دانشکده ریاضی

گروه علوم کامپیوتر و داده

گزارش تمرین شماره دو

درس داده کاوی

نام دانشجو

محمد گرامی فر

اردیبهشت ماه ۱۴۰۱

فهرست گزارش

دیتاست شماره ۱

در ابتدا کتابخانه‌هایی که نیاز داریم را وارد می‌کنیم:

```
[1]  # import libraries

     import numpy as np
     import pandas as pd

     from sklearn.model_selection import train_test_split
     from itertools import combinations
     from sklearn.base import clone
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import roc_auc_score, RocCurveDisplay, classification_report
     from sklearn.preprocessing import StandardScaler
     from sklearn.decomposition import PCA
     from sklearn.svm import SVC
     from sklearn.tree import DecisionTreeClassifier

     import matplotlib.pyplot as plt

     import warnings
     warnings.filterwarnings("ignore")
```

ابتدا فایل دیتا را با استفاده از کتابخانه pandas بارگذاری می‌کنیم:

```
[2]  #reading the file

     df = pd.read_csv('train.csv')

     # first 5 rows of the data

     df.head()
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	0	1	1
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	1	0	2
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	1	0	2
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	0	0	2
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	1	0	1

5 rows × 21 columns

به شکل دیتاست نگاهی می‌اندازیم:

```
[3]  # shap of the data

     df.shape
```

```
     (2000, 21)
```

با استفاده از متد info به نوع دیتاها را بررسی می‌نماییم. همچنین اگر در ستونی مقادیر از دست رفته وجود داشته باشد می‌توانیم آن را مشاهده کنیم:

```
[4]  # data types and null values of the train set

     df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   blue           2000 non-null   int64
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64
 4   fc             2000 non-null   int64
 5   four_g         2000 non-null   int64
 6   int_memory     2000 non-null   int64
 7   m_dep          2000 non-null   float64
 8   mobile_wt      2000 non-null   int64
 9   n_cores        2000 non-null   int64
 10  pc             2000 non-null   int64
 11  px_height      2000 non-null   int64
 12  px_width       2000 non-null   int64
 13  ram            2000 non-null   int64
 14  sc_h           2000 non-null   int64
 15  sc_w           2000 non-null   int64
 16  talk_time      2000 non-null   int64
 17  three_g        2000 non-null   int64
 18  touch_screen   2000 non-null   int64
 19  wifi           2000 non-null   int64
 20  price_range    2000 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 328.2 KB
```

با استفاده از متد describe شاخصه‌های آماری دیتاست را بررسی می‌کنیم:

```
[5]  df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
battery_power	2000.0	1238.51850	439.418206	501.0	851.75	1226.0	1615.25	1998.0
blue	2000.0	0.49500	0.500100	0.0	0.00	0.0	1.00	1.0
clock_speed	2000.0	1.52225	0.816004	0.5	0.70	1.5	2.20	3.0
dual_sim	2000.0	0.50950	0.500035	0.0	0.00	1.0	1.00	1.0
fc	2000.0	4.30950	4.341444	0.0	1.00	3.0	7.00	19.0
four_g	2000.0	0.52150	0.499662	0.0	0.00	1.0	1.00	1.0
int_memory	2000.0	32.04650	18.145715	2.0	16.00	32.0	48.00	64.0
m_dep	2000.0	0.50175	0.288416	0.1	0.20	0.5	0.80	1.0
mobile_wt	2000.0	140.24900	35.399655	80.0	109.00	141.0	170.00	200.0
n_cores	2000.0	4.52050	2.287837	1.0	3.00	4.0	7.00	8.0
pc	2000.0	9.91650	6.064315	0.0	5.00	10.0	15.00	20.0
px_height	2000.0	645.10800	443.780811	0.0	282.75	564.0	947.25	1960.0
px_width	2000.0	1251.51550	432.199447	500.0	874.75	1247.0	1633.00	1998.0
ram	2000.0	2124.21300	1084.732044	256.0	1207.50	2146.5	3064.50	3998.0
sc_h	2000.0	12.30650	4.213245	5.0	9.00	12.0	16.00	19.0
sc_w	2000.0	5.76700	4.356398	0.0	2.00	5.0	9.00	18.0
talk_time	2000.0	11.01100	5.463955	2.0	6.00	11.0	16.00	20.0
three_g	2000.0	0.76150	0.426273	0.0	1.00	1.0	1.00	1.0
touch_screen	2000.0	0.50300	0.500116	0.0	0.00	1.0	1.00	1.0
wifi	2000.0	0.50700	0.500076	0.0	0.00	1.0	1.00	1.0
price_range	2000.0	1.50000	1.118314	0.0	0.75	1.5	2.25	3.0

طبق دستور مسئله، محدوده قیمت گوشی‌ها را به دو دسته تبدیل می‌کنیم. برای این کار ابتدا مقادیر یکتا ستون price_range را به دست می‌آوریم و نهایتا آن‌ها را به دو دسته کاهش می‌دهیم:

```
[6]  # unique values belong to price_range

     df['price_range'].unique()
```

```
     array([1, 2, 3, 0])
```

```
[7]  # change 4 classes of price to 2

     for i in df[df['price_range'] == 1].index.tolist():
       df.iloc[i, 20] = 0

     for i in df[df['price_range'] == 2].index.tolist():
       df.iloc[i, 20] = 1

     for i in df[df['price_range'] == 3].index.tolist():
       df.iloc[i, 20] = 1
```

```
[8]  # check the unique values again

     df['price_range'].unique()
```

```
     array([0, 1])
```

۱)

کلاس نوشته شده در پایین با گرفتن مدل یادگیری ماشین و تعداد ویژگی‌ها، با استفاده از روش Forward Selection بهترین مجموعه‌ی ویژگی‌ها را برمی‌گزیند.

```
[9]  # create a class for forward selection model

     class SequentialForwardSelection():
         def __init__(self, estimator, k_features):
             self.estimator = clone(estimator)
             self.k_features = k_features
         def fit(self, X_train, X_test, y_train, y_test):
             max_indices = tuple(range(X_train.shape[1]))
             total_features_count = len(max_indices)
             self.subsets_ = []
             self.scores_ = []
             self.indices_ = []
             scores = []
             subsets = []
             for p in combinations(max_indices, r=1):
                     score = self._calc_score(X_train.values, X_test.values,
                                              y_train.values, y_test.values, p)
                     scores.append(score)
                     subsets.append(p)
             best_score_index = np.argmax(scores)
             self.scores_.append(scores[best_score_index])
             self.indices_ = list(subsets[best_score_index])
             self.subsets_.append(self.indices_)
             dim = 1
             while dim < self.k_features:
                 scores = []
                 subsets = []
                 current_feature = dim
                 idx = 0
                 while idx < total_features_count:
                     if idx not in self.indices_:
                         indices = list(self.indices_)
                         indices.append(idx)
                         score = self._calc_score(X_train.values, X_test.values,
                                                  y_train.values, y_test.values,indices)
                         scores.append(score)
                         subsets.append(indices)
                     idx += 1
                 best_score_index = np.argmax(scores)
                 self.scores_.append(scores[best_score_index])
                 self.indices_ = list(subsets[best_score_index])
                 self.subsets_.append(self.indices_)
                 dim += 1
             self.k_score_ = self.scores_[-1]
         def transform(self, X):
             return X.values[:, self.indices_]
         def _calc_score(self, X_train, X_test, y_train, y_test, indices):
             self.estimator.fit(X_train[:, indices], y_train.ravel())
             y_pred = self.estimator.predict(X_test[:, indices])
             score = roc_auc_score(y_test, y_pred)
             return score
```

دیتاست را به فضای ویژگی‌ها و بردار هدف برای train و test تقسیم می‌کنیم:

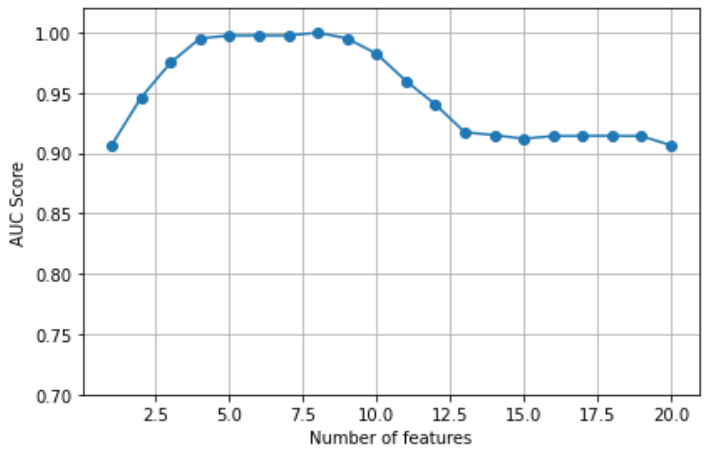
```
[ ]  # make a feature matrix (X) and a target vector (y)
     # and split dataset to train and test

     X_train, X_test, y_train, y_test = \
     train_test_split(df.iloc[:, :-1], df['price_range'],
                      test_size=0.20, random_state=2022)
```

مدل Logistic Regression را وارد Forward Selection می‌کنیم تا مجموعه بهترین ویژگی‌ها را بیابیم:

```
[ ]  # find the optimum model with best features

     lr = LogisticRegression(C=1.0, random_state=2022)

     n_features = 20 # number of features

     sfs = SequentialForwardSelection(lr, n_features)

     sfs.fit(X_train, X_test, y_train, y_test)
```

با مشاهده نمودار زیر پی می‌بریم که بهترین تعداد ویژگی‌ها، ۸ عدد از آن‌هاست:

```
[ ]  # plot AUC score for different combination of the features

     n_features = [len(n) for n in sfs.subsets_]
     plt.plot(n_features, sfs.scores_, marker='o')
     plt.ylim([0.7, 1.02])
     plt.ylabel('AUC Score')
     plt.xlabel('Number of features')
     plt.grid()
     plt.tight_layout()
     plt.show()
```



۲)

در این مرحله مدل Logistic Regression را بر روی ۸ ویژگی برتر اعمال می‌کنیم و معیارهای f1، recall و precision را برای آن به دست می‌آوریم

```
[ ]  # find the features that gives the best AUC score

     sfs.subsets_[7]
```

```
     [13, 11, 0, 12, 18, 19, 1, 8]
```

```
[ ]  # create a new df and sub datasets with above features

     df_new = df.iloc[:, [13, 11, 0, 12, 18, 19, 1, 8, 20]]

     X_train_new, X_test_new, y_train_new, y_test_new = \
     train_test_split(df_new.iloc[:, :-1], df_new['price_range'],
                      test_size=0.20, random_state=2022)
```

```
[ ]  # use Logistic Regression model with best features

     lr.fit(X_train_new, y_train_new)

     # report f1, recall and precision scores

     y_pred_new = lr.predict(X_test_new)

     print(f'Logistic Regression scores:\n')
     print(classification_report(y_test, y_pred_new))
```

```
     Logistic Regression scores:

                   precision    recall  f1-score   support

                0       1.00      1.00      1.00       192
                1       1.00      1.00      1.00       208

         accuracy                           1.00       400
        macro avg       1.00      1.00      1.00       400
     weighted avg       1.00      1.00      1.00       400
```

(۳

با استفاده از الگوریتم PCA تعداد ویژگی‌ها را به ۸ عدد کاهش می‌دهیم

```
[ ]  # PCA analysis

     sc = StandardScaler()

     X_train_st = sc.fit_transform(X_train)
     X_test_st = sc.transform(X_test)
```

```
[ ]  pca = PCA(n_components = 8)

     X_train_st = pca.fit_transform(X_train)
     X_test_st = pca.transform(X_test)
```

(۴

الگوریتم Logistic Regression را بر روی دیتاست با ویژگی‌های کاهش یافته پیاده می‌کنیم و معیارهای خواسته شده را برای آن بدست می‌آوریم

```
[ ]  # use Logistic Regression on analysed data

     lr.fit(X_train_st, y_train)

     # report f1, recall and precision scores

     y_pred = lr.predict(X_test_new)

     print(f'Logistic Regression scores after PCA analysis:\n')
     print(classification_report(y_test, y_pred))
```

```
Logistic Regression scores after PCA analysis:

              precision    recall  f1-score   support

           0       0.00      0.00      0.00       192
           1       0.52      1.00      0.68       208

    accuracy                           0.52       400
   macro avg       0.26      0.50      0.34       400
weighted avg       0.27      0.52      0.36       400
```

(۵

در الگوریتم ماشین بردار پشتیبان، داشتن یک خط راست بین این دو کلاس آسان است. وقتی داده ها خطی جدایی پذیر نباشند، باید از کلاس بندیهایی با مرزهای تصمیم گیری غیرخطی برای کلاس بندی داده ها استفاده کنیم. یکی از راه های کلاس بندی چنین داده هایی استفاده از توان دوم، سوم یا چندم متغیرهای ورودی است. ولی زمانی که وقتی از SVM استفاده می‌کنید، می‌توانیم از یک تکنیک ریاضی با نام Kernel Trickاستفاده کنید. استفاده از کرنل باعث میشود بدون اضافه کردن Polynomial Feature ، همان نتیجه‌ای رو بگیریم که انگار آن‌هارو اضافه کرده ایم. در این روش در واقع توابعی وجود دارند که فضای ورودی بُعد پایین را دریافت کرده و آن را به فضای بُعد بالاتر تبدیل می‌کنند. این تبدیل، یک مسئله غیر قابل جداسازی را به مسئله قابل جداسازی مبدل می‌کند. به این توابع، تابع‌های هسته (کرنل) گفته می‌شود.

از آنجایی که ما می خواهیم نمونه ها به صورت خطی در فضای ویژگی قابل تفکیک باشند، کیفیت فضای ویژگی برای عملکرد ماشینهای بردار پشتیبان حیاتی است. با این حال، ما نمیدانیم کدام توابع هسته خوب هستند، چرا که ما نگاشت ویژگی را نمیدانیم. بنابراین، انتخاب هسته بزرگترین عدم قطعیت ماشینهای بردار پشتیبان است. یک هسته ضعیف نمونه ها را به یک فضای ویژگی ضعیف نگاشت میکند و در نتیجه عملکرد ضعیفی دارد. به عبارت دیگر، نوع تابع هسته برای یک مساله معین از داده ها یاد گرفته نمیشود و باید آن را مشخص ک نیم. از اینرو، انتخاب تابع هسته یک ابرپارامتر است. برای مثال هر کدام از توابع کرنل در بعضی زمینه ها بیشتر کاربرد دارند:

کرنل چند جمله ای

این کرنل در پردازش تصویر پرکاربرد است. معادله آن به صورت زیر است:

$$k(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^d$$

کرنل گاوسی

این یک کرنل برای اهداف عمومی است. و هنگامی که هیچ دانش پیشینی در مورد داده ها وجود ندارد استفاده می شود. معادله آن به صورت زیر است:

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

تابع پایه شعاعی گاوسی (RBF)

این کرنلی برای اهداف عمومی کاربرد دارد. و هنگامی که هیچ دانش پیشینی در مورد داده ها وجود نداشته باشد، مورد استفاده قرار می گیرد. معادله آن به صورت زیر است:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

۶) و ۷)

بر روی دیتاســـت اصـــلی، مدل SVM را با پارامترهای مختلف محاســـبه می‌کنیم. پارامترهایی که تغییر داده می‌شوند ضریب C مدل و کرنل آن است. برای ۱۲ مدل SVM دقت‌های خواسته شده را هم چاپ می‌کنیم.

```
[ ]  # SVM

     svc_kernel = ['linear', 'poly', 'rbf', 'sigmoid']

     C_list = [0.001, 0.01, 0.1]

     def svm_analysis(c, Kernel, X_train, X_test, y_train, y_test):
       svc = SVC(C= c, kernel=Kernel)
       svc.fit(X_train, y_train)
       y_pred = svc.predict(X_test)
       print(f'SVM scores with C={c} and Kernel={Kernel}:\n')
       print(classification_report(y_test, y_pred), '\n\n\n')
```

```
[ ]  for i in svc_kernel:
       for j in C_list:
         svm_analysis(j, i, X_train, X_test, y_train, y_test)
```

```
SVM scores with C=0.001 and Kernel=linear:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       192
           1       1.00      1.00      1.00       208

    accuracy                           1.00       400
   macro avg       1.00      1.00      1.00       400
weighted avg       1.00      1.00      1.00       400


SVM scores with C=0.01 and Kernel=linear:

              precision    recall  f1-score   support

           0       0.99      1.00      1.00       192
```

```
           1     1.00        1.00        1.00        208

      accuracy                           1.00        400
   macro avg      1.00        1.00        1.00        400
weighted avg      1.00        1.00        1.00        400



SVM scores with C=0.1 and Kernel=linear:

              precision   recall  f1-score   support

           0     0.99        1.00        1.00        192
           1     1.00        1.00        1.00        208

      accuracy                           1.00        400
   macro avg      1.00        1.00        1.00        400
weighted avg      1.00        1.00        1.00        400



SVM scores with C=0.001 and Kernel=poly:

              precision   recall  f1-score   support

           0     0.73        1.00        0.84        192
           1     1.00        0.66        0.79        208

      accuracy                           0.82        400
   macro avg      0.87        0.83        0.82        400
weighted avg      0.87        0.82        0.82        400



SVM scores with C=0.01 and Kernel=poly:

              precision   recall  f1-score   support

           0     0.92        1.00        0.96        192
           1     1.00        0.92        0.96        208

      accuracy                           0.96        400
   macro avg      0.96        0.96        0.96        400
weighted avg      0.96        0.96        0.96        400

SVM scores with C=0.1 and Kernel=poly:

              precision   recall  f1-score   support
```

```
              0     0.96        1.00        0.98        192
              1     1.00        0.96        0.98        208

       accuracy                              0.98        400
   macro avg      0.98        0.98        0.98        400
weighted avg      0.98        0.98        0.98        400
```

SVM scores with C=0.001 and Kernel=rbf:

```
              precision   recall  f1-score   support

              0     0.48        1.00        0.65        192
              1     0.00        0.00        0.00        208

       accuracy                              0.48        400
   macro avg      0.24        0.50        0.32        400
weighted avg      0.23        0.48        0.31        400
```

SVM scores with C=0.01 and Kernel=rbf:

```
              precision   recall  f1-score   support

              0     0.93        0.94        0.94        192
              1     0.95        0.94        0.94        208

       accuracy                              0.94        400
   macro avg      0.94        0.94        0.94        400
weighted avg      0.94        0.94        0.94        400
```

SVM scores with C=0.1 and Kernel=rbf:

```
              precision   recall  f1-score   support

              0     0.95        0.99        0.97        192
              1     0.99        0.95        0.97        208

       accuracy                              0.97        400
   macro avg      0.97        0.97        0.97        400
weighted avg      0.97        0.97        0.97        400
```

SVM scores with C=0.001 and Kernel=sigmoid:

```
              precision    recall  f1-score   support

           0       0.48      1.00      0.65       192
           1       0.00      0.00      0.00       208

    accuracy                           0.48       400
   macro avg       0.24      0.50      0.32       400
weighted avg       0.23      0.48      0.31       400


SVM scores with C=0.01 and Kernel=sigmoid:

              precision    recall  f1-score   support

           0       0.63      0.72      0.67       192
           1       0.70      0.61      0.65       208

    accuracy                           0.66       400
   macro avg       0.67      0.66      0.66       400
weighted avg       0.67      0.66      0.66       400


SVM scores with C=0.1 and Kernel=sigmoid:

              precision    recall  f1-score   support

           0       0.51      0.49      0.50       192
           1       0.55      0.57      0.56       208

    accuracy                           0.53       400
   macro avg       0.53      0.53      0.53       400
weighted avg       0.53      0.53      0.53       400
```

۸)

در مدل SVM پارامتر c کنترل کننده soft یا hard بودن margin است اگر مقدار خیلی بزرگی به آن بدهیم (برای مثال ۱۰+e۱) مدل دارای hard margin است و برعکس. همان طور که در نتایج بالا مشاهده می‌شود برای SVM با کرنل چند جمله‌ای c=۰.۰۰۱ دقت برابر با ۸۲٪ بوده (soft margin) و برای SVM با کرنل چند جمله‌ای c=۰.۱ دقت برابر با ۹۸٪ بوده (hard margin).

۹ (آ)

در این قسمت بر روی ویژگی battery_power که دارای مقادیر پیوسته، binning انجام می‌دهیم و آن را به سه دسته low medium و high تبدیل می‌کنیم (تعداد هرکدام از این دسته‌ها متفاوت است)

```
[ ]  # bining 'battery_power' column

     print('First bin is battery powers less than ',
           (df['battery_power'].min() + df['battery_power'].mean()) /2, '\n')

     print('Second bin is battery powers between ', df['battery_power'].mean(),
           ' and ', (df['battery_power'].max()+df['battery_power'].mean()) / 2, '\n')

     print('Third bin is battery powers more than', df['battery_power'].max()+1)
```

```
First bin is battery powers less than  869.75925

Second bin is battery powers between  1238.5185  and  1618.25925

Third bin is battery powers more than 1999
```

```
[ ]  def get_bat_grp(battery_power):

         if battery_power < 869.75925:
             return 'low'

         elif (battery_power > 869.75925) & (battery_power < 1618.25925):
             return 'medium'

         elif (battery_power > 1618.25925) & (battery_power < 1999):
             return 'high'
```

```
[ ]  battery_bin = df['battery_power'].apply(get_bat_grp)

     df_bin = df.copy()

     df_bin.insert(1, 'battery_bin', battery_bin)

     df_bin.drop('battery_power', axis=1, inplace=True)

     df_bin.head()
```

	battery_bin	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	price_range
0	low	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	0	1	0
1	medium	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	1	0	1
2	low	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	1	0	1
3	low	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	0	0	1
4	high	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15	1	1	0	0

5 rows × 21 columns

۹(ب) و ۱۰

بر روی ویژگی‌های کیفی که تنها یک عدد از آن‌ها وجود دارد، OneHotEncoding را اعمال می‌کنیم و مدل‌های SVM که قبل تر گزارش شد را بر روی این دیتاست تغییر یافته هم اعمال می‌کنیم. همچنین معیارهای خواسته شده را برای آن‌ها نمایش می‌دهیم.

```
[ ]  # One Hot Encoding

     df_bin = pd.get_dummies(df_bin)

     price_column = df_bin.pop('price_range')

     df_bin.insert(22, 'price_range', price_column)

     df_bin.head()
```

	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	...	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	battery_bin_high	battery_bin_low	battery_bin_medium	price_range
0	0	2.2	0	1	0	7	0.6	188	2	2	...	9	7	19	0	0	1	0	1	0	0
1	1	0.5	1	0	1	53	0.7	136	3	6	...	17	3	7	1	1	0	0	0	1	1
2	1	0.5	1	2	1	41	0.9	145	5	6	...	11	2	9	1	1	0	0	1	0	1
3	1	2.5	0	0	0	10	0.8	131	6	9	...	16	8	11	1	0	0	0	1	0	1
4	1	1.2	0	13	1	44	0.6	141	2	14	...	8	2	15	1	1	0	1	0	0	0

5 rows × 23 columns

```
[ ]  X_train_bin, X_test_bin, y_train_bin, y_test_bin = \
     train_test_split(df_bin.iloc[:, :-1], df_bin['price_range'],
                      test_size=0.20, random_state=2022)

     for i in svc_kernel:
       for j in C_list:
         svm_analysis(j, i, X_train_bin, X_test_bin, y_train_bin, y_test_bin)
```

```
SVM scores with C=0.001 and Kernel=linear:

              precision    recall  f1-score   support

           0       0.90      0.98      0.94       192
           1       0.98      0.89      0.93       208

    accuracy                           0.94       400
   macro avg       0.94      0.94      0.93       400
weighted avg       0.94      0.94      0.93       400


SVM scores with C=0.01 and Kernel=linear:

              precision    recall  f1-score   support

           0       0.93      0.99      0.96       192
```

```
           1     0.99        0.93        0.96        208

      accuracy                           0.96        400
   macro avg      0.96        0.96        0.96        400
weighted avg      0.96        0.96        0.96        400
```

SVM scores with C=0.1 and Kernel=linear:

```
              precision    recall  f1-score   support

           0       0.96        1.00        0.98        192
           1       1.00        0.96        0.98        208

    accuracy                             0.98        400
   macro avg       0.98        0.98        0.98        400
weighted avg       0.98        0.98        0.98        400
```

SVM scores with C=0.001 and Kernel=poly:

```
              precision    recall  f1-score   support

           0       0.73        1.00        0.84        192
           1       1.00        0.66        0.79        208

    accuracy                             0.82        400
   macro avg       0.87        0.83        0.82        400
weighted avg       0.87        0.82        0.82        400
```

SVM scores with C=0.01 and Kernel=poly:

```
              precision    recall  f1-score   support

           0       0.86        0.99        0.92        192
           1       0.99        0.85        0.91        208

    accuracy                             0.92        400
   macro avg       0.93        0.92        0.92        400
weighted avg       0.93        0.92        0.92        400
```

SVM scores with C=0.1 and Kernel=poly:

```
              precision    recall  f1-score   support
```

```
              0     0.88        0.98        0.93        192
              1     0.98        0.88        0.93        208

       accuracy                            0.93        400
    macro avg      0.93        0.93        0.93        400
weighted avg       0.94        0.93        0.93        400
```

SVM scores with C=0.001 and Kernel=rbf:

```
              precision   recall  f1-score   support

              0     0.48        1.00        0.65        192
              1     0.00        0.00        0.00        208

       accuracy                            0.48        400
    macro avg      0.24        0.50        0.32        400
weighted avg       0.23        0.48        0.31        400
```

SVM scores with C=0.01 and Kernel=rbf:

```
              precision   recall  f1-score   support

              0     0.92        0.92        0.92        192
              1     0.93        0.93        0.93        208

       accuracy                            0.93        400
    macro avg      0.92        0.92        0.92        400
weighted avg       0.93        0.93        0.93        400
```

SVM scores with C=0.1 and Kernel=rbf:

```
              precision   recall  f1-score   support

              0     0.93        0.95        0.94        192
              1     0.96        0.94        0.95        208

       accuracy                            0.94        400
    macro avg      0.94        0.95        0.94        400
weighted avg       0.95        0.94        0.95        400
```

```
SVM scores with C=0.001 and Kernel=sigmoid:

              precision    recall  f1-score   support

           0       0.48      1.00      0.65       192
           1       0.00      0.00      0.00       208

    accuracy                           0.48       400
   macro avg       0.24      0.50      0.32       400
weighted avg       0.23      0.48      0.31       400


SVM scores with C=0.01 and Kernel=sigmoid:

              precision    recall  f1-score   support

           0       0.86      0.53      0.65       192
           1       0.68      0.92      0.78       208

    accuracy                           0.73       400
   macro avg       0.77      0.72      0.72       400
weighted avg       0.77      0.73      0.72       400


SVM scores with C=0.1 and Kernel=sigmoid:

              precision    recall  f1-score   support

           0       0.53      0.48      0.51       192
           1       0.56      0.61      0.59       208

    accuracy                           0.55       400
   macro avg       0.55      0.55      0.55       400
weighted avg       0.55      0.55      0.55       400
```

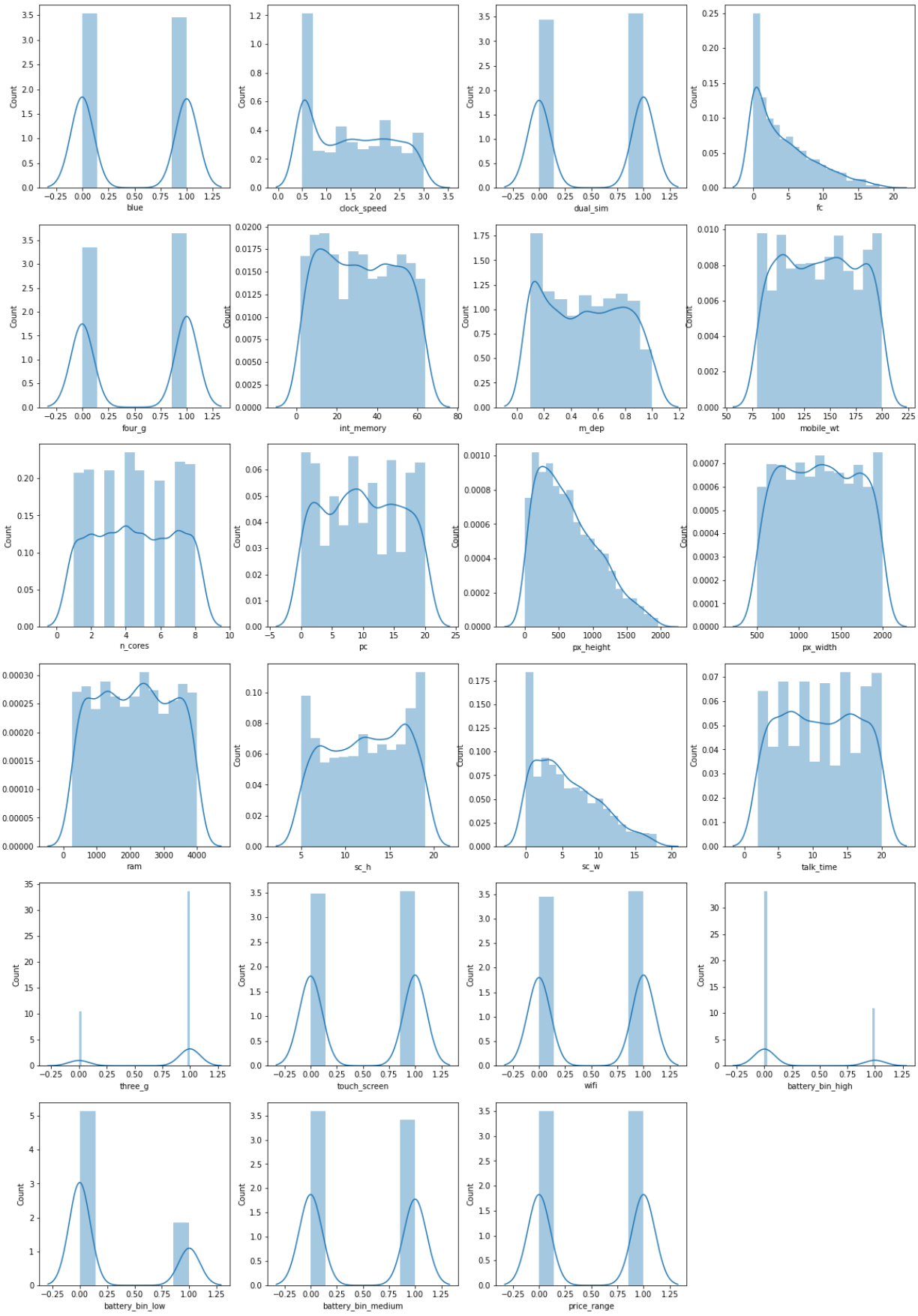
۹ (ج)

تبدیل لگاریتمی، در مواردی که دیتاست skew دارد کاربردی است و باعث می‌شود توزیع دادگان نرمال تر شود ابتدا بازه تغییرات ستون‌ها را در پایین چاپ می‌کنیم، و سپس توزیع احتمالی هر ستون را رسم می‌نماییم.

```
[ ] for col in df_bin.columns:
       print(f'{col} range: [{df_bin[col].min()}, {df_bin[col].max()}]')
```

```
blue range: [0, 1]
clock_speed range: [0.5, 3.0]
dual_sim range: [0, 1]
fc range: [0, 19]
four_g range: [0, 1]
int_memory range: [2, 64]
m_dep range: [0.1, 1.0]
mobile_wt range: [80, 200]
n_cores range: [1, 8]
pc range: [0, 20]
px_height range: [0, 1960]
px_width range: [500, 1998]
ram range: [256, 3998]
sc_h range: [5, 19]
sc_w range: [0, 18]
talk_time range: [2, 20]
three_g range: [0, 1]
touch_screen range: [0, 1]
wifi range: [0, 1]
battery_bin_high range: [0, 1]
battery_bin_low range: [0, 1]
battery_bin_medium range: [0, 1]
price_range range: [0, 1]
```

```
[ ] # plot the distribution of the columns
    import seaborn as sns
    i=1
    plt.figure(figsize=(20,30))
    for col in df_bin.columns:
       plt.subplot(6,4,i)
       sns.distplot(df_bin[col], kde=True)
       plt.xlabel(col)
       plt.ylabel('Count')
       i+=1
```


blue
clock_speed
dual_sim
fc
four_g
int_memory
m_dep
mobile_wt
n_cores
pc
px_height
px_width
ram
sc_h
sc_w
talk_time
three_g
touch_screen
wifi
battery_bin_high
battery_bin_low
battery_bin_medium
price_range
Count

مشاهده می‌شود که ستون‌های px_height ،fc و sc_w هر سه دارای skew می‌باشند. از آن جا که مقدار کمینه این ستون‌ها عدد صفر است اعمال تبدیل لگاریتمی در این مسئله امکان پذیر نمی‌باشد. با توجه به بازه‌های بدست آمده در تصویر قبل می‌توان دید مقادیر آن‌ها تفاوت بسیار زیادی از مرتبه 10^3 دارند و می‌توان برای رفع این مشکل از تبدیل استاندارد (max-min) استفاده کرد تا تمام دیتا در بازه صفر تا یک خلاصه شوند.

```
[ ]  # min-max normalization on columns

     df_norm = df_bin.copy()

     for col in df_norm:
         df_norm[col] = (df_bin[col] - df_bin[col].min()) / (df_bin[col].max() - df_bin[col].min())

     df_norm
```

	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	...	sc_h	sc_w	talk_time	three_g	touch_screen	wifi	battery_bin_high	battery_bin_low	battery_bin_medium	price_range
0	0.0	0.68	0.0	0.052632	0.0	0.080645	0.555556	0.900000	0.142857	0.10	...	0.285714	0.388889	0.944444	0.0	0.0	1.0	0.0	1.0	0.0	0.0
1	1.0	0.00	1.0	0.000000	1.0	0.822581	0.666667	0.466667	0.285714	0.30	...	0.857143	0.166667	0.277778	1.0	1.0	0.0	0.0	0.0	1.0	1.0
2	1.0	0.00	1.0	0.105263	1.0	0.629032	0.888889	0.541667	0.571429	0.30	...	0.428571	0.111111	0.388889	1.0	1.0	0.0	0.0	1.0	0.0	1.0
3	1.0	0.80	0.0	0.000000	0.0	0.129032	0.777778	0.425000	0.714286	0.45	...	0.785714	0.444444	0.500000	1.0	0.0	0.0	0.0	1.0	0.0	1.0
4	1.0	0.28	0.0	0.684211	1.0	0.677419	0.555556	0.508333	0.142857	0.70	...	0.214286	0.111111	0.722222	1.0	1.0	0.0	1.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1995	1.0	0.00	1.0	0.000000	1.0	0.000000	0.777778	0.216667	0.714286	0.70	...	0.571429	0.222222	0.944444	1.0	1.0	0.0	0.0	1.0	0.0	0.0
1996	1.0	0.84	1.0	0.000000	0.0	0.596774	0.111111	0.891667	0.428571	0.15	...	0.428571	0.555556	0.777778	1.0	1.0	1.0	1.0	0.0	0.0	1.0
1997	0.0	0.16	1.0	0.052632	1.0	0.548387	0.666667	0.233333	1.000000	0.15	...	0.285714	0.055556	0.166667	1.0	1.0	0.0	1.0	0.0	0.0	1.0
1998	0.0	0.16	0.0	0.210526	1.0	0.709677	0.000000	0.541667	0.571429	0.25	...	0.928571	0.555556	0.944444	1.0	1.0	1.0	0.0	0.0	1.0	0.0
1999	1.0	0.60	1.0	0.263158	1.0	0.693548	0.888889	0.733333	0.714286	0.80	...	1.000000	0.222222	0.000000	1.0	1.0	1.0	0.0	1.0	0.0	1.0

2000 rows × 23 columns

```
[ ]  df_norm.describe().T
```

	count	mean	std	min	25%	50%	75%	max
blue	2000.0	0.495000	0.500100	0.0	0.000000	0.000000	1.000000	1.0
clock_speed	2000.0	0.408900	0.326402	0.0	0.080000	0.400000	0.680000	1.0
dual_sim	2000.0	0.509500	0.500035	0.0	0.000000	1.000000	1.000000	1.0
fc	2000.0	0.226816	0.228497	0.0	0.052632	0.157895	0.368421	1.0
four_g	2000.0	0.521500	0.499662	0.0	0.000000	1.000000	1.000000	1.0
int_memory	2000.0	0.484621	0.292673	0.0	0.225806	0.483871	0.741935	1.0
m_dep	2000.0	0.446389	0.320462	0.0	0.111111	0.444444	0.777778	1.0
mobile_wt	2000.0	0.502075	0.294997	0.0	0.241667	0.508333	0.750000	1.0
n_cores	2000.0	0.502929	0.326834	0.0	0.285714	0.428571	0.857143	1.0
pc	2000.0	0.495825	0.303216	0.0	0.250000	0.500000	0.750000	1.0
px_height	2000.0	0.329137	0.226419	0.0	0.144260	0.287755	0.483291	1.0
px_width	2000.0	0.501679	0.288518	0.0	0.250167	0.498665	0.756342	1.0
ram	2000.0	0.499255	0.289880	0.0	0.254276	0.505211	0.750534	1.0
sc_h	2000.0	0.521893	0.300946	0.0	0.285714	0.500000	0.785714	1.0
sc_w	2000.0	0.320389	0.242022	0.0	0.111111	0.277778	0.500000	1.0
talk_time	2000.0	0.500611	0.303553	0.0	0.222222	0.500000	0.777778	1.0
three_g	2000.0	0.761500	0.426273	0.0	1.000000	1.000000	1.000000	1.0
touch_screen	2000.0	0.503000	0.500116	0.0	0.000000	1.000000	1.000000	1.0
wifi	2000.0	0.507000	0.500076	0.0	0.000000	1.000000	1.000000	1.0
battery_bin_high	2000.0	0.247500	0.431668	0.0	0.000000	0.000000	0.000000	1.0
battery_bin_low	2000.0	0.265500	0.441710	0.0	0.000000	0.000000	1.000000	1.0
battery_bin_medium	2000.0	0.487000	0.499956	0.0	0.000000	0.000000	1.000000	1.0
price_range	2000.0	0.500000	0.500125	0.0	0.000000	0.500000	1.000000	1.0

۹ (د) و ۱۰

ویژگی مساحت گوشی را با استفاده از طول و عرض آن بدست می‌آوریم. آن را به دیتا ست اضافه می‌کنیم و مدل‌های SVM را بر روی آن‌ها اعمال می‌کنیم

```
[ ]  # create an area feature

     df_bin['area'] = df_bin['px_height']*df_bin['px_width']

     price_column = df_bin.pop('price_range')

     df_bin.insert(23, 'price_range', price_column)

     df_bin.head()
```

	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	pc	...	sc_w	talk_time	three_g	touch_screen	wifi	battery_bin_high	battery_bin_low	battery_bin_medium	area	price_range
0	0	2.2	0	1	0	7	0.6	188	2	2	...	7	19	0	0	1	0	1	0	15120	0
1	1	0.5	1	0	1	53	0.7	136	3	6	...	3	7	1	1	0	0	0	1	1799140	1
2	1	0.5	1	2	1	41	0.9	145	5	6	...	2	9	1	1	0	0	1	0	2167308	1
3	1	2.5	0	0	0	10	0.8	131	6	9	...	8	11	1	0	0	0	1	0	2171776	1
4	1	1.2	0	13	1	44	0.6	141	2	14	...	2	15	1	1	0	1	0	0	1464096	0

5 rows × 24 columns

```
[ ]  X_train_bin, X_test_bin, y_train_bin, y_test_bin = \
     train_test_split(df_bin.iloc[:, :-1], df_bin['price_range'],
                      test_size=0.20, random_state=2022)

     for i in svc_kernel:
       for j in C_list:
         svm_analysis(j, i, X_train_bin, X_test_bin, y_train_bin, y_test_bin)
```

```
SVM scores with C=0.001 and Kernel=linear:

              precision    recall  f1-score   support

           0       0.88      0.98      0.93       192
           1       0.98      0.88      0.93       208

    accuracy                           0.93       400
   macro avg       0.93      0.93      0.93       400
weighted avg       0.94      0.93      0.93       400


SVM scores with C=0.01 and Kernel=linear:

              precision    recall  f1-score   support

           0       0.89      0.98      0.93       192
           1       0.98      0.88      0.93       208

    accuracy                           0.93       400
```

```
   macro avg      0.93        0.93        0.93        400
weighted avg      0.93        0.93        0.93        400
```

SVM scores with C=0.1 and Kernel=linear:

```
              precision   recall  f1-score   support

           0     0.89        0.98        0.94        192
           1     0.98        0.89        0.93        208

      accuracy                           0.94        400
   macro avg      0.94        0.94        0.93        400
weighted avg      0.94        0.94        0.93        400
```

SVM scores with C=0.001 and Kernel=poly:

```
              precision   recall  f1-score   support

           0     0.49        0.95        0.64        192
           1     0.62        0.07        0.13        208

      accuracy                           0.49        400
   macro avg      0.56        0.51        0.39        400
weighted avg      0.56        0.49        0.38        400
```

SVM scores with C=0.01 and Kernel=poly:

```
              precision   recall  f1-score   support

           0     0.49        0.95        0.65        192
           1     0.66        0.09        0.16        208

      accuracy                           0.50        400
   macro avg      0.57        0.52        0.40        400
weighted avg      0.58        0.50        0.39        400
```

SVM scores with C=0.1 and Kernel=poly:

```
              precision   recall  f1-score   support
```

```
              0     0.49        0.95        0.65        192
              1     0.66        0.09        0.16        208

       accuracy                              0.50        400
   macro avg      0.57        0.52        0.40        400
weighted avg      0.58        0.50        0.39        400
```

SVM scores with C=0.001 and Kernel=rbf:

```
              precision   recall  f1-score   support

              0     0.48        1.00        0.65        192
              1     0.00        0.00        0.00        208

       accuracy                              0.48        400
   macro avg      0.24        0.50        0.32        400
weighted avg      0.23        0.48        0.31        400
```

SVM scores with C=0.01 and Kernel=rbf:

```
              precision   recall  f1-score   support

              0     0.50        0.89        0.64        192
              1     0.63        0.18        0.28        208

       accuracy                              0.52        400
   macro avg      0.56        0.53        0.46        400
weighted avg      0.57        0.52        0.45        400
```

SVM scores with C=0.1 and Kernel=rbf:

```
              precision   recall  f1-score   support

              0     0.49        0.77        0.60        192
              1     0.56        0.27        0.37        208

       accuracy                              0.51        400
   macro avg      0.53        0.52        0.49        400
weighted avg      0.53        0.51        0.48        400
```

SVM scores with C=0.001 and Kernel=sigmoid:

```
              precision    recall  f1-score   support

           0       0.48      1.00      0.65       192
           1       0.00      0.00      0.00       208

    accuracy                           0.48       400
   macro avg       0.24      0.50      0.32       400
weighted avg       0.23      0.48      0.31       400


SVM scores with C=0.01 and Kernel=sigmoid:

              precision    recall  f1-score   support

           0       0.48      1.00      0.65       192
           1       0.00      0.00      0.00       208

    accuracy                           0.48       400
   macro avg       0.24      0.50      0.32       400
weighted avg       0.23      0.48      0.31       400


SVM scores with C=0.1 and Kernel=sigmoid:

              precision    recall  f1-score   support

           0       0.45      0.52      0.48       192
           1       0.49      0.42      0.45       208

    accuracy                           0.47       400
   macro avg       0.47      0.47      0.47       400
weighted avg       0.47      0.47      0.47       400
```

(۱۱

درخت های تصـــمیم گیری از قدرتمند ترین الگوریتم ها هســـتند که به عنوان زیر مجموعه ای از الگوریتم های تحت نظارت (supervised algorithms)در نظر گرفته می شـــوند. الگوریتم های مختلفی برای ســـاخت درخت تصـمیم وجود دارند. الگوریتم درخت تصـمیم به گونه ای عمل می کند که سـعی دارد گوناگونی و یا تنوع را در گره ها به حداقل ممکن برســـاند. این عدم یکنواختی در گره ها با اســـتفاده از معیارهای عدم خلوص قابل

اندازه گیری اســت که مهمترین و پرکاربرد ترین آن شــاخص جینی می باشــد .اغلب تفاوت انواع درخت های تصمیم در همین معیار اندازه گیری عدم خلوص، شیوه شاخه بندی و هرس کردن گره های درخت می باشد.

الگوریتمCART : برای برقراری درختهای رگرسیون و دسته بندی از این الگوریتم استفاده میشود. در سال ۱۹۸۴توســط Breiman و همکارانش ارائه شــده اســت. یکی از محبوب‌ترین و در عین حال ســاده‌ترین الگوریتم‌های درخت‌های تصمیم است که کاربردهای زیادی در طبقه بندی و رگرسیون دارد و بر اساس درخت های دودویی (باینری) بنا نهاده شده است. الگوریتم CART متغیرهای ورودی را برای یافتن بهترین تجزیه می آزماید تا شاخص ناخالصی حاصل از تجزیه کمترین مقدار باشد. در تجزیه دو زیر گروه تعیین می شود و هر کدام در مرحله بعد به دو زیر گروه دیگر تقسیم خواهند شد و این روند ادامه می یابد تا زمانی که یکی از معیار های توقف برآورده شــود. درخت CART بازگشــتی دو دویی اســت، که گره های والدین را دقیقا به دو گروه فرزند منشعب می کند و به طور بازگشتی منشعب کردن را تا زمانی که انشعاب دیگری نتواند ساخته شود ادامه می دهد.

الگوریتم ID3: یکی از الگوریتم‌های بسیار ساده درخت تصمیم که در سال ۱۹۸۶ توسط Quinlan مطرح شده است. اطلاعات به دست آمده به عنوان معیار تفکیک به کار می‌رود. این الگوریتم هیچ فرایند هرس کردن را به کار نمی‌برد و مقادیر اســمی و مفقوده را مورد توجه قرار نمی‌دهد. الگوریتم ID3 وظیفه پیدا کردنِ ویژگی‌هایی دارای اطلاعات زیادتر (Gain پیشــتر) را دارد و آن‌ها را در ســطوحِ 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. البته وقتی یک گره از درخت انتخاب شد، برای ساختِ زیر درخت‌های دیگر، مجموعه داده‌ها بر اساس مقدارِ گره‌ی انتخاب شده در شاخه‌های بالاتر، کوچکتر می‌شوند و هر چه در درخت پایین‌تر می‌رویم (به برگ‌ها نزدیک‌تر می‌شویم)، مجموعه داده‌ها برای محاسبه‌ی مقدار اطلاعات کمتر می‌شوند.

الگوریتم C4.5: این الگوریتم درخت تصــمیم، تکامل یافته ID3 اســت که در ســال ۱۹۹۳ توســط Quinlanمطرح شده است. ایشان بعد از اینکه به نقاط ضعفِ این الگوریتم پی‌برد، در مدتِ کوتاهی الگوریتمِ بعدی خود یعنی C4.5 را طراحی کرد. از نقاطِ ضعف الگوریتم ID3 که در C4.5 رفع شده است می‌توان به موارد زیر اشاره کرد:

۱ .الگوریتم C4.5 می‌تواند مقادیر گســسته یا پیوســته را در ویژگی‌ها درک کند. ولی الگوریتمِ ID3 اولیه نمی‌تواند تفاوتِ مقادیرِ عددیِ پیوسته را درک کند.

۲ .الگوریتمِ C4.5 قادر است تا مقادیری که موجود نیستند را هم تحمل کند.

۳. الگوریتم‌هایی مانند ID3 به خاطر اینکه سعی دارند تا حد امکان شاخه و برگ داشته باشند (تا به نتیجه مورد نظر برسند) با احتمال بالاتری دارای پیچیدگی در ساخت مدل می‌شوند. اما با عملیات هرس کردن درخت که در الگوریتم C4.5 انجام می‌شود، می‌توان مدل را به یک نقطه بهینه رساند که زیاد پیچیده نباشد.

۴. الگوریتم C4.5 این قابلیت را دارد که وزن‌های مختلف و غیر یکسانی را به برخی از ویژگی‌ها بدهد.

۱۲) و ۱۳)

مدل درخت تصمیم گیری را بر روی دیتاست با پارامترهای مختلف از جمله criterion و عمق متفاوت اعمال می‌کنیم. معیارهای دقت را برای آن‌ها چاپ می‌کنیم.

```
[ ]  # Decision Tree

     criterion_list = ['gini', 'entropy']

     max_depth_list = [2, 4, 6, 8, 10, 12]

     def dt_analysis(Criterion, Max_dept, X_train, X_test, y_train, y_test):
       dt = DecisionTreeClassifier(criterion=Criterion, max_depth=Max_dept)
       dt.fit(X_train, y_train)
       y_pred = dt.predict(X_test)
       print(f'DT scores with Criterion = {Criterion} and Max_dept = {Max_dept}:\n')
       print(classification_report(y_test, y_pred), '\n\n\n')
```

```
[ ]  for i in criterion_list:
       for j in max_depth_list:
         dt_analysis(i, j, X_train, X_test, y_train, y_test)
```

```
DT scores with Criterion = gini and Max_dept = 2:

              precision    recall  f1-score   support

           0       0.92      0.88      0.90       192
           1       0.89      0.93      0.91       208

    accuracy                           0.91       400
   macro avg       0.91      0.90      0.90       400
weighted avg       0.91      0.91      0.90       400


DT scores with Criterion = gini and Max_dept = 4:

              precision    recall  f1-score   support
```

```
              0      0.91   0.96   0.93         192
              1      0.96   0.91   0.94         208

       accuracy                           0.94         400
   macro avg        0.94   0.94   0.93         400
weighted avg        0.94   0.94   0.94         400
```

DT scores with Criterion = gini and Max_dept = 6:

```
              precision    recall  f1-score   support

              0      0.93   0.98   0.96         192
              1      0.98   0.93   0.96         208

       accuracy                           0.96         400
   macro avg        0.96   0.96   0.96         400
weighted avg        0.96   0.96   0.96         400
```

DT scores with Criterion = gini and Max_dept = 8:

```
              precision    recall  f1-score   support

              0      0.92   0.97   0.94         192
              1      0.97   0.92   0.95         208

       accuracy                           0.94         400
   macro avg        0.95   0.95   0.94         400
weighted avg        0.95   0.94   0.95         400
```

DT scores with Criterion = gini and Max_dept = 10:

```
              precision    recall  f1-score   support

              0      0.93   0.98   0.95         192
              1      0.98   0.93   0.96         208

       accuracy                           0.95         400
   macro avg        0.96   0.96   0.95         400
weighted avg        0.96   0.95   0.96         400
```

DT scores with Criterion = gini and Max_dept = 12:

```
              precision    recall  f1-score   support

              0      0.94   0.97   0.96         192
```

```
              1      0.98   0.94   0.96         208

       accuracy                           0.96         400
   macro avg        0.96   0.96   0.96         400
weighted avg        0.96   0.96   0.96         400



DT scores with Criterion = entropy and Max_dept = 2:

              precision    recall  f1-score   support

              0      0.92   0.88   0.90         192
              1      0.89   0.93   0.91         208

       accuracy                           0.91         400
   macro avg        0.91   0.90   0.90         400
weighted avg        0.91   0.91   0.90         400



DT scores with Criterion = entropy and Max_dept = 4:

              precision    recall  f1-score   support

              0      0.92   0.96   0.94         192
              1      0.96   0.92   0.94         208

       accuracy                           0.94         400
   macro avg        0.94   0.94   0.94         400
weighted avg        0.94   0.94   0.94         400



DT scores with Criterion = entropy and Max_dept = 6:

              precision    recall  f1-score   support

              0      0.95   0.98   0.97         192
              1      0.99   0.95   0.97         208

       accuracy                           0.97         400
   macro avg        0.97   0.97   0.97         400
weighted avg        0.97   0.97   0.97         400



DT scores with Criterion = entropy and Max_dept = 8:

              precision    recall  f1-score   support

              0      0.95   0.96   0.96         192
              1      0.96   0.96   0.96         208
```

```
       accuracy                           0.96         400
   macro avg        0.96   0.96   0.96          400
weighted avg        0.96   0.96   0.96          400


DT scores with Criterion = entropy and Max_dept = 10:

              precision    recall  f1-score   support

           0       0.95   0.95   0.95          192
           1       0.96   0.96   0.96          208

       accuracy                           0.95         400
   macro avg        0.95   0.95   0.95          400
weighted avg        0.95   0.95   0.95          400


DT scores with Criterion = entropy and Max_dept = 12:

              precision    recall  f1-score   support

           0       0.96   0.96   0.96          192
           1       0.96   0.96   0.96          208

       accuracy                           0.96         400
   macro avg        0.96   0.96   0.96          400
weighted avg        0.96   0.96   0.96          400
```

مشاهد می‌شود که عمق درخت و تعداد نمونه‌های موجود در هر گره تاثیر کمی بر دقت مدل دارند. به طوری که این تاثیر با افزایش عمق ابتدا موجب افزایش دقت می‌شود.

(۱۴

الگوریتم درخت تصمیم رشد درخت را بر اساس یک معیار توقف، متوقف میکند. ساده ترین معیار توقف، معیاری است که در آن همه نمونه های آموزشی در برگ متعلق به یک کلاس هستند. یک مشکل این است که ساخت درخت تصمیم تا این سطح ممکن است منجر به بیش برازش شود. چنین درختی به خوبی به نمونه های آزمایشی دیده نشده تعمیم نمی یابد. برای جلوگیری از کاهش دقتِ ناشی از بیش برازش، دسته بند از مکانیزم هرس استفاده میشود. درخت‌های هرس شده تمایل به کوچک‌تر بودن و پیچیدگی کم‌تر دارند و بنابراین به

راحتی قابل فهم می‌باشند. آن‌ها معمولا در طبقه‌بندی صحیح داده‌های تست سریع‌تر و بهتر از درخت‌های هرس نشده عمل می‌کنند. هرس کردن به دو روش pre-pruning و post-pruning صورت می‌گیرد.

پیش‌هرس (Pre-pruning) : در این شیوه، هرس کردن قبل از ساخت کامل درخت می‌باشد. به این صورت که به درختی که در حال رشد است اجازه رشد بیش از حد داده نشود. مثلا به گره تصمیمی می‌رسیم که در آن‌جا ۴ آیتم مثبت و ۱ آیتم منفی داریم و با توجه به آن‌که تمامی شروط توقف رعایت نشده است و می‌توان هم‌چنان از این گره، رشد درخت را ادامه داد ولی در این مرحله این گره تصمیم را به برگ تبدیل می‌نماییم و مقدار آن را براساس آیتم با مقادیر بیشتر که در مثال بالا آیتم‌های مثبت است برچسب‌گزاری می‌کنیم.

هرس بعد (Post-pruning): در این شیوه هرس کردن، درخت به صورت کامل ساخته می‌شود و سپس عملیات هرس کردن درخت آغاز می‌شود. به این‌صورت که از پایین درخت یا همان برگ‌ها به سمت ریشه حرکت می‌کنیم و یک‌سری گره‌های میانی را تبدیل به برگ می‌کنیم. این روش هرس کردن از شیوه‌ی قبل کمی کندتر است ولی دارای دقت بیشتری می‌باشد.

(۱۵

بوت استرپینگ در واقع تخمین ویژگی های (مثل واریانس)یک تخمین زننده است با استفاده از اندازه گیری همین ویژگیها در یک توزیع تقریبی از کل داده های نمونه. بوت استرپینگ این امکان را برای یک نفر فراهم می سازد که تعداد زیادی نسخه ی جایگزین از یک آماره را که به طور معمول از یک نمونه محاسبه میشود را جمع آوری کند. روش bootstrap روش کارآمدی برای محاسبه میزان دقت و خطای استاندارد متغیر تخمین زده شده است. به عنوان مثال، فرض کنید که ما علاقه مند به جمع آوری اطلاعات در مورد قد افراد در جهان هستیم. به دلیل اینکه نمیتوانیم کل جمعیت را اندازه گیری کنیم، تنها یک از قسمت کوچک نمونه برداری می کنیم. از این نمونه فقط یک آماره قابل محاسبه است، مثلا یک میانگین یا یک انحراف معیار. در نتیجه نمیتوانیم متوجه شویم که آماره ها چه قدر و در چه بازه ای تغییر میکنند. اما هنگامی که از بوت استرپ استفاده کنیم ما به صورت تصادفی یک نمونه ی n تایی از N تا داده ی نمونه بر می داریم، به طوریکه هر نفر حد اکثر t بار میتواند انتخاب شود. با چندین بار انجام این کار در واقع تعداد زیادی مجموعه ی داده میسازیم که برای هرکدام میتوانیم یک آماره حساب کنیم.

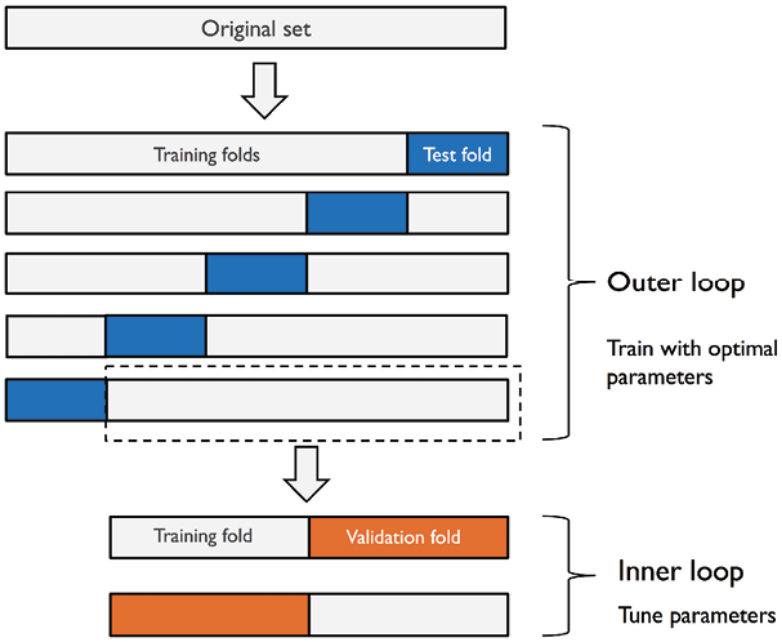
هر دو روش bootstrap و cross validation روشهای نمونه گیری مجدد هستند ولی همانظور که ذکر شد روش bootstrap برای ارزیابی میزان دقت تخمین یک پارامتر، کاربرد دارد ولی روش cross-validation، روشی برای ارزیابی عملکرد مدل و بدست آوردن تخمینی از خطای مدل است.

(۱۶

اگر زمانی که داریم مدل را بر اساس یک سری از هایپرپارامترها تنظیم میکنیم، بخواهیم همزمان مدلمان را ارزیابی کنیم، باید 5x2 Cross Validation انجام دهیم. در حالت عادی وقتی این کا را انجام ندهیم ، و از همان دیتایی که برای انتخاب بهترین هایپرپارامترها استفاده کردیم، برای test کردن مدل استفاده کنیم، چون مدل در حین tuning ، داده‌ها رو تجربه کرده، test-score واقعی را نشان نخواهد داد(بهتر از چیزی که در واقع است) و داده‌ها دچار overfit میشوند، و باید زمانی که عملکرد چندین مدل را مقایسه میکنیم به آن توجه کنیم.

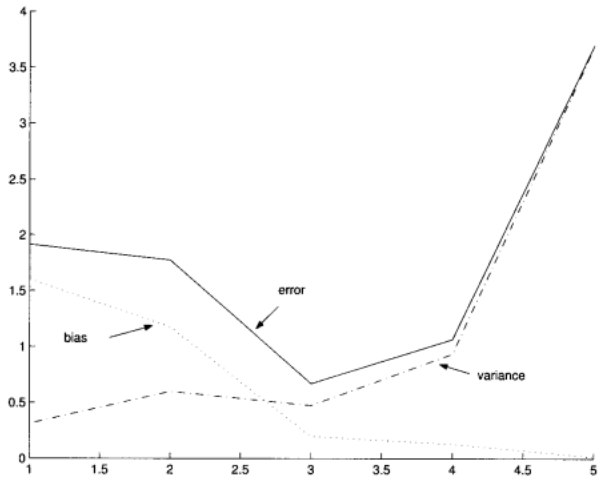
وقتی که از 5x2 CV استفاده می کنیم، از یک inner-cv برای انتخاب بهترین پارامتر، و از یک outer-cv برای تست کردن مدل استفاده میکنیم. بعنوان مثال (درصدها هم فرضی هستن):

اول دیتای مورد نظر را(در هر split از outer-cv) به دو قسمت ۷۰-۳۰ درصدی تقسیم میکنیم،بخش ۳۰ درصدی(test set) رو برای تست نهایی(روی مدلی که tune شده) کنار می گذاریم، و بعد آن ۷۰ درصد را به دو قسمت ۸۰-۲۰ تقسیم میکنیم. سپس هایپرپارامترهای مدل را (در هر split از Inner-Cv) روی این ۸۰ درصد train می کنیم(train set) و روی بخش ۲۰ درصدی (validation set)، test می کنیم تا بهترین پارامتر را با توجه به عملکردشان انتخاب کنیم. بعد از اینکه که مدل با بهترین پارامترا را انتخاب کردیم، این مدل را روی ۳۰ درصدی که در ابتدا کار کنار گذاشته بودیم test میکنیم، و این بار عملکرد واقعی‌تر و منطقی‌تری را از مدل انتظار داریم.



(۱۷

از روشelbow یا آرنج برای تعیین تعداد صحیح خوشه‌ها در یک دیتاست استفاده می‌شود. در این روش مقادیر افزایشـی k بر روی محور افقی و مجموع خطاهایی که در هنگام اسـتفاده از k میانگین رخ داده بر روی محور عمودی ترسیم می‌شود. هدف از استفاده از این روش یافتنk ای است که برای هر خوشه واریانس را زیاد افزایش ندهد. روش آرنج درصـد واریانس را به عنوان تابعی از تعداد خوشـه‌ها توضـیح می‌دهد: یکی باید به عنوان تعداد خوشـه‌ها انتخاب شـود به طوری که با اضـافه کردن خوشـه ای دیگر مدل‌سـازی داده بهتری بدسـت نیاید. اگر یک ترسیم (plot) درصد واریانس را تشریح کند طوریکه مخالف تعداد خوشه‌ها باشد اولین خوشه‌ها اطلاعات زیادی (توضیح بسیاری از واریانس) را اضافه می‌کنند، اما در بعضی نقطه‌ها حاشیه سود کاهش خواهد یافت و یک زاویه در نمودار به وجود می‌آورد. تعداد خوشـــه‌ها در این نقطه انتخاب شـــده‌اند یعنی همان معیار آرنج . این آرنج نمی‌تواند همیشه به روشنی مشخص شود.



با توجه به توضـیحات داده شـده میتوان با توجه به trade-off بایاس و واریانس میزان مرتبه مناسـب مدل را پیدا کرد. مثلا در شـــکل بالا درجه حدود ۳ بهترین مرتبه مدل می توان در نظر گرفت. از طرفی از آنجایی که روش elbow یک روش هیوریستیک است ممکن است در همه مسایل در عمل نتوان به خوبی این کار را انجام داد.

دیتاست شماره ۲

۱)

قضیه بیز بیان می‌کند که اگر A و B دو واقعه‌ی مستقل باشند، احتمال رخ دادن واقعه‌ی A اگر واقعه‌ی B رخ داده باشد، برابر است با حاصل ضرب احتمال رخ دادن واقعه‌ی B در صورت رخ دادن واقعه‌ی A، در احتمال رخ دادن واقعه‌ی A تقسیم بر احتمال رخ دادن واقعه‌ی B. به عبارتی دیگر قضیه بیز، احتمال وقوع یک رویداد را بر اساس دانش قبلی از شرایطی که ممکن است مربوط به رویداد باشد، توصیف می کند.

Gaussian Naive Bayes

برای محاسبه‌ی تابع احتمال، مانند Gaussian Bayes از توابع گوسی استفاده می‌کند با این تفاوت که در این مدل از اثر ویژگی‌ها بر یکدیگر صرف نظر شده یعنی عناصر غیر قطری ماتریس کوواریانس توابع احتمال گوسی برابر با ۰ فرض می‌شوند. استفاده در مسائل دارای مقادیر پیوسته که می‌توان توسط یک توزیع احتمال گوسی آنها را مدل کرد.

Multinomial Naive Bayes

برای محاسبه‌ی تابع احتمال، از توابع multinomial استفاده می‌کند. استفاده در مسائلی که مقادیر ویژگی‌ها نشان دهنده‌ی تعداد تکرار یا به عبارتی فرکانس می‌باشد.

Bernoulli Naive Bayes

برای محاسبه‌ی تابع احتمال، از توابع bernoulli استفاده می‌کند. استفاده در مسائلی که مقادیر ویژگی می‌تواند ۰ یا ۱ باشد.

۲)

ابتدا کتابخانه‌های مورد نیاز بارگذاری میشوند:

```
# import the required libraries

from scipy.stats import norm
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings
warnings.filterwarnings('ignore')
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.naive_bayes import GaussianNB
```

دیتاست مستقیما از لینک زیر در محیط colab بارگذاری شده و توسط ماژول pandas خوانده میشود:

```
# Load the dataset

!wget https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.cleveland.data
df = pd.read_csv('processed.cleveland.data', header=None)
df
```

```
--2022-05-16 15:17:00--  https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/process
ed.cleveland.data
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18461 (18K) [application/x-httpd-php]
Saving to: 'processed.cleveland.data.1'

processed.cleveland 100%[===================>]  18.03K  --.-KB/s    in 0.05s

2022-05-16 15:17:00 (352 KB/s) - 'processed.cleveland.data.1' saved [18461/18461]
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	2.0	0.0	7.0	1
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	2.0	2.0	7.0	2
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	2.0	1.0	7.0	3
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	2.0	1.0	3.0	1
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0	1.0	?	3.0	0

303 rows × 14 columns

نام ستون‌ها با نام های مشخص شده جایگزین میشوند:

```
# rename the columns

names = ['age', 'sex', 'cp', 'restbp', 'chol', 'fbs',
         'restecg', 'thalach', 'exang', 'oldpeak',
         'slope', 'ca', 'thal', 'target']

df.columns = names
df.head()
```

	age	sex	cp	restbp	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0

نوع دادگان هر ستون چاپ میشود و می‌بینیم در ستون‌های ca و thal مقادیر غیر عددی داریم. برای رفع این مشکل مقادیری که با ؟ مشخص شده‌اند را از دیتاست حذف کرده و نوع دادگان دو ستون ذکر شده را به float تغییر می‌دهیم.

```
# print data types

df.dtypes
```

```
age        float64
sex        float64
cp         float64
restbp     float64
chol       float64
fbs        float64
restecg    float64
thalach    float64
exang      float64
oldpeak    float64
slope      float64
ca          object
thal        object
target       int64
dtype: object
```

```
# get the unique values of 'ca'

df['ca'].unique()
```

```
array(['0.0', '3.0', '2.0', '1.0', '?'], dtype=object)
```

```
# get the unique values of 'thal

df['thal'].unique()
```

```
array(['6.0', '3.0', '7.0', '?'], dtype=object)
```

```
# check for missing values

for col in df.columns:
    missing = df[col].isnull().sum()
    print(f'{missing} missing values in {col}')
```

```
0 missing values in age
0 missing values in sex
0 missing values in cp
0 missing values in restbp
0 missing values in chol
0 missing values in fbs
0 missing values in restecg
0 missing values in thalach
0 missing values in exang
0 missing values in oldpeak
0 missing values in slope
4 missing values in ca
2 missing values in thal
0 missing values in target
```

```
# remove missing (?) values

df.dropna(axis=0, inplace=True)
df.reset_index(drop=True, inplace=True)
df
```

	age	sex	cp	restbp	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	0
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	2
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	1
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	0
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
292	57.0	0.0	4.0	140.0	241.0	0.0	0.0	123.0	1.0	0.2	2.0	0.0	7.0	1
293	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	2.0	0.0	7.0	1
294	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	2.0	2.0	7.0	2
295	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	2.0	1.0	7.0	3
296	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	2.0	1.0	3.0	1

297 rows × 14 columns

```
# convert 'ca' and 'thal' from object to float

df['ca'] = df['ca'].astype(float)
df['thal'] = df['thal'].astype(float)
```

```
# print data types

df.dtypes
```

```
age        float64
sex        float64
cp         float64
restbp     float64
chol       float64
fbs        float64
restecg    float64
thalach    float64
exang      float64
oldpeak    float64
slope      float64
ca         float64
thal       float64
target       int64
dtype: object
```

کلاس زیر برای مدل Naive Bayes classifier نوشته می‌شود:

```
# implement Naive Bayes

class NaiveBayesClassifier():

    mu = None
    sigma = None
    n_classes = None

    def __init__(self, priors):
        a = None
        self.priors=priors

    def pred(self, x):
        prob_vect = np.zeros(self.n_classes)

        for i in range(self.n_classes):
            prob_vect[i] = self.priors[i]
            for j in range(len(self.mu[i])):
                normal = norm(self.mu[i, j], self.sigma[i, j])
                prob_vect[i] *= normal.pdf(x[j])

        prob_vect = [p/(sum(prob_vect)) for p in prob_vect]

        return prob_vect

    def fit(self, X, y):
        self.n_classes = np.max(y) + 1
        self.mu = [[] for _ in range(self.n_classes)]
        self.sigma = [[] for _ in range(self.n_classes)]

        for i in range(self.n_classes):
            Xc = X[y==i]
            for j in range(Xc.shape[1]):
                mu_c_f = np.mean(Xc[:, j])
                self.mu[i].append(mu_c_f)
                sigma_c_f = np.std(Xc[:, j])
                self.sigma[i].append(sigma_c_f)

        self.mu = np.asarray(self.mu)
        self.sigma = np.asarray(self.sigma)
```

۳)

ابتدا ماتریس ویژگی‌های ذکر شده در صورت سوال و بردار هدف تشکیل می‌شوند و سپس به صورت ۸۰-۲۰ بین train و test تقسیم میشوند:

```
# create the features data (X) and the target (y)

X = df[['chol', 'restbp', 'thalach']]
y = df['target']
```

```
# split the dataset 80-20

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=2022)
```

با فرض توزیع پیشینه‌ی یکنواخت (ML) بین کلاس‌ها، مدل نوشته شده train و test میشود:

```
# train and test using the custom model

priors_ML = [1/y.nunique() for _ in range(y.nunique())]
print(f'Max Likelihood Priors: {priors_ML}\n')

nb_model_custom = NaiveBayesClassifier(priors=priors_ML)
nb_model_custom.fit(np.array(X_train), y_train)

y_pred = []
for i in range(X_test.shape[0]):
    y_pr_i = nb_model_custom.pred(np.array(X_test)[i])
    y_pred.append(np.argmax(y_pr_i))

print(f'Testing Accuracy: {round(100*accuracy_score(y_test, y_pred), 2)}%\n')
print(f'{classification_report(y_test, y_pred)}')
```

```
Max Likelihood Priors: [0.2, 0.2, 0.2, 0.2, 0.2]

Testing Accuracy: 43.33%

              precision    recall  f1-score   support

           0       0.78      0.56      0.65        32
           1       0.40      0.29      0.33        14
           2       0.17      0.40      0.24         5
           3       0.11      0.17      0.13         6
           4       0.17      0.33      0.22         3

    accuracy                           0.43        60
   macro avg       0.33      0.35      0.32        60
```

```
weighted avg       0.54      0.43      0.47        60
```

۴)

مانند قسمت قبل، اینبار مدل موجود در کتابخانه‌ی sklearn استفاده میشود:

```python
# train and test using the sklearn model

nb_model_sklearn = GaussianNB()
nb_model_sklearn.fit(X_train, y_train)
y_pred = nb_model_sklearn.predict(X_test)

print(f'Testing Accuracy: {round(100*accuracy_score(y_test, y_pred), 2)}%\n')
print(f'{classification_report(y_test, y_pred)}')
```

```
Testing Accuracy: 53.33%

              precision    recall  f1-score   support

           0       0.62      0.94      0.75        32
           1       0.00      0.00      0.00        14
           2       0.33      0.20      0.25         5
           3       0.11      0.17      0.13         6
           4       0.00      0.00      0.00         3

    accuracy                           0.53        60
   macro avg       0.21      0.26      0.23        60
weighted avg       0.37      0.53      0.43        60
```

۵)

با توجه به نتایج بدست آمده در دو قسمت قبل، دقت الگوریتم نوشته شده و الگوریتم آماده در کتابخانه‌ی sklearn با هم تفاوت کمی دارند که ناشی از بهینه‌تر و دقیق‌تر بودن مدل موجود در کتابخلنه‌ی sklearn می‌باشد.