دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش پروژه دوم درس مبانی یادگیری ماشین

مهرآذین مرزوق – ۴۰۰۳۶۱۳۰۵۵

فهرست

Feature Extraction

در این بخش قصد داریم از روی عکس‌های موجود در دیتاست، ویژگی‌های جدید استخراج کنیم.

```
df = pd.read_csv("leaves.csv", header=None)
df = df.join(get_new_features())
df.to_csv("new_data.csv", index=False)
```

get_new_features

در این تابع روی پوشه‌ی مربوط به هر گروه برگ iterate می‌کنیم و سپس روی هر عکس نیز iterate می‌کنیم. سپس ویژگی‌های هر عکس را استخراج می‌کنیم و در یک دیتافریم ذخیره می‌کنیم. در نهایت باید این دیتافریم و دیتافریم اصلی را یکی کنیم.

```
def get_new_features():
    cols = list()
    for i in range(16, 2516):
        cols.append(f'{i}')
    new_df = pd.DataFrame(columns=cols, dtype=float)

    for i in range(1, 37):
        if 15 < i < 22:
            continue
        path = f'leaves'
        full_path = find_folder(path, i)
        for filename in os.listdir(full_path):
            image_path = full_path + "\\" + filename
            row = extract_features(image_path)
            new_df.loc[new_df.shape[0]] = row

    return new_df
```

find_folder

این تابع برای پیدا کردن نام فولدری استفاده می‌شود که فقط کاراکترهای اول آن را می‌دانیم.

```
def find_folder(p: str, first_char: int):
    for entry in os.listdir(p):
        a = int(first_char / 10)
        b = first_char % 10
        if a == 0:
            if os.path.isdir(os.path.join(p, entry)) and entry[0] == str(b):
                return os.path.join(p, entry)  # Return full path
        else:
            if os.path.isdir(os.path.join(p, entry)) and entry[0] == str(a) and entry[1] ==
str(b):
                return os.path.join(p, entry)  # Return full path

    return None
```

extract_features

این تابع برای استخراج هر پیکسل عکس به عنوان یک عدد استفاده می‌شود.

```
def extract_features(image_path):
    image = Image.open(image_path)
    image = image.convert('L')
    image = image.resize((50, 50))
    features = list(image.getdata())
    return features
```

Classification

Dataframes

در این بخش، ابتدا فایل csv جدید که حاوی ویژگی‌های استخراج‌شده از عکس‌ها نیز هست را لود می‌کنیم و دیتا را به train و test تقسیم می‌کنیم.

```
df = pd.read_csv("new_data.csv")
df.drop('1',inplace=True,axis=1)

train , test = train_test_split(df, test_size=0.2, random_state=200)

test_y = test['0']
test.drop('0', axis = 1, inplace=True)
train_y = train['0']
train.drop('0', axis = 1, inplace=True)
```

Preprocessing

Outlier Analysis

در این مرحله داده‌های outlier را حذف می‌کنیم. علت این‌که whisker_width برابر با عدد نسبتا بزرگی است این است که در این دیتاست، اینکه داده‌ای از boxplot خارج باشد لزوما به معنای نویز بودن آن داده نیست. بلکه می‌تواند داده چالشی باشد. پس تعداد کمی از داده‌های خارج از boxplot را حذف می‌کنیم.

این عدد با آزمون و خطا به‌دست آمده‌است.

```
import numpy as np

whisker_width = 10

for col in train.columns:
  Q1 = train[col].quantile(0.25)
  Q3 = train[col].quantile(0.75)
```

```
IQR = Q3 - Q1
low_bound = Q1 - whisker_width * IQR
high_bound = Q3 + whisker_width * IQR
for i in train.index:
  if train.loc[i, col] < low_bound or train.loc[i, col] > high_bound:
    train.drop(i, inplace=True)
    train_y.drop(i, inplace=True)
```

normalization

در مرحله بعد باید داده‌ها را نرمال‌سازی کرد. برای شروع از نرمال‌سازی استاندارد استفاده می‌کنیم.

```
import numpy as np
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(train)

train = scaler.transform(train)
test = scaler.transform(test)

train = pd.DataFrame(train)
test = pd.DataFrame(test)
```

KNN

در مسائل دسته‌بندی، نخستین الگوریتمی که می‌توانیم بررسی کنیم، KNN است.

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(train, train_y)
```

```
test_y_pred = knn.predict(test)

print("Accuracy:", accuracy_score(test_y, test_y_pred))
```

دقت به‌دست آمده را به تفکیک مقادیر مختلف n_neighbors :

۲: ۴۱ درصد

۳: ۴۵ درصد

۴: ۳۹ درصد

۵: ۳۶ درصد

۶: ۳۹ درصد

۷: ۳۸ درصد

با توجه به اینکه حداکثر دقت به‌دست آمده برابر با ۴۵ درصد است، پس الگوریتم KNN مناسب این دیتاست نیست.

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

model = DecisionTreeClassifier(criterion="gini", max_depth=10, min_samples_split=5)
model.fit(train,train_y)

predictions = model.predict(test)

accuracy = accuracy_score(test_y, predictions)
print("Accuracy:", accuracy)
```

دقت مدل با عوض کردن criterion :

gini : ۳۵ درصد

entropy : ۲۲ درصد

log_loss : ۲۶ درصد

پس با انتخاب gini به عنوان criterion به بررسی حداکثر عمق می‌پردازیم:

۴: ۱۴ درصد

۶: ۲۶ درصد

۸: ۳۵ درصد

۱۰: ۳۵ درصد

۱۲: ۳۵ درصد

پس با انتخاب ۸ به عنوان حداکثر عمق به بررسی min_samples_split می‌پردازیم:

۳: ۳۵ درصد

۵: ۳۵ درصد

۱۰: ۳۵ درصد

دقت این مدل حداکثر ۳۵ درصد است؛ و می‌توان نتیجه گرفت که درخت تصمیم مناسب این دیتاست نیست.

Random Forest

الگوریتمی که یک مرحله از درخت تصمیم قوی‌تر است، الگوریتم جنگل تصادفی‌است.

```
from sklearn.ensemble import RandomForestClassifier

random_forest = RandomForestClassifier(n_estimators=100, criterion="gini",
max_depth=40, min_samples_split=7)

random_forest.fit(train, train_y)
```

```
predictions = random_forest.predict(test)

accuracy = accuracy_score(test_y, predictions)
print("Accuracy:", accuracy)
```

همانطور که در بخش قبل آزمایش شد، برای هایپرپارامتر criterion ، gini بهترین انتخاب است.

بررسی n_estimators :

۲۵: ۴۰ درصد

۵۰ : ۴۴ درصد

۱۰۰ : ۴۱ درصد

۲۰۰ : ۴۱ درصد

۳۰۰ : ۴۲ درصد

با انتخاب ۵۰ به بررسی حداکثر عمق می‌پردازیم.

۵: ۳۹ درصد

۱۰: ۴۴ درصد

۲۰: ۴۱ درصد

۴۰: ۴۱ درصد

۱۰۰: ۳۱ درصد

با انتخاب ۱۰ به بررسی min_samples_split می‌پردازیم.

۳: ۴۴ درصد

۷: ۴۴ درصد

۱۵: ۳۲ درصد

حداکثر دقت این مدل، ۴۴ درصد است.

می‌توان نتیجه گرفت که جنگل تصادفی با اینکه از درخت تصمیم قوی‌تر است اما همچنان برای این دیتاست به اندازه کافی مناسب نیست.

Ada Boost

```
from sklearn.ensemble import AdaBoostClassifier

ada_boost = AdaBoostClassifier(n_estimators=100, learning_rate=1)

ada_boost.fit(train, train_y)

predictions = ada_boost.predict(test)

accuracy = accuracy_score(test_y, predictions)
print("Accuracy:", accuracy)
```

بررسی n_estimators :

۲۵: ۱۰ درصد

۵۰ : ۱۰ درصد

۱۰۰ : ۸ درصد

۲۰۰ : ۱۴ درصد

۳۰۰ : ۱۴ درصد

اگر اندازه این هایپرپارامتر را بیشتر کنیم، پیچیدگی مدل بیشتر می‌شود که مناسب نیست. با ۲۰۰ به بررسی نرخ یادگیری می‌پردازیم:

۰.۰۱ : ۲۹ درصد

۰.۱ : ۵۸ درصد

۱ : ۱۴ درصد

حداکثر دقت این مدل، ۵۸ درصد است. با وجود اینکه این مدل، از مدل قبل دقت بیشتری دارد، اما همچنان مناسب این دیتاست نیست.

Bayes Classifier

```
from sklearn.naive_bayes import GaussianNB

bayes = GaussianNB()

bayes.fit(train, train_y)

predictions = bayes.predict(test)

accuracy = accuracy_score(test_y, predictions)
print("Accuracy:", accuracy)
```

دقت این مدل برابر با ۶۲ درصد است. این مدل از مدل قبلی بهتر است اما همچنان مناسب دیتاست نیست.

SVM

```
from sklearn import svm

clf = svm.SVC(kernel='linear')
clf.fit(train, train_y)

predictions = clf.predict(test)

accuracy = accuracy_score(test_y, predictions)
print("Accuracy:", accuracy)
```

بررسی کرنل‌های مختلف این مدل:

linear : ۸۵ درصد

Rbf : ۴۷ درصد

Poly : ۲۹ درصد

در این مسئله، کرنل خطی از بقیه کرنل‌ها مناسب‌تر است.

این الگوریتم می‌تواند یکی از الگوریتم‌های مناسب برای این دیتاست درنظر گرفته شود.

Clustring

Dataframe

```
df = pd.read_csv("leaves.csv", header=None)
df
df.drop(0, inplace=True, axis=1)
df.drop(1, inplace=True, axis=1)
```

پیش از آنکه از دیتاست حاوی ویژگی‌های عکس‌ها استفاده کنیم، از دیتاست اولیه استفاده می‌کنیم.

preprocessing

Outlier Analysis

```
whisker_width = 10

for col in df.columns:
  Q1 = df[col].quantile(0.25)
  Q3 = df[col].quantile(0.75)
  IQR = Q3 - Q1
  low_bound = Q1 - whisker_width * IQR
  high_bound = Q3 + whisker_width * IQR
  for i in df.index:
    if df.loc[i, col] < low_bound or df.loc[i, col] > high_bound:
      df.drop(i, inplace=True)
```

در این بخش همانند بخش دسته‌بندی عمل می‌کنیم.

Normalization

```
scaler = MinMaxScaler()
scaler.fit_transform(df)
```

K Means

نخستین الگوریتم مورد استفاده برای مسائل خوشه‌بندی، این الگوریتم است.

```
from sklearn.cluster import KMeans

def k_means():
    model = KMeans(n_clusters=10)
    model.fit(df)
    labels = model.labels_

    score = normalized_mutual_info_score(nmi, labels)
    print(f'NMI: {score}')
    score = silhouette_score(df, labels)
    print(f'silhouette score: {score}')
    score = davies_bouldin_score(df, labels)
    print(f'Dunn score: {score}')

k_means()
```

بررسی معیارها نسبت به تعداد کلاسترها

n_clusters	Silhouette score (mean)	Dunn score (mean)	NMI
2	0.7	0.4	0.15
3	0.7	0.2	0.18
5	0.3	0.7	0.41
10	0.3	0.8	0.57
20	0.3	0.8	0.62
30	0.3	0.8	0.64

این الگوریتم حداکثر ۶۴ درصد مناسب است.

Agglomorative Clustring

این الگوریتم بر پایه فاصله‌ی نقاط از یکدیگر است.

```
from sklearn.cluster import AgglomerativeClustering

def agglomerative_clustering():
  model = AgglomerativeClustering(n_clusters=30)
  labels = model.fit_predict(df)

  score = normalized_mutual_info_score(nmi, labels)
  print(f'NMI: {score}')
  score = silhouette_score(df, labels)
  print(f'silhouette score: {score}')
  score = davies_bouldin_score(df, labels)
  print(f'Dunn score: {score}')


agglomerative_clustering()
```

بررسی معیارها نسبت به تعداد کلاسترها

n_clusters	Silhouette score	Dunn score	NMI
2	0.7	0.4	0.15
3	0.7	0.2	0.18
5	0.3	0.7	0.39
10	0.3	0.8	0.55
20	0.3	0.8	0.61
30	0.3	0.7	0.64

این الگوریتم در حد ۶۴ درصد مناسب است.

DBSCAN Clustring

این الگوریتم با استفاده از بررسی چگالی فضا به خوشه‌بندی می‌پردازد.

```
from sklearn.cluster import DBSCAN

def dbscan_clustering():
  model = DBSCAN(eps=0.5, min_samples=10)
  labels = model.fit_predict(df)
  n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

  score = normalized_mutual_info_score(nmi, labels)
  print(f'NMI: {score}')
  print(f'Number of clusters: {n_clusters}')
  score = silhouette_score(df, labels)
  print(f'silhouette score: {score}')
  score = davies_bouldin_score(df, labels)
  print(f'Dunn score: {score}')


dbscan_clustering()
```

با تغییر min_samples و eps، این الگوریتم تعداد کلاستر متفاوتی را ارائه می‌دهد که در ادامه به بررسی آن می‌پردازیم

min_samples	eps	n_clusters	Silhouette score (mean)	Dunn score (mean)	NMI
10	1	3	0.7	1.2	0.18
8	1	3	0.7	1.4	0.18
8	0.7	4	0.5	1.7	0.25
10	0.7	2	0.6	2.7	0.21
12	1	1	0.7	0.4	0.15

این الگوریتم، رفتار الگوریتم‌های قبلی در مورد افزایش و کاهش معیارها را ندارد. بنابراین می‌تواند الگوریتم مناسبی برای این دیتاست باشد. اما با این حال در معیار NMI مناسب نیست.

Gaussian Mixture

این الگوریتم فرض می‌کند که داده‌ها توزیع گوسی دارند و برای هر بخش، یک مدل گوسی فیت می‌کند. (پس پیش از اجرای این الگوریتم می‌بایست از Standar Scaler برای نرمال‌سازی داده‌ها استفاده کنیم.

```
from sklearn.mixture import GaussianMixture

def mixture_of_gaussians():
  model = GaussianMixture(n_components=10)
  labels = model.fit_predict(df)

  score = normalized_mutual_info_score(nmi, labels)
  print(f'NMI: {score}')
  score = silhouette_score(df, labels)
  print(f'silhouette score: {score}')
  score = davies_bouldin_score(df, labels)
  print(f'Dunn score: {score}')

mixture_of_gaussians()
```

با تغییر n_components به بررسی معیارها می‌پردازیم.

n_components	Silhouette score (mean)	Dunn score (mean)
10	0.2	1.1
8	0.3	0.99
12	0.3	0.9

این مدل در برابر معیار silhouette اصلا خوب عمل نمی‌کند و مناسب این دیتاست نیست.