Image Processing and Computer Vision
Project-2

Crack Detection

Done by:
Mohd Wafa

Content

1. Task Description

Concrete surface crack is incredibly necessary for maintenance of concrete structures. Furthermore, tiny cracks that appear insignificant could grow and eventually cause severe structural failure. The firmness of structure is a very significant issue not solely within the construction part however conjointly within the conservation part. The significance of wellbeing in the facility is accumulated because of the development of tall structures, too long-range spans, and uneven buildings area units in style. Therefore, development security the board framework is effectively underneath improvement recently.

Dependability, execution, and life cycle costs square measure main problems for about all in-administration enormous structures, similar to structures, spans, atomic offices, power structures, and dams. Cracks on these structures square measure a normal improvement related to differed inside and outside powers, together with the erosion of implanted support, concoction crumbling of cement, thus the machine of unfriendly stacking to the structure. The presence of cracks off and on again shows imperative trouble inside the structures. In this way, to make certain the basic reliability and execution of the structure for an incredible duration, auxiliary wellbeing viewing (SHM) frameworks square measure required to forestall ruinous disappointment inside the primary stages.

SHM is the technique that actualizes a crack identification and methodology for building structures. The manual examination needs judgment inside the measuring. The target of this venture is to create a programmed break location framework that may investigate the solid surface and arrange the cracks with proficiency. In contrast with the typical manual assessment based crack discovery framework, ML-based methodologies are performed to adjust split location and recognizable proof frameworks. These techniques square measure essentially built upon to order the concrete image dataset and to appropriately build up splits. Different classification algorithms square measure applied to locate "crack" and "no-crack" conditions from solid picture information through extricating choices. Execution examination is finished to accomplish higher precision in crack recognizable proof. Machine learning algorithms are used for classifying images to workout whether it is crack or no longer. This may be accomplished via classification strategies to categorize the entire dataset and also to analyze the best model in keeping with its accuracy and precision

1.1 Dataset Used

The dataset used for this project is a collection of ‘Positives’ and ‘Negatives’. There are 20,000 sample data images for each class, so a total of 40,000 sample data images.

Positives relate to the images which have a concrete surface with a crack in it. Negatives relate to the images that do not have a crack in the concrete surfaces.

POSITIVE

NEGATIVE

Fig. 1 : Example image for the different classifications

2. Algorithm Description

The main algorithm has been categorized into 2 different sections :

1. Extraction the features descriptors
2. Training the model

Since there are only 2 classes, the problem becomes very trivial and the data set can be easily used. Now the main thought is to implement different combinations of descriptors and compare their accuracy scores with each other.
We also had to calculate the difference formed with the training set size for the final accuracy calculations.

We had to start training it with an increasing data set, starting from 1000 images.

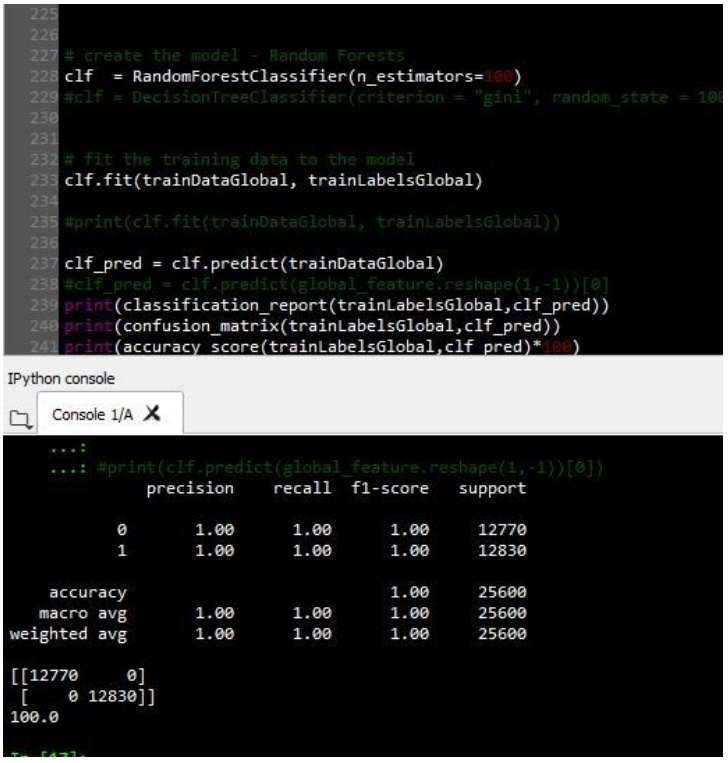
2.1 Feature Descriptors

I used 3 different feature descriptors for this project. They are as follows :

1) Haralick Texture

   Haralick texture features are calculated from a Gray Level Co-occurrence Matrix, (GLCM), a matrix that counts the co-occurrence of neighboring gray levels in the image. The GLCM is a square matrix that has the dimension of the number of gray levels *N* in the region of interest (ROI). Each texture feature is a function of the elements of the GLCM, and represents a specific relation between neighboring voxels. The texture features can indicate e.g. image contrast (large differences between neighboring voxels) or entropy, (the orderliness of the gray level distribution in the image).

   Accuracy of using the Haralick Texture is displayed in the snippet below :



Accuracy using Haralick Textures

==========

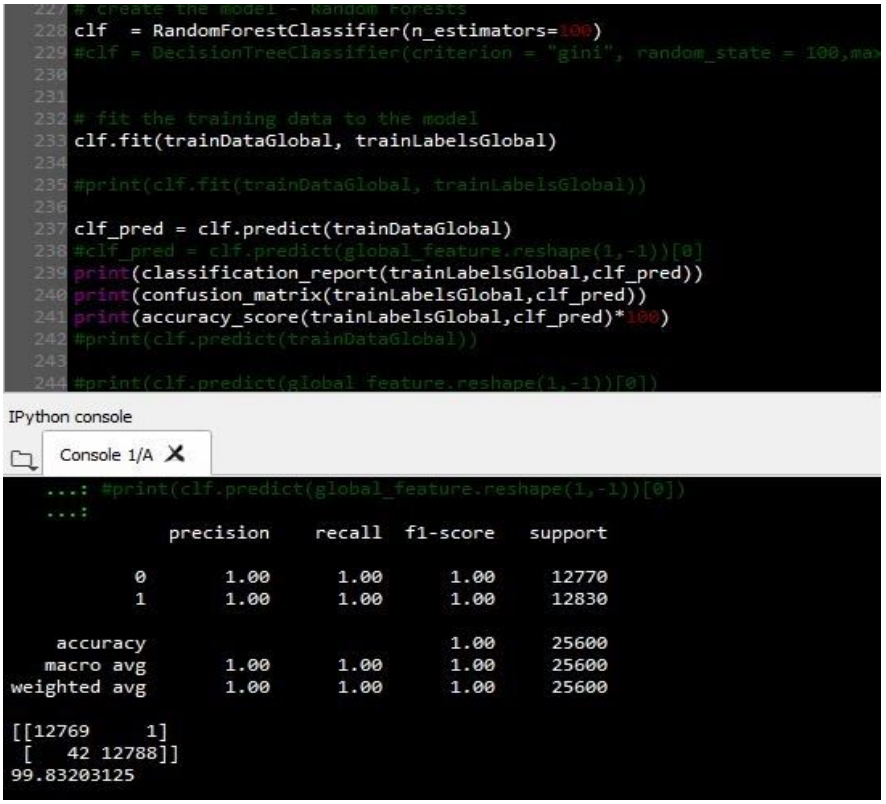
```
# feature-descriptor -2 Haralick Texture

def fd_haralick(image):
    # conver the image to grayscale
    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    # Ccompute the haralick texture fetature ve tor
    haralic = mahotas.features.haralick(gray).mean(axis=0)
    return haralic
```

==========

2) Hu Moments

Hu Moments (or rather Hu moment invariants) are a set of 7 numbers calculated using central moments that are invariant to image transformations. The first 6 moments have been proved to be invariant to translation, scale, and rotation, and reflection.

Accuracy of using Hu Moments is given in the snippet below :



Accuracy using Hu moments

==========

```
# features description -1:  Hu Moments

def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature
```

==========

3) Color Histogram

In image processing and photography, a color histogram is a representation of the distribution of colors in an image. For digital images, a color histogram represents the number of pixels that have colors in each of a fixed list of color ranges, that span the image's color space, the set of all possible colors.

The color histogram can be built for any kind of color space, although the term is more often used for three-dimensional spaces like RGB or HSV. For monochromatic images, the term intensity histogram may be used instead. For multi-spectral images, where each pixel is represented by an arbitrary number of measurements (for example, beyond the three measurements in RGB), the color histogram is *N*-dimensional, with N being the number of measurements taken. Each measurement has its own wavelength range of the light spectrum, some of which may be outside the visible spectrum.

==========

```
# feature-description -3 Color Histogram

def fd_histogram(image, mask=None):
    # conver the image to HSV colors-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    #COPUTE THE COLOR HISTPGRAM
    hist  = cv2.calcHist([image],[0,1,2],None,[bins,bins,bins],
[0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist,hist)
    # return the histog....
    return hist.flatten()
```

==========

Accuracy of using Color Histogram is given in the snippet below :

```
clf.fit(trainDataGlobal, trainLabelsGlobal)

#print(clf.fit(trainDataGlobal, trainLabelsGlobal))

clf_pred = clf.predict(trainDataGlobal)
#clf_pred = clf.predict(global_feature.reshape(1,-1))[0]
print(classification_report(trainLabelsGlobal,clf_pred))
print(confusion_matrix(trainLabelsGlobal,clf_pred))
print(accuracy_score(trainLabelsGlobal,clf_pred)*100)
#print(clf.predict(trainDataGlobal))

#print(clf.predict(global_feature.reshape(1,-1))[0])


# In[129]:
```

IPython console

Console 1/A

```
[STATUS] target labels shape: (32000,)
[STATUS] end of training..
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12770
           1       1.00      1.00      1.00     12830

    accuracy                           1.00     25600
   macro avg       1.00      1.00      1.00     25600
weighted avg       1.00      1.00      1.00     25600

[[12770     0]
 [    0 12830]]
100.0
```

Accuracy using Color Histogram

2.2 Machine Learning model

In this project, we were also required to use specific classifiers that would classify the images to POSITIVE or NEGATIVE, based on the type of picture.

The classifiers used are as follows :

1. Decision Tree :

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The

deeper the tree, the more complex the decision rules and the fitter the model.

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches.

2. K-nearest neighbours (KNN)

   KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification) or averages the labels (in the case of regression).

   In the case of classification and regression, we saw that choosing the right K for our data is done by trying several Ks and picking the one that works best.

3. Random Forest

Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest consists of trees. It is said that the more trees it has, the more robust a forest is. Random forests create decision trees on randomly selected data samples, get prediction from each tree and select the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

Random forests have a variety of applications, such as recommendation engines, image classification and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.

Below is a snippet for the machine learning code implemented in the project :

```
# In[128]:


clf = DecisionTreeClassifier(criterion = "gini", random_state = 100,max_depth=3, min_samples_leaf=5)


# fit the training data to the model
clf.fit(trainDataGlobal, trainLabelsGlobal)

#print(clf.fit(trainDataGlobal, trainLabelsGlobal))

clf_pred = clf.predict(trainDataGlobal)
#clf_pred = clf.predict(global_feature.reshape(1,-1))[0]
print(classification_report(trainLabelsGlobal,clf_pred))
print(confusion_matrix(trainLabelsGlobal,clf_pred))
print(accuracy_score(trainLabelsGlobal,clf_pred))
#print(clf.predict(trainDataGlobal))

#print(clf.predict(global_feature.reshape(1,-1))[0])
```

Here is an snippet of code used for Random Forest in the project :

```
# In[128]:


# create the model - Random Forests
clf   = RandomForestClassifier(n_estimators=100)
#clf = DecisionTreeClassifier(criterion = "gini", random_state = 100,max_depth=


# fit the training data to the model
clf.fit(trainDataGlobal, trainLabelsGlobal)

#print(clf.fit(trainDataGlobal, trainLabelsGlobal))

clf_pred = clf.predict(trainDataGlobal)
#clf_pred = clf.predict(global_feature.reshape(1,-1))[0]
print(classification_report(trainLabelsGlobal,clf_pred))
print(confusion_matrix(trainLabelsGlobal,clf_pred))
print(accuracy_score(trainLabelsGlobal,clf_pred)*100)
#print(clf.predict(trainDataGlobal))

#print(clf.predict(global_feature.reshape(1,-1))[0])
```

3. Results

3.1 Features Quality

As the task stated, the final set for features were divided into 2 major components :

80% training and 20% testing.

The accuracies for the individual features is presented in the section before, while explaining the features. All the accuracies were around 100% and the majority of the testing was very much successful. I thought that that would be a good place to mention the accuracies in the report.

Below in the table is given the feature performance based on the different IPC(Images per class).

Feature	1000 IPC	20,000 IPC
Haralick Textures	98.9%	99.2%
Hu moments	91.8%	94.3%
Color Histogram	98%	99.5%

From the above table, we can know that the most accurate feature descriptor for this project is clearly Color Histogram. This is the obvious and clear choice, since the cracks are a bit darker in colour, and so a color based feature can give the best result, hence stating in the percentage.

The feature descriptor that came in second is the Haralik Textures, which as above mentioned, also bases its results on colours. But it does not only

depend on colours, as a major factor for Haralick features is the spatial arrangement, which the feature also takes into account.

And lastly, the Hu Moments comes in last as it was significantly lower than others, based on others differences. But the lower percentage could be proven to be because of Hu moments shape recognition function, which isn't really the optimal solution for our project's problem.

But that was only for the features taken into account individually. As the task also stated to find the accuracies for the combination of different features and to compare them, so now I will be comparing the different combinations, namely : Hu+Haralick, Hu+Color_Histogram, Haralick+Color_Histogram, and finally all three of them combined :

1. Haralick feature + Hu moment :

```
145         #image = cv2.resize(image,fixed_size)
146
147         # Concatenate global features
148         global_feature = np.hstack([ fv_haralick, fv_hu_moments])
149
150         # update the list of labels and feature vectors
151         labels.append(current_label)
152         global_features.append(global_feature)
153
154         i += 1
155         k += 1
156     print("[STATUS] processed folder: {}".format(current_label))
157     j += 1
158
159 print("[STATUS] completed Global Feature Extraction...")
160
161
```

IPython console

Console 1/A

```
[STATUS] target labels shape: (32000,)
[STATUS] end of training..
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12770
           1       1.00      1.00      1.00     12830

    accuracy                           1.00     25600
   macro avg       1.00      1.00      1.00     25600
weighted avg       1.00      1.00      1.00     25600

[[12770     0]
 [    0 12830]]
100.0
```

2. Haralick feature + Color Histogram :

```
            fv_haralick   = fd_haralick(image)
            fv_histogram  = fd_histogram(image)
        #else:
            #print("image not loaded")

        #image = cv2.imread(file)
        #image = cv2.resize(image,fixed_size)

        # Concatenate global features
        global_feature = np.hstack([fv_histogram, fv_haralick])

        # update the list of labels and feature vectors
        labels.append(current_label)
        global_features.append(global_feature)

        i += 1
        k += 1
    print("[STATUS] processed folder: {}".format(current_label))
```

IPython console

Console 1/A

```
[STATUS] target labels shape: (32000,)
[STATUS] end of training..
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12770
           1       1.00      1.00      1.00     12830

    accuracy                           1.00     25600
   macro avg       1.00      1.00      1.00     25600
weighted avg       1.00      1.00      1.00     25600

[[12770     0]
 [    0 12830]]
100.0
```

3. Hu moment + Color Histogram :

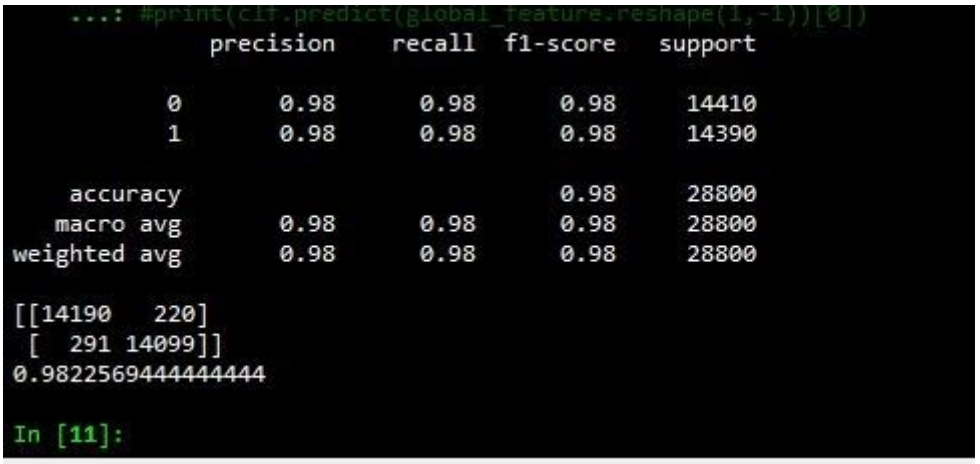
```
        # Concatenate global features
        global_feature = np.hstack([fv_histogram, fv_hu_moments])

        # update the list of labels and feature vectors
        labels.append(current_label)
        global_features.append(global_feature)

        i += 1
        k += 1
    print("[STATUS] processed folder: {}".format(current_label))
    j += 1

print("[STATUS] completed Global Feature Extraction...")


# In[30]:
```

IPython console

Console 1/A

```
[STATUS] target labels shape: (32000,)
[STATUS] end of training..
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12770
           1       1.00      1.00      1.00     12830

    accuracy                           1.00     25600
   macro avg       1.00      1.00      1.00     25600
weighted avg       1.00      1.00      1.00     25600

[[12770     0]
 [    0 12830]]
100.0
```

4. Hu moment + Haralick Moment + Color Histogram :



```
...: #print(clf.predict(global_feature.reshape(1,-1))[0])
              precision    recall  f1-score   support

           0       0.98      0.98      0.98     14410
           1       0.98      0.98      0.98     14390

    accuracy                           0.98     28800
   macro avg       0.98      0.98      0.98     28800
weighted avg       0.98      0.98      0.98     28800

[[14190   220]
 [  291 14099]]
0.9822569444444444

In [11]:
```

As we can see, the results with the combination of features was very successful, especially Haralick features + Color Histogram, as both of them are color based features, including Haralick features spatial arrangements.

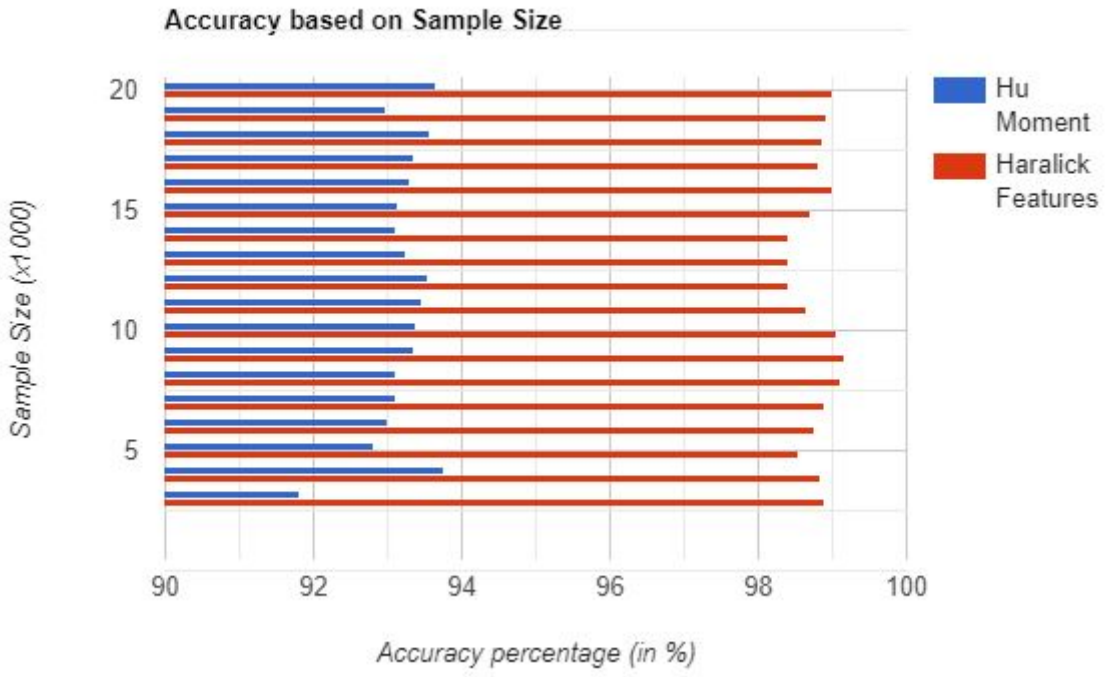
To make a summary for the combinations results, the overall result was very successful for all the different combinations of features used.

3.1 Impact of the Sample Size

For the impact of sample size, we have taken into account 2 features, namely, Haralick features and Hu moments.

IPC	Hu Moments	Haralick Textures
1000	91.8%	98.9%
2000	93.75%	98.85%
3000	92.8%	98.55%
4000	93%	98.755%
5000	93.1%	98.9%
6000	93.1%	99.1%
7000	93.35%	99.155%
8000	93.375%	99.05%
9000	93.45%	98.65%
10000	93.55%	98.4%
11000	93.25%	98.4%
12000	93.1%	98.4%
13000	93.125%	98.7%
14000	93.3%	99%
15000	93.355%	98.8%
16000	93.575%	98.875%
17000	92.975%	98.925%
18000	93.65%	99%
19000	94%	99.125%
20000	94.3%	99.2%

To make the above information more understandable, below is a bar graph with the above entries.



As we can clearly see, Haralick features are way ahead in accuracy percentage compared to that of Hu Moments. But it is easily explained as it was mentioned above that Haralick Features is a color assessing feature, and since, the cracks have a distinct darker colour than that of the plain surface, it gives a better accuracy percentage in detecting cracks

Hu Moments do not have such a high percentage as that of Haralick Features as Hu moments is assessing images based on shapes, which does detect cracks, but not at a higher level of accuracy as Haralick features.

4. Extra notes for the project.

Since the project required us to have the program divided into 80% training and 20% testing, please include 4,000 images of each type

in the test folder and the remaining 16,000 images into the train folder.
Also, please do not forget to update the directories in the CreateDataset.py file accordingly.

The .zip file of the project has been included with this report.

References

1. https://www.nature.com/articles/s41598-017-04151-4#:~:text=Theory-,Texture%20analysis,region%20of%20interest%20(ROI)

2. https://learnopencv.com/shape-matching-using-hu-moments-c-python/#:~:text=Hu%20Moments%20(%20or%20rather%20Hu,%2C%20and%20rotation%2C%20and%20reflection

3. https://en.wikipedia.org/wiki/Color_histogram#:~:text=In%20image%20processing%20and%20photography,set%20of%20all%20possible%20colors

4. https://chiragsehra42.medium.com/decision-trees-explained-easily-28f23241248#:~:text=Decision%20Trees%20(DTs)%20are%20a%20non%2Dparametric%20supervised%20learning,%2Dthen%2Delse%20decision%20rules.&text=The%20final%20result%20is%20a,has%20two%20or%20more%20branches

5. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761#:~:text=KNN%20works%20by%20finding%20the,in%20the%20case%20of%20regression)

6. https://www.datacamp.com/community/tutorials/random-forests-classifier-python

x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x-x