دانشگاه کاشان
دانشکده برق و کامپیوتر

گزارش پروژه درس «مبانی داده‌کاوی»
در رشته کامپیوتر گرایش نرم‌افزار

توسط:
حمیدرضا مهدوی‌پناه
۹۳۲۱۱۷۰۲۱۱

استاد درس:
دکتر سید مهدی وحیدی‌پور

۱۳۹۶/۱۱/۲۰

فهرست مطالب

توضیح دیتاست

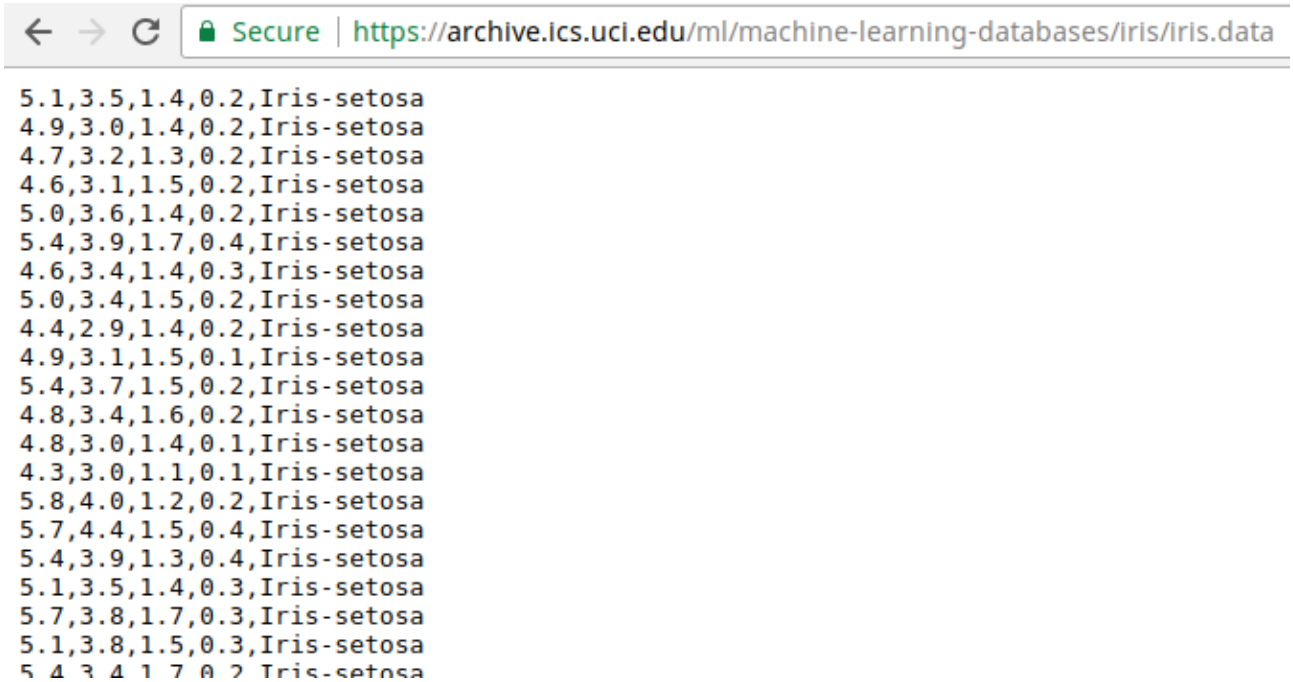
دیتاست مورد استفاده در این گزارش، Iris نام دارد. این دیتاست ۱۵۰ داده از انواع گل‌های زنبق را در خود دارد. این گل‌ها دریکی از سه دسته‌ی زیر جای می‌گیرند:

- Iris Setosa
- Iris Versicolour
- Iris Virginica

هر داده در این دیتاست چهار فیلد عددی مختلف دارد:

۱. طول کاسبرگ به سانتی‌متر (Sepal length)
۲. عرض کاسبرگ به سانتی‌متر (Sepal width)
۳. طول گلبرگ به سانتی‌متر (Petal length)
۴. عرض گلبرگ به سانتی‌متر (Petal width)

صفحه مربوط به این دیتاست از آدرس زیر قابل دسترسی است:
https://archive.ics.uci.edu/ml/datasets/Iris



تصویر ۱: نمایی ازفایل دیتاست

ابزارها

برای انجام پروژه از ابزارهای زیر استفاده شده است:

۱. زبان برنامه‌نویسی Python :
برای نصب این زبان به آدرس python.org مراجعه کنید.

۲. کتابخانه‌ی یادگیری ماشین scikit-learn :
برای نصب آن به آدرس scikit-learn.org مراجعه کنید.

۳. کتابخانه‌ی مصورسازی گراف Graphviz :
برای نصب آن به آدرس graphviz.org مراجعه کنید.

نحوه‌ی پیاده‌سازی

از آنجا که ابزار سایکیت‌لرن (scikit-learn) یک ابزار برای پروژه‌های صنعتی و کاربردی است، به شکل یک کتابخانه عرضه شده تا بتوان آن را به راحتی در برنامه‌های کاربردی مختلف گنجاند و استفاده کرد. لذا این ابزار از محیط گرافیکی خاصی بهره‌مند نیست و برای استفاده از آن باید یک برنامه‌ی پایتون نوشت و کتابخانه را در آن وارد و استفاده کرد.

سورس پیاده‌سازی پروژه به شکل اپن‌سورس و تحت مجوز MIT از آدرس https://github.com/mahdavipanah/DataMiningProject قابل دسترسی است.

فایل اصلی پروژه یک اسکریپت پایتون با نام main.py است که با اجرای آن اعداد مورد نیاز برای جداول ۱ تا ۳ در خروجی چاپ می‌شود.

خروجی اسکریپت در تصویر ۲ قابل مشاهده است.

در ادامه گزارش به شرح فایل‌های پروژه و اسکریپت اصلی و نحوه‌ی تولید مدل‌ها و ارزیابی آن‌ها پرداخته خواهد شد.

فایل‌های پروژه

۱. main.py : اسکریپت اصلی پروژه است که با اجرای آن مقادیر مورد نیاز جداول در خروجی چاپ می‌شود.
۲. k_medoids.py : پیاده‌سازی الگوریتم KMedoids است که در ادامه توضیح داده خواهد شد.
۳. requirements.txt : ماژول‌های مورد نیاز برای اجرای پروژه است که اگر ابزارهای بخش «ابزارها» در گزارش، به درستی نصب شده باشند نیازی به نصب دوباره‌ی آن‌ها نیست.
۴. iris_data.txt : دیتاست Iris

```
------------------------------------------------------------
                        Classification
------------------------------------------------------------

Decision tree with use training set:
    TP Rate (Sum):  150
    ____________________________________
    FP Rate (Sum):  0
    ____________________________________
    Precision Micro-average:  1.0
    ____________________________________
    Recall Micro-average:  1.0
    ____________________________________
    F-measure Micro-average:  1.0
------------------------------------------------------------
Decision tree with Cross-validation:
    TP Rate (Sum):  23.5
    ____________________________________
    FP Rate (Sum):  51.5
    ____________________________________
    Precision Micro-average:  0.31333333333333335
    ____________________________________
    Recall Micro-average:  0.31333333333333335
    ____________________________________
    F-measure Micro-average:  0.31333333333333335
------------------------------------------------------------
Ada boost:
    TP Rate (Sum):  144
    ____________________________________
    FP Rate (Sum):  6
    ____________________________________
    Precision Micro-average:  0.96
    ____________________________________
    Recall Micro-average:  0.96
    ____________________________________
    F-measure Micro-average:  0.96
------------------------------------------------------------
Random Forest:
    TP Rate (Sum):  150
    ____________________________________
    FP Rate (Sum):  0
    ____________________________________
    Precision Micro-average:  1.0
    ____________________________________
    Recall Micro-average:  1.0
    ____________________________________
    F-measure Micro-average:  1.0

------------------------------------------------------------
                          Clustering
------------------------------------------------------------

K-Means:
    k=2: 0.5193608056059371
    k=3: 0.7483723933229484
    k=4: 0.6456152164718862
    k=5: 0.5917891481655361
------------------------------------------------------------
K-Medoids:
    k=2: 0.5509814129042337
    k=3: 0.7759606304849536
    k=4: 0.6351928142776216
    k=5: 0.5945118250661453
```

تصویر ۲: خروجی اسکریپت و مقادیر جداول ۱ تا ۳

توضیح کد پیاده‌سازی

```
1   from os import path
2
3   from sklearn import tree, metrics
4   from sklearn.model_selection import KFold
5   from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
6   from sklearn.cluster import KMeans
7   from k_medoids import KMedoids
8   import graphviz
9   import numpy as np
10
11  # path of the iris data file
12  iris_file_path = path.join(
13      path.dirname(path.abspath(__file__)),
14      'iris.data.txt'
15  )
16
17  iris_data = ''
18  with open(iris_file_path) as iris_file:
19      iris_data = iris_file.read()
20
21  # contains attributes with the class itself
22  iris_data = [data.split(',') for data in iris_data.split('\n')][:-2]
23
24  # convert numeric values from string to float
25  for data in iris_data:
26      for i in range(4):
27          data[i] = float(data[i])
28
29  # contains attributes
30  iris_training_data = [data[:-1] for data in iris_data]
31
32  # contains class labels of training values
33  iris_training_labels = [data[-1] for data in iris_data]
```

تصویر ۳

خطوط ۱ تا ۹ : مربوط به وارد کردن کتابخانه‌ها، کلاس‌ها و توابع مورد نیاز برای برنامه می‌باشد که در ادامه کاربرد هرکدام مشخص خواهد شد.

خطوط ۱۲ تا ۱۵ : متغیر iris_file_path را برابر آدرس مطلق فایلی با نام iris.data.txt در کنار اسکریپت برنامه، مقداردهی می‌کنیم.

خطوط ۱۷ تا ۱۹ : فایل را از آدرس خوانده و محتوایت آن را درون متغیر iris_data می‌ریزد.

خط ۲۲ : محتویات فایل را به صورت یک آرایه‌ی دو بعدی درمی‌آوریم که هر سطر آن شامل یک داده از دیتاست می‌باشد و هر ستون آن برابر یک ویژگی داده‌ها است. بنابراین چیزی مثل

iris_data[10][2] ویژگی سوم از داده‌ی یازدهم را برمی‌گرداند. محتویات این متغیر در این مرحله به صورت آرایه دو بعدی و به این شکل است:

```
[
  ['5.1', '3.5', '1.4', '0.2', 'Iris-setosa'],
  ['4.9', '3.0', '1.4', '0.2', 'Iris-setosa'],
  ...
]
```

خطوط ۲۵ تا ۲۷ : از آنجا که ویژگی‌های داده‌ها اعداد اعشاری هستند، ویژگی‌های اول تا چهارم (به استثنای ویژگی پنجم که رشته است و کلاس داده را معلوم می‌کند) به float تبدیل می‌شوند.
محتوای این متغیر در این مرحله بدین صورت است:

```
[
  [5.1, 3.5, 1.4, 0.2, 'Iris-setosa'],
  [4.9, 3.0, 1.4, 0.2, 'Iris-setosa'],
  ...
]
```

خط ۳۰ : متغیر iris_training_data به نحوی مقدار دهی می‌شود که شبیه به ماترس قبل باشد اما ویژگی کلاس از آن حذف شود. محتوای این متغیر در این مرحله به این صورت است:

```
[
  [5.1,'3.5, 1.4, 0.2],
  [4.9, 3.0, 1.4, 0.2],
  ...
]
```

خط ۳۳ : متغیر iris_training_labels به شکل آرایه‌ای از اسامی کلاس‌ها در می‌آید که هر اندیس نشان‌دهنده‌ی کلاس داده‌ی (i+ ۱) ام است (با توجه به اینکه اندیس آرایه از صفر شروع می‌شود).
محتوای این متغیر در این مرحله به این صورت است:

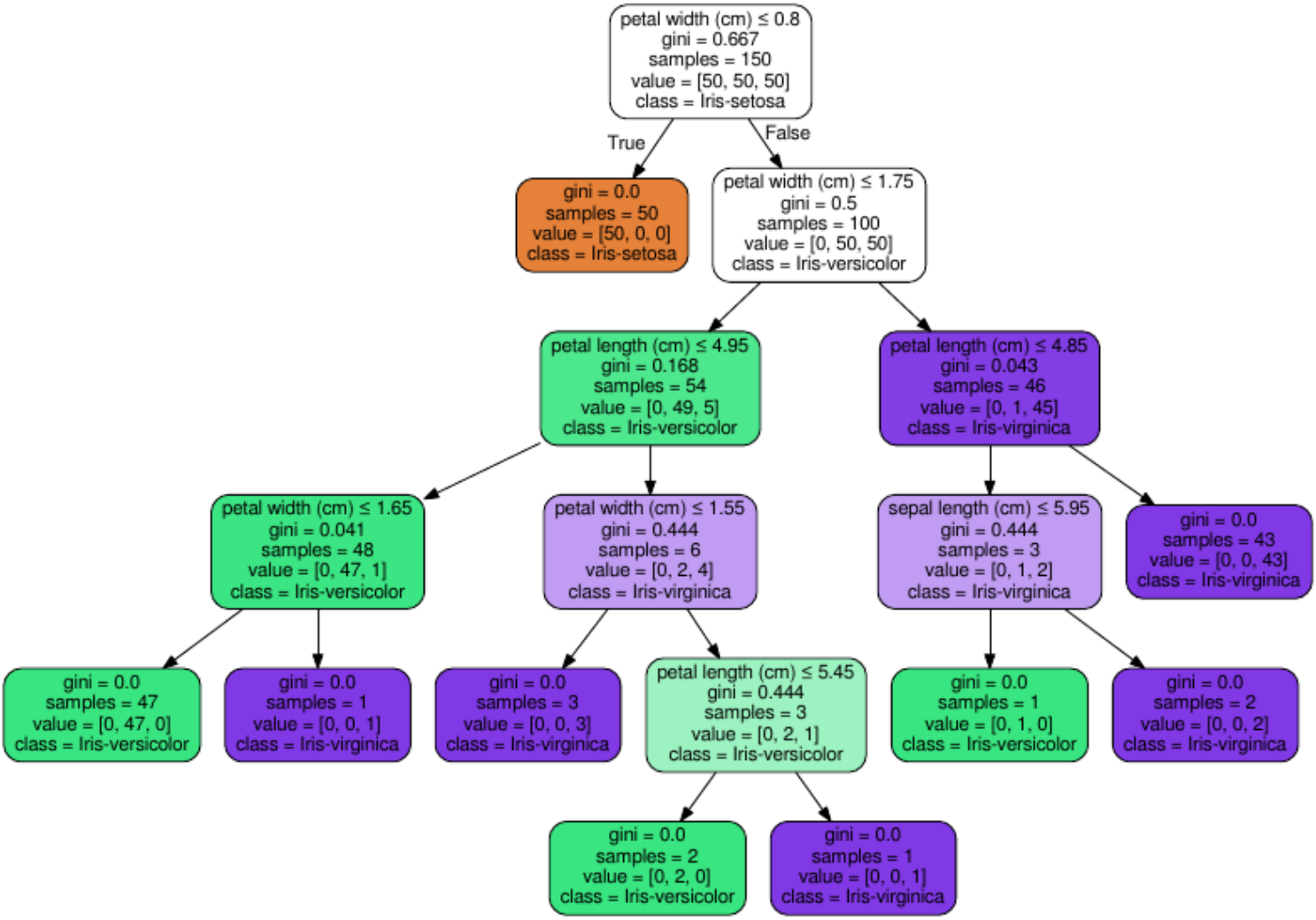
```
[
  'Iris-setosa',
  'Iris-setosa',
  ...
]
```

```
35      clf = tree.DecisionTreeClassifier()
36      clf.fit(iris_training_data, iris_training_labels)
37
38      iris_feature_names = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
39      iris_labels_names = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
40
41      # output the generated tree classifier
42      dot_data = tree.export_graphviz(clf, out_file=None,
43                                      feature_names=iris_feature_names,
44                                      class_names=iris_labels_names,
45                                      filled=True, rounded=True,
46                                      special_characters=True)
47      graph = graphviz.Source(dot_data)
48      graph.render('graphviz/output1/iris')
49
50      # predict the training the data using the generated model
51      predicted_labels = clf.predict(iris_training_data)
52
53      print('------------------------------------------------------------')
54      print('                      Classification                        ')
55      print('------------------------------------------------------------')
56      print()
57      print("Decision tree with use training set:")
58      confusion_matrix = metrics.confusion_matrix(iris_training_labels, predicted_labels, labels=iris_labels_names)
59      print("    TP Rate (Sum): ", confusion_matrix[0][0] + confusion_matrix[1][1] + confusion_matrix[2][2])
60      print("    ____________________________________")
61      print("    FP Rate (Sum): ",
62            confusion_matrix[1][0] + confusion_matrix[2][0] +
63            confusion_matrix[0][1] + confusion_matrix[2][1] +
64            confusion_matrix[0][2] + confusion_matrix[1][2]
65            )
66      print("    ____________________________________")
67      print("    Precision Micro-average: ", metrics.precision_score(
68          y_true=iris_training_labels,
69          y_pred=predicted_labels,
70          labels=iris_labels_names,
71          average='micro'
72      ))
```

تصویر ۴

خط ۳۵ : نمونه‌ای از یک دسته‌بند درخت تصمیم ساخته می‌شود.

خط ۳۶ : دسته‌بند ساخته شده توسط داده‌های آموزشی از قبل آماده شد آموزش داده می‌شود. تابع fit ویژگی‌های داده‌ها را به عنوان ورودی اول و کلاس هرکدام را به عنوان ورودی دوم می‌گیرد و یک درخت تصمیم می‌سازد.
خط ۳۸ تا ۴۸ : درخت تصمیم ساخته شده را مصور می‌کند و در یک فایل ذخیره می‌کند. این قسمت به دلیل ارتباط نداشتن با بحث داده‌کاوی و ابزار سایکیت‌لرن، به تفضیل توضیح داده نمی‌شود. تصویر درخت تصمیم ساخته شده در تصویر ۴ قابل مشاهده است.



تصویر ۵

خط ۵۸ : با توجه به اینکه اولین چیزی که در پی یافتنش هستیم، ارزیابی یک درخت تصمیم ساخته شده با مدل توسط همان داده‌های آموزشی به عنوان داده‌های آزمون است، در این خط متغیر predicted_labels را برابر با مقادیر پیش‌بینی شده توسط مدل قرار می‌دهیم. متد predict از یک دسته‌بند، لیستی از داده‌ها را به عنوان ورودی می‌گیرد و کلاس پیش‌بینی شده‌های آن‌ها را به شکل یک آرایه برمی‌گرداند.

خط ۵۸ : متغیر confusion_matrix برابر با ماتریس درهم‌ریختگیِ مدل ساخته شده قرار داده می‌شود. برای این کار از تابع confusion_matrix در کلاس metrics از کتابخانه‌ی سایکیت‌لرن استفاده می‌شود. این تابع مقادیر واقعی کلاس‌های داده‌های آزمون را به عنوان ورودی اول، مقادیر پیش‌بینی شده‌ی مدل را به عنوان ورودی دوم و اسامی کلاس‌های دیتاست را به عنوان ورودی سوم (یادآوری: در دیتاست ما سه کلاس وجود دارد که انواع گل‌های زنبق هستند) می‌گیرد و به عنوان نتیجه یک ماتریس را برمی‌گرداند. عنصر C[i][j] این ماتریس نشان‌دهنده‌ی تعداد داده‌هایی است که

متعلق به کلاس i هستند اما به عنوان کلاس j پیش‌بینی شده‌اند.

برای اطلاعات بیشتر راجع به تابع confusion_matrix به لینک زیر مراجعه کنید:
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

خط ۵۹ : مقدار TP از جمع TPهای تمام کلاس‌ها محاسبه می‌شود. طبق تعریف ماتریس درهم‌ریختگی (خط ۵۸) TPها درایه‌هایی از ماتریس هستند که شماره‌ی سطر و ستونشان باهم برابر است.

خط ۶۲ تا ۶۴ : مقدار FP از مجموع FPهای تمام کلاس‌ها محاسبه می‌شود.

خط ۶۷ تا ۷۲ : precision توسط تابع precision_score از کلاس metrics محاسبه می‌شود. این تابع کلاس‌های واقعی داده‌های آزمون (که در اینجا همان داده‌های آموزشی هستند) را به عنوان ورودی y_true، کلاس‌های پیش‌بینی شده را به عنوان ورودی y_pred، اسامی کلاس‌های دیتاست را به عنوان ورودی labels و با توجه با اینکه در این دیتاست بیشتر از دو کلاس داریم، نحوه‌ی محاسبه‌ی TP، FP و ... را برای محاسبه‌ی precision به عنوان ورودی average می‌گیرد. در اینجا ما برای average مقدار micro را داده‌ایم که طبق مستندات کتابخانه، روش micro همان مجموع گرفتن معیارهای (شبیه به کاری که در خطوط ۵۹ و ۶۲ تا ۶۴ کردیم) می‌باشد.

```
72      ))
73      print("    ————————————————————————————")
74      print("    Recall Micro-average: ", metrics.recall_score(
75          y_true=iris_training_labels,
76          y_pred=predicted_labels,
77          labels=iris_labels_names,
78          average='micro'
79      ))
80      print("    ————————————————————————————")
81      print("    F-measure Micro-average: ", metrics.f1_score(
82          y_true=iris_training_labels,
83          y_pred=predicted_labels,
84          labels=iris_labels_names,
85          average='micro'
86      ))
87
88      # change iris training data to NumPy array
89      iris_training_data_np = np.array(iris_training_data)
90
91      # change iris training data labels to NumPy array
92      iris_training_labels_np = np.array(iris_training_labels)
93
94      fold_10 = KFold(n_splits=10)
95
96      # Below values will contain each fold's measures
97      tp_rate = []
98      fp_rate = []
99      precision = []
100     recall = []
101     f_measure = []
102
103     # For each fold
104     for train, test in fold_10.split(iris_training_data):
105         train_set = iris_training_data_np[train]
106         train_set_labels = iris_training_labels_np[train]
107
108         test_set = iris_training_data_np[test]
109         test_set_labels = iris_training_labels_np[test]
110
111         clf = tree.DecisionTreeClassifier()
112         clf.fit(train_set, train_set_labels)
113
114         predicted_labels = clf.predict(test_set)
115
116         confusion_matrix = metrics.confusion_matrix(test_set_labels, predicted_labels, labels=iris_labels_names)
117
118         tp_rate.append(confusion_matrix[0][0] + confusion_matrix[1][1] + confusion_matrix[2][2])
```

تصویر ۶

برای اطلاعات بیشتر راجع به تابع ***precision_score*** *به لینک زیر مراجعه کنید:*
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

خط ۷۴ تا ۷۹ : مقدار recall شبیه به مقدار precision در خطوط ۶۷ تا ۷۲ محاسبه می‌شود.
برای اطلاعات بیشتر راجع به تابع ***recall_score*** *به لینک زیر مراجعه کنید:*
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html

خط ۸۱ تا ۸۶ : مقدار F-measure شبیه به مقدار precision در خطوط ۶۷ تا ۷۲ محاسبه می‌شود.
برای اطلاعات بیشتر راجع به تابع ***recall_score*** *به لینک زیر مراجعه کنید:*
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

خط ۸۹ تا ۹۲ : کتابخانه‌ی NumPy که همراه با سایکیت‌لرن باید نصب شود، کتابخانه‌ای است برای انجام محاسبات ماتریسی در پایتون. برای ادامه‌ی کار نیاز داریم تا متغیرهای iris_trainingdata و iris_training_labels به آرایه‌هایی از جنس آرایه‌های کتابخانه‌ی NumPy تبدیل شوند که برای این کار از تابع array در کتابخانه‌ی np (که در خطوط ابتدایی اسکریپت وارد شده) استفاده می‌شود.

در ادامه به اندازه‌گیری معیارها با درخت تصمیم و با روش Cross validation 10-Fold می‌پردازیم.

خط ۹۴ : یک دسته‌بند Kfold ساخته می‌شود که مقدار k آن برابر ۱۰ داده می‌شود.

خط ۹۷ تا ۱۰۱ : برای هر Fold در ادامه معیارها اندازه‌گیری می‌شود و در این آرایه‌ها ریخته می‌شود و در آخر میانگین آن‌ها مقدار آن معیار را مشخص می‌کند.

```
120         fp_rate.append(confusion_matrix[1][0] + confusion_matrix[2][0] +
121                        confusion_matrix[0][1] + confusion_matrix[2][1] +
122                        confusion_matrix[0][2] + confusion_matrix[1][2])
123
124         precision.append(metrics.precision_score(
125             y_true=test_set_labels,
126             y_pred=predicted_labels,
127             labels=iris_labels_names,
128             average='micro'
129         ))
130         recall.append(metrics.recall_score(
131             y_true=test_set_labels,
132             y_pred=predicted_labels,
133             labels=iris_labels_names,
134             average='micro'
135         ))
136         f_measure.append(metrics.f1_score(
137             y_true=test_set_labels,
138             y_pred=predicted_labels,
139             labels=iris_labels_names,
140             average='micro'
141         ))
142
143     print('---------------------------------------------------------------')
144     print("Decision tree with Cross-validation:")
145     print("    TP Rate (Sum): ", np.average(tp_rate))
146     print("    ----------------------------------")
147     print("    FP Rate (Sum): ", np.average(fp_rate))
148     print("    ----------------------------------")
149     print("    Precision Micro-average: ", np.average(precision))
150     print("    ----------------------------------")
151     print("    Recall Micro-average: ", np.average(recall))
152     print("    ----------------------------------")
153     print("    F-measure Micro-average: ", np.average(f_measure))
154
155     clf = AdaBoostClassifier()
156     clf.fit(iris_training_data, iris_training_labels)
157
158     # predict the training the data using the generated model
159     predicted_labels = clf.predict(iris_training_data)
```

تصویر ۷

خط ۱۰۴ تا ۱۴۱ : تابع split از کلاس KFold یک دیتاست را به عنوان ورودی می‌گیرد و سپس در هر بار حلقه اندیس‌هایی از داده‌هایی که باید به عنوان داده‌های آموزشی باشند و داده‌هایی که باید به عنوان داده‌های آزمون باشند را برمی‌گرداند. سپس ما هر بار یک درخت تصمیم را بر اساس داده‌ی آموزشی آن دور(همان Fold) می‌سازیم و سپس معیارهای مورد نظر را برای آن مدل با استفاده از داده‌های آزمون آن دور اندازه می‌گیریم و مقادیر معیارها را به لیست آن معیار اضافه می‌کنیم.

خط ۱۴۳ تا ۱۵۳ : با استفاده از تابع average در کتابخانه‌ی NumPy برای هر معیار، میانگین معیارهای اندازه‌گیری شده در فولدهای (Fold) آن معیار را محاسبه کرده و گزارش می‌کنیم. تابع average یک آرایه را به عنوان ورودی گرفته و میانگین مقادیرش را برمیگرداند.

```
161     print('------------------------------------------------------------')
162     print("Ada boost:")
163     confusion_matrix = metrics.confusion_matrix(iris_training_labels, predicted_labels, labels=iris_labels_names)
164     print("    TP Rate (Sum): ", confusion_matrix[0][0] + confusion_matrix[1][1] + confusion_matrix[2][2])
165     print("    ----------------------------------")
166     print("    FP Rate (Sum): ",
167           confusion_matrix[1][0] + confusion_matrix[2][0] +
168           confusion_matrix[0][1] + confusion_matrix[2][1] +
169           confusion_matrix[0][2] + confusion_matrix[1][2]
170           )
171     print("    ----------------------------------")
172     print("    Precision Micro-average: ", metrics.precision_score(
173         y_true=iris_training_labels,
174         y_pred=predicted_labels,
175         labels=iris_labels_names,
176         average='micro'
177     ))
178     print("    ----------------------------------")
179     print("    Recall Micro-average: ", metrics.recall_score(
180         y_true=iris_training_labels,
181         y_pred=predicted_labels,
182         labels=iris_labels_names,
183         average='micro'
184     ))
185     print("    ----------------------------------")
186     print("    F-measure Micro-average: ", metrics.f1_score(
187         y_true=iris_training_labels,
188         y_pred=predicted_labels,
189         labels=iris_labels_names,
190         average='micro'
191     ))
192
193     clf = RandomForestClassifier()
194     clf.fit(iris_training_data, iris_training_labels)
195
196     # predict the training the data using the generated model
197     predicted_labels = clf.predict(iris_training_data)
198
199     print('------------------------------------------------------------')
200     print("Random Forest:")
201     confusion_matrix = metrics.confusion_matrix(iris_training_labels, predicted_labels, labels=iris_labels_names)
202     print("    TP Rate (Sum): ", confusion_matrix[0][0] + confusion_matrix[1][1] + confusion_matrix[2][2])
203     print("    ----------------------------------")
204     print("    FP Rate (Sum): ",
205           confusion_matrix[1][0] + confusion_matrix[2][0] +
206           confusion_matrix[0][1] + confusion_matrix[2][1] +
207           confusion_matrix[0][2] + confusion_matrix[1][2]
208           )
```

تصویر ۸

خط ۱۵۵ تا ۱۹۱ : مشابه با دسته‌بندهای قبلی است با این تفاوت که این‌بار از AdaboostClassifier استفاده شده.

```
209     print("    ————————————————————————————————")
210     print("    Precision Micro-average: ", metrics.precision_score(
211         y_true=iris_training_labels,
212         y_pred=predicted_labels,
213         labels=iris_labels_names,
214         average='micro'
215     ))
216     print("    ————————————————————————————————")
217     print("    Recall Micro-average: ", metrics.recall_score(
218         y_true=iris_training_labels,
219         y_pred=predicted_labels,
220         labels=iris_labels_names,
221         average='micro'
222     ))
223     print("    ————————————————————————————————")
224     print("    F-measure Micro-average: ", metrics.f1_score(
225         y_true=iris_training_labels,
226         y_pred=predicted_labels,
227         labels=iris_labels_names,
228         average='micro'
229     ))
230 
231     print()
232     print('-------------------------------------------------------------')
233     print('                        Clustering                           ')
234     print('-------------------------------------------------------------')
235     print()
236     print("K-Means:")
237     for k in range(2, 6):
238         k_means = KMeans(n_clusters=k)
239         k_means.fit(iris_training_data)
240         predicted_labels = k_means.predict(iris_training_data)
241         print("    k={}: {}".format(
242             k,
243             metrics.adjusted_mutual_info_score(iris_training_labels, predicted_labels)
244         ))
245 
246     print('-------------------------------------------------------------')
247     print("K-Medoids:")
248     for k in range(2, 6):
249         k_medoids = KMedoids(n_clusters=k)
250         k_medoids.fit(iris_training_data)
251         predicted_labels = k_medoids.predict(iris_training_data)
252         print("    k={}: {}".format(
253             k,
254             metrics.adjusted_mutual_info_score(iris_training_labels, predicted_labels)
255         ))
256 
```

تصویر ۹

خط ۱۹۳ تا ۲۲۹ : مشابه با دسته‌بندهای قبلی است با این تفاوت که این‌بار از RandomForestClassifier استفاده شده.

از اینجا به بعد، قسمت خوشه‌بندی آغاز می‌شود و اطلاعات مربوط به جدول ۳ استخراج می‌شود.

خط ۲۳۷ : با توجه به اینکه الگوریتم‌های خوشه‌بندی را باید برای تعداد خوشه‌های مختلف (بین ۲ تا ۵) اجرا کنیم، این حلقه هر بار یک k را به ما می‌دهد تا با آن یک خوشه‌بند جدید بسازیم و معیار ارزیابی را اندازه‌گیری و گزارش کنیم.

خط ۲۳۸ : ابتدا شی جدیدی از کلاس KMeans ایجاد می‌کنیم و مقدار k حلقه را به عنوان ورودی تعداد خوشه به ورودی n_clusters آن می‌دهیم.

خط ۲۳۹ : خوشه‌بند ساخته شده را با داده‌های آموزشی، آموزش می‌دهیم.

خط ۲۴۰ : داده‌های آموزشی را به خوشه‌بند می‌دهیم تا دریابیم که چه خوشه‌هایی را به آن‌ها اختصاص داده و این خوشه‌های اختصاص یافته را در قالب یک آرایه برمیگرداند و ما آن را در متغیر predicted_values ذخیره می‌کنیم.

خط ۲۴۳ : با استفاده از معیار Adjusted Mutual Info Score میزان توافق بین دسته‌بندی واقعی داده‌ها و خوشه‌بندی‌ای که خوشه‌بند به ما ارائه داده را پیدا می‌کنیم. نتیجه عددی است بین صفر و یک. صفر به معنی این است که خوشه‌بندی هیچ شباهتی به دسته‌بندی واقعی داده‌ها ندارد و یک به معنی توافق کامل دسته‌بندی با خوشه‌بندی می‌باشد.

برای اطلاعات بیشتر راجع به این معیار به لینک‌های زیر مراجعه کنید:

- *http://scikit-learn.org/stable/modules/clustering.html#mutual-information-based-scores*
- *https://en.wikipedia.org/wiki/Adjusted_mutual_information*

خط ۲۴۸ تا ۲۵۵ : دقیقاً کار مشابه با KMeans (خطوط ۲۳۷ تا ۲۴۴) انجام می‌گیرد با این تفاوت که به جای الگوریتم KMeans از الگوریتم KMedoids استفاده می‌شود.
الگوریتم KMedoids در کتابخانه‌ی سایکیت‌لرن پیاده‌سازی نشده، لذا برای این الگوریتم از یک پیاده‌سازی سوم شخص که با کتابخانه‌ی سایکیت‌لرن سازگار است استفاده شده. این پیاده‌سازی در فایل k_medoids.py قرار دارد و منبع آن لینک زیر می‌باشد:
https://github.com/salspaugh/machine_learning/blob/master/clustering/kmedoids.py