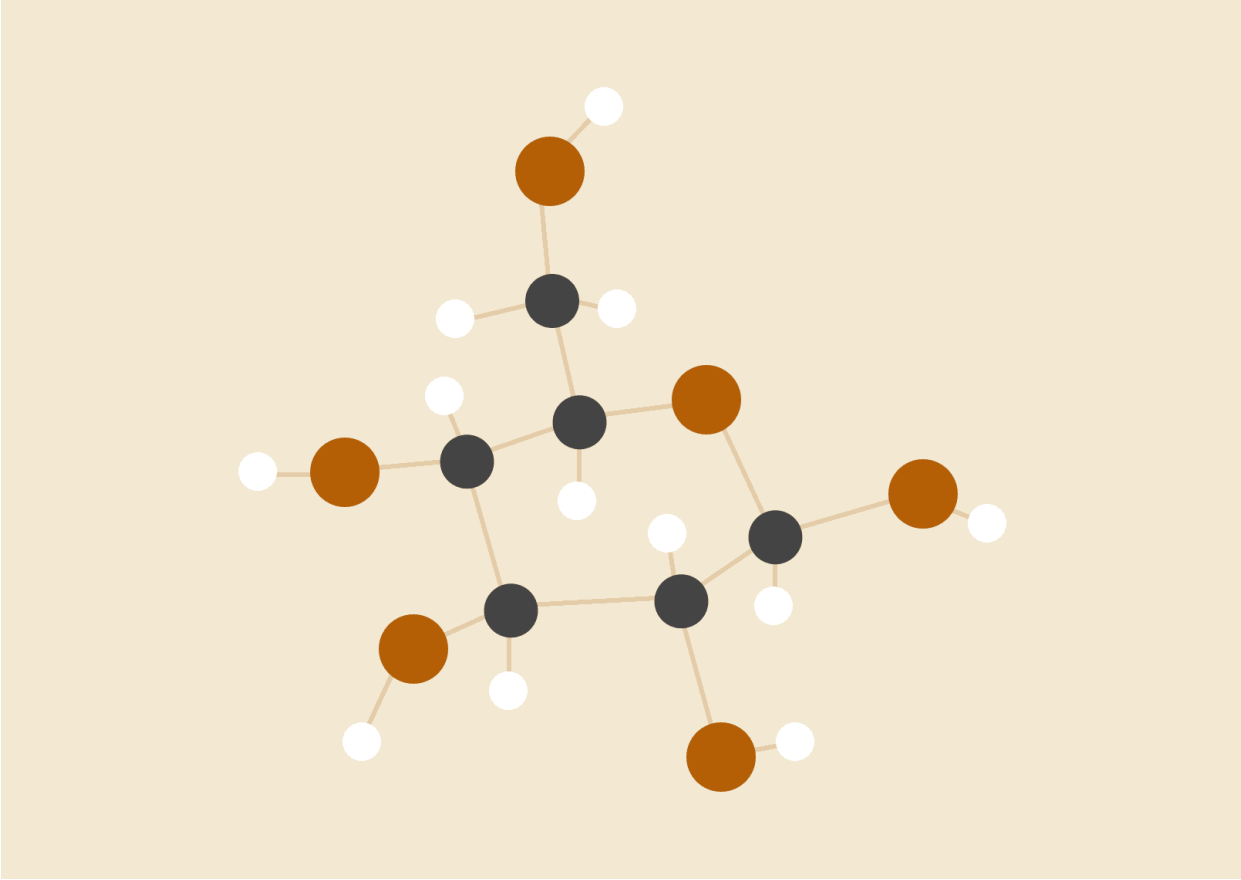
homework-4

Lorem ipsum dolor sit amet, consectetuer adipiscing elit



Alireza Javaheri

99422008

: SVM

یک مدل supervised است. که با بیشینه کردن فواصل ساپورت وکتورهای دو کلاساز ابر صفحه جداکننده به جدا کردن دو دسته به هم میپردازد. (svm فقط برای جدا کردن دو دسته به کار میکرد و اگر دسته بندی چند کلاسه باشد باید از روش های one vs one or one vs all استفاده شود.)

این مدل یک جداکننده خطی است اگر بخواهیم برای جداکردن مسایل پیچیده تر از svm استفاده کنیم باید از کرنل های غیر خطی مانند rbf استفاده کنیم.

SVM رو با پارامتر های مختلف وکرنل های مختلف رو دیتاست قیمت گوشی اجرا کردیم که نتایج به صورت زیر است.

بدون کرنل :

	precision	recall	f1-score	support
0	0.99	0.99	0.99	501
1	0.98	0.98	0.98	500
2	0.96	0.98	0.97	493
3	0.99	0.98	0.98	506
accuracy			0.98	2000
macro avg	0.98	0.98	0.98	2000
weighted avg	0.98	0.98	0.98	2000

نتایج با کرنل rbf :

	precision	recall	f1-score	support
0	0.99	0.99	0.99	498
1	0.98	0.97	0.98	506
2	0.98	0.97	0.98	503
3	0.98	1.00	0.99	493
accuracy			0.98	2000
macro avg	0.98	0.98	0.98	2000
weighted avg	0.98	0.98	0.98	2000

با کرنل poly:

	precision	recall	f1-score	support
0	0.97	1.00	0.98	484
1	0.99	0.96	0.97	514
2	0.99	0.94	0.96	527
3	0.95	1.00	0.97	475
accuracy			0.97	2000
macro avg	0.97	0.97	0.97	2000
weighted avg	0.97	0.97	0.97	2000

نتایج با کرنل sigmoid

	precision	recall	f1-score	support
0	0.91	0.92	0.92	494
1	0.85	0.82	0.83	519
2	0.82	0.82	0.82	498
3	0.90	0.92	0.91	489
accuracy			0.87	2000
macro avg	0.87	0.87	0.87	2000
weighted avg	0.87	0.87	0.87	2000

در این بخش soft-margin و hard-margin رو مورد بررسی قرار میدهیم.

Soft-margin:

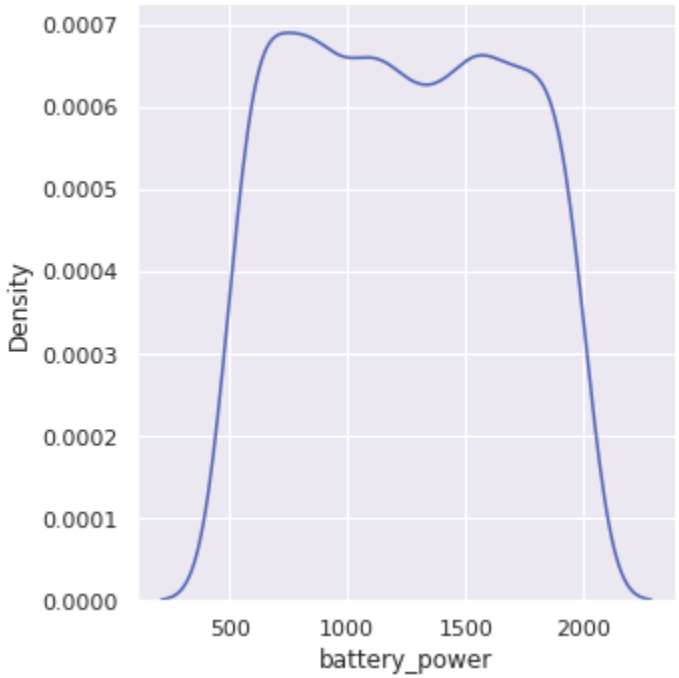
	precision	recall	f1-score	support
0	0.99	0.99	0.99	501
1	0.98	0.98	0.98	500
2	0.96	0.98	0.97	493
3	0.99	0.98	0.98	506
accuracy			0.98	2000
macro avg	0.98	0.98	0.98	2000
weighted avg	0.98	0.98	0.98	2000

:Hard-margin

	precision	recall	f1-score	support
0	1.00	1.00	1.00	500
1	0.99	1.00	0.99	499
2	0.99	0.99	0.99	500
3	0.99	0.99	0.99	501
accuracy			0.99	2000
macro avg	0.99	0.99	0.99	2000
weighted avg	0.99	0.99	0.99	2000

Feature Engineering

در ابتدا طبق خواسته صورت سوال میخواهیم battery_power رو به سه bin تقسیم کنیم. برای پیدا کردن رنج بازه ها نمودار توضیح این فیچر رو رسم کردیم.

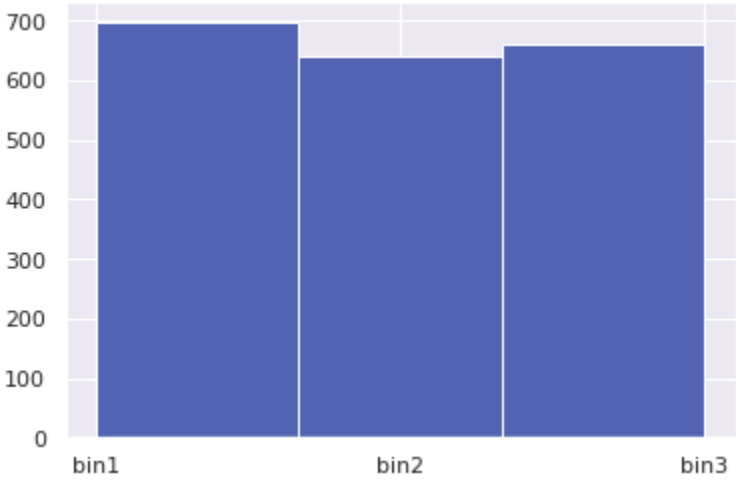


طبق نمودار بالا بازه های ۰-۱۰۰۰ و ۱۰۰۰ - ۱۵۰۰ و ۱۵۰۰ تا آخر مناسب است و کد آن به صورت زیر است.

```
bins = np.linspace(max_value, min_value, 4)

labels = ['bin1', 'bin2', 'bin3']

train_x['battery_power_bin'] = pd.cut(train_x['battery_power'], bins=bins, labels=labels, include_lowest=True)

train_x['battery_power_bin'].unique()
```

```
['bin1', 'bin2', 'bin3']
Categories (3, object): ['bin1' < 'bin2' < 'bin3']
```

بعد از این بین بندی توزیع داده های این فیچر در سه تا بین به صورت زیر است.



بعد از این بین بندی داده ها آن ها رو one hot میکنیم . نتیجه مدل به صورت زیر است .

	precision	recall	f1-score	support
0	0.96	0.97	0.96	497
1	0.93	0.92	0.92	506
2	0.93	0.93	0.93	501
3	0.96	0.97	0.97	496
accuracy			0.95	2000
macro avg	0.95	0.95	0.95	2000
weighted avg	0.95	0.95	0.95	2000

در این قسمت علاوه بر one hot کردن battery_power_bin فیچر wifi رو one hot میکنیم که نتیج آن به صورت زیر است.

	precision	recall	f1-score	support
0	0.97	0.97	0.97	499
1	0.93	0.92	0.92	506
2	0.92	0.93	0.92	499
3	0.96	0.97	0.97	496
accuracy			0.95	2000
macro avg	0.95	0.95	0.95	2000
weighted avg	0.95	0.95	0.95	2000

علاوه فیچر های بالا blue رو هم one hot میکنیم و نتیجه آن به صورت زیر است.

	precision	recall	f1-score	support
0	0.97	0.96	0.97	501
1	0.92	0.92	0.92	500
2	0.93	0.92	0.93	504
3	0.96	0.97	0.97	495
accuracy			0.95	2000
macro avg	0.95	0.95	0.95	2000
weighted avg	0.95	0.95	0.95	2000

فیچر dual_sim رو one-hot میکنیم . نتیجه آن به صورت زیر است.

	precision	recall	f1-score	support
0	0.96	0.96	0.96	501
1	0.92	0.92	0.92	502
2	0.92	0.92	0.92	500
3	0.96	0.97	0.97	497
accuracy			0.94	2000
macro avg	0.94	0.94	0.94	2000
weighted avg	0.94	0.94	0.94	2000

فیچر four_g رو one_hot میکینم و نتیجه آن به صورت زیر است.

	precision	recall	f1-score	support
0	0.97	0.96	0.97	503
1	0.93	0.92	0.93	501
2	0.93	0.93	0.93	497
3	0.97	0.97	0.97	499
accuracy			0.95	2000
macro avg	0.95	0.95	0.95	2000
weighted avg	0.95	0.95	0.95	2000

فیچر touch_screen رو one-hot میکنیم و نتیجه آن به صورت زیر است.

	precision	recall	f1-score	support
0	0.97	0.96	0.97	505
1	0.92	0.92	0.92	499
2	0.92	0.93	0.92	497
3	0.97	0.97	0.97	499
accuracy			0.94	2000
macro avg	0.95	0.94	0.94	2000
weighted avg	0.95	0.94	0.95	2000

حال یه فیچر جدید به نام مساحت که حاصل ضرب طول در عرض است رو تعریف میکنیم. و نتیجه آن به صورت زیر است.

	precision	recall	f1-score	support
0	0.97	0.96	0.97	502
1	0.92	0.92	0.92	501
2	0.93	0.92	0.93	502
3	0.96	0.97	0.97	495
accuracy			0.95	2000
macro avg	0.95	0.95	0.95	2000
weighted avg	0.95	0.95	0.95	2000

ساختن درخت

لگوریتم ID3 یکی از الگوریتم‌های پایه برای ساختِ درخت‌های تصمیم است.در یک درخت تصمیم، مهم است که کدام یک از ویژگی‌ها (یا همان ابعاد) را در سطوح بالاتری از درخت انتخاب کنیم تا به طبقه‌بندی کمک کند.

الگوریتم ID3 وظیفه پیداکردن ویژگی هایی دارای اطلاعات زیادتر (gain بیشتر) را دارد و آن ها را در سطوح بالاتری از درخت قرار میدهد.هر بار که یک ویژگی در سطحی از درخت انتخاب شد، زیر درخت‌های آن نیز دقیقا به همان صورت (ویژگی‌هایی با اطلاعات بالا) انتخاب می‌شوند و در سطوح و گره‌های بعدی قرار می‌گیرند.

الگوریتم C4.5

تئوري بر اين اساس است كه تعداد آزمونهايي كه باعث ميشود يک نمونه جديد در داخل پايگاه داده، دستهبندي شود، حداقل شود.الگوريتم C4.5 دامنه دستهبندي را علاوه بر صفات قياسي در انواع صفات عددي نيز توسعه ميدهد. الگوريتم اصولا صفتي را كه حداكثر درجه جداسازي بين دستهها را دارد را انتخاب ميكند و درخت تصميم را بر اساس آن ميسازد

با استفاده از پکیج sklearn الگوریتم decision tree رو پیاده کردیم که نتایج آن به صورت زیر است.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	500
1	1.00	1.00	1.00	500
2	1.00	1.00	1.00	500
3	1.00	1.00	1.00	500
accuracy			1.00	2000
macro avg	1.00	1.00	1.00	2000
weighted avg	1.00	1.00	1.00	2000

حال الگوریتم random forest رو پیاده میکنیم . نتایج آن به صورت زیر است.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	500
1	1.00	1.00	1.00	500
2	1.00	1.00	1.00	500
3	1.00	1.00	1.00	500
accuracy			1.00	2000
macro avg	1.00	1.00	1.00	2000
weighted avg	1.00	1.00	1.00	2000

هرس کردن (prunning)

درخت تصمیم تا جایی پیش میرود که در هر برگ آن یک داده باقی بماند و این ممکن است خیلی حاها باعث overfit شد شود و دقت تست آن ها به شدت کاهش می یابد. برای جلوگیری از این کار میتوان یک عمق برای این درخت ها در نظر گرفت که برای جلوگیری از مشکلات گفته درخت بیشتر از ان عمق نشود.

با استفاده از این کار ممکن است خطای داده های آموزشی مقداری افزایش یابد اما خطای داده های تست کاهش می یابد.

درخت و شبکه های عصبی

درسته که امروزه شبکه های عصبی بسیار پیشرفت کردن و نتایج هیلی هوبی میگرند اما با این حال درخت های تصمیم هنوز محبوب اند . این مدل ها به دلیل سادگی و تفسیرپذیری و مقاوم بودن به داده های غیربالانس کاربرد های زیادی دارند. این مدل ها خیلی سریع تز از شبکه های عصبی train میشوند و از طرفی با دیتاهای کم هم خوب کارمکنند در صورتی که شبکه عصبی data hungry است.

استفاده از درخت های تصمیم برای شبکه عصبی

به طور کلی درخت های تصمیم و random forest برای مسایب supervised استفاده میشوند. از این الگوریتم ها برای time series forecasting هم میتوانیم استفاده کنیم اگرچه این نیازمند این است دیتاست آن به شکل یم مسله supervised در بیاید . همچنین نیاز به استفاده از تکنیک های مخصوصی برای evaluate کردن مدل است که walk-forward validation نامیده میشود به عنوان evaluate کردن مدل با استفاده از k-fold corss validatio .

داده های bitcoin

ابتدا با استفاده از کد های زیر داده ها رو preprocess میکنیم.

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

df['Date'] = df['Date'].apply(lambda x: mdy_to_ymd(x))

df['Vol.'] = pd.to_numeric(df['Vol.'].apply(lambda x: x[:-1]))
df['Change %'] = pd.to_numeric(df['Change %'].apply(lambda x: x[:-1]))
```

```
def remove_comma(x):
  arr = x.split(',')
  number = ''
  for i in arr:
    number += i
  return float(number)

for column in ['Price', 'Open', 'High', 'Low']:
  df[column] = df[column].apply(remove_comma)
```

سپس طبق صورت سوال داده های تست و ترین رو جدا میکنم. و به ۱۰ مدل این دادهرو میدهیم. که مدل ها به صورت زیر است.

```
models = {
    'LinearRegression':LinearRegression(),
    'Ridge':Ridge(),
    'Lasso':Lasso(),
    'DecisionTreeRegressor':DecisionTreeRegressor(),
    'RandomForestRegressor':RandomForestRegressor(),
    'AdaBoostRegressor':AdaBoostRegressor(),
    'GradientBoostingRegressor':GradientBoostingRegressor(),
    'BayesianRidge()':BayesianRidge(),
}
```

که نتایج آن به صورت زیر است.

```
LinearRegression --> train mse: 6072.6493 ,  test mse: 39162.6274

Ridge --> train mse: 6072.6493 ,  test mse: 39162.6307

Lasso --> train mse: 7252.6713 ,  test mse: 43258.4955

DecisionTreeRegressor --> train mse: 0.0000 ,  test mse: 4010967.5819
```

```
RandomForestRegressor --> train mse: 1090.4298 ,  test mse: 4138685.4864

AdaBoostRegressor --> train mse: 66826.7938 ,  test mse: 4728619.8880

GradientBoostingRegressor --> train mse: 1496.7909 ,  test mse: 4076197.0881

BayesianRidge() --> train mse: 6072.8738 ,  test mse: 39207.8160
```

روش voting

از بین مدل های بالا دو مدل lasso و baysianRidge نتایج خوبی گرفتن آن ها رو با استفاده از voting تست میکنیم

```
vot_model = VotingRegressor([('BayesianRidge', BayesianRidge()), ('Lasso', Lasso())])
vot_model.fit(train_x, train_y)

train_pred = vot_model.predict(train_x)
train_mse = mean_squared_error(train_y, train_pred)

test_pred = vot_model.predict(test_x)
test_mse = mean_squared_error(test_y, test_pred)

print('voting model --> train mse: {:0.4f} ,  test mse: {:0.4f} \n'.format(train_mse, test_mse))
```

```
voting model --> train mse: 6366.0376 ,  test mse: 39796.6373
```

روش bagging

با n_estimators برابر ۲۰۰

```
bag_model = BaggingRegressor(base_estimator=DecisionTreeRegressor(), n_estimators=200)

bag_model.fit(train_x, train_y)
train_pred = bag_model.predict(train_x)
train_mse = mean_squared_error(train_y, train_pred)

test_pred = bag_model.predict(test_x)
test_mse = mean_squared_error(test_y, test_pred)

print('bag model (DecisionTreeRegressor)--> train mse: {:0.4f} ,  test mse: {:0.4f} \n'.format(train_mse, test_mse))
```

```
bag model (DecisionTreeRegressor)--> train mse: 1083.3992 ,  test mse: 4102538.7461
```

با n_estimators برابر ۱۵۰

```
bag_model = BaggingRegressor(base_estimator=RandomForestRegressor(), n_estimators=150)

bag_model.fit(train_x, train_y)
train_pred = bag_model.predict(train_x)
train_mse = mean_squared_error(train_y, train_pred)

test_pred = bag_model.predict(test_x)
test_mse = mean_squared_error(test_y, test_pred)

print('bagging model (RandomForestRegressor) --> train mse: {:0.4f} ,  test mse: {:0.4f} \n'.format(train_mse, test_mse))
```

```
bagging model (RandomForestRegressor) --> train mse: 3198.0355 ,  test mse: 4486981.0602
```

روش Boosting

```
boost_model = GradientBoostingRegressor()

boost_model.fit(train_x, train_y)
train_pred = boost_model.predict(train_x)
train_mse = mean_squared_error(train_y, train_pred)

test_pred = boost_model.predict(test_x)
test_mse = mean_squared_error(test_y, test_pred)

print('boost model (RandomForestRegressor) --> train mse: {:0.4f} ,  test mse: {:0.4f} \n'.format(train_mse, test_mse))
```

```
boost model (RandomForestRegressor) --> train mse: 1496.7909 ,  test mse: 4075739.9141
```

روش xgboost

```
xgb_model = XGBRegressor()

xgb_model.fit(train_x, train_y)
train_pred = xgb_model.predict(train_x)
train_mse = mean_squared_error(train_y, train_pred)

test_pred = xgb_model.predict(test_x)
test_mse = mean_squared_error(test_y, test_pred)

print('xgb_model --> train mse: {:0.4f} ,  test mse: {:0.4f} \n'.format(train_mse, test_mse))
```

```
[05:51:06] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
xgb_model --> train mse: 1678.5727 ,  test mse: 4076497.7814
```

روش random forest

با max_depth = 2

```
forest_model --> train mse: 519805.9049 ,  test mse: 18638898.3127
```

با max_depth = 4

```
forest_model --> train mse: 26673.4330 ,  test mse: 6130576.8858
```

با max_depth = 8

```
forest_model --> train mse: 1673.1792 ,  test mse: 4204163.9566
```

با max_depth = 10

```
forest_model --> train mse: 1216.2120 ,  test mse: 4138878.0938
```

سوال ۲۶ و ۲۷

در این قسمت از دو روش استفاده کردیم.

روش decision tree:

کد و نتایج آن به صورت زیر است.

```
clf = tree.DecisionTreeClassifier()

clf = clf.fit(train_x, train_y)

pred = clf.predict(train_x)

accuracy_score(train_y, pred)
1.0

test_pred = clf.predict(test_x)

accuracy_score(test_y, test_pred)
1.0
```

روش دوم استفاده از شبکه عصبی است. در این قسمت با استفاده از pytorch ابتدا یک دیتابیس میسازیم که به صورت زیر است.

```
class Mdataset(Dataset):
  def __init__(self, x, y):
    self.x = torch.tensor(x.values)
    self.y = torch.tensor(y.values)

  def __len__(self):
    return len(self.x)

  def __getitem__(self, idx):
    x = self.x[idx]
    y = self.y[idx]
    return (x, y)
```

مدل رو به صورت زیر تعریف میکنیم.

```
class Model(torch.nn.Module):

  def __init__(self):
    super(Model, self).__init__()
    self.conv = torch.nn.Conv1d(1, 256, 5)
    self.max_pool = torch.nn.AdaptiveAvgPool1d(1)
    self.flatten = torch.nn.Flatten()
    self.linear = torch.nn.Linear(256, 5)


  def forward(self, x):
    x = F.relu(self.conv(x))
    x = self.max_pool(x)
    x = F.relu(self.flatten(x))
    x = sel.linear(x)
    return x
```