Introduction to artificial intelligence
Application for Comparison of some classifiers in Python using Fruits dataset

Assyl Salah
Elif Ozdemir
Szymon Majorek

Supervised by: Jerzy Balicki

May 20, 2020

Contents

Chapter 1

Responsibilities of team members

Name	Tasks
Elif Ozdemir	1.Visualization 2.Description of some existing solutions 3.Conclusion 4.Presentation preparation
Szymon Majorek	1.Implementation of solution 2.The description of preferred solution 3.Organization of datasets 4.Presentation preparation
Assyl Salah	1.Problem description 2.Problem analysis 3.GUI 4.Presentation preparation

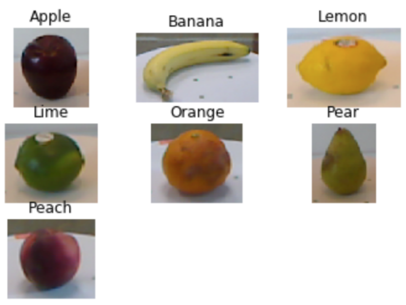
Chapter 2

Problem description

Nowadays precise classification of different kinds of fruit species, or food in general, is an important topic. It is not only a relevant topic in the field of academic research, but also in industrial applications. Many useful applications can be built based on such a classification system, for example it can be used in supermarkets to help cashiers. Cashiers need to identify not only the species of the fruit bought by the customer, but also its variety to determine the correct price. Such classification-based applications can identify the species purchased by the customer and match it automatically with the correct price. However, fruit classification based on computer vision remains difficult for the following reasons.

1. Shape, color, and texture similarity among numerous species of fruits, like the citrus genus, that contains oranges and grapefruits
2. Extremely high variation in a single class, which depends on the phase of fruit maturity, and the form in which the fruit is presented (e.g., fruits inside plastic bags, sliced on dishes, or unpicked).

Thus, we want to see how well can artificial intelligence complete the task of classifying them and which machine learning models of Scikit-learn library are the best in different comparison criteria.



Chapter 3

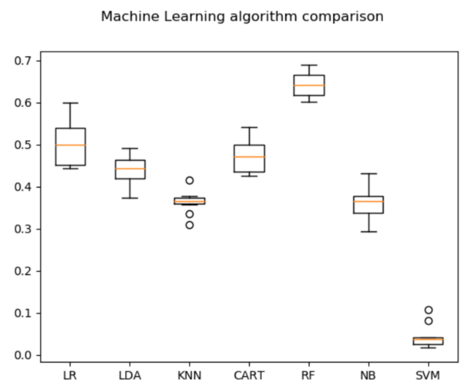
Problem analysis

In this project we compare Python Scikit-learn Classifiers depending on different criteria like accuracy, precision, mean absolute error, speed. The chosen Machine Learning models are the following:

- Logistic Regression
- Linear Discriminant Analysis
- K-Nearest Neighbors
- Decision Tree
- Random Forests
- Gaussian Naive Bayes
- Support Vector Machine
- Quadratic Discriminant Analysis
- Ada Boost classifier

The data is gathered from fruit images(28 thousand/44 species) by means of image processing. We extract global features like texture, color, shape and one local feature which is local binary pattern.
After training and testing we get K-Fold Cross Validation results for all machine learning models. The data split is done by the train_test_split method from sklearn.model_selection.

The following is an example of comparison chart of classifier accuracies.



Chapter 4

Description of some existing solutions

The following page provides an example usage and comparison of different Python machine learning models on flower dataset. `https://gogul.dev/software/image-classification-python`
The above mentioned solution builds an intelligent system that's trained with massive dataset of flower/plant images The system predicts the label/class of the flower/plant using Computer Vision techniques and Machine Learning algorithms. It uses a simpler approach to produce a baseline accuracy for the problem. It uses the following three global feature descriptors:

- Color Histogram that quantifies color of the flower.
- Hu Moments that quantifies shape of the flower.
- Haralick Texture that quantifies texture of the flower.

The next step is extracting and gathering the features and labels. After which the solution trains and tests machine learning models using scikit-learn library. It uses train_test_split function provided by scikit-learn to split our training dataset into train_data and test_data. By this way, we train the models with the train_data and test the trained model with the unseen test_data. The split size is decided by the test_size parameter. It uses a validation technique called K-Fold Cross Validation. Finally, it chooses the best classifier depending on the results.
At the end of the article it suggests methods to improve the accuracy of classifiers. For example, gathering more data (in this case images) is a good step to get better results. And since image processing is an important part of this and our project, it mentions that extracting local features is also a very powerful tool to enhance the accuracy.

The next page provides a very simple overview of the classifiers. `https://machinelearningmastery.com/compare-machine-learning-algorithms-python-scikit-learn/` The logic behind the algorithm is the same with the previous solution.

Chapter 5

The description of preferred solution

Our solution consists of two steps, the first one is extracting features and saving them to files, whereas the second one is reading those extracted features from file, processing them and then saving obtained results to file once again, so that they can be presented in nicer, more readable manner. By processing here I mean creating 9 different machine learning models from scikit-learn library, and then training as well as testing them.

The features that we consider in feature extraction are

- Shapes
- Haralick texture
- Colour histogram
- Local binary patterns

For shapes and colour histogram we use OpenCV library, for haralick texture - mahotas library and for local binary patterns - numpy and OpenCV. After extracting all of those features indvidually, we stack them together and save into a file so that we don't put everything into one script

The second step is the actual processing of the data. For that we first load extracted features from file using h5py library. After that we create our machine learning models from scikit-learn library. The selected models that we are interested in are the following:

- Random Forest Classifier
- Support Vector Machine
- KN neighbors Classifier
- Logistic Regression
- Linear Discriminant Analysis
- Gaussian Naive Bayes
- Decision Tree Classifier
- Ada Boost Classifier
- Quadratic Discriminant Analysis

Having done that, we use train_test_split on the dataset, the split it into training and testing parts and then we loop through all of our models, for each of them doing cross validation in the simplest possible manner, via using function from the same library as before. For each of the models we save the data that we are interested in, which is:

- Accuracy
- Mean squared error
- Fit time
- Score time

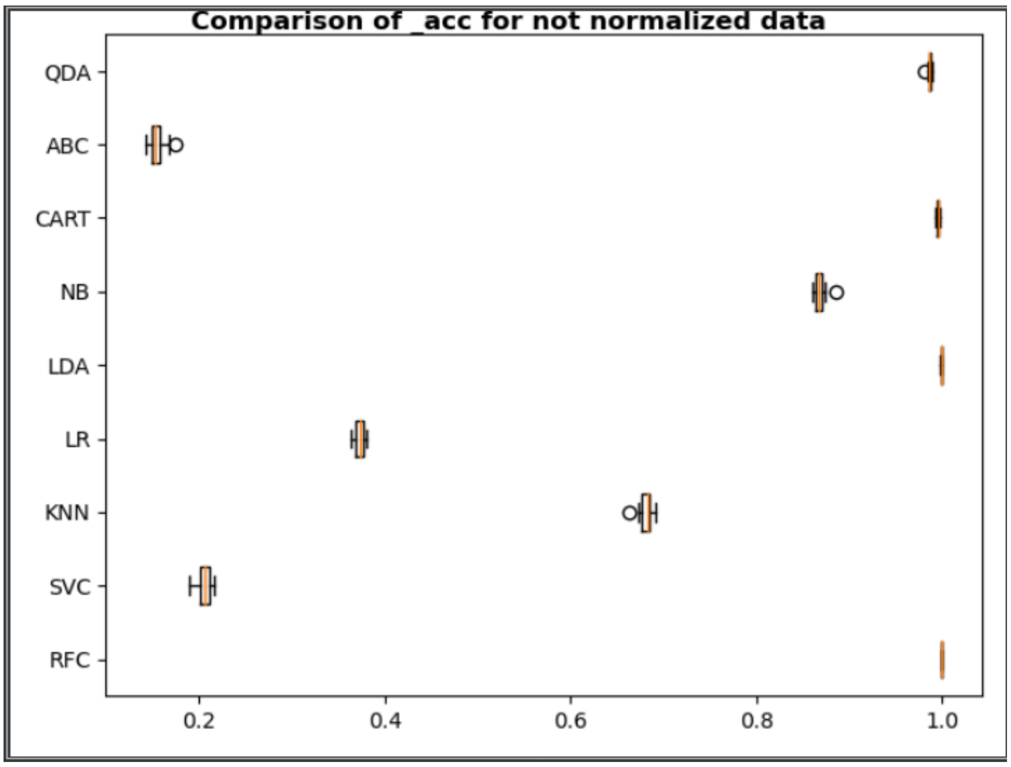
Having done all of that, we can proceed to the analysis, as well as visualization of the obtained results, comparing machine learning models which we have used to recognize different fruits.

Chapter 6

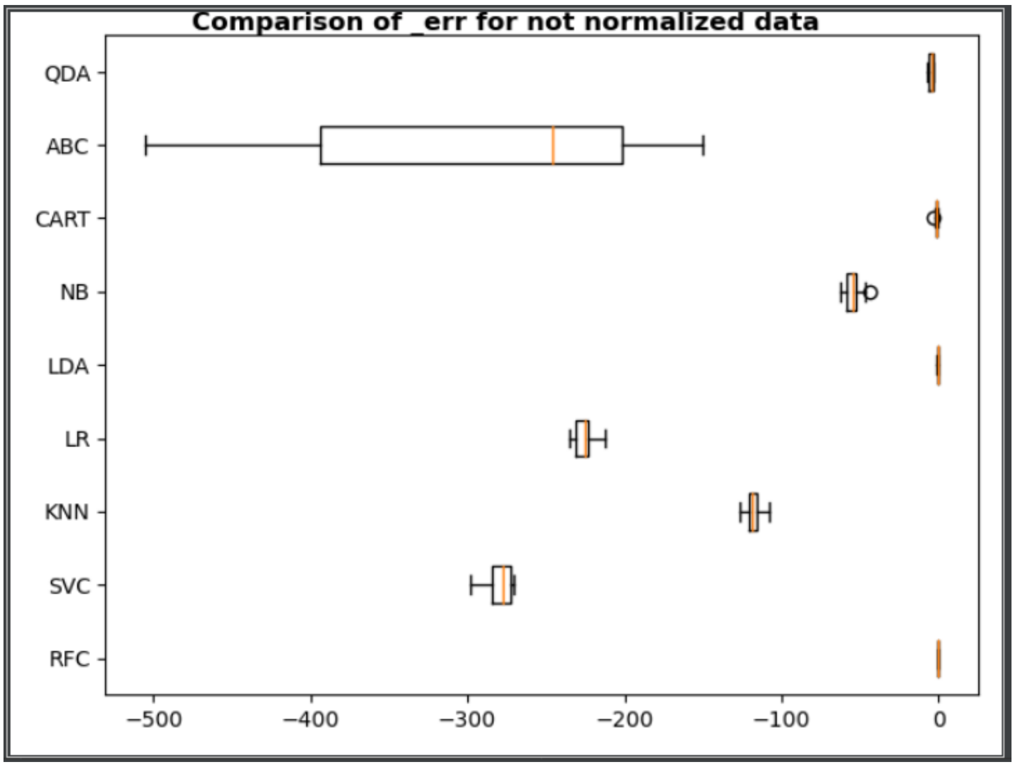
Results

1 Non normalized data set

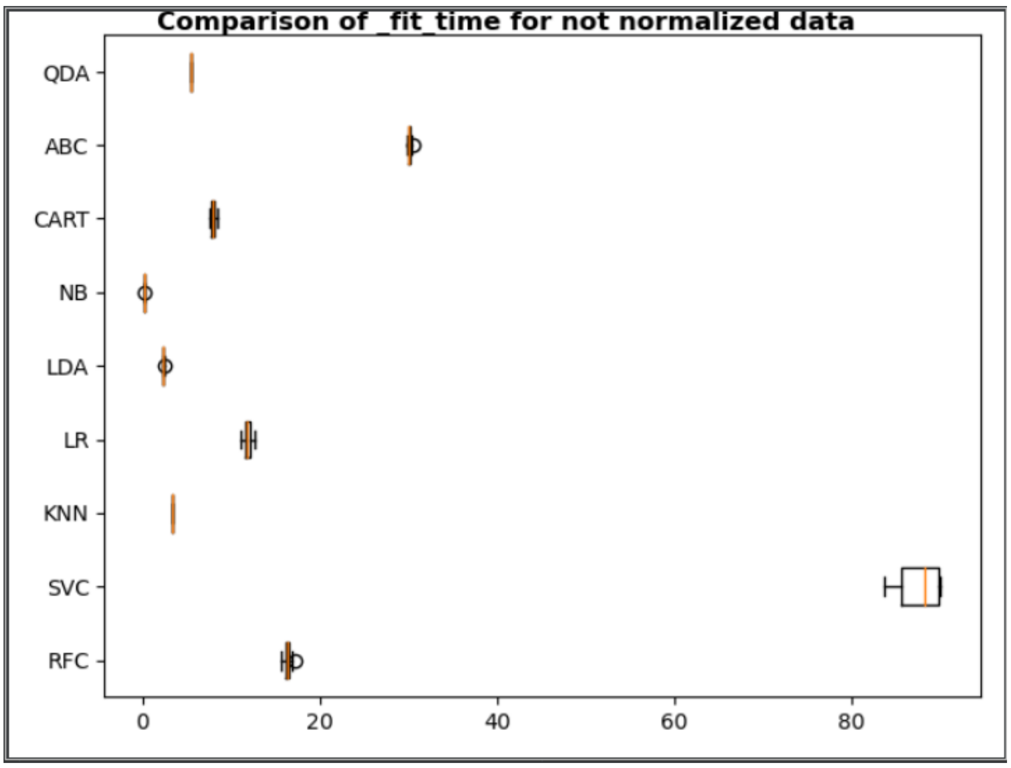
- Random Forests, Linear Discriminant Analysis and Decision Tree are highly successful. However, Ada Boost classifier and Support Vector Machine have very low accuracy.



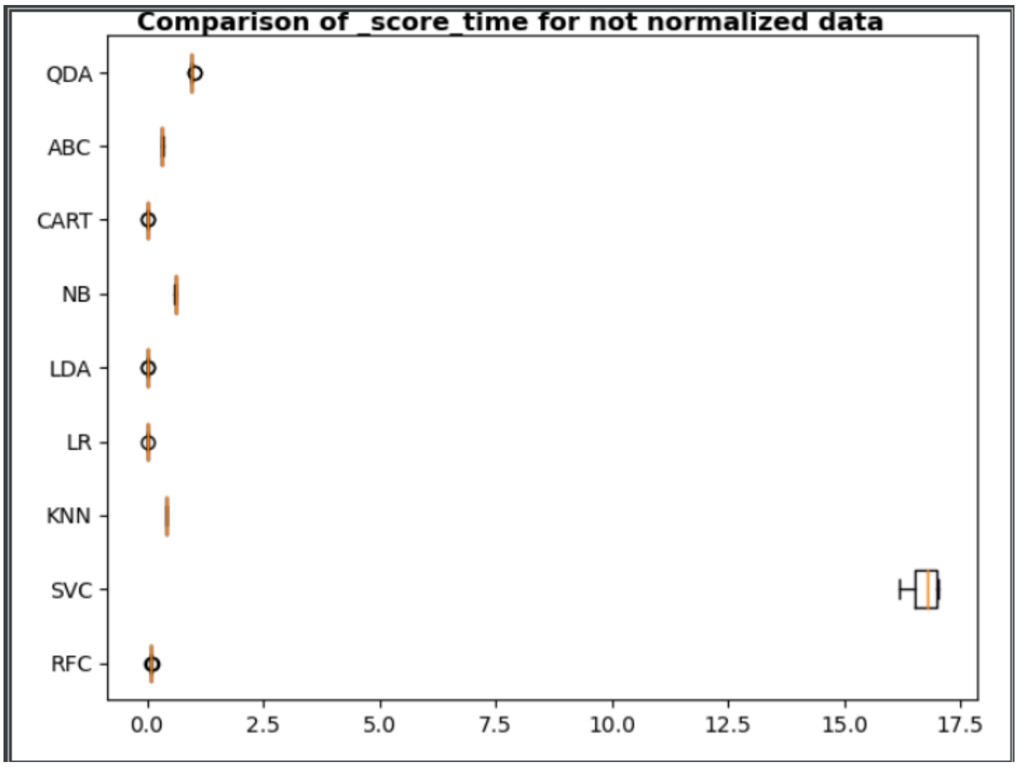
- For Quadratic Discriminant Analysis, Decision Tree, Linear Discriminant Analysis and Random Forests classifiers error rate is pretty low.



- Support Vector Machine Model takes at least 4 times more time compared to other classifiers. Except Support Vector Machine and Ada Boost classifier all the models fit the data in 20 seconds.

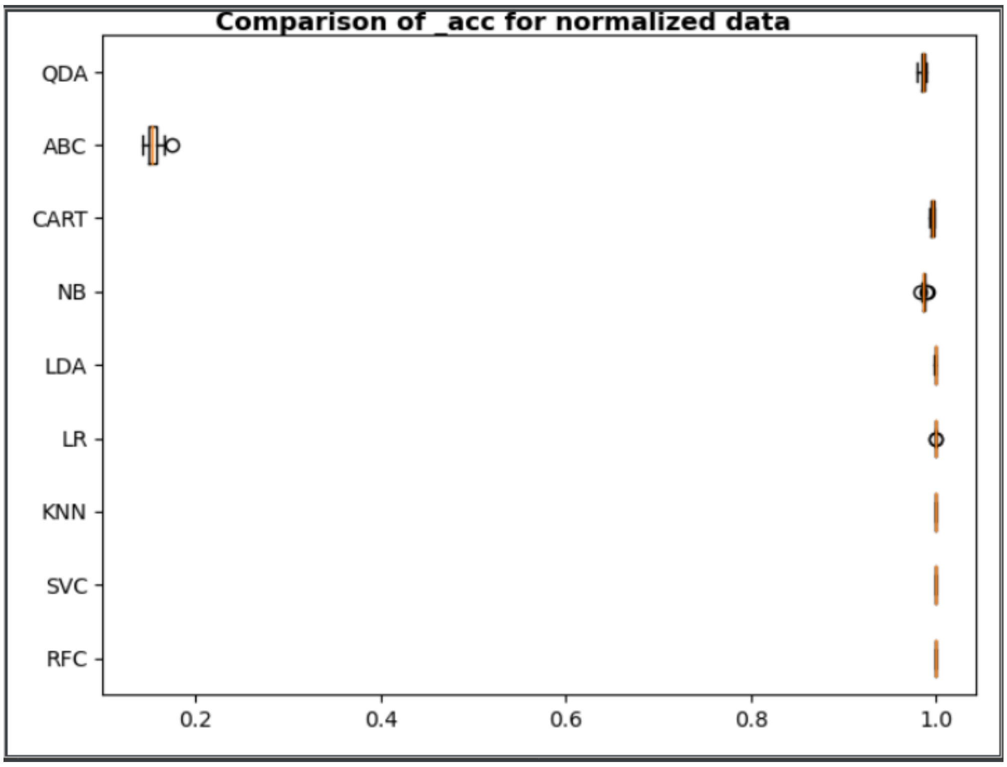


- Scoring time is the worst for Support Vector Machine Model.

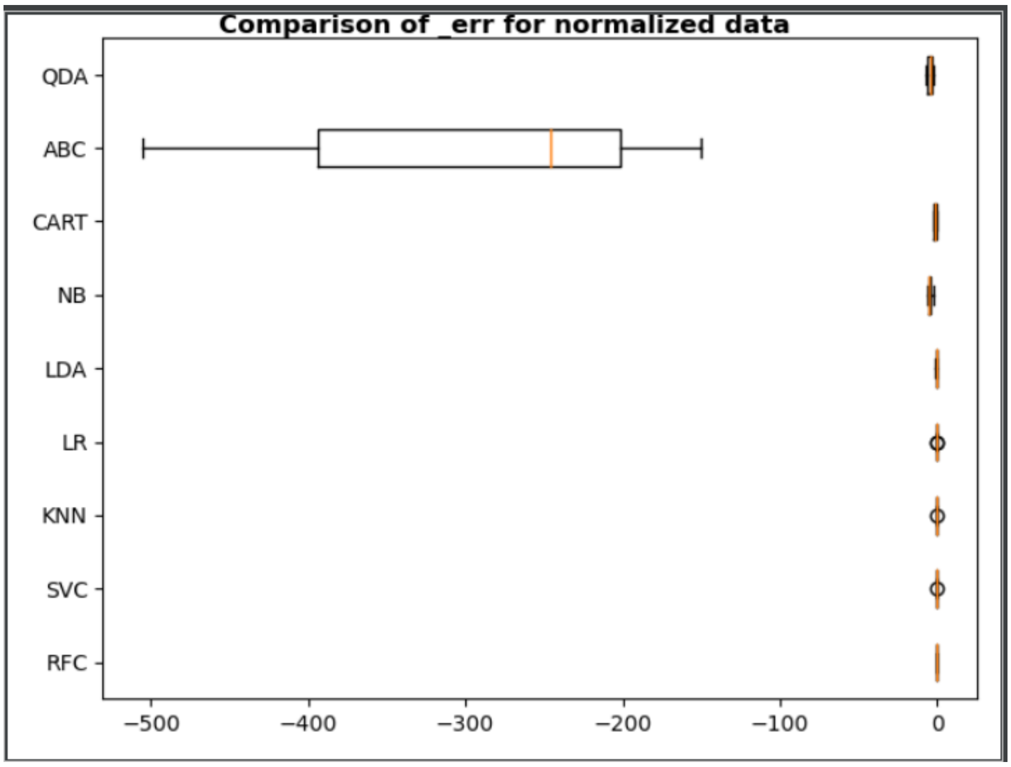


2 Normalized data set

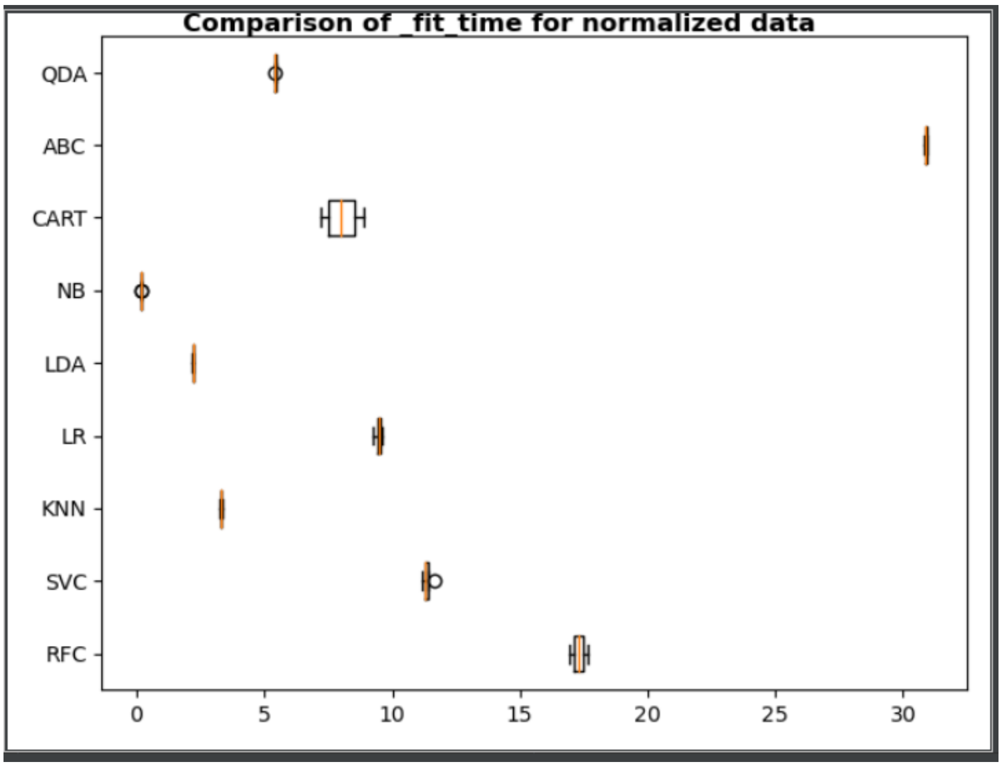
- As we see Ada Boost classifier gives low accuracy as before. But normalization of the data helps hugely to other classifiers. Especially Support Vector Machine, Logistic Regression, K-Nearest Neighbors and Gaussian Naive Bayes.



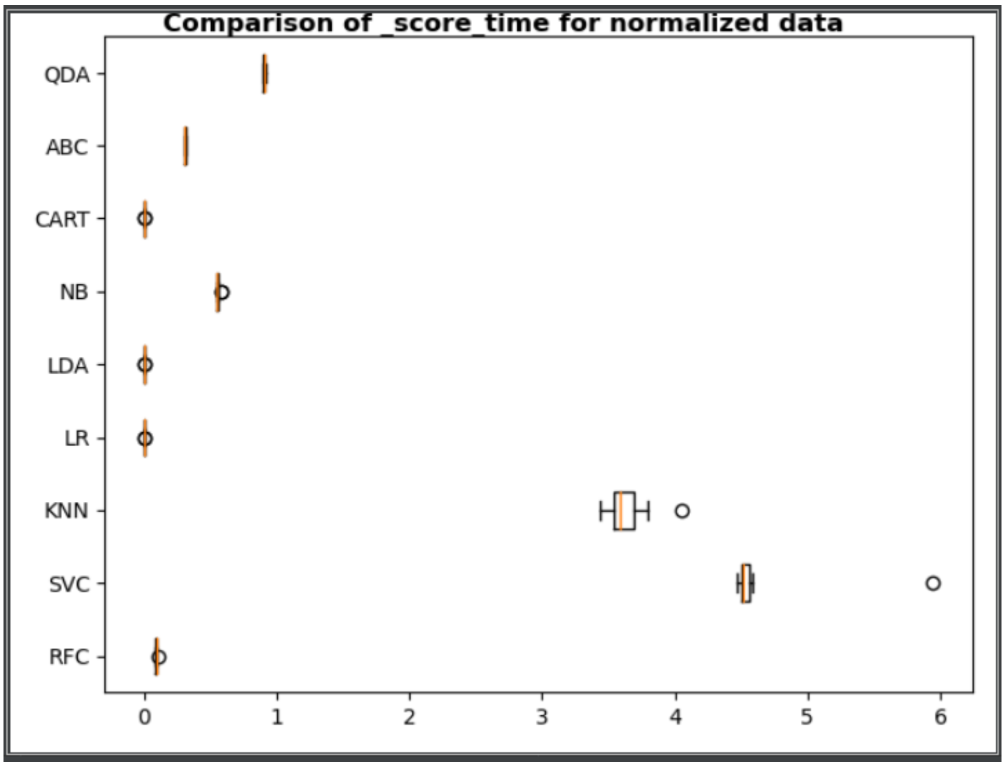
- The amount of errors is close to 0 in the majority of classifiers. However, Ada Boost is an exception.



- _fit_time is the time for fitting the estimator on the train set for each cross validate split. We observe that fit time varies highly: the best being Gaussian Naive Bayes and the worst Ada Boost Model. Gaussian Naive Bayes is the fastest in fitting the data.



- _score_time is the time for scoring the estimator on the test set for each cross validate split. Scoring time is the highest for Support Vector Machine Model. The second worst result is for K-Nearest Neighbors Model. The other classifiers have efficient results. Linear Discriminant Analysis, Logistic Regression and Decision Tree classifier with the random state seed size=9 are the most efficient.



Chapter 7

Source code

- Feature extraction

```
import numpy as np
import os
import h5py
import mahotas
import cv2 as cv
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from skimage import feature


def fd_hu_moments(image):  # shapes
    img = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    feat = cv.HuMoments(cv.moments(img)).flatten()
    return feat


def fd_haralick(image):  # haralick texture
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    return haralick


def fd_histogram(image):  # color histogram
    img = cv.cvtColor(image, cv.COLOR_BGR2HSV)
    hist = cv.calcHist([img], [0, 1, 2], None, [
        8, 8, 8], [0, 256, 0, 256, 0, 256])
    cv.normalize(hist, hist)
    return hist.flatten()


def fd_localBinaryPatters(image, numPoints=24, radius=8):  # local features
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    lbp = feature.local_binary_pattern(
        gray, numPoints, radius, method="uniform")
    (hist, _) = np.histogram(lbp.ravel(),
                             bins=np.arange(0, numPoints + 3),
                             range=(0, numPoints + 2))
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-7)
    return hist


def getGlobalFeatures(file):
    image = cv.imread(file)
    fv_hu_moments = fd_hu_moments(image)
    fv_haralick = fd_haralick(image)
    fv_histogram = fd_histogram(image)
    fv_lbp = fd_localBinaryPatters(image)
    return np.hstack([fv_hu_moments, fv_haralick, fv_histogram, fv_lbp])


trainPath = "data/Training"
h5Data = "output/dataNonNormalized.h5"  # output/data.h5 for normalized
h5Labels = "output/labelsNonNormalized.h5"  # output/data.h5 for normalized

trainLabels = os.listdir(trainPath)

globalFeatures = []
labels = []

for trainName in trainLabels:
    dir = os.path.join(trainPath, trainName)

    for file in os.listdir(dir):
        globalFeature = getGlobalFeatures(dir+"/"+file)

        labels.append(trainName)
        globalFeatures.append(globalFeature)
    print("{} class processed".format(trainName))

targetNames = np.unique(labels)
le = LabelEncoder()
target = le.fit_transform(labels)

#scaler = MinMaxScaler(feature_range=(0, 1))
#rescaledFeatures = scaler.fit_transform(globalFeatures)

h5fData = h5py.File(h5Data, "w")
h5fData.create_dataset("globalFeatures", data=np.array(globalFeatures))
h5fData.close()

h5fLabel = h5py.File(h5Labels, "w")
h5fLabel.create_dataset("labels", data=np.array(target))
h5fLabel.close()

print("\nFeatures extracted")
```

- Model comparison

```
import h5py
import numpy as np
from sklearn.model_selection import train_test_split, KFold, cross_validate
from sklearn.ensemble import RandomForestClassifier,  AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

import warnings

warnings.simplefilter("ignore")  # warning suppression

h5Data = "output/data.h5"
h5Labels = "output/labels.h5"
h5Results = "output/results.h5"

h5fData = h5py.File(h5Data, "r")
h5fLabels = h5py.File(h5Labels, "r")
h5fResults = h5py.File(h5Results, "w")

globalFeaturesString = h5fData["globalFeatures"]
globalFeatures = np.array(globalFeaturesString)
globalLabelsString = h5fLabels["labels"]
globalLabels = np.array(globalLabelsString)

h5fData.close()
h5fLabels.close()

nSplits = 10
treeNumber = 100
testSize = 0.1
seed = 9
scoring = {"acc": "accuracy", "err": "neg_mean_squared_error"}

models = []
models.append(("RFC", RandomForestClassifier(
    n_estimators=treeNumber, random_state=seed)))
models.append(("SVC", SVC(random_state=seed)))
models.append(("KNN", KNeighborsClassifier()))
models.append(("LR", LogisticRegression(random_state=seed)))
models.append(("LDA", LinearDiscriminantAnalysis()))
models.append(("NB", GaussianNB()))
models.append(("CART", DecisionTreeClassifier(random_state=seed)))
models.append(("ABC", AdaBoostClassifier()))
models.append(("QDA", QuadraticDiscriminantAnalysis()))

(trainDataGlobal, testDataGlobal, trainLabelsGlobal, testLabelsGlobal) = train_test_split(
    globalFeatures, globalLabels, test_size=testSize, random_state=seed)

# results = []
for name, model in models:
    kfold = KFold(n_splits=nSplits, random_state=seed)
    cvResult = cross_validate(model, trainDataGlobal, trainLabelsGlobal,
                              cv=kfold, scoring=scoring, return_train_score=True)
    # results.append((name, cvResult["test_acc"], cvResult["test_prec"]))
    h5fResults.create_dataset(name+"_acc", data=np.array(cvResult["test_acc"]))
    # h5fResults.create_dataset(
    #     name+"_prec", data=np.array(cvResult["test_prec"]))
    h5fResults.create_dataset(name+"_err", data=np.array(cvResult["test_err"]))
    h5fResults.create_dataset(
        name+"_fit_time", data=np.array(cvResult["fit_time"]))
    h5fResults.create_dataset(
        name+"_score_time", data=np.array(cvResult["score_time"]))

h5fResults.close()
```

- Gui and visualzation

```
import warnings
from tkinter.tix import *

import h5py
import matplotlib
import self as self
from matplotlib import pyplot
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
from pandas import np

matplotlib.use('TkAgg')
from tkinter import *


warnings.simplefilter("ignore")


def graph(metric, normal=False):
    metric_results = []

    for name in names:
        if normal is True:
            model_metricString = h5fResults[name + metric]
        else:
            model_metricString = h5fNonNormalizedResults[name + metric]
        model_metric = np.array(model_metricString)
        metric_results.append(model_metric)

    fig = Figure(figsize=(5, 3), dpi=100)

    if normal:

        fig.suptitle('Comparison of ' + metric + " for normalized data", fontweight='bold',
                     horizontalalignment='center', verticalalignment='top', y=0.999)
    else:
        fig.suptitle('Comparison of ' + metric + " for not normalized data", fontweight='bold',
                     horizontalalignment='center', verticalalignment='top', y=0.999)

    ax = fig.add_subplot(111)

    ax.boxplot(metric_results, vert=False, whis=1.5,
               positions=None, widths=None, patch_artist=False, )
    ax.set_yticklabels(names)
    canvas = FigureCanvasTkAgg(fig, master=top)
    canvas.get_tk_widget().grid(row=0, column=1, padx=20, pady=20, sticky=E)
    canvas.draw()


if __name__ == '__main__':
    top = Tk()
    top.title('Classifier comparsion app')
    top.geometry("700x400")
    top.resizable(0, 0)
    h5Results = "output/results.h5"
    h5fResults = h5py.File(h5Results, "r")
    h5NonNormalizedResults = "output/resultsNonNormalized.h5"
    h5fNonNormalizedResults = h5py.File(h5NonNormalizedResults, "r")
    names = ["RFC", "SVC", "KNN", "LR", "LDA", "NB", "CART", "ABC", "QDA"]

    metrics = ["_acc", "_err", "_fit_time", "_score_time"]

    labelframe = LabelFrame(top)
    labelframe.grid(row=0, column=0, padx=20, pady=130)

    variable = StringVar()
    variable.set(metrics[0])  # default value

    w = OptionMenu(labelframe, variable, *metrics)
    w.grid(row=2, column=0)


    var1 = BooleanVar()
    var1.set(False)
    Checkbutton(labelframe, text="Normalize data", variable=var1).grid(row=3, column=0)

    Button(labelframe, text="graph", command=lambda: graph(variable.get(), normal=var1.get())).grid(row=4, column=0)

    mainloop()

    top.mainloop()

    h5fResults.close()
    h5fNonNormalizedResults.close()
```

Chapter 8

Installing and running the project

For the source code and the data set clone the project from github

https://github.com/elif-pw/AI

Use IntelliJ PyCharm or any other software and install all required packages for Scikit-learn.

Run the following

- `extract_features`.py takes fruit images (located in data directory) as an input and produces HDF5 files (output directory).

  Additionaly if you wanted to swap between normalized and non-normalized data you can do that in this file.

  ```
  scaler = MinMaxScaler(feature_range=(0, 1))
  rescaledFeatures = scaler.fit_transform(globalFeatures)

  h5fData = h5py.File(h5Data, "w")
  h5fData.create_dataset("globalFeatures", data=np.array(rescaledFeatures))
  h5fData.close()
  ```

  This part of the code is responsible for either saving normalized or non-normalized data. These are lines 74 up to 79 in `extract_features`.py, if they're written the way they are above, we save normalized data. Alternatively we can change it to:

  ```
  #scaler = MinMaxScaler(feature_range=(0, 1))
  #rescaledFeatures = scaler.fit_transform(globalFeatures)

  h5fData = h5py.File(h5Data, "w")
  h5fData.create_dataset("globalFeatures", data=np.array(globalFeatures))
  h5fData.close()
  ```

  To save non-normalized data

- `train_test`.py Takes beforementioned HDF5 files as an input and trains, as well as tests given machine learning models.

- `gui`.py is responsible only for the layout and showing selected information to the user in the form of boxplot.

- `visualize`.py is for plotting the boxplots for normalized data and non normalized data

  for application : It has 2 main areas.

  1.Control panel: we can set metric of four criteria : Accuracy, Mean squared error, Fit time, Score time for which we want to compare classifiers, as well as whether we want to use normalized data.

  2.Graph display area

Chapter 9

Conclusion

The main purpose of this project application is to test efficiency of different Python machine learning models based on different criteria. Logistic Regression, Linear Discriminant Analysis, K-Nearest Neighbors, Decision Tree, Random Forests, Gaussian Naive Bayes, Support Vector Machine, Quadratic Discriminant Analysis, Ada Boost classifiers were trained and tested on fruit data set that was gathered by image processing.

- Non-normalized data: From our observations we could conclude that the following classifiers have similarly high performance: Linear Discriminant Analysis, Quadratic Discriminant Analysis, Decision Tree, Random Forests (the provided order indicates length of consumed time) Ada Boost classifier has the worst accuracy and the greatest rate of errors. Data fit time is also higher than other models (except Support Vector Machine which is the next worst model) The other machine learning model that we wouldn't advise to use is Support Vector Machine. It has a little bit better result than Ada Boost classifier in accuracy and the interquartile range of errors is smaller. However, it takes considerably big amount of time for fitting and scoring the data.
- Normalized data: Accuracy rate is close to 1 for all of the models except Ada Boost classifier which has no improvement compared to non-normalized data result. In the same way error rate is small for all of the models except Ada Boost classifier which again has a very high number of errors.

The success of classifiers depends on many factors, one of them being the range of the data set values. In the 5th section we showed the importance of normalization. We provided the results of the classifiers on the non-normalized and normalized data in boxplot graphs. As observed, classifiers that have bad outcomes for non-normalized data can have almost perfect scores for the normalized data. This points out on importance of the provided data.

All things considered, there are many factors that have high potential to affect the performance of a machine learning model. Data quality and quantity are just few of them. Above we mentioned the analyzed models and their performance. Our application will give an idea to the user about the accuracy, error rate, fitting and scoring time which define the success rate of the models.

Bibliography

[1] Dataset: https://github.com/Horea94/Fruit-Images-Dataset

[2] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

[3] https://www.researchgate.net/publication/321475443_Fruit_recognition_from_images_using_deep_learning

[4] https://www.kaggle.com/moltean/fruits

[5] https://www.researchgate.net/publication/321475443_Fruit_recognition_from_images_using_deep_learning

[6] https://gogul.dev/software/image-classification-python