نام خدا

تمرین سوم درس هوش مصنوعی

سوال اول

محمد امین سلطانی چم حیدری

۸۱۰۶۰۱۰۸۱

استاد مربوطه:

دکتر مسعود شریعت پناهی

بهار۱۴۰۲

بخش اول: دسته‌بندی گیاهان بر پایهٔ ویژگی‌های ظاهری آن‌ها

فناوری‌های گلخانهٔ هوشمند پیش از هر چیز نیازمند تشخیص خودکار نوع گیاهان مورد نظر است. این تشخیص معمولا به کمک ویژگی‌های ظاهری گیاهان صورت می‌گیرد. دادگان پیوست (فایل iris.csv) شامل پنج ستون است که چهار ستون اول ورودی‌ها (ویژگی‌ها)ی گیاه شامل طول و عرض کاسبرگ (sepal) و طول و عرض گلبرگ (petal) و ستون پنجم خروجی یا دسته (class) گیاه را نشان می‌دهد.

الف) ابتدا ۸۰٪ داده‌ها (شامل ۸۰٪ از داده‌های هر دسته) را برای آموزش و ۲۰٪ باقی مانده (شامل ۲۰٪ از داده‌های هر دسته) را برای آزمایش در نظر بگیرید. سپس به کمک الگوریتم KNN (K نزدیک ترین همسایه) با K= 5 مدل را برای دسته‌بندی گیاهان تربیت کنید و با محاسبهٔ خروجی‌های مدل برای داده‌های آزمایش و تشکیل ماتریس سردرگمی، امتیازهای accuracy, recall, precision و jaccard را محاسبه کنید. (برای آشنایی با امتیاز jaccard عبارت "multiclass jaccard similarity score" را جستجو کنید).

required libraries

In [10]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import pylab as pl
from sklearn import preprocessing
```

به منظور تربیت مدل برای این سوال ابتدا کتابخانه های مورد نظر در Jupyter notebook فراخوانی شده اند.

matplotlib ← نمایش data و رسم نمودار

pandas ← کار با داده های ساختار یافته مانند جداول،فایل csv و excel و...

Numpy ← انجام عملیات ریاضی،آماری و عددی

Pylab ← تجسم داده های علمی و عملی.ترکیبی از matplotlib و numpy می باشد

Sickit_learn ← یادگیری ماشین و تحلیل داده

open file

In [35]:
```
Iris_df = pd.read_csv("Iris.csv")
print(Iris_df.dtypes)
Iris_df.head()
```

```
Sepal_Length    float64
Sepal_Width     float64
Petal_Length    float64
Petal_Width     float64
Class            object
dtype: object
```

Out[35]:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

در مرحله بعد داده های مربوط به گیاه شامل ویژگی های ورودی و کلاس خروجی به کمک دستور pd از کتابخانه pandas باز شده اند.

clear data

In [36]:
```
print (Iris_df.info())
Iris_df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Sepal_Length  150 non-null    float64
 1   Sepal_Width   150 non-null    float64
 2   Petal_Length  150 non-null    float64
 3   Petal_Width   150 non-null    float64
 4   Class         150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
```

Out[36]:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

در قسمت بعد توصیفی از داده های ورودی مشاهده می شود.این مرحله به ان علت انجام شده که در صورت وجود هرگونه نقص و عیب در داده های ورودی سطر و ستون مربوط به ان داده ورودی حذف شود که البته در این سوال مشاهده می شود هیچ گونه نقصی در داده های ورودی وجود ندارد.

defin x & y

In [37]:
```
feature_df = Iris_df[['Sepal_Length', 'Sepal_Width','Petal_Length','Petal_Width']]
X = np.asarray(feature_df)
X[0:5]
```

Out[37]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

In [38]:
```
Iris_df['Class'] = Iris_df['Class'].astype('object')
y = np.asarray(Iris_df['Class'])
y [0:20]
```

Out[38]:
```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa'],
      dtype=object)
```

در این قسمت داده های ورودی که در یک فایل شامل 5 ستون قرار داشتند از یکدیگر تفکیک می شوند به این صورت که داده های ستون 1 تا 4 در متغیر X به عنوان ورودی و داده های ستون Class به عنوان خروجی در متغیر y ذخیره می شوند.

الف) ابتدا ۸۰٪ داده‌ها (شامل ۸۰٪ از داده‌های هر دسته) را برای آموزش و ۲۰٪ باقی مانده (شامل ۲۰٪ از داده‌های هر دسته) را برای آزمایش در نظر بگیرید. سپس به کمک الگوریتم KNN (K نزدیک ترین همسایه) با K= 5 مدل را برای دسته‌بندی گیاهان تربیت کنید و با محاسبهٔ خروجی‌های مدل برای داده‌های آزمایش و تشکیل ماتریس سردرگمی، امتیازهای accuracy, recall, precision و jaccard را محاسبه کنید. (برای آشنایی با امتیاز jaccard عبارت "multiclass jaccard similarity score" را جستجو کنید).

Data separation

In [56]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =train_test_split( X, y, test_size=0.2 ,random_state=50)
print ('Train set:', X_train.shape,  y_train.shape)
print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (120, 4) (120,)
Test set: (30, 4) (30,)
```

در صورت سوال ذکر شده که 80 درصد از داده ها برای تمرین و 20 درصد از داده ها برای تست در نظر گرفته شوند. برای این کار می توان از کتابخانه scikit learn استفاده کرد یا به صورتی که در کد بالا مشاهده می شود داده ها را به دوگروه تمرین و تست تقسیم کرد.

همچنین به منظور اینکه نتایج گزارش شده در این فایل با نتایج بدست آمده از کد همواره برابر باشد حالت random state به صورت ثابت تعریف می شود.می توان برای تصادفی بدست آمدن داده ها حالت random state را پاک کرد که در این صورت با هربار run کردن کد باتوجه به اینکه داده ها تصادفی در نظر گرفته می شوند نتایج متفاوت خواهد بود.

Training of the KNN algorithm for the model

In [60]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
y_true = y
y_pred
```

Out[60]:
```
array(['Iris-versicolor', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica'],
      dtype=object)
```

برای تربیت مدل با روش KNN برای دسته بندی گیاهان ابتدا از کتابخانه sklearn.neighbors دستور KNeighborsClassifier فراخوانی می شود. سپس K مربوط به این روش برابر با 3 قرار داده می شود و این مدل روی داده های آموزش fit می شود. پس از اینکه مدل روی داده های آموزش fit شد به کمک این مدل تربیت شده داده های جدا شده برای تست پیشبینی می شوند و در متغیر y_pred قرار می گیرند.

calculating the confusion matrix

In [62]:
```
from sklearn.metrics import confusion_matrix
y_pred = knn.predict(X_test)

labels = np.unique(y_train)
y_test_numeric = np.array([np.where(labels == label)[0][0] for label in y_test])
y_pred_numeric = np.array([np.where(labels == label)[0][0] for label in y_pred])

cm = confusion_matrix(y_test_numeric, y_pred_numeric)
print(cm)
```

```
[[ 9  0  0]
 [ 0 11  1]
 [ 0  0  9]]
```

در مرحله بعد برای محاسبه ماتریس سردرگمی ابتدا دستور مربوط به آن از sklearn.matrix فراخوانی می شود. برای تشکیل ماتریس سردرگمی باید داده های خروجی تست با داده های پیش بینی شده مقایسه شوند. از طرفی ستون خروجی از نوع object می باشد، ولی برای مقایسه ی دو مقدار به مقادیر کمی y_pred و y_test نیاز می باشد.

در پایان دستور فراخوانی شده برای تشکیل ماتریس سردرگمی بر روی داده های y_test_numeric و y_pred_numeric اعمال می شود و ماتریس سردرگمی به صورت شکل بالا یک ماتریس 3*3 محاسبه می شود.

در این ماتریس مقادیر ستون اصلی مقادیری هستند که به درستی پیش بینی شده اند.(True positive)

In [64]:
```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import jaccard_score
from sklearn.metrics import accuracy_score
precision_macro = precision_score(y_test_numeric, y_pred_numeric, average='macro')
recall_macro = recall_score(y_test_numeric, y_pred_numeric, average='macro')
jaccard_macro = jaccard_score(y_test_numeric, y_pred_numeric, average='macro')
print("Precision with macro-averaging:         ", precision_macro)
print("Recall with macro-averaging:            ", recall_macro)
print("Jaccard similarity with macro-averaging:", jaccard_macro)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:                               ', accuracy)
```

```
Precision with macro-averaging:          0.9666666666666667
Recall with macro-averaging:             0.9722222222222222
Jaccard similarity with macro-averaging: 0.9388888888888888
Accuracy:                                0.9666666666666667
```

مقادیر accuracy , jaccard similarity , Recall , Precision مطابق اعداد بالا بدست آمده اند.

Accuracy :

با توجه به اینکه دقت از مقادیر قطری ماتریس سردرگمی حساب می شود نیاز به میانگین گیری برای آن نیست و برای ماتریس سردرگمی به ازای همه خروجی ها یک accuracy بدست می آید.

Recall :

نسبت تعداد نمونه‌هایی است که الگوریتم به درستی به عنوان مثبت تشخیص داده است به تعداد کل نمونه‌های مثبت در داده‌ها.

Recall = TP / (TP + FN)

Jaccard :

برابر با اندازه‌ی تقاطع دو مجموعه تقسیم بر اندازه‌ی اتحاد آن دو مجموعه است. به عبارت دیگر، این امتیاز نشان می‌دهد که چه مقدار اشتراک بین دو مجموعه وجود دارد.

|J(A, B) = |A ∩ B| / |A ∪ B

Precision :

اندازه‌ی تعداد نمونه‌های مثبتی که الگوریتم به درستی به عنوان مثبت تشخیص داده است، تقسیم بر تعداد کل نمونه‌هایی که الگوریتم به عنوان مثبت تشخیص داده است.

Precision = TP / (TP + FP)

ب) با اعمال یک نرمال سازی استاندارد روی داده‌ها، خواسته‌های بند الف را (با همان روش و با همان مقدار K) مجددا بدست آورید. آیا امتیاز دقت (accuracy) تغییر می‌کند؟ چرا؟

normalizing

In [90]:
```
feature_df = Iris_df[['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']]
X = np.asarray(feature_df)
Scaler= preprocessing.StandardScaler().fit(X)
X =Scaler.transform(X.astype(float))
print(X[0:10])
```

```
[[-0.90068117  1.03205722 -1.3412724  -1.31297673]
 [-1.14301691 -0.1249576  -1.3412724  -1.31297673]
 [-1.38535265  0.33784833 -1.39813811 -1.31297673]
 [-1.50652052  0.10644536 -1.2844067  -1.31297673]
 [-1.02184904  1.26346019 -1.3412724  -1.31297673]
 [-0.53717756  1.95766909 -1.17067529 -1.05003079]
 [-1.50652052  0.80065426 -1.3412724  -1.18150376]
 [-1.02184904  0.80065426 -1.2844067  -1.31297673]
 [-1.74885626 -0.35636057 -1.3412724  -1.31297673]
 [-1.14301691  0.10644536 -1.2844067  -1.4444497 ]]
```

برای قسمت ب ابتدا داده های ورودی با دستور preprocessing از کتابخانه sklearn فراخوانی می شود و به کمک آن داده های ورودی normalize می شوند.

پس از اعمال normalizing سایر مراحل مانند قسمت الف انجام می شود تا مقادیر خواسته شده ی accuracy , Precision , Recall , jaccard similarity برای این سمت نیز محاسبه شوند.

یعنی مانند قسمت الف داده ها به دوقسمت تست و آموزش تقسیم می شوند و مدل با الگوریتم KNN با (K=3) تربیت می شود تا خروجی های مطلوب پیش بینی شوند.

calculating the confusion matrix

In [93]:
```
from sklearn.metrics import confusion_matrix
y_pred = knn.predict(X_test)

labels = np.unique(y_train)
y_test_numeric = np.array([np.where(labels == label)[0][0] for label in y_test])
y_pred_numeric = np.array([np.where(labels == label)[0][0] for label in y_pred])

cm = confusion_matrix(y_test_numeric, y_pred_numeric)
print(cm)
```

```
[[ 9  0  0]
 [ 0 10  2]
 [ 0  0  9]]
```

calculating the precision,recall,jaccard,accuracy

In [94]:
```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import jaccard_score
from sklearn.metrics import accuracy_score
precision_macro = precision_score(y_test_numeric, y_pred_numeric, average='macro')
recall_macro = recall_score(y_test_numeric, y_pred_numeric, average='macro')
jaccard_macro = jaccard_score(y_test_numeric, y_pred_numeric, average='macro')
print("Precision with macro-averaging:         ", precision_macro)
print("Recall with macro-averaging:            ", recall_macro)
print("Jaccard similarity with macro-averaging:", jaccard_macro)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:                               ', accuracy)
```

```
Precision with macro-averaging:          0.9393939393939394
Recall with macro-averaging:             0.9444444444444445
Jaccard similarity with macro-averaging: 0.883838383838384
Accuracy:                                0.9333333333333333
```

همانطور که مشاهده می شود مقادیر accuracy , jaccard similarity , Recall , Precision در این قسمت نیز پس از normalizing داده های ورودی بدست آورده شده اند.

همانطور که مشاهده می شود همه ی این مقادیر پس از normalizing کاهش یافته اند که در ادامه به مقایسه ماتریس سردر گمی و accuracy , jaccard similarity , Recall , Precision قبل و پس از اعمال normalizing پرداخته می شود.

قبل از normalizing :

```
Precision with macro-averaging:          0.9666666666666667
Recall with macro-averaging:             0.9722222222222222
Jaccard similarity with macro-averaging: 0.9388888888888888
Accuracy:                                0.9666666666666667

[[ 9  0  0]
 [ 0 11  1]
 [ 0  0  9]]
```

بعد از normalizing :

```
Precision with macro-averaging:          0.9393939393939394
Recall with macro-averaging:             0.9444444444444445
Jaccard similarity with macro-averaging: 0.883838383838384
Accuracy:                                0.9333333333333333

[[ 9  0  0]
 [ 0 10  2]
 [ 0  0  9]]
```

همانطور که مشاهده می شود پس از نرمال سازی داده های ورودی دقت پیشبینی مدل کاهش یافته است.

در مدل KNN، نرمال‌سازی داده‌ها می‌تواند در برخی موارد بهبود دقت را به دنبال داشته باشد اما در موارد دیگر، دقت را کاهش می‌دهد. یک دلیل این کاهش دقت پس از نرمال‌سازی داده‌ها در مدل KNN می‌تواند از دست‌رفتن اطلاعات پیرامونی باشد.

در مدل KNN، برای پیدا کردن همسایگان نزدیک به نمونه‌ی جدید، از فاصله‌ی اقلیدسی یا همان فاصله‌ی مربعی استفاده می‌شود. با نرمال‌سازی داده‌ها، مقادیر آن‌ها به یک دامنه‌ی مشخصی تغییر می‌کنند؛ به عبارت دیگر، مقادیر بیشتر و کوچکتر به یک‌دیگر نزدیک‌تر می‌شوند. این تغییر در مقادیر، ممکن است باعث شود که فاصله‌ی اقلیدسی (یا فاصله‌ی مربعی) بین دو نمونه کوچک‌تر یا بزرگ‌تر از واقعیت شود. با این حال، در مدل KNN، فاصله‌ی اقلیدسی (یا فاصله‌ی مربعی) بین نمونه‌ها بسیار مهم است و به عنوان یکی از معیارهای اصلی برای پیدا

کردن همسایگان نزدیک استفاده می‌شود. بنابراین، نرمال‌سازی داده‌ها ممکن است باعث از دست رفتن اطلاعات پیرامونی و بهبود دقت در مدل KNN نشود.

به طور کلی، در مدل KNN، نرمال‌سازی داده‌ها باید با دقت انجام شود و اگر نرمال‌سازی بهبودی در دقت مدل نداشت، می‌تواند به عنوان یکی از عوامل کاهش دقت در نظر گرفته شود. بهترین روش برای تعیین اینکه آیا نرمال‌سازی داده‌ها در مدل KNN بهبودی در دقت مدل دارد یا نه، تست مدل با و بدون نرمال‌سازی داده‌ها است.