Decision Trees and Probabilistic Models

Mirolla Wael (8602)
John Wael (9077)
John Ayman (9078)

Part A: Gaussian Generative Classifier for Handwritten Digits

1. What We Are Doing

We want to make a computer recognize handwritten digits (0–9) from small 8×8 images.

We use a generative approach, which means we try to model how the data is generated for each digit. Then, when a new image comes, we can guess which digit it most likely is.

2. How the Model Works

1. **Class probabilities (priors)**
   We look at how many examples of each digit we have in the training set. If digit 3 appears 15% of the time, the model knows that digit 3 is a little less likely than others.

2. **Class means**
   For each digit, we calculate the "average image" by averaging all training examples of that digit. This average is called **μ_k**.

3. **Shared covariance**
   We also calculate how the pixels vary together across all digits. This is called **Σ**, the shared covariance matrix.

4. **Regularization**
   Sometimes the covariance matrix might not work well (it could be "singular" or unstable). To fix this, we add a small number **λ** to the diagonal, which stabilizes calculations.

5. **Prediction**
   For a new image, we calculate a score for each digit based on how likely the image is under that digit's Gaussian model. We also include the prior

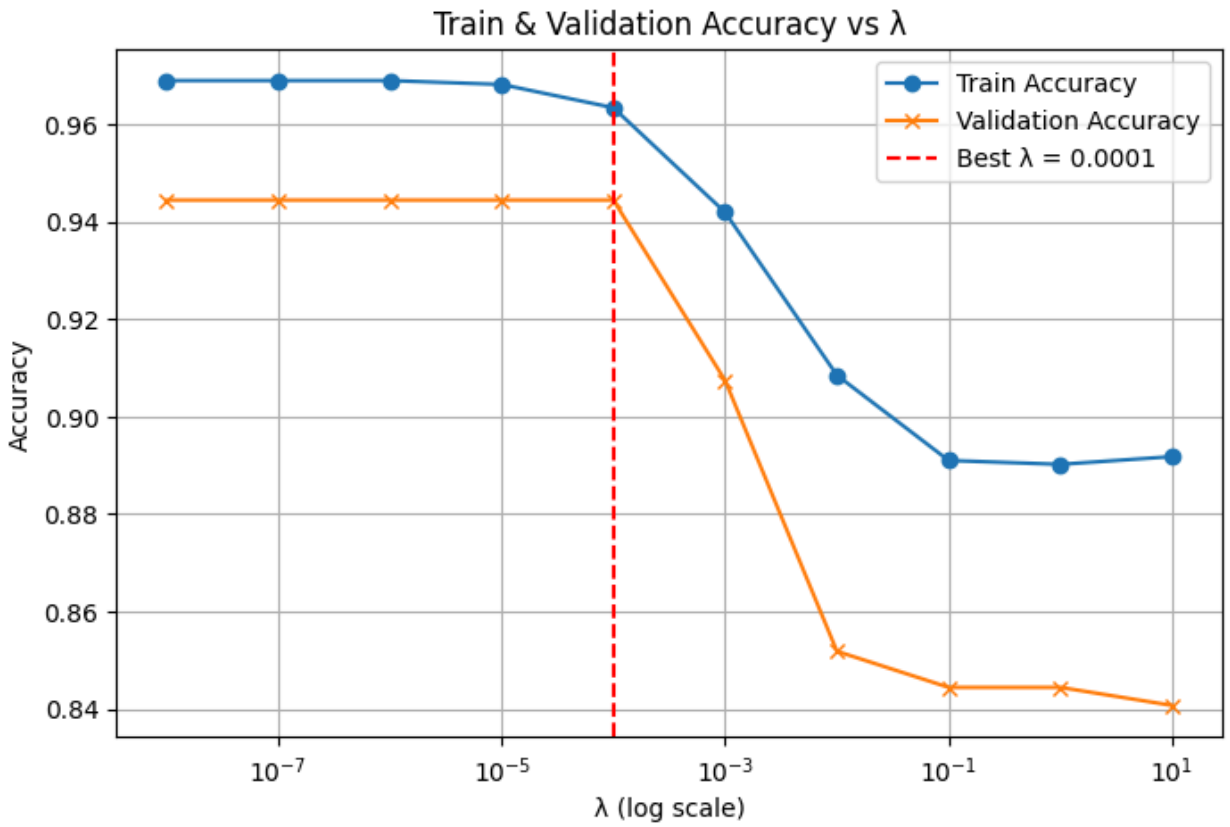
probability. The digit with the **highest score** is chosen as the prediction.

3. Choosing the Best λ

- We try different λ values and check which gives the best performance on a **validation set**.
- Small λ → almost exact covariance, but might be unstable.
- Large λ → very stable, but might underfit (not capture the real data).

We plot training and validation accuracy vs λ to pick the best one.

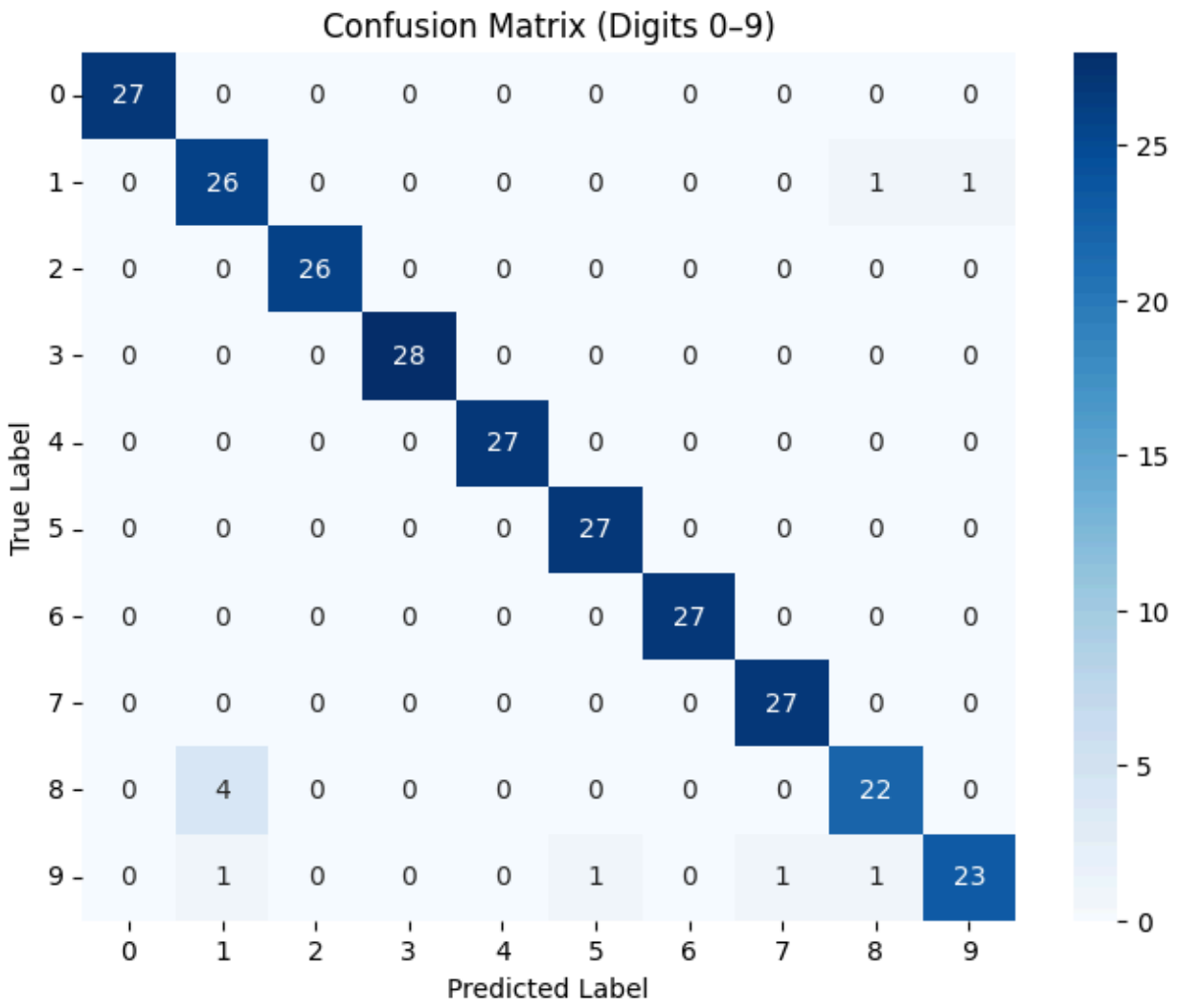
```
λ = 1e-08, Train Acc = 0.9690, Val Acc = 0.9444
λ = 1e-07, Train Acc = 0.9690, Val Acc = 0.9444
λ = 1e-06, Train Acc = 0.9690, Val Acc = 0.9444
λ = 1e-05, Train Acc = 0.9682, Val Acc = 0.9444
λ = 1e-04, Train Acc = 0.9634, Val Acc = 0.9444
λ = 1e-03, Train Acc = 0.9419, Val Acc = 0.9074
λ = 1e-02, Train Acc = 0.9085, Val Acc = 0.8519
λ = 1e-01, Train Acc = 0.8910, Val Acc = 0.8444
λ = 1e+00, Train Acc = 0.8902, Val Acc = 0.8444
λ = 1e+01, Train Acc = 0.8918, Val Acc = 0.8407
Best Lamda = 0.0001
```



4. Final Results

After picking the best λ, we train the model on **train + validation** and test it on unseen data.

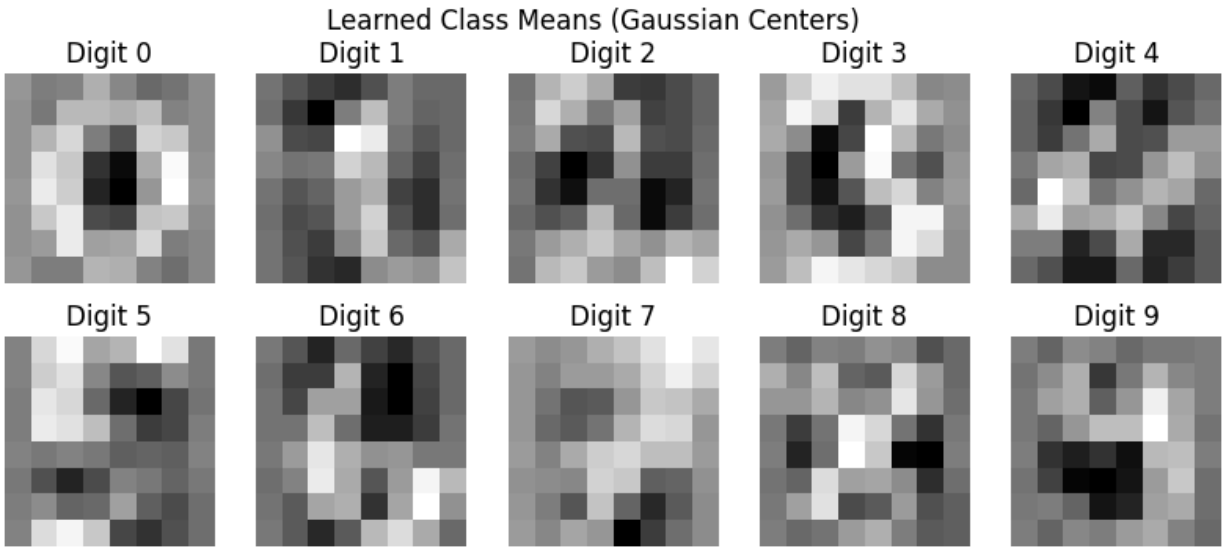
```
=== FINAL MODEL EVALUATION ===
Test Accuracy: 0.963
Macro Precision: 0.9642
Macro Recall: 0.9627
Macro F1-Score: 0.9627
```



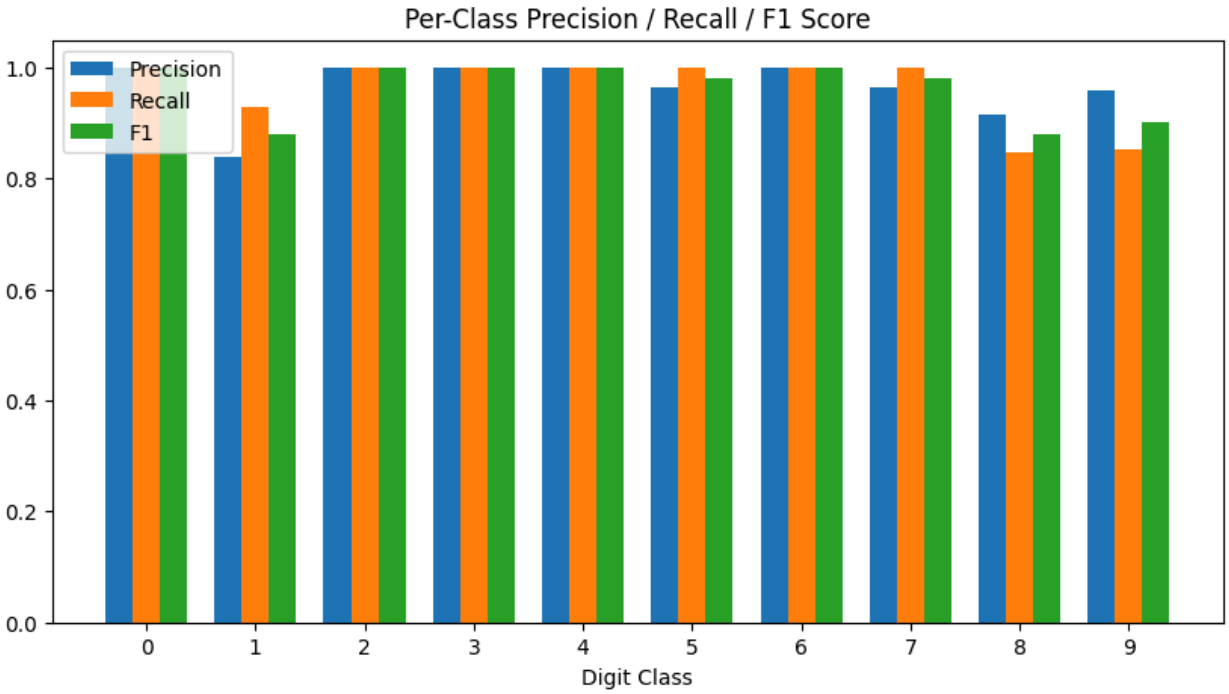
We also look at confusion between digits. 8 is confused with 1

5. Visualizing What the Model Learned

- We can **draw the average image for each digit**. These are the “centers” of the Gaussians and show what the model thinks each digit looks like.



- We also plot **precision, recall, F1-score for each digit** to see which digits are recognized better than others.



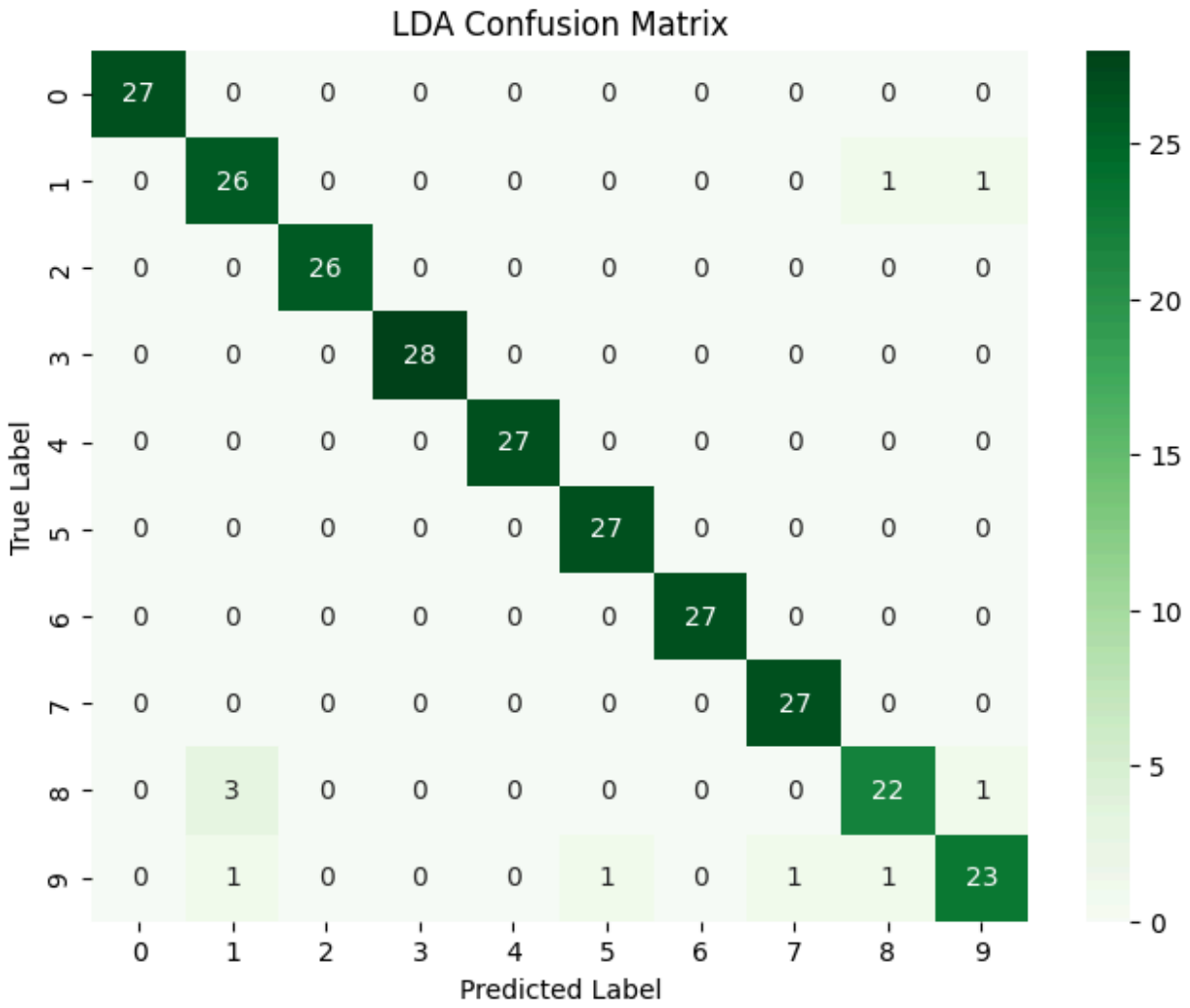
6.Sklearn Implementation

We also implemented the Gaussian Generative Classifier (GGC) using scikit-learn's LinearDiscriminantAnalysis (LDA) for comparison. Conceptually, this is equivalent to our GGC: both assume that the features of each class follow a Gaussian distribution with a shared covariance matrix, compute class means and priors, and predict by selecting the class with the highest discriminant score. Using LDA provides a verified and optimized implementation of the same generative approach.

```
=== LDA (Shared Covariance Gaussian) ===
Test Accuracy: 0.963
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        27
           1       0.87      0.93      0.90        28
           2       1.00      1.00      1.00        26
           3       1.00      1.00      1.00        28
           4       1.00      1.00      1.00        27
           5       0.96      1.00      0.98        27
           6       1.00      1.00      1.00        27
           7       0.96      1.00      0.98        27
           8       0.92      0.85      0.88        26
           9       0.92      0.85      0.88        27

    accuracy                           0.96       270
   macro avg       0.96      0.96      0.96       270
weighted avg       0.96      0.96      0.96       270
```



Both models produce very similar predictions, which verifies the correctness of our implementation.

6. Summary

- **Strengths:** The model is simple, interpretable, and works well on digits.
- **Weaknesses:** Assumes all digits share the same covariance and Gaussian shape; may struggle if digits overlap a lot.
- **Effect of λ:** Small changes in λ don't affect accuracy much, but very large λ reduces performance.

Part B: Categorical Naive Bayes Classifier for Adult Income

1. What We Are Doing

We want to predict whether a person earns more than 50K per year based on categorical information like workclass, education, marital status, occupation, relationship, race, sex, and native country.

We use a Categorical Naive Bayes (CNB) model, which assumes all features are independent given the class. This means each feature contributes separately to the probability of a person earning >50K or ≤50K.

2. How the Model Works

Class probabilities (priors)

We look at how many people in the training set earn >50K or ≤50K. This gives the **prior probability** for each class.

Feature likelihoods

For each feature (like education or occupation), we calculate the probability of each value given the class. For example, what fraction of people with Bachelors earn >50K? This gives **P(X=value | class)**.

Laplace Smoothing (α)

Sometimes a feature value doesn't appear in a class (e.g., very rare occupations). To avoid zero probabilities, we add a small number **α**. This is called **Laplace smoothing**.

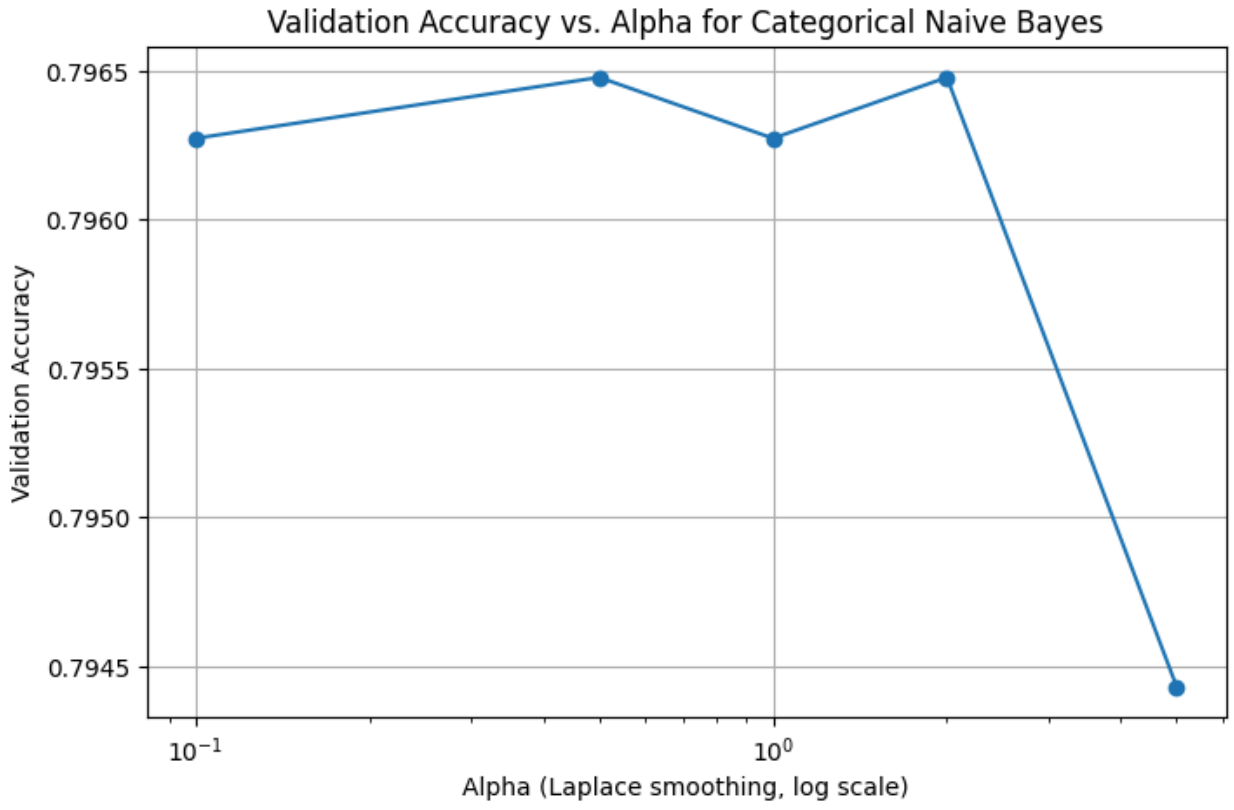
Prediction

For a new person, we multiply the probabilities of all feature values for each class and multiply by the prior. The class with the highest probability is predicted.

3. Choosing the Best α (Smoothing Parameter)

- We try different α values: 0.1, 0.5, 1.0, 2.0, 5.0.
- **Small α** → more sensitive to rare values, may overfit.
- **Large α** → smoother probabilities, may underfit.
- We pick the α that gives the **best validation accuracy**.

```
Alpha = 0.1: Validation Accuracy = 79.63%
Alpha = 0.5: Validation Accuracy = 79.65%
Alpha = 1.0: Validation Accuracy = 79.63%
Alpha = 2.0: Validation Accuracy = 79.65%
Alpha = 5.0: Validation Accuracy = 79.44%
```



4. Feature Selection

- We tried using subsets of 3 features at a time to see which features are most predictive.

- Some features, like education, occupation, and marital-status, tend to give higher accuracy than others.

- This confirms that **not all features are equally informative**.

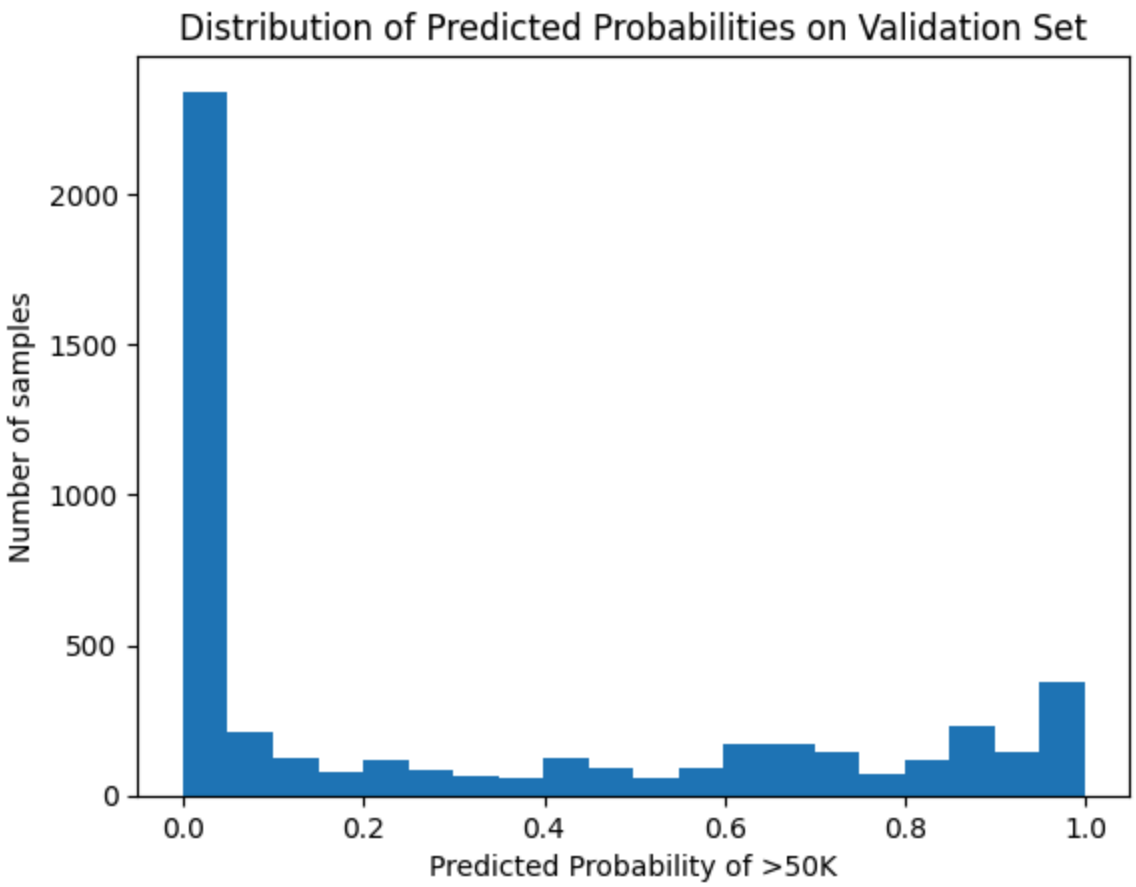
```
Features ('workclass', 'education', 'marital-status'): Validation Accuracy = 82.08%
Features ('workclass', 'education', 'occupation'): Validation Accuracy = 77.29%
Features ('workclass', 'education', 'relationship'): Validation Accuracy = 82.08%
Features ('workclass', 'education', 'race'): Validation Accuracy = 77.46%
Features ('workclass', 'education', 'sex'): Validation Accuracy = 78.40%
```

5. Probability Analysis

- We can compute the predicted probability that a person earns >50K.

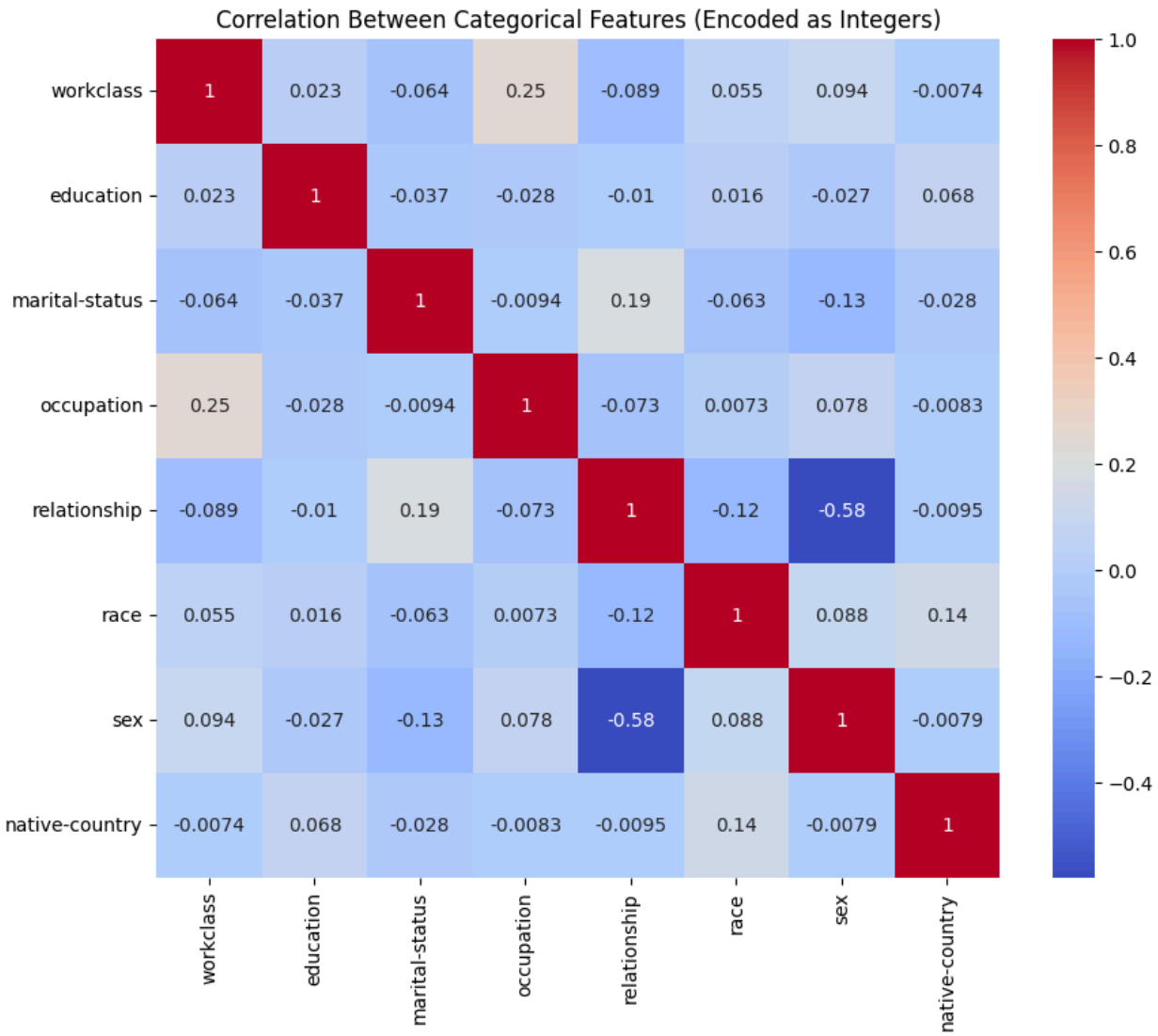
- Plotting these probabilities shows how confident the model is for different people.

- Most predictions are close to 0 or 1, but some are in between, showing **uncertainty**.



6. Assumption Check: Feature Independence

- Categorical Naive Bayes assumes features are independent given the class.
- We computed correlation between features (after encoding them as integers).
- Some correlations exist (like marital-status and relationship), but the model still works reasonably well.


Correlation Between Categorical Features (Encoded as Integers)
workclass
education
marital-status
occupation
relationship
race
sex
native-country
1 0.023 -0.064 0.25 -0.089 0.055 0.094 -0.0074
0.023 1 -0.037 -0.028 -0.01 0.016 -0.027 0.068
-0.064 -0.037 1 -0.0094 0.19 -0.063 -0.13 -0.028
0.25 -0.028 -0.0094 1 -0.073 0.0073 0.078 -0.0083
-0.089 -0.01 0.19 -0.073 1 -0.12 -0.58 -0.0095
0.055 0.016 -0.063 0.0073 -0.12 1 0.088 0.14
0.094 -0.027 -0.13 0.078 -0.58 0.088 1 -0.0079
-0.0074 0.068 -0.028 -0.0083 -0.0095 0.14 -0.0079 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

7. Sklearn Implementation

- We compared our implementation with **sklearn's MultinomialNB** using α = 1.0.
- Validation Accuracy:
    - Our CNB: 79.4%
    - Sklearn MultinomialNB: 79.4%

Both models produce very similar predictions, which verifies the correctness of our implementation.

8. Final Observations

Strengths:

- Simple and interpretable.
- Works well on categorical data.
- Easy to implement from scratch.

Weaknesses:

- Assumes feature independence, which is not always true.
- Sensitive to rare feature values if α is too small.

Effect of α:

- Small α → more extreme probabilities, higher risk of overfitting.
- Large α → probabilities are smoother, may underfit.

Part C: Decision Tree Classifier for Breast Cancer Dataset

1. What We Are Doing

We want to predict whether a tumor is **malignant (1)** or **benign (0)** using the **Breast Cancer dataset** from sklearn.datasets.

We implement a **Decision Tree classifier from scratch**, using **information gain (entropy)** as the splitting criterion. This allows us to see which features are most informative and understand the decision process.

2. How the Model Works

Tree Structure:

- Each node splits the dataset based on a **feature and threshold**.
- Leaf nodes contain the **majority class** of the samples that reach them.

Entropy and Information Gain:

- Entropy measures how “mixed” the labels are.
- Information gain measures how much a split reduces uncertainty.
- The algorithm picks the split with the **highest information gain** at each node.

Stopping Criteria:

- Maximum depth (max_depth)
- Minimum number of samples to split (min_samples_split)
- If all samples belong to the same class, the node becomes a leaf.

Prediction:

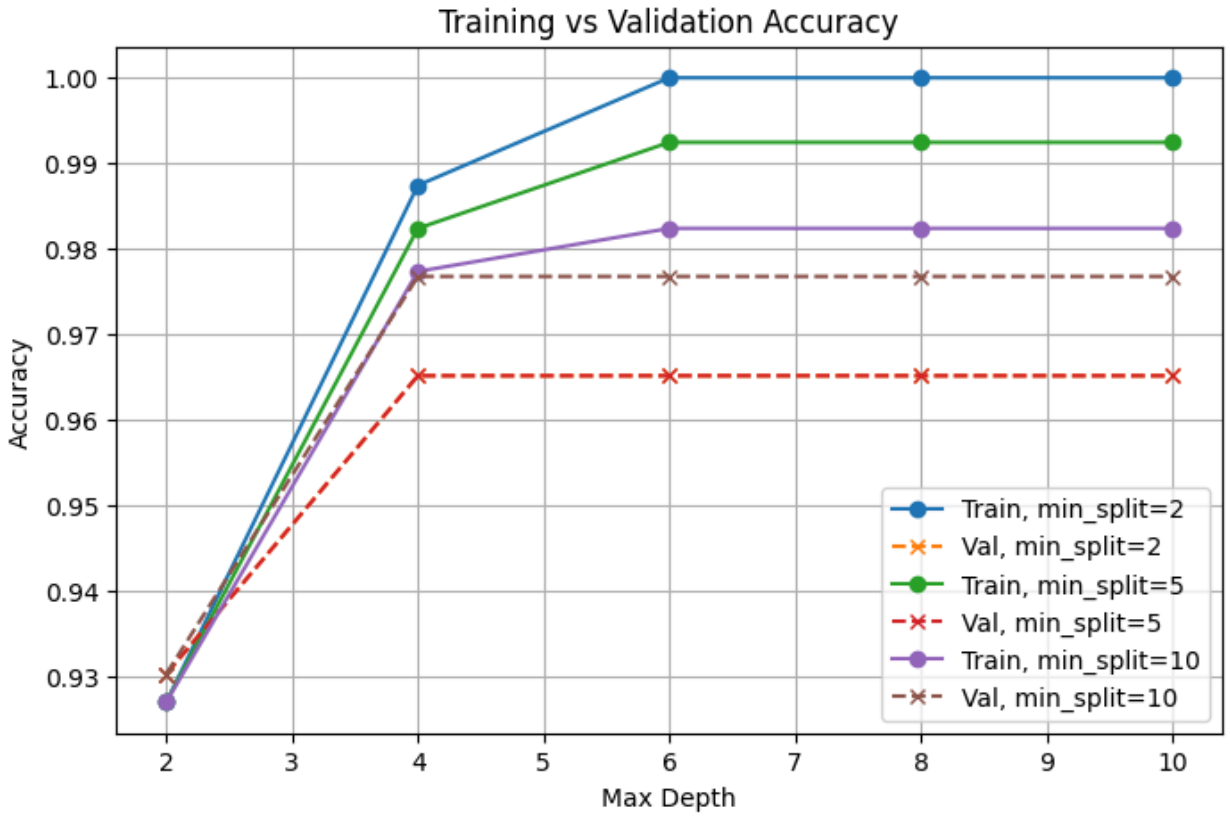
- For a new sample, follow the tree from root to leaf.
- The leaf's class is the predicted label.

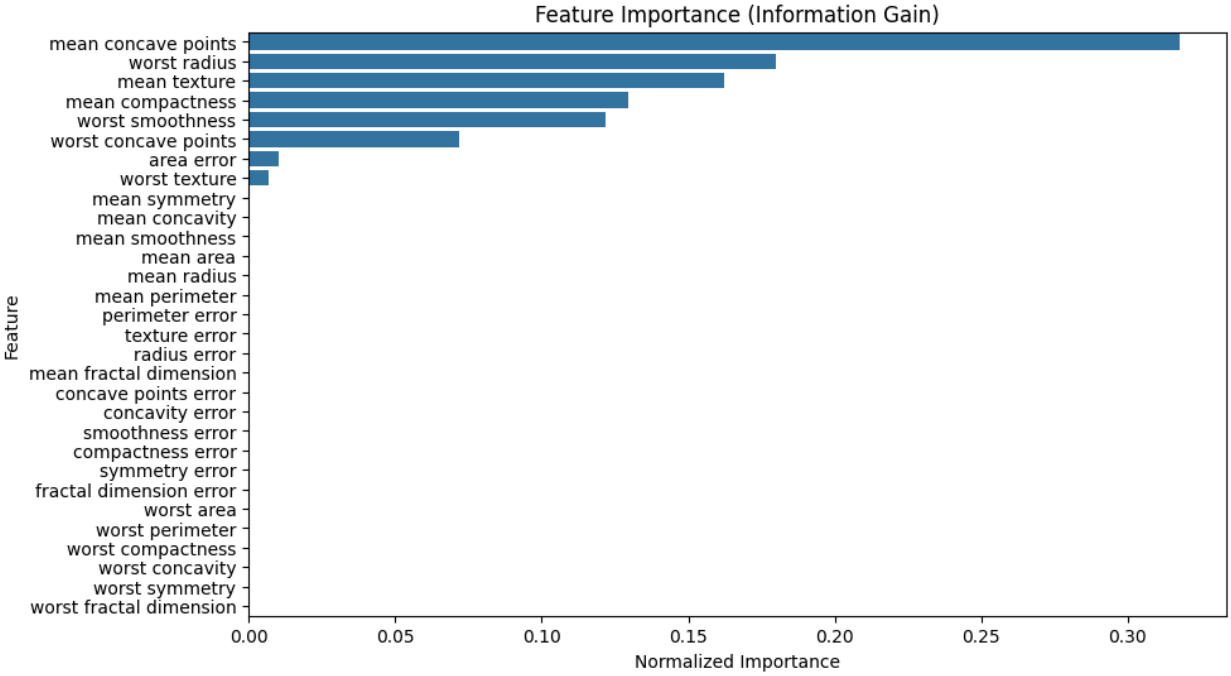
3. Hyperparameter Tuning

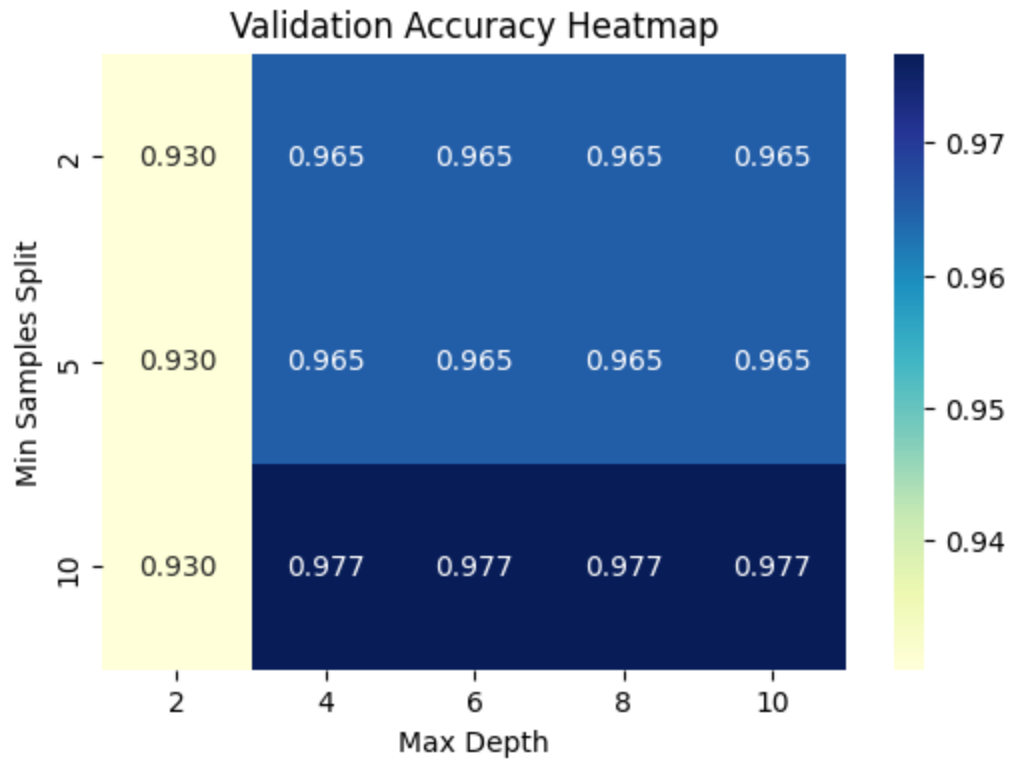
We try different values of:

- max_depth = [2, 4, 6, 8, 10]
- min_samples_split = [2, 5, 10]

```
Hyperparameter Tuning Results:
    max_depth  min_samples_split  train_acc   val_acc
0           2                  2   0.926952  0.930233
1           2                  5   0.926952  0.930233
2           2                 10   0.926952  0.930233
3           4                  2   0.987406  0.965116
4           4                  5   0.982368  0.965116
5           4                 10   0.977330  0.976744
6           6                  2   1.000000  0.965116
7           6                  5   0.992443  0.965116
8           6                 10   0.982368  0.976744
9           8                  2   1.000000  0.965116
10          8                  5   0.992443  0.965116
11          8                 10   0.982368  0.976744
12         10                  2   1.000000  0.965116
13         10                  5   0.992443  0.965116
14         10                 10   0.982368  0.976744
Best Hyperparameters: max_depth=4, min_samples_split=10
```


Training vs Validation Accuracy
Accuracy
Max Depth
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
2
3
4
5
6
7
8
9
10
Train, min_split=2
Val, min_split=2
Train, min_split=5
Val, min_split=5
Train, min_split=10
Val, min_split=10


Feature Importance (Information Gain)
Feature
Normalized Importance
mean concave points
worst radius
mean texture
mean compactness
worst smoothness
worst concave points
area error
worst texture
mean symmetry
mean concavity
mean smoothness
mean area
mean radius
mean perimeter
perimeter error
texture error
radius error
mean fractal dimension
concave points error
concavity error
smoothness error
compactness error
symmetry error
fractal dimension error
worst area
worst perimeter
worst compactness
worst concavity
worst symmetry
worst fractal dimension
0.00
0.05
0.10
0.15
0.20
0.25
0.30



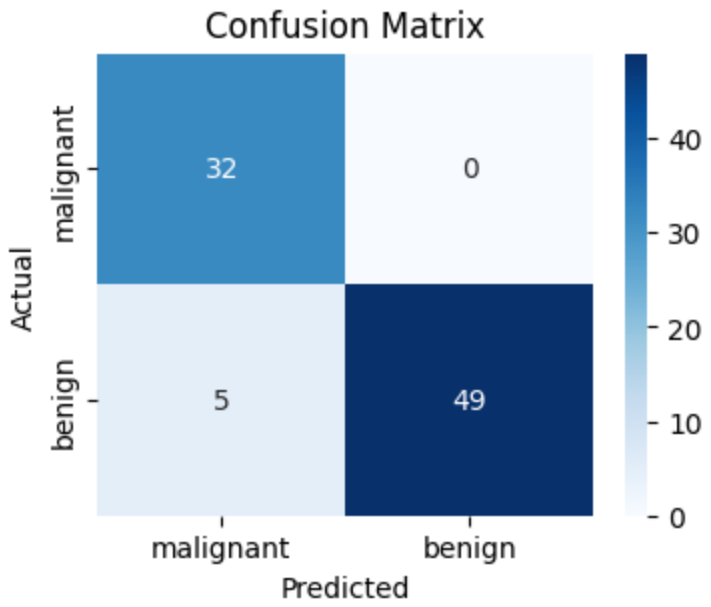
→ **Best hyperparameters:** max_depth=4 , min_samples_split=10

4. Evaluation on Test Set

Using the final tree trained on **training + validation sets**:

```
Test Set Performance:
Accuracy: 0.9419
Precision: 1.0000
Recall: 0.9074
F1-score: 0.9515
Confusion Matrix:
 [[32  0]
 [ 5 49]]
```

```
Metrics per class:
Class 0: Precision=0.8649, Recall=1.0000, F1=0.9275
Class 1: Precision=1.0000, Recall=0.9074, F1=0.9515
```

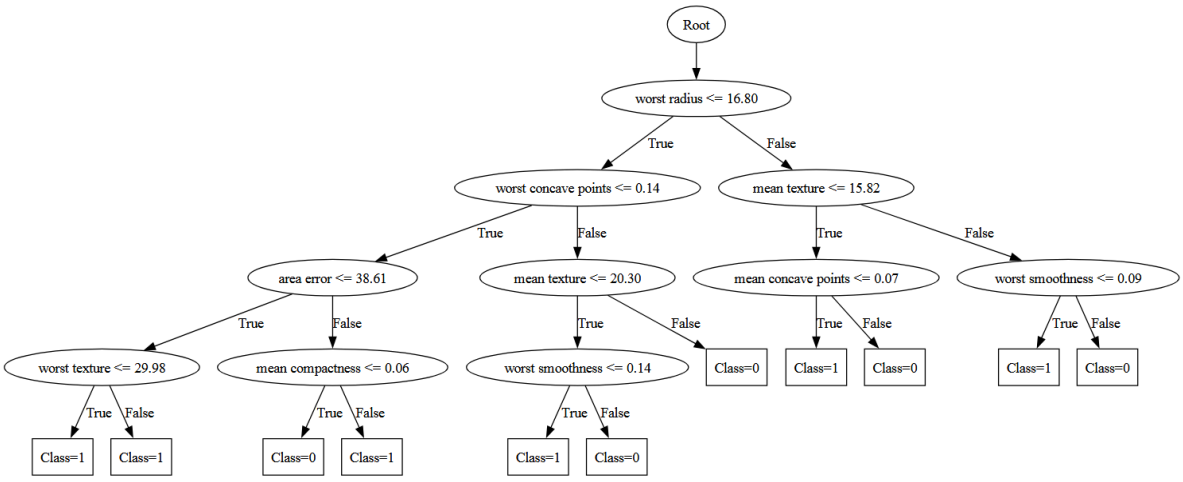


Observations:

- The tree performs well with high accuracy and balanced precision/recall.
- Feature importance shows that worst radius, worst concave points, mean texture are the most informative features.

5. Visualizing the Tree

- Printed tree shows splits with features and thresholds.
- Graphviz visualization gives a clear **hierarchical view** of decisions.



Sklearn Implementation

```
Sklearn Decision Tree Classifier Performance:
Accuracy score: 0.92
              precision    recall  f1-score   support

   malignant       0.86      0.94      0.90        32
      benign       0.96      0.91      0.93        54

    accuracy                           0.92        86
   macro avg       0.91      0.92      0.91        86
weighted avg       0.92      0.92      0.92        86
```

Both our model and sklearn have the same accuracy whicd verifies our work

Strengths:

- Interpretable and simple to understand.
- Good performance on structured datasets.

Weaknesses:

- Can overfit if tree is too deep.
- Sensitive to small changes in data.

Part D: Random Forest Classifier for Breast Cancer Dataset

1. What We Are Doing

We want to **improve the performance and stability** of a single decision tree using **Random Forests**:

- Build multiple decision trees on **bootstrap samples** of the data.
- Each tree sees a **random subset of features** at each split.
- Predictions are combined using **majority voting**.

2. How the Model Works

Bootstrap Sampling:

- Each tree is trained on a random sample of the dataset with replacement.

Random Feature Subsets:

- At each split, only a subset of features is considered.
- This decorrelates trees and improves generalization.

Prediction:

- Each tree predicts the class for a sample.
- The **most common prediction** among all trees is the final prediction.

3. Hyperparameter Tuning

We tune:

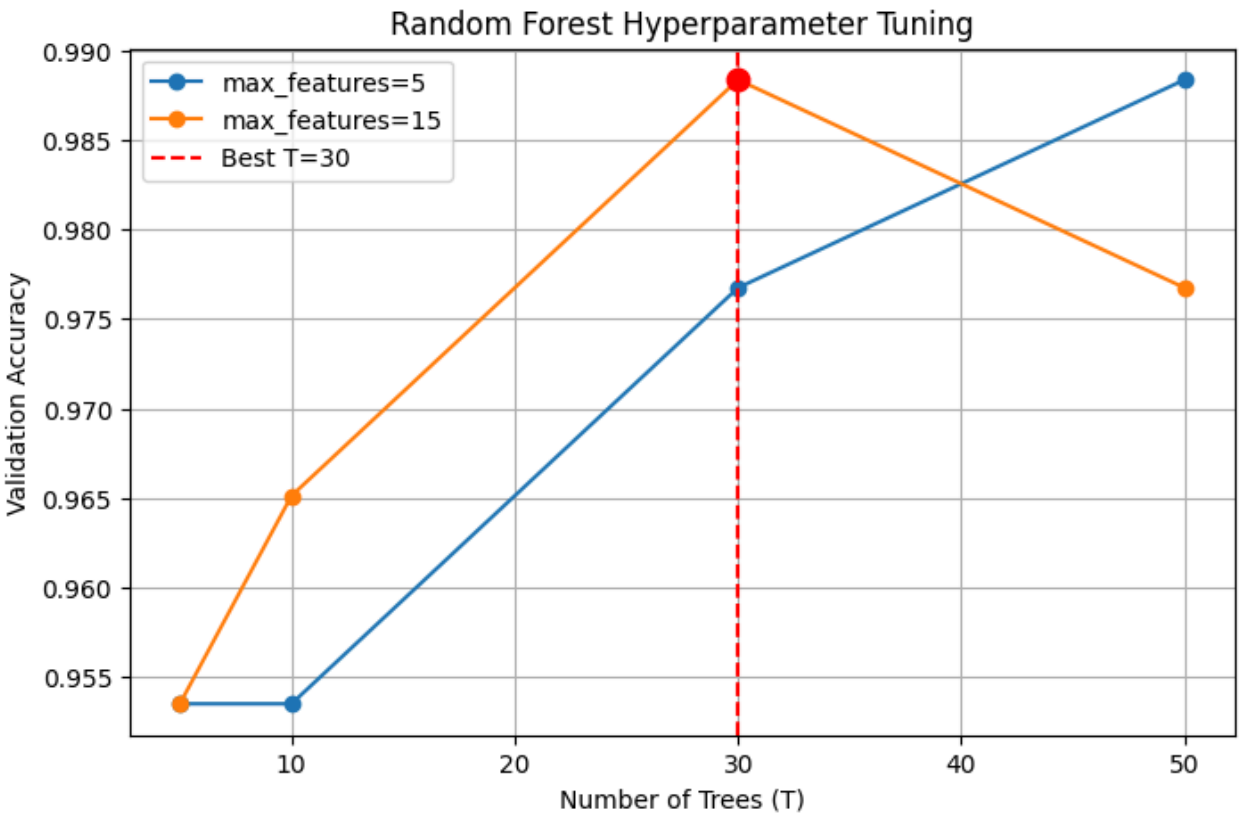
- Number of trees n_trees = [5, 10, 30, 50]
- Max features per tree max_features = [sqrt(d), d/2]
- Max depth = best depth from single decision tree (max_depth=6)
- Min samples split = best from single tree (min_samples_split=2)

```
T=5, max_features=5, val_acc=0.9418604651162791
T=5, max_features=15, val_acc=0.9767441860465116
T=10, max_features=5, val_acc=0.9651162790697675
T=10, max_features=15, val_acc=0.9767441860465116
T=30, max_features=5, val_acc=0.9767441860465116
T=30, max_features=15, val_acc=0.9767441860465116
T=50, max_features=5, val_acc=0.9767441860465116
T=50, max_features=15, val_acc=0.9767441860465116

Best hyperparameters: (5, 15) Val Accuracy: 0.9767441860465116
```

```
T=5, max_features=5, val_acc=0.9534883720930233
T=5, max_features=15, val_acc=0.9534883720930233
T=10, max_features=5, val_acc=0.9534883720930233
T=10, max_features=15, val_acc=0.9651162790697675
T=30, max_features=5, val_acc=0.9767441860465116
T=30, max_features=15, val_acc=0.9883720930232558
T=50, max_features=5, val_acc=0.9883720930232558
T=50, max_features=15, val_acc=0.9767441860465116

Best hyperparameters: (30, 15) Val Accuracy: 0.9883720930232558
```



Validation Results:

→ **Best Random Forest:** n_trees=30, max_features=15

4. Evaluation on Test Set

```
Single Decision Tree Test Accuracy: 0.9418604651162791
```

```
(5, 15)
Final Random Forest Test Accuracy: 0.9418604651162791
```

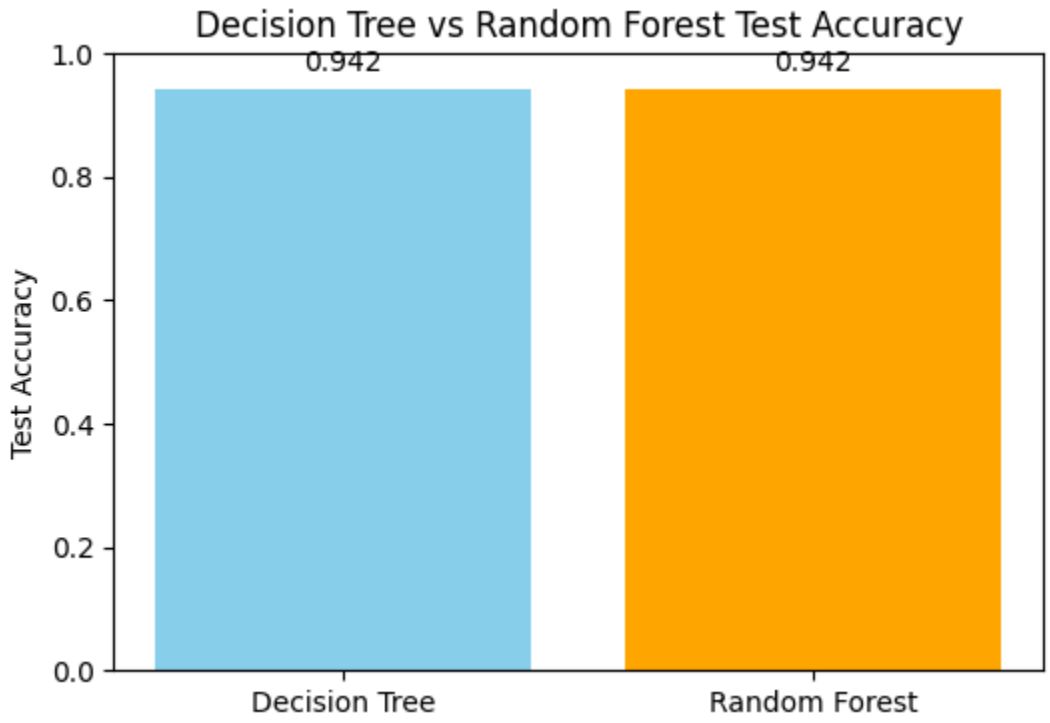
```
(30, 15)
Final Random Forest Test Accuracy: 0.9418604651162791
```

Observations:

- Both models reach same accuracy
- Reduces overfitting and improves stability.
- Feature importance is aggregated over all trees, confirming key features like mean radius and mean texture.

Visual Comparison:

- Bar plot comparing test accuracy of **Decision Tree vs Random Forest**



5. Summary

Decision Tree:

- Simple, interpretable.
- Can overfit and sensitive to data changes.

Random Forest:

- More robust and accurate.
- Handles variance better by averaging multiple trees.
- Slightly less interpretable, but feature importance is still visible.