پروژه Butterfly Image Classification

استاد علی امیری

مسئول پروژه خانم خلیلی

بهداد بهرام آبادیان

97463111

هشدار جهت کم شدن حجم عکس ها حذف شده است برای اجرای کد نیاز است عکس ها در پوشه dataset/train جایگذاری شوند.

هدف

در این پروژه تعدادی تصویر از انواع پروانه ها و همچنین نژاد هر یک از آنها در اختیار ما گذاشته شده است و هدف این است که با استفاده از مدل های مختلفی که برای classification وجود دارد ما بتوانیم پس از آموزش مدل خود اقدام به پیشبینی و تعیین نوع پروانه ها از روی تصاویر تست موجود کنیم.

که در ادامه به تشریح روش انجام این کار پرداخته و توضیحاتی در رابطه با کد پایتون نوشته شده برای این پروژه میپردازیم.

مدل Support Vector Machine(SVM)

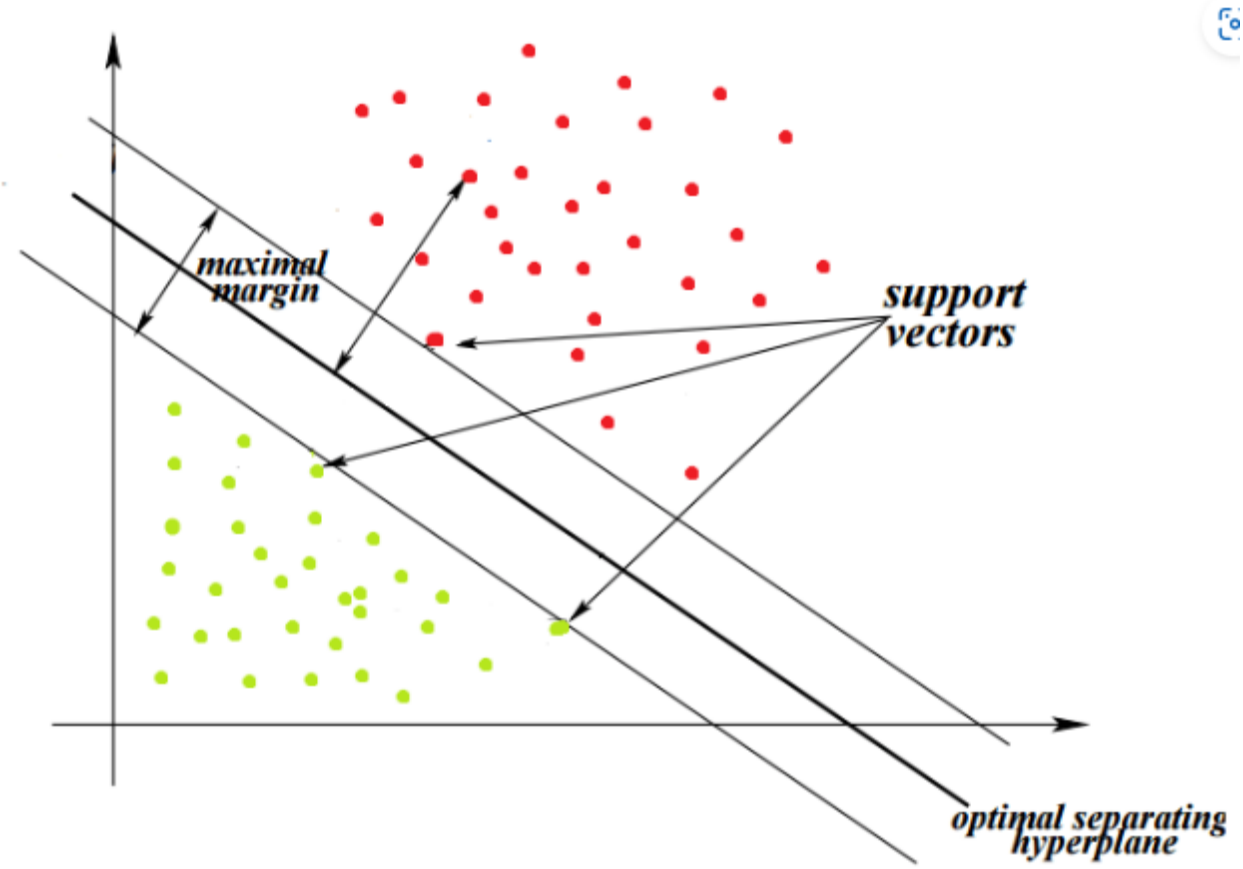
معرفی

ماشین بردار پشتیبان یک الگوریتم یادگیری ماشینی نظارت شده است که می تواند برای مسائل طبقه بندی و رگرسیون استفاده شود. برای تبدیل داده ها از تکنیکی به نام ترفند هسته پیروی می کند و بر اساس این تبدیل ها، مرز بهینه ای را بین خروجی های ممکن پیدا می کند.

به عبارت ساده، به تبدیل داده های بسیار پیچیده می پردازد تا بفهمد چگونه داده ها را بر اساس برچسب ها یا خروجی های تعریف شده جدا کند.

روش کار

به دنبال شناسایی ابر صفحه جدا کننده ای است که داده های آموزش را با حداکثر مارجین از هم جدا کند.



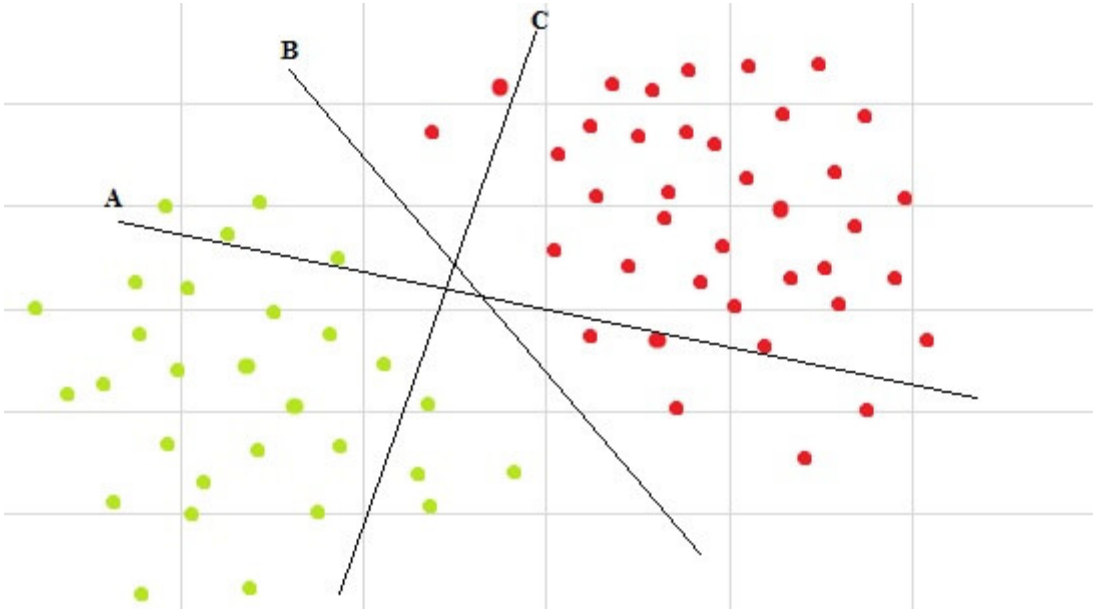
ابرصفحه جدا کننده

اصطلاحی است که بطور کل به جدا کننده دیتا های گفته میشود اما همان طور که در تصویر بالا نیز مشخص است در این نمونه داده ها توسط یک خط از یکدیگر جدا شده اند توضیح دقیق تر در ادامه ذکر شده است:

- در داده های یک بعدی جدا شدن توسط یک نقطه انجام میشود.
- در داده های دو بعدی جدا شدن توسط یک خط انجام میشود.
- در داده های سه بعدی جدا شدن توسط یک صفحه انجام میشود.

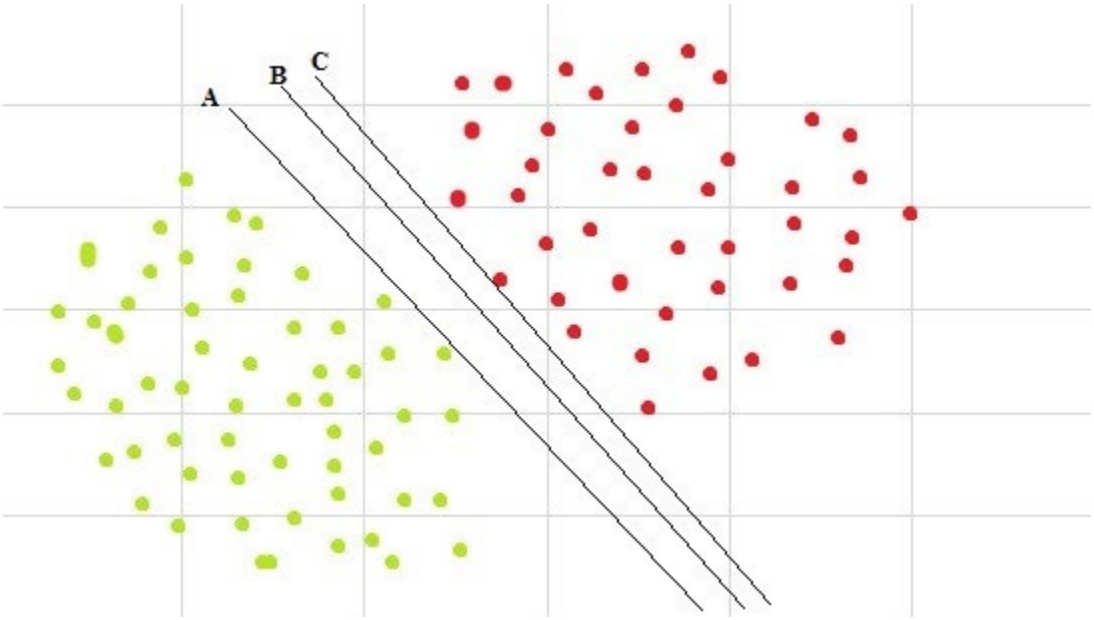
- در دادهای فراتر از سه بعد جدا شدن توسط یک ابرصفحه انجام میشود.

ابرصفحه چندگانه



امکان وجود چندین ابرصفحه جدا کننده وجود دارد اما ابر صفحه اپتیمال و مورد قبول ابرصفحه ای است که به بهترین شکل داده ها را از هم جدا کند.

همچنین امکان دارد چندین صفحه که به شکلی مناسب داده ها را جدا کنند مانند تصویر زیر وجود داشته باشد.



که در این جا ابرصفحه ای اپتیمال است که حداکثر مارجین را داشته باشد.

چگونگی انجام این کار ها

برای محاسبه واقعی موارد ذکر شده طبیعتا به محاسبات ریاضیاتی نیاز است که تشریح چگونگی انجام این امور بسیار طولانی و خارج از حوصله است.

برای اطلاعات تکمیلی میتوان از منابع زیر استفاده کرد:

Simple Tutorial on SVM and Parameter Tuning in Python and R - HackerEarth Blog

توضیحات مورد نیاز در رابطه با کد نوشته شده

```
#کد تا حد متوسط کامنت گذاری شده است توضیحات دقیقتر و تکمیل تر در فایل گزارش وجود دارد
import cv2
import numpy as np
import pandas as pd
import matplotlib as plt
from sklearn.preprocessing import LabelEncoder
import random

DataSetPath = 'Dataset/train/'                          #آدرس تصاویر
df_train = pd.read_csv('Dataset/Training_set.csv')      #خواندن اطلاعات train
NumOfImg=6499     #تعداد تصاویر که در اموزش شرکت میکنند
IMG_SIZE=40      #سایز تصاویر شرکت کننده
Img_Data_Train = []
Img_Data_Test = []
Label = []
Label_Train =[]
Label_Test = []
Ext_DATA_Train = []
Ext_LABEL = []
i=0
NumOfextImg = 0
NumOftrainImg = 0
NumOfXTest = 0
```

در ابتدا کتابخانه های لازم ایمپورت شده و متغییر های مورد نیاز در طی پروژه تعریف شده اند.

```
#قطعه کد تبدیل داده های رشته ای لیبل ها به عدد
le = LabelEncoder()
df_train['label'] = le.fit_transform(df_train['label'])
Lb = df_train['label']
Label = Lb[0:NumOfImg]
```

بدلیل اینکه مدل ها برای تحلیل داده ها وابسته به مقادیر عددی هستند و مقادیر لیبل ها یا همان نژاد پروانه ها بصورت رشته است، ما با استفاده از کد بالا که بطور خودکار به هر رشته یک عدد خاص اختصاص میدهد دیتا لیبل ها را به فرم مناسب برای مدل و تحلیل تبدیل میکنیم در این روش برای مثال اگر رشته ای به شکل زیر داشته باشیم

Red,blue,green,red,blue توسط این که به 1,2,3,1,2 تبدیل میشود که یک نماینده رنگ قرمز است و بقیه نیز به همین شکل.

پردازش های روی تصاویر

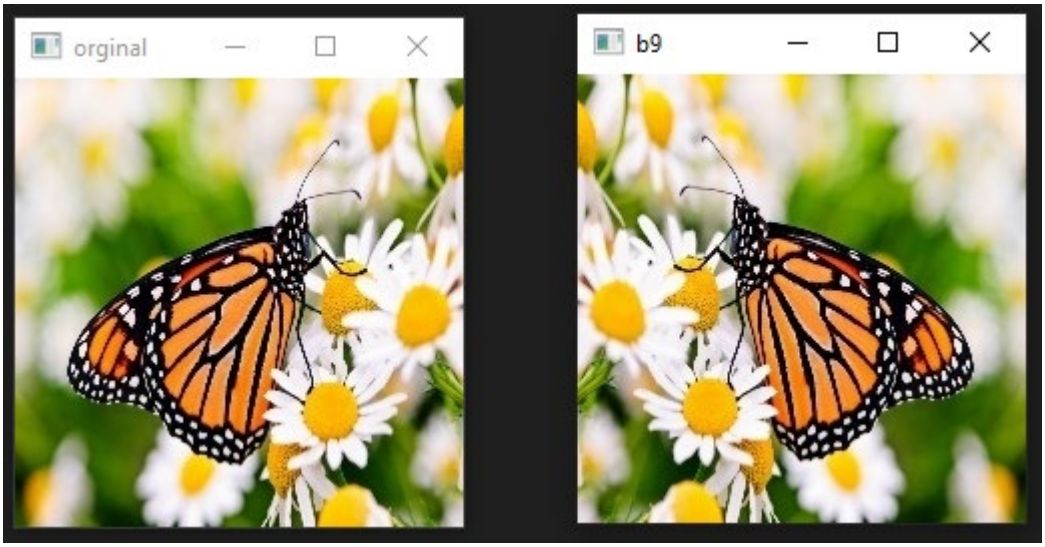
از آنجایی که تعداد تصاویر ما برای تحلیل محدود است ما میتوانیم با ایجاد تغییراتی برای روی تصاویر موجود، تصاویر جدیدی ایجاد کرده و میزان دیتا های خود به این طریق بیشتر کنیم و مدل خود را بهتر آموزش دهیم که راه های مختلفی برای این منظور وجود دارد.

قرینه کردن تصویر

در این روش توسط قطعه کد زیر تصویر را در جهت دلخواه قرینه میکنیم

```
#قطعه کد فلیپ کردن تصاویر برای ساخت و اضافه کردن تصاویر جدید از تصاویر موجود
def flip(img):
    img_flip = cv2.flip(img,1)
    return img_flip
```

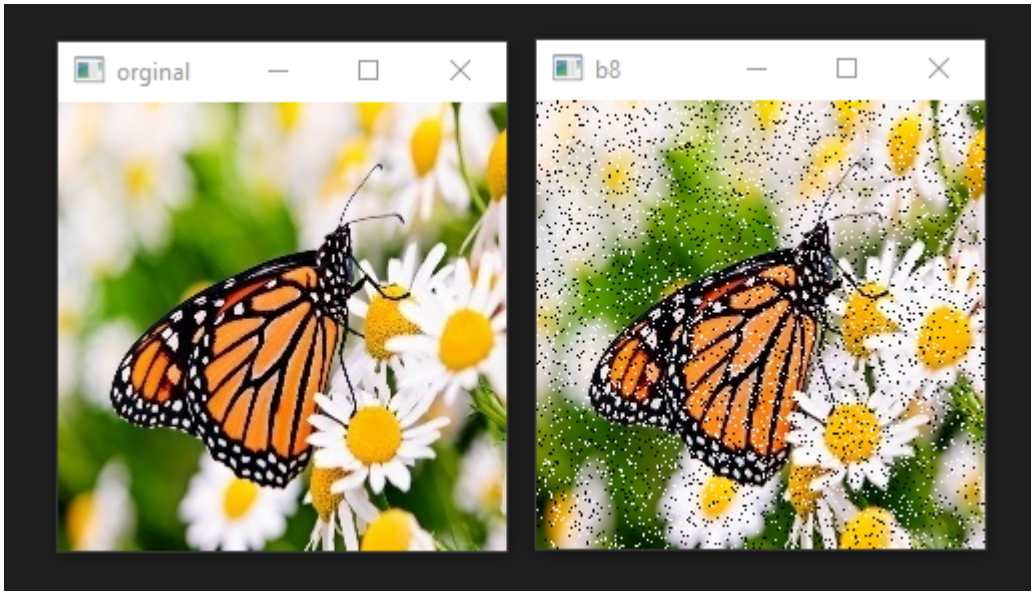
که در نتیجه این کد تصویر زیر بدست می آید



افزودن نویز به تصویر

```python
#قطعه کد اضافه کردن نویز به تصاویر
def noise(image,prob):
    output = np.zeros(image.shape,np.uint8)
    thres = 1 - prob
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            rdn = random.random()
            if rdn < prob:
                output[i][j] = 0
            elif rdn > thres:
                output[i][j] = 255
            else:
                output[i][j] = image[i][j]
    return output
```

توسط کد بالا بصورت رندوم و به مقدار قابل تنظیم توسط خودمان به تصویر نویز اضافه میکنیم که نتیجه در ادامه مشخص است



روش های دیگر

انواع روش های دیگری نیز برای این منظور وجود دارد که برای مثال میتوان به کراپ کردن تصاویر پرداخت تا برای مثال تصویر بالا حاشیه های اضافه کمتر و پروانه فضای بیشتری از تصویر را اشغال کند.

همچنین میتوان برای ساخت تصویر جدید از ترکیب چند روش استفاده کرد برای مثال هم تصویر قرینه شود و هم نویز به آن اضافه شود.

اصلاح رنگ و سایز

```python
#قطعه کد خواندن تصویر اصلاح رنگ و سایز
def img_preprocess(img,size,gray = False):

    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    if gray == True:
        img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
        new_img = cv2.resize(img_gray,(size,size))
    else:
        new_img = cv2.resize(img_rgb,(size,size))

    return new_img
```

توسط قطعه کد بالا تصویری که دریافت میکند در ابتدا به فرمت rgb تبدیل میشود در ادامه در صورت TRUE بودن ورودی gray تصویر سیاه سفید میشود و در ادامه سایز تصویر نیز با توجه به مقدار ورودی اصلاح میشود.

ساخت تصاویر جدید از تصاویر موجود

```python
#ساخت تصاویر جدید از تصاویر موجود برای افزایش مقدار دیتا های قابل تحلیل
def extra_data(img,prop,size,gray = False):

    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    if gray == True:
        img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
        new_img = cv2.resize(img_gray,(size,size))
    else:
        new_img = cv2.resize(img_rgb,(size,size))

    PFlipNoise = np.random.choice([0, 1, 2], p=[0.7, 0.2,0.1])   #ست

    if PFlipNoise==0:       #تنها فلیپ شدن عکس
        return flip(new_img)
    elif PFlipNoise==1:     #تنها اضافه شدن نویز به تصویر
        return noise(new_img,0.2)

    elif PFlipNoise==2:     #هم فلیپ و هم اضافه شدن نویز به تصویر
        T = flip(new_img)
        return noise(T,0.2)
```

این کد با دریافت تصویر ابتدا مانند بخش قبل به اصلاح رنگ و سایز تصویر ورودی میپردازد، در ادامه با توجه به احتمالات تعیین شده اقدام به انجام یکی از سه کار مشخص شده میکند

- در صورت 0 بودن که احتمال آن 70 درصد است صرفا تصویر را قرینه میکند.
- در صورت 1 بودن که احتمال آن 10 درصد است صرفا به تصویر نویز اضافه میکند.
- در صورت 2 بودن که احتمال آن 20 درصد است هم تصویر قرینه شده و هم به آن نویز اضافه میشود.

نکته: این نسبت ها با چند مرتبه تست بدست آمده است که بنظر نسبت مناسبی است و قرینه شدن تصاویر بیشترین تاثیر مثبت را دارد.

البته امکان وجود نسبت بهتری حتما وجود دارد که برای بدست آوردن تست های بسیار زیادی نیاز است.

پروسس نهایی بروی تصاویر

در این این بخش با استفاده از توابع که در بخش های قبل معرفی کردیم شروع به پردازش بروی تصاویر و اضافه کردن آنها به آرایه های مناسب کرده تا در بخش های بعد مدل را براساس این دیتا ها آموزش داده و تست کنیم.

نکته: در ابتدا من اول اقدام به اضافه کردن تصاویر اصلی کرده و بعد تصاویر جدید ساخته شده را نیز به آن اضافه میکردم و بعد در بخش های بعدی با قطعه کد زیر اقدام به جدا کردن 10 درصد برای تست و 90 درصد برای آموزش میکردم

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.1)
```

و به این روش موفق شدم درصد های بالا تا نزدیک 60 درصد بدست بیارم اما بعد متوجه شدم که دلیل این درصد بالا اینه که من بعد از ترکیب تصاویر اصلی با تصاویر ساخته شده توسط خودم اقدام به جدا کردن مقادیر آموزش و تست کرده که کاملا اشتباه است زیرا با احتمال زیادی عکسی که برای تست جدا شده مثلا به حالت قرینه در آموزش نیز وجود داشته و دلیل درصد بالا این موضوع است.

پس برای جلوگیری از این موضوع توسط قطعه کد

```python
def img_process(path,length,propTT,propED):
    global i
    global NumOfextImg
    global NumOftrainImg
    global NumOfXTest
    while(i<length):
        img_bgr = cv2.imread(path + df_train.iloc[i]['filename'])

        Prop_train_test = np.random.choice([0, 1], p=[1-propTT, propTT]) #احتمال حضور عکس در ترین یا در تست بودن
        if Prop_train_test==1:
            Img_Data_Train.append(img_preprocess(img_bgr,IMG_SIZE,False))
            Label_Train.append(Label[i])
            NumOftrainImg = NumOftrainImg+1
            Prop_extraData = np.random.choice([0, 1], p=[1-propED, propED]) #احتمال ساخته شدن عکس جدید
            if Prop_extraData==1:
                Ext_DATA_Train.append(extra_data(img_bgr,.9,IMG_SIZE,False))
                Ext_LABEL.append(Label[i])
                NumOfextImg = NumOfextImg+1
        else:
            Img_Data_Test.append(img_preprocess(img_bgr,IMG_SIZE,False))
            Label_Test.append(Label[i])
            NumOfXTest = NumOfXTest+1
        i =i+1
```

ابتدا تصاویر آموزش و تست را با نسبت مشخص جدا کرده و در ادامه تنها از تصاویر موجود در بخش آموزش اقدام به ساخت تصاویر جدید کردم و تصاویر تست به هیچ عنوان در بخش آموزش به صورت قرینه یا هر شکلی دیگری حضور ندارند. که بعد این اقدام درصد دقت بین 30 تا 40 متغیر است.

اقدامات نهایی بروی داده ها

```python
X=[]
y=[]

Img_Data_Data = np.concatenate((Img_Data_Train, Ext_DATA_Train)) #ادغام داده های تصاویر اصلی و تصاویر ساخته شده توسط ما
X = Img_Data_Data
y = Label_Train
y= np.concatenate((y, Ext_LABEL))   #ادغام لیبل تصاویر اصلی و تصاویر ساخته شده توسط ما
X = np.array(X).reshape(NumOfextImg + NumOftrainImg,-1)   #دو بعدی کردن آرایه های دریافتی برای اینکه مدل ان وی ام تنها با داده های دو بعدی کار میکند
print(X.shape)
#X = np.where(X > np.mean(X), 1.0, 0.0)
X = X/255.0     #تقسیم مقادیر آرایه ها بر 255 تا بازه اعداد در بین 0 تا 1 باشد
y = np.array(y)
print(y.shape)

X_test = Img_Data_Test  #تصاویر تست
X_test = np.array(X_test).reshape(NumOfXTest,-1)
X_test = X_test/255
y_test = Label_Test #لیبل های تست
y_test = np.array(y_test)
```

در این بخش اقدام به اتصال داده های تصاویر اصلی و تصاویر ساخته شده توسط خودمان کردیم و آرایه های چند بعدی را به 2 بعدی که مناسب کار با مدل است تبدیل کرده ایم

این اقدام در صورت نیاز برای مقادیر لیبل ها و داده های تست نیز در آخر صورت گرفته که تمامی داده های تصاویر آموزش، لیبل های آموزش، تصاویر تست، لیبل های تست در فرمت و ترتیب مناسب برای شروع آموزش مدل باشند.

تنظیم کردن مقادیر مدل SVM

```
svc = SVC(C=1.0, kernel='linear', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False,
 tol=0.001, cache_size=200, class_weight='balanced', verbose=False,
 max_iter=-1, decision_function_shape='ovo', break_ties=False, random_state=None)   #فراخانی مدل و مشخص کردن پارامتر های آن
```

در این بخش اقدام به تنظیم پارامتر های مدل میکنیم که به معرفی بعضی از آنها میپردازیم:

> **kernel : *{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'***

کرنل که مقادیر مختلفی میگیرد به چگونگی هسته الگوریتم مورد استفاده در آموزش اشاره دارد که انواع آن در تصویر بالا مشخص است که هر یک برای داده های متفاوتی مورد استفاده قرار میکیرد که براساس تست هایی من انجام دادم حالت linear بالا ترین دقت رو در حداقل داده های ما دارد.

بیشتر پارامتر های دیگر مثل degree و gamma برای روش نوع های دیگر هسته هستند برای مثال مدل poly و تغییر دادن آنها در روش انتخابی ما تاثیری نمیگذارد.

probability : *bool, default=False*

Whether to enable probability estimates. This must be enabled prior to calling `fit`, will slow down that method as it internally uses 5-fold cross-validation, and `predict_proba` may be inconsistent with `predict`. Read

این پارامتر نیز حداقل در داده های ما صرفا موجب افزایش مدت آموزش میشود و تاثیر در نتیجه ظاهرا ندارد.

decision_function_shape : *{'ovo', 'ovr'}, default='ovr'*

Whether to return a one-vs-rest ('ovr') decision function of shape (n_samples, n_classes) as all other classifiers, or the original one-vs-one ('ovo') decision function of libsvm which has shape (n_samples, n_classes * (n_classes - 1) / 2). However, note that internally, one-vs-one ('ovo') is always used as a multi-class strategy to train models; an ovr matrix is only constructed from the ovo matrix. The parameter is ignored for binary classification.

این پارامتر نیز با توجه به تست های من در صورت انتخاب حالت ovo حدود 5 درصد دقت بهتری میتوان بدست آورد.

نکته: بقیه پارامتر های نیز تغییر آنها از حالت دیفالت به مقادیر دیگر حداقل در تست های من تاثیر که بتوان تشخیص داد نداشت.

برای اطلاع دقیق از عملکرد هر یک از پارامتر ها میتوان از منبع زیر استفاده نمود:

[sklearn.svm.SVC — scikit-learn 1.3.0 documentation]

آموزش

```
print("Training... !")
svc.fit(X, y)       #اعمال مدل بر داده ها

y2 = svc.predict(X_test)        #پیشبینی کردن توسط مدل بر روی داده های تست

print("Accuracy: ",accuracy_score(y_test,y2))  #تعیین میزان دقت


result = pd.DataFrame({'original' : y_test,'predicted' : y2})   #نمایش پیشبینی مدل و مقدار واقعی
print(result)
```

در آخر مدل را بر داده ها اعمال کرده و همچنین اقدام به پیشبینی لیبل های داده های تست کرده و میزان دقت این پیشبینی را محاسبه کرده و نمایش میدهیم.

نتیجه

تصویر زیر نتیجه یک اجرا است که در آن تصاویر با رزولیشن 40 * 40 شرکت کرده اند و تمامی 6499 تصویر حضور دارند که از این مقدار 5828 تا از تصاویر اصلی در آموزش شرکت داشته اند و 5255 عکس توسط کد ساخته شده اند و 671 تصویر از 6499 در آموزش حضور نداشته و صرفا در بخش تست مورد استفاده قرار گرفته اند.

که با توجه با پارامترهایی که در بخش های قبل توضیح داده شده اند دقت 36 درصد بدست آمده است که البته در تست های مختلف میتواند بین 30 تا 44 درصد متغیر باشد.

همچنین بالا بردن رزولیشن میتواند در افزایش دقت تاثیر گذار باشد.

```
Size of image: 40
Orginal image in train: 5828
Extra image in train: 5255
Orginal image split for test: 671
img_process done
Train data is ready!
Training... !
Accuracy:  0.3651266766020864
     original  predicted
0          66         66
1          14         40
2          40         40
3          21         28
4          53          7
..        ...        ...
666        71         46
667        73         43
668        65         30
669        36         46
670         5          5
```

چالش ها و نکات

1- بدلیل اینکه در این تصاویر در تشخیص نوع پروانه رنگ میتواند تاثیر زیادی داشته باشد بنده از تصاویر رنگی برای آموزش استفاده کرده ام، البته امکان استفاده از تصاویر سیاه و سفید نیز در کد فراهم است و صرفا نیاز در توابع مشخص شده:

```
def img_preprocess(img,size,gray = False):
```

```
def extra_data(img,prop,size,gray = False):
```

مقدار Gray برابر true باشد تا تصاویر بصورت سیاه و سفید مورد بررسی قرار گیرند.

2- رزولیشن در تست ها برابر 40 است افزایش این مقدار ممکن است منجر به افزایش دقت شود اما بدلیل بسیار طولانی شدن مدت اجرا برنامه از این کار صرف نظر شده است.

3- احتمالا در صورت استفاده از مدل CNN شبکه عصبی بتوان نتایج بسیار بهتری از الگوریتم مورد استفاده svm بدست آورد.

4- آموزش توسط مدل XGBoost نیز انجام شده است که فایل آن نیز موجود است البته دقت این روش هم در حدود روش SVM است و فرق چندانی ندارد

لینک گیتهاب پروژه

lllBehdadlll/Butterfly-Image-Classification (github.com)

منابع

Image classification using SVM (92% accuracy) | Kaggle

Image Classification Using Machine Learning-Support Vector Machine(SVM) | by Vegi Shanmukh | Analytics Vidhya | Medium

sklearn.svm.SVC — scikit-learn 1.3.0 documentation

Boosting Image Classification Accuracy | by Mashrur Mahmud | intelligentmachines | Medium

Simple Tutorial on SVM and Parameter Tuning in Python and R - HackerEarth Blog

و ...