# pip install opencv-python

Collecting opencv-python

Downloading opencv\_python-4.6.0.66-cp36-abi3-win\_amd64.whl (35.6 MB)

Requirement already satisfied: numpy>=1.19.3 in c:\users\husey\anaconda3\lib\site-packages (from opencv-python) (1.21.5)

Installing collected packages: opencv-python

Successfully installed opencv-python-4.6.0.66

Note: you may need to restart the kernel to use updated packages.

import pandas as pd

import numpy as np

import cv2

from tqdm import tqdm

tqdm.pandas()

data\_train = pd.read\_csv("train.csv")

data\_test = pd.read\_csv("test.csv")

data\_sub = pd.read\_csv("sample\_submission.csv")

data\_image = "../Project/Patika\_Project2/Plant\_Pathology\_2020/images/"

data\_train.head()

image\_id healthy multiple\_diseases rust scab

0 Train\_0 0 0 0 1

1 Train\_1 0 1 0 0

2 Train\_2 1 0 0 0

3 Train\_3 0 0 1 0

4 Train\_4 1 0 0 0

data\_test.head()

image\_id

0 Test\_0

1 Test\_1

2 Test\_2

3 Test\_3

4 Test\_4

data\_sub.head()

image\_id healthy multiple\_diseases rust scab

0 Test\_0 0.25 0.25 0.25 0.25

1 Test\_1 0.25 0.25 0.25 0.25

2 Test\_2 0.25 0.25 0.25 0.25

3 Test\_3 0.25 0.25 0.25 0.25

4 Test\_4 0.25 0.25 0.25 0.25

epochs = 20

sample\_len = 100

def load\_image(image\_id):

file\_path = image\_id + ".jpg"

image = cv2.imread(data\_image + file\_path)

return cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

train\_images = data\_train["image\_id"][:sample\_len].progress\_apply(load\_image)

100%|████████████████████████████████████████████████████████████████████████████████| 100/100 [00:03<00:00, 26.61it/s]

import plotly.express as px

import plotly.graph\_objects as go

import plotly.figure\_factory as ff

from plotly.subplots import make\_subplots

import matplotlib.pyplot as plt

fig = px.imshow(cv2.resize(train\_images[4], (205, 136)))

fig.show()

İlk olarak bu veri seti görsel öğrenme sonucu içerik sunacak bir yapıdadır. Görselleri opencv ile okuyup gerekli bazı görsel kütüphanelerle bunların üzerinde işlem yapacağız.. Kaggle üzerindeki veri setinde toplam 800mb'lik 3 bin küsur görsel var. Eğitim amaçlı bunları indirdim ve kaggle üzerinde proje ile ilgilenen bireylerin yaptıklarından yola çıkıyorum.. Opencv bilgimin çok yeterli olmamasından mütevellit.

red\_values = [np.mean(train\_images[idx][:, :, 0]) for idx in range(len(train\_images))]

green\_values = [np.mean(train\_images[idx][:, :, 1]) for idx in range(len(train\_images))]

blue\_values = [np.mean(train\_images[idx][:, :, 2]) for idx in range(len(train\_images))]

values = [np.mean(train\_images[idx]) for idx in range(len(train\_images))]

Burada görsellerin belli noktalarındaki çizgilerinin ortalamasını aldık. Renklerle belirtme sebebimiz her rengin aslında bitkiyi tanımlayabilecek bölgeleri oluşturuyor olmasıdır. Kırmızı, yeşil ve mavi bölgelerin ortalamasını alıyoruz...

fig = ff.create\_distplot([red\_values], group\_labels=["R"], colors=["red"])

fig.update\_layout(showlegend=False, template="simple\_white")

fig.update\_layout(title\_text="Kırmızı kanal değerlerinin dağılımı")

fig.data[0].marker.line.color = 'rgb(0, 0, 0)'

fig.data[0].marker.line.width = 0.5

fig

fig = ff.create\_distplot([green\_values], group\_labels=["G"], colors=["green"])

fig.update\_layout(showlegend=False, template="simple\_white")

fig.update\_layout(title\_text="Yeşil Kanal Değerlerinin Dağılımı")

fig.data[0].marker.line.color = 'rgb(0, 0, 0)'

fig.data[0].marker.line.width = 0.5

fig

fig = ff.create\_distplot([blue\_values], group\_labels=["B"], colors=["blue"])

fig.update\_layout(showlegend=False, template="simple\_white")

fig.update\_layout(title\_text="Mavi Kanal Değerlerinin Dağılımı")

fig.data[0].marker.line.color = 'rgb(0, 0, 0)'

fig.data[0].marker.line.width = 0.5

fig

fig = go.Figure()

for idx, values in enumerate([red\_values, green\_values, blue\_values]):

if idx == 0:

color = "Red"

if idx == 1:

color = "Green"

if idx == 2:

color = "Blue"

fig.add\_trace(go.Box(x=[color]\*len(values), y=values, name=color, marker=dict(color=color.lower())))

fig.update\_layout(yaxis\_title="Ortalama Değer", xaxis\_title="Renk Kanalları",

title="Ortalama değerler ve Renk kanalları", template="plotly\_white")

def visualize\_leaves(cond=[0, 0, 0, 0], cond\_cols=["healthy"], is\_cond=True):

if not is\_cond:

cols, rows = 3, min([3, len(train\_images)//3])

fig, ax = plt.subplots(nrows=rows, ncols=cols, figsize=(30, rows\*20/3))

for col in range(cols):

for row in range(rows):

ax[row, col].imshow(train\_images.loc[train\_images.index[-row\*3-col-1]])

return None

cond\_0 = "healthy == {}".format(cond[0])

cond\_1 = "scab == {}".format(cond[1])

cond\_2 = "rust == {}".format(cond[2])

cond\_3 = "multiple\_diseases == {}".format(cond[3])

cond\_list = []

for col in cond\_cols:

if col == "healthy":

cond\_list.append(cond\_0)

if col == "scab":

cond\_list.append(cond\_1)

if col == "rust":

cond\_list.append(cond\_2)

if col == "multiple\_diseases":

cond\_list.append(cond\_3)

data = data\_train.loc[:99]

for cond in cond\_list:

data = data.query(cond)

images = train\_images.loc[list(data.index)]

cols, rows = 3, min([3, len(images)//3])

fig, ax = plt.subplots(nrows=rows, ncols=cols, figsize=(30, rows\*20/3))

for col in range(cols):

for row in range(rows):

ax[row, col].imshow(images.loc[images.index[row\*3+col]])

plt.show()

Healthy

visualize\_leaves(cond=[1, 0, 0, 0], cond\_cols=["healthy"])

Sağlık bakımından sıralanmış bitki görselleri. Yukarıda her bir ölçekte bu veri setindeki görselleri incelemek için bir fonksiyon kurduk. Healthy, scab, rust ve multiple gibi kategorilerce görseller içerisinden arama gerçekleştirecek...

Scab

visualize\_leaves(cond=[0, 1, 0, 0], cond\_cols=["scab"])

Burada da kabuk bakımından görseller sınandı ve dokuz adet görsel çıkardı...

Rust

visualize\_leaves(cond=[0, 0, 1, 0], cond\_cols=["rust"])

Burada da paslanmış veya hasara uğramış görseller üzerinden gözlem yapma imkanımız doğuyor..

Multiple Diseases

visualize\_leaves(cond=[0, 0, 0, 1], cond\_cols=["multiple\_diseases"])

Çoklu hasara uğramış bitki görselleri.

Şimdi de artık model oluşturma, metriklerle sınama gibi yapıları bu veri seti üzerinden kontrol edelim... Fakat veri sadece 0 ve 1'lerden oluşuyor. Dolayısıyla pek tatmin olacağımız sonuçlara ulaşamayabiliriz. Aslında bu numeric değerleri categorical değerlere çevirip ardından classifacation modelleme gerçekleştirebiliriz fakat pek değişim olacağını sanmıyorum.

df = data\_train.copy()

print(f" Veri setinin boyut sayısı: {df.ndim}\n",

f"Veri setinin boyut bilgisi: {df.shape}\n",

f"Veri setindeki toplam eleman sayısı: {df.size}\n")

Veri setinin boyut sayısı: 2

Veri setinin boyut bilgisi: (1821, 5)

Veri setindeki toplam eleman sayısı: 9105

df.info()

RangeIndex: 1821 entries, 0 to 1820

Data columns (total 5 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 image\_id 1821 non-null object

1 healthy 1821 non-null int64

2 multiple\_diseases 1821 non-null int64

3 rust 1821 non-null int64

4 scab 1821 non-null int64

dtypes: int64(4), object(1)

memory usage: 71.3+ KB

df.describe().T

count mean std min 25% 50% 75% max

healthy 1821.0 0.283361 0.450754 0.0 0.0 0.0 1.0 1.0

multiple\_diseases 1821.0 0.049973 0.217948 0.0 0.0 0.0 0.0 1.0

rust 1821.0 0.341571 0.474367 0.0 0.0 0.0 1.0 1.0

scab 1821.0 0.325096 0.468539 0.0 0.0 0.0 1.0 1.0

y\_df = df[["healthy"]]

df.drop(["image\_id","healthy"], axis = 1, inplace = True)

X\_df = df

y\_df.head()

healthy

0 0

1 0

2 1

3 0

4 1

X\_df.head()

multiple\_diseases rust scab

0 0 0 1

1 1 0 0

2 0 0 0

3 0 1 0

4 0 0 0

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_df, y\_df, test\_size = 0.30, random\_state = 42)

Random Forests - RF Modelleme:

Model ve Tahmin bölümü:

from sklearn.ensemble import RandomForestRegressor

rf\_model = RandomForestRegressor().fit(X\_train, y\_train.values.ravel())

y\_pred = rf\_model.predict(X\_test)

np.sqrt(mean\_squared\_error(y\_test, y\_pred))

0.0

Model Tunning bölümü:

from warnings import filterwarnings

filterwarnings('ignore')

rf\_params = {"max\_depth": list(range(1,10)),

"max\_features": [3, 5, 10, 15],

"n\_estimators": [100, 200, 300, 500]}

rf\_model = RandomForestRegressor(random\_state=42)

rf\_cv\_model = GridSearchCV(rf\_model, rf\_params, cv=10, n\_jobs=-1, verbose=2).fit(X\_train, np.ravel(y\_train,order='C'))

print(f"En iyi parametreler: {str(rf\_cv\_model.best\_params\_)}")

Fitting 10 folds for each of 144 candidates, totalling 1440 fits

En iyi parametreler: {'max\_depth': 3, 'max\_features': 3, 'n\_estimators': 100}

rf\_tuned = RandomForestRegressor(max\_depth=3, max\_features=3, n\_estimators=100)

rf\_tuned.fit(X\_train, np.ravel(y\_train, order = 'C'))

y\_pred = rf\_tuned.predict(X\_test)

np.sqrt(mean\_squared\_error(y\_test, y\_pred))

0.0

Bir model daha kuracağım eğer sonuç yine böyle çıkarsa veri setini bu şekilde tamamlayacağım..

Gradient Boosting Machines - GBM Model

Model ve Tahmin bölümü:

from sklearn.ensemble import GradientBoostingRegressor

gbm\_model = GradientBoostingRegressor().fit(X\_train, y\_train.values.ravel())

y\_pred = gbm\_model.predict(X\_test)

np.sqrt(mean\_squared\_error(y\_test, y\_pred))

1.1809963755282e-05

Model Tunning bölümü:

gbm\_params = {"learning\_rate": [0.001,0.01,0.1,0.2],

"max\_depth": [3,5,8,10,15],

"n\_estimators": [100,200,500,1000],

"subsample": [1,0.5,0.75]}

gbm = GradientBoostingRegressor()

gbm\_cv\_model = GridSearchCV(gbm, gbm\_params, cv = 10, n\_jobs=-1, verbose=2).fit(X\_train, y\_train.values.ravel())

print(f"En iyi parametrelerimiz: {str(gbm\_cv\_model.best\_params\_)}")

Fitting 10 folds for each of 240 candidates, totalling 2400 fits

En iyi parametrelerimiz: {'learning\_rate': 0.2, 'max\_depth': 3, 'n\_estimators': 100, 'subsample': 0.5}

gbm\_tuned = GradientBoostingRegressor(learning\_rate=0.2, max\_depth=3, n\_estimators=100, subsample=0.5)

gbm\_tuned.fit(X\_train, y\_train)

y\_pred = gbm\_tuned.predict(X\_test)

np.sqrt(mean\_squared\_error(y\_test, y\_pred))

1.2933908250306654e-08

Extreme Gradient Boosting - XGB Model

Model ve Tahmin bölümü:

# pip install xgboost

Collecting xgboostNote: you may need to restart the kernel to use updated packages.

Downloading xgboost-1.7.0.post0-py3-none-win\_amd64.whl (89.1 MB)

Requirement already satisfied: numpy in c:\users\husey\anaconda3\lib\site-packages (from xgboost) (1.21.5)

Requirement already satisfied: scipy in c:\users\husey\anaconda3\lib\site-packages (from xgboost) (1.7.3)

Installing collected packages: xgboost

Successfully installed xgboost-1.7.0

from xgboost import XGBRFRegressor

xgb = XGBRFRegressor().fit(X\_train, y\_train.values.ravel())

y\_pred = xgb.predict(X\_test)

np.sqrt(mean\_squared\_error(y\_test,y\_pred))

0.07909571259136784

ilkel parametrelere dayalı bir şekilde yaptığımız bu karşılaştırma bile iyi bir sonuç verdi..

Model Tunning bölümü:

# ?xgb Değer atayacağımız parametrelerini gözlemledik..

xgb\_grid = {"colsample\_bytree": [0.1, 0.2, 0.3, 0.4, 0.5],

"n\_estimators": [100, 200, 500],

"max\_depth": [2,3,4,5,6],

"learning\_rate": [0.1,0.01,0.5]}

xgb\_model = XGBRFRegressor()

xgb\_cv\_model = GridSearchCV(xgb\_model, xgb\_grid, cv=10, n\_jobs=-1, verbose=2).fit(X\_train, y\_train.values.ravel())

print(f"En iyi parametrelerimiz: {str(xgb\_cv\_model.best\_params\_)}")

Fitting 10 folds for each of 225 candidates, totalling 2250 fits

En iyi parametrelerimiz: {'colsample\_bytree': 0.1, 'learning\_rate': 0.5, 'max\_depth': 2, 'n\_estimators': 500}

xgb\_tuned = XGBRFRegressor(colsample\_bytree = 0.1, learning\_rate=0.5, max\_depth = 2, n\_estimators = 500)

xgb\_tuned.fit(X\_train, y\_train.values.ravel())

y\_pred = xgb\_tuned.predict(X\_test)

np.sqrt(mean\_squared\_error(y\_test, y\_pred))

0.4305145660275954

Şimdi mean squarred error ile birlikte diğer metrikleri de bu modeler üzerinde gözlemleyelim..

rf\_pre = rf\_tuned.predict(X\_test)

gbm\_pre = gbm\_tuned.predict(X\_test)

xgb\_pre = xgb\_tuned.predict(X\_test)

predict = [rf\_pre, gbm\_pre, xgb\_pre]

algoritma\_adlari = ["Random Forests Regresyon", "Gradient Boosting Regresyon", "Extreme Gradient Regresyon"]

def metrics(y\_pred):

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

data = [mae, mse, rmse, r2]

return data

seriler = []

metric\_s = ["Mean Absolute Error", "Mean squared Error", "Root Mean Squared Error", "R2"]

for i in predict:

data = metrics(i)

seriler.append(data)

df\_df = pd.DataFrame(data=seriler, index=algoritma\_adlari, columns = metric\_s)

pd.set\_option('display.colheader\_justify', 'center')

print(df\_df.to\_string())

Mean Absolute Error Mean squared Error Root Mean Squared Error R2

Random Forests Regresyon 0.000000e+00 0.000000e+00 0.000000e+00 1.000000

Gradient Boosting Regresyon 1.165120e-08 1.672860e-16 1.293391e-08 1.000000

Extreme Gradient Regresyon 4.228802e-01 1.853428e-01 4.305146e-01 0.060892