# Gaussian Naive Bayes Modeling Project

Eva Azazoglu, Zoey Katzive, Alex Manioudakis, Huda Saeed

# What is Gaussian Naive Bayes?

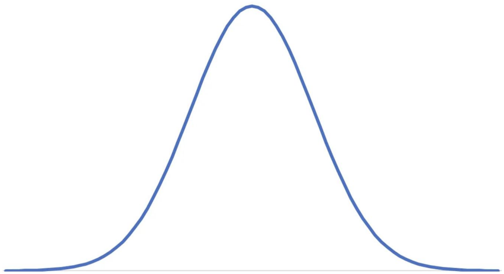
# Naive Bayes Models

- Generative modeling: goal is to estimate the parameters of the distribution of data
- Core assumption: features are conditionally independent given the class
- Priors and likelihoods derived from Bayes' Theorem

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)}$$

# Gaussian Naive Bayes

- Tailored for continuous features
- Assumes features given each label follow a Gaussian (normal) distribution



# Advantages

- Computationally efficient and simple
  - Only calculates mean and variance
- Handles high-dimensional data well
  - Does not require storing or processing individual data points once parameters are learned
- Probabilistic outputs allow for nuanced interpretation of results
  - Outputs probability associated with each predicted class
- Independence assumption reduces “noise” from less relevant features

# Limitations

- Does not capture dependencies between features
  - Conditional independence assumption does not hold in real-life data
- Performance weakens when distribution is not perfectly normal or decision boundaries between classes are complex
- Sensitive to outliers
- May not identify minority classes accurately when the classes are imbalanced
  - Prior probability $P(Y)$ influences the posterior probability

# The Math Behind the Algorithm

# Representation

Consists of the domain, range, sampling method, and function

1. All d features assumed to be normally distributed and independent given the class label → $X \in \mathbb{R}^d$
2. The range is a class label; for K labels we have → $Y \in \{0, 1, \ldots, K\}$
3. The sample is taken from the features (normal distribution assumed conditioned on label) paired with the label outcome (unknown distribution)
4. Observation → Output (class that has the maximum probability for an observation)

## GNB Function

- Prior probability calculated
    - Number of times that a class occured/number of rows in the dataset
- For each feature, subset based on the class → calculate mean and standard deviation
    - Result: K*D mean and variance pairs (K: number of classes, D: number of features)
- Observation's value for each attribute and each mean-variance pair are plugged into the Gaussian probability function:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$x$: observation's value for attribute d
$\mu$: mean of attribute d given class k
$\sigma$2: variance of attribute d given class k

# Representation: Bayes Theorem

$$P(Y|X) = \frac{P(X|Y) \cdot P(Y)}{P(X)}$$

*Bayes' Theorem*

$$P(Y|x_1, x_2, \dots, x_D) \propto P(Y) \prod_{i=1}^{D} P(x_i|Y)$$

*Bayes' Theorem in our model*

$$\log P(Y) + \sum_{i=1}^{D} \log P(x_i|Y)$$

*Convert to log space*

$$P(Y|x_1, x_2, \dots, x_D) \propto \exp\left( \log P(Y) + \sum_{i=1}^{D} \log P(x_i|Y) \right)$$

*Exponentiate to convert to probability*

# Loss

We can define the log loss of a Gaussian Naive Bayes algorithm over all N data points:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log P(y_i|x_i, \theta)$$

- Theta → the parameters of the function for Gaussian likelihood (mean and variance)
- P → the Gaussian probability

## Optimizer

- Empirical risk minimization → want to find the minimum of the loss
- For *m* data points, we want

$$\arg\min_{\theta} \mathcal{L}(\theta)$$

- Note: Naive Bayes is a generative algorithm, so the loss and optimizer exist in relation to distributions of the continuous features.

# Pseudocode

```
GIVEN:
      n_classes: number of classes
      X: 2D array representing training features
      y: 1D array representing training labels
      inputs: 2D array representing new examples to predict
```

# Pseudocode

```
TRAIN(X, y):
     Initialize priors as a 1D array of length n_classes to all zeros
     Initialize means as a 2D array of zeros with dimensions (n_classes, d)
     Initialize stds as a 2D array of zeros with dimensions (n_classes, d)

     For each class j:
          class_count = number of elements in y that are equal to j
          priors[j] = class_count / (total number of elements in y)

     For each class j:
          For each feature i:
               Determine ind as the indices of all rows in y where y == j
               means[j, i] = the average value of feature i across ind
               stds[j, i] = the standard deviation of feature i across ind

     Return priors, means, stds
```

# Pseudocode

```
PREDICT(inputs):
      log_priors = natural log of priors
      Initialize predictions as empty array

      For each input i:
            Set prob = copy of log_priors
            For each class j:
                  Add the log of the Gaussian PDF for each feature f to prob[j]
```

$$-\frac{1}{2}\ln\left(2\pi\sigma_{j,f}^2\right) - \frac{(x_{i,f}-\mu_{j,f})^2}{2\sigma_{j,f}^2}$$

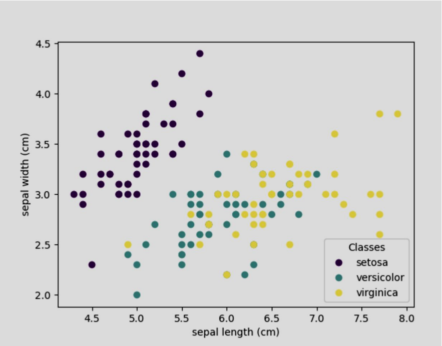
```
                  prob[j] = exponential_function(prob[j])

            Identify the class that corresponds to highest probability
            Append that class to predictions

      Return predictions as an array
```

# Our Project

# The Iris Dataset




- Classic machine learning experimentation dataset
- Consists of 150 observations from three distinct species of iris
- Includes measurements of four key features: sepal length, sepal width, petal length, and petal width
- Each sample is accompanied by a label that indicates its species

# Reproducing a GNB Model on Iris

- Bernd Klein's "Machine Learning with Python Tutorial" textbook implements a Gaussian Naive Bayes classifier on the Iris dataset using scikit-learn
- Outputs to match: precision, recall, f1 score, support, confusion matrix
- Work to reproduce: classification report and confusion matrix

```
GaussianNB()
             precision    recall  f1-score   support

          0       1.00      1.00      1.00        50
          1       0.94      0.94      0.94        50
          2       0.94      0.94      0.94        50

avg / total       0.96      0.96      0.96       150

[[50  0  0]
 [ 0 47  3]
 [ 0  3 47]]
```

# Our Process

- Create a class GNB() with methods for train, predict, classification report, and confusion matrix
- Create unit tests to validate train and predict methods
- Run GNB() on the Iris dataset
- Verify that output matches reproduced work (it does!)

# Gaussian Naive Bayes in the Real World

# Other Works

**“Human Activity Recognition Using Gaussian Naïve Bayes Algorithm in Smart Home” (Shen & Fang, 2020)**

*https://iopscience.iop.org/article/10.1088/1742-6596/1631/1/012059*

- **IoT sensors to predict one’s activities in the household**
- Used GNB to generate distributions on attributes created from sensor data
- Assumption that continuous attributes are normally distributed and independent
- Features were constructed from collected sensor data to show things like “time spent on activity” or “number of ‘on’ sensors”
- Optimal feature values are determined throughout the experiment
- 7% higher accuracy with GNB (89.5%) than with NB (82.7%)
- Impact: aging populations = increased need to cater to the at-home needs of the elderly
    - IoT can help, but ethical issues with privacy implications

# Takeaways

# Summary

Challenges:

- Finding work to reproduce with purely quantitative features
- Defining the loss and the optimizer for a generative model

What we found interesting:

- How viable GNB is in applications despite strong assumptions of normality + independence
- Logistic regression is often a much better alternative
- Works for discrete quantitative features

# Questions?