J. P. Morgan Chase Quant Finance Mentorship Program 2023

Deadline for submission: **June 12th, 2023 Monday 2359 Hrs.**

General Instructions

This document contains the Data Science case study on ML-based Optical Character Recognition. You are expected to attempt all the questions.

This case study introduces the basics of digital image processing, machine learning and optical character recognition. A question may have multiple sub-parts, all of which need to be answered. The scores for the sub-parts have been indicated against the question. The solution format for specific questions has been specified alongside the question, In general please provided the code with comments and explain your findings & approach as well.

The case study will be done in a team of 2. Please take note of the Team No. you are allotted, all doubts and submissions will be considered based on that.

You are expected to provide the solution in a ipynb file. The filename should be **Team_<TeamNo.>**. You can use google colab to code your solution. We will provide brief instructions on how to use google colab for uploading the images. Please write comments in code wherever necessary. Use **'QMP_CaseStudy2023_Submission_<TeamNo.>'** as subject for your submission email.

Please note, any other format of submission or any file other than ipynb will not be evaluated. Only submit in the required format. Keep aside some time before the deadline to be able to successfully submit the case study.

You are free to use online resources to improve your understanding of the problem statement. **Plagiarism of any kind will not be tolerated.**

You may reach out to jpmqrmentorship.mumbai@jpmorgan.com throughout the duration of the challenge for any kinds of queries/doubts. Please use **'QMP_CaseStudy2023_Doubt'** as subject line for a quick response.

All the best!

Colab Instructions

You will need to use the images in samples folder for this exercise. We provide the general instructions on how you can upload the samples folder in google drive, mount it in colab, and load the images:

- Open a web browser and go to the Google Drive website: https://drive.google.com/. Sign in to your Google account if prompted.
- Upload the samples folder in google drive i.e. drag and drop the entire samples folder from your local machine to the Google Drive.
- Now, open a new or existing Google Colab notebook: https://colab.research.google.com/. Sign in to your Google account if prompted.
- In the notebook, add a code cell and run the following code to mount your Google Drive:

  ```
  from google.colab import drive
  drive.mount('/content/drive')
  ```

- After running the code cell, a link will be displayed. Click on the link to authenticate and grant access to your Google Drive. Follow the instructions and copy the authorization code.
- Paste the authorization code into the code cell in Google Colab and press Enter. This will mount your Google Drive and create a connection between Colab and your Drive.
- You can now access your Google Drive files and folders within Colab. To load the images from the uploaded folder, you can use the file path of the samples folder in your code.

What is a Digital Image?

An image may be defined as a two-dimensional function, f(x,y), where x and y are spatial (plane) coordinates, and the amplitude of f at any pair of coordinates (x,y) is called the intensity or gray level of the image at that point. When x, y and the intensity values of f are all finite, discrete quantities, we call the image a digital image.

Hence, a digital image is composed of a finite number of elements, each of which has a particular location and value. These elements

are called picture elements, image elements, pels, and pixels. Pixel is the term used most widely to denote the elements of a digital image. In computer language, it is an array or a matrix pixel (picture elements) arranged in columns and rows.

Basic Example of an Image - Number Display

In [1]:
```
# You can change 'N' here to get different displays (digits should be 1,2 or 3 only)
N = 123

# Imports
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline

# Initialise with blank display, out_img is a 2D array
out_img = np.zeros((300, 660), dtype=np.uint8)

# The center coordinates of all 15 circles for one digit are stored in this.
all_circle_center_coordinates = []
x=30
y=30
for j in range(3):
    y=30;
    for i in range(5):
        all_circle_center_coordinates = all_circle_center_coordinates + [[x,y]]
        y=y+60
    x=x+60

# Space between any two consecutive digits is 480.
shift = 480

# For each digit, generate their image.
while(N!=0):
    digit = N%10
    N = (int)(N/10)

    if(digit==1):
        for i in range(len(all_circle_center_coordinates)):
            all_circle_center_coordinates[i][0] = all_circle_center_coordinates[i][0] + shift
            # Digit 1 requires only 5 circles out of 15, hence we check which those 5 circles should be.
            if i not in [0,1,2,3,4,10,11,12,13,14]:
```

```
                cv2.circle(out_img, tuple(all_circle_center_coordinates[i]), 25, 255, -1);
            all_circle_center_coordinates[i][0] = all_circle_center_coordinates[i][0] - shift

    if(digit==2):
        for i in range(len(all_circle_center_coordinates)):
            all_circle_center_coordinates[i][0] = all_circle_center_coordinates[i][0] + shift
            # Digit 2 requires 11 circles out of 15, hence we check which those 11 circles should be.
            if i not in [1,6,13,8]:
                cv2.circle(out_img, tuple(all_circle_center_coordinates[i]), 25, 255, -1);
            all_circle_center_coordinates[i][0] = all_circle_center_coordinates[i][0] - shift

    if(digit==3):
        for i in range(len(all_circle_center_coordinates)):
            all_circle_center_coordinates[i][0] = all_circle_center_coordinates[i][0] + shift
            # Digit 3 requires 11 circles out of 15, hence we check which those 11 circles should be.
            if i not in [1,6,3,8]:
                cv2.circle(out_img, tuple(all_circle_center_coordinates[i]), 25, 255, -1);
            all_circle_center_coordinates[i][0] = all_circle_center_coordinates[i][0] - shift

    # We first generate the right-most digit and then we move left.
    shift = shift-240;

# uint8 means the range will be from 0 to 255.
out_img = np.uint8(out_img)

# Show the image as a grayscale image.
plt.imshow(out_img, 'gray')
```

Image Processing Example

Image processing refers to the manipulation and analysis of digital images using various techniques and algorithms. It involves applying mathematical operations, algorithms, and computational methods to images in order to enhance them, extract useful information, or perform specific tasks.

Imports

In [9]:
```
import skimage.io as io
import numpy as np
from matplotlib import pyplot as plt
```

```
from skimage.filters import threshold_otsu
from skimage.morphology import erosion, dilation
%matplotlib inline
```

In [10]:
```
# Load the images and convert into grey scale
img = io.imread('samples/akuls.png', as_gray = True)
plt.imshow(img, 'gray', aspect='auto')
print("The shape of the image: ", img.shape)
```

Thresholding

Thresholding is a technique which converts a gray scale image into a binary image. We use Otsu thresholding here to perform image binarization. You can read more about it here: https://hbyacademic.medium.com/otsu-thresholding-4337710dc519

In [11]:
```
threshold = threshold_otsu(img)
img_binary = img > threshold

# Plot the original and the binary image
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(4, 3))

ax1.imshow(img, 'gray', aspect='auto')
ax1.set_title('Original image')
ax1.set_xticks([])
ax1.set_yticks([])

ax2.imshow(img_binary, 'gray', aspect='auto')
ax2.set_title('Binary image')
ax2.set_xticks([])
ax2.set_yticks([])

plt.tight_layout()
plt.show()
```

Then we will try to remove the noise (the line that traverses the image). We will perform Erosions and Dilations (because it is black on white, erosion dilates and dilation erodes). These operation are Morphological Transformations: mathematical operations perfomed on the image's pixels. They will traverse the image with a matrix of nxm (3 by 3 in our case) and multiply the image with it and save the result.

Dilation

Dilation employs a structuring element, which is a small matrix or kernel with a predefined shape and size. The structuring element slides over the image, examining each pixel and its neighborhood. For each position, the dilation operation checks if any part of the structuring element overlaps with the foreground pixels in the image. If there is any overlap, the center pixel of the structuring element is considered to be part of the foreground in the resulting image. It is primarily used for expanding or enlarging foreground regions, filling gaps, joining broken lines, and accentuating features within an image.

In [12]:
```
kernel = np.ones((3,3), np.uint8)
img_dilated = dilation(img_binary, kernel)

# Plot the image before and after dilation.
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(4, 3))

ax1.imshow(img_binary, 'gray', aspect='auto')
ax1.set_title('Original image')
ax1.set_xticks([])
ax1.set_yticks([])

ax2.imshow(img_dilated, 'gray', aspect='auto')
ax2.set_title('Dilated image')
ax2.set_xticks([])
ax2.set_yticks([])

plt.tight_layout()
plt.show()
```

Erosion

Erosion operation considers a structuring element, which is typically a small matrix or kernel with a predefined shape and size. The structuring element slides over the image, examining each pixel and its neighborhood. For each position, the erosion operation checks if all the foreground pixels in the structuring element coincide with the foreground pixels in the image. If they do, the center pixel is considered to be part of the foreground in the resulting image. Otherwise, it is eroded or turned into background. It is primarily used for removing small objects, smoothing boundaries, eliminating noise, and thinning lines within an image.

In [13]:
```
img_eroded = erosion(img_dilated)

# Plot the image before and after erosion.
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(4, 3))
```

```
ax1.imshow(img_dilated, 'gray', aspect='auto')
ax1.set_title('Original image')
ax1.set_xticks([])
ax1.set_yticks([])

ax2.imshow(img_eroded, 'gray', aspect='auto')
ax2.set_title('Eroded image')
ax2.set_xticks([])
ax2.set_yticks([])

plt.tight_layout()
plt.show()
```

Now we perform a last Morphological Transformations but this time the kernel is 4x1 to reduce noise further.

In [14]:
```
kernel = np.ones((4,1), np.uint8)
img_dilated_2 = dilation(img_eroded, kernel)

# Plot the image before and after dilation.
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(4, 3))

ax1.imshow(img_eroded, 'gray', aspect='auto')
ax1.set_title('Original image')
ax1.set_xticks([])
ax1.set_yticks([])

ax2.imshow(img_dilated_2, 'gray', aspect='auto')
ax2.set_title('Dilated image')
ax2.set_xticks([])
ax2.set_yticks([])

plt.tight_layout()
plt.show()
```

In [15]:
```
titles4 = ['Original', 'Dilation', 'Erosion', 'Dilation2']
images4 = [img, img_dilated, img_eroded, img_dilated_2]

# Plot the image before and after dilation.
fig, axs = plt.subplots(2, 2, figsize=(10, 4))
fig.tight_layout(pad=2.0)

for i in range(4):
```

```
    plt.subplot(2, 2, i + 1), plt.imshow(images4[i], 'gray', aspect='auto')
    plt.title(titles4[i])
    plt.xticks([]), plt.yticks([])

plt.show()
```

Optical Character Recognition

Optical Character Recognition or OCR is a technology that enables the conversion of printed or handwritten text into machine-readable text. OCR is used to extract text from various sources such as scanned documents, images, or even videos, and convert it into editable and searchable formats.

The process of OCR involves several steps such as scanning the input document or image, segregating individual characters based on predefined patterns and recognizing them by comparing characters with some known character database or by using machine learning.

OCR in Finance

In the finance world, OCR (Optical Character Recognition) plays a crucial role in streamlining various processes and improving efficiency. Here's a brief introduction to OCR in the finance industry:

- **Document Processing**: OCR is used to convert physical documents, such as invoices, receipts, and financial statements, into digital formats. By extracting the text from these documents, OCR enables faster and more accurate data entry and eliminates the need for manual transcription.
- **Data Extraction**: Financial institutions deal with vast amounts of data on a daily basis. OCR helps extract relevant information from documents like bank statements, tax forms, and contracts.
- **Automated Invoice Processing**: OCR is widely used in automating the invoice processing workflow. It can extract key details from invoices, such as vendor name, invoice number, dates, and line items, and input them directly into accounting systems. This significantly accelerates the invoice approval and payment process.
- **Check Processing**: OCR technology is employed in banking and financial institutions to read and process checks. It helps in capturing crucial information from checks, such as account numbers, amounts, and payee details, automatically. OCR-based check processing systems enable faster check clearing, reduce errors, and improve fraud detection.

- **Compliance and Regulatory Reporting**: OCR assists financial institutions in complying with regulations by efficiently extracting data from legal and regulatory documents. It helps in automating the extraction of information required for compliance reporting, such as customer identification details, transaction records, and account information.

Overall, OCR technology has revolutionized various aspects of financial operations, enabling faster data processing, reducing manual effort, improving accuracy, and enhancing regulatory compliance hence streamlining financial workflows, improving efficiency, and enhancing the overall customer experience in the finance world.

Machine Learning - A Brief Introduction

Machine Learning is designing algorithms that ingest data and learn a model of the data. The learned model can be used to:

- Detect patterns/structures/themes/trends etc. in the data
- Make predictions about future data and make decisions

ML enables intelligent systems to be data-driven rather than rule-driven by supplying training data and building statistical models of data.

- No need to pre-define and hard-code all the rules (infeasible/impossible anyway)
- The rules are not “static”; can adapt as the ML algo ingests more and more data

A good ML model must generalize well on unseen (test data) hence Simpler models should be preferred over more Complex ones. You can use the below program to fit various degree of polynomials to the data and see how increasing the model complexity affects the training process.

In [7]:
```
# Imports
import numpy as np
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot
%matplotlib inline

# input(x) and output(y) variables
x = np.array([107.608, 108.632, 109.773, 110.929, 112.075, 113.27, 115.094, 116.219, 117.388, 118.734,
              119.500, 120.445, 121.95, 123.366, 125.368, 126.7, 127.852, 130.081])
y = np.array([60.323, 61.122, 60.171, 61.187, 63.221, 63.639, 64.989, 63.761, 60.019, 67.857, 60.9,
```

```
              68.169, 66.513, 65.2, 63.45, 61.4, 60.76, 70.551])

# We add an extra axis in x so that its shape becomes (18, 1)
x = x[:,np.newaxis]

def regressor(poly_degree = 2):
    """ Define the complexity of the model by specifying the polynomial degree."""

    # Create the feature matrix which will have all powers of x upto poly_degree
    # i.e. X = [x, x**2, x**3, ..., x**(poly_degree-1)]
    X = x
    for degree in range(poly_degree):
        X = np.concatenate([X, x**degree], axis=1)

    # fit the linear model
    model = LinearRegression().fit(X, y)

    # x_line is the range of all points between min(x) and max(x) on which we make predictions about y.
    x_line = np.arange(min(x), max(x), 1)
    x_line = x_line[:,np.newaxis]
    Xt = x_line
    for degree in range(poly_degree):
        Xt = np.concatenate([Xt, x_line**degree], axis=1)

    # y_line is the prediction made by model on x_line
    y_line = model.predict(Xt)

    # Plot the graph.
    pyplot.xlabel("x")
    pyplot.ylabel("y")
    pyplot.title("Model: Polynomial of Degree " + str(poly_degree))
    pyplot.scatter(x, y)
    pyplot.plot(x_line, y_line, '--', color='red')
    pyplot.show()

# final call to the function with polynomial degree as the argument.
# You can put your own different polynomial here and test out the model fitting.
for polynomial_degree in [1,3,5,10, 25]:
    regressor(polynomial_degree)
```

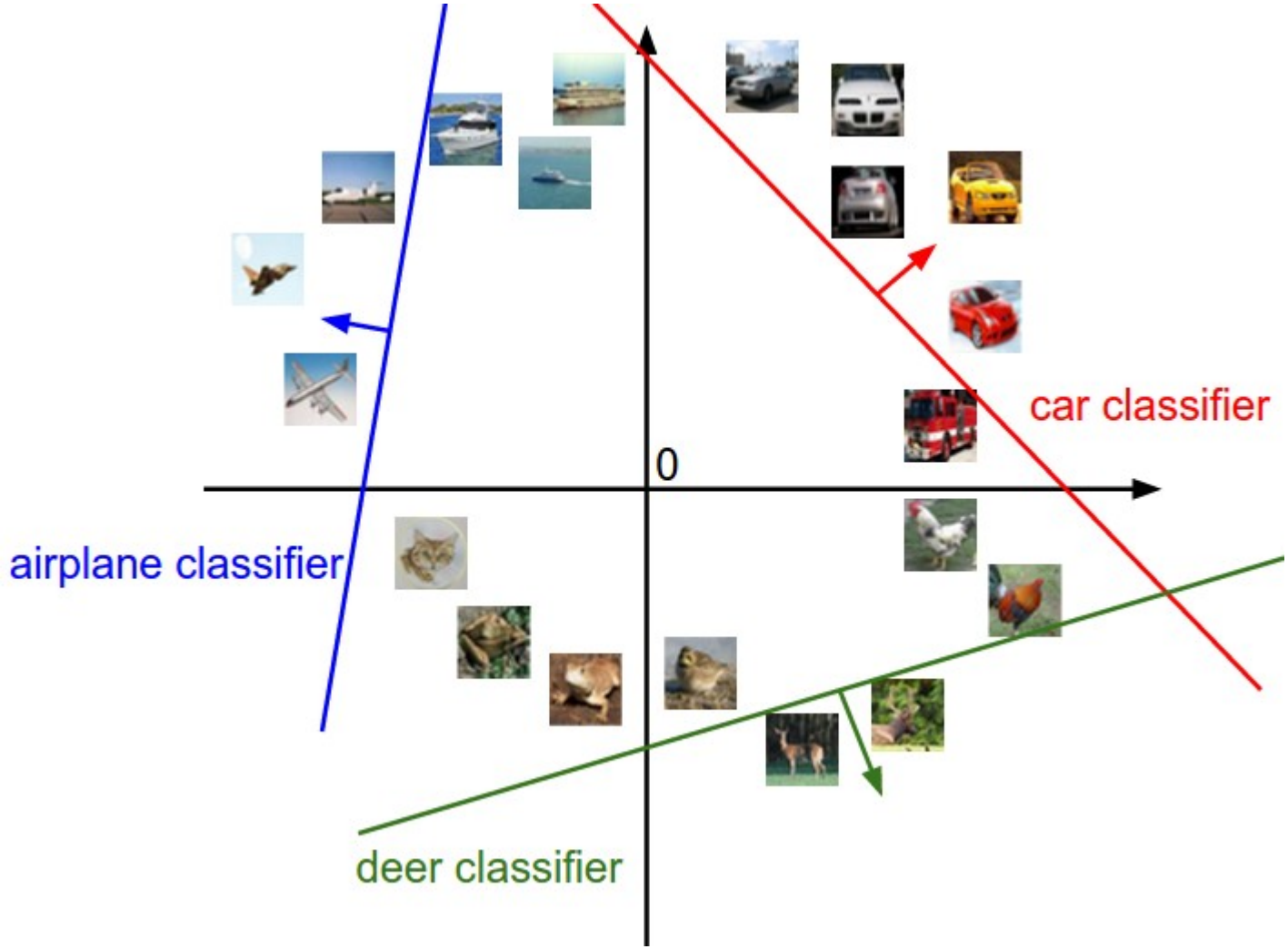
Classification

Machine learning classification is a subfield of artificial intelligence that focuses on developing algorithms and models capable of automatically categorizing or classifying data into different predefined classes or categories. The goal is to train a model using a labeled dataset, where each data instance is associated with a known class or category. The model then learns patterns and relationships within the data to make predictions about the class of unseen or future instances.

The process of building a classification model typically involves several steps. It starts with data preprocessing, which includes tasks like cleaning, transforming, and normalizing the input data. The next step involves selecting or engineering relevant features that capture the essential information for classification. Then, a training set is used to train the model, adjusting its internal parameters to minimize errors or maximize accuracy.

Once the model is trained, it can be evaluated using a separate test set to assess its performance and generalization ability. Metrics like accuracy, precision, recall, and F1-score are commonly used to measure the model's effectiveness. If the performance is satisfactory, the trained model can be deployed to make predictions on new, unseen data, enabling automated classification tasks in various domains, such as spam filtering, sentiment analysis, fraud detection, image recognition, and medical diagnosis.

Classification models employ various techniques and algorithms, such as decision trees, logistic regression, support vector machines (SVM), random forests, and neural networks. These models can handle a wide range of data types, including numerical, categorical, and textual features. The choice of algorithm depends on the characteristics of the data and the specific problem at hand. Brief Information about some of the algorithms you will need to use in this assignment is given below:



Logistic Regression

Logistic Regression is a linear classifier which is used to predict a categorical variable. Let's take an example of email classification wherein you need to classify an email as either spam (1) or non-spam (0). There are two modelling choices we will make in logistic regression:

1. Instead of directly modeling the discrete outcomes of {0, 1} we model the log-odds of an outcome, since it is easier to model a continuous target variable. *The odds of an outcome is defined as the ratio of probability of success and probability of failure. The range of odds is [0, ∞), and log-odds is (-∞, ∞).*

2. The classifier is assumed to be linear i.e. the target variable is a linear combination of the input variables.

Assuming that the probability of an email being a spam email is *pi* and the input variable *x* is d-dimensional, we write

$$ln(\frac{pi}{1-pi}) = w_0 + w_1 * x_1 + \cdots + w_d * x_d$$

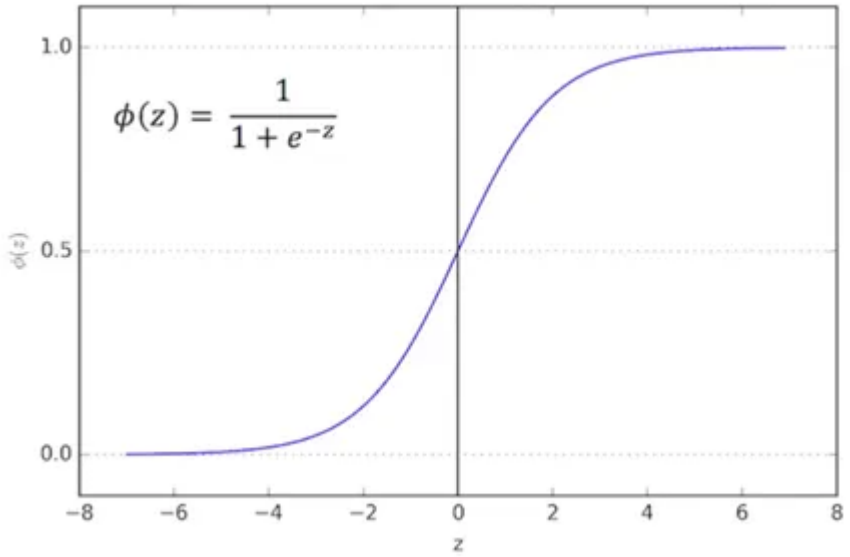
In vector form,

$$ln(\frac{pi}{1-pi}) = w^T x$$

Rearranging the equation, we get –

$$pi = \frac{1}{1 + exp(-w^T x)}$$

The last function is called the logit (or sigmoid) function. The logit function looks like the below graph –



It is clear from the graph that the range of the logit function is [0, 1], that is the model outputs a number in the range of [0, 1] which is interpreted as the probability of the email being spam as discussed earlier. Once we get the probability we pass it through a decision boundary which will divide the outcome into classes. Generally, we define the decision boundary at 0.5, so the outcome will be

classified as spam if pi >= 0.5, else it will be classified as non-spam. However, you can define your own decision rule depending on the use case.

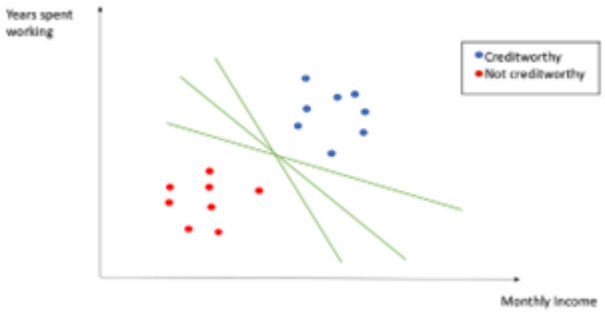
Further details:
You can learn more about logistic regression and its implementation here –
https://www.analyticsvidhya.com/blog/2021/08/conceptual-understanding-of-logistic-regression-for-data-science-beginners/

Support Vector Machine

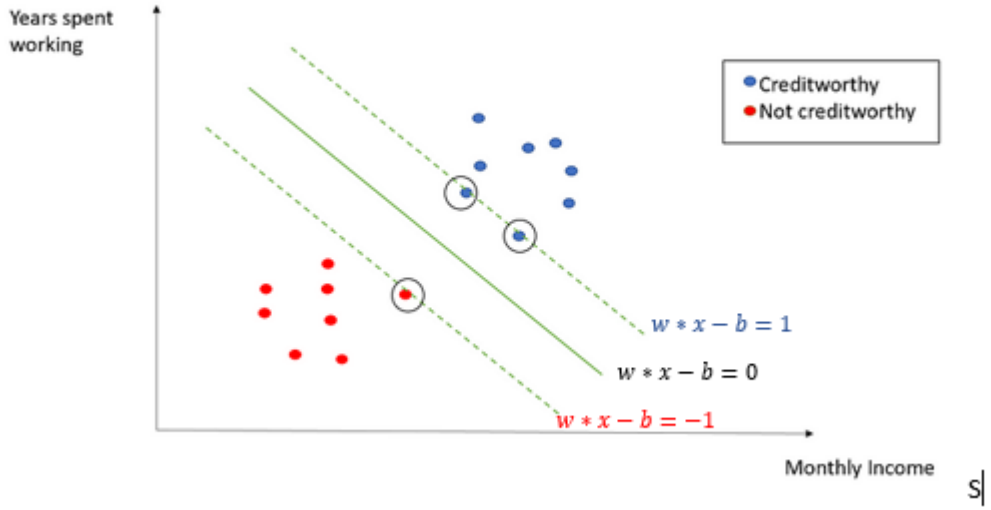
The Support Vector Machine (SVM) is also a learning algorithm that is primarily used for classification. The SVM finds to find the best separation plane between the two classes.

For example, look at the training data below:

We can draw multiple boundaries to separate these two classes.



However, we prefer a line with larger margin as shown below as it has lesser room for misclassification.



The observations encircled in the above image are called **support vectors** and hence the name, support vector machine. The task here is to find the maximum-margin hyperplane that divides the group of points into separating classes.

We are given a training dataset of n points of the form $(x_1,y_1),\ldots,(x_n,y_n)$ where y_i are either 1 or -1, each indicating the class to which point x_i belongs. We can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the margin, and the maximum-margin hyperplane is the hyperplane that lies halfway between them. These hyperplanes can be described by the equations:

$$w^T x - b = 1 (\text{Anything on or above this boundary with label } 1)$$

$$w^T x - b = -1 (\text{Anything on or below this boundary with label -1})$$

The distance between the two hyperplanes is $\frac{2}{\|w\|}$. So, to maximize the distance between the two planes we minimize the $\|w\|$.

The data in this toy example could be separated by a single line. In most real-world cases, this is not possible. In such cases, we can use a technique called the kernel trick. However, we will not cover it in this brief introduction.

Implementation

A sample implementation of a Linear SVM can be found here -
https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-python/

Decision Trees, Random Forests

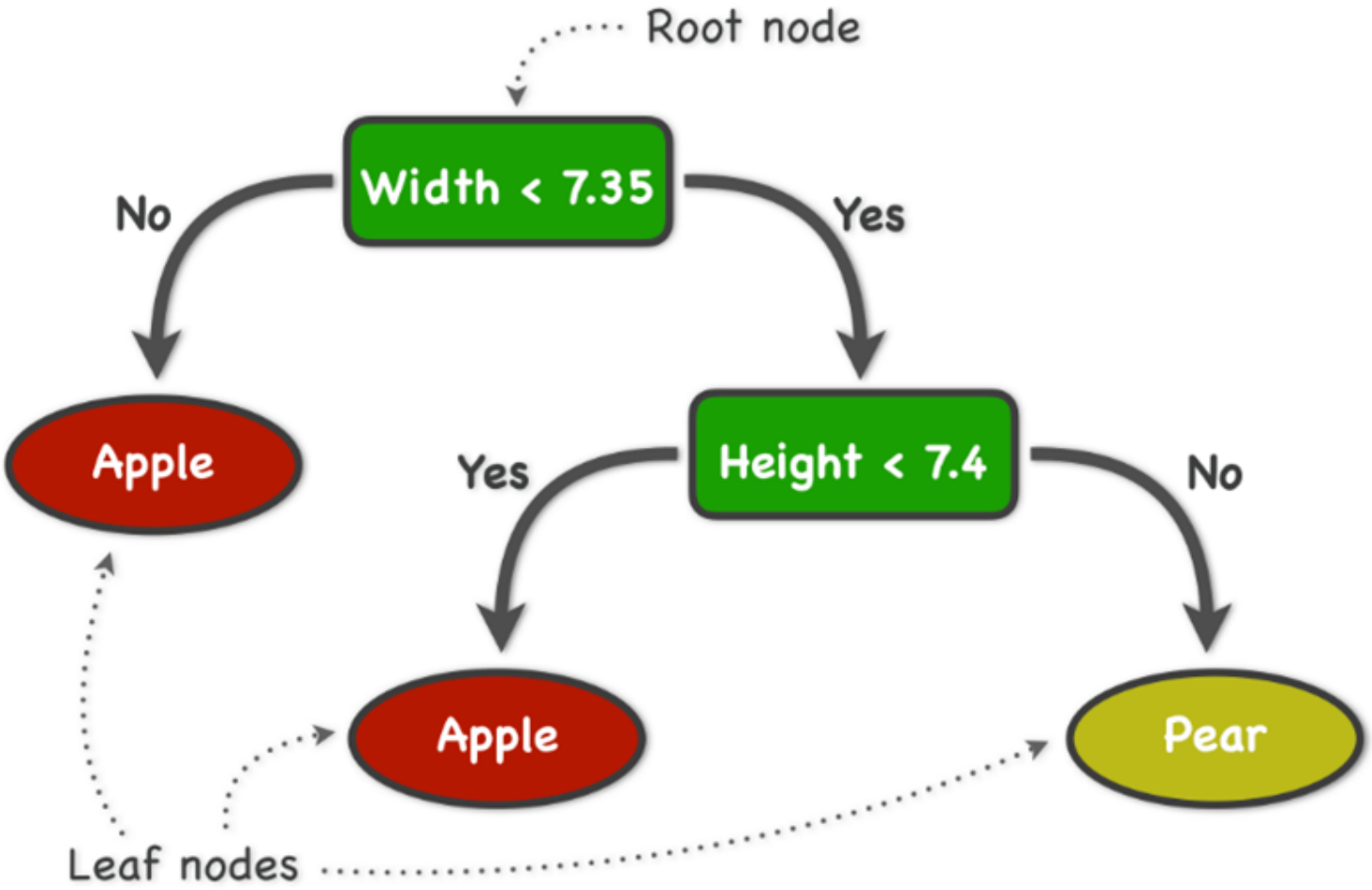
Decision trees and random forests are supervised learning algorithms that are used in a variety of machine learning tasks. We will have a brief look at these algorithms in the following sections -

Decision Trees

It is easier to understand the decision tree algorithm in a classification framework, even though it can be used in other problems as well owing to its flexibility.
Suppose we are given a task to classify fruits into 'apples' and 'pears', when we are only given the data about their width and height. This is the data that we already have based on our observations.

Width	Height	Fruit
7.1	7.3	Apple
7.9	7.5	Apple
7.4	7.0	Apple
8.2	7.3	Apple
7.6	6.9	Apple
7.8	8.0	Apple
7.0	7.5	Pear
7.1	7.9	Pear
6.8	8.0	Pear
6.6	7.7	Pear
7.3	8.2	Pear
7.2	7.9	Pear

Intuitively, apples are shorter and fatter, whereas pears are longer and thinner. Based on this knowledge, we can classify a new fruit into 'apples' or 'pears' by checking a series of conditions such as this.

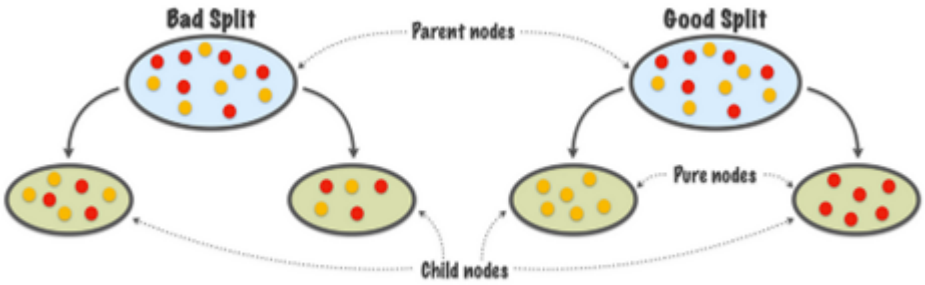


This is an example of a decision tree. The leaves of the tree correspond to the classes in the observed data.

We can use a greedy algorithm to build this tree where at every step, we choose the best split for the data and then iteratively split the data until we arrive at a pre-decided stopping condition. The challenge is to find a way to find the best split.

Finding the Best Split – Entropy

Let us see what kind of split is favorable for us. In the image below, suppose we have two classes of objects, one red and other yellow. The split on the right completely separates the red and yellow classes and therefore, it is more favorable.



Intuitively, the 'purer' the child nodes are, the better the split is for us. A popular choice for measuring 'purity' is a numerical quantity called **entropy**. It is measured by the formula:

$$E = -\sum_{i=1}^{n} p_i \, log_2(p_i)$$

Implementation

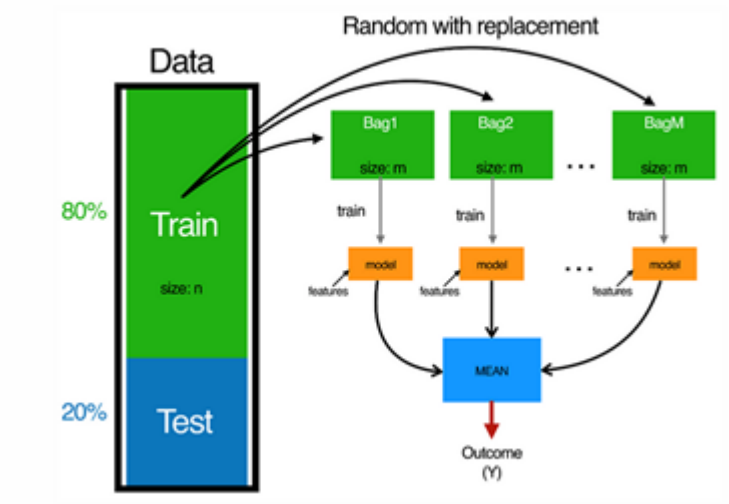
You can have a look at the following link for a sample implementation - https://medium.com/codex/building-and-visualizing-decision-tree-in-python-2cfaafd8e1bb

Random Forest

The **Random Forest** algorithm is an algorithm that uses the decision tree as a blackbox. As discussed earlier, decision trees suffer from high variance as they are prone to overfitting the data.

Bagging

A technique called **bagging** can be used to reduce the variance of our model. A model that suffers from high variance can change a lot even if one of the training data is altered. So, we create multiple training datasets (called bags) by sampling at random with replacement from our original training dataset.

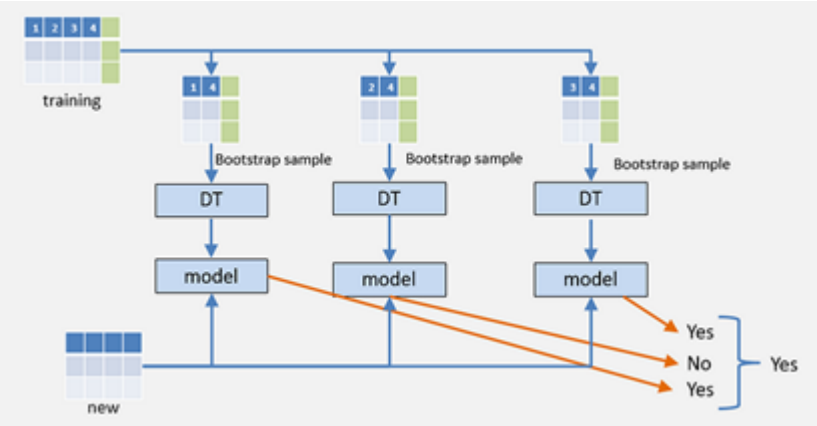


We fit different models on all these bags and finally, take an average of all the model outputs to arrive at the final model output.

Bagging on Decision Trees

While applying the bagging technique on decision trees, we make one more enhancement. In addition to randomly sampling the observations, we also randomly select a subset of the features in each bag as well.

We do so to avoid the correlation between the trees. Suppose that we have an extraordinarily strong predictor in the data set along with several other moderately strong predictors, then in the collection of bagged trees, most or all our decision trees will use the extraordinarily strong predictor for the first split! All bagged trees will look similar. Hence all the predictions from the bagged trees will be highly correlated.



Implementation

A sample implementation can be found at this link - https://medium.com/@harshdeepsingh_35448/understanding-random-forests-aa0ccecdbbbb

Problem Statement - ML-based OCR

ML techniques can be employed in different stages of OCR, such as image preprocessing, text localization, character segmentation, and character recognition.

For our case, the training phase involves feeding the algorithms with labeled examples of characters, allowing them to learn the relationships and features that distinguish different characters. During the recognition phase, the ML models apply the learned knowledge to recognize and classify characters in new inputs.

The given dataset contains 1000 images and each image contains 5 characters which are all alphabets. Our objective is to split the image into the characters and do character recognition. In the end, using the trained model we will try to predict characters in different images entirely and thereby checking the robustness of our system in recognition phase.

Exercise

1. From the folder of 1000 images, load the images. You can use image preprocessing techniques (As given in the above example, you can perform erosion/ dilation or any other techniques) to remove the noise from the image.
2. Once, all the images have been preprocessed, notice that in each image, each character lies in a window of width of 20 pixels and height of 100 pixels. Split each image into 5 character images. Now, the dataset contains 5000 images. Create a training and validation set from these images. The train set should roughly contain 80% of the data and validation set the rest 20% of data.
3. As mentioned above, we want to identify the characters from the image. For accomplishing this, train the model using these three ML algorithms - Logistic Regression, Support Vector Machine, Random Forest for the classification task. For the above mentioned algorithms, answer the questions below based on the validation data.

Questions:

After the model is fit, answer the below questions:

1. Which of the above classifiers yield the best accuracy on the validation set? Do you think accuracy is the best metric to compare the algorithms, if not - which other metric could be used? [3 points]

2. Tweak the hyperparameters of the classifier selected in Q1 and try to improve the accuracy. What hyperparameters did you change, why do you think it worked? [5 points]

3. Find Precision, Recall and F1 score in all cases. [2 points]

4. a. Find the confusion matrix using the best classifier. [2 points]
   b. Which characters have low accuracy? [3 points]
   c. What can be the reason for this? [3 points]

5. Try any other techniques/algorithms in your research that could improve the accuracy. [7 points]

Note: In All Questions, we are doing classification and finding all the metrics for the individual characters (not the entire images). Please submit the answers to these questions along with the code in the ipynb notebook.