CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

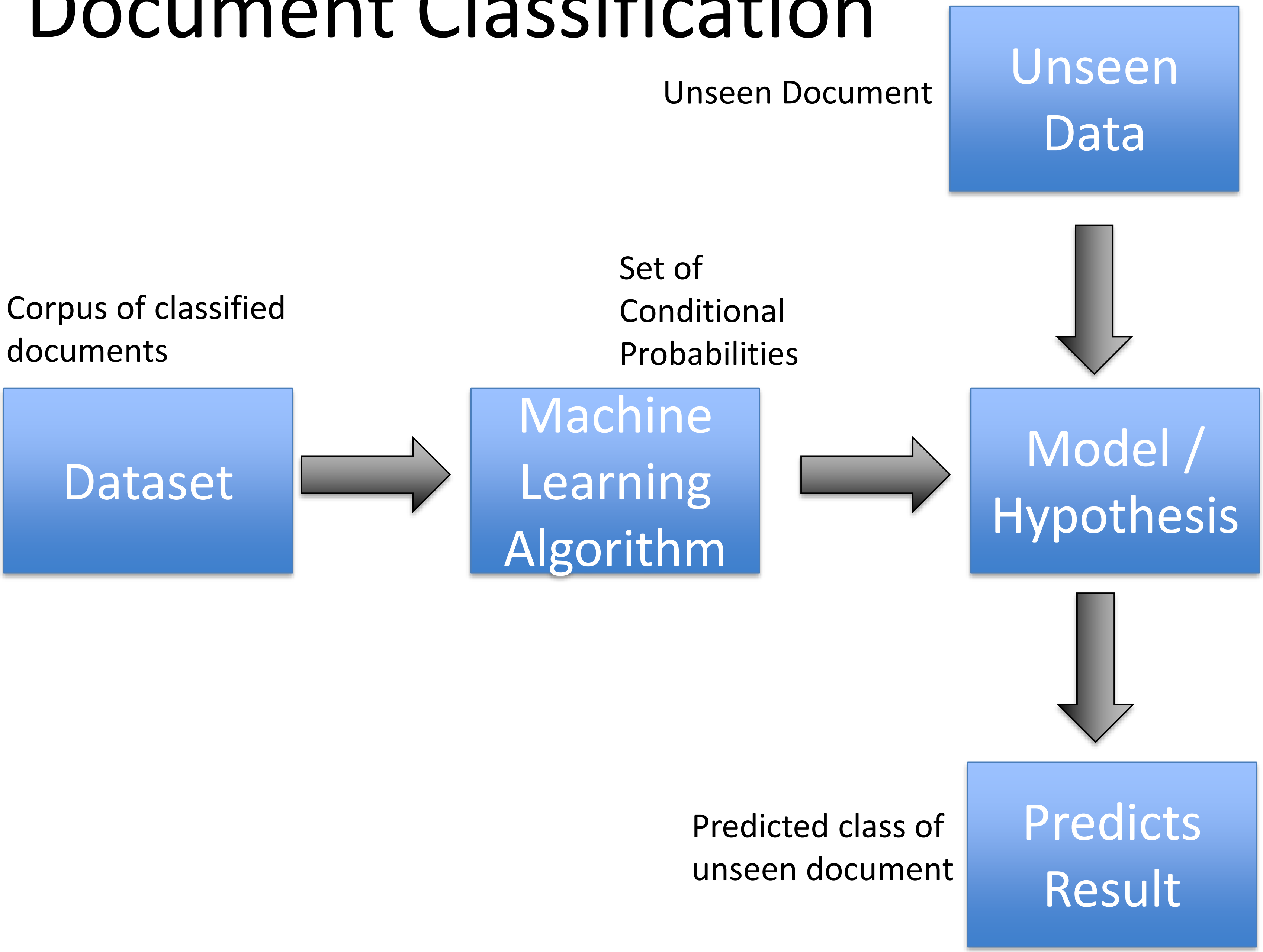
# Machine Learning

**Machine Learning**

Lecture: Bayesian Classification

Ted Scully

# Document Classification



# Calculating Prior Probabilities

- A Bayesian classifier will typically either adopt a **bag** of words or **set** of words approach.
  - (<u>Multinomial Model</u>) **Bag of words**, counts the total occurrences of a word across all documents.
  - (<u>Bernoulli model</u>) **Set of words**, counts the number of documents where a word occurs

$$argmax_{c \in C} \log P(c) + \sum_{w \in W} \log P(w \mid c)$$

---

- The first thing we need to do is calculate the prior probabilities (that is, the probability of the class). This calculation is the same for both multinomial and binomial.

$$P(c) = \frac{\text{Number of documents of class c}}{Total\ Number\ of\ documents}$$

# Naïve Bayes - <u>Multinomial</u> Model

$$c_{MAP} = argmax_{c \in C} \log P(c) + \sum_{w \in W} \log P(w \mid c)$$

- Calculation of the probabilities in the multinomial model are as follows (notice we use <u>laplace smoothing</u> here):

  - $P(w \mid c) = \frac{count(w,c)+1}{count(c)+|V|}$

***count(w, c)*** is the number of <u>occurrences of the word w</u> in all documents of class c.

***count(c)*** The total <u>number of words</u> in all documents of class c (**including duplicates**).

***|V|*** The number of words in the vocabulary, which is all unique words irrespective of class.

| | Doc | Words | Class |
|---|---|---|---|
| Training | 1 | Cloud Java Cloud | Comp |
| | 2 | Cloud Cloud Spring | Comp |
| | 3 | Cloud Software | Comp |
| | 4 | Referendum Software Election | Politics |
| Test | 5 | Java Software Java Election | ? |

$$c_{MAP} = argmax_{c \in C} \log P(c) + \sum_{w \in W} \log P(w \mid c)$$

$$P(Comp) = \frac{3}{4}$$

$$P(Politics) = \frac{1}{4}$$

| | Doc | Words | Class |
|---|---|---|---|
| Training | 1 | Cloud Java Cloud | Comp |
| | 2 | Cloud Cloud Spring | Comp |
| | 3 | Cloud Software Java | Comp |
| | 4 | Referendum Software Election | Politics |
| Test | 5 | Java Software Java Election | ? |

$$\boxed{P(w \mid c) = \frac{count(w,c) + 1}{count(c) + |V|}}$$

Notice we use Laplace smoothing here

$$P(Cloud \mid Comp) = \frac{5+1}{9+6}$$

$$P(Java \mid Comp) = \frac{2+1}{9+6}$$

$$P(Software \mid Comp) = \frac{1+1}{9+6}$$

$$P(Spring \mid Comp) = \frac{1+1}{9+6}$$

$$P(Referendum \mid Comp) = \frac{0+1}{9+6}$$

$$P(Election \mid Comp) = \frac{0+1}{9+6}$$

| | Doc | Words | Class |
|---|---|---|---|
| Training | 1 | Cloud Java Cloud | Comp |
| | 2 | Cloud Cloud Spring | Comp |
| | 3 | Cloud Software Java | Comp |
| | 4 | Referendum Software Election | Politics |
| Test | 5 | Java Software Java Election | ? |

$$P(w \mid c) = \frac{count(w,c) + 1}{count(c) + |V|}$$

Notice we use Laplace smoothing here

$$P(Cloud \mid Politics) = \frac{0+1}{3+6}$$

$$P(Java \mid Politics) = \frac{0+1}{3+6}$$

$$P(Software \mid Politics) = \frac{1+1}{3+6}$$

$$P(Spring \mid Politics) = \frac{0+1}{3+6}$$

$$P(Referendum \mid Politics) = \frac{1+1}{3+6}$$

$$P(Election \mid Politics) = \frac{1+1}{3+6}$$

| | Doc | Words | Class |
|---|---|---|---|
| Test | 5 | Java Software Java Election | ? |

$$P(Cloud \mid Comp) = \frac{6}{15}$$

$$P(Java \mid Comp) = \frac{3}{15}$$

$$P(Software \mid Comp) = \frac{2}{15}$$

$$P(Spring \mid Comp) = \frac{2}{15}$$

$$P(Election \mid Comp) = \frac{1}{15}$$

$$P(Referendum \mid Comp) = \frac{1}{15}$$

$$P(Cloud \mid Politics) = \frac{1}{9}$$

$$P(Java \mid Politics) = \frac{1}{9}$$

$$P(Software \mid Politics) = \frac{2}{9}$$

$$P(Spring \mid Politics) = \frac{1}{9}$$

$$P(Election \mid Politics) = \frac{2}{9}$$

$$P(Referendum \mid Politics) = \frac{2}{9}$$

$$P(Comp) = \frac{3}{4} \qquad P(Politics) = \frac{1}{4}$$

| | Doc | Words | Class |
|---|---|---|---|
| Test | 5 | Java Software Java Election | ? |

$$P(c \mid W) = \log P(c) + \sum_{w \in W} \log P(w \mid c)$$

$P(Comp \mid Test) =$ log(3/4) + log(3/15) + log(2/15) + log (3/15) + log (1/15) = **-3.57**

$P(Politics \mid Test) =$ log(1/4) + log(1/9) + log(2/9)+log(1/9)+log(2/9)= **-3.81**

**Classify the document as being of class Comp**

# Naïve Bayes: Text Classification for Multinomial

*Examples* are a set of training documents.

*V* is the set of classes (ex. Spam / NotSpam)

Learn_naive_Bayes_text($Examples$, $V$)

1. collect all words that occur in $Examples$
   $Vocabulary \leftarrow$ all distinct words in $Examples$
2. calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms
   For each target value $v_j$ in $V$ do
   - $docs_j \leftarrow$ subset of $Examples$ for which the target value is $v_j$
   - $P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$
   - $Text_j \leftarrow$ a single document created by concatenating all members of $docs_j$
   - $n \leftarrow$ total number of words in $Text_j$ (counting duplicate words multiple times)
   - for each word $w_k$ in $Vocabulary$
     - $n_k \leftarrow$ number of times word $w_k$ occurs in $Text_j$
     - $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$

# Document Classification

- Classify_naive_Bayes_text(newDoc)
  - We take in an unseen document *newDoc*, we extract all words from the document and store in *allWords* (the same word may appear multiple time)

$$\underset{v_j \in V}{\operatorname{arg}max} \left( logP(v_j) + \sum_{x \in allWords} \log P(x \mid v_j) \right)$$

# Naïve Bayes - Bernoulli Model

$$c_{MAP} = argmax_{c \in C} \log P(c) + \sum_{w \in W} \log P(w \mid c)$$

- Calculation of the probabilities in the Bernoulli model as are follows (notice we use plus one smoothing here):

- $$P(w \mid c) = \frac{countDocs(w,c)+1}{countDocs(c)+2}$$

***countDocs(w, c)*** is the number of documents of class c where the word w occurs.

***countDocs(c)*** The total number of documents of class c.

| | Doc | Words | Class |
|---|---|---|---|
| Training | 1 | Cloud Java Cloud | Comp |
| | 2 | Cloud Cloud Spring | Comp |
| | 3 | Cloud Software Java | Comp |
| | 4 | Referendum Software Election | Politics |
| Test | 5 | Java Software Java Election | ? |

Notice we use +1 smoothing here

$$P(Cloud \mid Comp) = \frac{3+1}{3+2}$$

$$P(Java \mid Comp) = \frac{2+1}{3+2}$$

$$P(Software \mid Comp) = \frac{1+1}{3+2}$$

$$P(Spring \mid Comp) = \frac{1+1}{3+2}$$

$$P(Referendum \mid Comp) = \frac{0+1}{3+2}$$

$$P(Election \mid Comp) = \frac{0+1}{3+2}$$

| | Doc | Words | Class |
|---|---|---|---|
| Training | 1 | Cloud Java Cloud | Comp |
| | 2 | Cloud Cloud Spring | Comp |
| | 3 | Cloud Software Java | Comp |
| | 4 | Referendum Software Election | Politics |
| Test | 5 | Java Software Java Election | ? |

Notice we use +1 smoothing here

$$P(Cloud \mid Politics) = \frac{0+1}{1+2}$$

$$P(Java \mid Politics) = \frac{0+1}{1+2}$$

$$P(Software \mid Politics) = \frac{1+1}{1+2}$$

$$P(Spring \mid Politics) = \frac{0+1}{1+2}$$

$$P(Referendum \mid Politics) = \frac{1+1}{1+2}$$

$$P(Election \mid Politics) = \frac{1+1}{1+2}$$

| | Doc | Words | Class |
|---|---|---|---|
| Test | 5 | Java Software Java Election | ? |

$$P(Cloud \mid Comp) = \frac{4}{5}$$

$$P(Java \mid Comp) = \frac{3}{5}$$

$$P(Software \mid Comp) = \frac{2}{5}$$

$$P(Spring \mid Comp) = \frac{2}{5}$$

$$P(Election \mid Comp) = \frac{1}{5}$$

$$P(Referendum \mid Comp) = \frac{1}{5}$$

$$P(Cloud \mid Politics) = \frac{1}{3}$$

$$P(Java \mid Politics) = \frac{1}{3}$$

$$P(Software \mid Politics) = \frac{2}{3}$$

$$P(Spring \mid Politics) = \frac{1}{3}$$

$$P(Election \mid Politics) = \frac{2}{3}$$

$$P(Referendum \mid Politics) = \frac{2}{3}$$

| | Doc | Words | Class |
|---|---|---|---|
| Test | 5 | Java Software Java Election | ? |

$$P(c \mid W) = \log P(c) + \sum_{w \in W} \log P(w \mid c)$$

When classifying a new document in Bernoulli we go through every **word in the vocabulary** and we incorporate the probability of the word **occurring** and the word **not occurring** given the class.

The probability of a word occurring given the class is $\boldsymbol{P(w \mid c)}$. Note that the probability of a word w not occurring given the class c is $\boldsymbol{1 - P(w \mid c)}$

| | Doc | Words | Class |
|---|---|---|---|
| Test | 5 | Java Software Java Election | ? |

Cloud
Java
Software
Spring
Election
Referendum

$$P(c \mid W) = \log P(c) + \sum_{w \in W} \log P(w \mid c)$$

$$\begin{aligned} &P(Comp \mid Test) \\ &= \log\big(\mathrm{P}(Comp)\big) + \log\big(1 - \mathrm{P}(\mathrm{Cloud}|\mathrm{Comp})\big) + \log\big(\mathrm{P}(\mathrm{Java}|\mathrm{Comp})\big) \\ &+ \log\big(\mathrm{P}(\mathrm{Software}|\mathrm{Comp})\big) + \log\big(1 - \mathrm{P}(\mathrm{Spring}|\mathrm{Comp})\big) \\ &+ \log\big(\mathrm{P}(\mathrm{Election}|\mathrm{Comp})\big) + \log(1 - \mathrm{P}(\mathrm{Referendum}|\mathrm{Comp})) \end{aligned}$$

$$\begin{aligned} &P(Politics \mid Test) \\ &= \log\big(\mathrm{P}(Politics)\big) + \log\big(1 - \mathrm{P}(\mathrm{Cloud}|\mathrm{Politics})\big) + \log\big(\mathrm{P}(\mathrm{Java}|\mathrm{Politics})\big) \\ &+ \log\big(\mathrm{P}(\mathrm{Software}|\mathrm{Politics})\big) + \log\big(1 - \mathrm{P}(\mathrm{Spring}|\mathrm{Politics})\big) \\ &+ \log\big(\mathrm{P}(\mathrm{Election}|\mathrm{Politics})\big) + \log(1 - \mathrm{P}(\mathrm{Referendum}|\mathrm{Politics})) \end{aligned}$$

| | Doc | Words | Class |
|---|---|---|---|
| Test | 5 | Java Software Java Election | ? |

$$P(c \mid W) = \log P(c) + \sum_{w \in W} \log P(w \mid c)$$

Cloud
Java
Software
Spring
Election
Referendum

$P(Comp \mid Test)$ = log(3/4) + log(1-(4/5))+log(3/5)+log(2/5)+log(1-(2/5))+log(1/5)+log(1-(1/5)) = -2.46

$P(Politics \mid Test)$ = log(1/4) + log(1-(1/3))+log(1/3)+log(2/3)+log(1-(1/3))+log(2/3)+log(1-(2/3)) = -2.26

**Classify the document as being of class Politics**

# Pre-processing for Document Classification using Naïve Bayes

- Quite often a range of pre-processing activities can be used to clean the data prior to it's usage by Naïve Bayes.

- These pre-processing steps can include very basic steps such as removal of **punctuation, URLs** and **lower-casing** all words. The objective of many of these techniques is reducing the number of features (words) in the dataset.

- However, there is a host of more advanced techniques that we can also apply and may improve classification accuracy. Many of these techniques are available in Python's NLTK.

# Stemming and Lemmatization

- Stemming and lemmatization attempt to truncate words to their stem or root word.
  - A stemmer for English, for example, should identify the string "fishing", "fished", and "fisher" to the root word, "fish“, and "stemmer", "stemming", "stemmed" as based on "stem".
  - Porter’s Stemming Algorithm (There are stemmers available from the natural language toolkit in Python)
  - Typically a stemming algorithm will truncate existing words to form the root.
  - In contrast lemmatization attempts to do this by using a vocabulary.

| | |
|---|---|
| cats -- cat | cats -- cat |
| cacti -- cacti | cacti -- cactus |
| geese -- gees | geese -- goose |
| rocks -- rock | rocks -- rock |
| python -- python | python -- python |
| wolves -- wolv | wolves -- wolf |

# Emoticons, Stop-Words, Misspelled Words

- It is important in the process of sentiment analysis to identify the graphical cues for sentiment as represented by **emoticons**
  - One common approach is to use a dictionary that has emoticons labelled according to their emotional state.
  - For example, “:)” is labelled as positive whereas “:-(” is labelled as negative. Commonly each emoticon is given one of the following labels
    - Extremely-positive, Extremely-negative, Positive, Negative, Neural

- Another common parsing techniques is the **removal of stop-words**. There are freely available dictionaries of stopwords (http://xpo6.com/list-of-english-stop-words/ ) NLTK also provides a stopword dictionary.

- Detection and correction of **misspelled words** using a dictionary (using tool such as PyEnchant).

# N Grams

- In  the n-gram model, a token can be defined as a sequence of n items.
- The simplest case is the so-called **unigram (1-gram)** where each **token** consists of exactly **one word**.
- Everything that we have looked at so far has been uni-gram.
- In a bi-gram (2-gram) each token consists of **two adjacent works**, in a tri-gram (3-gram) it consists of **three adjacent words**.
- N-grams can often have a positive impact on accuracy but also significant increase the number of features (hence the size of your vocabulary)

| | | | | | | |
|---|---|---|---|---|---|---|
| Uni-gram | **The** | **new** | **starwars** | **film** | **got** | **great** |
| Bi-gram | **The new** | **new starwars** | **starwars film** | **....** | | |
| Tri-gram | **The new starwars** | **new starwars film** | **starwars film got** | **....** | | |

# Dealing with Continuous Features

- So far we dealt only with categorical features and to calculate the probability of an event, we have just counted how often the event occurred and divided this by how often the event could have occurred.
- Clearly adopting the above approach is not practical for a continuous features because it can have an **infinite number of values in it's domain.**
- One common approach to dealing with this issue is binning.

How would NB work if Temp was a continuous valued feature?

**Anyone for Tennis?**

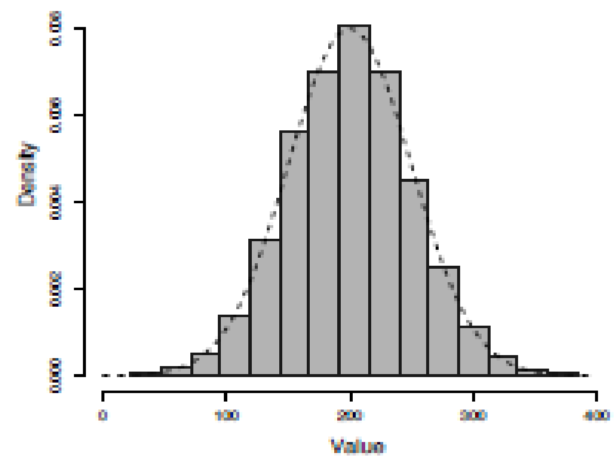
| ID | Outlook | Temp | Humidity | Windy | Play? |
|---|---|---|---|---|---|
| A | sunny | hot | high | false | no |
| B | sunny | hot | high | true | no |
| C | overcast | hot | high | false | yes |
| D | rainy | mild | high | false | yes |
| E | rainy | cool | normal | false | yes |
| F | rainy | cool | normal | true | no |
| G | overcast | cool | normal | true | yes |
| H | sunny | mild | high | false | no |
| I | sunny | cool | normal | false | yes |
| J | rainy | mild | normal | false | yes |
| K | sunny | mild | normal | true | yes |
| L | overcast | mild | high | true | yes |
| M | overcast | hot | normal | false | yes |
| N | rainy | mild | high | true | no |

# Binning Continuous Features

- An approach to dealing with continuous features is to convert them into categorical variables using binning.

- To perform binning, we define a series of ranges (called bins) for the continuous feature that correspond to the levels of the new categorical feature we are creating.

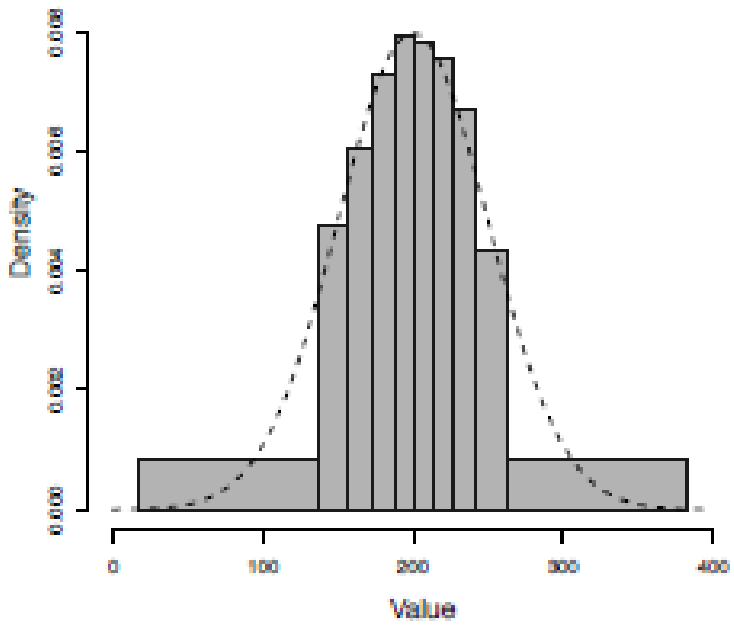
- **Equal-width binning** - The equal-width binning algorithm splits the range of the feature values into b bins each of size range/b.



# Binning Continuous Variables

- **Equal-frequency binning** first sorts the continuous feature values into ascending order and then places an equal number of instances into each bin, starting with bin 1.

- The number of instances placed in each bin is simply the total number of instances divided by the number of bins, b..



# Strengths of Naïve Bayes

- Training Set Size and Speed
  - Naïve Bayes is a **very fast algorithm**
    - The process of calculating the probabilities is the only potentially time consuming component.
  - Another advantage of Naïve Bayes is that it is a probabilistic classifier so it provides **some degree of certainty** in it's conclusions.
    - For example, we may only wish to classify the polarity of a tweet if we are more than 75% confident that the tweet is positive or negative.

| | | **Confidence Prediction** | | |
|---|---|---|---|---|
| | | **50%** | **70%** | **90%** |
| **Baseline NB** | % Accuracy | 76.5 | 84.2 | 90.2 |
| | % Predicted | 100 | 73.2 | 43.5 |

# Strengths of Naïve Bayes

- Naïve Bayes is **less sensitive to irrelevant features**...
  - Suppose we are trying to classify a persons gender based on several features, including eye colour (Of course, eye colour is completely irrelevant to a persons gender)
  - How would Naïve Bayes deal with such an irrelevant attribute.

p(eye = brown | female) * p(long_hair= yes | female) * .....

p(eye = brown | male) * p(long_hair = yes | male) * .....

p(eye = brown | female) * p(long_hair = yes | female) * .....
=> 5000/10000 * 9,500/10000

p(eye = brown | male) * p(long_hair = yes | male) * .....
=> 5000/10000 * 500/10000

# Weakness of Naïve Bayes

- Naïve Bayes is primarily a classification algorithm. While studies have adapted NB as for <u>**regression**</u> problems it's performance on such problems has been generally poor.

- The "Naive" term comes from the fact that the model **assumes that all features are fully independent** given the class, which in real problems they almost never are.

- In practice this approach still works reasonably well for many real-world problems.
- However, we can adopt a more realistic approach that will incorporate <u>certain</u> dependencies amongst the variables in our domain using <u>Bayesian Networks</u>.