CPSC 340: Machine Learning and Data Mining

Probabilistic Classification

Fall 2017
Admin

• Assignment 0 is due tonight: you should be almost done.
  – 1 late day to hand it in Monday, 2 late days for Wednesday.
• Assignment 1 is coming Monday: start early.

• Important webpages:
**Last Time: Training, Testing, and Validation**

- **Training step:**
  
  \[\text{Input: set of } n \text{ training examples } x_i \text{ with labels } y_i\]
  
  \[\text{Output: a model that maps from arbitrary } x_i \text{ to a } y_i\]

- **Prediction step:**

  \[\text{Input: set of } l \text{ testing examples } \tilde{x}_i \text{ and a model.}\]
  
  \[\text{Output: predictions } \hat{y}_i \text{ for the testing examples.}\]

- What we are interested in is the **test error:**
  
  – Error made by prediction step on new data.
Last Time: Fundamental Trade-Off

• We decomposed test error to get a fundamental trade-off:

\[ E_{\text{test}} = E_{\text{approx}} + E_{\text{train}} \]

  \(
  \text{“test error”}
  \quad \text{“approximation error”}
  \quad \text{“training error”}
  \)

  - Where \( E_{\text{approx}} = (E_{\text{test}} - E_{\text{train}}) \).

• \( E_{\text{train}} \) goes down as model gets complicated:
  - Training error goes down as a decision tree gets deeper.

• But \( E_{\text{approx}} \) goes up as model gets complicated:
  - Training error becomes a worse approximation of test error.
• **Golden rule**: we can’t look at test data during training.
• But we can approximate $E_{test}$ with a **validation error**:
  – Error on a set of training examples we “hid” during training.

\[
X = \begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix} \quad Y = \begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\]

  “train”

  “validation”

  – Find the decision tree based on the “train” rows.
  – Validation error is the error of the decision tree on the “validation” rows.
Should you trust them?

• Scenario 1:
  – “I built a model based on the data you gave me.”
  – “It classified your data with 98% accuracy.”
  – “It should get 98% accuracy on the rest of your data.”

• Probably not:
  – They are reporting training error.
  – This might have nothing to do with test error.
  – E.g., they could have fit a very deep decision tree.

• Why ‘probably’?
  – If they only tried a few very simple models, the 98% might be reliable.
  – E.g., they only considered decision stumps with simple 1-variable rules.
Should you trust them?

• Scenario 2:
  – “I built a model based on half of the data you gave me.”
  – “It classified the other half of the data with 98% accuracy.”
  – “It should get 98% accuracy on the rest of your data.”

• Probably:
  – They computed the validation error once.
  – This is an unbiased approximation of the test error.
  – Trust them if you believe they didn’t violate the golden rule.
Should you trust them?

• Scenario 3:
  – “I built 10 models based on half of the data you gave me.”
  – “One of them classified the other half of the data with 98% accuracy.”
  – “It should get 98% accuracy on the rest of your data.”

• Probably:
  – They computed the validation error a small number of times.
  – Maximizing over these errors is a biased approximation of test error.
  – But they only maximized it over 10 models, so bias is probably small.
  – They probably know about the golden rule.
Should you trust them?

• Scenario 4:
  – “I built 1 billion models based on half of the data you gave me.”
  – “One of them classified the other half of the data with 98% accuracy.”
  – “It should get 98% accuracy on the rest of your data.”

• Probably not:
  – They computed the validation error a huge number of times.
  – Maximizing over these errors is a biased approximation of test error.
  – They tried so many models, one of them is likely to work by chance.

• Why ‘probably’?
  – If the 1 billion models were all extremely-simple, 98% might be reliable.
Should you trust them?

• Scenario 5:
  – “I built 1 billion models based on the first third of the data you gave me.”
  – “One of them classified the second third of the data with 98% accuracy.”
  – “It also classified the last third of the data with 98% accuracy.”
  – “It should get 98% accuracy on the rest of your data.”

• Probably:
  – They computed the first validation error a huge number of times.
  – But they had a second validation set that they only looked at once.
  – The second validation set gives unbiased test error approximation.
  – This is ideal, as long as they didn’t violate golden rule on the last third.
  – And assuming you are using IID data in the first place.
Validation Error and Optimization Bias

- **Optimization bias** is *small if you only compare a few models*:
  - Best decision tree on the training set among depths, 1, 2, 3,..., 10.
  - Risk of overfitting to validation set is low if we try 10 things.

- **Optimization bias** is *large if you compare a lot of models*:
  - All possible decision trees of depth 10 or less.
  - Here we’re using the validation set to pick between a billion+ models:
    - Risk of overfitting to validation set is high: could have low validation error by chance.
  - If you did this, you might want a second validation set to detect overfitting.
Cross-Validation (CV)

• Isn’t it wasteful to only use part of your data?
• 5-fold cross-validation:
  – Train on 80% of the data, validate on the other 20%.
  – Repeat this 5 more times with different splits, and average the score.

$$X = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \end{bmatrix} \quad Y = \begin{bmatrix} \vdots & \text{"fold" 1} & \text{"fold" 2} & \text{"fold" 3} & \text{"fold" 4} & \text{"fold" 5} \end{bmatrix}$$

1. Train on folds \( \{1, 2, 3, 4, 5\} \), compute error on fold 5.
2. Train on folds \( \{1, 2, 3, 5\} \), compute error on fold 4.
3. Train on folds \( \{1, 2, 4, 5\} \), compute error on fold 3.
4. Take average of the 5 errors as approximation of test error.
Cross-Validation (CV)

• You can take this idea further:
  – 10-fold cross-validation: train on 90% of data and validate on 10%.
    • Repeat 10 times and average.
  – Leave-one-out cross-validation: train on all but one training example.
    • Repeat n times and average.

• Gets more accurate but more expensive with more folds.
  – To choose depth we compute the cross-validation score for each depth.

• As before, if data is ordered then folds should be random splits.
  – Randomize first, then split into fixed folds.
(pause)
The “Best” Machine Learning Model

- Decision trees are not always most accurate on test error.
- **What is the “best” machine learning model?**
- First we need to define *generalization error*:
  - Test error restricted to new feature combinations (no \( x_i \) from train set).
- **No free lunch theorem:**
  - There is no “best” model achieving the best generalization error for every problem.
  - If model A generalizes better to new data than model B on one dataset, there is another dataset where model B works better.
- **This question** is like asking which is “best” among “rock”, “paper”, and “scissors”.


The “Best” Machine Learning Model

• Implications of the lack of a “best” model:
  – We need to learn about and try out multiple models.

• So which ones to study in CPSC 340?
  – We’ll usually motivate each method by a specific application.
  – But we’re focusing on models that have been effective in many applications.

• Caveat of no free lunch (NFL) theorem:
  – The world is very structured.
  – Some datasets are more likely than others.
  – Model A really could be better than model B on every real dataset in practice.

• Machine learning research:
  – Large focus on models that are useful across many applications.
Application: E-mail Spam Filtering

• Want to build a system that detects spam e-mails.
  – Context: spam used to be a big problem.

• Can we formulate as supervised learning?
Spam Filtering as Supervised Learning

• Collect a large number of e-mails, gets users to label them.

<table>
<thead>
<tr>
<th></th>
<th>$</th>
<th>Hi</th>
<th>CPSC</th>
<th>340</th>
<th>Vicodin</th>
<th>Offer</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>…</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>…</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

 Spam?

1

1

0

…

• We can use \( y_i = 1 \) if e-mail ‘i’ is spam, \( y_i = 0 \) if e-mail is not spam.

• Extract features of each e-mail (like bag of words).
  – \( x_{ij} = 1 \) if word/phrase ‘j’ is in e-mail ‘i’, \( x_{ij} = 0 \) if it is not.
Feature Representation for Spam

• Are there better features than bag of words?
  – We add bigrams (sets of two words):
    • “CPSC 340”, “wait list”, “special deal”.
  – Or trigrams (sets of three words):
    • “Limited time offer”, “course registration deadline”, “you’re a winner”.
  – We might include the sender domain:
    • <sender domain == “mail.com”>.
  – We might include regular expressions:
    • <your first and last name>.

• Also, note that we only need list of non-zero features for each $x_i$. 
Review of Supervised Learning Notation

- We have been using the notation ‘$X$’ and ‘$y$’ for supervised learning:

  - $X$ is matrix of all features, $y$ is vector of all labels.
  - We use $y_i$ for the label of object ‘$i$’ (element ‘$i$’ of ‘$y$’).
  - We use $x_{ij}$ for feature ‘$j$’ of object ‘$i$’.
  - We use $x_i$ as the list of features of object ‘$i$’ (row ‘$i$’ of ‘$X$’).

- So in the above $x_3 = [0 \ 1 \ 1 \ 1 \ 0 \ 0 \ ...]$. 

$$
\begin{bmatrix}
\text{\$} & \text{Hi} & \text{CPSC} & \text{340} & \text{Vicodin} & \text{Offer} & \ldots \\
1 & 1 & 0 & 0 & 1 & 0 & \ldots \\
0 & 0 & 0 & 0 & 1 & 1 & \ldots \\
0 & 1 & 1 & 1 & 0 & 0 & \ldots \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\end{bmatrix}
$$

$$
\begin{bmatrix}
\text{Spam?} \\
1 \\
1 \\
0 \\
\ldots \\
\end{bmatrix}
$$
Probabilistic Classifiers

• For years, best spam filtering methods used naïve Bayes.
  – A probabilistic classifier based on Bayes rule.
  – It tends to work well with bag of words.
  – Last year shown to improve on state of the art for CRISPR “gene editing” (link).

• Probabilistic classifiers model the conditional probability, \( p(y_i \mid x_i) \).
  – “If a message has words \( x_i \), what is probability that message is spam?”

• Classify it has spam if probability of spam is higher than not spam:
  – If \( p(y_i = \text{“spam”} \mid x_i) > p(y_i = \text{“not spam”} \mid x_i) \)
    • return “spam”.
  – Else
    • return “not spam”.
Spam Filtering with Bayes Rule

• To model conditional probability, naïve Bayes uses Bayes rule:

\[ p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"}) p(y_i = \text{"spam"})}{p(x_i)} \]

• So we need to figure out three types of terms:
  – Marginal probabilities \( p(y_i) \) that an e-mail is spam.
  – Marginal probability \( p(x_i) \) that an e-mail has the set of words \( x_i \).
  – Conditional probability \( P(x_i \mid y_i) \) that a spam e-mail has the words \( x_i \).
    • And the same for non-spam e-mails.
Spam Filtering with Bayes Rule

\[ p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"})p(y_i = \text{"spam"})}{p(x_i)} \]

• What do these terms mean?

ALL E-MAILS
(including duplicates)
Spam Filtering with Bayes Rule

\[ p(y_i = \text{"spam"} | x_i) = \frac{p(x_i | y_i = \text{"spam"}) p(y_i = \text{"spam"})}{\hat{p}(x_i)} \]

- \( p(y_i = \text{"spam"}) \) is probability that a random e-mail is spam.
  - This is easy to approximate from data: use the proportion in your data.

This is a “maximum likelihood estimate”, a concept we’ll discuss in detail later. If you’re interested in a proof, see here.
Spam Filtering with Bayes Rule

\[ p(y_i = "\text{spam}" \mid x_i) = \frac{p(x_i \mid y_i = "\text{spam}" \cdot p(y_i = "\text{spam}")}{p(x_i)} \]

- \( p(x_i) \) is probability that a random e-mail has features \( x_i \):
  - This is hard to approximate (there are so many possible e-mails).

\[ p(x_i) = \frac{\# \text{e-mails with features } x_i}{\# \text{e-mails total}} \]
Spam Filtering with Bayes Rule

\[ p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"}) p(y_i = \text{"spam"})}{p(x_i)} \]

- \( p(x_i) \) is probability that a random e-mail has features \( x_i \):
  - This is hard to approximate (there are so many possible e-mails), but it turns out we can ignore it:

  **Naive Bayes returns "spam" if**

  \[ p(y_i = \text{"spam"} \mid x_i) > p(y_i = \text{"not spam"} \mid x_i). \]

  **By Bayes rule this means**

  \[ \frac{p(x_i \mid y_i = \text{"spam"}) p(y_i = \text{"spam"})}{p(x_i)} > \frac{p(x_i \mid y_i = \text{"not spam"}) p(y_i = \text{"not spam"})}{p(x_i)}. \]

  **Multiply both sides by** \( p(x_i) \):

  \[ p(x_i \mid y_i = \text{"spam"}) p(y_i = \text{"spam"}) > p(x_i \mid y_i = \text{"not spam"}) p(y_i = \text{"not spam"}). \]
Spam Filtering with Bayes Rule

\[ p(y_i = \text{"spam"} \mid x_i) = \frac{p(x_i \mid y_i = \text{"spam"}) p(y_i = \text{"spam"})}{p(x_i)} \]

- \( p(x_i \mid y_i = \text{"spam"}) \) is probability that spam has features \( x_i \).

- Also hard to estimate.
  - And we need it.
Naïve Bayes

- Naïve Bayes makes a **big assumption** to make things easier:

  \[
  p(\text{hello, vicodin, CPSC 340 | spam}) \approx p(\text{hello | spam}) \cdot p(\text{vicodin | spam}) \cdot p(\text{CPSC 340 | spam})
  \]

  \[\text{HARD}\]

  \[\text{easy} \quad \text{easy} \quad \text{easy}\]

- We assume all features \(x_i\) are **conditionally independent** given label \(y_i\):
  - Once you know it’s spam, probability of “vicodin” doesn’t depend on “CPSC 340”.
  - Definitely not true, but sometimes a good approximation.

- And now we only need easy quantities like \(p(“vicodin” = 1 | y_i = “spam”)\).
Naïve Bayes

- $p(\text{“vicodin”} = 1 \mid \text{“spam”} = 1)$ is probability of seeing “vicodin” in spam.

- Easy to estimate:
  \[
p(\text{vicodin}=1 \mid \text{spam}=1) = \frac{\# \text{spam messages w/ vicodin}}{\# \text{spam messages}}\]
Naïve Bayes

• Naïve Bayes more formally:

\[ p(y_i | x_i) = \frac{p(x_i | y_i) p(y_i)}{p(x_i)} \]  
(First use Bayes rule)

\[ \propto p(x_i | y_i) p(y_i) \]  
("denominator doesn't matter")

\[ \approx \frac{1}{d} \left[ \prod_{j=1}^{d} p(x_{ij} | y_i) \right] p(y_i) \]  
(Conditional independence assumption)

Only needs easy probabilities.

• Post-lecture slides: how to train/test by hand on a simple example.
Summary

• **Optimization bias**: using a validation set too much overfits.
• **Cross-validation**: allows better use of data to estimate test error.
• **No free lunch theorem**: there is no “best” ML model.
• **Probabilistic classifiers**: try to estimate $p(y_i | x_i)$.
• **Naïve Bayes**: simple probabilistic classifier based on counting.
  – Uses conditional independence assumptions to make training practical.

• Next time:
  – A “best” machine learning model as ‘$n$’ goes to $\infty$. 
Naïve Bayes Training Phase

• Training a naïve Bayes model:

\[ X = \begin{bmatrix}
0 & 1 \\
1 & 1 \\
1 & 0 \\
1 & 1 \\
1 & 0 \\
1 & 1 \\
1 & 0 \\
1 & 0 \\
\end{bmatrix}, \quad y = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
0 \\
\end{bmatrix} \]
Naïve Bayes Training Phase

• Training a naïve Bayes model:

1. Set $n_c$ to the number of times ($y_i = c$).
Naïve Bayes Training Phase

• Training a naïve Bayes model:
  1. Set $n_c$ to the number of times ($y_i = c$).
  2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.

\[
X = \begin{bmatrix}
0 & 1 \\
1 & 1 \\
0 & 0 \\
1 & 1 \\
1 & 1 \\
0 & 0 \\
1 & 0 \\
1 & 0 \\
1 & 1 \\
1 & 0 \\
\end{bmatrix}, \quad y = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}
\]

\[
p(y_i = 1) = \frac{6}{10} \leftarrow n_i = 6
\]

\[
p(y_i = 0) = \frac{4}{10} \leftarrow n_0 = 4
\]
Naïve Bayes Training Phase

- Training a naïve Bayes model:

1. Set $n_c$ to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.
3. Set $n_{ck}$ as the number of times $(y_i = c, x_j = k)$.

$$X = \begin{bmatrix}
0 & 1 & 1 \\
1 & 1 & 1 \\
0 & 0 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
0 & 0 & 0 \\
1 & 0 & 0 \\
1 & 1 & 0 \\
1 & 1 & 1 \\
1 & 0 & 0 \\
\end{bmatrix}, \quad y = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}$$

$$p(y_i = 1) = \frac{6}{10} \iff n_i = 6$$

$$p(y_i = 0) = \frac{4}{10} \iff n_0 = 4$$

$n_{121} = 4$
Naïve Bayes Training Phase

• Training a naïve Bayes model:

1. Set $n_c$ to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.
3. Set $n_{ck}$ as the number of times $(y_i = c, x_i = k)$.
4. Estimate $p(x_i = k | y_i = c)$ as $\frac{n_{ck}}{n_i}$.

\[ X = \begin{bmatrix}
0 & 1 & 1 \\
1 & 1 & 1 \\
0 & 0 & 1 \\
1 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
1 & 0 & 1 \\
1 & 0 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix}, \quad y = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
0 \\
0 \\
0
\end{bmatrix} \]

\[
\frac{p(x_i = 1 | y_i = 1)}{p(y_i = 0)} = \frac{4}{10} \triangleq n_0 = 4
\]

\[
\frac{p(y_i = 1)}{p(y_i = 0)} = \frac{6}{10} \triangleq n_1 = 6
\]
Naïve Bayes Training Phase

• Training a naïve Bayes model:

1. Set $n_c$ to the number of times $(y_i = c)$.
2. Estimate $p(y_i = c)$ as $\frac{n_c}{n}$.
3. Set $n_{cjk}$ as the number of times $(y_i = c, x_{ij} = k)$.
4. Estimate $p(x_{ij} = k \mid y_i = c)$ as $\frac{n_{cjk}}{n}$.
5. Use that $p(x_{ij} = k \mid y_i = c) = \frac{p(x_{ij} = k, y_i = c)}{p(y_i = c)} = \frac{n_{cjk}/n}{n_c/n} = \frac{n_{cjk}}{n_c}$

\[
p(x_i=1 \mid y_i=1) = \frac{4}{10} = \frac{2}{3}
\]
\[
p(y_i=0) = \frac{4}{10}
\]
\[
p(y_i=1) = \frac{6}{10} \leftarrow n_i = 6
\]
\[
x = \begin{bmatrix}
0 & 1 & 1 \\
1 & 1 & 0 \\
0 & 0 & 1 \\
1 & 1 & 1 \\
0 & 0 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
1 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix}
\]
\[
y = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
0 \\
1 \\
0 \\
0
\end{bmatrix}
\]
\[
n_{121} = 4
\]
Naïve Bayes Prediction Phase

• Prediction in a naïve Bayes model:

Given a test example \( \tilde{x}_i \), we want to find the \( 'c' \) maximizing \( p(\tilde{x}_i | \tilde{y}_i = c) \)

Under the naïve Bayes assumption, we can maximize:

\[
p(\tilde{y}_i = c | \tilde{x}_i) \propto \prod_{j=1}^{d} \left[p(\tilde{x}_{ij} | \tilde{y}_i = c) \right] p(\tilde{y}_i = c)
\]
Naïve Bayes Prediction Phase

• Prediction in a naïve Bayes model:

Given a test example $\mathbf{x}_i$, we set prediction $\hat{y}_i$ to the 'c' maximizing $p(\mathbf{x}_i | \hat{y}_i = c)$

Under the naïve Bayes assumption we can maximize:

$$p(\hat{y}_i = c | \mathbf{x}_i) \propto \prod_{j=1}^{d} \left[ p(\mathbf{x}_{ij} | \hat{y}_i = c) \right] p(\hat{y}_i = c)$$
Naïve Bayes Prediction Phase

• Prediction in a naïve Bayes model:

Consider $\mathbf{x}_i = [1, 1]$ in this data set.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
Naïve Bayes Prediction Phase

• Prediction in a naïve Bayes model:

Consider $\tilde{x}_i = [1 1]$ in this data set.

\[
p(\tilde{y}_i = 0 | \tilde{x}_i) \propto p(\tilde{x}_1 = 1 | \tilde{y}_i = 0) p(\tilde{x}_2 = 1 | \tilde{y}_i = 0) p(\tilde{y}_i = 0)
\]

\[
= (1) \times (0.25) \times (0.4) = 0.1
\]

\[
X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}
\]
Naïve Bayes Prediction Phase

• Prediction in a naïve Bayes model:

Consider $\tilde{x}_i = [1, 1]$ in this data set. $p(\tilde{y}_i = 0 | \tilde{x}_i) \propto p(\tilde{x}_1 = 1 | \tilde{y}_i = 0) p(\tilde{x}_2 = 1 | \tilde{y}_i = 0) p(\tilde{y}_i = 0)$

$= (1) \cdot (0.25) \cdot (0.4) = 0.1$

$p(\tilde{y}_i = 1 | \tilde{x}_i) \propto p(\tilde{x}_1 = 1 | \tilde{y}_i = 1) p(\tilde{x}_2 = 1 | \tilde{y}_i = 1) p(\tilde{y}_i = 1)$

$= (0.5) \cdot (0.666...) \cdot (0.6) = 0.2$

$X = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$
Naïve Bayes Prediction Phase

• Prediction in a naïve Bayes model:

Consider $\mathbf{x}_i = [1 \ 1]$ in this data set.

\[
p(\tilde{y}_i = 0 | \tilde{x}_i) \propto p(\tilde{x}_i = 1 | \tilde{y}_i = 0) p(\tilde{x}_2 = 1 | \tilde{y}_i = 0) p(\tilde{y}_i = 0) = (1)(0.25)(0.4) = 0.1
\]

\[
x = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}
\]

\[
p(\tilde{y}_i = 1 | \tilde{x}_i) \propto p(\tilde{x}_i = 1 | \tilde{y}_i = 1) p(\tilde{x}_2 = 1 | \tilde{y}_i = 1) p(\tilde{y}_i = 1) = (0.5)(0.666...) (0.6) = 0.2
\]

Since $p(\tilde{y}_i = 1 | \tilde{x}_i)$ is bigger than $p(\tilde{y}_i = 0 | \tilde{x}_i)$, naïve Bayes predicts $\hat{y}_i = 1$.

(Don't sum to 1 because we're ignoring $p(\tilde{x}_i)$.)
Avoiding Underflow

- During the prediction, the probability can underflow:

\[ p(y_i = c \mid x_i) \propto \prod_{j=1}^{d} \left[ p(x_{ij} \mid y_i = c) \right] p(y_i = c) \]

All these are \(<1\) so the product gets very small.

- Standard fix is to (equivalently) maximize the logarithm of the probability:

Remember that \( \log(ab) = \log(a) + \log(b) \) so \( \log(\prod a_i) = \sum_{i} \log(a_i) \)

Since \( \log \) is monotonic the 'c' maximizing \( p(y_i = c \mid x_i) \) also maximizes \( \log p(y_i = c \mid x_i) \)

So maximize \( \log \left( \prod_{j=1}^{d} \left[ p(x_{ij} \mid y_i = c) \right] p(y_i = c) \right) = \sum_{j=1}^{d} \log(p(x_{ij} \mid y_i = c)) + \log(p(y_i = c)) \)
Back to Decision Trees

• Instead of validation set, you can use CV to select tree depth.

• But you can also use these to decide **whether to split**:
  – Don’t split if validation/CV error doesn’t improve.
  – Different parts of the tree will have different depths.

• Or fit deep decision tree and **use CV to prune**:
  – Remove leaf nodes that don’t improve CV error.

• Popular implementations that have these tricks and others.
Cross-Validation Theory

• Does CV give unbiased estimate of test error?
  – Yes!
    • Since each data point is only used once in validation, expected validation error on each data point is test error.
    – But again, if you CV to select among models then it is no longer unbiased.

• What about variance of CV?
  – Hard to characterize.
  – CV variance on ‘n’ data points is worse than with a validation set of size ‘n’.
    • But we believe it is close.
Handling Data Sparsity

• Do we need to store the full bag of words 0/1 variables?
  – No: only need list of non-zero features for each e-mail.
  – Math/model doesn’t change, but more efficient storage.

| $ | Hi | CPSC | 340 | Vicodin | Offer | ...
|---|-----|------|-----|---------|-------|-----
| 1 | 1   | 0    | 0   | 1       | 0     | ...|
| 0 | 0   | 0    | 0   | 1       | 1     | ...|
| 0 | 1   | 1    | 1   | 0       | 0     | ...|
| 1 | 1   | 0    | 0   | 0       | 1     | ...

Non-Zeroes

\{1,2,5,\ldots\}
\{5,6,\ldots\}
\{2,3,4,\ldots\}
\{1,2,6,\ldots\}
Less-Naïve Bayes

- Given features \{x_1, x_2, x_3, \ldots, x_d\}, naïve Bayes approximates \(p(y|x)\) as:
  \[
p(y | x_1, x_2, \ldots, x_d) \propto p(y) p(x_1 | y) p(x_2 | x_1, y) p(x_3 | x_2, x_1, y) \cdots p(x_d | x_{d-1}, y)
  \approx p(y) p(x_1 | y) p(x_2 | y) p(x_3 | y) \cdots p(x_d | y)
\]
  \(\text{product rule applied repeatedly}\)

- The assumption is very strong, and there are “less naïve” versions:
  - Assume independence of all variables except up to ‘k’ largest ‘j’ where \(j < i\).
    - E.g., naïve Bayes has \(k=0\) and with \(k=2\) we would have:
      \[
      \approx p(y) p(x_1 | y) p(x_2 | y) p(x_3 | y) p(x_4 | x_3, x_2, y) \cdots p(x_d | x_{d-2}, x_{d-1}, y)
      \]
    - Fewer independence assumptions so more flexible, but hard to estimate for large ‘k’.
  - Another practical variation is “tree-augmented” naïve Bayes.
Gaussian Discriminant Analysis

• Classifiers based on Bayes rule are called generative classifier:
  – They often work well when you have tons of features.
  – But they need to know $p(x_i | y_i)$, probability of features given the class.
    • How to “generate” features, based on the class label.

• To fit generative models, usually make BIG assumptions:
  – Naïve Bayes (NB) for discrete $x_i$:
    • Assume that each variables in $x_i$ is independent of the others in $x_i$ given $y_i$.

  – Gaussian discriminant analysis (GDA) for continuous $x_i$.
    • Assume that $p(x_i | y_i)$ follows a multivariate normal distribution.
    • If all classes have same covariance, it’s called “linear discriminant analysis”.
Computing $p(x_i)$ under naïve Bayes

- **Generative models** don’t need $p(x_i)$ to make decisions.
- However, it’s **easy to calculate** under the naïve Bayes assumption:

$$p(x_i) = \sum_{c=1}^{K} p(x_i, y = c) \quad (\text{marginalization rule})$$

$$= \sum_{c=1}^{K} p(x_i \mid y = c) p(y = c) \quad (\text{product rule})$$

$$= \sum_{c=1}^{K} \left[ \prod_{j=1}^{d} p(x_{ij} \mid y = c) \right] p(y = c) \quad (\text{naïve Bayes assumption})$$

These are the quantities we compute during training.