچ گونه تبدیلی). حتی میتوان گفت که تبدیل توان دوم اثر منفی داشته است چرا که دقت عملکرد مدل به ۰.۸۸۵ کاهش یافته است.



mobile_wt Feature:

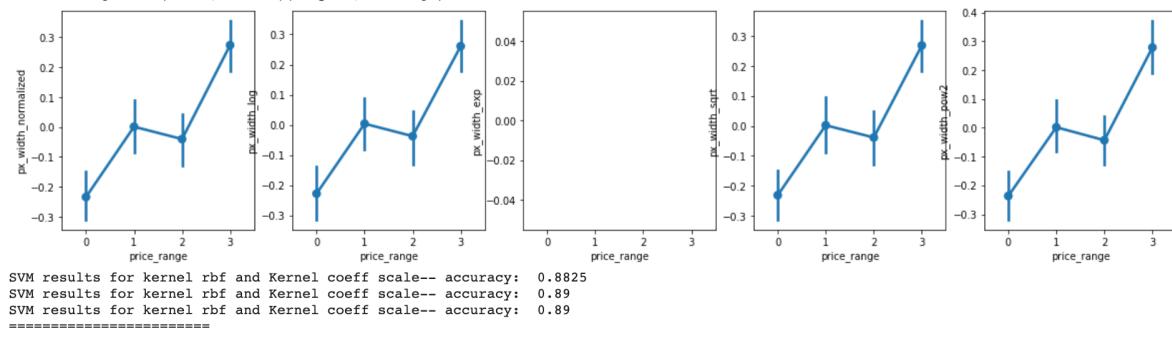
با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل‌ها فرق چشم‌گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده‌های کلاس‌های مختلف تغییری نکرده است. هم‌چنان دقت مدل بهبود چشمگیری نداشته است (۰.۸۹ در بهترین حالت تبدیل‌ها به نسبت ۰.۸۸۷۵، بدون هیچ گونه تبدیلی). حتی میتوان گفت که تبدیل اکسپوننسیال اثر منفی داشته است چرا که دقت عملکرد مدل به ۰.۸۷۵ کاهش یافته است و جداسازی کلاس‌ها با این فیچر بدتر شده است چرا که رنج مقدار این فیچر در کلاسهای مختلف اشتراک زیادی داشته است! (نمودار سوم وسطی)



px_width Feature:

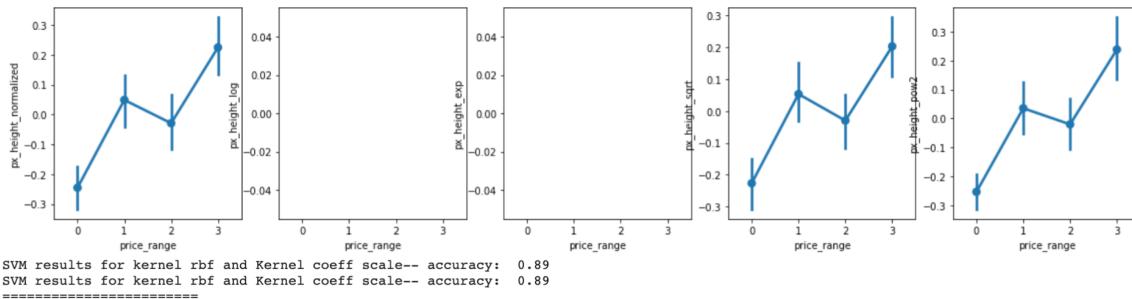
با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل‌ها فرق چشم‌گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده‌های کلاس‌های مختلف تغییری نکرده است. هم‌چنان دقت مدل بهبود چشمگیری نداشته است (۰.۸۹ در بهترین حالت تبدیل‌ها به نسبت ۰.۸۸۷۵، بدون هیچ گونه تبدیلی). حتی میتوان گفت که تبدیل لگاریتمی اثر منفی داشته است چرا که دقت عملکرد مدل به ۰.۸۸۲۵ کاهش یافته است. تبدیل نمایی برای این فیچر باعث شده است که اکثر داده‌ها مقدار بسیار بسیار زیادی بگیرند و overflow کرده و مقدار آنها در متغیر float جا نشود بنابراین نمودار مربوط به این تغییر خالی است و هم‌چنان دقت

مدل با این تبدیل گزارش نشده است



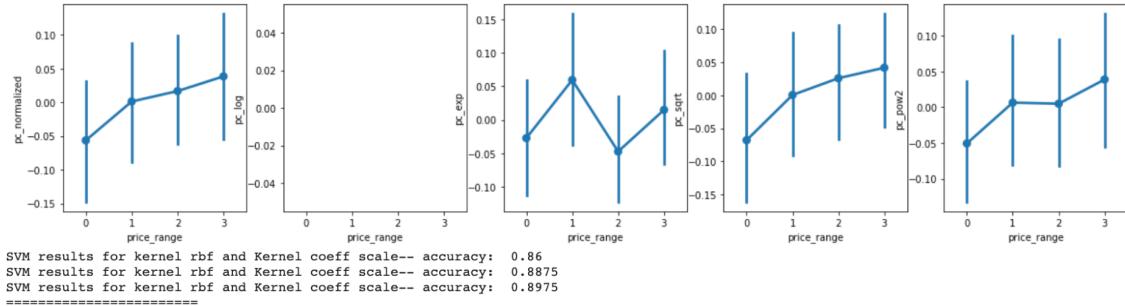
px_height Feature:

با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل‌ها فرق چشم‌گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده‌های کلاس‌های مختلف تغییری نکرده است. هم‌چنان دقت مدل بهبود چشمگیری نداشته است (۰.۸۹ در بهترین حالت تبدیل‌ها به نسبت ۰.۸۷۷۵ بدون هیچ گونه تبدیل). تبدیل نمایی برای این فیچر باعث شده است که اکثر داده‌ها مقدار بسیار پسیار زیادی بگیرند و overflow کرده و مقدار آنها در متغیر float جا نشود بنابراین نمودار مربوط به این تغییر خالی است و هم‌چنان دقت مدل با این تبدیل گزارش نشده است. به طور مشابه تبدیل لگاریتمی نیز باعث شده است مقدار این فیچر برای اکثر داده‌ها شود و دقت مدل گزارش نشده است.



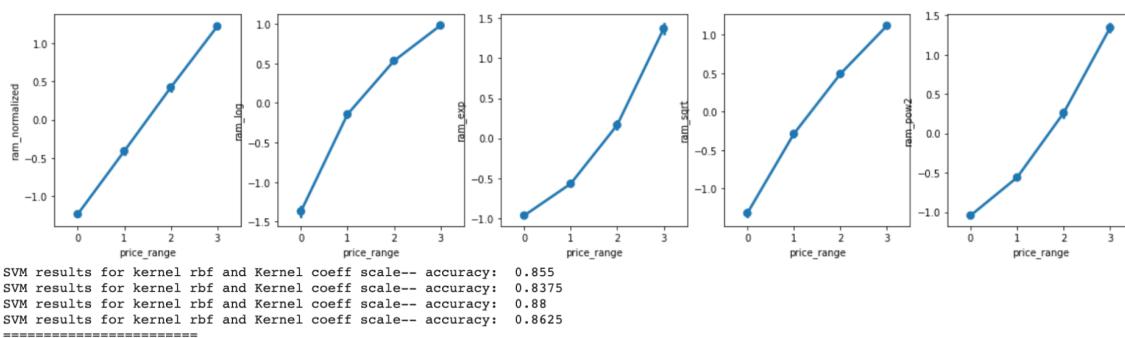
pc Feature:

با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل‌ها فرق چشم‌گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده‌های کلاس‌های مختلف تغییری نکرده است. هم‌چنان دقت مدل بهبود چشمگیری نداشته است (۰.۸۹۷۵ در بهترین حالت تبدیل‌ها به نسبت ۰.۸۷۷۵ بدون هیچ گونه تبدیل). در تبدیل جذری (نمودار راست) جداسازی داده‌های کلاس ۱ و ۲ نسبت به حالت بدون تبدیل کمی بهبود داشته است و دقت مدل هم کلی بالاتر رفته و به ۰.۸۹۷۵ رسیده است. هم‌چنان میتوان گفت که تبدیل نمایی اکسپوننشیال اثر منفی داشته است چرا که دقت عملکرد مدل به ۰.۸۶ کاهش یافته است و جداسازی کلاس‌ها با این تبدیل فیچر بدتر شده است چرا که در تبدیل‌های دیگر و بدون تبدیل هر چه مقدار این فیچر بالاتر بود شناسی بیشتری داشت که به کلاسهای با شماره بالاتر تعلق داشته باشد ولی با تبدیل اکسپوننشیال رنج مقدار این فیچر در کلاسهای مختلف اشتراک زیادی داشته است! (نمودار سوم وسطی با تبدیل اکسپوننشیال به نسبت بدون تبدیل در نمودار سمت چپ)



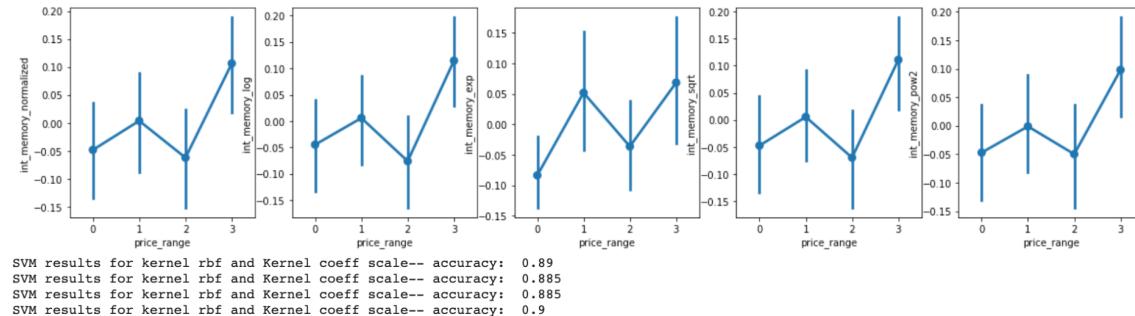
RAM Feature:

با توجه به ۵ نمودار زیر میتوان گفت که رنج مقداری فیچرها بعد از هیچ کدام از تبدیل‌ها فرق چشم‌گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده‌های کلاس‌های مختلف تغییری نکرده است. حتی قدرت مدل کمتر هم شده است چرا که با تبدیل اکسپوننشیال قدرت به ۰.۸۳۷ هم کاهش پیدا کرده است. تبدیل توان دوم بین ۴ تبدیل بهتر بوده است و دقیق‌تر است و لی باز به نسبت حالتی که تبدیلی نباشد دقیق‌تر است!



int_memory Feature:

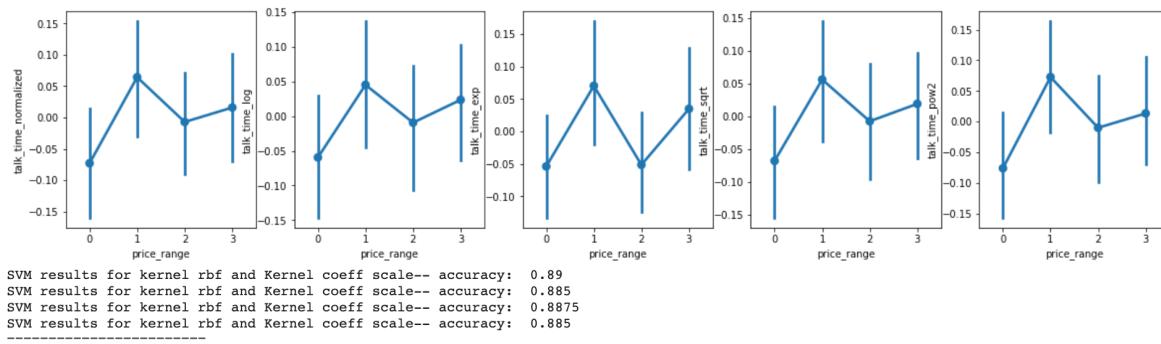
با توجه به ۵ نمودار زیر میتوان گفت که به طور کلی رنج مقداری فیچرها بعد از هیچ کدام از تبدیل‌ها فرق چشم‌گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده‌های کلاس‌های مختلف تغییری نکرده است. فقط دقیق‌تر و عملکرد مدل بعد از تبدیل توان دوم کمی بهبود یافته است (۰.۹ به نسبت ۰.۸۸۷۵). هم‌چنان‌به‌جز‌این ۴ تبدیل برای این فیچر ما هم مموري را برابر ۱۰۲۴ نیز تقسیم کردیم تا به صورت گیکابایت شود ولی عملکرد مدل تغییری نداشت و همان ۰.۸۸۷۵ بود.



Talk_Time Feature:

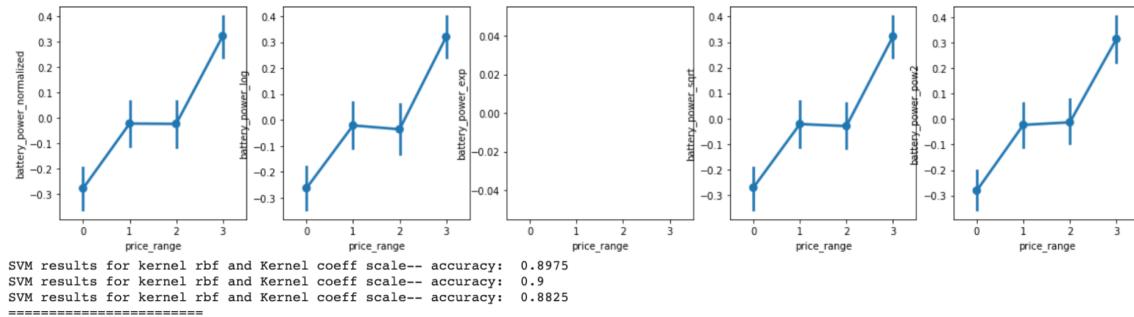
با توجه به ۵ نمودار زیر میتوان گفت که به طور کلی رنج مقداری فیچرها بعد از هیچ کدام از تبدیل‌ها فرق چشم‌گیری با قبل تبدیل نداشته است و قدرت تمیز دادن داده‌های کلاس‌های مختلف تغییری نکرده است. در بهترین حالت عملکرد مدل ۰.۸۹ با تبدیل

لگاریتمی است که بهبود چندانی نسبت به حالت بدون تبدیل که دقت ۰.۸۸۷۵ است ندارد. برای تبدیل اکسپوننشیال و توانی دقت کمی هم کاهش پیدا کرده است و به ۰.۸۸۵ رسیده است.



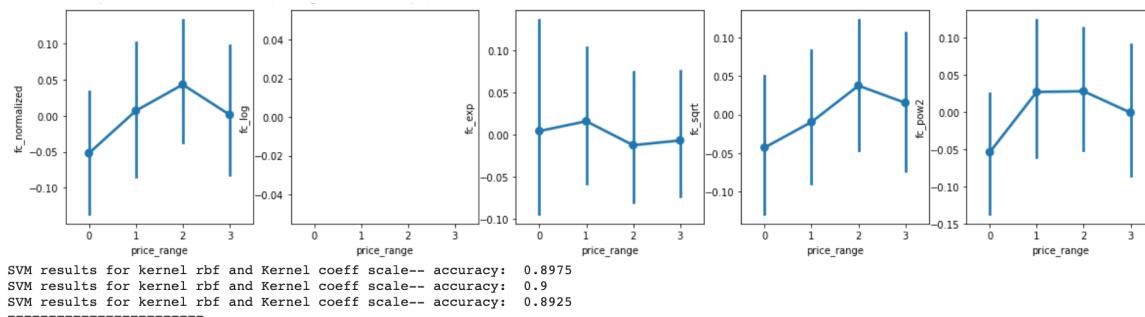
Battery_power Feature:

به طور کلی رنج مقداری فیچرها با این تبدیل تغییری نکرده است. دقت مدل پس از تبدیل جذری کمی بهبود یافته است (دقت ۰.۹ به نسبت ۰.۸۸۷۵).



fc Feature:

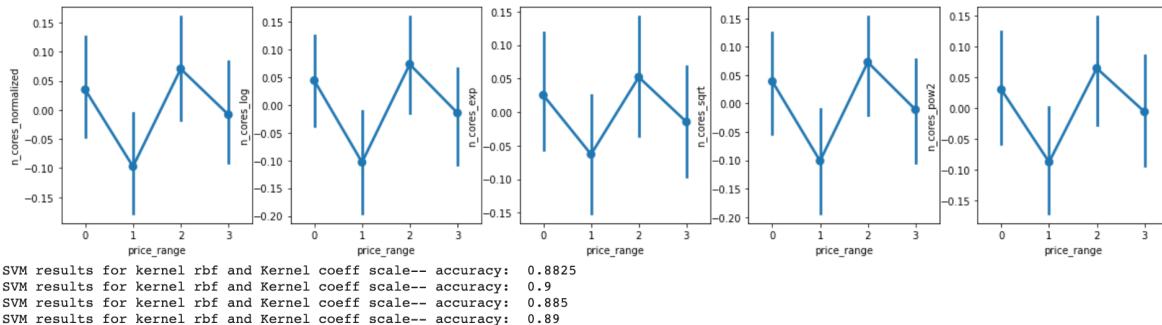
در این حالت تنها تبدیل جذری توائسته است قدرت مدل را کمی بهبود دهد و به ۰.۹ برساند. اگر به نمودارها هم نگاه کنیم این تبدیل باعث شده است که کلاس ۰ و ۱ قابلیت جداسازی کمی بهتری داشته باشند.



n_cores Feature:

همانطور که مشخص است تبدیل اکسپوننشیال دقت مدل را نسبت به باقی تبدیل‌ها بیشتر بهبود داده است (۰.۹ با تبدیل اکسپوننشیان به نسبت ۰.۸۸۷۵ بدون تبدیل). دلیل اثر مثبت این تبدیل این است که این فیچر تعداد کورهای موبایل را نشان میدهد و هرچه بیشتر باشد قاعdet قیمت موبایل باید بالاتر باشد. تبدیل اکسپوننشیال باعث میشود که کورهای بالاتر مقدار بالاتری برای این فیچر داشته باشند

و بهتر از داده های با کور پایین تر قابل جداسازی باشند. بنابراین این تبدیل باعث میشود که کلاس قیمت بالاتر جداسازی بهتری داشته باشد.



Case 4) New Feature: Area w.r.t to Pixel and phone size

برای این سوال دو نوع فیچر را تست کردیم. یکی فیچر مساحت با توجه به پیکسلهای گوشی و دیگری مساحت با توجه به اندازه خود گوشی. در حالت اول دقیق‌تر و در حالت دوم دقیق‌تر بیهود یافت:

- یک فیچر جدید به نام مساحت صفحه گوشی با توجه به پیکسل ساختیم که معادل ضرب طول و عرض پیکسل هاست. دقیق‌تر در زیر گزارش شده است. دقیق‌تر کنین به منظور ارزیابی این فیچر جدید، دو فیچر مرتبط قبلی px_height and px_width را حذف کردیم چون اگر حذف نمیکردیم نگه داشتن فیچر تکراری باعث بدتر شدن عملکرد مدل میشود!

```
X[ 'area_pixel' ] = X[ 'px_height' ] * X[ 'px_width' ]
```

```

SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.7925
Precision: 0.7964 [0.89473684 0.69298246 0.72043011 0.87755102]
Recall: 0.7914 [0.81730769 0.7745098 0.70526316 0.86868687]
Fscore: 0.7929 [0.85427136 0.73148148 0.71276596 0.87309645]
=====
```

- همانطور که میبینیم دقیق‌تر مدل از ۰.۸۸۷۵ به ۰.۷۹۲۵ کاهش یافته است. این نشان میدهد ترکیب کردن این دو فیچر تاثیر مثبتی داشته است و همانطور که قبلاً بودند مدل عملکرد بهتری داشت.
- یک فیچر جدید به نام مساحت صفحه گوشی با توجه به اندازه گوشی ساختیم که معادل ضرب طول و عرض موبایل هاست. دقیق‌تر در زیر گزارش شده است. دقیق‌تر کنین به منظور ارزیابی این فیچر جدید، دو فیچر مرتبط قبلی را حذف کردیم چون اگر حذف نمیکردیم نگه داشتن فیچر تکراری باعث بدتر شدن عملکرد مدل میشود!

```

SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.8975
Precision: 0.8999 [0.95918367 0.81981982 0.86170213 0.95876289]
Recall: 0.897 [0.90384615 0.89215686 0.85263158 0.93939394]
Fscore: 0.8978 [0.93069307 0.85446009 0.85714286 0.94897959]
=====
```

همانطور که میبینیم دقیق‌تر مدل از ۰.۸۸۷۵ به ۰.۸۹۷۵ افزایش یافته است. این نشان میدهد ترکیب کردن این دو فیچر برخلاف فیچرهای مرتبط به پیکسل تاثیر مثبتی داشته است

Case 5) One hot encoding +5 equal binning + Transformation

در این حالت ترکیب حالت‌های قبلی را بر روی فیچرهای کنگوریکال را one hot encoding کردیم. برای فیچر battery power عمل binning ۵ گروه مساوی در نظر گرفتیم. فیچر مساحت خود گوشی را اضافه کردیم (مساحت بر اساس پیکسل را اضافه نکردیم چرا که باعث کاهش دقت شده بود). از بین تابع‌های تبدیل روی فیچرهای مختلف بهترین‌ها را که باعث افزایش عملکرد شده بودند را انتخاب کردیم. تابع تبدیل اکسپوننسیال روی speed وتابع تبدیل اکسپوننسیال روی n_cores و تابع تبدیل جذر روی فیچر clock

```
x_w_dummies_all = X_w_dummies.copy(deep = True)
X_w_dummies_all['battery_power_5_bins'] = pd.cut(X_w_dummies_all['battery_power'], 5, retbins = True, labels=range(5))#labels = range(10)
X_w_dummies_all['area'] = X_w_dummies_all['sc_h'] * X_w_dummies_all['sc_w']

#exp transformation on clock_speed
f_val = train_df_org['clock_speed']
z1 = np.exp2(f_val)
X_w_dummies_all.loc[:, 'clock_speed'+ '_exp'] = ((z1 - z1.mean()) / z1.std())

#exp transformation on n_cores
f_val = train_df_org['n_cores']
z1 = np.exp2(f_val)
X_w_dummies_all.loc[:, 'n_cores'+ '_exp'] = ((z1 - z1.mean()) / z1.std())

#sqrt transformation on fc
f_val = train_df_org['fc']
z1 = np.sqrt(f_val)
X_w_dummies_all.loc[:, 'fc'+ '_sqrt'] = ((z1 - z1.mean()) / z1.std())
```

عملکرد مدل بعد از تمام این تبدیل‌ها کاهش پیدا کرد!!! مدل با کرنل آر بی اف در نظر گرفتیم. دقت از حالت معمولی ۰.۸۸۷۵ و ۰.۷۹۲۵ precision recall f1 نیز کاهش پیدا کردند. دلیل این انافق این است که این فیچرهای همگی با هم باعث افزایش واریانس مدل شده‌اند. مدل بر روی داده اموزشی خوب عمل می‌کند ولی بر روی تست بد می‌شود.

```
SVM results for kernel rbf and Kernel coeff scale-- accuracy: 0.7925
Precision: 0.7951 [0.89      0.6952381 0.69      0.90526316]
Recall: 0.7916 [0.85576923 0.71568627 0.72631579 0.86868687]
Fscore: 0.793 [0.87254902 0.70531401 0.70769231 0.88659794]
=====
```

سوال ۷

به طور کلی الگوریتم‌های ساخت درخت تصمیم در موارد زیر با یکدیگر متفاوت هستند.

- معیار تقسیک (splitting criterion): در فرآیند ساخت درخت تصمیم برای تقسیک داده‌ها در هر راس (node) به یک معیار نیاز داریم. این معیار در درخت‌های کلاسه‌بندی (داده‌های گستره) و رگرسیون (داده‌های پیوسته) متفاوت است. در مساله کلاسه‌بندی هدف تقسیم داده‌ها به گروه‌های کوچکتر و همگون تر (homogeneous) و خالص تر (pure) می‌باشد به گونه‌ای که تا حدی الامکان داده‌ها در هر راس از یک کلاس باشند. در نتیجه می‌توان از معیارهایی نظیر نظریه مجموع مربعات خطای SSE و یا کاهش در واریانس استفاده کرد. در مقابل برای مساله رگرسیون می‌باشد از معیارهایی نظیر مجموع اطلاعات اطلاعات (Information Gain) استفاده کرد. در این حالت می‌توان از این معیارهایی نظیر overfitting بازگشتی یا recursive (recursiveness) این مقدار کم و کمتر شود.

- روش‌های کاهش overfitting
- توانایی حل مساله با داده‌های غیر کامل (incomplete)

به عنوان نمونه در روش ID3 درخت تصمیم کلاسه‌بندی به صورت حریصانه و بالا به پایین ساخته می‌شود به گونه‌ای که در هر مرحله فیچری انتخاب می‌شود که به ترتیب به کمترین و بیشترین مقدار Entropy Information Gain منجر شود. نسل بعدی این روش الگوریتم C4.5 است که می‌تواند با فیچرهای گستره و پیوسته کار کند. همچنین روش CART که معروف به درخت کلاسه‌بندی و رگرسیون است که مانند روش C4.5 عمل می‌کند اما برخلاف آن به جای استفاده از مجموعه قوانین (rule sets) از تقسیک عددی بازگشتی (recursive numerical splitting) استفاده می‌کند.

سوال ۸ و ۹

در ابتدا یک درخت با پارامترهای دیفالت پکیج میسازیم. در پارامترهای پیش فرض معیار **gini** است. هم چنین به شکل پیش فرض مینیم داده های مورد نیاز در هر گره ۱ و مینیم داده های مورد نیاز برای هر **node** برابر با ۲ است. هم چنین هیچ محدودیتی روی عمق درختها وجود ندارد. نتایج درخت با حالت پیش فرض در زیر گزارش شده است:

```
Decision Tree Results with Default Parameters
Accuracy: 0.8075
Precision: 0.8088 [0.89320388 0.69444444 0.72043011 0.92708333]
Recall: 0.806 [0.88461538 0.73529412 0.70526316 0.8989899]
Fscore: 0.8072 [0.88888889 0.71428571 0.71276596 0.91282051]
```

هم چنین با استفاده از روش **grid search** پارامترهای مختلف و مقادیر مختلف پارامترها را ازمایش کردیم که در زیر گزارش شده اند:

- Criterion: Gini, Entropy
- Max depth: maximum depth of the tree: 3, 5, 7, 10
- Min samples split: minimum sample required to split a node: 1,2, 3, 4, 5, 6, 7, 8, 9
- Min samples leaf: minimum sample required for a leaf: 1,2, 3, 4,5, 6, 7,8,9

در روش **cross validation** از روش **grid search** با ۵ فولد استفاده میشود و میانگین عملکرد در فولدها در نظر گرفته میشود.

بهترین درخت برای پارامترهای زیر است که از روش **entropy** استفاده شود و ماکسیمم عمق درخت ۱۰ باشد و مینیم نمونه مورد نیاز برای هر گره ۳ و مینیم نمونه برای برگها ۵ باشد. عملکرد نسبت به درخت ساخته شده با پارامترهای پیش فرض بسیار بهتر است و دقت از ۰.۸۰۷۵ به ۰.۸۸۰۰ بهبود یافته است.

همانطور که از نتایج مشخص است افزایش عمق درخت ها به بهبود عملکرد تاثیر مثبتی دارد. هرچه تعداد نمونه های برگ و گره ها بیشتر باشد عملکرد بهتر میشود ولی از یک حدی اگر بیشتر شود عملکرد کاهش پیدا میکند بنابراین بهترین مقدار برای نمونه های برگ و گره حداقل ۵ و ۳ است.

```
Fitting 5 folds for each of 512 candidates, totalling 2560 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done 956 tasks      | elapsed:      5.0s
Grid Search best parameters
{'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 3}
Decision Tree Results with Best Estimator
Accuracy: 0.88
Precision: 0.8788 [0.92307692 0.83333333 0.84615385 0.91262136]
Recall: 0.8791 [0.92307692 0.83333333 0.81052632 0.94949495]
Fscore: 0.8788 [0.92307692 0.83333333 0.82795699 0.93069307]
[Parallel(n_jobs=-1)]: Done 2560 out of 2560 | elapsed:     17.2s finished
```

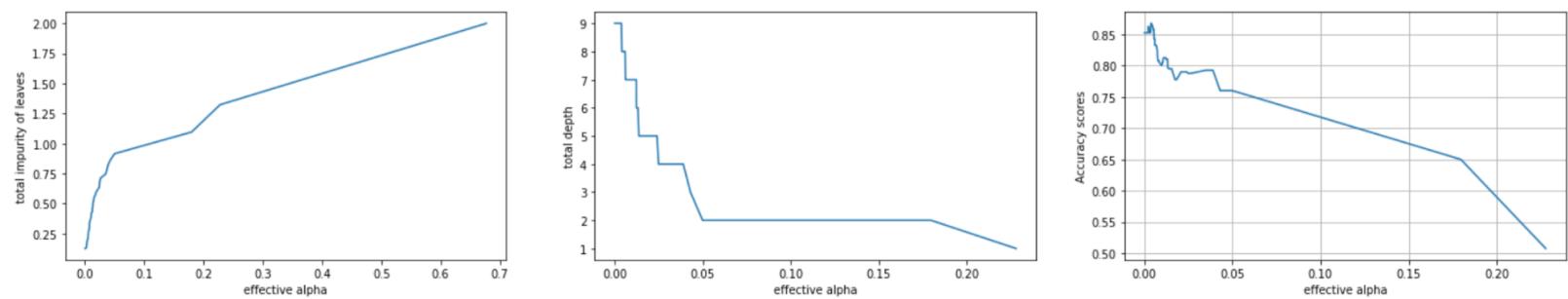
سوال ۱۰

در بسیاری از موارد درخت های تصمیم **overfit** می کنند به این ترتیب که عمل داده های مجموعه آموزش (train) را به خاطر سپرده اند (memorize). دلیل این مشکل اینست که درخت های تصمیم متمایل به افزایش اندازه دارند و در بدترین حالت ممکن است برای هر داده یک برگ (leaf) اختصاص دهند. در چنین حالتی خطای آموزش صفر خواهد شد و درخت دچار overfit می شود. یک روش برای حل این مشکل استفاده از مفهوم حرس یا pruning می باشد. در این روش برخی از شاخه های درخت و

همچنین راس های تصمیم (decision nodes) حذف می شوند (با شروع از برگ ها) تا از افزایش بی رویه درخت جلوگیری شود. برای این کار داده های آموزشی به دو مجموعه آموزشی و اعتباری (validation) تقسیم شده و درخت با استفاده از داده های آموزشی ساخته و با استفاده از مجموعه اعتباری حرس می شود.

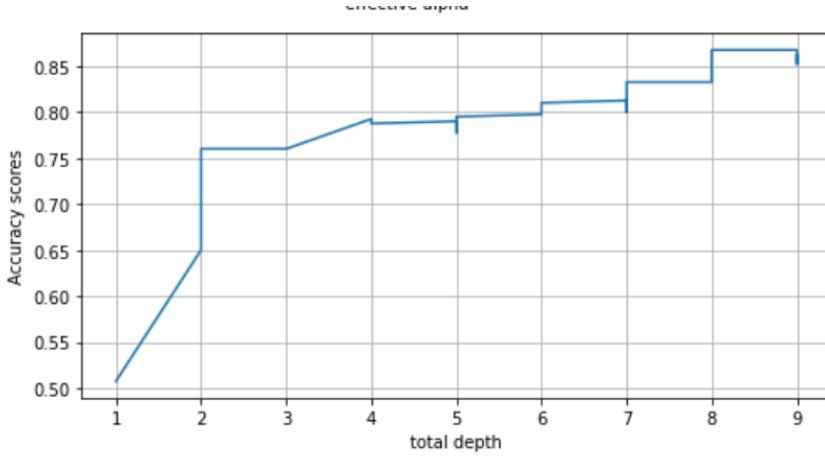
سوال ۱۱

در سوال ۸ و ۹ با بررسی پارامترهایی مثل عمق درخت و مینیم نمونه های مورد نیاز برای گره و نود مساله pre pruning درخت را بررسی کردیم و ارزیابی کردیم که برای این داده چه پارامترهایی بهترین هستند. بنابراین برای این سوال ابتدا درخت با بهترین پارامترها را که توسط سوال ۸ و ۹ پیدا کردیم میسازیم. برای post pruning ازتابع cost complexity pruning است cost complexity pruning path است که مقدار بهینه پارامتر آلفا را پیدا کنیم. بنابراین اطلاعات مربوط به تمام درخت های ممکن با توجه به پارامترهای و هدف این است که مقدار آلفا را ذخیره میکنیم و رابطه بین آلفا و موارد زیر را میسنجم: عمق درخت، دقت مدل و impurity برگ ها. نمودارهای زیر رابطه مقادیر مختلف آلفا با هر کدام از این موارد را نشان میدهد:



همانطور که مشاهده میکنیم با افزایش آلفا مقدار عمق درخت کاهش پیدا میکند. هم چنین با افزایش آلفا مقدار Impurity برگ ها بیشتر میشود. قابل ذکر است که هرچقدر مقدار impurity بیشتر شود عملکرد مدل میتواند بدتر شود چرا که ما به دنبال برگهایی هستیم که تا جای ممکن pure باشند. هم چنین قابل ذکر است که اگر اندازه نمونه های برای هر برگ خیلی کوچک باشد باعث overfitting مدل میشود. نمودار سمت راست که رابطه بین دقت و آلفا است نشان میدهد که با افزایش آلفا در ابتداء دقت افزایش میابد ولی از یک نقطه ای به بعد (وقتی آلفا بیشتر از 0.003 میشود) دقت شروع به کاهش میکند. این نشان میدهد که افزایش آلفا تا حدی نه تنها باعث بهیود عملکرد میشود، بلکه باعث کمتر شدن عمق درخت میشود و هم چنین از overfit شدن مدل جلوگیری میکند.

همچنین رابطه بین دقت مدل و عمق درخت را در نمودار زیر نمایش دادیم که نشان میدهد با افزایش عمق عملکرد بهتر میشود ولی هرچه عمق بالاتر میرود، سرعت افزایش دقت مدل کمتر میشود. علی الخصوص افزایش عمق ۸ به ۹ حتی کمی باعث کم شدن دقت میشود



نهایتاً با توجه به این **post pruning** مترجعه میشویم که بهترین مقدار آلفا $\alpha = 0.037$ میباشد با عمق درخت ۹ و هم چنین دقت مدل 0.8675 میباشد. عملکرد مدل با توجه به معیارهای precision recall F1 نیز در زیر نمایش داده شده است:

```
Precision: 0.8682 [0.91262136 0.78899083 0.85057471 0.92079208]
Recall: 0.8663 [0.90384615 0.84313725 0.77894737 0.93939394]
Fscore: 0.8666 [0.90821256 0.81516588 0.81318681 0.93 ]
```

سوال ۱۲

از پکیج موجود در پایتون استفاده کردیم. یکبار از پارامترهای پیش فرض برای ساختن **random forest** استفاده کردیم که تعداد درختها 100 عدد و معیار **gini** است. هم چنین به شکل پیش فرض مینیم داده های مورد نیاز در هر گره ۱ و مینیم داده های مورد نیاز برای هر **node** برابر با ۲ است. هم چنین هیچ محدودیتی روی عمق درختها وجود ندارد. نتایج جنگل با حالت پیش فرض در زیر گزارش شده است:

```
Random Forest Results with Default Parameters
Accuracy: 0.8525
Precision: 0.8556 [0.93      0.75213675 0.82926829 0.91089109]
Recall: 0.8505 [0.89423077 0.8627451 0.71578947 0.92929293]
Fscore: 0.8509 [0.91176471 0.80365297 0.76836158 0.92 ]
```

در این حالت **accuracy** مدل **random forest** برابر با 0.8525 است که بهتر از دقت **decision tree model** است که 0.8075 میباشد.

هم چنین با استفاده از روش **grid search** پارامترهای مختلف و مقادیر مختلف پارامترها را ازمایش کردیم که در زیر گزارش شده اند:

- Criterion: Gini, Entropy
- number of estimators in random forest: 20, 50, 100, 150, 200, 300, 400, 500
- Max depth: maximum depth of the tree: 3, 5, 7, 10
- Min samples split: minimum sample required to split a node: 1, 2, 3, 4, 5, 6, 7, 8, 9
- Min samples leaf: minimum sample required for a leaf: 1, 2, 3, 4, 5, 6, 7, 8, 9

در روش **grid search** از روش **cross validation** با ۵ فولد استفاده میشود و میانگین عملکرد در فولد ها در نظر گرفته میشود.

```

Fitting 5 folds for each of 512 candidates, totalling 2560 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks    | elapsed:   7.9s
[Parallel(n_jobs=-1)]: Done 196 tasks    | elapsed:  31.2s
[Parallel(n_jobs=-1)]: Done 446 tasks    | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 796 tasks    | elapsed:  2.3min
[Parallel(n_jobs=-1)]: Done 1246 tasks   | elapsed:  3.9min
[Parallel(n_jobs=-1)]: Done 1796 tasks   | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done 2446 tasks   | elapsed:  8.7min
[Parallel(n_jobs=-1)]: Done 2560 out of 2560  | elapsed:  9.2min finished
Grid Search best parameters
{'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2}
Random Forest Results with Best Estimator
Accuracy: 0.89
Precision: 0.8911 [0.96      0.81651376 0.83870968 0.94897959]
Recall:  0.889 [0.92307692 0.87254902 0.82105263 0.93939394]
Fscore:  0.8897 [0.94117647 0.8436019 0.82978723 0.94416244]

```

همانطور که میبینم عملکرد random forest در بهترین حالت برای پارامترهایی است که معیار entropy است و ماکسیمم عمق درخت ها ۱۰ باشد و هم چنین مینیمم داده های مورد نیاز برای هر node و هر برگ ۲ باشد. در این حالت accuracy مدل ۰.۸۹ میباشد که بهتر از عملکرد بهترین درخت در سوال ۸ و ۹ میباشد که برابر با ۰.۸۸ است.

در کل میتوان نتیجه گرفت که چه با پارامترهای پیش فرض چه با بهترین پارامترها که با grid search انتخاب میشوند نتایج مدل بهتر از نتایج مدل decision tree است. دلیل این مساله این است که random forest از گروه مدل های ensemble است که مجموعه ای از درخت ها است و هدف در این مدل random forest این است که مدل های مختلف (درخت های مختلف ساخته شده) با هم ترکیب شوند و نتیجه نهایی به گونه ای برگرفته از ترکیب عملکرد هر کدام از این درختها میباشد. این کار باعث میشود که خطای مدل در نهایت کمتر شود چرا که خطای هر یک از مدلها در مورد یک نمونه داده شده به گونه ای به عملکرد مدل های دیگر که برای آن نمونه خطای ندارند، پوشش داده میشود.

سوال ۱۳

درخت های تصمیم برخلاف بسیاری از روش های یادگیری ماشین نظریه یادگیری عمیق و شبکه های عصبی قابلیت شرح پذیری (interpretability) و تفسیر پذیری (explainability) دارند. اگر چه امروزه در بسیاری از مسائل یادگیری عمیق نتایج باور نکردنی ارائه می دهد و به همین خاطر از آن ها استفاده گسترده ای می شود اما نتایج و مدل های ارائه شده توسط آن برای انسان قابل فهم نیست (مفهوم جعبه سیاه یا black box). درخت های تصمیم اما مجموعه ای از قوانین if...then...else... می تولید می کند که برای انسان قابل درک بوده و نیاز به مفسر ندارند. این مساله بخصوص برای مدیران غیرفنی (non-technical) که نیاز به تصمیم گیری بر اساس نتایج مدل دارند اهمیت بیشتری پیدا می کند. البته درخت تصمیم و شبکه های عصبی شباهت هایی نیز دارند و آن قابلیت پردازش داده های غیرخطی و تعامل بین فیچرهای interactions می باشد.

سوال ۱۴

برخلاف بسیاری از روش های یادگیری ماشین که برای انسان قابل فهم نیستند روش های استخراج قوانین (rule induction) مانند IREP و RIPEER که بر اساس مجموعه قوانین (ruleset) عمل می کنند همانند درخت های تصمیم شرح پذیر هستند. بنابراین شباهت این روش ها و درخت های تصمیم در تولید قوانین و شرح پذیری است. اما برخلاف درخت های تصمیم مشکل درخت های overfitting را ندارند و سلسه مرتبی (hierarchical) نیستند. همچنین مدل های تولید شده توسط روش های استخراج قوانین فشرده تر (compact) هستند. همچنین قوانین تولید شده توسط این روش ها پیچیدگی کمتری داشته و به صورت

تکراری (recursive) و نه بازگشته (iterative) تولید می‌شوند. در روش‌های استخراج قوانین مانند بسیاری از روش‌های یادگیری ماشین داده‌ها به دو دسته آموزش و تست تقسیم شده و سپس بر روی داده‌های دسته اول آموزش داده می‌شوند. به عنوان نمونه روش IREP به شرح زیر آموزش داده می‌شود:

- تقسیم داده‌های آموزش به دو دسته برای رشد (growset) و هرس (pruneset) قوانین. مثلاً ۲/۳ برای دسته اول و ۱/۳ برای دسته دوم.
- رشد قوانین بر اساس افزایش information gain. در اینجا p_0 و n_0 تعداد داده مثبت و منفی برای قانون موجود و n_0 تعداد داده مثبت و منفی برای قانون جدید می‌باشد.

$$p_0 \left(\log_2 \left(\frac{p_1}{p_1 + n_1} \right) - \log_2 \left(\frac{p_0}{p_0 + n_0} \right) \right)$$

- هرس قوانین بر اساس کاهش خطای افزایش معیار هرس (pruning metric) مانند زیر:
- $$\frac{p + (N - n)}{P + N}$$

در روش RIPPER که پیچیده‌تر از روش IREP می‌باشد شرط توقف بر خلاف معیار هرس بر اساس تئوری ریاضی (theoretically-tight) و پیچیدگی اطلاعات (بر اساس بیت) تعریف می‌شود (مانند روش C4.5). در این روش قوانین با شرایط (conditionals) بیشتر پیچیده‌تر از قوانین با شرایط کمتر می‌باشد چرا که با مجموعه بزرگتر (larger pool) از قوانین برای انتخاب رویرو هستیم.

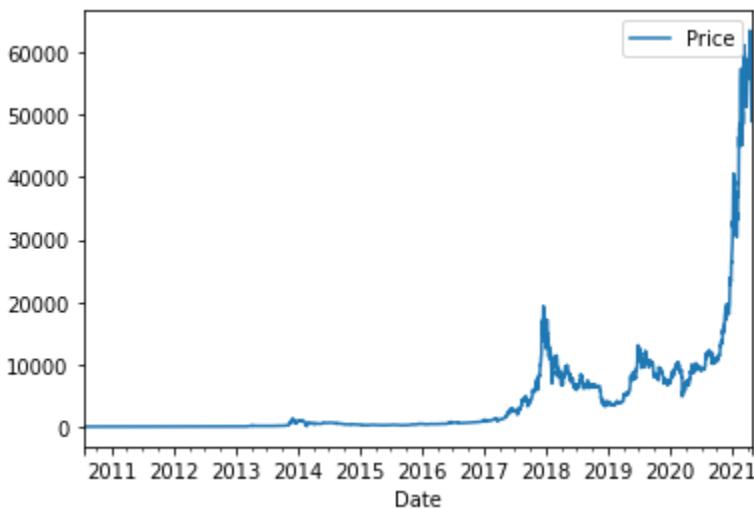
سوال ۱۵

برای مسائل زمانی (time-series) از روش‌های گوناگونی نظیر Moving average و ARIMA می‌توان بهره برد. اما روش‌های یادگیری ماشین نظیر رگرسیون خطی و درخت‌های تصمیم و مدل‌های ترکیبی Ensemble نظیر Bagging و Boosting نیز استقاده کرد. ایده کلی برای استقاده از درخت‌های تصمیم در حل مسائل زمانی این است که در ابتدا داده‌های را به دو دسته گذشته (past) و آینده (future) تقسیم کرده و سپس داده‌های گذشته را به فرم جدولی با سطرهایی به تعداد داده‌ها و ستون‌های زمانی t_1, t_2, \dots, t_n در بیاوریم. سپس از روش‌های پنجه‌ای (window-based) با اندازه‌های مختلف برای مهندسی و استخراج ویژگی بهره برد تا بتوان ویژگی‌های منحصر بفرد داده‌های زمانی نظیر trend و seasonality را حفظ کرد. پس از تبدیل فرم داده به روش بالا اکنون می‌توان از درخت تصمیم برای یادگیری داده‌های زمانی استقاده کرد. نکته‌ای که باید به آن توجه کرد اینست که درخت تصمیم نباید اولین انتخاب برای داده‌های زمانی باشد چرا که دقت کمتری برای اینگونه مسائل دارند اگرچه تفسیر پذیری بالای آن‌ها همچنان می‌تواند دلیلی برای انتخاب آن‌ها باشد.

سوال ۱۶

داده بیت کوین را می‌گیریم و داده‌های قبل از تاریخ ۱۱/۲۰/۲۰ را به عنوان train و داده‌های بعد از آن را به عنوان تست استقاده می‌کنیم. دقت کنیم که در سوالهای آینده تنها از مقدار فیچر price برای آموزش مدل استقاده می‌کنیم و هم‌چنین هدف نهایی پیش‌بینی مقدار این فیچر price در آینده است (مگر به جز این ذکر شده باشد که از فیچرهای دیگر نیز استقاده کنیم). در زیر مقدار‌های مختلف price طی زمان می‌بینیم.

train size: 3455, test size: 486, all: 3941



سوال ۱۷

برای این سوال مدل‌های زیر را انتخاب کردیم:

1. Autoregression (AR)
2. Moving Average
3. Prophet: Facebook model for time series prediction
4. Autoregressive Integrated Moving Average (ARIMA)
5. Simple Exponential Smoothing (SES)
6. Holt Winter's Exponential Smoothing (HWES)
7. LSTM
8. Linear Regression
9. Ridge Linear Regression
10. Seasonal Autoregressive Integrated Moving-Average (SARIMA)
11. Decision Tree Regressor

برای آموزش تمامی مدل‌ها به جز مدل‌های Linear Regression, Ridge Linear Regression, Decision Tree REgressor , LSTM •

در ابتدا تمامی داده‌های train را برای آموزش مدل استفاده کردیم. سپس برای هر داده test مدل را ارزیابی

کرده و سپس داده تست را به داده‌های train اضافه کردیم و مدل را دوباره آموزش دادیم

model. دلیل این کار این است که مدل همواره از داده‌هایی که دیده است استفاده کند تا عملکردش بهبود یابد.

در غیر این صورت عملکرد مدل به شدت بد می‌شود مدل بعد از یک مدتی پیش بینی یکسان برای تمامی زمان‌ها

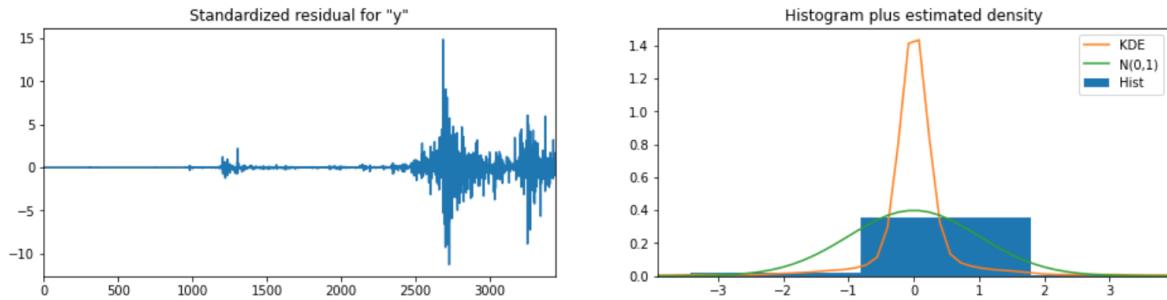
خواهد داشت و یا به عبارتی flat prediction. به خصوص در مورد داده‌هایی مانند قیمت بیت‌کوین که به

سرعت تغییر قیمت ناگهان و چشم‌گیر داشته‌اند اصلاً خوب عمل نمی‌کند! به طور مثال در زیر عملکرد مدل

برای مدل ARIMA را بدون retrain کردن مدل می‌بینیم:

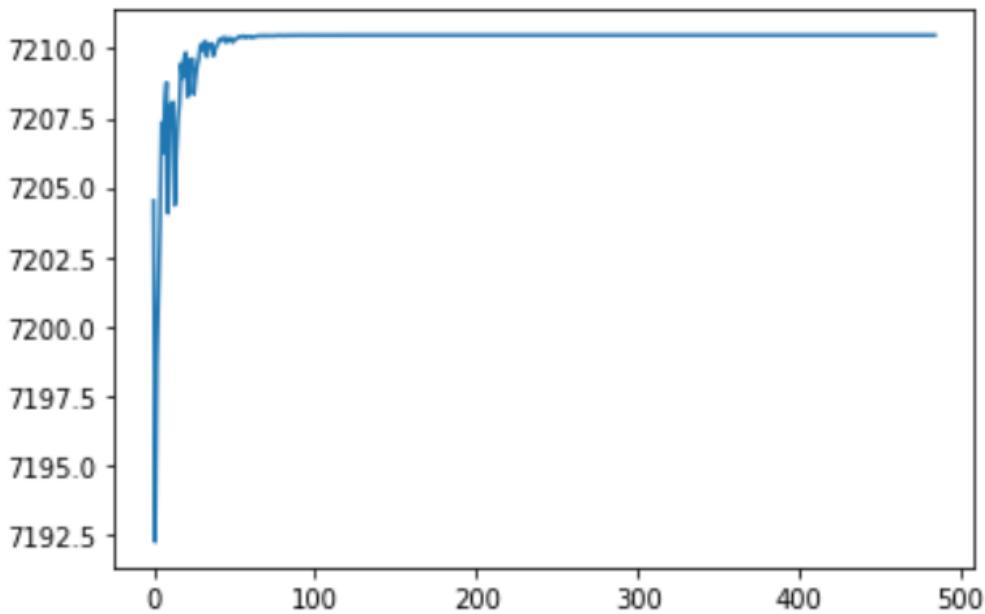
همانطور که می‌بینیم خطای داده تست بسیار زیاد شده است(نمودار سمت چپ) و هم چنین خطای پیش بینی ها

دیگر از توزیع نمایی پیروی نمی‌کند (نمودار راست) این نشان میدهد عملکرد مدل به شدت کاهش پیدا کرده است:



هم چنین میزان پیش بینی مدل برای داده تست را در زیر میبینیم که از یک جایی به بعد بر روی عدد ۷۲۰۰ تقریباً ثابت مانده است. در حالی که قیمت بیت کوین در واقعیت در داده تست تا به ۴۰۰۰ هم رسید. در نمودار زیر flat prediction به خوبی قابل مشاهده است

[<matplotlib.lines.Line2D at 0x7f07664b4b50>]



برای آموزش مدل‌های Linear Regression, Ridge Linear Regression, Decision Tree REgressor • به روشنی زیر عمل کردیم:

- این مدلها برای آموزش نیاز به فیچر دارند. برای داده هر روز فیچر در این مدلها در واقع قیمت بیت کوین در روز قبل است. به این ترتیب برچسبی که باید پیش بینی کنیم قیمت بیت کوین در امروز و فیچر در واقع قیمت دیروز است. برای آموزش این مدل از کل داده ترین استفاده میکنیم و برای ارزیابی نیز از داده تست. تفاوت این روش با روشی که برای مدل‌های قبلی به کار برده این است که دیگر مدل با دیدن داده های تست هر بار retrain نمیشود ولی چون فیچر قیمت روز قبل است، این روش نیز مانند روش قبلی به نوعی از تمامی داده های روز قبل برای پیش بینی قیمت استفاده میکند.

در زیر عملکرد هر کدام از ۱۱ مدل را میبینیم. خطای RMSE و Accuracy with 5% band را برای داده ترین و تست گزارش کردیم. هم چنین عملکرد مدل بر روی هر کدام از داده های ترین و تست و کل دیتا نمایش دادیم(به ترتیب از چپ به راست).

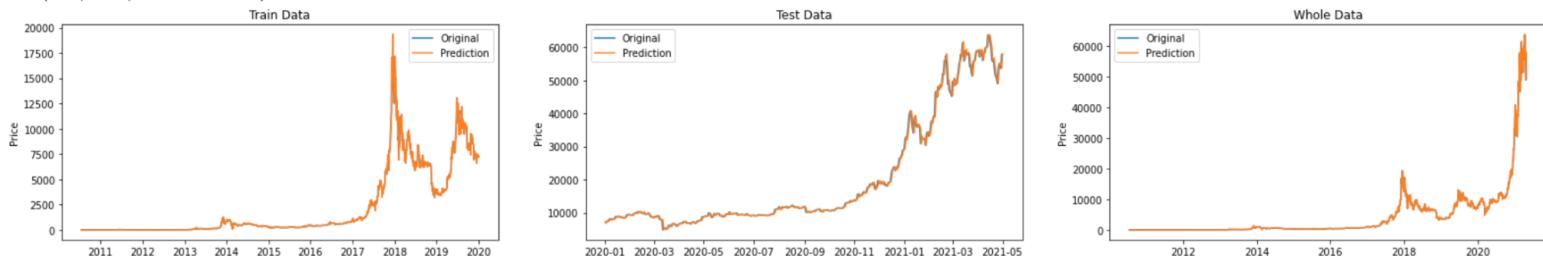
خطوط آبی رنگ مقادیر واقعی طی زمان هستند و خطوط نارنجی مقادیر پیش بینی شده توسط مدل هستند. هرچه اختلاف بین این دو کمتر باشد عملکرد مدل بهتر است.

Autoregression (AR)

- این مدل مقدار مرحله بعدی را به شکل تابع خطی از مشاهدات مراحل قبلی محاسبه میکند
- این روش برای تایم سری هایی تکی (در واقع فقط یک تایم سری باشد) و univariate و بدون ترند و رفتار های فصلی مناسب است.

همانطور که میبینیم RMSE برابر با 20.4 و 10.84 برای ترین و تست است. هم چنین Accuracy برابر با 58% و 84% برای ترین و تست است. RMSE برای داده تست بالاتر از ترین است و دلیل این است که کوچکترین تفاوت بین مقدار پیش بینی شده و واقعی باعث زیاد شدن مقدار خطا میشود در حالی که اگر به دقت نگاه کنیم 84% درصد است و نشان میدهد در 84% موقع داده تست پیش بینی شده در بازی 95 تا 105 درصدی مقدار واقعی بوده است.

```
train error: 204.46173709728777, test error: 1084.8582458258516
2030 3454 3454
413 486 486
train accuracy 5%: 58.77243775332948, test accuracy 5%: 84.97942386831275
Text(0.5, 1.0, 'Whole Data')
```



Moving Average

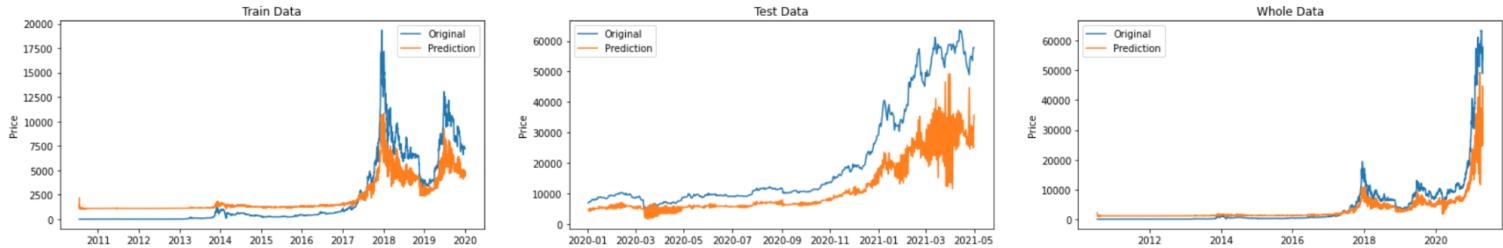
- این مدل مقدار مرحله بعدی را به شکل تابع خطی از خطای پرسه میانگین گیری در مراحل قبلی محاسبه میکند
- این روش برای تایم سری هایی تکی (در واقع فقط یک تایم سری باشد) و univariate و بدون ترند و رفتار های فصلی مناسب است.

همانطور که میبینیم RMSE برابر با 17.86 و 12.925 برای ترین و تست است. هم چنین Accuracy برابر با 10.5% و 10.3% برای ترین و تست است. RMSE برای داده تست بالاتر از ترین است و دلیل این است که کوچکترین تفاوت بین مقدار پیش بینی شده و واقعی باعث زیاد شدن مقدار خطا میشود و اگر به دقت نگاه کنیم 10.3% درصد است و نشان میدهد در 1% موقع داده تست پیش بینی شده در بازی 95 تا 105 درصدی مقدار واقعی بوده است. هرچند مدل توانسته است ترند قیمت را به درستی پیش بینی کند ولی مقادیر پیش بینی شده تفاوت زیادی با مقدار اصلی دارند.

```

train error: 1786.3782441068747, test error: 12925.85145114704
36 3455 3455
5 486 486
train accuracy 5%: 1.0419681620839363, test accuracy 5%: 1.02880658436214
Text(0.5, 1.0, 'Whole Data')

```



Prophet: Facebook model for time series prediction

- این روش توسط شرکت فیس بوک به شکل open source library به اشتراک گذاشته شده است و
- این روش برای پیش بینی تایم سری ها هست و ترند ها و رفتار های فصلی و تعطیلات را ساپورت می کند.
- این روش برای تایم سری هایی تکی (در واقع فقط یک تایم سری باشد) و univariate مناسب است.

Resource:

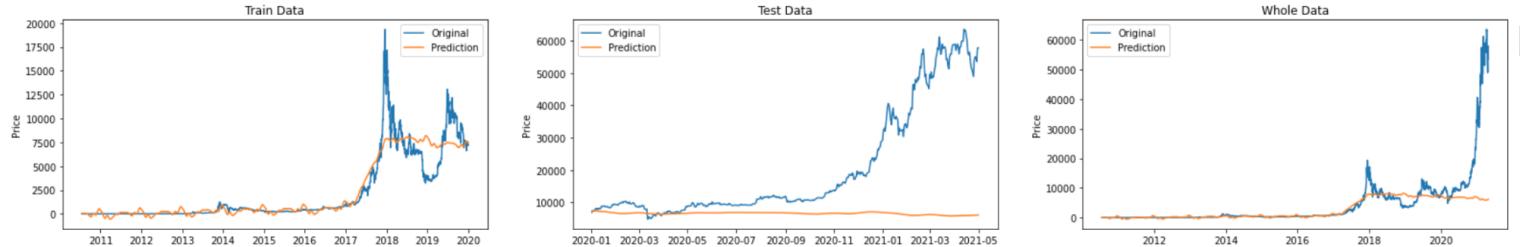
<https://machinelearningmastery.com/time-series-forecasting-with-prophet-in-pytorch/>

همانطور که می بینیم RMSE برابر با ۱۴۳۲ و ۲۲۲۴۶ برای ترین و تست است. هم چنین Accuracy برابر با ۹۶.۳۴ و ۹۳.۵٪ برای ترین و تست است. مدل در کل برای ترین و تست نتوانسته است عالی عمل کند. حتی بر روی داده ترین در زمان های اخیر عملکرد به شدت بد شده است و نتوانسته جهش ناگهانی در قیمت را پیش بینی کند. برای داده تست وضعیت بدتر است و مقادیر به شکل خط flat پیش بینی شده است.

```

train error: 1432.8616850082872, test error: 22246.027632499416
219 3455 3455
17 486 486
train accuracy 5%: 6.338639652677279, test accuracy 5%: 3.4979423868312756
Text(0.5, 1.0, 'Whole Data')

```

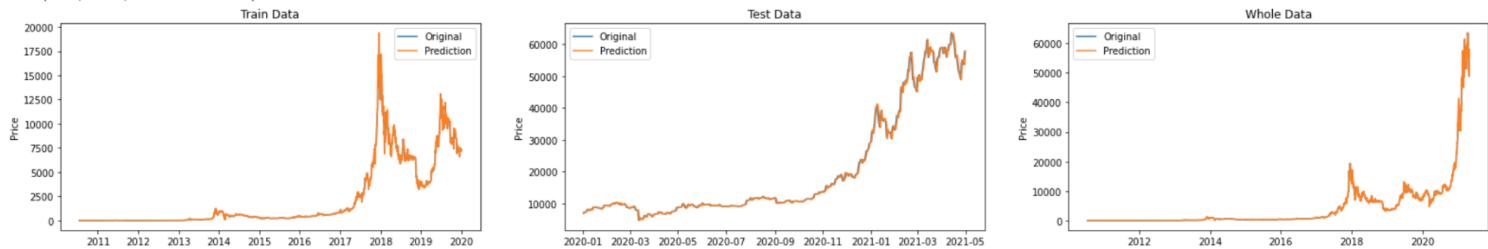


Autoregressive Integrated Moving Average (ARIMA)

- این مدل مقدار مرحله بعدی را به شکل تابع خطی از تفاوت مشاهدات و خطاهای در مراحل قبلی محاسبه می کند
- این روش برای تایم سری هایی تکی (در واقع فقط یک تایم سری باشد) و univariate و با ترند ولی بدون رفتار های فصلی مناسب است.
- از (order = 1,1,1) استفاده کردیم.

همانطور که میبینیم RMSE برابر با 203.9 و 1090.88 برای ترین و تست است. هم چنین Accuracy برابر با 81% و 84.9% برای ترین و تست است. RMSE برای داده تست بالاتر از ترین است و دلیل این است که کوچکترین نکلوت بین مقدار پیش بینی شده و واقعی باعث زیاد شدن مقدار خطأ میشود در حالی که اگر به دقت نگاه کنیم 84% درصد است و نشان میدهد در 84% موقع داده تست پیش بینی شده در بازی 95 تا 105 درصدی مقدار واقعی بوده است. عملکرد مدل از روش AR (روش یک) بهتر است.

```
train error: 203.92150351047783, test error: 1090.8758581140335
2811 3455 3455
413 486 486
train accuracy 5%: 81.3603473227207, test accuracy 5%: 84.97942386831275
Text(0.5, 1.0, 'Whole Data')
```

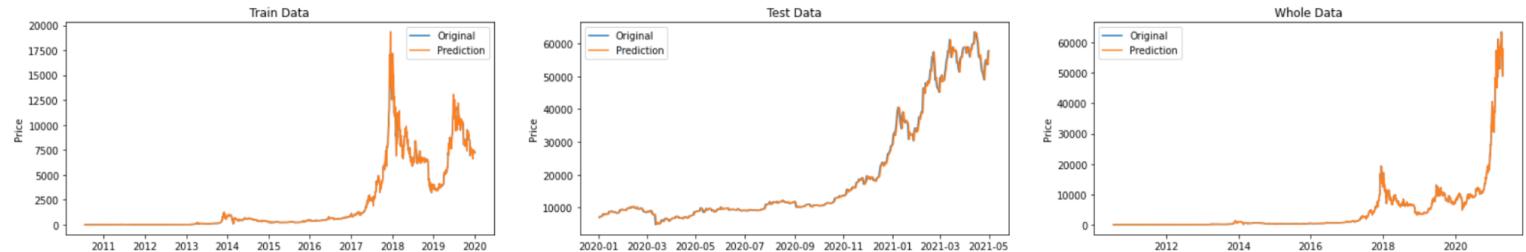


Simple Exponential Smoothing (SES)

- این مدل مقدار مرحله بعدی را به شکل تابع خطی از exponentially weighted مشاهدات قبلی محاسبه میکند
- این روش برای تایم سری هایی تکی (در واقع فقط یک تایم سری باشد) و univariate و بدون تренд و بدون رفتار های فصلی مناسب است.

همانطور که میبینیم RMSE برابر با 204.5 و 1082.6 برای ترین و تست است. هم چنین Accuracy برابر با 81% و 85.4% برای ترین و تست است.

```
train error: 204.53847752459498, test error: 1082.601079960296
2813 3454 3454
415 486 486
train accuracy 5%: 81.44180660104226, test accuracy 5%: 85.39094650205762
Text(0.5, 1.0, 'Whole Data')
```

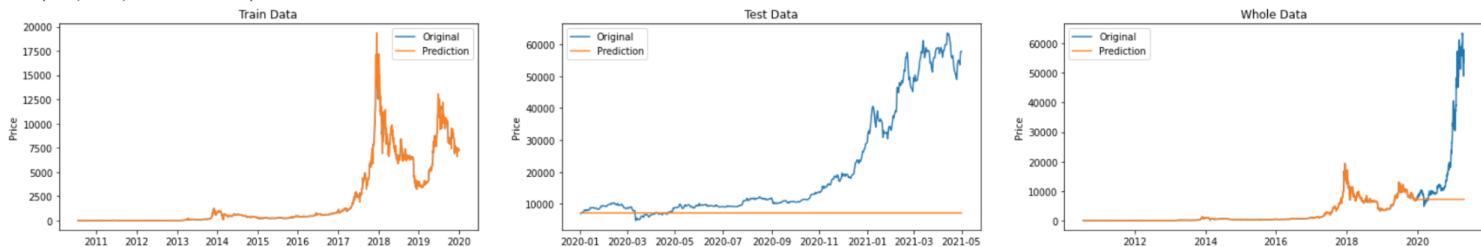


Holt Winter's Exponential Smoothing (HWES)

- این مدل مقدار مرحله بعدی را به شکل تابع خطی از exponentially weighted مشاهدات قبلی محاسبه میکند و هم چنین تрендها و رفتار های فصلی را نیز در نظر میگیرد.
- این روش برای تایم سری هایی تکی (در واقع فقط یک تایم سری باشد) و univariate و با تренд و بدون و یا همراه رفتار های فصلی مناسب است.

همانند روش قبل، همانطور که میبینیم RMSE برابر با ۲۰۴.۵ و ۲۱۶۸۳ برای ترین و تست است. هم چنین Accuracy برابر با ۸۱٪ و ۴.۷٪ برای ترین و تست است. همانطور که از شکل هم مشخص است برای داده تست مدل از اول نتوانسته است پیش بینی خوبی داشته باشد و یک خط صاف پیش بینی کرده است

```
train error: 204.5754729759361, test error: 21683.516740373798
2812 3454 3454
23 486 486
train accuracy 5%: 81.4128546612623, test accuracy 5%: 4.732510288065844
Text(0.5, 1.0, 'Whole Data')
```

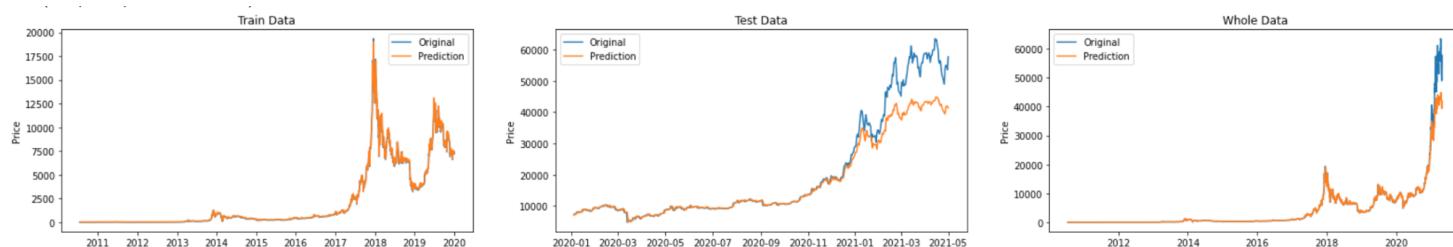


LSTM

مدل ۴ لایه LSTM میباشد. همانطور که توضیح دادیم فیچر هر داده در واقع قیمت روز قبل است و برچسبی که باید پیش بینی شود قیمت امروز است. در این مدل از adam optimizer استفاده کردیم و loss function mean square error است. اندازه هر batch برابر با یک داده است و تعداد اپیک ها ۵۰ است. هم چنین داده را قبل از جدا کردن ترین و تست با روش minmax نرمالایز میکنیم.

همانطور که میبینیم RMSE برابر با ۲۰۷.۸۹ و ۵۵۶۵ برای ترین و تست است. هم چنین Accuracy برابر با ۳۳٪ و ۶۵٪ برای ترین و تست است. این مدل نتوانسته است که ترند قیمت در داده های تست را به خوبی پیش بینی کند ولی مقداری که پیش بینی کرده در روز های اخر داده تست تفاوت زیادی با مقادیر اصلی دارد.

```
Train Score: 207.89 RMSE
Test Score: 5565.06 RMSE
train error: 207.89033076277784, test error: 5565.060309229974
train accuracy 5%: 33.10165073848827, test accuracy 5%: 65.08264462809917
```



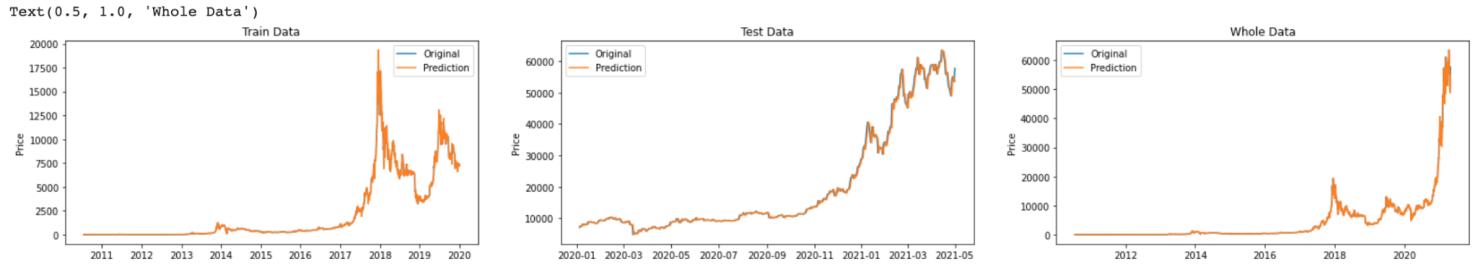
Linear Regression

در این روش داده را قبل جدا کردن ترین و تست با روش minmax نرمالایز میکنیم. همانطور که میبینیم RMSE برابر با ۲۰۴.۵ و ۱۰۸۸ برای ترین و تست است. هم چنین Accuracy برابر با ۵۸.۷٪ و ۸۴.۵٪ برای ترین و تست است. عملکرد مدل بهتر از مدل دیپ لرنینگ LSTM است. دلیل این است که در حالت LSTM مدل به نسبت تعداد داده ها زیادی پیچیده بود و به علت کم بودن تعداد داده ها مدل نتوانسته بود به خوبی ترین شود.

```

train error: 204.49128512488338, test error: 1088.7592811272946
2029 3453 3453
409 484 484
train accuracy 5%: 58.76049811757892, test accuracy 5%: 84.50413223140497

```



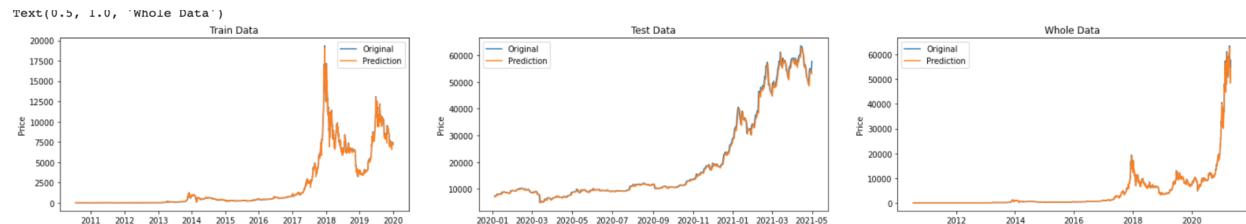
Ridge Linear Regression

در این روش داده را قبل جدا کردن ترین و تست با روش **minmax** نرمالایز میکنیم. هم چنین مقدار ضریب رگولايزر و یا همان پارامتر آلفا را برابر با 1.0×10^{-6} قرار میدهیم.
همانطور که میبینیم **RMSE** برابر با 207.2 و 1145.21 برای ترین و تست است. هم چنین **Accuracy** برابر با 42% و 82.6% برای ترین و تست است. این روش از **linear regression** معمولی بهتر عمل کرده است چرا که رگولايزر باعث جلوگیری از **overfitting** مدل میشود.

```

train error: 207.22415607055342, test error: 1145.212748167857
1461 3453 3453
400 484 484
train accuracy 5%: 42.3110338835795, test accuracy 5%: 82.64462809917356

```



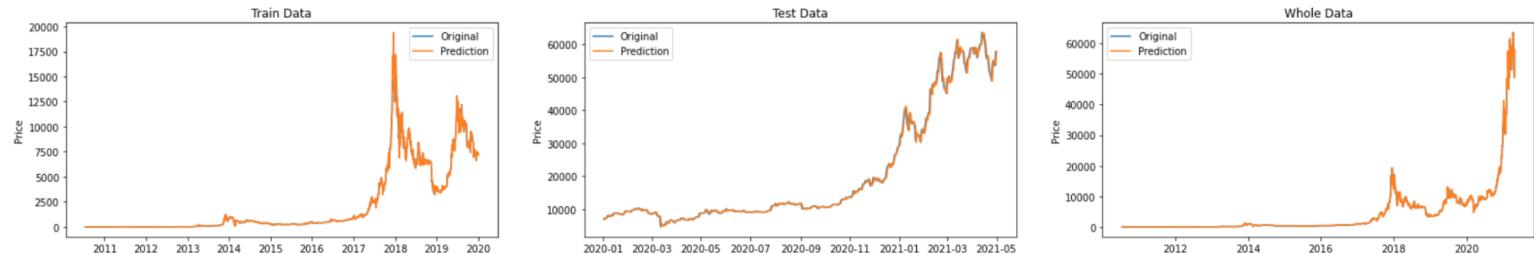
Seasonal Autoregressive Integrated Moving-Average (SARIMA)

- این مدل مقدار مرحله بعدی را به شکل تابع خطی از مشاهدات قبلی خطاهای و تفاوت‌های مشاهدات رفتارهای فصلی در زمانهای قبلی محاسبه میکند.
- مدل **arima** را با توانی انجام **autoregression** و **moving average** ترکیب میکند.
- این روش برای تایم سری هایی تکی (در واقع فقط یک تایم سری باشد) و **univariate** و با تренд و بدون و یا همراه رفتارهای فصلی مناسب است.

- Order = (1,1,1) & seasonal = (0,0,0,0)

همانطور که میبینیم RMSE برابر با 203.9 و 1090.8 برای ترین و تست است. هم چنین Accuracy برابر با 81% و 84.9% برای ترین و تست است. این روش نسبت به باقی روش‌های آماری Arima, exponential timeseries prediction مانند smoothing و ... بهتر عمل کرده است.

```
train error: 203.92150351047783, test error: 1090.8758581140335
2811 3455 3455
413 486 486
train accuracy 5%: 81.3603473227207, test accuracy 5%: 84.97942386831275
Text(0.5, 1.0, 'Whole Data')
```



Decision Tree Regressor

دیتا را قبل از جدا کردن ترین و تست نرمالایز میکنیم. همچنین با روش grid search بهترین مقدارهای پارامترها را پیدا میکنیم. پارامترهای زیر را با مقادیر زیر امتحان کردیم:

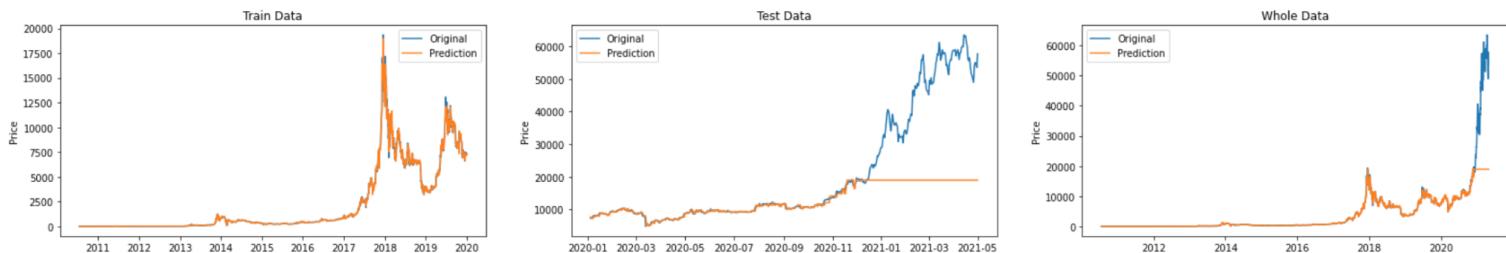
- 'criterion' : ['mse', 'mae'],
- 'max_depth' : [3, 5, 7, 10],
- 'min_samples_split' : range(2, 10, 1),
- 'min_samples_leaf' : range(2, 10, 1)

نهایتاً بهترین درخت با بهترین عملکرد با پارامترهای زیر بود:

```
'criterion': 'mae', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 2
```

همانطور که میبینیم RMSE برابر با 184.9 و 15412.44 برای ترین و تست است. هم چنین Accuracy برابر با 81% و 58.67% برای ترین و تست است. همانطور که از شکل هم مشخص است برای داده تست مدل از در داده های اخیر نتوانسته است پیش بینی خوبی داشته باشد و یک خط صاف پیش بینی کرده است. این مدل برای داده ترین خیلی خوب عمل میکند ولی برای داده تست عملکرد خوبی ندارد. یک دلیل این رفتار این overfitting مدل به دیتا ترین است. امکان در مدلها با پایه درخت بیشتر از باقی مدلها است.

```
train error: 184.98367025087512, test error: 15412.440433727801
2826 3453 3453
284 484 484
train accuracy 5%: 81.84187662901824, test accuracy 5%: 58.67768595041323
```



سوال ۱۸

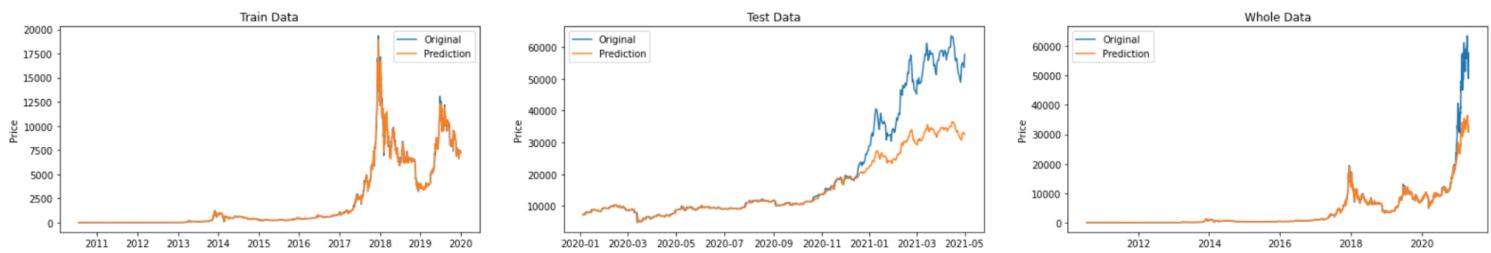
Voting

سه مدل درخت با پارامترهای مختلف را انتخاب میکنیم و با روش voting نتایج آنها را ترکیب میکنیم:

```
reg1 = DecisionTreeRegressor(random_state=0, criterion= 'mae', max_depth=5, min_samples_leaf =5, min_samples_split= 2)
reg2 = DecisionTreeRegressor(random_state=0, criterion= 'mae', max_depth=10, min_samples_leaf =5, min_samples_split= 2)
reg3 = DecisionTreeRegressor(random_state=0, criterion= 'mse', max_depth=10, min_samples_leaf =5, min_samples_split= 2)
```

که یکی از این درختها درخت با بهترین پارامتر بود که در سوال قبلی پیدا کردیم. نتایج در زیر آمده است. همانطور که مشاهده میکنیم عملکرد مدل در داده تست بهتر از یک درخت (بهترین درخت) است که عملکردش در سوال ۱۸ گزارش شده بود! (دقت ۵۸٪ در مقابل دقต ۶۱٪ در داده تست). هم چنین همانطور که از نمودارها میبینیم ترند پیش بینی شده بسیار بهتر از ترند پیش بینی شده توسط فقط یک درخت است (سوال ۱۷). خالتی که فقط یک درخت داشتیم در سوال ۱۷ مدل از یک جایی به بعد در داده تست تنها یک خط صاف پیش بینی کرده بود ولی در این حالت توانسته است ترند را تا حد خوبی پیش بینی کند

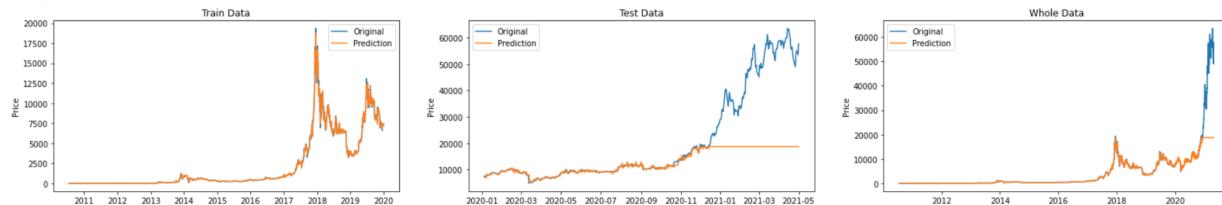
```
train error: 185.96497160937074, test error: 9440.73212461816
1947 3453 3453
297 484 484
train accuracy 5%: 56.385751520417024, test accuracy 5%: 61.36363636363637
```



Bagging

مدل پایه انتخاب شده در این روش درخت است. تعداد مدلها را ۵۰ گذاشتیم که بهترین مقدار پارامتر بود. برای مقادیر مدلها بیشتر و یا کمتر از این مقدار عملکرد مدل بدتر میشد!! در زیر عملکرد را میبینیم: عملکرد نسبت به حالت voting بدتر شده است. هم چنین مدل نهایی بر خلاف روش voting نتوانسته است ترند را پیش بینی کند و در زمانهای اخیر در داده تست تنها یک خط صاف پیش بینی کرده است. پس هم از نظر دقّت و خطأ و هم از نظر خود تایم سری پیش بینی شده روش voting بهتر از bagging عمل کرده است.

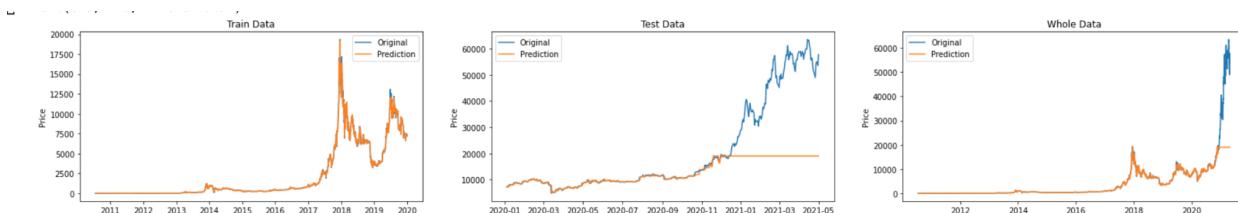
```
train error: 96.44827605908856, test error: 15508.654800969683
2177 3453 3453
252 484 484
train accuracy 5%: 63.046626122212565, test accuracy 5%: 52.066115702479344
```



Boosting

این روش از حالت **bagging** کمی بهتر عمل کرده است. تعداد مدلها را 100 انتخاب کردیم چرا که برای تعداد مدل‌های کمتر و یا بیشتر عملکرد نهایی مدل بدتر نمی‌شد. هم چنین مدل نهایی برخلاف روش **voting** نتوانسته است تренд را پیش‌بینی کند و در زمانهای اخیر در داده تست تنها یک خط صاف پیش‌بینی کرده است. پس هم از نظر دقت و خطأ و هم از نظر خود تایم سری پیش‌بینی شده روشن bagging و boosting عمل کرده است. و هم چنین روشن voting بهتر از bagging بوده است.

```
train error: 171.35724745352337, test error: 15376.594234692775
2501 3453 3453
290 484 484
train accuracy 5%: 72.42977121343759, test accuracy 5%: 59.917355371900825
```



سوال ۱۹

همانند سوال قبل از جدا سازی ترین و تست نرمالایز کردیم. سپس با استفاده از روش **grid search** بهترین پارامترها را پیدا می‌کنیم. پارامترهای زیر با مقادیر زیر را امتحان کردیم. loss function برای این مدل mean absolute error بود

```
grid_params = {
    'n_estimators' : [10, 40, 80, 100],
    'learning_rate' : [0.01, 0.1, 1],
    'loss': ['linear', 'square'],
    'random_state' : [0],
    'base_estimator': [DecisionTreeRegressor(criterion= 'mae', max_depth=10, min_samples_leaf =5, min_samples_split= 2), ]}
```

در نهایت بهترین عملکرد برای پارامترها با مقادیر زیر بود. هم چنین عملکرد مدل در بهترین حالت گزارش شده است:

```

Grid Search Best Parameters
{'base_estimator': DecisionTreeRegressor(ccp_alpha=0.0, criterion='mae', max_depth=10,
                                         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=None, splitter='best'), 'learning_rate': 0.01, 'loss': 'square', 'n_estimators': 80, 'random_state': 0}

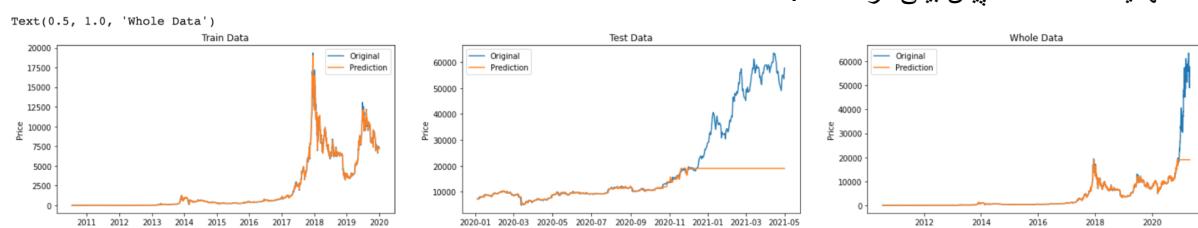
AdaBoost Results with Best Estimator
train error: 163.13972913464605, test error: 15389.877389562414
2874 3453 3453
294 484 484
train accuracy 5%: 83.23197219808863, test accuracy 5%: 60.74380165289256

```

هم چنین جدول زیر عملکرد مدل‌های مختلف با پارامترهای مختلف را نشان میدهد. با توجه به پارامترها و مقادیر امتحان شده در کل ۲۴ مدل داشتیم:

	mean_test_score	param_loss	param_n_estimators	param_learning_rate
0	0.849759	linear	10	0.01
1	0.848306	linear	40	0.01
2	0.850270	linear	80	0.01
3	0.849723	linear	100	0.01
4	0.850107	square	10	0.01
5	0.850159	square	40	0.01
6	0.851057	square	80	0.01
7	0.851016	square	100	0.01
8	0.847570	linear	10	0.1
9	0.833040	linear	40	0.1
10	0.815857	linear	80	0.1
11	0.808251	linear	100	0.1
12	0.837681	square	10	0.1
13	0.835739	square	40	0.1
14	0.831692	square	80	0.1
15	0.832729	square	100	0.1
16	0.825285	linear	10	1
17	-0.567398	linear	40	1
18	-129.949458	linear	80	1
19	-129.954789	linear	100	1
20	0.833410	square	10	1
21	0.400130	square	40	1
22	0.151370	square	80	1
23	0.001119	square	100	1

نهایتاً سری زمانی پیش بینی شده با مدل به شکل زیر است. مدل نتوانسته است ترند را پیش بینی کند و در زمانهای اخیر در داده تست تنها یک خط صاف پیش بینی کرده است.



Random Forest

سوال ۲۰

همانند سوال قبل دیتا را قبل از جدا سازی ترین و تست نرمالایز کردیم. سپس با استفاده از روش grid search بهترین پارامترها را پیدا میکنیم. پارامترهای زیر با مقادیر زیر را امتحان کردیم.

```
grid_params = {
    'criterion' : ['mse', 'mae'],
    'max_depth' : [3, 5, 7, 10],
    'min_samples_split' : range(2, 10, 2),
    'min_samples_leaf' : range(2, 10, 2)
}
```

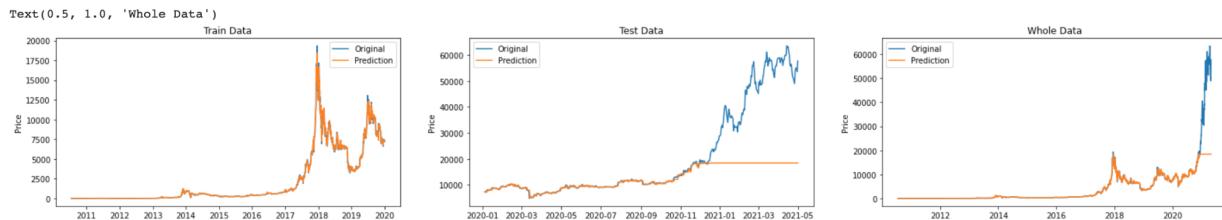
در نهایت بهترین عملکرد برای پارامترها با مقادیر زیر بود. هم چنین عملکرد مدل در بهترین حالت گزارش شده است. مدل Adaboost در سوال ۱۹ اندکی بهتر از مدل random forest در این سوال عمل کرده است (دقت ۵۸٪ در مقابل دقت ۶۰٪).

```
Grid Search best parameters
{'criterion': 'mae', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 6}
Random Forest Results with Best Estimator
train error: 162.52432129318845, test error: 15663.870053356814
2918 3453 3453
284 484 484
train accuracy 5%: 84.50622646973646, test accuracy 5%: 58.67768595041323
```

هم چنین جدول زیر عملکرد مدل‌های مختلف را نشان میدهد. با توجه به پارامترها و مقادیر امتحان شده در کل ۲۵ مدل داشتیم که در زیر نتایج ۲۵ حالت را به طور نمونه گزارش کردیم:

	mean_test_score	param_criterion	param_max_depth	param_min_samples_split	param_min_samples_leaf
0	-448.097142	mse	3	2	2
1	-450.255570	mse	3	4	2
2	-448.973224	mse	3	6	2
3	-448.536742	mse	3	8	2
4	-448.357140	mse	3	2	4
5	-450.663251	mse	3	4	4
6	-448.455088	mse	3	6	4
7	-446.974320	mse	3	8	4
8	-446.265050	mse	3	2	6
9	-448.324854	mse	3	4	6
10	-447.591566	mse	3	6	6
11	-447.020613	mse	3	8	6
12	-446.862966	mse	3	2	8
13	-450.666431	mse	3	4	8
14	-450.074167	mse	3	6	8
15	-445.583462	mse	3	8	8
16	-0.351946	mse	5	2	2
17	-0.329615	mse	5	4	2
18	-0.372577	mse	5	6	2
19	-0.371389	mse	5	8	2
20	-0.350451	mse	5	2	4
21	-0.370794	mse	5	4	4
22	-0.346832	mse	5	6	4
23	-0.366723	mse	5	8	4
24	-0.367264	mse	5	2	6

نهایتاً سری زمانی پیش بینی شده با مدل به شکل زیر است. مدل نتوانسته است ترند را پیش بینی کند و در زمانهای اخیر در داده تست تنها یک خط صاف پیش بینی کرده است.



سوال ۲۱

نتایج مدل‌های مختلف در سوالهای ۱۷ تا ۲۰ در برای هر سوال در بالا بیان شده است.

سوال ۲۲

در این بخش در ابتدا ستون تغییر قیمت (Change %) را به باینری تبدیل کرده تا برای استفاده در مساله کلاسه بندی آماده شود. برای این کار عده‌های منفی را برابر صفر و عده‌های مثبت را برابر ۱ در نظر گرفته و تمام ستون را به عنوان برچسب (label) کلاسه بندی در نظر می‌گیریم. سپس مدل LSTM را بر روی دیتاست جدید ترین می‌کنیم. همانطور که در نوت بوک دیده می‌شود دقت کلاسه بندی ۳۳.۲۶٪ می‌باشد. همچنین معیارهای دیگر برای دو کلاس ۰ و ۱ به ترتیب به شرح زیر می‌باشد:

Precision: 0.72, 0.17

Recall: 0.27, 0.58

F1: 0.39, 0.26

سوال ۲۳

در این سوال ستون تغییر یافته قیمت در سوال قبل را به عنوان فیچر به دیتاست اضافه کرده و مدل رگرسیون جدید را با LSTM ترین کرده و قیمت را بر روی داده تست تخمین می‌زنیم. RMSE برابر ۵۷۵۰.۵ می‌شود. همچنین خطای محدوده ۰.۰۵ برابر با ۵۷٪ می‌باشد. با مقایسه با سوال ۱۷ (به ترتیب ۵۵۶۵ و ۴۵.۰۸٪) ملاحظه می‌شود که جواب بدتر می‌شود. اما بر روی داده ترین جواب بهتر می‌شود (به ترتیب ۱۴۹.۳۶ و ۳۵.۲۷٪ برای این سوال و ۲۰.۷ و ۳۳.۱٪ برای سوال ۱۷) که نشان دهنده overfitting می‌باشد.

سوال ۲۴

دیتاست را همانند سوال قبل آماده کرده و از مدل‌های رگرسیون متفاوت شامل Ensemble Learning برای تخمین قیمت استفاده کرده و به ترتیب MSE و RMSE را گزارش می‌کنیم. مدل‌های استفاده شده به شرح زیر است:

XGB: اساساً مانند Extreme Gradient Boosting بوده با این تفاوت که برای مساله رگرسیون استفاده می‌شود. مدل‌های پیشگو (predictor) به ترتیب و بصورت یکی یکی (sequential) به مدل نهایی اضافه می‌شود به گونه‌ای که مدل جدید بر روی خطای باقیمانده از مدل قبلی فیت می‌شود.

LR: همان رگرسیون خطی (Linear Regression) است.

DT: ساخت درخت تصمیم‌گیری و تقسیم‌بندی شامل طرح یک سری سوال بوده اما با این تفاوت که در اینجا بر اساس MSE عمل می‌شود. به این ترتیب که در حین تصمیم‌گیری و تقسیم‌بندی به چند مجموعه (subset) مجموعه‌ای انتخاب می‌شود که دارای کمترین MSE باشد.

RF: همچنان که درخت تصمیم‌گیری برای مساله کلاسیفیکی از مشکل overfitting رنج می‌برد در مساله رگرسیون نیز این مشکل وجود دارد. برای حل این مشکل می‌توان از Ensemble Learning و به صورت خاص Random Forest استفاده کرد. این روش یک روش Bagging بوده که در آن جنگل‌ها (forests) که مجموعه‌ای از درخت‌های تصمیم‌گیری هستند به صورت موازی (parallel) ساخته شده و در حین ساخت هیچ تقابلی با هم ندارند. درخت‌ها بر روی زیرمجموعه‌های تصادفی ساخته می‌شوند و همین امر باعث پیشگیری از overfitting می‌شود.

BAG: روش BAGGING که همانند روش قبیل یعنی RF دارای مدل‌های پایه با ساختار مشابه است. در اینجا از SVM به عنوان مدل پایه استفاده می‌کنیم.

VOT: روش VOT که برای آن از دو مدل LR و RF استفاده می‌کنیم (مدل‌های با ساختار مقاومت).

EXT: همانند روش RF بوده با این تفاوت که از الگوریتم ساده‌تری برای ساخت درخت‌های تصمیم‌گیری استفاده می‌کند. به این ترتیب که درخت‌های تصمیم‌گیری بدون حرسر (unpruned) را بر روی تمامی داده‌های ترین (برخلاف RF که درخت‌ها را بر روی زیرمجموعه‌های تصادفی از داده‌های ترین می‌سازد) می‌سازد و تقسیم راس‌ها نیز کاملاً تصادفی انتخاب می‌افتد (RF از بهینه‌سازی استفاده می‌کند). در نهایت برای مساله رگرسیون میانگین پیش‌بینی از مدل‌های مختلف را حساب می‌کند.

ADA: روش ADABOOST که مانند RF بوده با این تفاوت که مدل‌ها به صورت یک به یک (sequential) به مدل نهایی اضافه می‌شوند و به همین دلیل Boosting می‌باشد. این مدل به صورت پیش‌فرض از مدل‌های پایه درخت تصمیم‌گیری استفاده می‌کند اما می‌توان آن را به دلخواه عوض کرد. باید دقت کرد که در این روش وزن داده‌ها بر اساس خطای مرور آپدیت می‌شود. برای داده‌هایی که مقادیر آن‌ها به اشتباه پیش‌بینی شده است وزن بیشتری اختصاص داده می‌شود تا در مراحل بعدی پیش‌بینی صحیح برای آنها انجام شود.

XGBoost (XGB) : 0.059588748609196364, 0.2441080674807704

Linear Regression (LR) : 0.00029374816679205833, 0.017139083020747008

Decision Trees (DT) : 0.05878280480005465, 0.2424516545624192

Random Forest (RF) : 0.059816032088914214, 0.24457316305947024

Bagging (BAG) (SVM) : 0.056212347399047805, 0.23709143257200965

Voting (VOT) (LR + RF) : 0.015373209626986763, 0.12398874798539891

Extra Trees (EXT) : 0.058669735695529225, 0.24221836366289246

AdaBoost (ADA) : 0.06184227682186922, 0.2486810745148678

همانطور که مشاهده می‌شود نتایج بسیار بهتر از روش LSTM می‌شود.

سوال ۲۵

برای این سوال شاخص‌های مختلفی را امتحان کرده و نتایج را شرح می‌دهیم:

Simple Moving Average (SMA):

این روش میانگین قیمت را بر روی بازه زمانی داده شده محاسبه می کند و روند (trend) بورس را نشان می دهد. در ابتدا این مقدار را برای پنجره های با طول ۱۵ و ۵ حساب کرده و مقدار ۱۵ را بر ۵ تقسیم کرده و آن را به عنوان یک فیچر نهایی اضافه می کنیم. مشکل این روش اینست که وزن یکسانی به مقادیر گذشته اختصاص می دهد.

```
bitcoin_df['SMA_5'] = bitcoin_df['Price'].transform(lambda x: x.rolling(window = 5).mean())
bitcoin_df['SMA_15'] = bitcoin_df['Price'].transform(lambda x: x.rolling(window = 15).mean())
bitcoin_df['SMA_Ratio'] = bitcoin_df['SMA_15'] / bitcoin_df['SMA_5']
```

Simple Moving Average Volume (SMA-Volume):

مانند روش قبل اما میانگین حجم را بر روی بازه زمانی داده شده محاسبه می کند و قدرت (Strength) بورس را نشان می دهد. در ابتدا این مقدار را برای پنجره های با طول ۱۵ و ۵ حساب کرده و مقدار ۱۵ را بر ۵ تقسیم کرده و آن را به عنوان یک فیچر نهایی اضافه می کنیم.

```
bitcoin_df['SMA5_Volume'] = bitcoin_df['Vol.'].transform(lambda x: x.rolling(window = 5).mean())
bitcoin_df['SMA15_Volume'] = bitcoin_df['Vol.'].transform(lambda x: x.rolling(window = 15).mean())
bitcoin_df['SMA_Volume_Ratio'] = bitcoin_df['SMA15_Volume'] / bitcoin_df['SMA5_Volume']
```

Average True Range (ATR):

این روش نوسان (volatility) بورس را نشان می دهد. مقادیر زیادتر با نوسان بیشتر مرتبط هستند. برای محاسبه این مقدار در ابتدا از استفاده می کنیم تا اثر Bias از روش SMA که بالاتر به آن اشاره شد خنثی شود. سپس مقدار Wilder برای ۱۵ روز و ۵ روز محاسبه شده و تفاوت آنها به عنوان یک فیچر اضافه می شود.

```
bitcoin_df['TR'] = np.maximum((bitcoin_df['High'] - bitcoin_df['Low']),
                             np.maximum(abs(bitcoin_df['High'] - bitcoin_df['prev_close']),
                                       abs(bitcoin_df['prev_close'] - bitcoin_df['Low'])))

TR_data = bitcoin_df.copy(deep=True)
bitcoin_df['ATR_5'] = Wilder(TR_data['TR'], 5)
bitcoin_df['ATR_15'] = Wilder(TR_data['TR'], 15)

bitcoin_df['ATR_Ratio'] = bitcoin_df['ATR_5'] / bitcoin_df['ATR_15']
```

Relative Strength Index (RSI):

نشان دهنده تغییر قیمت بورس و سرعت این تغییر است. تغییر سرعت برای بازه زمانی ۵ روزه و ۱۵ روزه محاسبه شده و نتیجه تقسیم ۵ روزه بر ۱۵ روزه به عنوان فیچر نهایی اضافه می شود.

```

bitcoin_df['Diff'] = bitcoin_df['Price'].transform(lambda x: x.diff())
bitcoin_df['Up'] = bitcoin_df['Diff']
bitcoin_df.loc[(bitcoin_df['Up']<0), 'Up'] = 0

bitcoin_df['Down'] = bitcoin_df['Diff']
bitcoin_df.loc[(bitcoin_df['Down']>0), 'Down'] = 0
bitcoin_df['Down'] = abs(bitcoin_df['Down'])

bitcoin_df['avg_5up'] = bitcoin_df['Up'].transform(lambda x: x.rolling(window=5).mean())
bitcoin_df['avg_5down'] = bitcoin_df['Down'].transform(lambda x: x.rolling(window=5).mean())

bitcoin_df['avg_15up'] = bitcoin_df['Up'].transform(lambda x: x.rolling(window=15).mean())
bitcoin_df['avg_15down'] = bitcoin_df['Down'].transform(lambda x: x.rolling(window=15).mean())

bitcoin_df['RS_5'] = bitcoin_df['avg_5up'] / bitcoin_df['avg_5down']
bitcoin_df['RS_15'] = bitcoin_df['avg_15up'] / bitcoin_df['avg_15down']

bitcoin_df['RSI_5'] = 100 - (100/(1+bitcoin_df['RS_5']))
bitcoin_df['RSI_15'] = 100 - (100/(1+bitcoin_df['RS_15']))

bitcoin_df['RSI_Ratio'] = bitcoin_df['RSI_5']/bitcoin_df['RSI_15']

```

Moving Average Convergence Divergence (MACD):

این روش از دو میانگین متحرک نمایی (exponential) استفاده می کند و آنالیز روند همگرایی (convergence) و واگرایی (divergence) بورس را حساب می کند. از تفاضل ۱۵ روز و ۵ روز برای محاسبه مقادیر مدنظر استفاده می کنیم.

```

▶ bitcoin_df['5Ewm'] = bitcoin_df['Price'].transform(lambda x: x.ewm(span=5, adjust=False).mean())
bitcoin_df['15Ewm'] = bitcoin_df['Price'].transform(lambda x: x.ewm(span=15, adjust=False).mean())
bitcoin_df['MACD'] = bitcoin_df['15Ewm'] - bitcoin_df['5Ewm']

```

با اضافه کردن تمامی این فیچرها بار دیگر رگرسورهای معرفی شده در سوال ۲۴ را ترین کرده و نتایج را شرح می دهیم. با وجود اضافه کردن این فیچرها به طور کلی خطای نسبت به سوال ۲۳ کمتر و نسبت به سوال ۲۴ بیشتر می شود که نشان دهنده اینست که تغییر قیمت در سوال ۲۴ همچنان فیچر بهتری نسبت به شاخصهای تعریف شده در این سوال می باشد. از طرفی روش های Ensemble همچنان از روش LSTM برای این مساله بهتر هستند.

```

MSE, RMSE error for XGB: 0.154615580519115, 0.3932118773881519
MSE, RMSE error for LR: 0.15462266574612846, 0.3932208867114366
MSE, RMSE error for DT: 0.1546226657461284, 0.3932208867114365
MSE, RMSE error for RF: 0.1545972361143889, 0.3931885503348093
MSE, RMSE error for BAG: 0.15469541891867805, 0.39331338512524344
MSE, RMSE error for VOT: 0.15465693518026233, 0.39326445959463757
MSE, RMSE error for EXT: 0.15462266574612854, 0.3932208867114367
MSE, RMSE error for ADA: 0.1399759097718205, 0.37413354537092836

```

سوال ۲۶

دیتاست مورد نظر بسیار بزرگ بوده (شامل ۵۰۱۸۰۹ داده ترین و ۱۷۲۵۶۰۳ داده تورنمنت) و در حافظه بارگذاری نشد. به این لیل از مجموعه کوچکتری از داده ها (۱۰ هزار داده از هر مجموعه) استفاده می کنیم. همانطور که در صورت سوال شرح داده شد جهت حفظ محترمانگی دیتاست مدنظر تغییر یافته و ما اطلاعی از فیچرهای واقعی نداریم. بخشی از داده ها در زیر نمایش داده شده است:

id	feature_intelligence1	feature_intelligence2	feature_intelligence3
n000315175b67977	0.00	0.50	0.25
n0014af834a96cdd	0.00	0.00	0.00
n001c93979ac41d4	0.25	0.50	0.25
n0034e4143f22a13	1.00	0.00	0.00
n00679d1a636062f	0.25	0.25	0.25

5 rows × 310 columns

پس از آماده سازی داده های ترین و تست متوجه می شویم که برچسب داده های ترین (y_{train}) شامل اعداد ۰.۰۵، ۰.۰۷۵ و ۰.۱ می باشد. از مدل های استفاده شده در سوال ۲۴ استفاده کرده و نتایج را شرح می دهیم:

```
MSE, RMSE error for XGB: 0.05101390215442374, 0.2258625736026749
MSE, RMSE error for LR: 0.05278123475952705, 0.2297416696194381
MSE, RMSE error for DT: 0.11835, 0.34402034823539146
MSE, RMSE error for RF: 0.050840847499999994, 0.2254791509208778
MSE, RMSE error for BAG: 0.05596810163949772, 0.23657578413586144
MSE, RMSE error for VOT: 0.05107260476268298, 0.22599248828817958
MSE, RMSE error for EXT: 0.051203181874999996, 0.22628120088730305
MSE, RMSE error for ADA: 0.05005330248051367, 0.223725953971625
```

با توجه به نتایج بالا مشاهده می شود که ADA و RF (که هر دو از DT استفاده می کنند) بهتر از بقیه عمل کرده اند چرا که کمترین میزان خطای دارند و DT بدترین نتایج را بدست آورده است. مشاهده می شود که بقیه روش ها به جز DT بسیار نزدیک به هم هستند. در اینجا BAGGING هم از مدل های پایه SVM که شبیه هم هستند استفاده می کند. دقت کنید که روش VOT تنها روشی است که از مدل های با ساختار متفاوت استفاده کرده است (مدل رگرسیون خطی و Random Forests).

سوال ۲۷

نتایج بدست آمده را با استفاده از Spearman Correlation با نتایج واقعی مقایسه می کنیم. مقادیر -۱ و ۱ برای rho به ترتیب نشان دهنده رابطه خطی منفی و مثبت و مقدار ۰ نشان دهنده هیچ رابطه ای نیست. همچنین pval نشان دهنده احتمال تولید مقادیری با حداقل (extreme) مقادیر بدست آمده توسط یک سیستم بدون رابطه (uncorrelated) می باشد. در این معیار فرض H_0 عدم وجود رابطه را نشان می دهد ($\rho = 0$). فرض کلی این معیار بر بدون رابطه بودن است (پذیرش فرض H_0). به ترتیب rho و pval برای مدل های مختلف و مقدار واقعی نشان داده شده است.

```
Spearman correlation (rho, pval) between XGB and y_true: -0.015677615078634587,
0.11696006662725852
```

```
Spearman correlation (rho, pval) between LR and y_true: -0.0015004768539846427,  
0.8807417254055534  
Spearman correlation (rho, pval) between DT and y_true: -0.010461672684597542,  
0.2955309709091788  
Spearman correlation (rho, pval) between RF and y_true: -0.0014734940785052406,  
0.882870711258991  
Spearman correlation (rho, pval) between BAG and y_true: -0.011644831126105147,  
0.24427125665803498  
Spearman correlation (rho, pval) between VOT and y_true: -0.0039875420253721455,  
0.6901098903154018  
Spearman correlation (rho, pval) between EXT and y_true: -0.015566345691622992,  
0.11958109830750738  
Spearman correlation (rho, pval) between ADA and y_true: -0.01847725125314812,  
0.06465272160591216
```

با فرض $\alpha=0.05$ ملاحظه می شود که H_0 برای هیچ روشی رد نمی شود. به این معنی که رابطه ای بین خروجی روش ها و مقدار واقعی وجود ندارد. از طرفی با توجه به اینکه مقدار ρ برای تمامی روشها نزدیک به صفر است میتوان نتیجه گرفت که هیچ رابطه ای وجود ندارد. اما باید دقت کرد که اگرچه نتایج در این سوال معنی دار (significant) نیست اما همچنان از نتایج MSE و RMSE در سوال ۲۶ می توان نتیجه گرفت که مدل ها اگر از نظر ساختاری شبیه باشند بهتر نتیجه می گیریم.