Classification – Naïve Bayes

Natural Language Processing
Issues
Applications

M. R. Hasan
Readings

• Bishop: 2.1, 2.2
• Murphy: 1.2, 1.3, 2.3.1, 2.3.2, 3.5, 3.5.1, 3.5.2, 3.5.3, 3.5.4, 3.5.5
Naïve Bayes Classifier

• To ground our discussion into **practical scenarios**, let’s consider a document classification problem.
• Consider an example of a **set of emails** that consists of both spam and non-spam (or ham) emails.
• We want to **automatically classify** a new email pertaining to either spam or ham.

<table>
<thead>
<tr>
<th>Index</th>
<th>Document</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“toy”, “kid”, “toy”</td>
<td>Ham</td>
</tr>
<tr>
<td>2</td>
<td>“lottery”, “Congratulations”, “lottery”, “toy”</td>
<td>Spam</td>
</tr>
<tr>
<td>3</td>
<td>“Congratulations”, “paper”, “accepted”, “win”</td>
<td>Ham</td>
</tr>
<tr>
<td>4</td>
<td>“lottery”, “win”, “prize”, “win”</td>
<td>Spam</td>
</tr>
</tbody>
</table>
Naïve Bayes Classifier: Text Classification

• We search through the set of spam and ham emails and **find the most commonly occurring words or terms** (not including so-called ‘stop words’ such as ‘a’ or ‘the’).
• Let’s say that we have found $d$ number of **unique terms**.
• Using these terms we create a **vocabulary** or **dictionary**.

<table>
<thead>
<tr>
<th>Index</th>
<th>Document</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“toy”, “kid”, “toy”</td>
<td>Ham</td>
</tr>
<tr>
<td>2</td>
<td>“lottery”, “congratulations”, “lottery”, “toy”</td>
<td>Spam</td>
</tr>
<tr>
<td>3</td>
<td>“congratulations”, “paper”, “accepted”, “win”</td>
<td>Ham</td>
</tr>
<tr>
<td>4</td>
<td>“lottery”, “win”, “prize”, “win”</td>
<td>Spam</td>
</tr>
</tbody>
</table>

**Vocabulary** = [“toy”, “kid”, “lottery”, “congratulations”, “paper”, “accepted”, “win”, “prize”]
Naïve Bayes Classifier: Text Classification

• The terms in the Vocabulary are called **features**.
• We **represent an email** via a **feature vector**.
• Its **length** is equal to the number of terms in the vocabulary/dictionary.

<table>
<thead>
<tr>
<th>Index</th>
<th>Document</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“toy”, “kid”, “toy”</td>
<td>Ham</td>
</tr>
<tr>
<td>2</td>
<td>“lottery”, “congratulations”, “lottery”, “toy”</td>
<td>Spam</td>
</tr>
<tr>
<td>3</td>
<td>“congratulations”, “paper”, “accepted”, “win”</td>
<td>Ham</td>
</tr>
<tr>
<td>4</td>
<td>“lottery”, “win”, “prize”, “win”</td>
<td>Spam</td>
</tr>
</tbody>
</table>

**Vocabulary** = [“toy”, “kid”, “lottery”, “congratulations”, “paper”, “accepted”, “win”, “prize”]
Naïve Bayes Classifier: Text Classification

• Thus, each email is represented by a **d-dimensional vector** of features.
• This is called the “**bag of words**” representation.
• This is a **crude representation** of the document since it **discards word order**.

<table>
<thead>
<tr>
<th>Index</th>
<th>Document</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“toy”, “kid”, “toy”</td>
<td>Ham</td>
</tr>
<tr>
<td>2</td>
<td>“lottery”, “congratulations”, “lottery”, “toy”</td>
<td>Spam</td>
</tr>
<tr>
<td>3</td>
<td>“congratulations”, “paper”, “accepted”, “win”</td>
<td>Ham</td>
</tr>
<tr>
<td>4</td>
<td>“lottery”, “win”, “prize”, “win”</td>
<td>Spam</td>
</tr>
</tbody>
</table>
Naïve Bayes Classifier

- **Each sample** in the training data (i.e., document) is represented as a **feature vector** $\mathbf{x} = [x_1 \ldots x_d]$
- For example, the features are the **terms** (words) in the vocabulary.
- The number of features is $d$.
- The number of **possible values of each feature** is $M$: $\mathbf{x} \in \{0, \ldots, M\}^d$
- Here $M$ denotes the number of terms in a particular document.

<table>
<thead>
<tr>
<th>Index</th>
<th>Document</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“toy”, “kid”, “toy”</td>
<td>Ham</td>
</tr>
<tr>
<td>2</td>
<td>“lottery”, “congratulations”, “lottery”, “toy”</td>
<td>Spam</td>
</tr>
<tr>
<td>3</td>
<td>“congratulations”, “paper”, “accepted”, “win”</td>
<td>Ham</td>
</tr>
<tr>
<td>4</td>
<td>“lottery”, “win”, “prize”, “win”</td>
<td>Spam</td>
</tr>
</tbody>
</table>
Naïve Bayes Classifier

• In the feature vector $\vec{x} = [x_1 \ldots x_d]$, each $x$ is a **random variable**.

• The random variable $y$ represents the **class of the sample** (e.g., Ham or Spam).

• It can be one of C **discrete states**:

$$y = \{y_1, y_2, y_3, \ldots, y_C\}$$

<table>
<thead>
<tr>
<th>Index</th>
<th>Document</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“toy”, “kid”, “toy”</td>
<td>Ham</td>
</tr>
<tr>
<td>2</td>
<td>“lottery”, “congratulations”, “lottery”, “toy”</td>
<td>Spam</td>
</tr>
<tr>
<td>3</td>
<td>“congratulations”, “paper”, “accepted”, “win”</td>
<td>Ham</td>
</tr>
<tr>
<td>4</td>
<td>“lottery”, “win”, “prize”, “win”</td>
<td>Spam</td>
</tr>
</tbody>
</table>
Naïve Bayes Classifier

- We have \( N \) training examples.
  \[
  D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}
  \]
- Our goal is to **classify the feature vector** \( \mathbf{x} \in \{0, 1, \ldots, M\}^d \)
- In other words, we want to determine the **conditional probability** of the class \( y = c \) given the feature vector \( \mathbf{x} \):
  \[
  p(y = c \mid \mathbf{x})
  \]
- Out of all possible states of \( y \), we want to pick the state that maximizes the probability.

<table>
<thead>
<tr>
<th>Index</th>
<th>Document</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“toy”, “kid”, “toy”</td>
<td>Ham</td>
</tr>
<tr>
<td>2</td>
<td>“lottery”, “congratulations”, “lottery”, “toy”</td>
<td>Spam</td>
</tr>
<tr>
<td>3</td>
<td>“congratulations”, “paper”, “accepted”, “win”</td>
<td>Ham</td>
</tr>
<tr>
<td>4</td>
<td>“lottery”, “win”, “prize”, “win”</td>
<td>Spam</td>
</tr>
</tbody>
</table>

Classify document 1:
\[
\mathbf{x} = \{“toy”, “kid”\}
\]
\[
p(y = Ham \mid \mathbf{x})
\]
\[
p(y = Spam \mid \mathbf{x})
\]
Naïve Bayes Classifier

• For this classification problem we will use a **generative approach**:
  \[ p(\hat{x}, y = c) = p(y = c)p(\hat{x} | y = c) \]

• We compute \( p(y = c) \) and \( p(\hat{x} | y = c) \).

• Then, using the Bayes’ rule, we get the **conditional probability** of the class \( y = c \) given the feature vector \( \hat{x} \):
  \[ p(y = c | \hat{x}) \propto p(y = c)p(\hat{x} | y = c) \]

• Applying the **NB assumption** (i.e., the features are conditionally independent given the class):
  \[ p(y = c | \hat{x}) \propto p(y = c) \prod_{j=1}^{d} p(x_j | y = c) \]
Naïve Bayes Classifier

• Carefully note the key aspect of the NB algorithm.
• Our goal is to learn the discriminative function $p(y = c | \hat{x})$
• To do this, we model the sample & class probabilities.

$$p(y = c | \hat{x}) \propto p(y = c)p(\hat{x} | y = c)$$

Thus, NB algorithm is rich as it enables us to model (reproduce) the reality!
Naïve Bayes Classifier

• \(p(y = c \mid \mathbf{x}) \propto p(y = c) \prod_{j=1}^{d} p(x_j \mid y = c)\)
• \(p(y = c)\): prior probability distribution of the class (e.g., number of documents belonging to a class)
  • It will be denoted by \(\pi_c\)
• \(p(x_j \mid y)\): likelihood of a feature component given the class
  • It will be denoted by \(\theta_{jc}\)
• Here \(\theta_{jc}\) is the probability that feature \(j\) occurs in class \(c\).
• Note that \(\mathbf{\theta}\) is a d-dimensional vector: for every feature it gives the probability of the appearance of that feature in a class.
Naïve Bayes Classifier

• Thus the **posterior** distribution for a **single sample** $\tilde{x}_i$ (e.g., document, indexed by $i$) is $p(y_i = c \mid \tilde{x}_i)$:

$$
p(y_i = c \mid \tilde{x}_i) = p(y_i) \prod_{j=1}^{d} p(x_{ij} \mid y_i = c)
$$

$$
p(y_i = c \mid \tilde{x}_i, \tilde{\pi}, \tilde{\theta}) = \pi_c \prod_{j=1}^{d} \theta_{jc}
$$

Our goal is to **learn the parameters** $\theta_{jc}$ & $\pi_c$
Naïve Bayes Classifier

• We will discuss **two types of features**:
  - Categorial Features
  - Real-Valued Features
Naïve Bayes Classifier

• For **categorical** features, we will consider **two possible cases**:

• **Case 1**: Feature values are Boolean (binary valued): $\tilde{x} \in \{0, 1\}^d$

• For example, it captures whether a word **occurs** in the document (1) or not (0).

• **Case 2**: Feature values are K-dimensional categorical.

• For example, feature values are **count of feature occurrences**, showing **how many times** the corresponding word occurs in the document.
Naïve Bayes Classifier

- For categorial features, we will consider **two possible cases**:

<table>
<thead>
<tr>
<th>Index</th>
<th>Document</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“toy”, “kid”, “toy”</td>
<td>Ham</td>
</tr>
<tr>
<td>2</td>
<td>“lottery”, “congratulations”, “lottery”, “toy”</td>
<td>Spam</td>
</tr>
<tr>
<td>3</td>
<td>“congratulations”, “paper”, “accepted”, “win”</td>
<td>Ham</td>
</tr>
<tr>
<td>4</td>
<td>“lottery”, “win”, “prize”, “win”</td>
<td>Spam</td>
</tr>
</tbody>
</table>

**Feature: “toy” & “lottery” in Document 1**

**Case 1 (Boolean Features):**
- “toy” = 1
- “lottery” = 0

**Case 2 (Multi-valued Categorical):**
- “toy” = 2
- “lottery” = 0
Naïve Bayes Classifier

• **Real-Valued Features**
  • Feature values are continuous (real-valued) numbers.
Naïve Bayes Classifier: Model Fitting

• How do we “train” a naïve Bayes classifier?
• There are two approaches for the training.
  - Frequentist Learning Approach
  - Bayesian Learning Approach

The Bayesian Learning approach prevents overfitting.

We can show that the Frequentist model is a special case of the Bayesian model.
Frequentist Learning Approach
Frequentist Learning Approach

• In the Frequentist approach, we **construct the likelihood function** for the data.
• Then, we **maximize the likelihood** using MLE.
• For constructing the likelihood function we need to **model the probability distribution** for two model parameters:
  • $\theta_{jc}$: Feature Likelihood
  • $\pi_c$: Class Prior

\[
p(y_i = c \mid \vec{x}_i) = p(y_i) \prod_{j=1}^{d} p(x_{ij} \mid y_i = c)
\]

\[
p(y_i = c \mid \vec{x}_i, \bar{\pi}, \bar{\theta}) = \pi_c \prod_{j=1}^{d} \theta_{jc}
\]
$$p(y_i = c | \mathbf{x}_i) = p(y_i) \prod_{j=1}^{d} p(x_{ij} | y_i = c)$$

For $N$ documents, class prior probability distribution is similar to the **multiple coin toss scenario**.

We need to determine: out of $N$ documents (e.g., $N$ coin tosses), 

#times class “c” occurred and #times class “c” didn’t occur.

Thus we can represent the class prior probability by using a **binomial distribution**:

$$p(y_i = c) \propto (\pi_c)^{\sum_{i=1}^{N} \mathbb{I}(y_i=c)} (1 - \pi_c)^{\sum_{i=1}^{N} \mathbb{I}(y_i=not \ c)}$$

$\sum_{i=1}^{N} \mathbb{I}(y_i = c)$ number of samples in class $c$.

$\sum_{i=1}^{N} \mathbb{I}(y_i = not \ c)$: number of samples not in class $c$. 
We consider whether in document $i$, the feature $j$ occurred or not.

It’s similar to a single coin toss in which either head/tail turns up.

Thus, feature likelihood $p(x_j \mid y = c)$ can be modeled as Bernoulli distribution for each class $c$ and each feature $j$:

$$p(x_j \mid y = c) = (\theta_{jc})^{x_j} (1 - \theta_{jc})^{1-x_j}$$
The multi-valued categorical feature, $x_{ij} ∈ \{0, 1, \ldots, M_i\}$, is similar to a side of a d-sided die.

For a document $i$, $M_i$ represents the total number of words (i.e., the total number of times the die is rolled).

We use $x_{ij}$ to denote the #times times feature $j$ occurred, (e.g., #times the $j$ side of the die showed up).

Thus, the distribution of the multi-valued categorical features can be represented by the Multinomial distribution.

$$p(\hat{x}_i \mid y_i = c) = \prod_{j=1}^{d} p(x_{ij} \mid y_i = c) = \frac{M_i!}{\prod_{j=1}^{d} x_{ij}!} \prod_{j=1}^{d} (\theta_{jc})^{x_{ij}}$$
Feature likelihood:

\[ p(y_i = c | \tilde{x}_i) = p(y_i) \prod_{j=1}^{d} p(x_{ij} | y_i = c) \]

Case 2: Multi-valued features

We consider in document \( i \) the **number of times** feature \( j \) occurred.

It’s similar to **multiple coin tosses** in which we count

#times head occurred and

#times it didn’t

For **document \( i \)**, that has \( M_i \) words, we use \( x_{ij} \) to denote the #times times

feature \( j \) occurred, and \( (M_i - x_{ij}) \) times

it **did not occur**.

Thus, feature likelihood \( p(x_j | y = c) \) can be modeled as

**binomial distribution** for each class \( c \) and each feature \( j \):

\[ p(x_{ij} | y_i = c) \propto (\theta_{jc})^{x_{ij}} (1 - \theta_{jc})^{M_i - x_{ij}} \]
Assume that for each class \( y = c \), the distribution of each continuous \( x_j \)
\( p(x_j | y = c) \) follows a **Gaussian distribution**.

For each document \( i \), the Gaussian distribution is defined by the **mean and standard deviation** specific to \( x_{ij} \) and \( y_i = c \):

\[
p(x_{ij} | y_i = c, \mu_{jc}, \sigma_{jc}^2) = \frac{1}{\sqrt{2\pi\sigma_{jc}^2}} e^{-\frac{1}{2\sigma_{jc}^2}(x_{ij} - \mu_{jc})^2}
\]

To compute feature likelihood, we need to **estimate** the mean and standard deviation of each of these Gaussians:

\[
\hat{\mu}_{jc} = \quad \hat{\sigma}_{jc}^2 =
\]
**Class Prior Probability Distribution:**

**Binomial distribution:**

\[ p(y_i = c) \propto (\pi_c)^{\sum_{i=1}^{N} I(y_i=c)}(1 - \pi_c)^{\sum_{i=1}^{N} I(y_i=\text{not } c)} \]

**Feature likelihood:**

**Case 1: Binary-valued features**

**Bernoulli distribution:**

\[ p(x_j | y = c) = (\theta_{jc})^{x_j}(1 - \theta_{jc})^{1-x_j} \]

**Case 2: Multi-valued features**

**Binomial distribution:**

\[ p(x_{ij} | y_i = c) \propto (\theta_{jc})^{x_{ij}}(1 - \theta_{jc})^{M_i-x_{ij}} \]

**Multinomial distribution:**

\[ p(\hat{x}_i | y_i = c) \propto \prod_{j=1}^{d}(\theta_{jc})^{x_{ij}} \]
Frequentist Learning Approach

- Multivariate Bernoulli NB model
- Multinomial NB model
- Gaussian NB model

Categorical Features are Binary Valued
Categorical Features are Multi-valued
Real-valued Features
In the Bayesian approach, we use prior distributions for the two model parameters and compute their posteriors:

- $\theta_j$: Feature Likelihood
- $\pi_c$: Class Prior
Bayesian: Multivariate Bernoulli NB

- Categorical features: **Binary valued**
- Bayesian: Multivariate Bernoulli model parameter estimation.

\[
\bar{\pi}_c = \frac{\alpha_c + N_c}{\sum_{c=1}^{C} \alpha_c + N}
\]

\[\hat{\pi}_c = p(y = c) = \frac{N_c}{N}\]

\[
\bar{\theta}_{jc} = \frac{\beta_1 + N_{jc}}{\beta_0 + \beta_1 + N_c}
\]

\[\hat{\theta}_{jc} = p(x_j = 1 \mid y = c) = \frac{N_{jc}}{N_c}\]

\[= \frac{\text{Number of samples belonging to class } c \text{ that contain feature } j}{\text{Number of samples belonging to class } c}\]
Bayesian: Multivariate Bernoulli NB

- Often we just take $\alpha = 1$ and $\beta = 1$, corresponding to add-one or Laplace smoothing.

\[
\bar{\pi}_c = \frac{\alpha_c + N_c}{\sum_{c=1}^{C} \alpha_c + N} \\
\bar{\pi}_c = \frac{1 + N_c}{\text{#class} + N} = \frac{\text{Number of samples belonging to class c}}{\text{Total number of samples}}
\]

\[
\bar{\theta}_{jc} = \frac{\beta_1 + N_{jc}}{\beta_0 + \beta_1 + N_c} \\
\bar{\theta}_{jc} = \frac{1 + N_{jc}}{2 + N_c} = \frac{\text{Number of samples belonging to class c that contain feature j}}{\text{Number of samples belonging to class c}}
\]
Bayesian: Multinomial NB

- Categorical features: **Multi-valued**
- Bayesian: Multinomial model parameter estimation.

\[ \bar{\pi}_c = \frac{\alpha_c + N_c}{\sum_{c=1}^{C} \alpha_c + N} \]

\[ \hat{\pi}_c = p(y = c) = \frac{N_c}{N} \]

\[ \bar{\theta}_{jc} = \frac{\alpha_j + M_{jc}}{\sum_{j=1}^{d} \alpha_j + M_c} \]

\[ \hat{\theta}_{jc} = p(x_j | y = c) = \frac{M_{jc}}{M_c} \]

\[ = \frac{\text{Number of times feature } j \text{ occurs in the documents belonging to class } c}{\text{Number of terms in the documents belonging to class } c} \]
Bayesian: Multinomial NB

- Often we just take $\alpha = 1$, corresponding to add-one or Laplace smoothing.

$$\pi_c = \frac{\alpha_c + N_c}{\sum_{c=1}^C \alpha_c + N}$$

$$\bar{\pi}_c = \frac{1 + N_c}{\text{#class} + N}$$

$$= \frac{\text{Number of samples belonging to class } c}{\text{Total number of samples}}$$

$$\bar{\theta}_{jc} = \frac{\alpha_j + M_{jc}}{\sum_{j=1}^d \alpha_j + M_c}$$

$$\bar{\theta}_{jc} = \frac{1 + M_{jc}}{|\text{Vocabulary}| + M_c}$$

$$= \frac{\text{Number of times feature } j \text{ occurs in the documents belonging to class } c}{\text{Number of terms in the documents belonging to class } c}$$
GNB Classifier: Model Fitting

Using MLE we learn the GNB parameters:

\[
p(x_{ij} \mid y_i = c, \mu_{jc}, \sigma_{jc}^2) = \frac{1}{\sqrt{2\pi\sigma_{jc}^2}} e^{-\frac{1}{2\sigma_{jc}^2}(x_{ij} - \mu_{jc})^2}
\]

\[
\hat{\mu}_{jc} = \frac{1}{\sum_{i=1}^{N} \mathbb{I}(y_i = c)} \sum_{i=1}^{N} x_{ij} \mathbb{I}(y_i = c)
\]

\[
\sum_{i=1}^{N} \mathbb{I}(y_i = c) : \text{Number of documents belonging to class } c
\]

\[
x_{ij} \mathbb{I}(y_i = c) : \text{Value of feature } x_{ij} \text{ belonging to class } c \text{ in sample } i
\]

\[
\hat{\sigma}_{jc}^2 = \frac{1}{\sum_{i=1}^{N} \mathbb{I}(y_i = c)} \sum_{i=1}^{N} (x_{ij} - \hat{\mu}_{jc})^2 \mathbb{I}(y_i = c)
\]
GNB Classifier: Model Fitting

- Using **MLE** we learn the class **prior** as before:

\[
\hat{\pi}_c = p(y_i = c) = \frac{N_c}{N} \]

\[
p(x_{ij} \mid y_i = c, \mu_{jc}, \sigma_{jc}^2) = \frac{1}{\sqrt{2\pi\sigma_{jc}^2}} e^{-\frac{1}{2\sigma_{jc}^2}(x_{ij} - \mu_{jc})^2}
\]
Feature Extraction
Naïve Bayes Classifier: Feature Extraction

• Before we apply the NB classifier, we need to extract the features.
• Feature extraction consists in transforming arbitrary data, such as text or images, into numerical features usable for machine learning.
Naïve Bayes Classifier: Feature Extraction

• Let’s discuss the **steps of feature extraction** for a **text classification** problem.

• This can be generalized to feature extraction for **image classification** problem.
Naïve Bayes Classifier: Feature Extraction

• Generally there are **three steps** in feature extraction.
  - Text Normalization (Stemming & Lemmatization)
  - Text Preprocessing (Tokenization, removing stop words, etc.)
  - Vectorization of the features
Naïve Bayes Classifier: Feature Extraction

• Text normalization (stemming, lemmatization)
• Before we do text preprocessing (e.g., tokenize, remove stop words, etc.) and convert to vectors of numbers, sometimes it is useful to **normalize the text**.
• Following example shows how the text normalization works:
• Original text: *The boy’s cars are different colors*
• Normalized text: *the boy car be differ color*
Naïve Bayes Classifier: Feature Extraction

• Following **two techniques** are used for text normalization:
  - Stemming
  - Lemmatization

• Stemming and Lemmatization both **generate the root words** (e.g., play) form of the derived or inflected words (e.g., played).

• The difference is that **stem might not be an actual word** whereas, lemma is an actual language word.

• For more detail, see the jupyter notebook “**Feature Extraction for Text Classification**”:
  
  https://github.com/rhasanbd/Naive-Bayes-Algorithms-Foray-Into-Text-Classification/blob/master/Feature%20Extraction%20for%20Text%20Classification.ipynb
Naïve Bayes Classifier: Feature Extraction

- **Feature Vectorization**
  - We have discussed **two ways** to create word count vectors.
    - Binary count (whether a term occurs in a document)
    - Counting term frequency (number of times a term occurs in a document)
  - For some scenarios neither of these vectorization techniques are appropriate.
Naïve Bayes Classifier: Feature Extraction

- For example, in a large text corpus, some words will be very present (e.g. “the”, “a”, “is” in English).
- These words carry very little meaningful information about the actual contents of the document.
Naïve Bayes Classifier: Feature Extraction

• If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms.

• In order to re-weight the feature counts into floating point values suitable for usage by a classifier, it is very common to use the TF-IDF representation.

• The TF-IDF stands for “term frequency inverse document frequency”.
Naïve Bayes Classifier: Feature Extraction

• The idea is to replace the word count vector with a new feature vector called the TF-IDF representation.
• We define this as follows.
• First, the term frequency is defined as a log-transform of the count:

\[ \text{tf}(x_{ij}) \triangleq \log(1 + x_{ij}) \]

This reduces the impact of words that occur many times within one document.

The difference between the tf of a word that appears many times in a document (e.g., the, a) and the word that appears few times will not be significant.
Naïve Bayes Classifier: Feature Extraction

• Second, the **inverse document frequency** is defined as:

\[
\text{idf}(j) \triangleq \log \frac{N}{1 + \sum_{i=1}^{N} \mathbb{I}(x_{ij} > 0)}
\]

- **N** is the **total number of documents**.
- The denominator counts how many documents **contain term j**.
- If a word appears in **all documents** (e.g., the, a), then its idf will be **small**.
Naïve Bayes Classifier: Feature Extraction

Then, **TF-IDF** is defined as:

\[
\text{tf}(x_{ij}) \triangleq \log(1 + x_{ij})
\]

\[
\text{idf}(j) \triangleq \log \frac{N}{1 + \sum_{i=1}^{N} \mathbb{I}(x_{ij} > 0)}
\]

This **reduces the impact of words** that occur many times (e.g., the, a) within one document.

If a word appears in **all documents** (e.g., the, a), then its idf will be **small**.

Then, TF-IDF is defined as:

\[
\text{tf-idf}(x_i) \triangleq [\text{tf}(x_{ij}) \times \text{idf}(j)]_{j=1}^{V}
\]
Naïve Bayes Classifier: Feature Extraction

- However, TF-IDF representation of features will **not necessarily improve** the performance of the classifier.
- Consider the calculation of IDF.
- It **reduces the weight** of the most frequent words.
- The assumption is that the most frequent words (e.g., the, a) will be **spread evenly in all classes**.

\[
idf(j) \triangleq \log \frac{N}{1 + \sum_{i=1}^{N} \mathbb{I}(x_{i,j} > 0)}
\]

If a word appears in **all documents** (e.g., the, a), then its idf will be **small**.
Naïve Bayes Classifier: Feature Extraction

• However, if a most frequent word is actually important (e.g., lottery) and if it occurs more often in one class (e.g., spam emails) than other classes, then reducing its weight will be detrimental to the performance of the classifier.
Naïve Bayes Classifier: Feature Extraction

• For text classification we have used the **bag of words** model, which is a **collection of unigrams**.

• There are **some limitations** of this model:
  - It cannot capture phrases and multi-word expressions.
  - It effectively disregards any word order dependence.
  - It doesn’t account for potential misspellings or word derivations.
Naïve Bayes Classifier: Feature Extraction

- A better and sophisticated model for feature representation is the n-grams model.
- Instead of building a simple collection of unigrams (n=1), one might prefer a collection of bigrams (n=2).
- In a bigram occurrences of pairs of consecutive words are counted.
Naïve Bayes Classifier: Feature Extraction

• The n-grams model provides a set of co-occurring words within a given window.
• When computing the n-grams we typically move n word forward.
• For example, consider the sentence: Life is beautiful.
• If n = 2 (bigrams), then the n-grams would be:
  - Life is
  - is beautiful

We might alternatively consider a collection of character n-grams, a representation resilient against misspellings and derivations.
Implementing Naïve Bayes Classifier: Practical Issues
Naïve Bayes Classifier

• There are **some issues** that we need to be aware of before implementing Naïve Bayes classifier models.
  - Numerical Stability
  - Redundant Features
  - Feature Selection
Numerical Stability
Naïve Bayes Classifier: Numerical Stability

• We need to consider the **numerical stability** of our implementation of the NB classifier.

• Consider the class posterior:

\[
p(y = c \mid \vec{x}) \propto p(y = c) \prod_{j=1}^{d} p(x_j \mid y = c)
\]

• The calculation of the likelihood of feature components involves **multiplying a lot of small numbers together**.

• This can lead to an **underflow of numerical precision**.
Naïve Bayes Classifier: Numerical Stability

• To prevent this, we compute log probability by taking the log of each side.

\[
p(y = c | \vec{x}) \propto p(y = c) \prod_{j=1}^{d} p(x_j | y = c)
\]

\[
=> \log p(y = c | \vec{x}) \propto \log \left[ p(y = c) \prod_{j=1}^{d} p(x_j | y = c) \right]
\]

\[
=> \log p(y = c | \vec{x}) \propto \log p(y = c) + \log \prod_{j=1}^{d} p(x_j | y = c)
\]

\[
=> \log p(y = c | \vec{x}) \propto \log p(y = c) + \sum_{j=1}^{d} \log p(x_j | y = c)
\]
Redundant Features
Naïve Bayes Classifier: Redundant Features

• The performance might degrade if the data contains **highly correlated features**.

• This happens because the highly correlated features are voted **for twice** in the model, over inflating their importance.
Naïve Bayes Classifier: Redundant Features

• We need to evaluate the correlation of attributes pairwise with each other using a correlation matrix and remove those features that are the most highly correlated.

• We **must test our problem** before and after such a change and stick with the form of the problem that leads to the better results.
Feature Selection
Naïve Bayes Classifier: Feature Selection

• Since a Naïve Bayes Classifier is fitting a joint distribution over potentially many features, it might suffer from overfitting.

• Also the complexity will increase for a large feature dimension.

• One common approach to tackling both of these problems is to perform feature selection.

• We remove “irrelevant” features that do not help much with the classification problem.
Naïve Bayes Classifier: Feature Selection

• The simplest approach to feature selection is to evaluate the relevance of each feature separately.

• Then take the top $K$, where $K$ is chosen based on some tradeoff between accuracy and complexity.

• This approach is known as variable ranking, filtering, or screening.
Naïve Bayes Classifier: Feature Selection

- One way to measure relevance is to use **mutual information** between feature $x_j$ and the class label $y = c$.
- The mutual information can be thought of as the reduction in entropy on the label distribution once we observe the value of feature $j$.
- See Murphy 3.5.4 for detail.
Naïve Bayes: Non-Linear Classifier
Naïve Bayes Classifier

- Naïve Bayes is a **non-linear** classifier.
- It can classify non-linear dataset by creating **non-linear decision boundary**.
- It doesn’t need to augment the dataset (i.e., creating high-dimensional features).
Naïve Bayes Classifier

- For an **empirical understanding** see my Github repository:
- [https://github.com/rhasanbd/Naive-Bayes-Non-Linear-Classifier](https://github.com/rhasanbd/Naive-Bayes-Non-Linear-Classifier)
The Simplicity & Power of the NB Algorithm in 5 Questions
Naïve Bayes Classifier

- **Question 1**: Is the “naïve” assumption about conditional independence a problem? Does it hurt the performance?
- Not always!
- In spite of their apparently over-simplified assumptions, naïve Bayes classifiers have *worked quite well* in many real-world situations, famously document classification and spam filtering.
- The effectiveness of the NB assumption depends on how we model the feature vector.
Naïve Bayes Classifier

• **Question 2:** How much data we need to train a NB classifier?

• Naive Bayes **does not need a lot of data** to perform well.

• It needs **enough data** to understand the probabilistic relationship of each attribute in isolation with the output variable.
Question 3: Why don’t NB classifiers need a lot of data?

Because the interactions between feature components are ignored in the model.

We do not need examples of these interactions and therefore generally less data than other algorithms, such as logistic regression.

Thus, NB classifier is less likely to overfit the training data with a smaller sample size.

Try Naive Bayes if training data is scarce!
Naïve Bayes Classifier

- **Question 4:** What is the complexity of the NB classifier in *high-dimensional* dataset?
- Naïve Bayes learners and classifiers can be extremely fast compared to more sophisticated methods.
- The *decoupling* of the class conditional feature distributions means that each distribution can be independently estimated as a *one dimensional distribution*.
- This in turn helps to alleviate problems stemming from the *curse of dimensionality*.
Naïve Bayes Classifier

• **Question 5:** How is the generative approach of the NB classifier useful?

• For determining the **class posterior**, we need to compute the class **prior** and feature **likelihood**.

\[
p(y = c \mid \mathbf{x}) \propto p(y = c) \prod_{j=1}^{d} p(x_j \mid y = c)
\]

• These probabilities are extremely powerful because we can **generate new (unseen) data** by using these probabilities:

\[
p(\mathbf{x}, y = c) = p(y = c)p(\mathbf{x} \mid y = c)
\]
Naïve Bayes Classifier

• $p(y = c \mid \mathbf{x}) \propto p(y = c)p(\mathbf{x} \mid y = c)$.

• Note that we are not only modeling the distribution of the output $p(y = c \mid \mathbf{x})$, we are also modeling the distribution of input $p(\mathbf{x} \mid y = c)$.

• Thus, we are not only able to make predictions about an unseen data, we can actually generate an unseen data.
Naïve Bayes Classifier

• We can generate instances of the problem.
• For example, in text classification problem, the NB model can be used to create fictitious input documents.
• So, it can help to provide context for what the model has characterized.
Naïve Bayes Classifier: Applications
Naïve Bayes Classifier: Text Classification

• We have used the spam filter application as a motivating scenario for NB classifier.
• Classifying emails is one example of a broader set of problems called text classification.
• There are various forms of this application.
Naïve Bayes Classifier: Text Classification

- Despite the simplicity of Naïve Bayes, it can classify novel documents surprisingly well!

Idiot’s Bayes—Not So Stupid After All?

David J. Hand¹ and Keming Yu²

¹Department of Mathematics, Imperial College, London, UK. E-mail: d.j.hand@ic.ac.uk
²University of Plymouth, UK

Summary

Folklore has it that a very simple supervised classification rule, based on the typically false assumption that the predictor variables are independent, can be highly effective, and often more effective than sophisticated rules. We examine the evidence for this, both empirical, as observed in real data applications, and theoretical, summarising explanations for why this simple rule might be effective.
Naïve Bayes Classifier: Text Classification

- The Naïve Bayes assumption seems to work quite well!
- Intuitively a potential justification for the conditional independence assumption is:
- If we know that a document is about politics, this is a good indication of the kinds of other words we will find in the document.
Naïve Bayes Classifier: Text Classification

- There are various forms of the text classification application.
- For example: object classification.
Naïve Bayes Classifier: Computer Vision

• In **computer vision**, the “bag of words” model can be used for **object classification**.
• We represent an object as a **bag of visual words**.

Patches that are described by a **certain descriptor**.
Naïve Bayes Classifier: Computer Vision

• We can use the bag of words model for **object categorization** by constructing a **large vocabulary of many visual words**.

Then, **represent each image as a histogram** of the frequency words that are in the image.
Naïve Bayes Classifier: Computer Vision

• How exactly do we construct the model?
• First, we need to **build a visual dictionary**.
• We do that by taking a large set of object images and **extracting descriptors** from them.
• Next, we **cluster the set of descriptors** (using k-means for example) to k clusters.
• The **cluster centers** act as our dictionary’s visual words.
Naïve Bayes Classifier: Computer Vision

• Given a **new image**, we represent it using our model in the following manner:

• First, **extract descriptors from the image** on a grid or around detected keypoints.

• Next, for each extracted descriptor, **compute its nearest neighbor** in the dictionary.
Naïve Bayes Classifier: Computer Vision

• Finally, build a histogram of length $k$ where the $i$’th value is the frequency of the $i$’th dictionary word:
Naïve Bayes Classifier: Computer Vision

• This model can be used in conjunction with Naïve Bayes classifier or with a support vector machine for object classification.
In Praise of the Naïve Bayes Classifier
In Praise of Naïve Bayes Classifier

• Naïve Bayes requires **minimal storage and has fast training**.
• That’s why it has been applied to **time and storage critical applications**, such as automatically classifying webpages into types, and spam filtering.

---

**An Experimental Comparison of Naive Bayesian and Keyword-Based Anti-Spam Filtering with Personal E-mail Messages**

Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos and Constantine D. Spyropoulos  
Software and Knowledge Engineering Laboratory  
Institute of Informatics and Telecommunications  
National Centre for Scientific Research “Demokritos”  
153 10 Ag. Paraskevi, Athens, Greece  
e-mail: {ionandr, jkoutsi, kostel, costass}@iit.demokritos.gr

---

**Classification of Web Documents Using a Naive Bayes Method**

Yong Wang, Julia Hodges, Bo Tang  
*Department of Computer Science & Engineering, Mississippi State University*  
Mississippi State, MS 39762-9637  
ywang@cse.msstate.edu, hodges@cse.msstate.edu, btang@cse.msstate.edu
In Praise of Naïve Bayes Classifier

- While not necessarily the very best classification algorithm, the Naïve Bayes classifier often works surprisingly well.
- NB often performs well, even when assumption is violated!
- It is often also a very good “first thing to try,” given its simplicity and ease of implementation.

On the Optimality of the Simple Bayesian Classifier under Zero-One Loss

PEDRO DOMINGOS  
MICHAEL PAZZANI

Department of Information and Computer Science, University of California, Irvine, CA 92697