Final Project - Classification for Diseased Cotton Plants

Link to the raw dataset: https://www.kaggle.com/janmejaybhoi/cotton-disease-dataset

On the raw dataset, I have combined the data from the three folders (test, train and val) into one folder and I have deleted the data for "diseased cotton plants" and "fresh cotton plants". So the modified data has two folders one with images of diseased cotton leaves and the other with images of fresh cotton leaves.

Link to the modified dataset: https://www.dropbox.com/sh/16moe0036fohpi3/AAD4KAlBzWGWbE3PPnHSBvBUa?dl=0

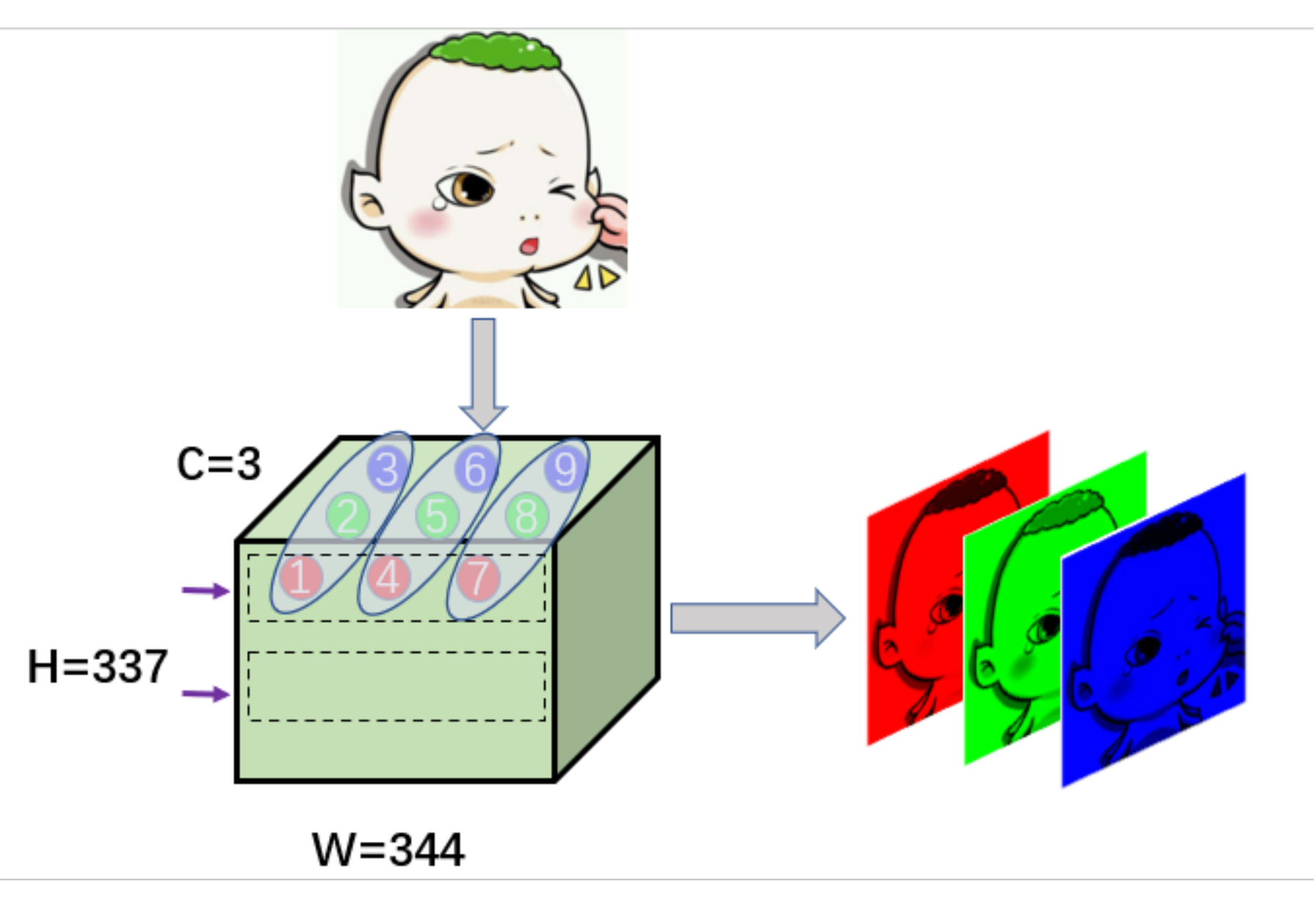
We start by importing all the necessary modules.

In [1]:
```
%matplotlib inline

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import os
import joblib
import skimage
import time

from skimage.io import imread
from skimage.transform import resize
from skimage.feature import hog
from skimage.transform import rescale

from collections import Counter

from sklearn.model_selection import train_test_split
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler, Normalizer

from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

from sklearn.metrics import accuracy_score, confusion_matrix
```

Data Ingestion

Then we load the data into the notebook. This process takes some time because there are a lot of images.

The imread() function used below converts an image into a 3D numpy array. For each and every pixel on the image the RGB values are stored creating a structure similar to one show in the picture below.



Note: ".DS_Store" is a file which macOS automatically adds to a folder to store metadata. If you are using a Windows computer this file should probably not exist.

In [2]:
```
data_path = r"/Users/thoughtworks/Dropbox/My Mac (Keshavs-MacBook-Pro.local)/Documents/edX/Python for Data Science/Final Project/data"

data = dict()
data["label"] = []
data["data"] = []

folders = ["diseased", "fresh"]

for folder_name in folders:
    try:
        print(folder_name)
        new_path = os.path.join(data_path, folder_name)

        for file in os.listdir(new_path):
            if file != ".DS_Store":
                im = imread(os.path.join(new_path, file)) # converts the image to a 3D Numpy array which stores R, G and B values for each pixel
                im = resize(im, (100, 100)) # resizing the image to a uniform smaller file size to quickly manipulate image data
                data["label"].append(folder_name)
                data["data"].append(im)
    except NotADirectoryError: # error checking in case other miscellaneous files like .DS_Store are present
        pass
```

```
diseased
fresh
```

Check the number of samples in the data and the shape of the first image.

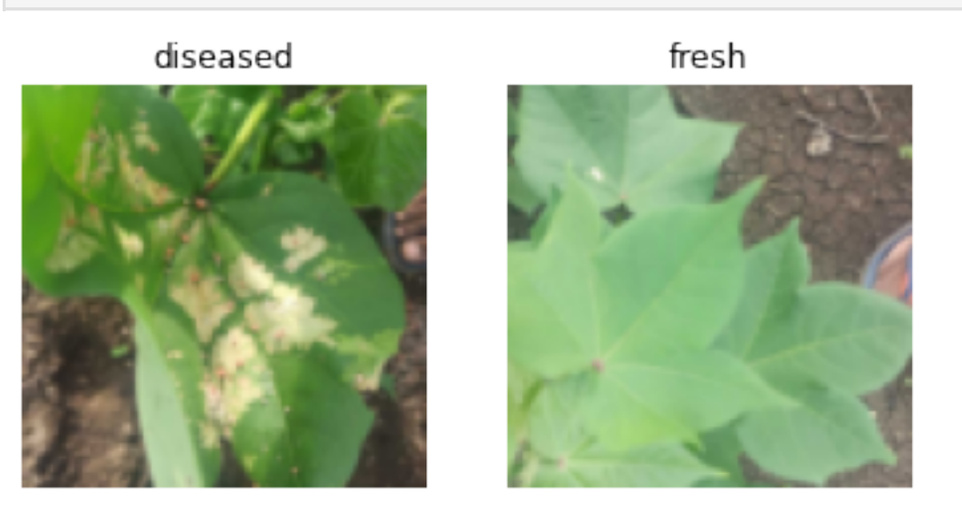
In [3]:
```
print('number of samples: ', len(data['data']))
print('image shape: ', data['data'][0].shape)

Counter(data['label'])
```

```
number of samples:  857
image shape:  (100, 100, 3)
```

Out[3]:
```
Counter({'diseased': 346, 'fresh': 511})
```

Display one random image of a diseased leaf and one random image of a fresh leaf.

In [4]:
```
labels = {'diseased':0, 'fresh':500}

fig, axes = plt.subplots(1, 2)

for a, key in zip(axes, list(labels.keys())):
    a.imshow(data['data'][labels[key]])
    a.axis('off')
    a.set_title(key)
```



Data Processing

Using the formula y = f(x) where the function f does the classification, we split the data into these two sets. Then, using train_test_split() from sklearn we split the train and test data in the ratio 4:1.

In [5]:
```
X = np.array(data['data'])
y = np.array(data['label'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=42)
```

X_train, X_train, X_test, y_train, y_test are all arrays of 3D arrays (representing each image). The classifier will not be able to ingest them and train the model. It can only take an array of 1D arrays. We use the following three systems chronologically to process each image.

RGB to Gray - This converts the RGB image to grayscale eliminating the 3 layers of Red, Green and Blue and thus converting the array to a 2D array. The 2D array now stores single values for each pixel instead of an array of 3 values.

Hog - HOG stands for Histogram Oriented Gradients. It is used to make the image simpler and reduces the shape of the array further down to a 1D array. If you want to know more you can read this article: https://learnopencv.com/histogram-of-oriented-gradients/.

Scaling - We then scale this 1D array to normalise the data.

In [6]:
```
class HogTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, y=None, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(3, 3), block_norm='L2-Hys'):
        self.y = y
        self.orientations = orientations
        self.pixels_per_cell = pixels_per_cell
        self.cells_per_block = cells_per_block
        self.block_norm = block_norm

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):

        def local_hog(X):
            return hog(X,
                       orientations=self.orientations,
                       pixels_per_cell=self.pixels_per_cell,
                       cells_per_block=self.cells_per_block,
                       block_norm=self.block_norm)

        try:
            return np.array([local_hog(img) for img in X])
        except:
            return np.array([local_hog(img) for img in X])
```

In [7]:
```
hogify = HogTransformer(
    pixels_per_cell=(14, 14),
    cells_per_block=(2,2),
    orientations=9,
    block_norm='L2-Hys'
)
scalify = StandardScaler()

X_train_gray = np.array([skimage.color.rgb2gray(img) for img in X_train])
X_train_hog = hogify.fit_transform(X_train_gray)
X_train_prepared = scalify.fit_transform(X_train_hog)

print(X_train.shape)
print(X_train_gray.shape)
print(X_train_hog.shape)
print(X_train_prepared.shape)
```

```
(685, 100, 100, 3)
(685, 100, 100)
(685, 1296)
(685, 1296)
```

We also similarly prep the test data.

In [8]:
```
X_test_gray = np.array([skimage.color.rgb2gray(img) for img in X_test])
X_test_hog = hogify.transform(X_test_gray)
X_test_prepared = scalify.transform(X_test_hog)
```

Classification Using 11 Different Classifiers

In [9]:
```
names = ["Decision Tree", "SGD", "Gaussian", "KNeighbours", "Naive Bayes", "Linear SVM", "RBF SVM", "Random Forest", "AdaBoost", "MLP", "Discriminant Analysis"]
accuracy_scores = []
execution_time = []
```

In [10]:
```
start_time = time.time()

decision_tree = DecisionTreeClassifier(max_leaf_nodes=30, random_state=42)
decision_tree.fit(X_train_prepared, y_train)
y_pred = decision_tree.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = y_pred) * 100)
execution_time.append(time.time() - start_time)
```

In [11]:
```
start_time = time.time()

sgd = SGDClassifier(random_state=42, max_iter=1000, tol=1e-3)
sgd.fit(X_train_prepared, y_train)
y_pred = sgd.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = y_pred) * 100)
execution_time.append(time.time() - start_time)
```

In [12]:
```
start_time = time.time()

gaussian = GaussianProcessClassifier(1.0 * RBF(1.0))
gaussian.fit(X_train_prepared, y_train)
y_pred = gaussian.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = y_pred) * 100)
execution_time.append(time.time() - start_time )
```

In [13]:
```
start_time = time.time()

kn = KNeighborsClassifier(3)
kn.fit(X_train_prepared, y_train)
y_pred = kn.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = y_pred) * 100)
execution_time.append(time.time() - start_time)
```

In [14]:
```
start_time = time.time()

naive_bayes = GaussianNB()
naive_bayes.fit(X_train_prepared, y_train)
y_pred = naive_bayes.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = y_pred) * 100)
execution_time.append(time.time() - start_time)
```

In [15]:
```
start_time = time.time()

svc_linear = SVC(kernel="linear", C=0.025)
svc_linear.fit(X_train_prepared, y_train)
y_pred = svc_linear.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = y_pred) * 100)
execution_time.append(time.time() - start_time)
```

In [16]:
```
start_time = time.time()

rbf_svc = SVC(gamma=2, C=1)
rbf_svc.fit(X_train_prepared, y_train)
y_pred = rbf_svc.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = y_pred) * 100)
execution_time.append(time.time() - start_time)
```

In [17]:
```
start_time = time.time()

random_forest = RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1)
random_forest.fit(X_train_prepared, y_train)
y_pred = random_forest.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = y_pred) * 100)
execution_time.append(time.time() - start_time)
```

In [18]:
```
start_time = time.time()

adaboost = AdaBoostClassifier()
adaboost.fit(X_train_prepared, y_train)
y_pred = adaboost.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = y_pred) * 100)
execution_time.append(time.time() - start_time)
```

In [19]:
```
start_time = time.time()

mlp = MLPClassifier(alpha=1, max_iter=1000)
mlp.fit(X_train_prepared, y_train)
mlp_y_pred = mlp.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = mlp_y_pred) * 100)
execution_time.append(time.time() - start_time)
```

In [20]:
```
start_time = time.time()

qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train_prepared, y_train)
y_pred = qda.predict(X_test_prepared)

accuracy_scores.append(accuracy_score(y_true = y_test, y_pred = y_pred) * 100)
execution_time.append(time.time() - start_time)
```

```
/Users/thoughtworks/opt/anaconda3/lib/python3.8/site-packages/sklearn/discriminant_analysis.py:808: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
```

Creating a DataFrame to find out which classifier is the best suited for this project.

In [21]:
```
classifiers = pd.DataFrame(index = names, columns = ["Accuracy Score", "Execution Time"])
classifiers["Accuracy Score"] = accuracy_scores
classifiers["Execution Time"] = execution_time
classifiers
```

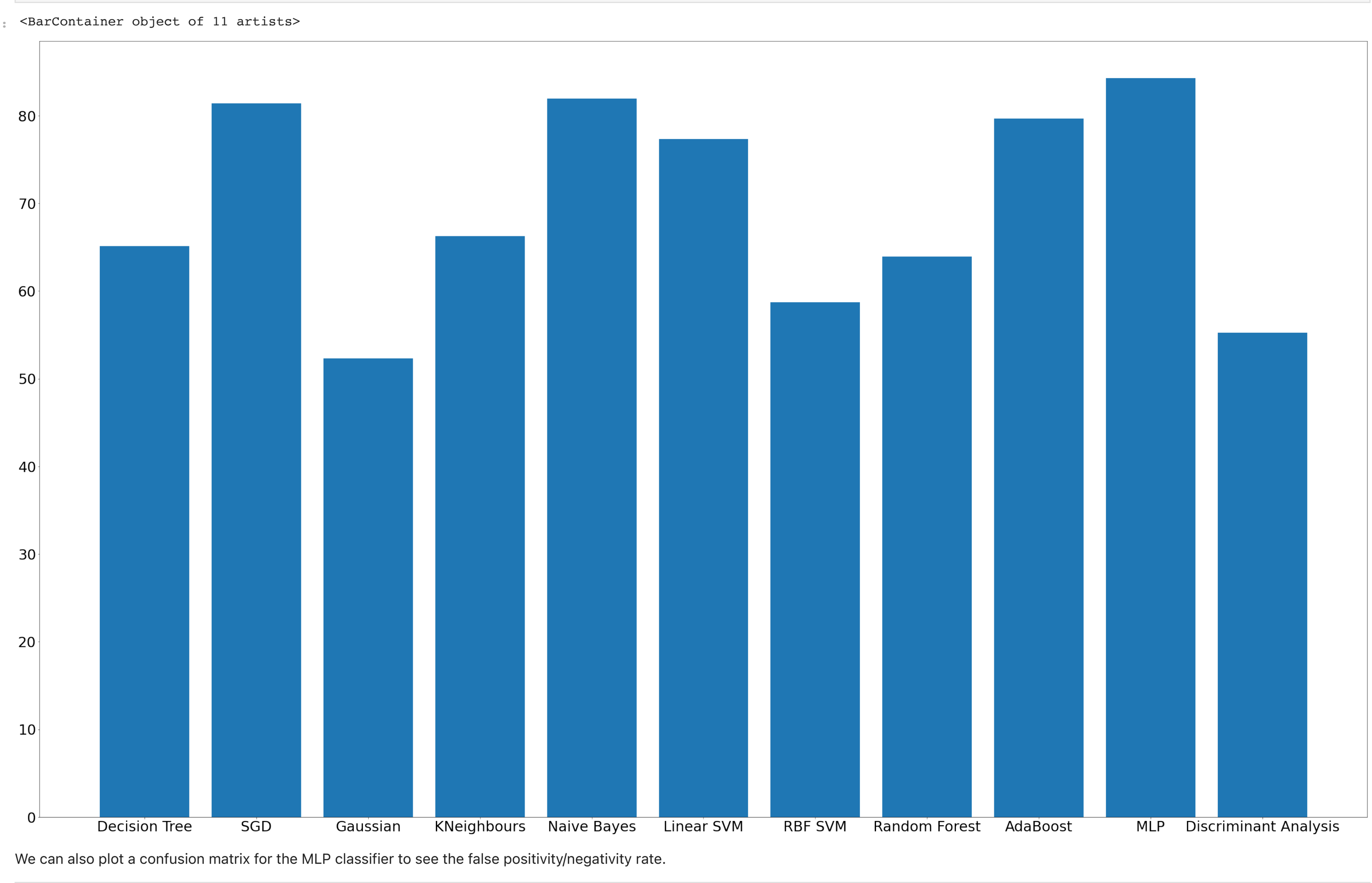
Out[21]:

	Accuracy Score	Execution Time
Decision Tree	65.116279	0.384982
SGD	81.395349	0.049020
Gaussian	52.325581	5.266633
KNeighbours	66.279070	0.016497
Naive Bayes	81.976744	0.014772
Linear SVM	77.325581	0.141030
RBF SVM	58.720930	0.297794
Random Forest	63.953488	0.019509
AdaBoost	79.651163	3.661435
MLP	84.302326	2.281766
Discriminant Analysis	55.232558	0.151461

From here, we see that the best classifier is MLP. It has the highest accuracy and has a moderate execution time.

In [22]:
```
plt.figure(figsize=(50, 30))
plt.xticks(fontsize=30)
plt.yticks(fontsize=30)
plt.bar(classifiers.index, classifiers["Accuracy Score"])
```

Out[22]:
```
<BarContainer object of 11 artists>
```



We can also plot a confusion matrix for the MLP classifier to see the false positivity/negativity rate.

In [23]:
```
matrix = confusion_matrix(y_test, mlp_y_pred, labels=["fresh", "diseased"])
df = pd.DataFrame(matrix, columns=["fresh", "diseased"], index=["fresh", "diseased"])
df.columns.name = "prediction"
df
```

Out[23]:

prediction	fresh	diseased
fresh	87	14
diseased	13	58

The left/main/leading diagonal of the matrix shows very high values signifying that the number of correct predictions is high and the right diagonal shows very low values signifying that the number of wrong predictions is low.

In [24]:
```
plt.imshow(matrix)
plt.xticks([])
plt.yticks([])
plt.colorbar();
```