1. Algorithm Explanation

KNN Algorithm

Implementation of a KNN classifier can be described with the steps given below:

1. Load the data
2. Initialise the value of k
3. For every test data, iterate from 1 to the total number of training data points
    a. Calculate the distance between test data and each row of training data.
        i. Euclidean Distance between two points **x** = $(x_1,....,x_n)$ and **y** = $(y_1,....,y_n)$ is :

$$d = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} .$$

        ii. Cosine distance = 1- cosine similarity. Similarity between two points **A** and **B** is

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

    b. Sort the calculated distances in ascending order based on distance values
    c. Get top k rows from the sorted array
    d. Get the most frequent class of these rows
    e. Return the predicted class

Naive Bayes Classifier

Implementation of a Naive Bayes classifier can be described with the steps given below:

1. Read the training dataset and separate by class
2. Calculate the mean and standard deviation (μ_k, σ_k) for each attribute in each class C_k.
3. For all the attribute variables ($x_1,.....x_n$) of test data **X**, Calculate the probability of x_i using gaussian density equation in each class.

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

4. Calculate the likelihood for each class and return the class with greatest likelihood.

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \prod_{i=1}^{n} p(x_i \mid C_k).$$

Cross validation test

In our program, we have trained the algorithm using 5 fold cross-validation. For any k-fold cross-validation, the general procedure is

- Shuffle the dataset randomly.
- Split the dataset into k groups
- For each unique group:
    - Take one of the group as test data set
    - Take the remaining (k-1) groups as a training data set
    - Fit a model on the training set and evaluate it on the test set
    - Retain the evaluation score and discard the model
    - Average the accuracy and error obtained for k different experiments

2.Results
KNN Classifier

The table shows the experimental result of average accuracy over 5 fold cross validation on varying parameter k on two different distance metrics (Euclidean and Cosine). We can observe that the accuracy of classifier is better for large value of k i.e. k = 15 on both distance metrics whereas for small value of k, the performance degrades in both cases.

Metric	Accuracy		
	K = 1	K = 5	K = 15
Cosine distance	96.0	96.66	97.33
Euclidean distance	95.33	96.0	97.33

Table1. Evaluation of KNN classifier for different k and distance metric

The best performing confusion matrix for model of KNN classifier for 'iris' dataset is observed with K=15 and distance metric=Euclidean as shown in Fig1. We can see that 8% of the test samples which belonged to class 'Iris-virginica' was mis-classified as 'Iris-veriscolor'.

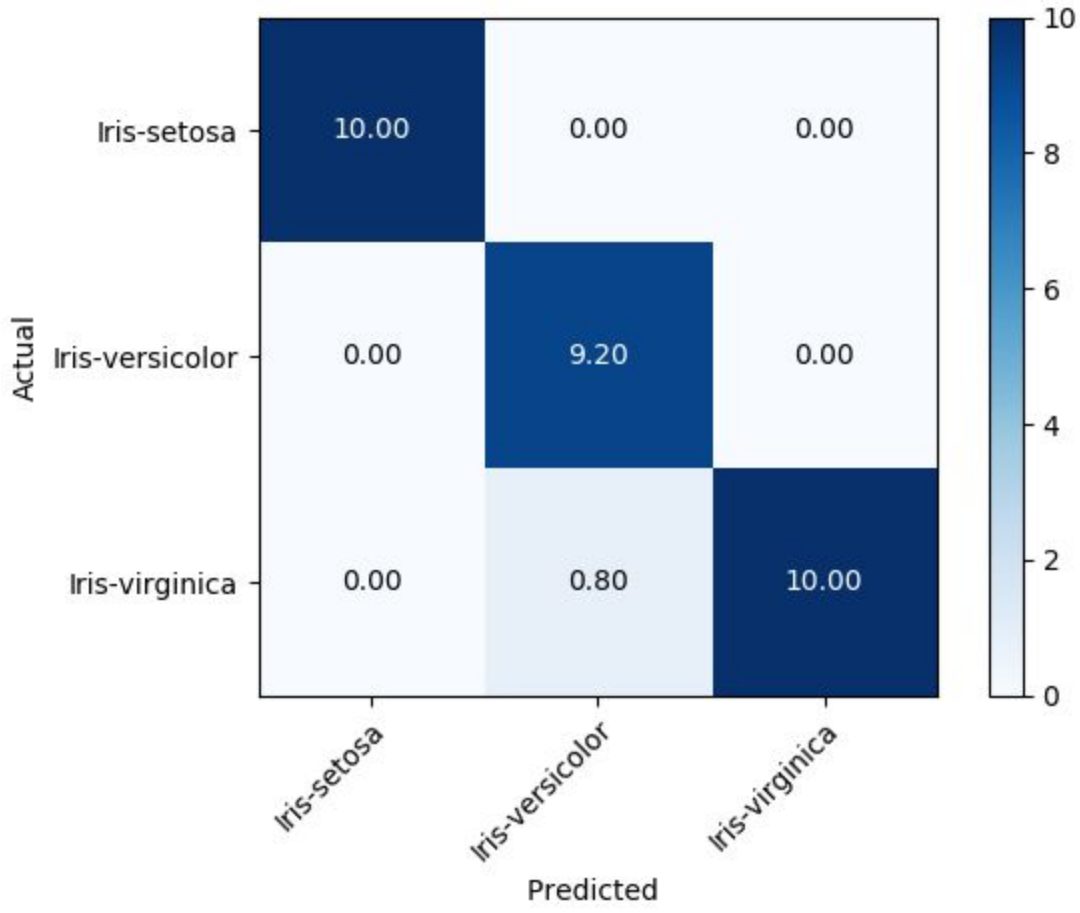

Fig1: Confusion Matrix for KNN classifier with parameters: K=15 and Distance Metric= Euclidean

Naive Bayes Classifier

Using Naive Bayes classifier, we were able to predict the classes of test dataset with an average accuracy of 95.33 which is slightly less than than the KNN classifier. The best performing confusion matrix for this classifier is shown in Fig2, where we can see that 8% of the test data which belonged to class ‘Iris-veriscolor’ was mis-classified as ‘Iris-virginica and 6% of the data that belonged to class ‘Iris-virginica’ was mis-classified as ‘Iris-versicolor’.

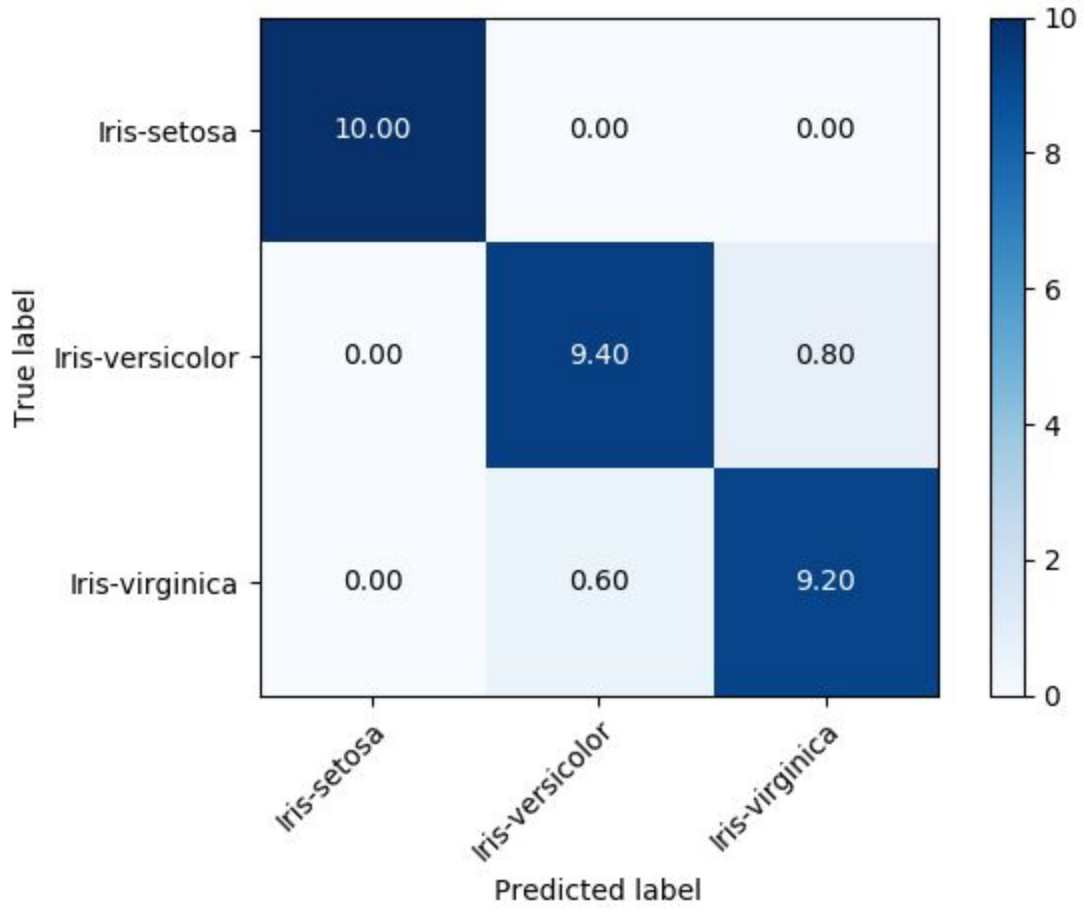

Fig 2: Confusion Matrix for Naive Bayes Classifier

Q1. Explain the difference between MLE and MAP procedures for naive Bayes (e.g. estimating the class priors).

If we have an instance to be classified, represented by a vector **X** = $(x_1,.....x_n)$, representing some n features (independent variables), it assigns to this instance probabilities

$$p(C_k \mid x_1, \ldots, x_n)$$

for each of K *classes* C_k.

Using Bayes' theorem, the conditional probability can be decomposed as

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

the above equation can be written as:

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

MAP or Maximum A Posteriori Estimation works on the posterior distribution instead of the likelihood like MLE does. This means MAP maximizes the posterior probability (likelihood times prior probability) .

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k).$$

But, MLE maximizes only likelihood as:

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}}\ \prod_{i=1}^{n} p(x_i \mid C_k).$$

More precisely, MLE attempts to find the value of $\boldsymbol{C_k}$ that maximizes $p(\mathbf{x}|\ \boldsymbol{C_k})$, while MAP estimation attempts to find the value of $\boldsymbol{C_k}$ that maximizes $p(\mathbf{x}|\ \boldsymbol{C_k}) * p(\boldsymbol{C_k})$.

Q2. In other words, if you had to implement this using MAP how would your code have been different?

For implementation of MAP in our MLE code, we just need to multiply the prior probability after computing the likelihood for each class, this then gives the probability of that class.

Q3. Identify one case in which MLE and MAP would produce the same/similar answers.

In conditions where the prior probability is constant (e.g. rolling a die) then MAP happens to be similar as MLE. Therefore, MLE can be seen as a special case of MAP where the prior probability is constant.

Instructions to run the program

1. Install the following required dependencies
   ```
   $ sudo apt-get install python3-pip
   $ pip3 install numpy
   $ pip3 install matplotlib
   ```
2. Unzip the file “4805322.zip”
   ```
   $ sudo apt-get install unzip
   $ unzip 4805322.zip
   ```
3. From inside the directory, run the python files
   ```
   $ python3 knn_4805322.py -k 5 -m cosine
   $ python3 byes_4805322.py
   ```

References:

[1] https://en.wikipedia.org/wiki/Naive_Bayes_classifier
[2] https://en.wikipedia.org/wiki/Cosine_similarity
[3] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm