دانشگاه فردوسی مشهد

Ferdowsi University of Mashhad

دانشکده مهندسی

داده کاوی

استاد محترم

جناب آقای بهروز بلوریان

ارائه دهندگان

منا رداد - زهرا ترویج الاسلامی

شماره دانشجویی

9712762261-9712762794

بهمن00

1

فاز دوم پروژه

1.1 اعمال الگوریتم خوشه‌بندی(پارامترهای مختلف بررسی شود)

خوشه بندی یکی از روش‌های یادگیری بدون نظارت است و هدف آن تقسیم بندی داده‌ها به خوشه‌های مختلف است به طوری که داده‌های درون یک خوشه بیشترین شباهت به یکدیگر را داشته باشند و از طرف دیگر داده‌های قرار گرفته در خوشه‌های مختلف بیشترین تفاوت را داشته باشند. خوشه بندی روشی است که هم در داده کاوی و هم در بینایی ماشین کاربرد دارد. از روش‌های رایج خوشه بندی می‌توان به خوشه بندی با الگوریتم K-means و DBSCAN اشاره کرد:

1.1.1 الگوریتم K-means

یکی از روش‌های خوشه بندی ساده و سریع است. این الگوریتم دارای یک پارامتر به نام k است که تعداد خوشه‌هایی که باید به دست آید را مشخص می‌کند. پارامترهای این الگوریتم شامل:

- n-clusters : مهم‌ترین پارامتر است و تعداد خوشه‌های تشکیل شده را مشخص می‌کند (پیش فرض =8)
- init : پارامتر مهم دیگری است که مقدار اولیه روش را برای استفاده تعریف می‌کند (پیش فرض = k-means++) الگوریتم k-means++ از تکنیک seed برای مقداردهی اولیه مرکز استفاده می‌کند که بهتر از روش اولیه سازی تصادفی است.

در این قسمت روی داده house_sales_prediction الگوریتم k-means را ابتدا با 2 خوشه و بعد با 4 خوشه طبقه بندی می‌کنیم:

ایمپورت کتابخانه‌های مورد نیاز پایتون

```
[27] from sklearn.cluster import KMeans
     import pandas as pd
     from sklearn.preprocessing import MinMaxScaler
     from matplotlib import pyplot as plt
     %matplotlib inline
     from sklearn.model_selection import cross_val_score
     from sklearn.model_selection import train_test_split
     print("done")

     done
```

تقسیم‌بندی داده‌ها به داده train و test

```
[28] features =["bedrooms", "bathrooms","sqft_living" ,"sqft_lot" ,"floors" ,"condition","grade","sq
     X = df[features]
     Y = df['price']

     x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.15, random_state=1)


     print("number of test samples:", x_test.shape[0])
     print("number of training samples:",x_train.shape[0])
```

```
number of test samples: 3242
number of training samples: 18371
```

الگـوریتم k-means بـا 2 cluster

```
[33] #k-means whit 2 clusters
     km = KMeans(n_clusters=2,init='k-means++', random_state=0)
     y_predicted = km.fit(df.drop(['date', 'price',],axis=1))
     y_predicted
```

```
KMeans(n_clusters=2, random_state=0)
```

مرکز cluster ها

```
[34] km.cluster_centers_
```

```
array([[ 3.37022038e+00,  2.10808297e+00,  2.06564574e+03,
         1.08526592e+04,  1.49272462e+00,  3.41170654e+00,
         7.64579017e+00,  1.77390766e+03,  1.97078584e+03,
         9.80785546e+04,  4.75614512e+01, -1.22217165e+02,
         1.97906772e+03,  1.00530023e+04],
       [ 3.40583554e+00,  2.49071618e+00,  2.88281167e+03,
         2.54747533e+05,  1.58355438e+00,  3.28116711e+00,
         8.28116711e+00,  2.60420424e+03,  1.98335809e+03,
         9.80433103e+04,  4.74812637e+01, -1.22029756e+02,
         2.40816180e+03,  1.65726992e+05]])
```

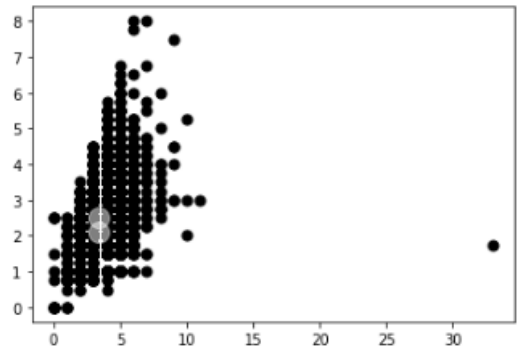
برچسب کلاس ها

```
[35] km.labels_
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int32)
```

رسم نمودار k-means با clusters 2

```
[37] c=df.drop(['date', 'price',],axis=1)
     plt.scatter(c.iloc[:, 0], c.iloc[:, 1], s=50, c='black')
     plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], c='white', s=200, alpha=0.5)
     plt.savefig("clusterflop")
```



در این قسمت برای معرفی معیارهای ارزیابی ابتدا باید برچسب داده هارا مشخص کنیم، ابتدا میانگین کلاس قیمت را مشخص میکنیم:

```
[38] df['price'].mean()

     540088.1417665294
```

برای هر سطر مقدار هزینه بالاتر از این میانگین باشد cluster برچسب 1 می‌گیرد در غیر اینصورت برچسب صفر می‌گیرد:

```
[39] def cluster_vals (cost):
         if cost > 540088.1417665294:
             return 1
         else:
             return 0
     df['Cluster']=df['price'].apply(cluster_vals)
```

```
[40] df.head(2)
```

bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above	yr_built	zipcode	lat	long	sqft_living15	sqft_lot15	Cluster
1.00	1180	5650	1.0	3	7	1180	1955	98178	47.5112	-122.257	1340	5650	0
2.25	2570	7242	2.0	3	7	2170	1951	98125	47.7210	-122.319	1690	7639	0

با بدست اوردن لیبل‌ها میتوانیم معیارهای ارزیابی را برای این روش محاسبه کنیم:

```
from sklearn.metrics import confusion_matrix,classification_report
print("CONFUSION MATRIX:\n\n ",confusion_matrix(df['Cluster'],km.labels_))
print("\nCLASSIFICATION REPORT:\n\n",classification_report(df['Cluster'],km.labels_))
```

```
CONFUSION MATRIX:

  [[13533   161]
 [ 7703   216]]

CLASSIFICATION REPORT:

              precision    recall  f1-score   support

           0       0.64      0.99      0.77     13694
           1       0.57      0.03      0.05      7919

    accuracy                           0.64     21613
   macro avg       0.61      0.51      0.41     21613
weighted avg       0.61      0.64      0.51     21613
```

همانطور که مشـاهده می‌شـود CONFUSION MATRIX و معیارهـای precision، recall، f1-score و accuracy محاسبه شده اند، دقت این مدل با 2 cluster به 0.64 رسیده است.

حال برای 4 cluster طبقه بندی را انجام می‌دهیم:

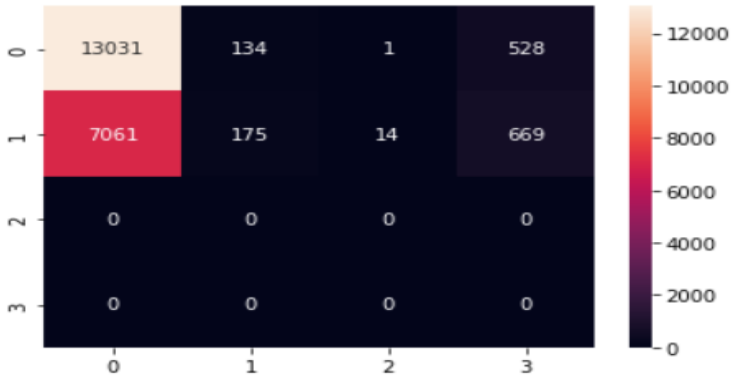
```
#k-means whit 4 clusters
km = KMeans(n_clusters=4,init='k-means++', random_state=0)
y_predicted = km.fit(df.drop(['date', 'price',],axis=1))
y_predicted
```

```
KMeans(n_clusters=4, random_state=0)
```

در این حالت معیارهای ارزیابی مطابق شکل زیر خواهند بود:

```
[68] from sklearn.metrics import confusion_matrix,classification_report
     print("CONFUSION MATRIX:\n\n ",confusion_matrix(df['Cluster'],km.labels_))
     print("\nCLASSIFICATION REPORT:\n\n",classification_report(df['Cluster'],km.labels_))
```

```
CONFUSION MATRIX:

  [[13031   134     1   528]
 [ 7061   175    14   669]
 [    0     0     0     0]
 [    0     0     0     0]]

CLASSIFICATION REPORT:

              precision    recall  f1-score   support

           0       0.65      0.95      0.77     13694
           1       0.57      0.02      0.04      7919
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0

    accuracy                           0.61     21613
   macro avg       0.30      0.24      0.20     21613
weighted avg       0.62      0.61      0.50     21613
```



همانطور که مشاهده می‌شود دقت با 4 cluster به 0.61 درصد رسیده است که دقت پایین تـری نسبت به 2 cluster می‌باشد. بنابراین الگوریتم k-means با 2 خوشه عملکرد بهتری دارد.

در روش supervised ، داده‌ها را به داده‌های اموزش و تست تقسـیم بنـدی مـی‌کنیم و مـدل k-means را بر روی این داده‌ها بدست می‌آوریم:

```
features =["bedrooms", "bathrooms","sqft_living" ,"sqft_lot" ,"floors" ,"condition","grade","sqft_above","yr_built","zipcode","lat","long"
X = df[features]
Y = df['Cluster']

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.15, random_state=1)


print("number of test samples:", x_test.shape[0])
print("number of training samples:",x_train.shape[0])
```

```
number of test samples: 3242
number of training samples: 18371
```

k-means با 4 clusters

در روش supervised

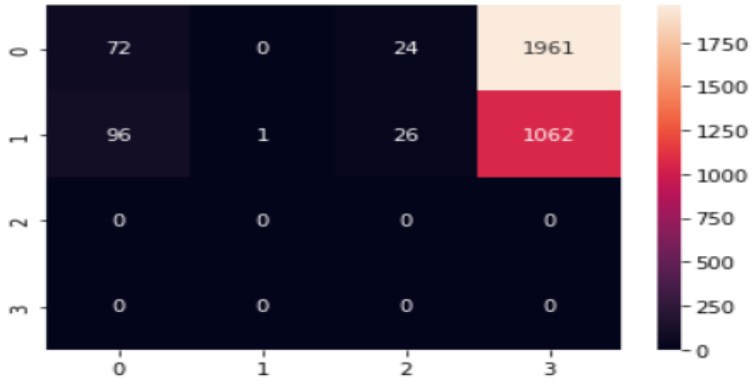
```
[30] #k-means whit 4 clusters
     km = KMeans(n_clusters=4,init='k-means++', random_state=0)
     y_predicted = km.fit(x_train,y_train)
     y_predicted
```

```
KMeans(n_clusters=4, random_state=0)
```

```
[31] predictions = km.predict(x_test)
```

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0	0.43	0.04	0.06	2057
1	1.00	0.00	0.00	1185
2	0.00	0.00	0.00	0
3	0.00	0.00	0.00	0
accuracy			0.02	3242
macro avg	0.36	0.01	0.02	3242
weighted avg	0.64	0.02	0.04	3242



در این روش با 4 خوشه دقت به 0.02 رسیده است که به عبارتی یعنی مدل با این روش آموزش ندیده است.

k-means با 2 clusters

در روش supervised

```
[34] #k-means whit 2 clusters
     km = KMeans(n_clusters=2,init='k-means++', random_state=0)
     y_predicted = km.fit(x_train,y_train)
     y_predicted

     KMeans(n_clusters=2, random_state=0)
```

```
[35] predictions = km.predict(x_test)
```

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

print(classification_report(y_test, predictions))
```

	precision	recall	f1-score	support
0	0.64	0.99	0.77	2057
1	0.53	0.03	0.05	1185
accuracy			0.64	3242
macro avg	0.58	0.51	0.41	3242
weighted avg	0.60	0.64	0.51	3242

در این روش با 2 خوشه دقت به 0.64 درصد رسیده است که نشان میدهد نسبت به 4 خوشه دقت بالاتری است.

1.1.2 الگوریتم DBSCAN

پارامترهای این الگوریتم شامل:

- Minpts : کمترین تعداد امتیاز مورد نیاز برای تشکیل یک خوشه
- Epsilon : حداکثر فاصله دو نقطه از یکدیگر در حالی که هنوز به یک خوشه تعلق دارند.

در این پروژه مدل را با مقادیر مختلف پارامتر اموزش دادیم ولی خروجی درستی برای این مدل به دست نمی آید.

```
[ ] from sklearn.cluster import DBSCAN

    epsilon =1.2
    min_samples = 50
    db = DBSCAN(eps=epsilon, min_samples=min_samples).fit(df.drop(['date', 'price',],axis=1))
```

```
[ ] db.labels_

    array([-1, -1, -1, ..., -1, -1, -1])
```

```
[ ] cluster_labels = pd.Series(db.labels_)
    cluster_labels.value_counts()

    -1    21613
    dtype: int64
```

```
[24] from sklearn.metrics import confusion_matrix,classification_report
     print("CONFUSION MATRIX:\n\n ",confusion_matrix(df['Cluster'],db.labels_))
     print("\nCLASSIFICATION REPORT:\n\n",classification_report(df['Cluster'],db.labels_))

     CONFUSION MATRIX:

       [[    0     0     0]
      [13694     0     0]
      [ 7919     0     0]]

     CLASSIFICATION REPORT:

                   precision    recall  f1-score   support

              -1       0.00      0.00      0.00       0.0
               0       0.00      0.00      0.00   13694.0
               1       0.00      0.00      0.00    7919.0

        accuracy                           0.00   21613.0
       macro avg       0.00      0.00      0.00   21613.0
    weighted avg       0.00      0.00      0.00   21613.0
```

الگوریتم dbscan را یک بار بر روی داده های نرمالیزه شده توسط pca هم بدست می آوریم:

```
[38] from sklearn.cluster import DBSCAN

     epsilon =1
     min_samples = 60
     db = DBSCAN(eps=epsilon, min_samples=min_samples).fit(X_r)
```

```
[39] db.labels_

     array([-1, -1, -1, ..., -1, -1, -1])
```

```
[40] cluster_labels = pd.Series(db.labels_)
     cluster_labels.value_counts()

     -1    18555
      0      269
      2      189
      5      111
      3       83
      4       82
      1       70
      6       60
     dtype: int64
```

در روش supervised مدل را بروی داده‌های اموزش و تست، اموزش میدهیم:

```
[47] from sklearn.cluster import DBSCAN

     epsilon =1
     min_samples = 60
     db = DBSCAN(eps=epsilon, min_samples=min_samples).fit(x_train,y_train)
```

```
[48] db.labels_

     array([-1, -1, -1, ..., -1, -1, -1])
```

```
[49] cluster_labels = pd.Series(db.labels_)
     cluster_labels.value_counts()

     -1    18371
     dtype: int64
```

```
[54] db.labels_[:3242].shape

     (3242,)
```

```
[55] from sklearn.metrics import confusion_matrix,classification_report
     print("CONFUSION MATRIX:\n\n ",confusion_matrix(y_test,db.labels_[:3242]))
     print("\nCLASSIFICATION REPORT:\n\n",classification_report(y_test,db.labels_[:3242]))

     CONFUSION MATRIX:

      [[   0    0    0]
      [2057    0    0]
      [1185    0    0]]

     CLASSIFICATION REPORT:

                   precision    recall  f1-score   support

               -1       0.00      0.00      0.00       0.0
                0       0.00      0.00      0.00    2057.0
                1       0.00      0.00      0.00    1185.0

         accuracy                           0.00    3242.0
        macro avg       0.00      0.00      0.00    3242.0
     weighted avg       0.00      0.00      0.00    3242.0
```

با مقایسه روش بانظارت و بدون نظارت به این نتیجه میرسیم که روش بدون نظارت درصورتی که بـروی داده های نرمالش شده pca اموزش ببیند خوشه بندی جواب می‌دهد. بـا ایـن حـال بـین ایـن دو مـدل خوشه بند، k-means با cluster 2 و با دقت 0.64 درصد بهترین عملکرد را دارد.

1.2 اعمال الگوریتم طبقه‌بندی

1.2.1 SVM

ماشین بردار پشتیبان یکی از الگوریتم‌های نظارت‌شـده یـادگیری ماشـین اسـت. کـه از آن بـرای طبقه‌بندی و رگرسیون استفاده می‌کنند. مبنای کاری دسته‌بندی کننده SVM دسته‌بندی خطی داده‌ها است و در تقسیم خطی داده‌ها سعی می‌کنیم خطی را انتخاب کنیم که حاشیه اطمینان بیشتری داشـته باشد. ماشین بردار پشتیبان یک الگوریتم دسته‌بندی بسیار قدرتمند است. وقتی از آن همراه با الگـوریتم های جنگل تصادفی و دیگر ابزارهای یادگیری ماشین استفاده کنیم، این الگوریتم می‌تواند مـدلی بسـیار قابل توجه برای دسته‌بندی داده‌ها ارائه کند. الگوریتم ماشین بردار پشتیبان هنگامی که قدرت پیش‌بینی بالا مورد نیاز باشد یک گزینه بسیار عالی است.

در این قسمت با استفاده از طبقه‌بند svm طبقه بندی را برای دیتاست house_sales_prediction با اعمال پارامترهای مختلف svm، انجام می‌دهیم تا تغییر دقت را برای هر پارامتر مشاهده کنیم.

برای اینکه طبقه‌بند Support Vector Machine را آموزش دهیم ابتدا کتابخانه‌های مورد نیاز پایتون برای راه اندازی مدل svm ایمپورت می‌کنیم. برای طبقه‌بندی svm از دستور SVC(kernel='linear') استفاده می‌کنیم. به ازای پارامترهای مختلف طبقه بندی را انجام میدهیم تا درنهایت بهترین پارامترها را برای طبقه بندی بدست اوریم و شبکه را با این پارامترها اموزش دهیم.

این پارامترها شامل:

1- c این پارامتر همان ضریب c است که به ضریب پنالتی نیز معروف است، این پارامتر به صورت دیفالت 1 درنظر گرفته می‌شود و می‌توانیم این مقدار را کم و زیاد کنیم.

2- KernelFunction -این پارامتر نوع کرنل را مشخص می‌کند که می‌تواند linear، gaussian، rbf و polynomial باشد.

3-CacheSize گاهی svm به حافظه زیادی نیاز دارد، این پارامتر حافظه مورد نیاز را در اختیار ما قرار می‌دهد.

4-gama این پارامتر ضریب گاما را نشان می‌دهد که در حالت پیش فرض این ضریب 0.1 در نظر گرفته می‌شود.

در این قسمت svm را به ازای پارامترهای مختلف بدست میاوریم و در نهایت برای بهترین پارامترها مدل را اموزش می‌دهیم.

تعـداد ضـریب پنـالتی را 1 ، 10، 100 و 1000 درنظر میگیریم و محاسبه میکنیم که به ازای چه ضـریبی بهتـرین جـواب را میگیریم

کرنل را rbf درنظر میگیریم

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

parameters = {'kernel':['rbf'],
              'C':[1, 10, 100, 1000],
              'gamma':[1e-3, 1e-4],'probability':[True]}
clf = GridSearchCV(SVC(), parameters)
clf.fit(x_train, y_train)
```

```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                         'kernel': ['rbf'], 'probability': [True]})
```

بهترین پارامترها شامل ضـریب c بـا مقدار 1 و گامای 0.0001 می‌باشد.

```
[34] svmclf = clf.best_estimator_
     svmclf.fit(x_train, y_train)

     SVC(C=1, gamma=0.0001, probability=True)
```

```
[35] y_testSVM = svmclf.predict(x_test)

     from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

     print(classification_report(y_test, y_testSVM))
     print("Accuracy: {0}".format(accuracy_score(y_test, y_testSVM)))

                   precision    recall  f1-score   support

                0       0.64      1.00      0.78      2057
                1       0.60      0.01      0.01      1185

         accuracy                           0.64      3242
        macro avg       0.62      0.50      0.40      3242
     weighted avg       0.62      0.64      0.50      3242

     Accuracy: 0.6354102405922271
```

دقت در این حالت 0.63 درصد بدست می‌آید.

1.2.2 KNN

الگوریتم K نزدیکترین همسایه (KNN) نوعی از الگوریتم‌های یادگیری ماشینی نظارت شده است. پیاده سازی KNN در ابتدایی ترین شکل آن بسیار آسان است و در عین حال وظایف طبقه بندی بسـیار پیچیده‌ای را انجام می‌دهد. این الگوریتم lazy بوده زیرا فاز آموزشی تخصصی ندارد. بلکه از تمام داده‌هـا برای آموزش در حین طبقه‌بندی یک نقطه یا نمونه داده جدید استفاده می‌کنـد. پارامترهـای ایـن مـدل شامل:

1-n_neighbors : تعداد همسایه‌ها

2- weights : که بر اساس uniform باشد یا distance

3-metric : که نوع فاصله را نشان می‌دهید، minkowski، Euclidean وmanhattan باشد.

تعـــداد همســـایه‌ها را 1 ، 5، 10 و 30 درنظر میگیریم و محاسبه میکنیم کـه بـه ازای چه تعداد همسایه بهتـرین جـواب را

Knn را بـــه ازای پارامترهـــای مختلـف بدســت میـاوریم و در نهایـت بـرای بهتـرین پارامترهـا مدل را اموزش میدهیم

```
[ ]  from sklearn.neighbors import KNeighborsClassifier
     from sklearn.model_selection import GridSearchCV

     parameters = {"n_neighbors": [1, 5, 10, 30],
                   "weights": ['uniform', 'distance'],
                   "metric": ['minkowski','euclidean','manhattan'],
                   "algorithm": ['auto', 'ball_tree', 'kd_tree', 'brute']}
     kclf = KNeighborsClassifier()
     kgclf = GridSearchCV(kclf, param_grid=parameters)
     kgclf.fit(x_train, y_train)
```

```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'metric': ['minkowski', 'euclidean', 'manhattan'],
                         'n_neighbors': [1, 5, 10, 30],
                         'weights': ['uniform', 'distance']})
```

بهتـرین پارامترهـا شـامل فاصـله بـا manhattan، تعـداد همســایگی 30 می‌باشد.

```
[ ]  kclf = kgclf.best_estimator_
     kclf.fit(x_train, y_train)
```

```
KNeighborsClassifier(metric='manhattan', n_neighbors=30, weights='distance')
```

```
y_testKNN = kclf.predict(x_test)

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score


print(classification_report(y_test, y_testKNN))
print("Accuracy: {0}".format(accuracy_score(y_test, y_testKNN)))
```

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84      2057
           1       0.76      0.63      0.69      1185

    accuracy                           0.79      3242
   macro avg       0.78      0.76      0.77      3242
weighted avg       0.79      0.79      0.79      3242

Accuracy: 0.7930289944478717
```

دقت بدست آمده با استفاده از این روش 0.79 درصد است.

1.2.3 BAYES

برای مدل بیز از طبقه بند naïve-bayes استفاده میکنیم که یک مدل احتمالاتی است.

```
[ ]  from sklearn.naive_bayes import GaussianNB

     model = GaussianNB()
     model.fit(x_train, y_train)

     GaussianNB()
```

```
[ ]  y_pred = model.predict(x_test)
```

مدل نایوبیز را روی داده های train آموزش میدهیم و روی داده های تست predict میکنیم.

```
[ ] from sklearn.metrics import classification_report, confusion_matrix
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

رســـــــم confusion_matrix

ومعیارهای ارزیابی

```
[[1843  214]
 [ 539  646]]
              precision    recall  f1-score   support

           0       0.77      0.90      0.83      2057
           1       0.75      0.55      0.63      1185

    accuracy                           0.77      3242
   macro avg       0.76      0.72      0.73      3242
weighted avg       0.77      0.77      0.76      3242
```

```
model.score(x_test,y_test)
```

```
0.7677359654534238
```

دقت در این روش 0.76 درصد بدست آمده است.

در حالت بعدی مدل را به ازای پارامترهای priors و var_smoothing اموزش می‌دهیم، در حالت دیفالت var_smoothing مقدار 1e-09 را دارد در این جا ما این مقدار را به 1 تغییر می‌دهیم:

```
[88] from sklearn.naive_bayes import GaussianNB

     model = GaussianNB(priors=None,var_smoothing=1)
     model.fit(x_train, y_train )
```

```
     GaussianNB(var_smoothing=1)
```

```
[89] y_pred = model.predict(x_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[2019   38]
 [1144   41]]
              precision    recall  f1-score   support

           0       0.64      0.98      0.77      2057
           1       0.52      0.03      0.06      1185

    accuracy                           0.64      3242
   macro avg       0.58      0.51      0.42      3242
weighted avg       0.59      0.64      0.51      3242
```

دقت در این حالت مقدار 0.64 درصد بدست آمده است.

با مقایسه درمیابیم دقت در حالت پیش فرض پارامترها مقدار بالاتری دارد.

1.2.4 DT

یکی از پرکاربردترین الگوریتم‌های داده‌کاوی، الگوریتم درخت تصمیم است. در داده‌کاوی، درخت تصمیم یک مدل پیش‌بینی کننده است به طوری که می تواند برای هر دو مدل رگرسیون و طبقه‌ای مورد استفاده قرار گیرد. زمانی که درخت برای کارهای طبقه بندی استفاده می شود، به عنوان درخت طبقه بندی (Classification Tree) شناخته می‌شود و هنگامی که برای فعالیت‌های رگرسیونی به کار می‌رود درخت رگرسیون (Regression Decision Tree) نامیده می‌شود.

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

در این پروژه درخت تصمیم را برای داده‌های house_sales_prediction بدست می‌اوریم:

درخت تصمیم را روی داده‌های train اموزش میدهیم و بروی داده های test، predict میکنیم.

```
[ ] from sklearn.tree import DecisionTreeClassifier
    clf = DecisionTreeClassifier()

    clf = clf.fit(x_train,y_train)
```

```
[ ] y_pred = clf.predict(x_test)
```

```
[ ] from sklearn.metrics import classification_report, confusion_matrix
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

```
[[1835  222]
 [ 203  982]]
              precision    recall  f1-score   support

           0       0.90      0.89      0.90      2057
           1       0.82      0.83      0.82      1185

    accuracy                           0.87      3242
   macro avg       0.86      0.86      0.86      3242
weighted avg       0.87      0.87      0.87      3242
```

دقت بدست آمده با این مدل 0.87 درصد است.

در حالت بعد به ازای پارامترهای مختلف مدل را آموزش می‌دهیم، در این حالت عمق درخت 3، criterion='entropy' و splitter='best' در نظر گرفته شده است.

```
[66] from sklearn.tree import DecisionTreeClassifier
     clf = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0,splitter='best')

     clf = clf.fit(x_train,y_train)
```

```
[67] y_pred = clf.predict(x_test)
```

```
[68] from sklearn.metrics import classification_report, confusion_matrix
     print(confusion_matrix(y_test, y_pred))
     print(classification_report(y_test, y_pred))
```

```
[[1786  271]
 [ 240  945]]
              precision    recall  f1-score   support

           0       0.88      0.87      0.87      2057
           1       0.78      0.80      0.79      1185

    accuracy                           0.84      3242
   macro avg       0.83      0.83      0.83      3242
weighted avg       0.84      0.84      0.84      3242
```

دقت در این حالت به 0.84 درصد رسیده است.

در حالت بعد splitter='random' در نظر میگیریم، در این حالت داریم:

```
[69] from sklearn.tree import DecisionTreeClassifier
     clf = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0,splitter='random')

     clf = clf.fit(x_train,y_train)
```

```
[70] y_pred = clf.predict(x_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[1628  429]
 [ 363  822]]
              precision    recall  f1-score   support

           0       0.82      0.79      0.80      2057
           1       0.66      0.69      0.67      1185

    accuracy                           0.76      3242
   macro avg       0.74      0.74      0.74      3242
weighted avg       0.76      0.76      0.76      3242
```

دقت در این حالت به 0.76 درصد رسیده است.

با مقایسه این حالت‌ها، درمیابیم که حالت پیش‌فرض دقت بالاتری دارد.

1.2.5 ensemble

در روش ensemble بـا تعیـین پارامترهـایی ماننـد max-depth، n-estimators و learning-rate طبقه بند را اموزش می‌دهیم.

```
from sklearn import ensemble
clf = ensemble.GradientBoostingRegressor(n_estimators = 400, max_depth = 5, min_samples_split = 2,
          learning_rate = 0.1, loss = 'ls')

clf= clf.fit(x_train, y_train)
```

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

```
clf.score(x_test,y_test)
```

```
0.7441796692567957
```

دقت بدست امده با استفاده از این روش 0.74 درصد است.

مدل bagging را برروی الگوریتم درخت تصمیم اموزش می‌دهیم در این حالت داریم:

```
[26] from sklearn import model_selection
     from sklearn.ensemble import BaggingClassifier
     from sklearn.tree import DecisionTreeClassifier
     import pandas as pd
```

```
base_cls = DecisionTreeClassifier()
# no. of base classifier
num_trees = 500

# bagging classifier
model = BaggingClassifier(base_estimator = base_cls,
                          n_estimators = num_trees,
                          ).fit(x_train,y_train)

results = model_selection.cross_val_score(model, x_test,y_test)
print("accuracy :")
print(results.mean())
```

```
accuracy :
0.9046881241796496
```

در این حالت دقت به 0.90 درصد می‌رسد که نشان دهنده این است که استفاده از bagging بروی دقت مدل بسیار تاثیر گذار است.

1.2.6 MLP

برای مدل mlp که شبکه عصبی می‌باشد از mlpclassifier استفاده میکنیم.

```
[ ] from sklearn.neural_network import MLPClassifier

    clf = MLPClassifier(hidden_layer_sizes=(2,),
                        random_state=5,
                        verbose=True,
                        learning_rate_init=0.01)
```

```
    clf.fit(x_train,y_train)
```

```
Iteration 1, loss = 2.81171933
Iteration 2, loss = 0.69970890
Iteration 3, loss = 0.66705122
Iteration 4, loss = 0.65907255
Iteration 5, loss = 0.65706344
Iteration 6, loss = 0.65701755
Iteration 7, loss = 0.65703631
Iteration 8, loss = 0.65703417
Iteration 9, loss = 0.65706129
Iteration 10, loss = 0.65705054
Iteration 11, loss = 0.65711018
Iteration 12, loss = 0.65703775
Iteration 13, loss = 0.65707936
Iteration 14, loss = 0.65710225
Iteration 15, loss = 0.65702881
Iteration 16, loss = 0.65709041
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
MLPClassifier(hidden_layer_sizes=(2,), learning_rate_init=0.01, random_state=5,
              verbose=True)
```

```
[ ] ypred=clf.predict(x_test)
```

```
[ ] from sklearn.metrics import accuracy_score

    # Calcuate accuracy
    accuracy_score(y_test,ypred)
```

```
0.634484885872918
```

```
[31] from sklearn.metrics import classification_report, confusion_matrix
     print(confusion_matrix(y_test, ypred))
     print(classification_report(y_test, ypred))
```

```
[[2057    0]
 [1185    0]]
              precision    recall  f1-score   support

           0       0.63      1.00      0.78      2057
           1       0.00      0.00      0.00      1185

    accuracy                           0.63      3242
   macro avg       0.32      0.50      0.39      3242
weighted avg       0.40      0.63      0.49      3242
```

دقت بدست آمده با استفاده از این روش 0.63 درصد است.

1.2.7 MLP+SVM

برای ترکیب شبکه عصبی mlp با یک لایه پنهان و مدل طبقه بند svm ابتدا باید داده ها را نرمال کنیم تا برای ورودی شبکه عصبی مناسب باشند.

```
[ ]  import tensorflow
     from tensorflow.keras.datasets import mnist
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.utils import to_categorical
```

ایمپورت کتابخانه های مورد نیاز شبکه عصبی

```
[ ]  from sklearn.preprocessing import MinMaxScaler
     scaler = MinMaxScaler()
     x_train = scaler.fit_transform(x_train)
     x_test = scaler.transform(x_test)
```

نرمال سازی داده ها برای ورودی مدل شبکه عصبی

معمـاری شـبکه بـا سـه لایـه و تعـداد نورون های 19 و 1

مـدل را بـا دسـتور compile. بـرای اموزش اماده میکنیم.

```
[ ]  model = Sequential()

     model.add(Dense(19, activation='relu'))
     model.add(Dense(19,activation='relu'))
     model.add(Dense(1))

     model.compile(optimizer='adam', loss='categorical_crossentropy')
```

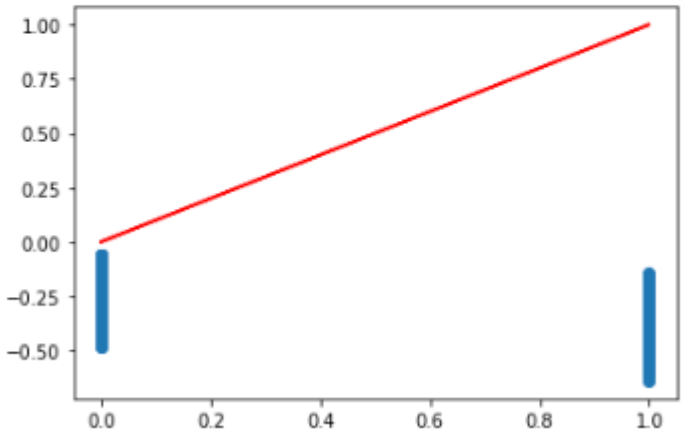
با دستور fit. مدل را اموزش مـی دهیم برای epochs 200

```
[ ]  model.fit(x=x_train, y=y_train, validation_data=(x_test, y_test), epochs=200)

     Epoch 1/200
     575/575 [==============================] - 2s 2ms/step - loss: 4.3697e-08 - val_loss: 4.3573e-08
     Epoch 2/200
     575/575 [==============================] - 1s 2ms/step - loss: 4.3697e-08 - val_loss: 4.3573e-08
     Epoch 3/200
```

```
[ ]  predictions = model.predict(x_test)
```

```
plt.scatter(y_test, predictions)
plt.plot(y_test, y_test, 'r')
```

```
[<matplotlib.lines.Line2D at 0x7fa75617f9d0>]
```



با استفاده از دستور زیر استخراج ویژگی را انجام میدهیم، تا با استفاده از این ویژگی ها مدل برای طبقه بندی با svm و knn اماده شود.

```
from keras import backend as K

f = K.function([model.layers[0].input],
                                    [model.layers[2].output])
```

ویژگی های داده های train و test

```
[ ] exTrain = f([x_train[:18000]])[0]

[ ] exTest = f([x_train[:3000]])[0]

[ ] exTrain[0]

    array([-0.22573474], dtype=float32)

[ ] exTest.shape

    (3000, 1)
```

برچسب داده های train و test

```
[ ] Y_train = y_train[:18000]
    Y_test = y_test[:3000]

[ ] print(exTrain.shape, exTest.shape, Y_train.shape, Y_test.shape)

    (18000, 1) (3000, 1) (18000,) (3000,)
```

امـوزش بـا طبقـه بنـد svm بـا
بهترین پارامترهای معرفی شده

```
[ ]  from sklearn.svm import SVC
     from sklearn.model_selection import GridSearchCV

     parameters = {'kernel':['rbf'],
                   'C':[1, 10, 100, 1000],
                   'gamma':[1e-3, 1e-4],'probability':[True]}
     clf = GridSearchCV(SVC(), parameters)
     clf.fit(exTrain, Y_train)
```

```
GridSearchCV(estimator=SVC(),
             param_grid={'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                         'kernel': ['rbf'], 'probability': [True]})
```

```
[ ]  svmclf = clf.best_estimator_
     svmclf.fit(exTrain, Y_train)
```

```
SVC(C=1000, gamma=0.0001, probability=True)
```

```
y_testSVM = svmclf.predict(exTest)

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

print(classification_report(Y_test, y_testSVM))
print("Accuracy: {0}".format(accuracy_score(Y_test, y_testSVM)))
```

```
              precision    recall  f1-score   support

           0       0.62      0.75      0.68      1897
           1       0.34      0.22      0.27      1103

    accuracy                           0.56      3000
   macro avg       0.48      0.49      0.48      3000
weighted avg       0.52      0.56      0.53      3000

Accuracy: 0.5576666666666666
```

دقت مدل mlp+svm 0.55 درصد بدست امده است.

MLP+KNN 1.2.8

امـوزش بـا طبقـه بنـد knn بـا بهترین پارامترهای معرفی شده

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

parameters = {"n_neighbors": [1, 5, 10, 30],
              "weights": ['uniform', 'distance'],
              "metric": ['minkowski','euclidean','manhattan'],
              "algorithm": ['auto', 'ball_tree', 'kd_tree', 'brute']}
kclf = KNeighborsClassifier()
kgclf = GridSearchCV(kclf, param_grid=parameters)
kgclf.fit(exTrain, Y_train)
```

```
GridSearchCV(estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'metric': ['minkowski', 'euclidean', 'manhattan'],
                         'n_neighbors': [1, 5, 10, 30],
                         'weights': ['uniform', 'distance']})
```

```
kclf = kgclf.best_estimator_
kclf.fit(exTrain, Y_train)
```

```
KNeighborsClassifier(n_neighbors=30)
```

```
kclf = kgclf.best_estimator_
kclf.fit(exTrain, Y_train)
```

```
KNeighborsClassifier(n_neighbors=30)
```

```
y_testKNN = kclf.predict(exTest)

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score

#print_cmx(y_test1000.T[0], y_testKNN)
print(classification_report(Y_test, y_testKNN))
print("Accuracy: {0}".format(accuracy_score(Y_test, y_testKNN)))
```

```
              precision    recall  f1-score   support

           0       0.63      0.81      0.71      1897
           1       0.35      0.18      0.23      1103

    accuracy                           0.58      3000
   macro avg       0.49      0.49      0.47      3000
weighted avg       0.53      0.58      0.53      3000

Accuracy: 0.5756666666666667
```

دقت مدل mlp+knn 0.57 درصد بدست آمده است.

1.3 مقایسه مدل‌ها

	accuracy	precision	recall	F1-score
k-means	0.64	0.61	0.64	0.51
SVM	0.63	0.62	0.64	0.50
KNN	0.79	0.79	0.79	0.79
NB	0.77	0.77	0.77	0.76
DT	0.87	0.87	0.87	0.87
ensemble	0.74	-	-	-
MLP	0.63	0.40	0.63	0.49
MLP+SVM	0.55	0.52	0.56	0.53
MLP+KNN	0.57	0.53	0.58	0.53

1.4 الگوهای پرتکرار و قوانین انجمنی

در این حالت برای گزارش الگوهای پرتکرار و قوانین انجمنی از دستور apyori استفاده می‌نماییم:

```
[74] import numpy as nm
     import matplotlib.pyplot as mtp
     import pandas as pd
```

```
[75] #Importing the dataset
     dataset = pd.read_csv('house_sales_prediction.csv')
     transactions=[]
     for i in range(0, 7501):
         transactions.append([str(dataset.values[i,j])  for j in range(0,20)])
```

```
[76] from apyori import apriori
     rules= apriori(transactions= transactions, min_support=0.003, min_confidence = 0.2, min_lift=3, min_length=2, max_length=2)
```

```
[77] results= list(rules)
     results
```

```
[RelationRecord(items=frozenset({'1.5', '1926'}), support=0.004932675643247567, ordered_statistics=[OrderedStatistic(
 RelationRecord(items=frozenset({'1927', '1.5'}), support=0.003199573390214638, ordered_statistics=[OrderedStatistic(
 RelationRecord(items=frozenset({'1.5', '1928'}), support=0.0035995200639914677, ordered_statistics=[OrderedStatistic
 RelationRecord(items=frozenset({'1.5', '1929'}), support=0.003199573390214638, ordered_statistics=[OrderedStatistic(
 RelationRecord(items=frozenset({'10', '3.25'}), support=0.005065991201173177, ordered_statistics=[OrderedStatistic(i
 RelationRecord(items=frozenset({'3.5', '10'}), support=0.006532462338354886, ordered_statistics=[OrderedStatistic(it
 RelationRecord(items=frozenset({'98040', '10'}), support=0.003332888948140248, ordered_statistics=[OrderedStatistic(
 RelationRecord(items=frozenset({'10', '98075'}), support=0.005732568990801226, ordered_statistics=[OrderedStatistic(
 RelationRecord(items=frozenset({'3.5', '11'}), support=0.0041327822956939075, ordered_statistics=[OrderedStatistic(i
 RelationRecord(items=frozenset({'1942', '6'}), support=0.006399146780429276, ordered_statistics=[OrderedStatistic(it
 RelationRecord(items=frozenset({'1943', '6'}), support=0.0038661511798426876, ordered_statistics=[OrderedStatistic(i
 RelationRecord(items=frozenset({'6', '1944'}), support=0.003999466737768298, ordered_statistics=[OrderedStatistic(it
 RelationRecord(items=frozenset({'1948', '6'}), support=0.003999466737768298, ordered_statistics=[OrderedStatistic(it
 RelationRecord(items=frozenset({'1984', '2.25'}), support=0.004266097853619517, ordered_statistics=[OrderedStatistic
 RelationRecord(items=frozenset({'2', '820'}), support=0.003999466737768298, ordered_statistics=[OrderedStatistic(ite
 RelationRecord(items=frozenset({'9', '2014'}), support=0.004266097853619517, ordered_statistics=[OrderedStatistic(it
 RelationRecord(items=frozenset({'6', '860'}), support=0.0030662578322890282, ordered_statistics=[OrderedStatistic(it
 RelationRecord(items=frozenset({'6', '98146'}), support=0.004532728969470737, ordered_statistics=[OrderedStatistic(i
 RelationRecord(items=frozenset({'98168', '6'}), support=0.0061325156645780565, ordered_statistics=[OrderedStatistic(
 RelationRecord(items=frozenset({'98178', '6'}), support=0.004532728969470737, ordered_statistics=[OrderedStatistic(i
 RelationRecord(items=frozenset({'9', '98029'}), support=0.005065991201173177, ordered_statistics=[OrderedStatistic(i
 RelationRecord(items=frozenset({'98074', '9'}), support=0.006932409012131715, ordered_statistics=[OrderedStatistic(i
```

```
[78] for item in results:
         pair = item[0]
         items = [x for x in pair]
         print("Rule: " + items[0] + " -> " + items[1])

         print("Support: " + str(item[1]))
         print("Confidence: " + str(item[2][0][2]))
         print("Lift: " + str(item[2][0][3]))
         print("====================================")
```

```
Rule: 1.5 -> 1926
Support: 0.004932675643247567
Confidence: 0.5068493150684932
Lift: 3.2439221094955353
====================================
Rule: 1927 -> 1.5
Support: 0.003199573390214638
Confidence: 0.5853658536585367
Lift: 3.7464413551985354
====================================
Rule: 1.5 -> 1928
Support: 0.0035995200639914677
Confidence: 0.6136363636363636
Lift: 3.927377443375737
====================================
Rule: 1.5 -> 1929
Support: 0.003199573390214638
Confidence: 0.6315789473684211
Lift: 4.042213041135262
====================================
Rule: 10 -> 3.25
Support: 0.005065991201173177
```