## Introduction

The Naive Bayes algorithm is one of the simplest and most effective classifiers used in machine learning. It is based on Bayes' theorem, which calculates the probability of a hypothesis based on prior knowledge. What makes Naive Bayes unique is the assumption that all features are independent of each other, given the class label. While this "naive" assumption may not always hold in real-world data, the algorithm performs surprisingly well in various applications, such as spam detection, sentiment analysis, and medical diagnosis.

The main objective of this assignment was to implement the Naive Bayes algorithm from scratch to classify data from a dataset called play_tennis.csv. This dataset records weather conditions and whether or not a game of tennis was played. The task involved calculating probabilities, making predictions, and evaluating the model's performance using metrics such as accuracy and a confusion matrix.

Through this assignment, I aimed to deepen my understanding of probabilistic classifiers and gain hands-on experience in implementing them without relying on machine learning libraries like scikit-learn.

## Methodology

### Data Preparation

The dataset, play_tennis.csv, contains categorical features: *Outlook*, *Temperature*, *Humidity*, and *Wind*, along with the target variable *PlayTennis* (values: "Yes" or "No"). Data was loaded using pandas, and the feature columns (*X*) and the target column (*y*) were separated.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Outlook | Temperatı | Humidity | Wind | PlayTenis |
| 2 | Sunny | Hot | High | Weak | No |
| 3 | Sunny | Hot | High | Strong | No |
| 4 | Overcast | Hot | High | Weak | Yes |
| 5 | Rain | Mild | High | Weak | Yes |
| 6 | Rain | Cool | Normal | Weak | Yes |
| 7 | Rain | Cool | Normal | Strong | No |
| 8 | Overcast | Cool | Normal | Strong | Yes |
| 9 | Sunny | Mild | High | Weak | No |
| 10 | Sunny | Cool | Normal | Weak | Yes |
| 11 | Rain | Mild | Normal | Weak | Yes |
| 12 | Sunny | Mild | Normal | Strong | Yes |
| 13 | Overcast | Mild | High | Strong | Yes |
| 14 | Overcast | Hot | Normal | Weak | Yes |
| 15 | Rain | Mild | High | Strong | No |

**Implementation Details**
The implementation followed these steps:

1. **Likelihood Calculation**:
    - For each feature, the conditional probabilities (likelihoods) were calculated based on the frequency of feature values, given the target classes ("Yes" and "No").
    - Results were saved to a JSON file for reusability.
2. **Prior and Marginal Probabilities**:
    - Priors for each class (P(Yes) and P(No)) were calculated as the relative frequencies of "Yes" and "No" in the target variable.
    - Marginal probabilities for feature values were computed to ensure correct denominator usage.
3. **Prediction**:
    - For a given instance, posterior probabilities were calculated using the likelihoods and priors, applying Laplace smoothing to handle zero probabilities.
    - The class with the highest posterior was assigned as the prediction.
4. **Performance Evaluation**:
    - Accuracy, precision, recall, and F1 score were calculated.
    - A confusion matrix was constructed to show correct and incorrect classifications.

**Challenges and Solutions**

- **Zero Probabilities**: Laplace smoothing was applied to avoid zero probabilities by adding 1 to each frequency count.
- **Class Imbalance**: Normalized probabilities were used to ensure unbiased predictions.

---

**Results**

```
Confusion Matrix:
          Predicted: Yes  Predicted: No
Actual: Yes    9               0
Actual: No     1               4

Accuracy: 0.93
Precision (Yes): 0.90
Recall (Yes): 1.00
F1 Score (Yes): 0.95
```

## Discussion

The classifier performed well, achieving an accuracy of 93%. Most misclassifications occurred due to overlapping probabilities between feature values, especially in small datasets with limited samples.

- **Reasons for Misclassifications**:
  - Some feature values were equally probable across classes, leading to incorrect predictions.
  - Noise or outliers in the dataset may have skewed probabilities.
- **Limitations**:
  - The implementation assumes independence between features, which might not hold true in real-world scenarios.
  - The dataset size was small, which can limit the generalizability of results.

---

## Conclusion

This project demonstrated the implementation of a Naive Bayes classifier from scratch. Key learnings included handling categorical data, implementing probabilistic computations, and evaluating classifier performance. Despite its simplicity, the Naive Bayes algorithm proved effective in predicting outcomes for the given dataset. Future improvements could involve extending the classifier to handle continuous features and incorporating additional techniques to address feature dependency.

---

## References


1. Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.
2. Dataset: *play_tennis.csv* (custom dataset for classification task).
3. Bayes' Theorem and Naive Bayes Classifier: Wikipedia.
4. https://medium.com/@rangavamsi5/naïve-bayes-algorithm-implementation-from-scratch-in-python-7b2cc39268b9