به نام هستی بخش

گزارش تمرین دوم

درس شناسایی الگو

نویسنده: حمیدرضا ابوئی

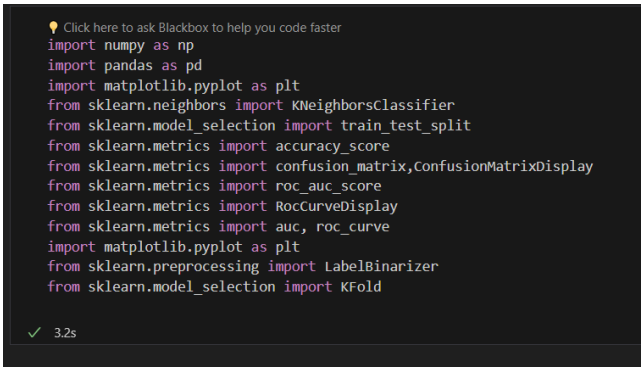
شماره دانشجویی: ۴۰۲۶۱۷۵۰۹

استاد: دکتر دلیری

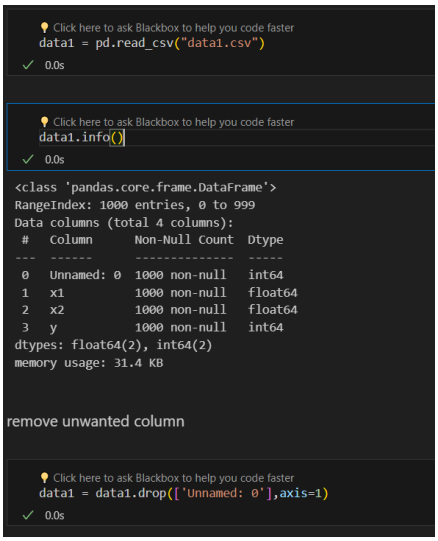
تاریخ: ۱۹ دی ۱۴۰۲

سوال اول:

ابتدا کتابخانه‌های مورد نیاز را وارد می‌کنیم.

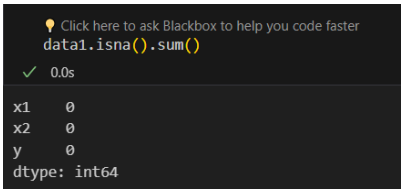


در ادامه داده‌ها را وارد می‌کنیم. در این سوال data1 به عنوان داده‌ی مساله داده شده است.



با بررسی داده ها متوجه می‌شویم که ستون اول فقط شماره‌گذاری داده هاست و آن را حذف می‌کنیم.

سپس داده‌ها را برای احتمال عدم وجود داده (missing value) بررسی می‌کنیم.



در ادامه به بررسی و حذف outlier می‌پردازیم. در این دادگان، ۸ داده‌ی پرت پیدا و حذف شدند.

remove outlier

```
Click here to ask Blackbox to help you code faster
numeric_col = data1.keys()

for x in numeric_col:
    q75,q25 = np.percentile(data1.loc[:,x],[75,25])  # a box plot of the quartile range and min/max values method
    IQR = q75-q25
    max_data = q75+(1.5*IQR)
    min_data = q25-(1.5*IQR)

    data1.loc[data1[x] < min_data,x] = np.nan   #filling the outliers values with 'nan'
    data1.loc[data1[x] > max_data,x] = np.nan   #filling the outliers values with 'nan'


data1.isna().sum()
```

[6] ✓ 0.0s

```
x1    8
x2    0
y     0
dtype: int64
```

```
Click here to ask Blackbox to help you code faster
data1 = data1.dropna()
data1.isna().sum()
```

✓ 0.0s

```
x1    0
x2    0
y     0
dtype: int64
```

```
Click here to ask Blackbox to help you code faster
data1.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 992 entries, 0 to 999
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   x1      992 non-null    float64
 1   x2      992 non-null    float64
 2   y       992 non-null    float64
dtypes: float64(3)
memory usage: 31.0 KB
```

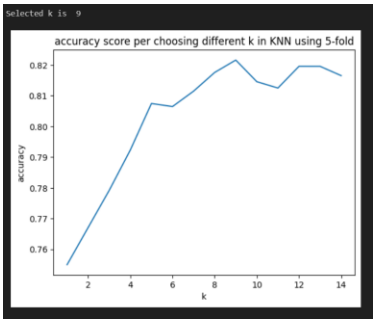
پس از حذف مشاهده می‌شود که از ۱۰۰۰ داده‌ی اولیه، ۹۹۲ داده در دسترس است. در ادامه، تارگت و ویژگی‌ها جدا می‌شوند.

```
Click here to ask Blackbox to help you code faster
X = data1.drop(['y'],axis = 1)
Y = data1['y']
```

[10] ✓ 0.0s

در ادامه قصد داریم که مقدار k برتر را برای الگوریتم KNN به دست آوریم. بدین منظور قصد داریم از الگوریتم K-fold استفاده کنیم تا نتیجه‌ای پایدارتر به دست آوریم. در الگوریتم‌های معمولی بهتر است از k-fold به همراه shuffle استفاده شود تا بایاس خاصی در انتخاب گروه‌ها وجود نداشته باشد اما با توجه به اینکه داده‌ها از ابتدا shuffle شده بودند، از k-fold بدون shuffle استفاده کردیم و داده‌ها را به ۵ قسمت مساوی تقسیم کرده و هر سری یکی از قسمت‌ها را به عنوان test و باقی را به عنوان train استفاده کردیم. سپس الگوریتم KNN را به ازای مقادیر مختلف K بین ۱ تا ۱۵ اعمال کردیم و بیشترین دقت را به عنوان K برتر یعنی ۹ در انتها به دست آوردیم.

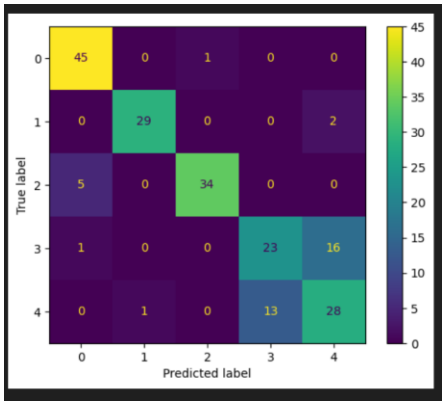
```
Click here to ask Blackbox to help you code faster
n_splits = 5
max_k = 15
accuracy_matrix = np.zeros((n_splits, max_k-1))

kf = KFold(n_splits=n_splits)
for i, (train_index, test_index) in enumerate(kf.split(X)):
    X_train, X_test, Y_train, Y_test = X.iloc[train_index],X.iloc[test_index],Y.iloc[train_index],Y.iloc[test_index]

    for k in range(1,max_k):
        # initialize K nearest neighborhood
        neigh = KNeighborsClassifier(n_neighbors=k)
        neigh.fit(X_train,Y_train)
        y_pred = neigh.predict(X_test)
        acc = accuracy_score(Y_test,y_pred)
        accuracy_matrix[i,k-1] = acc
        # print( f'k={k}',"\tis ",acc)
acc_final = np.mean(accuracy_matrix,axis=0)

selected_k = np.argmax(acc_final)+1
print("Selected k is ",selected_k)
plt.figure()
plt.plot(range(1,15),acc_final)
plt.ylabel("accuracy")
plt.xlabel("k")
plt.title(f"accuracy score per choosing different k in KNN using {n_splits}-fold")
plt.show()
✓ 1.8s
Selected k is  9
```



برای رسم ماتریس ابهام، یکی از fold های الگوریتم k-fold را انتخاب کردیم و الگوریتم KNN را به ازای برترین K یعنی ۹ محاسبه کردیم و میخواهیم مطابق زیر ماتریس ابهام را رسم کنیم:

Confusion Matrix

```
Click here to ask Blackbox to help you code faster
neigh = KNeighborsClassifier(n_neighbors=selected_k)
# Using the data of one of the kfolds (last one) to generate and show confusion matrix
neigh.fit(X_train,Y_train)
y_pred = neigh.predict(X_test)
acc = accuracy_score(Y_test,y_pred)
y_score = neigh.predict_proba(X_test)
[14] ✓ 0.0s

Click here to ask Blackbox to help you code faster
print(y_score.shape)
[15] ✓ 0.0s
... (198, 5)

Click here to ask Blackbox to help you code faster
confmat = confusion_matrix(Y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=confmat)
disp.plot()
[16] ✓ 0.7s
```



مشاهده می‌شود که الگوریتم KNN در تشخیص عمده کلاس‌ها به خوبی عمل کرده است ولی بین دو کلاس ۳ و۴ نتوانسته به خوبی باقی کلاس‌ها تفاوت قائل شود.

در ادامه برای رسم و بررسی منحنی ROC و مساحت زیر نمودار آن (AUC) باید چند کلاس را به دو کلاس تبدیل کنیم. زیرا این الگوریتم‌ها به خودی خود فقط در حالت باینری تعریف شده اند.

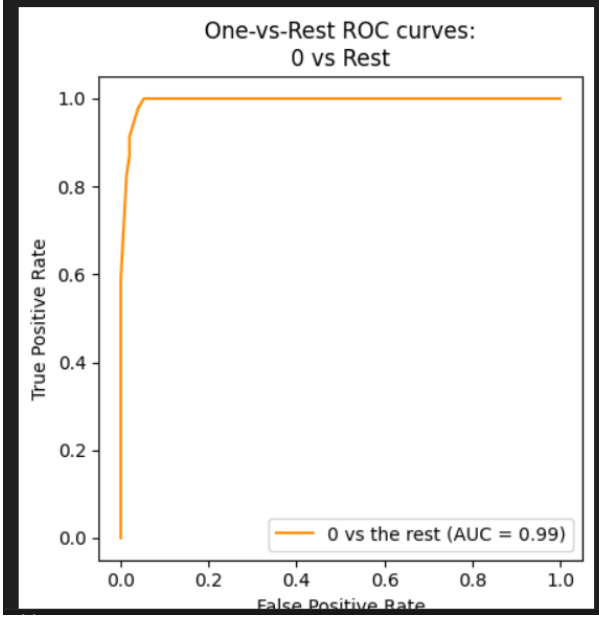
می‌توان به دو صورت تقسیم بندی کرد. اولین حالت One-VS-All و دیگری One-VS-One است. در اولی هر کلاس مورد نظر در یک کلاس و باقی کلاس‌ها همه در یک کلاس قرار می‌گیرند. در حالت دوم هر تک کلاس با هر تک کلاس به صورت مجزا بررسی می‌شوند که محاسبات بیشتری دارند. در این گزارش از روش اول استفاده شده است. ابتدا داده‌ها به صورت یک رشته اعداد باینری تقسیم می‌شود.

```
Click here to ask Blackbox to help you code faster

label_binarizer = LabelBinarizer().fit(Y_train)
y_onehot_test = label_binarizer.transform(Y_test)
y_onehot_test.shape  # (n_samples, n_classes)
✓ 0.0s

(198, 5)

Click here to ask Blackbox to help you code faster
label_binarizer.transform([2,3])
✓ 0.0s

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

Click here to ask Blackbox to help you code faster
class_of_interest = 0
class_id = np.flatnonzero(label_binarizer.classes_ == class_of_interest)[0]
class_id
✓ 0.0s

0
```

در ابتدا یکی از کلاس‌ها را به عنوان کلاس مورد نظر انتخاب می‌کنیم و منحنی ROC آن را در مقابل باقی کلاس‌ها رسم می‌کنیم

```
Click here to ask Blackbox to help you code faster
RocCurveDisplay.from_predictions(
    y_onehot_test[:, class_id],
    y_score[:, class_id],
    name=f"{class_of_interest} vs the rest",
    color="darkorange")
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("One-vs-Rest ROC curves:\n0 vs Rest")
plt.legend()
plt.show()
```

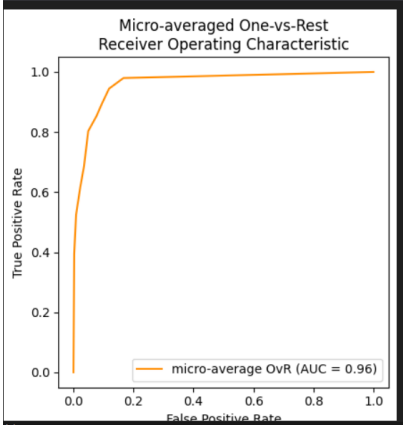
✓ 0.3s



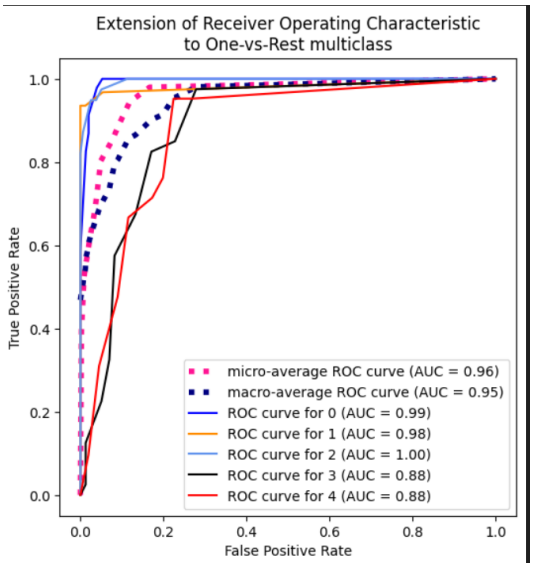
سپس دو حالت Micro averaging و Macro Avaraging را نیز تعریف و رسم می‌کنیم.

```
RocCurveDisplay.from_predictions(
    y_onehot_test.ravel(),
    y_score.ravel(),
    name="micro-average OvR",
    color="darkorange"
)
plt.axis("square")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Micro-averaged One-vs-Rest\nReceiver Operating Characteristic")
plt.legend()
plt.show()
```

✓ 0.3s



در ادامه باقی محاسبات مربوط به تک تک موارد در کد ذکر شده ولی در اینجا پاسخ و نتیجه‌ی ROC و AUC همه موارد را مشاهده می‌کنیم:



سوال دوم

پس از اعمال مراحل پیش پردازش مشابه سوال قبل، در اینجا قصد داریم با استفاده از الگوریتم PCA تعداد کامپوننت‌ها را کاهش دهیم. ابتدا پس از استفاده از K-fold با k=5، داده‌ها را به ۵ قسمت تقسیم کرده و این الگوریتم‌ها را ۵ بار اعمال می‌کنیم تا نتایج پایدار تر شوند. ابتدا پس از نرمالیزیشن داده‌ها با استفاده از میانگین و انحراف معیار داده‌های train، PCA را بر روی داده‌های خود اعمال می‌کنیم و مطابق صورت مساله ۲۰ و ۱۰ کامپوننت برتر را به ما بازگرداند. سپس الگوریتم SVM را بر روی آن اعمال می‌کنیم و دقت الگوریتم را برای هر دو حالت محاسبه کنیم.

```python
acc10 = []
acc20 = []
n_splits = 5
kf = KFold(n_splits=n_splits)
for i, (train_index, test_index) in enumerate(kf.split(X)):

    X_train, X_test, y_train, y_test = X.iloc[train_index],X.iloc[test_index],y.iloc[train_index],y.iloc[test_index]

    #Normalization
    X_train_mean = X_train.mean()
    X_train_std = X_train.std()
    X_train = (X_train - X_train_mean) / X_train_std
    X_test = (X_test - X_train_mean) / X_train_std

    pca1 = PCA(n_components=20)
    pca1.fit(X_train)

    pca2 = PCA(n_components=10)
    pca2.fit(X_train)

    pca1_train = pca1.transform(X_train)
    pca1_test = pca1.transform(X_test)

    pca2_train = pca2.transform(X_train)
    pca2_test = pca2.transform(X_test)

    clf = SVC()
    clf.fit(pca1_train,y_train)
    y_pred_svm1 = clf.predict(pca1_test)
    acc1 = accuracy_score(y_pred_svm1,y_test)
    acc10.append(acc1)

    clf = SVC()
    clf.fit(pca2_train,y_train)
    y_pred_svm2 = clf.predict(pca2_test)
    acc2 = accuracy_score(y_pred_svm2,y_test)
    acc20.append(acc2)
```

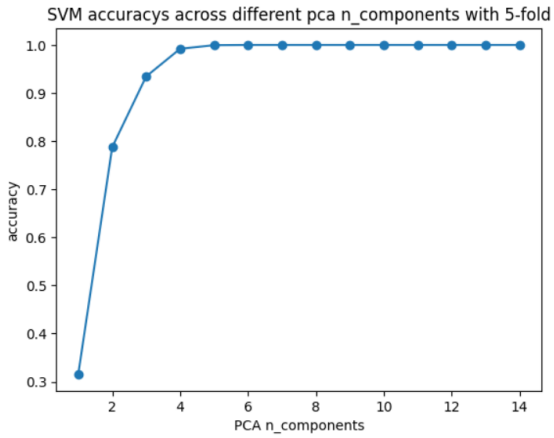
سپس نتایج را بین foldهای مختلف میانگین گیری و نمایش می‌دهیم.

```python
        acc2 = accuracy_score(y_pred_svm2,y_test)
        acc20.append(acc2)

    print(f"SVM with pca n = 20 accuracy {n_splits}-fold: {np.mean(acc10)}")
    print(f"SVM with pca n = 10 accuracy {n_splits}-fold: {np.mean(acc20)}")
```

```
SVM with pca n = 20 accuracy 5-fold: 1.0
SVM with pca n = 10 accuracy 5-fold: 1.0
```

مشاهده می‌شود که در هر دو صورت، ما دقت ۱۰۰ درصد از داده‌ها دریافت می‌کنیم.

در ادامه برای بررسی بیشتر، با استفاده از یک حلقه، PCA را با کامپوننت‌های مختلف از ۱ تا ۱۵ بررسی می‌کنیم و دقت را گزارش می‌کنیم. در این جا نیز از k-fold استفاده شده است.



مشاهده می‌شود که اگر تعداد کامپوننت‌ها را در PCA کمتر از ۵ در نظر بگیریم، دقت طبقه‌بند SVM ما کاهش می‌یابد.

با تشکر.