محمدرضا کاری

تکلیف سوم:

سوال یک:

در حالت کلی در svm از کرنل خطی استفاده میکنیم اما در صورتی که رابطه به صورتی پیچیده تر از خطی باشد به کرنل های دیگری نیاز داریم تا در ابعادی بیشتر بتوانند خط بین داده هارا تشخیص دهد در اصل استفاده از کرنل های عیر خطی برای انست که داده هایی که خطی نیستند را بتوانیم با استفاده از svm اموزش بدهیم. از مشهور ترین کرنل ها داریم:

کرنل چندجمله ای:

این کرنل بیشتر در پردازش تصویر استفاده میشوند

کرنل گاوسی:

این کرنل برای وقتی بکار میرود که از پراکندگی داده ها خبر نداریم

کرنل rbf :

این کرنل هم مانند کرنل گاوسی زمانی که از داده ها اطلاعات زیادی نداریم بکار میرود و در حالت کلی میتوان ازین کرنل استفاده کرد

کرنل سیگموید:

از این کرنل بیشتر برای واسطی میان شبکه های عصبی استفاده کرد.

سوال ۲:

کد سوال های ۱-۶ در فایل quest_1_6.py نوشته شده است

در اولین مدل یک svm ساده درست کرده ایم که خروجی زیر را میدهد:

```
clf = svm.SVC(gamma='auto', kernel='linear')
clf.fit(x_tr, y_tr)
predict = clf.predict(x_test)
print("kernel type = linear, test score:", clf.score())
print("test accuracy:", accuracy_score(predict, y_test))
```

kernel type = linear, test score0.9725

test accuracy: 0.9725

سوال ۳:

حال برای این سوال پنج حالت مختلف دیگر را نیز امتحان میکنیم که خصوصیات انان متفاوت است. کد های این سوال از خط ۳۹ تا خط 86 قرار دارد و برای مدل های متفاوت مقادیر زیر را داریم:

```
kernel type = linear, test score: 0.9725
test accuracy: 0.9725

kernel type = rbf, test score: 0.2375
test accuracy: 0.2375

kernel type = poly and degree = 3, test score: 0.96
test accuracy: 0.96

kernel type = poly and degree = 5, test score: 0.955
test accuracy: 0.955

kernel type = poly and degree = 2, test accuracy: 0.97
test accuracy: 0.97

kernel type = linear, and regression test accuracy: 0.9575

kernel type = sigmoid test accuracy: 0.2375
test accuracy: 0.2375
```

همانطور که مشاهده میکنیم برای این مسئله کرنل rbf مناسب نیست و دقت کافی را ندارد و علاوه بر ان کرنل سیگموید هم خوب عمل نمیکند. و بهترین دقت برای کرنل خطی است. علاوه بر استفاده از SVM در مدل اخر از SVR استفاده کرده ایم که رگرسیون میگیرد و بخوبی میبینیم که عمل کرده است.

سوال ۴:

به طور کلی از سافت مارجین برای جلوگیری از overfit شدن استفاده میکنیم و hard margin برای انست که به ازای هر داده ای حساسیت بیشتری خرج داده و ممکن است که باعث overfit شدن بشود در اینجا این دو را بر روی داده بررسی میکنیم:

در اینجا کد های ما از خط 89 تا 101 قرار دارد. برای هارد مارجین و سافت مارجین داریم:

```
kernel type = linear, C = 4 hard margin, test accuracy: 0.965

kernel type = lin, C = 0.1 soft margin, test accuracy: 0.975
```

که سافت مارجین مقداری بهتر از بقیه عمل میکند.

سوال های ۵ و۶:

حالت الف:

از خط ۱۰۶ تا ۱۱۴ کد این قرار دارد که این خروجی را به ما میدهد:

```
kernel type = linear and battery power is binned , test accuracy:
0.93
test accuracy: 0.93
```

که در دقت خود کاهش داریم پس بهتر است از binning در اینجا استفاده نکنیم. در اینجا battery power را به چهار قسمت تقسیم کرده ایم که ۱و ۲و ۳و۴ را به ما برمیگرداند.

حالت ب:

در روش one hot encoding ، ما فیچر های زیر را تبریل کرده ایم:

```
battery_power
n_cores
fc
```

حال میرویم به سراغ دیدن نتایج:

```
kernel type = linear and one hot encoding for categorical
features , test accuracy: 0.9325
```

که کمی از binning بهتر است اما باز هم بر روی داده های تست از حالت عادی دقت کمتری دارد. کد های این قسمت از خط ۱۱۸ تا ۱۴۴ نوشته شده است.

حالت ج:

برای حالت ج از دو تبدیل نرمال و log transform استفاده میکنیم با توجه به اینکه برخی از فیچر ها مقداری بیشتر از ۲۰ ندارند پس بهتر است از تبدیل این فیچر ها خود داری کنیم و بهتر است فیچر هایی جه پراکنده هستند را بررسی کنیم پس فیچر های زیر را بررسی میکنیم:

```
df_log['mobile_wt'] = np.log(df_log['mobile_wt']+1)
df_log['px_height'] = np.log(df_log['px_height']+1)
df_log['px_width'] = np.log(df_log['px_width']+1)
df_log['ram'] = np.log(df_log['ram']+1)
```

پس برای این دو تبدیل داریم:

```
kernel type = linear and log transformed data: 0.9175
test accuracy: 0.9175
kernel type = linear and normalized data: 0.96
test accuracy: 0.96
```

که اولی برای تبدیل log و دومی حالت نرمال است که میبینیم نرمال تبدیل بهتری است.

حالت د:

در این حالت برای انکه area روند ترین را بسیار طولانی میکند بخاطر اینکه اعداد ان بزرگ هستند انرا نرمالایز میکنیم تا روند سریعتر پیش برود کد ان در خط های ۱۴۷-۱۵۹ قرار دارد.

```
kernel type = linear with new feature named area , test accuracy:
0.9625
test accuracy: 0.9625
```

حال باید همه این حالت هارا با هم دیگر ترکیب کنیم کافیست که دیتا فریمی که میسازیم را از بین نبریم:

```
kernel type = linear all one hot encoding and others combined
together
test accuracy: 0.90
```

که همانطور که میبینیم روش خوبی نشده است. به علت اینکه با binning کردن power battery ترین کردن بسیار طول میکشد انرا با بقیه روش ها ترکیب نکرده ایم. علاوه بر این پارامتر runFast در صورتی که true باشد تنها از ۲۰ درصد داده ها برای ترین استفاده میشود که برای سریعتر کردن روند ترین هست البته در اینجا برای هیچکدام استفاده نشده.

سوال ۷:

در الگوریتم ID3 بصورت ساده یک کلاس داریم تا بر روی ان و لیبل هایش یک درخت تصمیم بسازیم. الگوریتم اول یک نود ریشه درست کرده و سپس اگر تمام داده های درون یک کلاس یک لیبل یکسان داشته باشند انهارا درون درخت میگزاریم و دیگر به ان کاری نداریم. در غیر اینصورت کلاسی را انتخاب میکنیم که برای ما بیشتر اطلاعات را داشته باشد و سپس داده هارا تقسیم میکنیم و یک شاخه بیشتر درست میکنیم و سپس به صورت بازگشتی برای همه ی ریشه ها اینکار را انجام میدهیم تا همگی در یک لیبل باشند. بعد از انکه ID3 معرفی شد نسخه بعدی ان یعنی C4.5 معرفی شد. در این روش محدودیت اینکه داده ها حتما باید لیبل داشته باشند برداشته شد به این معنی که داده ها میتوانند پیوسته باشند و مشکلی نداریم در اینجا ما درخت را به بخش هایی با if-else های زیاد تقسیم میکنیم. هرس کردن این درخت به اینصورت است که اگر دقت بدون شاخه هم کم نشود ان را هرس میکنیم.

CART یا regression tree تفاوتش با ID3 اینست که میتواند به عنوان رگرسیون استفاده شود. به این معنی که بسیار به C4.5 شبیه است اما متغیر های هدف میتواند عدد باشد و rule set هارا محاسبه نمیکند. در حالت کلی تفاوت اصلی الگوریتم ها در اینست که ایا حتما نیاز به لیبل دارند یا انکه بدون لیبل هم میتوان از انان استفاده کرد یا نه.

سوال و ۸ ۹:

در اینجا کد در فایل quest_7_15.py هست که برای سوال ۸ و ۹ از خط ۴۱ شروع و تا خط ۵۴ ادامه میابد. خروجی مدل ها بصورت زیر است:

```
desicion tree test accuracy: 0.81
desicion tree, max_depth = 10 and min_leaf = 10. test accuracy:
0.83
desicion tree max depth = 3. test accuracy: 0.72
```

که با توجه به چیزی که میبینیم بهترین حالت برای وقتی است که حداقل برگ هارا بر روی ۱۰ قرار میدهیم و در صورتی که عمق را بسیار کم کنیم درخت ما بخوبی ترین نمیشود.

سوال ۱۰:

در روش هرس کردن ما شاخه ی درخت تصمیم را هرس میکنیم شاخه هایی که اضافی بوده و در عمل به کار ما نمیاند و شاخه هایی که به مدل ما دقت زیادی اضافه نمیکنند پس اینکار باعث میشود دقت ما بر روی داده های تست بیشتر بشود برای انکه هرس کنیم میتوانیم تعداد حداکثر شاخه های مجاز برای درخت را تعیین کنیم. در کل هرس کردن درخت برای جلوگیری از overfitting است زیرا شاخه های بسیار زیاد ممکن است نتها به ما کمکی نکنند بلکه باعث حساسیت زیاد درخت به داده های پرت هم بشود.

سوال ۱۱:

تعداد ایده ال شاخه ها در این مدل ها ۴۰ بوده است

```
clf = DecisionTreeClassifier(random_state=0, max_leaf_nodes=40)
clf.fit(x_tr, y_tr)
print("desicion tree purning test accuracy:", clf.score(x_test,
y_test))
```

که خروجی زیر را داده

```
desicion tree purning test accuracy: 0.8625
```

که بهتر از حالت عادی است اگر تعداد شاخه هارا کمتر کنیم و بیشتر هرس کنیم درخت ما دقتش از حالت عادی کمتر هم میشود.

سوال ۱۲:

برای رندوم فارست داریم:

```
clf = RandomForestClassifier(max_depth=10, random_state=0)
clf.fit(x_tr, y_tr)
print("random forest test accuracy:", clf.score(x_test, y_test))
```

که خروجی زیر را میدهد:

```
random forest test accuracy: 0.8675
```

که از درخت تصمیم به تنهایی بهتر است زیرا درخت تصمیم تنها از یک درخت استفاده میکند در صورتی که رندوم فارست از چندین درخت برای الگوریتم خود استفاده کرده است که باعث میشود overfit نشود و همین باعث میشود از درخت تصمیم بر روی داده های تست بهتر عمل کند.

سوال ۱۳:

درخت های تصمیم برای دیتا ست های غیر خطی بخوبی کار میکنند. از خوبی های این مدل های میتوان به خواندنی بودن و راحت تخلیل شدن اشره کرد بطوریکه میتوان این درخت هارا به راحتی خواند و تحلیل کرد.و خروجی های این مدل ها هم تحلیل شدنی هستند بدون حتی اینکه به دانش عمیق امار نیازی داشته باشیم. دیگر برتری ان اینست که به راحتی میتوان انرا اماده کرد و بر روی داده ها نیازی به تغییرات زیاد نیست.و علاوه بر اینها به تمیز کاری داده ها هم کمتر نیاز داریم.

سوال ۱۴:

الگوریتم ripper یا بعبارتی Repeated Incremental Pruning to Produce Error Reduction

الگوریتمی است که بر پایه قواعد است و یک الگوریتم کلاسبندی است. از خوبی های این الگوریتم میتوان به اینکه بر روی داده های نامتوازن به خوبی کار میکند و با داده هایی که بسیار نویز دارد هم خوب کار میکند اشاره کرد. این الگوریتم از بین داده هایی که دارد کلاس اکثریت را شناسایی کرده و انرا به عنوان پیش فرض در نظر میگیرد. حال برای کلاس دیگر تلاش میکند تا قواعدی که باعث میشود ان کلاس باشد را در بیاورد. در حالتی که کلاس ها بیشتر از دوتا است هم کلاسی که بیشترین تعداد را دارد کلاس پیش فرض در نظر میگیریم. بعد از انکه یک قاعده را شناسایی کردیم انرا به قاعده هایی که داریم اضافه میکنیم. درست کردن یک قاعده جدید هم به اینصورت است که اول یک قاعده کلی داریم و سپس بهترین قاعده ای که میتوان ساخت را میسازیم و انرا ارزیابی میکنیم اگر بخوبی کار کرد انرا اضافه کرده و اگر نه انرا هرس میکنیم. در حالت کلی بصورت تکراری قاعده هارا اضافه کرده و در صورتی که در ارزیابی موفق نباشند انهارا هرس میکنیم تا بجایی برسیم که دیگر نتوانیم بهتر عمل کنیم.

الگوریتم IREP یک الگوریتم برپایه قاعده است که تلاش میکند ارور کلی را کاهش دهد و در عین حال قاعده های بیشتری درست کند. اندازه ارور هم بصورت تعداد رکورد هایی که به اشتباه کلاس بندی شده اند را میگزاریم که این تعداد باید کم بشود. در حالت کلی این الگوریتم قواعدی را اضافه میکند که ارور کمتر شود و ادامه میدهد تا ارور به کمترین حد خود برسد این الگوریتم کوتاه شده repeated incremental pruning to produce error reduction است.

سوال ۱۵:

بله میتوان به عنوان مثال میتوان داه هارا بصورتی تبدیل کرد که بتوان به عنوان یک مسئله رگرسیون یا کلاسبندی ازان استفاده کرد.
به این صورت که چندین داده قبلی را برای داده بعدی استفاده کنیم و سپس مدل خود را بر روی ان اموزش دهیم.علاوه بر ان چون این یک مدل بر پایه قاعده است پس میتوان از ان به راحتی برای پیشبینی هم استفاده کرد و مشکلی ندارد.

سوال ۱۷:

مدل های این سوال در ۳ فایل quest16_25.py و quest17lstm.py و quest17_cnn.py ترین شده اند

برای این سوال با استفاده از چندین الگوریتم مدل را اموزش داده ایم:

```
RandomForest
DecisionTree
SVR
AdaBoost
BayesianRidge
MLPRegressor
LinearRegression
XGBoost
VotingRegressor
BaggingRegressor
LSTM
cnn
RidgeCV
```

حال برای داده ها داریم:

```
RidgeCV accuracy: 0.9672727272727273
RidgeCV RMSE: 1294.5634226689417

RandomForest with default parameters accuracy: 0.5909090909090909
RandomForest with default parameters RMSE: 17558.097958795406

DecisionTree accuracy: 0.5709090909090909
DecisionTree RMSE: 17253.756006292027
```

```
SVR accuracy: 0.0
SVR RMSE: 27535.585352825146

AdaBoost with default parameters accuracy: 0.6
AdaBoost with default parameters RMSE: 17406.594212122272

BayesianRidge accuracy: 0.9618181818181818
BayesianRidge RMSE: 1296.242809704768

MLPRegressor accuracy: 0.9636363636363636
MLPRegressor RMSE: 1311.958736701546

LinearRegression accuracy: 0.9618181818181818
LinearRegression RMSE: 1298.10427437145

XGBoost accuracy: 0.6218181818181818
XGBoost RMSE: 17508.502979470286
```

داریم و برای LSTM داریم:

```
LSTM accuracy: 0.1
LSTM RMSE: 21386.54731130825
```

و cnn هم نتوانست بخوبی این مسئله را پیش بینی کند.

سوال ۱۸:

بهترین مدل ها در اینجا مدل های زیر هستند:

```
# best 4 -> BayesianRidge, LinearRegression, MLPRegressor,
XGBoost
```

که خود xgboost خود یک روش boosting است. تنها تفاوت gradient boosting regressor با xgboost در سرعت انهاست.

برای baggin هم از تکه کد زیر استفاده میکنیم:

```
bays_bagging_reg = BaggingRegressor(base_estimator=bay_reg,
                                    n_estimators=15,
random_state=0)
bays_bagging_reg.fit(x_tr, y_tr)
predict_test = bays_bagging_reg.predict(x_test)
```

```
print('BayesianRidge bagging accuracy:',
return_accuracy(predict_test, y_test))
print('BayesianRidge bagging RMSE:', mean_squared_error(y_test,
predict_test, squared=False))


bays_bagging_reg = BaggingRegressor(base_estimator=lin_reg,
                                    n_estimators=20,
random_state=0)
bays_bagging_reg.fit(x_tr, y_tr)
predict_test = bays_bagging_reg.predict(x_test)
print('LinearRegression bagging accuracy:',
return_accuracy(predict_test, y_test))
print('LinearRegression bagging RMSE:',
mean_squared_error(y_test, predict_test, squared=False))
```

که در اینجا دقتشان اینگونه است:

```
BayesianRidge bagging accuracy: 0.9636363636363636
BayesianRidge bagging RMSE: 1302.7030438298902

LinearRegression bagging accuracy: 0.9618181818181818
LinearRegression bagging RMSE: 1305.140575063054
```

که دقت بمراتب بسیار خوب است

برای روش voting هم از تکه کد زیر استفاده کرده ایم:

```
## voting

bay_reg = linear_model.BayesianRidge()
lin_reg = LinearRegression()
MLP_reg = MLPRegressor(random_state=1, max_iter=1500)
XG_reg = GradientBoostingRegressor(random_state=0)

vot_reg = VotingRegressor([('bay_reg', bay_reg), ('lin_reg',
lin_reg),
                       ('MLP_reg', MLP_reg), ('XG_reg', XG_reg)])
vot_reg.fit(x_tr, y_tr)
predict_test = vot_reg.predict(x_test)
print('voting accuracy:', return_accuracy(predict_test, y_test))
```

```
print('voting RMSE:', mean_squared_error(y_test, predict_test,
squared=False))
```

که نتایج ان بصورت زیر است:

```
voting accuracy: 0.6836363636363636
voting RMSE: 4545.3598284987675
```

که نسبت به روش هایی که ازان استفاده کرده ایم خوب نیست.

در روش boosting هم از روش زیر استفاده کرده ایم:

```
### boosting

regr11 = GradientBoostingRegressor(random_state=0)
regr11.fit(x_tr, y_tr)
predict_test = regr11.predict(x_test)
print('XGBoost accuracy:', return_accuracy(predict_test, y_test))
print('XGBoost RMSE:', mean_squared_error(y_test, predict_test,
squared=False))
```

که خروجی ان میشود مشکل در اینجا مربوط به سخت افزار ضعیف است که برای هر epoch بسیار طول میکشد تا کند پس مجبور شدم که تعداد انرا به ۱۰۰ تا محدود کنم اما در کد انرا ۱۰۰۰۰ قرار داده ام پس بخاطر epoch های کم نمیتوان قضاوتی درباره این مدل داشت:

```
XGBoost accuracy: 0.6218181818181818
XGBoost RMSE: 17508.502979470286
```

که دقت خوبی نیست.

سوال ۱۹:

الگوریتم adaboost را با روش های متفاوت اجرا کردیم و خروجی ان به ترتیب زیر شد:

```

AdaBoost with default parameters accuracy: 0.6
AdaBoost with default parameters RMSE: 17406.594212122272
```

```
AdaBoost with learning rate 0.1 accuracy: 0.5818181818181818
RMSE: 17867.667780376887

AdaBoost with loss=square accuracy: 0.6090909090909091
RMSE: 17433.278681226227

AdaBoost with loss=square and n_estimators = 500 and
learning_rate=0.1 accuracy: 0.6054545454545455
RMSE: 17420.320771419094
```

که نشان میدهد با تغییر loss به مربعی کمی مدل ما بهبود میابد اما تغییر زیادی نمیکند و زیاد کردن estimator ها هم مقداری در کم کردن RMSE کمک میکند.

سوال ۲۰:

```
RandomForest with default parameters accuracy: 0.5909090909090909
RandomForest with default parameters RMSE: 17558.097958795406

RandomForest with max_depth=2 accuracy: 0.2690909090909091
RandomForest with max_depth=2 RMSE: 20582.126560519824

RandomForest with max_depth=10 accuracy: 0.5945454545454546
RandomForest with max_depth=10 RMSE: 17542.163057236765

RandomForest with max_depth=10 and criterion=mae accuracy:
0.5927272727272728
RandomForest with max_depth=10 and criterion=mae RMSE:
17622.33596677024

RandomForest with max_depth=10 and criterion=mse and
min_samples_leaf=6 accuracy: 0.5872727272727273
RandomForest with max_depth=10 and criterion=mse and
min_samples_leaf=6 RMSE: 17920.99126168898
```

در اینجا رندم فارست را با پارامتر های مختلف اجرا کرده ایم که میتوان دید در حالت بهینه به ۶۰ درصد دقت میرسیم و تغییر پارامتر ها کمک زیادی به ما نمیکند اما میتوان دید که اگر عمق حداکثر را کم کنیم مدل ما بشدت دقتش کم میشود و باعث میشود یادگیری به خوبی پیش نرود.

سوال ۲۲:

برای اینکار در صورتی که قیمت ها افزایش پیدا کند ۱ و در غیر اینصورت ۰ رد میکنیم پس با توجه به مدلمان میبینیم که کد زیر به این صورت کار میکند:

```
model_cat = Sequential()
model_cat.add(LSTM(550, input_shape=(1, 550),
return_sequences=True))
model_cat.add(Dense(550))
model_cat.compile(optimizer='adam',
                  loss='mean_absolute_error',
                  metrics=['accuracy'])
model_cat.fit(x_tr, y_train_cat, epochs=400, batch_size=1,
verbose=2, validation_data=(x_test, y_test_cat))
```

که خروجی میشود:

```
Epoch 398/400
 - 0s - loss: 0.0418 - accuracy: 0.0000e+00 - val_loss: 0.5536 -
val_accuracy: 0.0000e+00
Epoch 399/400
 - 0s - loss: 0.0477 - accuracy: 0.0000e+00 - val_loss: 0.5553 -
val_accuracy: 0.0000e+00
Epoch 400/400
 - 0s - loss: 0.0459 - accuracy: 0.0000e+00 - val_loss: 0.5528 -
val_accuracy: 0.0000e+00
categorial LSTM accuracy: 0.5218181818181818
```

سوال ۲۶ و ۲۷:

این دو سوال در فایل quest26_27.py نوشته شده.

در اینجا بدلیل سنگین بودن دیتاست و سریع بودن این سه مدل را انتخاب کردیم که میتوان کد انهارا دید:

```
reg = linear_model.BayesianRidge()
reg.fit(x_tr, y_tr)
```

```
predict_test = reg.predict(x_test)
print('BayesianRidge accuracy:', return_accuracy(predict_test,
y_test))
print('BayesianRidge RMSE:', mean_squared_error(y_test,
predict_test, squared=False))
print('score:', reg.score(x_test, y_test))
print('spearman correlation:', stats.spearmanr(predict_test,
y_test), '\n')


reg = reg = linear_model.BayesianRidge()
reg.fit(x_tr, y_tr)
predict_test = reg.predict(x_test)
print('BayesianRidge accuracy:', return_accuracy(predict_test,
y_test))
print('BayesianRidge RMSE:', mean_squared_error(y_test,
predict_test, squared=False))
print('score:', reg.score(x_test, y_test))
print('spearman correlation:', stats.spearmanr(predict_test,
y_test), '\n')


reg = LinearRegression()
reg.fit(x_tr, y_tr)
predict_test = reg.predict(x_test)
print('LinearRegression accuracy:', return_accuracy(predict_test,
y_test))
print('LinearRegression RMSE:', mean_squared_error(y_test,
predict_test, squared=False))
print('score:', reg.score(x_test, y_test))
print('spearman correlation:', stats.spearmanr(predict_test,
y_test), '\n')


regr = DecisionTreeRegressor(max_depth=8)
regr.fit(x_tr, y_tr)
predict_test = regr.predict(x_test)
print('DecisionTree accuracy:', return_accuracy(predict_test,
y_test))
print('DecisionTree RMSE:', mean_squared_error(y_test,
predict_test, squared=False))
print('score:', reg.score(x_test, y_test))
```

```
print('spearman correlation:', stats.spearmanr(predict_test,
y_test), '\n')
()
reg.fit(x_tr, y_tr)
predict_test = reg.predict(x_test)
print('LinearRegression accuracy:', return_accuracy(predict_test,
y_test))
print('LinearRegression RMSE:', mean_squared_error(y_test,
predict_test, squared=False))
print('score:', reg.score(x_test, y_test))
print('spearman correlation:', stats.spearmanr(predict_test,
y_test), '\n')


regr = DecisionTreeRegressor(max_depth=8)
regr.fit(x_tr, y_tr)
predict_test = regr.predict(x_test)
print('DecisionTree accuracy:', return_accuracy(predict_test,
y_test))
print('DecisionTree RMSE:', mean_squared_error(y_test,
predict_test, squared=False))
print('score:', reg.score(x_test, y_test))
print('spearman correlation:', stats.spearmanr(predict_test,
y_test), '\n')
```

که در اینجا از الگوریتم های

```
BayesianRidge
LinearRegression
DecisionTreeRegressor
```

استفاده کرده ایم تا سریع و خوب باشند. که میتوان نتایج خروجی انهارا دید:

```
BayesianRidge accuracy: 0.5014248420717005
BayesianRidge RMSE: 0.22249774105961753
score: 0.0018412158176822624
spearman correlation:
SpearmanrResult(correlation=0.043235971227992 48,
pvalue=9.691101086828854e-43)

LinearRegression accuracy: 0.5014049142105578
LinearRegression RMSE: 0.22253698860310672
```

```
score: 0.0014890438336315759
spearman correlation:
SpearmanrResult(correlation=0.04067024420253731,
pvalue=5.158374428910637e-38)

DecisionTree accuracy: 0.4886909388015384
DecisionTree RMSE: 0.22347613019230392
score: 0.0014890438336315759
spearman correlation:
SpearmanrResult(correlation=0.02420912122944257,
pvalue=1.7136860151186796e-14)
```

که تقریبا در پیش بینی در یک رنج قرار میگیرند.

حال با استفاده از voting, bagging و ترکیب این دو حالت داریم:

```
oting accuracy with all 3 models: 0.5008867898208486
voting RMSE: 0.22256947004662234
score: 0.0011975377394043418
spearman correlation:
SpearmanrResult(correlation=0.04155522647324363,
pvalue=1.3024421834351081e-39)

BayesianRidge bagging with 3 estimator accuracy:
0.5014248420717005
BayesianRidge bagging RMSE: 0.22250996931401235
score: 0.0017314971761452913
spearman correlation:
SpearmanrResult(correlation=0.04150024450537839,
pvalue=1.6406552683639232e-39)

linear regression bagging with 4 estimator accuracy:
0.5013750224188438
linear regression bagging RMSE: 0.2225417801904677
score: 0.0014460442045275412
spearman correlation:
SpearmanrResult(correlation=0.04021213352389492,
pvalue=3.358854255890155e-37)

linear regression bagging with 4 estimator accuracy:
0.5013750224188438
```

```
linear regression bagging RMSE: 0.2225417801904677
score: 0.0014460442045275412
spearman correlation:
SpearmanrResult(correlation=0.04021213352389492,
pvalue=3.358854255890155e-37)

decision tree bagging with 5 estimator accuracy:
0.4984655546920149
decision tree bagging RMSE: 0.22281959937918777
score: -0.0010486838631573736
spearman correlation:
SpearmanrResult(correlation=0.024033062949633942,
pvalue=2.6436575090775234e-14)

voting and bagging combined accuracy: 0.5014248420717005
voting and bagging combined RMSE: 0.2225125790119127
score: 0.0017080807467493297
spearman correlation:
SpearmanrResult(correlation=0.041661036194269335,
pvalue=8.345362590973376e-40)
```

که میتوان دید از نظر دقت به برتری زیادی نمیرسیم اما بغیر از حالت bagging برای رگرسیون دقتمان کم هم نمیشود اما در حالت کلی میتوان گفت در صورتی که مدل ها ساختار متفاوتی داشته باشند کمی خروجی بهتر میشود.

دیتاست ها همان دیتاست های داده شده در تکلیف برای هر سوال است:

سوال ۱-۶:

```
train.csv
```

سوال ۷-۱۵:

```
train.csv
```

سوال ۱۶-۲۵:

```
bitcoin_data.csv
```

سوال ۲۶ و ۲۷:

```
datas/training_data.csv
```