Assignment 1
Foundations of Machine Learning

Name: Ahmik Virani
Roll Number: ES22BTECH11001

Contents

1 Question 1: k-NN

1.1 Part (a): Training Error vs. Neighbor Size

- **Question:** Describe what happens to the training error (using all available data) when the neighbor size k varies from n to 1.

- **Answer:** When the value of k is equal to n, for any input test, the model will just predict the class which appears most often, not considering which points are closer to this test data. And even during training, the model will always predict the value which appears most regardless of any other factor, leading to high training error. Any training instance belonging to the minority class will be misclassified.

  In contrary, if the value of k equals 1, the model basically memorizes. Now here, it could be ambiguous, if you consider that each training point can be considered its own neighbour, then the training error will be zero. However if you assume that this is not the case, then any points that are outliers, as shown in the fugues below, any training data that is closer to them will be predicted as the outlier itself. . Generally as k decreases, the decision boundary gets more and more complex. In some value between 1 and n, you would reach an optimal value of k which gives the best results for that particular training set. Basically, the training error generally decreases from k = n to k = 1. (Note that when you think of the testing error, then an optimum is reached somewhere in the middle).

  I have attached some photos of kNN classification on a dataset from the internet varying the values of k.

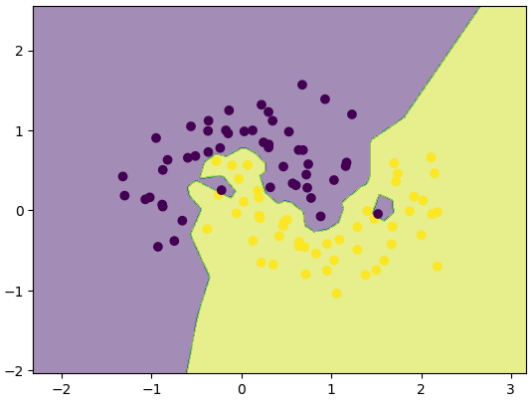

Figure 1: Decision boundary for $k = 1$.

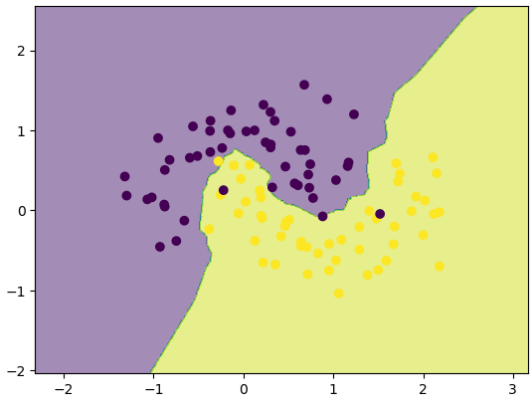

Figure 2: Decision boundary for $k = 5$.

–

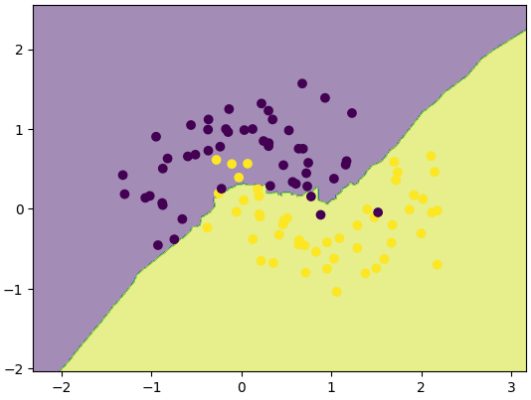

Figure 3: Decision boundary for $k = 30$.

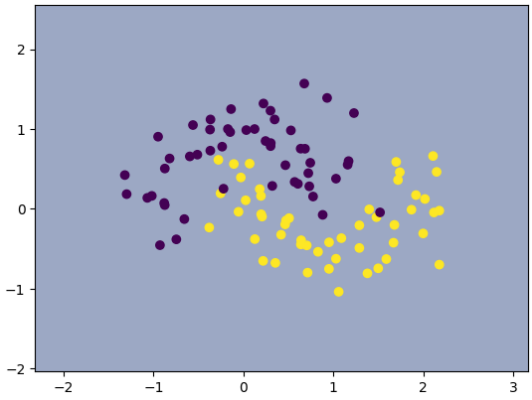

Figure 4: Decision boundary for $k = n$.

–

1.2 Part (b): Generalization Error

- **Question:** Predict and explain with a sketch how the generalization error would change when k varies? Explain your reasoning.

- **Answer:** Generalization Error refers to how well a machine learning model performs on new, unseen data. It is the difference between the model's performance on the training data and its performance on unseen data. In other words, it reflects how well the model has learned the patterns in the data rather than just memorizing the training set. The generalization error is typically estimated by evaluating the model on a separate test set or through techniques like cross-validation.

  As k is small, i.e. closer to 1, it is a case of the model memorizing the dataset. It is a case of over fitting, therefore a new unseen data will have high generalization error in such a case. When k is closer to n, the model may underfit, assigning the majority class to any new points, which can also lead poor generalization error. Whereas as k increases and approaches some optimal value of k for the particular training set, this generalization error decreases, however at the same time the training error increases as k increases. At the point of local optimum of generalization error (where the model balances bias and variance), we propose that corresponding k as the 'best' value for our model.

  Attached below are two images, the first one I have plotted on a dataset from the internet. The second one is a hand sketch. Generally, the curve is a 'U' shape, with generalization error first decreasing from k = 1 to some optimum value of k, then again increasing as k increases.

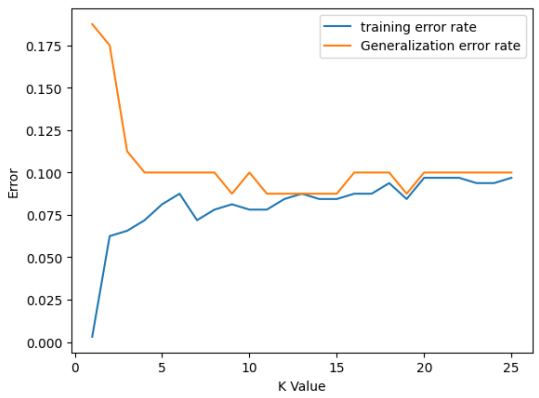

Figure 5: Generalization error and Training error v/s k

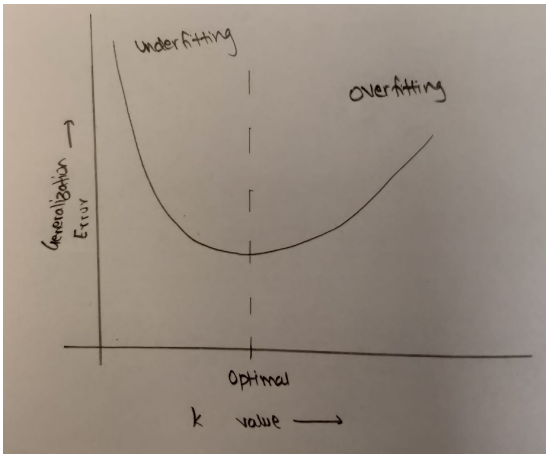

Figure 6: Generalization error v/s k Hand sketch

–

1.3 Part (c): Decision Tree Comparison

- **Question:** Is it possible to build a univariate decision tree that classifies exactly similar to a 1-NN using the Euclidean distance measure? If so, explain how. If not, explain why not.
- **Answer:** We know that for decision tree, the decision boundaries, i.e. the querying conditions will be lines that are parallel to the coordinate axes. However the same in not true regarding the kNN classifier. In a 1-NN classifier, the decision boundaries correspond to a Voronoi diagram, which divides the space into regions based on the distance to the nearest training point. These boundaries are generally not axis-aligned and can take any shape based on the data points.

  Below are the images attached:

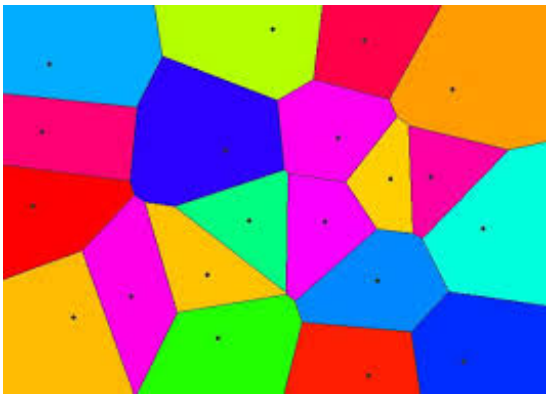

Figure 7: Vernoi diagram

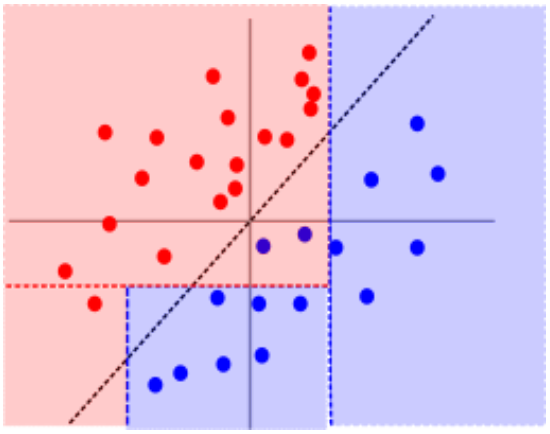

Figure 8: Decision Tree and kNN models

–

As we can clearly see, the models are quite different by nature. The stair case like graph corresponds to the decision tree, whereas the other one to KNN. Therefore, the nature of these models is fundamentally different.

Note that it is not impossible however, to have the same model. Take a 'toy' example as follows.

Data set with only two points (0,0), (2,0), each belonging to a different class. The decision boundary of a 1-NN will be the perpendicular bisector of the segment. And it will be the same for decision tree too.

In more complex, higher-dimensional datasets, the differences between the two models become more clear. A univariate decision tree, which can only create axis-aligned decision boundaries, cannot replicate the non-axis-aligned boundaries of a 1-NN model using the Euclidean distance measure. Thus, while it is possible in trivial cases, it is generally not feasible for more complex data.

Image Credits: https://stats.stackexchange.com/questions/262930/can-a-decision-tree-recreate-the-exact-same-classification-as-a-nearest-neighbor

2 Question 2: Bayes Classifier

2.1 Part (a): Gaussian Fit and Probability Estimation

- **Question:** Fit a Gaussian using Maximum Likelihood to each of the two classes and estimate the class probabilities. What is the probability that the test point $x = 0.6$ belongs to class 1?

- **Answer:** Given the training examples:

  - Class 1: {0.5, 0.1, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.35, 0.25}
  - Class 2: {0.9, 0.8, 0.75, 1.0}

  Step 1: Compute the Mean for Each Class

  For Class 1, mean μ_1 is calculated as:

$$\mu_1 = \frac{1}{10}\sum_{i=1}^{10} x_i = \frac{0.5 + 0.1 + 0.2 + 0.4 + 0.3 + 0.2 + 0.2 + 0.1 + 0.35 + 0.25}{10} = 0.26$$

  For Class 2, mean μ_2 is calculated as:

$$\mu_2 = \frac{1}{4}\sum_{i=1}^{4} x_i = \frac{0.9 + 0.8 + 0.75 + 1.0}{4} = 0.8625$$

  Step 2: Estimate Class Probabilities p_1 and p_2

  The class probabilities are the relative frequencies of the classes in the training set:

$$p_1 = \frac{n_1}{n_1 + n_2} = \frac{10}{14} \approx 0.714$$
$$p_2 = \frac{n_2}{n_1 + n_2} = \frac{4}{14} \approx 0.286$$

  Step 3: Calculate the Probability That $x = 0.6$ Belongs to Class 1

  We use Bayes' theorem:

$$P(C_1 \mid x = 0.6) = \frac{P(x = 0.6 \mid C_1) \cdot P(C_1)}{P(x = 0.6 \mid C_1) \cdot P(C_1) + P(x = 0.6 \mid C_2) \cdot P(C_2)}$$

  Where the likelihoods $P(x \mid C_i)$ are calculated using the Gaussian distribution:

$$P(x \mid C_i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x - \mu_i)^2}{2\sigma_i^2}\right)$$

  Given:

$$\sigma_1^2 = 0.0149, \quad \sigma_2^2 = 0.0092$$

$$P(x \mid C_1) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) \approx 0.0675$$

$$P(x \mid C_2) = \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right) \approx 0.0983$$

Using these terms in the above formula, the probability that $x = 0.6$ belongs to Class 1 is:

$$P(C_1 \mid x = 0.6) \approx 0.632$$

2.2 Part (b): Text Classifier

- **Question:** Using a maximum likelihood naive Bayes classifier, what is the probability that the document $x = (1, 0, 0, 1, 1, 1, 1, 0)$ is about politics?
- **Answer:** To calculate the probability that the document $x = (1, 0, 0, 1, 1, 1, 1, 0)$ is about politics using a maximum likelihood naive Bayes classifier, follow these steps:

  Step 1: Calculate Prior Probabilities

$$P(\text{politics}) = \frac{\text{number of politics documents}}{\text{total number of documents}} = \frac{6}{12} = \frac{1}{2}$$
$$P(\text{sport}) = \frac{\text{number of sport documents}}{\text{total number of documents}} = \frac{6}{12} = \frac{1}{2}$$

  Step 2: Calculate Likelihoods

  For each word in x, calculate the likelihood $P(x_i = 1 \mid \text{politics})$ and $P(x_i = 0 \mid \text{politics})$, where x_i is the i-th attribute in x. The training data for politics documents is:

$$x_{\text{politics}} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$x_{\text{sport}} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

  Calculate the likelihoods:

$$P(x_1 = 1 \mid \text{politics}) = \frac{2}{6} = \frac{1}{3}, \quad P(x_2 = 0 \mid \text{politics}) = \frac{5}{6}, \quad P(x_3 = 0 \mid \text{politics}) = \frac{5}{6}$$

$$P(x_4 = 1 \mid \text{politics}) = \frac{5}{6}, \quad P(x_5 = 1 \mid \text{politics}) = \frac{5}{6}, \quad P(x_6 = 1 \mid \text{politics}) = \frac{1}{6}$$

$$P(x_7 = 1 \mid \text{politics}) = \frac{4}{6} = \frac{2}{3}, \quad P(x_8 = 0 \mid \text{politics}) = \frac{1}{6}$$

Therefore,

$$P(x = (1,0,0,1,1,1,1,0) \mid \text{politics}) = \frac{1}{3} \times \frac{5}{6} \times \frac{5}{6} \times \frac{5}{6} \times \frac{5}{6} \times \frac{1}{6} \times \frac{2}{3} \times \frac{1}{6} = \frac{625}{209952}$$

Step 3: Calculate the Total Probability $P(x)$

Assuming that the probabilities are independent:

We need to find:

$$P(x_1 = 1) = \frac{6}{12} = \frac{1}{2}, \quad P(x_2 = 0) = \frac{7}{12}, \quad P(x_3 = 0) = \frac{10}{12} = \frac{5}{6}$$

$$P(x_4 = 1) = \frac{9}{12}, = \frac{3}{4} \quad P(x_5 = 1) = \frac{6}{12}, = \frac{1}{2} \quad P(x_6 = 1) = \frac{2}{12} = \frac{1}{6}$$

$$P(x_7 = 1) = \frac{4}{12} = \frac{1}{3}, \quad P(x_8 = 0) = \frac{6}{12} = \frac{1}{2}$$

$$P(x) = P(x_1 = 1) \cdot P(x_2 = 0) \cdot P(x_3 = 0) \cdot P(x_4 = 1) \cdot P(x_5 = 1) \cdot P(x_6 = 1) \cdot P(x_7 = 1) \cdot P(x_8 = 0)$$

$$P(x) = \frac{1}{2} \cdot \frac{7}{12} \cdot \frac{5}{6} \cdot \frac{3}{4} \cdot \frac{1}{2} \cdot \frac{1}{6} \cdot \frac{1}{3} \cdot \frac{1}{2}$$

$$P(x) = \frac{35}{13824}$$

Step 4: Calculate Posterior Probability for Politics

Finally, we calculate:

$$P(\text{politics} \mid x) = \frac{P(x \mid \text{politics}) \times P(\text{politics})}{P(x)} = \frac{\frac{625}{209952} \times \frac{1}{2}}{\frac{35}{13824}} = \frac{1000}{1701}$$

Therefore, the probability that the document $x = (1,0,0,1,1,1,1,0)$ is about politics is ≈ 0.5879.

- Please note that above I have assumed independence. However, if we use conditional independence on the same problem, the value we would get for $P(x)$ is:

$$P(x) = P(x \mid \text{politics}) \times P(\text{politics})$$

Note that the term relating to sport becomes zero because:

$$P(x_7 = 1 \mid \text{sport}) = 0$$

Therefore, the numerator and denominator are equal, and the answer equals 1.

– If we assume **independence**, we get the answer 0.5879.
– If we assume **conditional independence**, we get the answer 1.00.

3 Question 3: Decision Tree Report

3.1 Part (a): Decision Tree Implementation

- **Question:** Implement your own version of the decision tree using binary univariate split, entropy, and information gain.
- **Answer:** The libraries I have used are: numpy, pandas, collections.Counter from collections module, sklearn.metrics.accuracy_score, sklearn.model_selection.train_test_split from scikit-learn

  The implementation I have used is similar to the one discussed in class, where I recursively grow the tree finding the split corresponding to the features which gives best information gain. Note that along with the submission, I have attached a photo which I have made of the decision tree. In this implementation I have done no pruning and let the tree grow on its own. Clearly we can count that here the depth of the tree is 26 from the image.

  I have used the train test split library to train the model here and randomly split the data into training set and testing set. Note that this module will randomly generate the results. I have used test set size as 10% so we can compare it with the cross validation in next part.

  The accuracy I have reported as of now (subject to change) is 0.8554

3.2 Part (b): Cross-Validation

- **Question:** Evaluate your decision tree using 10-fold cross-validation.
- **Answer:** For Cross Validation, I am randomizing the data set. To do this, I am first creating an array of all the indexes from 1 to size of data set. Then using the permutation function in numpy, I am randomly redistributing the data set. Next, because we are doing 10 fold cross validation, I am diving this data in 10 parts. Note that, all parts are of size: round down size of data set divided by 10. However, the data set size may not be divisible by 10. So, the last set will just contain all the remaining data.

  For the validation process, I am using one part as test data and the remaining as the train data, and finding the accuracy on the test data.

  I am reporting the average accuracy: 0.8462

  Also, the maximum accuracy: 0.8582

  Clearly, as we saw from part (a), the maximum accuracy is more than that reported from part (a), as it should have been.

3.3 Part (c): Improvement Strategies

- **Question:** Report your performance as an outcome of any two improved strategies.
- **Answer:** I am using 2 strategies:
  - **Gini Index**

    For Gini Index Case, I am just changing the entropy formula to gini index formula. Running this model on 10 fold cross validation, the mean accuracy I am getting is 0.8456, and the max is 0.8690. This seems that on average we get a lower value, but on the best case, we are getting a better result.

The possible reasons for the observations above are as follows: Gini tends to produce slightly different splits, sometimes preferring more balanced splits in the tree. This can lead to more conservative tree structures that slightly under perform on average. And gini index tends to be less sensitive to outliers compared to entropy, therefore being able to generalize better.

- **Pruning** Also, I have also used pruning. Here, I have run the data across various depths and seen that at depth, 23 the best result is coming. The possible reason could be that, when we let the tree grow, which currently has height 26, the tree overfits, and reducing some nodes at the end helps in generalization.

  Note the results: 0.8517 at depth 23, which is higher than the average of both entropy and gini index case.

Note: I have taken inference from the video https://www.youtube.com/watch?v=NxEHSAfFlK8 to build my decision tree. I have made changes and improvements as per the problem statement on my own.

4 Question 4: Method Comparison

4.1 Part (a): Computational Complexity

- **Question:** Analyze and compare the computational complexity of kNN, decision trees, and Naive Bayes classifiers during training and inference.

- **Answer:**

  For kNN, training data is equivalent to just storing the data basically. Which means, that each piece of data will take $O(1)$ time, and training a set of n data pieces of dimension d, will take $O(n \cdot d)$ time. Note, here we are storing the nodes, which may be of the order of number of training set. Which equals O(n).

  On the contrary, for testing, first we need to find the distance between the new data point with all the points, which will take $O(n \cdot d)$ time. Next we need to sort this data, taking $O(n \cdot \log(n))$ time. Then we need to extract the top k distances and check the class that occurs the most, taking $O(k)$ time. Thus overall time for testing one data point is $O(k + n \cdot \log(n) + n \cdot d)$. If we are testing a set of m data points, then it will take $O(m \cdot (k + n \cdot \log(n) + n \cdot d))$ time. Note, we just need an additional data structure to store the distance, so requires an additional O(n) space.

  For kNN, training is fast, however testing is much more time consuming. And the space required is just for storing the amount of data provided.

  For decision tree, the training time is $O(d \cdot n \cdot \log(n))$, where n is the number of points in the training set, and d is the dimensionality. The $n \cdot \log(n)$ term is for sorting the n numbers in the data set. And we multiply with dimensionality because we need to test for each parameter before choosing the apt one at that stage.

  After training the data, all you store are the queries (the nested if-else conditions). The space complexity it takes is equal to the number of nodes. Say we store k decisions, which means k is the max depth of the decision tree. Thus, the testing complexity is $O(k)$ i.e. O(depth of tree). This is because at most we need to check every decision before coming to the conclusion. Generally, $k \approx$ order of log(nodes of tree). While testing, you just need to traverse the tree checking the queries of the nodes. So no additional space is required.

  For decision trees, training is slow, however testing is fast. Also, storage is not too high.

  For Naive Bayes Classifier, we don't really do much in training except store the data. Assuming storing each piece of data takes $O(1)$ time, storing n sets of dimension d will take $O(n \cdot d)$ time. Note, that the same complexity, similarly is also $O(n \cdot d)$.

  For testing the data, we perform probability checks. We know that multiplication is very fast and can assume it takes $O(1)$ time per multiplication operation. Also, say we need to test some data for the class, we need to find its probability of occurrence in all classes, say we have k classes, and for each class we need to find the probability by iterating through the features, it takes $O(k \cdot d)$ time. Similarly, we just need to store this, $O(k \cdot d)$ and come to the conclusion.

  Finally, for Naive Bayes, training is just storing the data, while testing is checking all probabilities. Therefore, it is more costly to test. Space complexity if just storing the entire dataset.

4.2 Part (b): Strengths and Weaknesses

- **Question:** Discuss the strengths and weaknesses of each method (KNN, NBC, DT) in different dataset settings.
- **Answer:**
  - **KNN:**
    - Generally fast to train and slow for testing
    - Requires storing all the data, leading to high space complexity
    - If the dataset is very large and dimensionality is low, $O(n \cdot \log(n))$ dominates and becomes computationally heavy
    - If the dimensionality is high, $O(n \cdot d)$ dominates, making it even more computationally heavy
    - **Strengths:**
      - Simple to understand and implement
      - No explicit training phase, the model just remembers the data set, and adapts quickly
      - Not to difficult to find an optimal value of K for the given training set.
    - **Weaknesses:**
      - Computationally expensive at inference time, especially with large datasets
      - Performance can degrade with high-dimensional data, as can be seen by the time complexity
      - Need to normalize the data into a constant scale before applying this method.
  - **Naive Bayes:**
    - Generally fast to train and test
    - Requires storing the training data and computed probabilities, which can take up significant space, especially with many features and classes
    - Computational complexity increases with the number of classes and features as we need to compute probabilities for each class and feature
    - **Strengths:**
      - Requires a small amount of training data to estimate parameters (probabilities)
      - Works well with categorical features and in cases with a large number of features relative to the number of training examples
      - Fast at inference time due to its straightforward probabilistic model
    - **Weaknesses:**
      - Assumes feature independence, which is not always true in real-world data, so may not give optimal results.
      - May perform poorly with highly correlated features
      - Requires large amounts of data to estimate probabilities accurately for many features and classes
  - **Decision Tree:**
    - Generally slow to train, especially with large datasets and high dimensionality, because we need to find optimal splits at each node

* Requires significant memory for storing the structure of the tree, which can become large with complex trees
* Testing (inference) is fast because it involves traversing the tree from the root to a leaf node
* **Strengths:**
  - Decision rules are clearly represented in the tree structure
  - Can handle both numerical and categorical data
  - No need for feature scaling or normalization
  - Automatically handles missing values and can provide feature importance measures
* **Weaknesses:**
  - Prone to overfitting, especially with very deep trees or when there is noise in the data
  - Can be unstable: small changes in the data can result in a completely different tree structure
  - Decision boundaries created by trees are axis-aligned, which may not be suitable for some types of data
  - Requires careful pruning and tuning to avoid overfitting and improve generalization