### Probabilistic Model of Data

- Assume that the labeled examples \((x_1,t_1), (x_2,t_2), \ldots, (x_N,t_N)\) are generated independently from some unknown but fixed distribution.
- Captures distribution over features in \(x_i\).
- Captures distribution over labels in \(t_i\).
- Captures their correlation/dependence.

\[
p(t_{\text{new}} | x_{\text{new}}) = \frac{p(x_{\text{new}} | t_{\text{new}}) p(t_{\text{new}})}{p(x_{\text{new}})}
\]

### Is it Gold?

- Testing marbles during the gold rush ...
  - \(\oplus\) means Gold and \(\ominus\) means not Gold
  - \(p(\oplus) = 0.01\) \(p(\ominus) = 0.99\)
  - Prof $$ developed a gold detector
    - \(p(D = \text{yes} | \oplus) = 0.98\) \(p(D = \text{yes} | \ominus) = 0.04\)
- A marble is evaluated and tests yes.
- Should we buy it?

### The Bayesian Classifier

- In order to apply this scheme we need to know the quantities \(p(t)\) and \(p(x | t)\).
- For discrete classes \(p(t)\) is easy to tabulate.
  - Typically \(x\) is high dimensional.
  - How can we represent \(p(x | t)\)?
  - When \(x\) is continuous? discrete?

### Naïve Bayes Model

Features are conditionally independent given the label.

- \(x\) has \(k\) dimensions \(p(x | t) = \prod_{j=1}^{k} p(x_j | t)\)
- Implying the following classification rule
  \[
p(t_{\text{new}} = \oplus | x_{\text{new}}) \propto p(t_{\text{new}} = \oplus) \prod_j p(x_{\text{new},j} | t_{\text{new}} = \oplus)
p(t_{\text{new}} = \ominus | x_{\text{new}}) \propto p(t_{\text{new}} = \ominus) \prod_j p(x_{\text{new},j} | t_{\text{new}} = \ominus)
\]

These are not probabilities but we can still pick the maximizing values.
Naive Bayes Prediction: Example

\[ p(\oplus) = 0.4 \quad p(\ominus) = 0.6 \]

3 Binary features

\[ p(x_1 = \text{yes} | \oplus) = 0.2 \quad p(x_1 = \text{yes} | \ominus) = 0.4 \]
\[ p(x_2 = \text{yes} | \oplus) = 0.6 \quad p(x_2 = \text{yes} | \ominus) = 0.6 \]
\[ p(x_3 = \text{yes} | \oplus) = 0.8 \quad p(x_3 = \text{yes} | \ominus) = 0.1 \]

Predict the label for \( x = 010 \):

\[ \oplus : 0.4 \cdot 0.8 \cdot 0.6 \cdot 0.2 = 0.0384 \]
\[ \ominus : 0.6 \cdot 0.6 \cdot 0.6 \cdot 0.9 = 0.1944 \]
\[ \Rightarrow \text{predict } \ominus \]

Naïve Bayes Model

- Same Equations hold for more than two discrete labels
- Can adapt for real valued attributes, for example, by using a univariate Gaussian distribution for each such feature
- How can we estimate \( p(t) \) and \( p(x_j | t) \)?
  - Use maximum likelihood estimate
  - And potentially add smoothing

Learning/Estimation

Maximum likelihood: Write likelihood, take derivatives and solve to get

\[ \hat{p}(t = c) = \frac{\text{number of examples with class } c}{\text{number of examples}} \]
\[ \hat{p}(x_j = a | t = c) = \frac{\text{num of ex with class } c \text{ and } x_j = a}{\text{number of examples with class } c} \]

Learning/Estimation

\[ \hat{p}(t = c) = \frac{\# [t = c]}{N} \]
\[ \hat{p}(x_j = a | t = c) = \frac{\# [t = c, x_j = a]}{\# [t = c]} \]

Naïve Bayes for Text Classification

- Document 1: \{what a nice day\}
  - Label: +
- Document 2: \{a green cat chased a green dog\}
  - Label: +
- Document 3: \{green umbrella a nice day\}
  - Label: -
- Document 4: \{a nice day\}
- Document 5: \{what a green umbrella\}

Naïve Bayes for Text Classification

- Classes: Yes, No
- Features: word-slot in document has value word-i in lexicon
- Features have a huge number of values
- Number of features is document length

\[ \hat{p}(x_j = \text{“halligan”} | t = c) = \frac{\# [t = c, x_j = \text{“halligan”}]}{\# [t = c]} \]

This is naïve. If every location index in document has different parameters then estimation is problematic.
Naïve Bayes for Text Classification

- Classes: Yes, No
- Features: word-slot in document has value word-i in lexicon
- Features have a huge number of values
- Number of features is document length

Practical model assumes that all positions behave the same so word counts are pooled across all positions

\[
\hat{p}(x_j = \text{“halligan”}|t = c) = \frac{\#\text{[slots with “halligan” when } t = c]}{\#\text{[slots when } t = c]}
\]

- What happens if “halligan” never appeared in class c in training data?

\[
\hat{p}(t_{\text{new}} = \oplus|x_{\text{new}}) \propto \hat{p}(t_{\text{new}} = \ominus) \prod_j \hat{p}(x_{\text{new},j}|t_{\text{new}} = \ominus)
\]

\[
\hat{p}(t_{\text{new}} = \ominus|x_{\text{new}}) \propto \hat{p}(t_{\text{new}} = \ominus) \prod_j \hat{p}(x_{\text{new},j}|t_{\text{new}} = \ominus)
\]

- Is this good? bad? Can it be fixed?

Learning/Estimation

Laplace smoothing (for standard case, not text application):

\[
\hat{p}(t = c) = \frac{\#[t = c]}{N}
\]

\[
\hat{p}(x_j = a|t = c) = \frac{\#[t = c, x_j = a] + 1}{\# [t = c] + V}
\]

V is the number of values that \( x_j \) takes

Laplace smoothing (for text application):

\[
\hat{p}(t = c) = \frac{\#[t = c]}{N}
\]

\[
\hat{p}(x_j = a|t = c) = \frac{\#([t = c, x_j = a] + p)}{\# [t = c] + pV}
\]

V is the number of values that \( x_j \) takes

We can also control the amount of smoothing

Text Classification

- Data pre-processing can significantly improve results
  - Remove “stop words”
  - Remove words with very small counts
  - Apply stemming
Text Classification

• With other algorithms e.g. kNN, DT
• Requires features

• Common approach is

• “Bag of Words”: feature representation indexed by word (not location)
  - Binary: “Halligan appeared in document”
  - or TFIDF weighting
    \[
    \text{TFIDF} = \frac{\text{#t at doc} \times \log \left( \frac{\text{#docs}}{\text{#docs that have t}} \right)}
    \]

Recap: Naïve Bayes Algorithm

• Very simple
• Efficient

• Quite successful in many cases (but often we can do better)
• So can serve as a baseline