



# CS 6820 – Machine Learning

Lecture 11

Instructor: Eric S. Gayles, PhD.

Feb 14, 2018

# Project 2 – Due 3/2

- Build a Machine Learning model that maps a movie name, 2 lead actors, and release date to a movie classification
- Inputs: (Yes, there are extra credit opportunities)
  - Training & Test data – you build as a team. Everyone will use the same training data
    - You define the training data and the fields.
  - Note: The grading test data will come later 😊
- Grading:
  - 5 points if you are managing part of the process for the entire class
    - Must be confirmed by the class
  - 1-5 points for having a significant contribution for the class effort
    - Must be confirmed by the activity's manager
  - 20 points if your model is in the top 4 for accuracy
  - 10 points if your model is in the next 6 for accuracy
  - 30 points if your model does a reasonable job of prediction
  - 0-25 points for just trying to build a model that at least makes a prediction
  - 25 points for writing code that looks remotely like it might do something related to Machine Learning

# Movie Classes

- Action
- Adventure
- Sci-Fi
- Fantasy
- Drama
- Comedy
- Horror
- Thriller
- Documentary
- Romance
- Western
- Animation
- War
- History
- Sports
- Musical
- Biography
- Family

# Data Modeling

- In the real world, Data is often produced from a process that is not completely known.
- We model this uncertainty as a random process.
- The process might actually be deterministic.
- However because we do not have access to complete knowledge about it, we model it as random and use probability theory to analyze it.

# Data Modeling

- The extra pieces of knowledge that we do not have access to are the *unobservable variables*.
- In the coin tossing example, the only *observable variable* is the outcome of the toss.
- $\mathbf{x} = \mathbf{f}(\mathbf{z})$  where, the un-observables are  $\mathbf{z}$  and the observable is  $\mathbf{x}$
- $\mathbf{f}()$  is the deterministic function that defines the outcome from the unobservable pieces of knowledge.
- We define the outcome  $\mathbf{X}$  as a random variable drawn from a probability distribution  $\mathbf{P}(\mathbf{X} = \mathbf{x})$  that specifies the process.

# Data Modeling

- We've discussed credit scoring examples related to Machine Learning
- In the financial services industry some customers are flagged as “low-risk” while others are labelled “high-risk”.
- Analyzing data, we would like to learn the class “high-risk customer” so that in the future, when there is a new application for a loan, we can reduce default risk
- We choose attributes that are observable.
- We observe them because we have reason to believe that they give us an idea about the credibility of a customer.
- For example, we observe customer's yearly income and savings, which we represent by two random variables  $X_1$  and  $X_2$ .

# Classification

- Credit scoring: Inputs are income and savings.
- Output is low-risk vs high-risk
- Input:  $\mathbf{x} = [x_1, x_2]^T$ , Output:  $C \in \{0, 1\}$

- Prediction:

$$\text{choose } \begin{cases} C = 1 & \text{if } P(C = 1 | x_1, x_2) > 0.5 \\ C = 0 & \text{otherwise} \end{cases}$$

or

$$\text{choose } \begin{cases} C = 1 & \text{if } P(C = 1 | x_1, x_2) > P(C = 0 | x_1, x_2) \\ C = 0 & \text{otherwise} \end{cases}$$

# Naive Bayes

- Learning and classification methods based on probability theory.
- Bayes theorem plays a critical role in probabilistic learning and classification.
- Uses *prior* probability of each category given no information about an item.
- **Categorization** produces a ***posterior* probability** distribution over the possible categories given a description of an item.



# Naive Bayes Algorithm

- Based on [Bayes' Theorem](#) with an assumption of independence among predictors.
- **In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.**
- For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter.
- Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.
- Naive Bayes model is easy to build and particularly useful for very large data sets.
- Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

# Naive Bayes

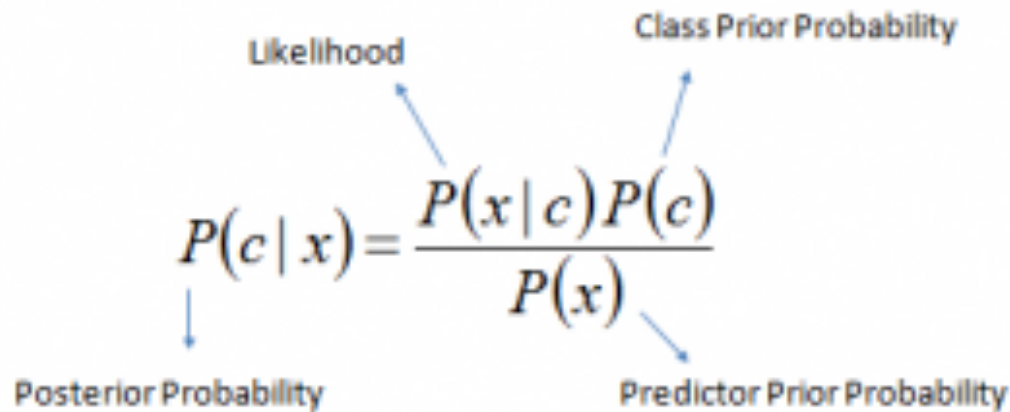
- Naive Bayes is a simple technique for **constructing classifiers**: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set.
- All naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.
- Abstractly, Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector

$$\mathbf{x} = (x_1, \dots, x_n)$$

- Representing some  $n$  features (independent variables), it assigns to this instance probabilities for each of  $K$  possible outcomes or classes

$$p(C_k \mid x_1, \dots, x_n)$$

# Bayes Theorem



A diagram showing the Bayes' Theorem formula with labels and arrows. The formula is  $P(c | x) = \frac{P(x | c)P(c)}{P(x)}$ . Arrows point from labels to parts of the formula: 'Likelihood' points to  $P(x | c)$ , 'Class Prior Probability' points to  $P(c)$ , 'Posterior Probability' points to  $P(c | x)$ , and 'Predictor Prior Probability' points to  $P(x)$ .

$$P(c | x) = \frac{P(x | c)P(c)}{P(x)}$$

Likelihood

Class Prior Probability

Posterior Probability

Predictor Prior Probability

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \cdots \times P(x_n | c) \times P(c)$$

# Independence Assumption

- Independence assumption (in practice we focus on the numerator)

$$p(x_i \mid x_{i+1}, \dots, x_n, C_k) = p(x_i \mid C_k) .$$

Thus, the joint model can be expressed as

$$\begin{aligned} p(C_k \mid x_1, \dots, x_n) &\propto p(C_k, x_1, \dots, x_n) \\ &\propto p(C_k) p(x_1 \mid C_k) p(x_2 \mid C_k) p(x_3 \mid C_k) \dots \\ &\propto p(C_k) \prod_{i=1}^n p(x_i \mid C_k) . \end{aligned}$$

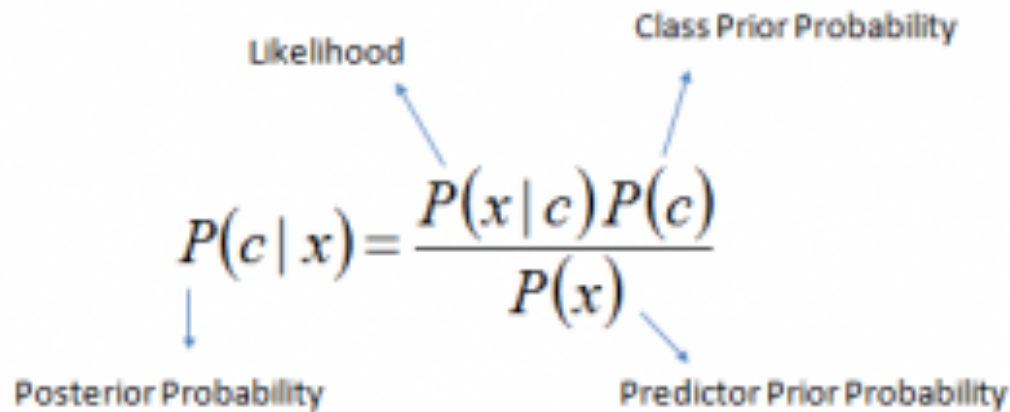
- where the evidence becomes a scaling factor

$$Z = p(\mathbf{x}) = \sum_k p(C_k) p(\mathbf{x} \mid C_k) \text{ is a scaling factor dependent only on } x_1, \dots, x_n .$$

Chain rule for repeated application of the definition of conditional probability

$$\begin{aligned} p(C_k, x_1, \dots, x_n) &= p(x_1, \dots, x_n, C_k) \\ &= p(x_1 \mid x_2, \dots, x_n, C_k) p(x_2, \dots, x_n, C_k) \\ &= p(x_1 \mid x_2, \dots, x_n, C_k) p(x_2 \mid x_3, \dots, x_n, C_k) p(x_3, \dots, x_n, C_k) \\ &= \dots \\ &= p(x_1 \mid x_2, \dots, x_n, C_k) p(x_2 \mid x_3, \dots, x_n, C_k) \dots p(x_{n-1} \mid x_n, C_k) p(x_n \mid C_k) p(C_k) \end{aligned}$$

# Bayes Theorem



A diagram showing the Bayes' Theorem formula with labels and arrows. The formula is  $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$ . Arrows point from labels to parts of the formula: 'Likelihood' points to  $P(x|c)$ , 'Class Prior Probability' points to  $P(c)$ , 'Posterior Probability' points to  $P(c|x)$ , and 'Predictor Prior Probability' points to  $P(x)$ .

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Labels and arrows:  
Likelihood (points to  $P(x|c)$ )  
Class Prior Probability (points to  $P(c)$ )  
Posterior Probability (points to  $P(c|x)$ )  
Predictor Prior Probability (points to  $P(x)$ )

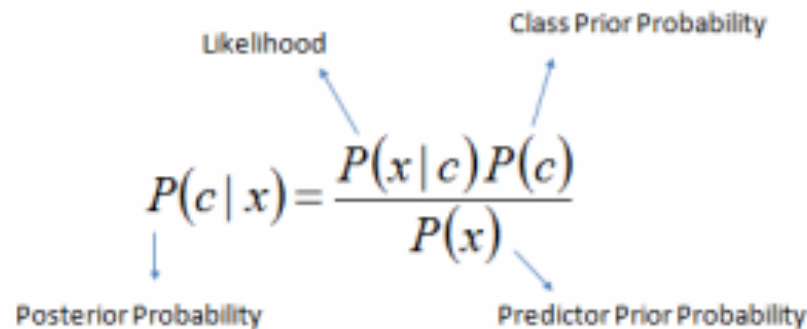
$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \cdots \times P(x_n | c) \times P(c)$$

# Bayes Theorem

- $P(\mathbf{c}/\mathbf{x})$  is the posterior probability of *class*  $c$  from the set of *target classes* given *predictor*  $\mathbf{x}$  from the set of *attributes*.
- $P(\mathbf{c})$  is the prior probability of *class*.
- $P(\mathbf{x}/\mathbf{c})$  is the **class likelihood** which is the probability of *predictor*  $\mathbf{x}$  given *class*  $\mathbf{c}$ .
- $P(\mathbf{x})$  is the prior probability of *predictor*.
  - the evidence or **marginal probability** that an observation  $\mathbf{x}$  is seen, regardless of whether it is a positive or negative example of the class.

$$p(\mathbf{x}) = \sum_c p(\mathbf{x}, C) = p(\mathbf{x}|C = 1)P(C = 1) + p(\mathbf{x}|C = 0)P(C = 0)$$

# Bayes Theorem



A diagram showing the Bayes' Theorem formula  $P(c | x) = \frac{P(x | c)P(c)}{P(x)}$ . Blue arrows point from labels to the corresponding parts of the formula: 'Likelihood' points to  $P(x | c)$ , 'Class Prior Probability' points to  $P(c)$ , 'Posterior Probability' points to  $P(c | x)$ , and 'Predictor Prior Probability' points to  $P(x)$ .

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \cdots \times P(x_n | c) \times P(c)$$

- $P(c|x)$  is the posterior probability of *class*  $c$  from the set of *target classes* given *predictor*  $x$  from the set of *attributes*.
- $P(c)$  is the prior probability of *class*.
- $P(x/c)$  is the **class likelihood** which is the probability of *predictor*  $x$  given *class*  $c$ .
- $P(x)$  is the prior probability of *predictor*.

# Bayesian Methods

$$P(C_i) \geq 0 \text{ and } \sum_{i=1}^K P(C_i) = 1$$

$p(\mathbf{x}|C_i)$  is the probability of seeing  $\mathbf{x}$  as the input when it is known to belong to class  $C_i$ . The posterior probability of class  $C_i$  can be calculated as

$$P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_i)P(C_i)}{\sum_{k=1}^K p(\mathbf{x}|C_k)P(C_k)}$$



# Naive Bayes

- Constructing a classifier from the probability model
  - The naive Bayes classifier combines the probability model with a decision rule.
  - One common rule is to pick the hypothesis that is most probable; this is known as the **maximum a posteriori or MAP decision rule**.
  - The corresponding classifier, a Bayes classifier, is the function that assigns a class label

$$\hat{y} = C_k$$

- for some  $k$  as follows:

$$\hat{y} = \operatorname{argmax}_{k \in \{1, \dots, K\}} p(C_k) \prod_{i=1}^n p(x_i \mid C_k).$$

# Naive Bayes

- $P(c_j)$ 
  - Can be estimated from the frequency of classes in the training examples.
- $P(x_1, x_2, \dots, x_n | c_j)$ 
  - $O(|X|^n \cdot |C|)$  parameters
  - Could only be estimated if a very, very large number of training examples was available.
- **Independence Assumption**: attribute values are conditionally independent given the target value: **naïve Bayes**.

$$P(x_1, x_2, \dots, x_n | c_j) = \prod_i P(x_i | c_j)$$

$$c_{NB} = \arg \max_{c_j \in C} P(c_j) \prod_i P(x_i | c_j)$$

# Bayesian Methods

- **Real time Prediction:** Naive Bayes is an eager learning classifier and it is sure fast. Thus, it could be used for making predictions in real time.
- **Multi class Prediction:** This algorithm is also well known for multi class prediction feature. Here we can predict the probability of multiple classes of target variable.
- **Text classification/ Spam Filtering/ Sentiment Analysis:** Naive Bayes classifiers mostly used in text classification (due to better result in multi class problems and independence rule) have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments)
- **Recommendation System:** Naive Bayes Classifier and [Collaborative Filtering](#) together builds a Recommendation System that uses machine learning and data mining techniques to filter unseen information and predict whether a user would like a given resource or not

# Bayesian Methods

- Scikit learn (python library) can help to build a Naive Bayes model in Python. There are three types of Naive Bayes model under scikit learn library:
- **Gaussian**: It is used in classification and it assumes that features follow a normal distribution.
- **Multinomial**: It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider Bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", you can think of it as "number of times outcome number  $x_i$  is observed over the  $n$  trials".
- **Bernoulli**: The binomial model is useful if your feature vectors are binary (i.e. zeros and ones). One application would be text classification with 'bag of words' model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.

# Naive Bayes

## 1.9.1. Gaussian Naive Bayes

**GaussianNB** implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:

$$P(x_i | y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The parameters  $\sigma_y$  and  $\mu_y$  are estimated using maximum likelihood.

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(iris.data, iris.target).predict(iris.data)
>>> print("Number of mislabeled points out of a total %d points : %d"
...       % (iris.data.shape[0], (iris.target != y_pred).sum()))
Number of mislabeled points out of a total 150 points : 6
```

# Naive Bayes

## 1.9.2. Multinomial Naive Bayes

**MultinomialNB** implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors  $\theta_y = (\theta_{y1}, \dots, \theta_{yn})$  for each class  $y$ , where  $n$  is the number of features (in text classification, the size of the vocabulary) and  $\theta_{yi}$  is the probability  $P(x_i \mid y)$  of feature  $i$  appearing in a sample belonging to class  $y$ .

The parameters  $\theta_y$  is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where  $N_{yi} = \sum_{x \in T} x_i$  is the number of times feature  $i$  appears in a sample of class  $y$  in the training set  $T$ , and  $N_y = \sum_{i=1}^{|T|} N_{yi}$  is the total count of all features for class  $y$ .

The smoothing priors  $\alpha \geq 0$  accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting  $\alpha = 1$  is called Laplace smoothing, while  $\alpha < 1$  is called Lidstone smoothing.

# Naive Bayes

## 1.9.3. Bernoulli Naive Bayes

`BernoulliNB` implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. Therefore, this class requires samples to be represented as binary-valued feature vectors; if handed any other kind of data, a `BernoulliNB` instance may binarize its input (depending on the `binarize` parameter).

The decision rule for Bernoulli naive Bayes is based on

$$P(x_i | y) = P(i | y)x_i + (1 - P(i | y))(1 - x_i)$$

which differs from multinomial NB's rule in that it explicitly penalizes the non-occurrence of a feature  $i$  that is an indicator for class  $y$ , where the multinomial variant would simply ignore a non-occurring feature.

In the case of text classification, word occurrence vectors (rather than word count vectors) may be used to train and use this classifier. `BernoulliNB` might perform better on some datasets, especially those with shorter documents. It is advisable to evaluate both models, if time permits.

# Bayesian Methods

$$P(C_i) \geq 0 \text{ and } \sum_{i=1}^K P(C_i) = 1$$

$p(\mathbf{x}|C_i)$  is the probability of seeing  $\mathbf{x}$  as the input when it is known to belong to class  $C_i$ . The posterior probability of class  $C_i$  can be calculated as

$$P(C_i|\mathbf{x}) = \frac{p(\mathbf{x}|C_i)P(C_i)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|C_i)P(C_i)}{\sum_{k=1}^K p(\mathbf{x}|C_k)P(C_k)}$$



# Bayesian Methods

- Based on Bayes Theorem, we can compute the *Maximum A Posterior* (MAP) hypothesis for the data
- We are interested in the best hypothesis for some space  $H$  given observed training data  $D$ .

all the attributes are conditionally independent given the class

$$P(\mathbf{x} \mid c_j) = \prod_{i=1}^n P(X_i = x_i \mid c_j)$$

$P(\mathbf{x} \mid c_j)$  can be decomposed into a product of  $n$  terms, one term for each attribute

$$c^* = h_{NB}(\mathbf{x}) = \arg \max_{j=1..m} P(c_j) \prod_{i=1}^n P(X_i = x_i \mid c_j)$$

Given an arbitrary set  $X$ , a totally ordered set  $Y$ , and a function,  $f: X \rightarrow Y$ , the  $\arg \max$  over some subset,  $S$ , of  $X$  is defined by

$$\arg \max_{x \in S \subseteq X} f(x) := \{x \mid x \in S \wedge \forall y \in S : f(y) \leq f(x)\}.$$

# Advantages

- Probabilistic learning: Calculate explicit probabilities for hypothesis, among the most practical approaches to certain types of learning problems
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct. Prior knowledge can be combined with observed data.
- Probabilistic prediction: Predict multiple hypotheses, weighted by their probabilities
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

## 8.20.1. sklearn.naive\_bayes.GaussianNB

*class* sklearn.naive\_bayes.**GaussianNB**

Gaussian Naive Bayes (GaussianNB)

**Parameters :** **X** : array-like, shape = [n\_samples, n\_features]

Training vector, where n\_samples is the number of samples and n\_features is the number of features.

**y** : array, shape = [n\_samples]

Target vector relative to X

### Examples

```
>>> import numpy as np
>>> X = np.array([[ -1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> Y = np.array([1, 1, 1, 2, 2, 2])
>>> from sklearn.naive_bayes import GaussianNB
>>> clf = GaussianNB()
>>> clf.fit(X, Y)
GaussianNB()
>>> print clf.predict([[-0.8, -1]])
[1]
```

## Attributes

<i>class_prior_</i>	array, shape = [n_classes]	probability of each class.
<i>theta_</i>	array, shape = [n_classes, n_features]	mean of each feature per class
<i>sigma_</i>	array, shape = [n_classes, n_features]	variance of each feature per class

## Methods

<b>fit</b> (X, y)	Fit Gaussian Naive Bayes according to X, y
<b>get_params</b> ([deep])	Get parameters for the estimator
<b>predict</b> (X)	Perform classification on an array of test vectors X.
<b>predict_log_proba</b> (X)	Return log-probability estimates for the test vector X.
<b>predict_proba</b> (X)	Return probability estimates for the test vector X.
<b>score</b> (X, y)	Returns the mean accuracy on the given test data and labels.
<b>set_params</b> (**params)	Set the parameters of the estimator.

### **\_\_init\_\_**()

x.\_\_init\_\_(...) initializes x; see help(type(x)) for signature

### **class\_prior**

DEPRECATED: GaussianNB.class\_prior is deprecated and will be removed in version 0.12. Please use `GaussianNB.class_prior_` instead.

### **fit**(X, y)

Fit Gaussian Naive Bayes according to X, y

**Parameters :** **X** : array-like, shape = [n\_samples, n\_features]

Training vectors, where n\_samples is the number of samples and n\_features is the number of features.

**y** : array-like, shape = [n\_samples]

Target values.

## 1.9. Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. Given a class variable  $y$  and a dependent feature vector  $x_1$  through  $x_n$ , Bayes' theorem states the following relationship:

$$P(y \mid x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

Using the naive independence assumption that

$$P(x_i \mid y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i \mid y),$$

for all  $i$ , this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^n P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Since  $P(x_1, \dots, x_n)$  is constant given the input, we can use the following classification rule:

$$\begin{aligned} P(y \mid x_1, \dots, x_n) &\propto P(y) \prod_{i=1}^n P(x_i \mid y) \\ &\Downarrow \\ \hat{y} &= \arg \max_y P(y) \prod_{i=1}^n P(x_i \mid y), \end{aligned}$$

and we can use Maximum A Posteriori (MAP) estimation to estimate  $P(y)$  and  $P(x_i \mid y)$ ; the former is then the relative frequency of class  $y$  in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of  $P(x_i \mid y)$ .