A Comparative Analysis and Implementation on Gaussian Naive Bayes (GNB) and PCA-GNB Classifiers

Enno Moench, Janet Joseph, Xueru Ma, Yishu Li

Contents

Naive Bayes: Foundation of Gaussian Naive

Bayes Theorem : $P(y|x_1, ..., x_n) = \frac{P(y)P(x_1, ..., x_n|y)}{P(x_1, ..., x_n)}$

Assumption: **Features are conditionally independent**:

$$P(x_1, ..., x_n|y) = \prod_{i=1}^{n} P(x_i|y)$$

Predict the class label $\hat{y}$ that maximizes the posterior probability.

Simplified Classification Rule: $\hat{y} = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$

Simplifies probabilistic classification through independence assumptions.

Gaussian Naive Bayes(GNB)

Likelihood Assumption:Likelihood Assumption: Each feature follows a Gaussian (normal) distribution. $$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_{y,i}^2}} \exp\left(-\frac{(x_i-\mu_{y,i})^2}{2\sigma_{y,i}^2}\right)$$

Parameters to Estimate:

- Mean(μ): $\mu_{y,i} = \frac{1}{N_y}\sum_{j=1}^{N_y} x_{j,i}$
- Variance(σ^2) $\sigma_{y,i}^2 = \frac{1}{N_y-1}\sum_{j=1}^{N_y}(x_{j,i}-\mu_{y,i})^2$

Class Posterior in Log Form:

$$\log\left(P(y|x_1,\ldots,x_n)\right) = \log(P(y)) + \sum_{i=1}^{n}\log(P(x_i|y))$$

Gaussian Naive Bayes extends Naive Bayes with Gaussian-distributed likelihoods.

What is Principal Component Analysis (PCA)?

PCA **reduced dimensionality** by **identifying directions (principal components) of maximum variance** in the data.

Steps for PCA calculation:

a. Compute the covariance matrix.
b. Perform eigenvalue decomposition.
c. Sort eigenvalues in descending order.
d. Select top components to form a reduced feature space.

In our work, PCA helps reduce the noise and dimensionality of the data, improving classification accuracy and addressing the independence assumption inherent in Naive Bayes.

Gaussian Naive Bayes

The Code

Code Overview

- Train the model
- Predict outcomes
  - Gaussian Probability Function
- Evaluate accuracy
- Save and load models
- Model test results

Train the model

- Calculate the mean and variance for each feature per class
- Compute the prior probabilities based on class frequencies

```
function train_model(X_train, y_train):
    # Get the number of features (columns in X_train)
    n_features = number of features in X_train

    # Get the number off unique classes in y_train
    n_classes = number of unique values in y_train

    # Initialize arrays to store means, variances, and priors for each class
    means = 2D array of zeros with shape (n_classes, n_features)
    variances = 2D array of zeros with shape (n_classes, n_features)
    priors = 1D array of zeros with length n_classes

    # Loop through each class
    for each class c in n_classes:
        # Extract rows in X_train corresponding to class c
        X_c = subset of X_train where y_train equals c

        # Calculate the mean of each feature for class c
        means[c] = mean of X_c along the rows

        # Calculate the variance of each feature for class c
        variances[c] = variance of X_c along the rows

        # Calculate the prior probability of class c
        priors[c] = number of rows in X_c divided by total rows in X_train

    # Return the means, variances, and priors
    return means, variances, priors
```

Predict outcomes

- predict_log_prob: Compute log of joint probabilities using:

$$\log \mathrm{P}(x|c) = \sum \log \mathrm{P}(x_i|c)$$

```
function predict_log_prob(inputs):
    # Initialize an empty list to store log probabilities for each class
    log_probs = []

    # Loop through each class
    for each class c in number of classes:
        # Compute the log of the prior probability for class c
        log_prior = log(prior probability of class c)

        # Compute the log-likelihood by summing the log of the Gaussian probabilities
        log_likelihood = sum of log(Gaussian probability of each feature for class c)

        # Append the joint log probability (log_prior + log_likelihood) to the list
        log_probs.append(log_prior + log_likelihood)

    # Convert the list of log probabilities to a 2D array and transpose it
    log_probs = convert log_probs to a 2D array and transpose

    # Return the array of log probabilities
    return log_probs
```

Gaussian Probability Function

- Calculates the likelihood of a feature given a Gaussian distribution

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

```
function calculate_gaussian_prob(x, mu, sig, epsilon=1e-8):
    # Adjust the variance to avoid division by zero (add a small epsilon)
    adjusted_sig = sig + epsilon

    # Calculate the exponent of the Gaussian probability formula
    exponent = exp(-(x - mu)^2 / (2 * adjusted_sig))

    # Compute the Gaussian probability using the formula
    probability = (1 / sqrt(2 * pi * adjusted_sig)) * exponent

    return probability
```

Predict outcomes

- predict_prob: Exponentiate the joint probabilities

```
function predict_prob(inputs):
    # Compute the log probabilities using the predict_log_prob function
    log_probs = calculate log probabilities for each input

    # Convert log probabilities to normal probabilities using the exponential function
    probabilities = exp(log_probs)

    return probabilities
```

- predict: Determine the class with the highest probability

```
function predict(inputs):
    # Compute the probabilities for each class using the predict_prob function
    probabilities = calculate probabilities for each input

    # Find the class index with the highest probability for each input
    predictions = index of maximum probability for each input (along rows)

    # Return the predicted class labels
    return predictions
```

Evaluate accuracy

Process:

- Predict labels for test data
- Compare with true labels

Metric: Proportion of correct predictions

```
function calculate_accuracy(X_test, y_test):
    # Use the model's predict function to get predictions for the test data
    y_pred = predict class labels for X_test

    # Compare predictions with actual labels (y_test) and calculate the mean
    accuracy = mean value of (y_pred == y_test)

    # Return the accuracy as a float between 0 and 1
    return accuracy
```

Save and load models

Save Functionality:

- `save(self, filename)`: Serialize the model using `pickle`

Load Functionality:

- `load(filename)`: Deserialize the model from a file

Use Case: Enables persistence and reuse of trained models

```
function save_model(filename):
    # Open the specified file in binary write mode
    with file opened in write binary mode as f:
        # Serialize the current object (self) and write it to the file
        save object to file using pickle
    # File is automatically closed after exiting the 'with' block
```

```
function load_model(filename):
    # Open the specified file in binary read mode
    with file opened in read binary mode as f:
        # Deserialize the object from the file using pickle
        loaded_object = load object from file using pickle

    return loaded_object
```

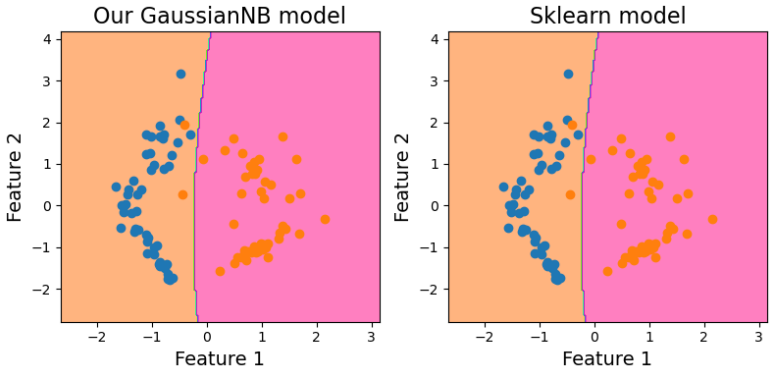
GNB Test Results

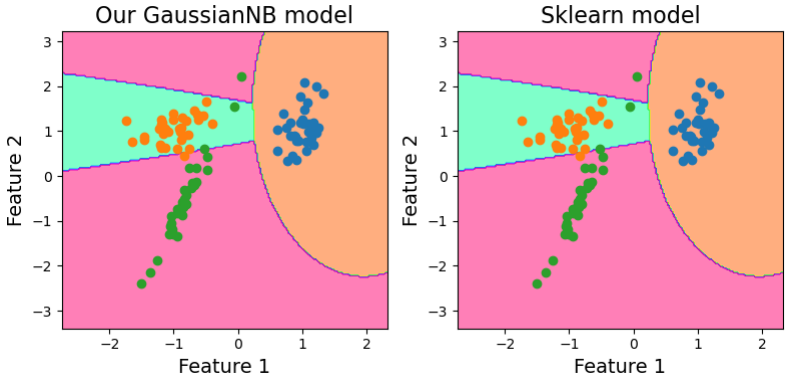
- We compared our model with the Sci-kit learn model using three sklearn-provided datasets

```
Test model accuracies...
Using n_components=4 for PCA-GNB model here.
---------------- Iris ------------------
Our model's accuray:  1.0
Sklearn model's accuray:  1.0
PCA-GNB model's accuracy:  1.0

---------------- Wine ------------------
Our model's accuray:  0.9444444444444444
Sklearn model's accuray:  0.9444444444444444
PCA-GNB model's accuracy:  0.9629629629629629

---------------- Breast Cancer ------------------
Our model's accuray:  0.9122807017543859
Sklearn model's accuray:  0.9239766081871345
PCA-GNB model's accuracy:  0.935672514619883
```

GNB Visualization

- Using scikit-learn package ‘make_classification’ method to create 2D fake dataset for visualization

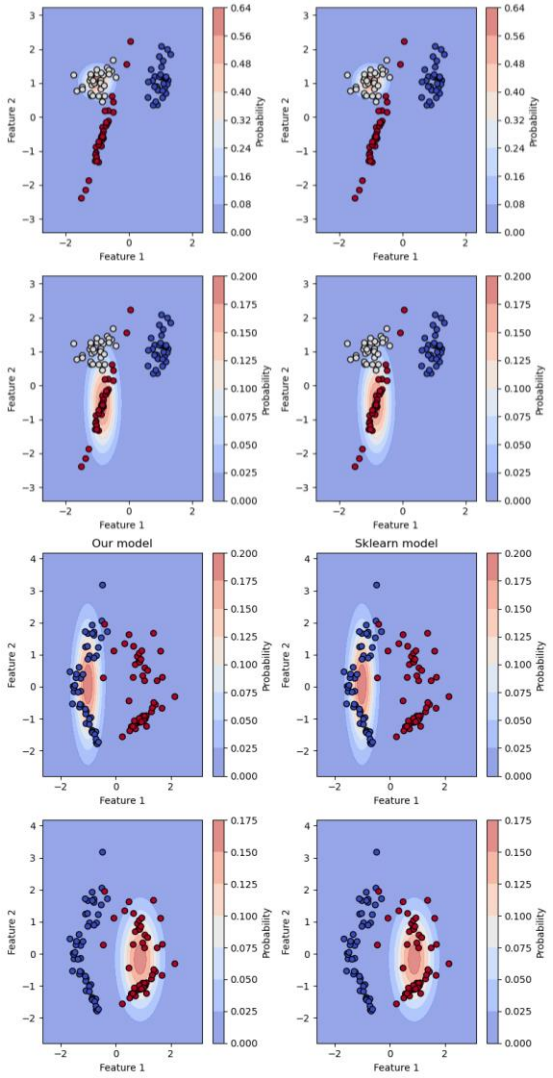




GNB Visualization

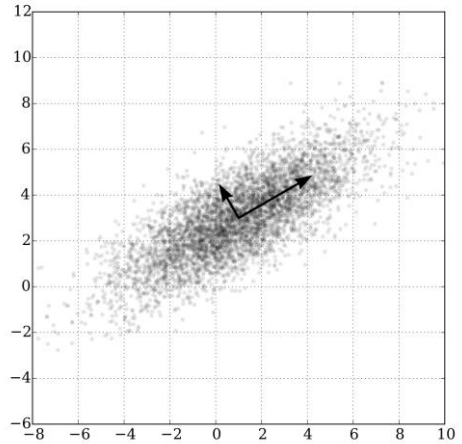
- The Gaussian ellipses' axes are all perpendicular to the feature axis
- This is caused by the assumption of GNB that all the features are independent

- Use PCA to eliminate the dependency

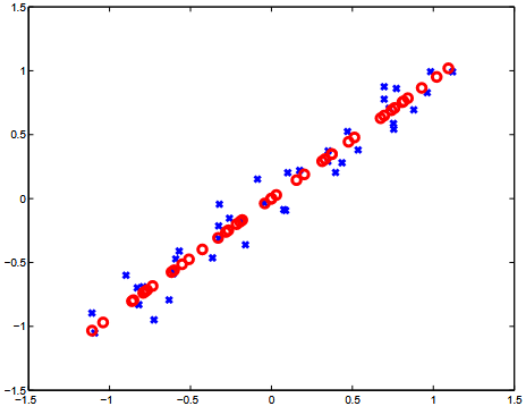


PCA-GNB

- PCA
  - Principal component analysis (PCA) is a linear dimensionality reduction technique that can transform the data linearly onto a new coordinate system.
  - The principal components are calculated using eigenvalue decomposition, which makes them orthogonal to each other
  - $A = U^T DU$, where *A* is the covariance matrix of the data, *U* is the principle components, and *D* is a diagonal matrix of eigenvalues.



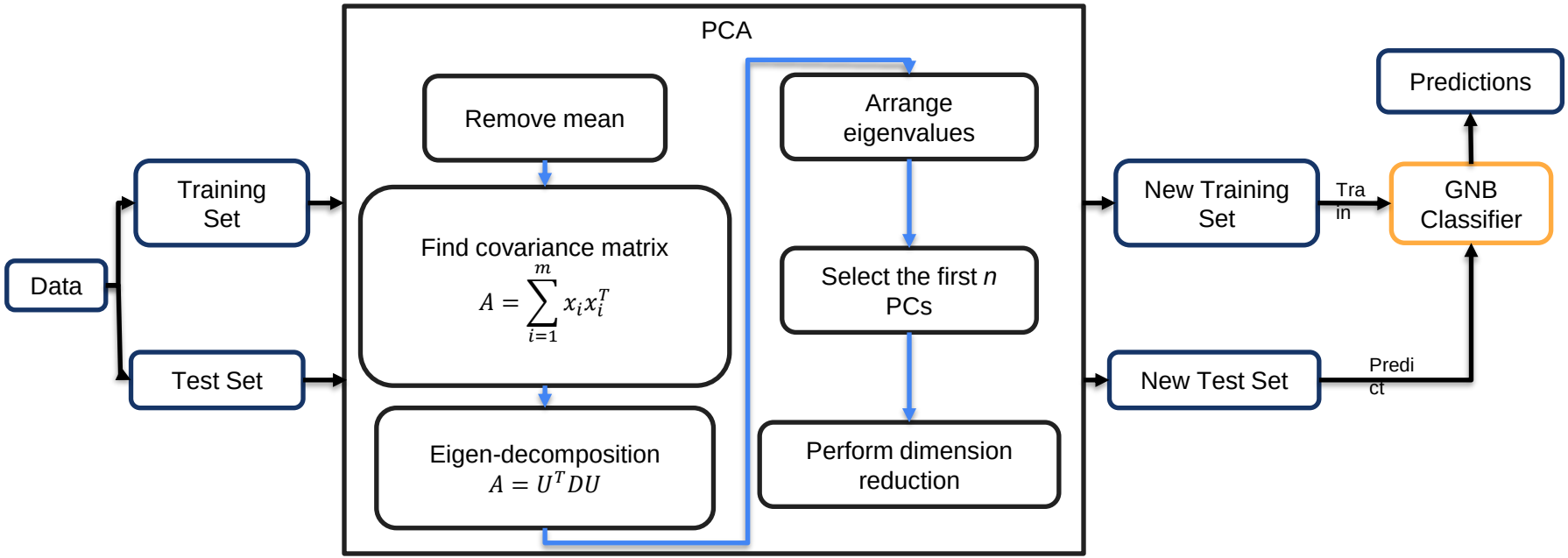
PCA rotates the feature axis



PCA dimension reduction

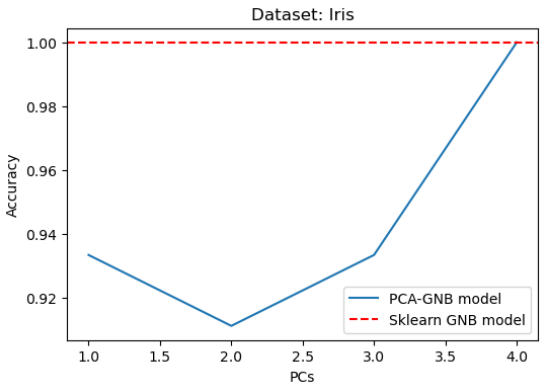
PCA-GNB

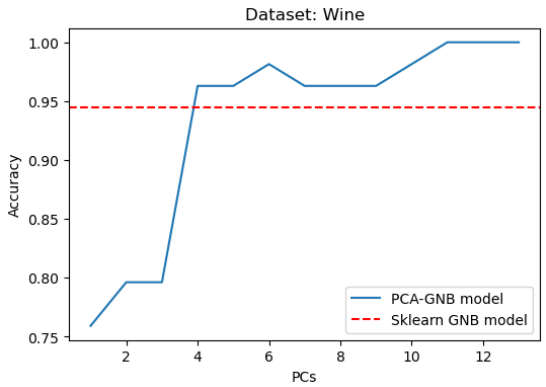
- Structure PCA-GNB decoder

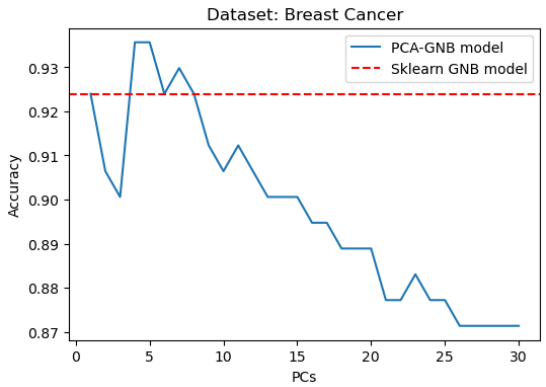


PCA-GNB

- Classifier performance
  - We used scikit-learn package built-in dataset to test our model performance
  - The PCA-GNB model outperforms GNB model in some datasets

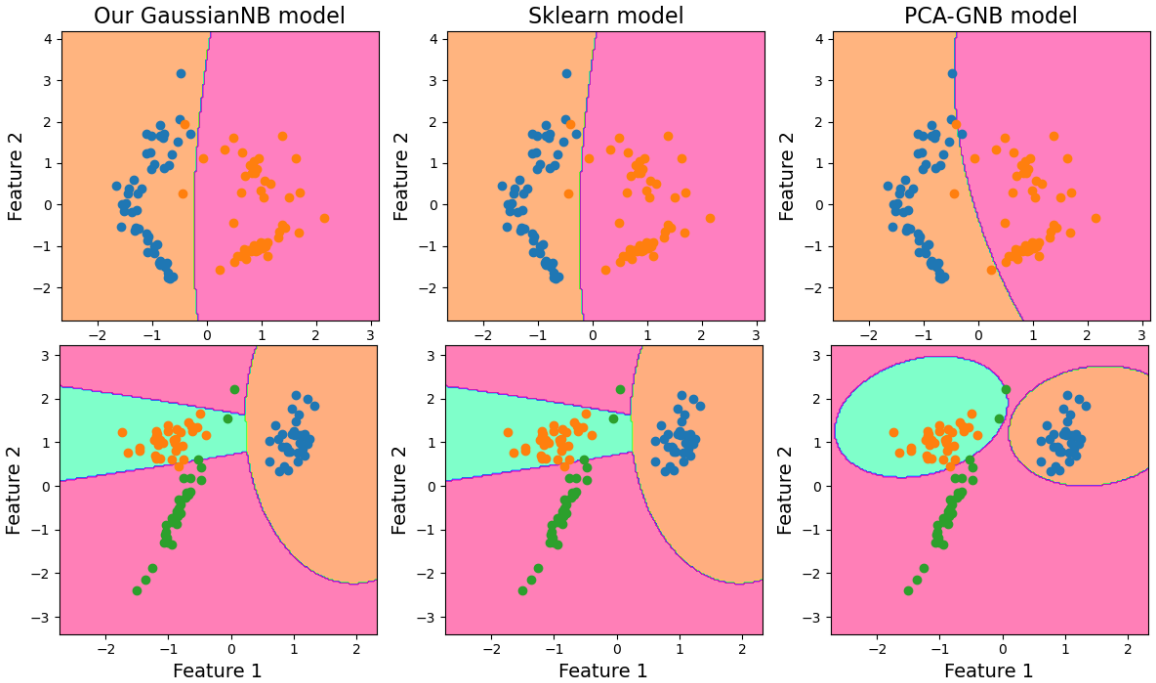






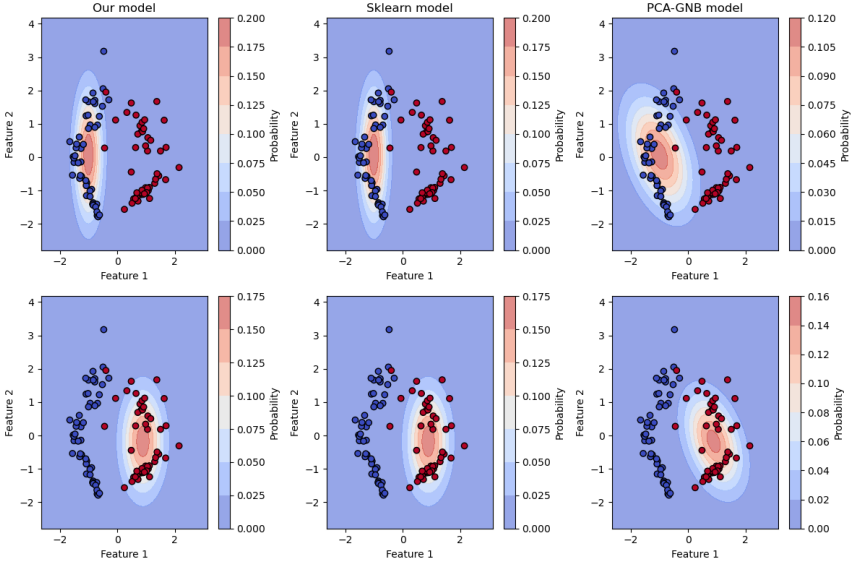
PCA-GNB

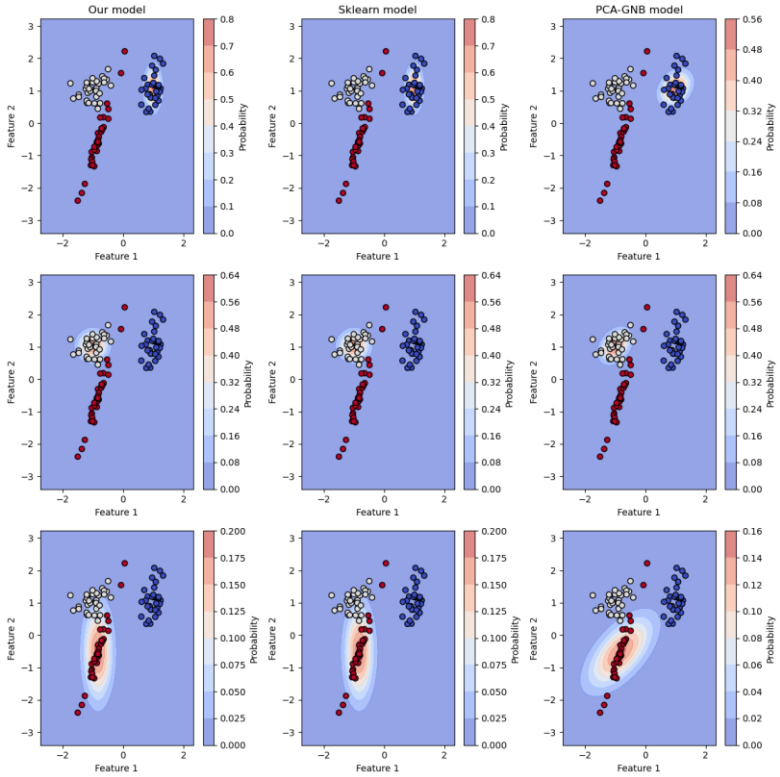
- Visualize the PCA-GNB model



PCA-GNB

- Visualize the Gaussian distribution
- We find that the Gaussian ellipses are now rotated in the same direction





What else?
Train different PCA models for different classes?

Conclusion

- We implemented the GNB model from the mean and variance from the training set and calculated the Gaussian distribution for each class.
- We used the obtained Gaussian distribution to calculate the likelihood of the test data and then predict.
- Our model has the same accuracy as scikit-learn package
- To stress the assumption of Naïve Bayes, we added a PCA before the GNB model, and it outperformed the single GNB model.

Q & A

- Thanks for listening.