# 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)

## 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)

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)

برسی 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.