Lecture 13: Naïve Bayes Classifier Review
Outline

- Naïve Bayes and machine learning wrap-up
- Midterm review
whosaid: a Naïve Bayes classifier

- How did the classifier do?
  - **0.951 accuracy** on the test data, using a fixed random data split.

- Training set: **15,152 sentences**
  - **6,672** are Austen
    - \( P(\text{austen}) = 0.44 \)
    - **Austen prior**
  - **8,480** are Melville
    - \( P(\text{melville}) = 0.56 \)
    - **Melville prior**

  \( \leftarrow \) Sentences have a higher chance of being Melville out of the gate!
whosaid: error analysis

- **ma** (really Melville, classified as Austen)
  
  0.9947 At first sight, you would not think it so strong as it really is.  
  0.8933 He feels that his dreadful punishment is just.  
  0.7817 And here, shipmates, is true and faithful repentance; not clamorous for pardon, but grateful for punishment.  
  0.6192 I knew no one in the place.  
  0.5713 Indeed, in other respects, you can hardly regard any creatures of the deep with the same feelings that you do those of the shore.  
  0.5528 "Oh!"

- **am** (really Austen, classified as Melville)
  
  0.9911 It is a sort of prologue to the play, a motto to the chapter; and will be soon followed by matter - of - fact prose."  
  0.9639 In this age of literature, such collections on a very grand scale are not uncommon.  
  0.8823 And at others, what a heap of absurdities it is!  
  0.7826 shark is only one syllable.  
  0.7251 said he, offering his hand.  
  0.6601 "Here is April come!"
### Informative features (_all)_

<table>
<thead>
<tr>
<th>Feature</th>
<th>Score</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'contains-emma', 1</td>
<td>1864.5</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-whale', 1</td>
<td>1522.5</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-harriet', 1</td>
<td>1048.5</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-weston', 1</td>
<td>926.5</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-knightley', 1</td>
<td>840.1</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-elton', 1</td>
<td>771.5</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-ship', 1</td>
<td>696.3</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-ahab', 1</td>
<td>666.4</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-woodhouse', 1</td>
<td>652.0</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-jane', 1</td>
<td>613.9</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-fairfax', 1</td>
<td>507.1</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-churchill', 1</td>
<td>469.0</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-boat', 1</td>
<td>424.1</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-miss', 1</td>
<td>381.7</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-hartfield', 1</td>
<td>362.2</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-whales', 1</td>
<td>345.4</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-queequeg', 1</td>
<td>337.5</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-stubb', 1</td>
<td>325.0</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-sperm', 1</td>
<td>318.7</td>
<td>melvil : austen</td>
</tr>
<tr>
<td>'contains-bates', 1</td>
<td>311.4</td>
<td>melvil : austen</td>
</tr>
</tbody>
</table>
Informative features, noCharNames

<table>
<thead>
<tr>
<th>Feature</th>
<th>Melvil: Austen</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>contains-whale, 1</td>
<td>melvil: austen</td>
<td>1522.5</td>
</tr>
<tr>
<td>contains-ship, 1</td>
<td>melvil: austen</td>
<td>696.3</td>
</tr>
<tr>
<td>contains-boat, 1</td>
<td>melvil: austen</td>
<td>424.1</td>
</tr>
<tr>
<td>contains-miss, 1</td>
<td>austen: melvil</td>
<td>381.7</td>
</tr>
<tr>
<td>contains-whales, 1</td>
<td>melvil: austen</td>
<td>345.4</td>
</tr>
<tr>
<td>contains-sperm, 1</td>
<td>melvil: austen</td>
<td>318.7</td>
</tr>
<tr>
<td>contains-deck, 1</td>
<td>melvil: austen</td>
<td>271.5</td>
</tr>
<tr>
<td>contains-crew, 1</td>
<td>melvil: austen</td>
<td>195.9</td>
</tr>
<tr>
<td>contains-boats, 1</td>
<td>melvil: austen</td>
<td>195.9</td>
</tr>
<tr>
<td>contains-mast, 1</td>
<td>melvil: austen</td>
<td>175.5</td>
</tr>
<tr>
<td>contains-whaling, 1</td>
<td>melvil: austen</td>
<td>175.5</td>
</tr>
<tr>
<td>contains--`, 1</td>
<td>austen: melvil</td>
<td>166.5</td>
</tr>
<tr>
<td>contains-thee, 1</td>
<td>melvil: austen</td>
<td>162.9</td>
</tr>
<tr>
<td>contains-ll, 1</td>
<td>melvil: austen</td>
<td>142.4</td>
</tr>
<tr>
<td>contains-sail, 1</td>
<td>melvil: austen</td>
<td>137.7</td>
</tr>
<tr>
<td>contains-voyage, 1</td>
<td>melvil: austen</td>
<td>137.7</td>
</tr>
<tr>
<td>contains-flask, 1</td>
<td>melvil: austen</td>
<td>134.5</td>
</tr>
<tr>
<td>contains-ships, 1</td>
<td>melvil: austen</td>
<td>125.1</td>
</tr>
<tr>
<td>contains-leviathan, 1</td>
<td>melvil: austen</td>
<td>125.1</td>
</tr>
<tr>
<td>contains-cabin, 1</td>
<td>melvil: austen</td>
<td>118.8</td>
</tr>
</tbody>
</table>
He, she, very

```python
>>> whosaid.classify(gen_feats('He knows the truth'.split()))
'melville'
>>> whosaid.prob_classify(gen_feats('He knows the truth'.split())).prob('austen')
0.44921141639835876
>>> whosaid.prob_classify(gen_feats('She knows the truth'.split())).prob('austen')
0.9314339848201395
>>> whosaid.feature_weights('contains-he', 1)
{'melville': 0.155461574106827, 'austen': 0.16881462610520007}
>>> whosaid.feature_weights('contains-she', 1)
{'melville': 0.011496285815351963, 'austen': 0.2079274689045407}
>>> whosaid.feature_weights('contains-very', 1)
{'melville': 0.0321306449711119, 'austen': 0.13899295669114342}
```
A sentence with "whale" categorized Austen?
- Start with "a whale", then gradually add words to make the sentence more "Austen".

A perfectly ambiguous sentence?
- Can you come up with a sentence that's at least 5 words long that is as close to 50-50 Austen-Melville?

Which word feature is neutral?
- You will need to think "odds ratio".
- whosaid.feature_weights('contains-...', 1) is the function to use.

When you and your buddy have an answer, paste a screenshot on MS Teams!
def getOddsRatio(word):
    fw = whosaid.feature_weights('contains-' + word, 1)
    print(fw)
    aweight = fw['austen']
    mweight = fw['melville']
    if aweight > mweight:
        print('austen-melville odds ratio', round(aweight/mweight, 2))
    else:
        print('melville-austen odds ratio', round(mweight/aweight, 2))

>>> getOddsRatio('sea')
{'melville': 0.04533663483079826, 'austen': 0.0018732204405814477}
melville-austen odds ratio 24.20
>>> getOddsRatio('unfortunate')
{'melville': 0.00029477655936799903, 'austen': 0.0011239322643488685}
austen-melville odds ratio 3.81
>>> getOddsRatio('must')
{'melville': 0.02835750501120151, 'austen': 0.07095759028922524}
austen-melville odds ratio 2.50
>>> getOddsRatio('!')
{'melville': 0.11348897535667964, 'austen': 0.06601228832609021}
melville-austen odds ratio 1.72
>>> getOddsRatio('why')
{'melville': 0.01232166018158236, 'austen': 0.006518807133223438}
melville-austen odds ratio 1.89
>>> getOddsRatio('the')
{'melville': 0.5981016389576701, 'austen': 0.37636745092162444}
melville-austen odds ratio 1.59
>>> getOddsRatio('at')
{'melville': 0.12186062964273081, 'austen': 0.11846246066237075}
melville-austen odds ratio 1.03

Many function words are not neutral, lean towards Melville or Austen

'at' is almost perfectly neutral
Austen vs. *whale*

Can a sentence with 'whale' ever be classified as 'austen'?

```python
>>> whosaid.prob_classify(gen_feats('a whale'.split())).prob('austen')
0.00046963208159057055
>>> whosaid.prob_classify(gen_feats('a beautiful whale'.split())).prob('austen')
0.001629566209242024
>>> whosaid.prob_classify(gen_feats('she married a whale'.split())).prob('austen')
0.10371709682345985
>>> whosaid.prob_classify(gen_feats('she married a beautiful whale'.split())).prob('austen')
0.28673216572155275
>>> whosaid.prob_classify(gen_feats('she married a very beautiful whale'.split())).prob('austen')
0.6349019382913935
```

Even though 'whale' never occurs in Austen, 'contains-whale', 1 for 'austen' gets assigned a tiny weight through smoothing.
More in homework KEY

- We went over the solutions in class.
- Will be posted on Canvas! (Along with HW2 KEY)
whosaid vs. movie review classifier

- whosaid on tiny sentences with strong features:

```python
>>> whosaid.prob_classify(gen_feats('he was a whale'.split())).prob('austen')
0.0008505667723433306
>>> whosaid.prob_classify(gen_feats('she was delighted'.split())).prob('austen')
0.9967617928216123
```

- The movie review classifier behaves very differently:

```python
contains(outstanding) = True  pos : neg = 11.0 : 1.0
contains(mulan) = True        pos : neg = 7.7 : 1.0
contains(seagal) = True       neg : pos = 7.4 : 1.0
contains(damon) = True        pos : neg = 5.7 : 1.0
contains(awful) = True        neg : pos = 5.6 : 1.0

>>> classifier.prob_classify(document_features('damon was outstanding'.split())).prob('neg')
0.9999998931163593
>>> classifier.prob_classify(document_features('seagal was awful'.split())).prob('neg')
0.9999999999655637
```

Both strongly neg? How could this be?
whosaid vs. movie review classifier

- whosaid on tiny sentences with strong features:

  ```
  >>> whosaid.prob_classify(gen_feats('he was a whale'.split())).prob('austen')
  0.000850566772343306
  >>> whosaid.prob_classify(gen_feats('she was delighted'.split())).prob('austen')
  0.9967617928216123
  ```

  Whosaid only encodes presence of a word as a feature.

  *Four features of value 1 for this sentence*

- The movie review classifier behaves very differently:

  ```
  contains(outstanding) = True
  contains(mulan) = True
  contains(seagal) = True
  contains(damon) = True
  contains(awful) = True
  ```

  ```
  contains(outstanding) = True  pos : neg = 11.0 : 1.0
  contains(mulan) = True       pos : neg = 7.7 : 1.0
  contains(seagal) = True       neg : pos = 7.4 : 1.0
  contains(damon) = True       pos : neg = 5.7 : 1.0
  contains(awful) = True       neg : pos = 5.6 : 1.0
  ```

  ```
  classifier.prob_classify(document_features('damon was outstanding'.split())).prob('neg')
  0.9999998931163593
  classifier.prob_classify(document_features('seagal was awful'.split())).prob('neg')
  0.9999999999655637
  ```

  Here, 2000 most common words are encoded as 'presence' or 'absence' features.

  *Becomes a set of 2,000 True/False features!*
whosaid vs. movie review classifier

- whosaid on tiny sentences with strong features:

```python
>>> whosaid.prob_classify(gen_feats('he was a whale'.split())).prob('austen')
0.0008505667723433306
>>> whosaid.prob_classify(gen_feats('she was delighted'.split())).prob('austen')
0.9967617928216123
```

- The movie review classifier behaves very differently:

```python
contains(outstanding) = True            pos : neg    =     11.0 : 1.0
contains(mulan) = True                  pos : neg    =      7.7 : 1.0
contains(seagal) = True                 neg : pos    =      7.4 : 1.0
contains(damon) = True                  neg : pos    =      5.7 : 1.0
contains(awful) = True                  neg : pos    =      5.6 : 1.0
```

```python
>>> classifier.prob_classify(document_features('damon was outstanding'.split())).prob('neg')
0.99999998931163593
>>> classifier.prob_classify(document_features('seagal was awful'.split())).prob('neg')
0.9999999999655637
```

What is NOT in this sentence does not affect labeling decision at all. All top 2,000 words, even those not in this review, affect the labeling decision!
Collective power of features

Voting for "positive":
- 'damon' & 'outstanding', strongly positive-leaning, for being IN the review
- All negative-learning words (e.g., 'awful') for NOT BEING IN the review

Voting for "negative":
- 'was', which turns out leans slightly negative, for being IN the review
- All the rest (1000+!!) positive-learning words for NOT BEING IN the review

```
contains(outstanding) = True            pos : neg    =     11.0 : 1.0
contains(mulan) = True            pos : neg    =      7.7 : 1.0
contains(seagal) = True            neg : pos    =      7.4 : 1.0
contains(damon) = True            pos : neg    =      5.7 : 1.0
contains(awful) = True            neg : pos    =      5.6 : 1.0

>>> classifier.prob_classify(document_features('damon was outstanding'.split())).prob('neg')
0.9999998931163593

>>> classifier.prob_classify(document_features('seagal was awful'.split())).prob('neg')
0.9999999999655637
```
Naïve Bayes classifier: variants

(1) WhoSaid
(2) Movie Review classifier

In both, features had *discreet, categorical* values (1, True/False)

- Can we use actual word *count* (2, 3, 5, ...) as numerical feature values, instead of just presence/(absence)?
  - "movie is fantastic ... fantastic ... fantastic" ← 3 times!
  - Yes it can be done. It's common to convert raw counts into what's known as **TF-IDF** (Term Frequency -- Inverse Document Frequency) with a normalized value between 0 and 1.
Naïve Bayes: strength

- whosaid is a fairly simple statistical model.
- Yet it achieves 95.1% accuracy.
- Why is it so successful?
  - Is it just because of a handful of strong, topical features like 'whale', 'ship'...?
- What are the strengths of Naïve Bayes classifier?
A classification decision involves reconciling **multiple features** with different levels of predictive power.

Different types of classifiers use different algorithms for:

1. Determining the **weights of individual features** in order to maximize its labeling success in the training data
2. When given an input, using the feature weights to **compute the likelihood of a label**

Popular machine learning methods:

- **Naïve Bayes**
- Hidden Markov model (HMM)
- Maximum entropy (ME)
- Decision tree
- Support vector machine (SVM)
- Neural network → Deep learning (!!!)
Weighting the evidence

- A classification decision involves reconciling multiple features with different levels of predictive power.

Different types of classifiers use different algorithms for:

1. Determining the weights of individual features in order to maximize its labeling success in the training data.
2. When given an input, using the feature weights to compute the likelihood of a label.

- Popular machine learning methods:
  - Naïve Bayes
  - Hidden Markov model (HMM)
  - Maximum entropy (ME)
  - Decision tree
  - Support vector machine (SVM)
  - Neural network → Deep learning (!!!)

With more sophisticated ML models, the relationship becomes more complex to the point of almost completely opaque (Deep Learning).
'Machine Learning is too easy'  [https://hunch.net/?p=634](https://hunch.net/?p=634)  
(2009: before Deep Learning's time)

- **WEKA**: a collection of machine learning algorithms for data mining
- **Scikit-Learn** (Python library for ML)

Deep Learning libraries
  - PyTorch (Facebook)
    - [https://pytorch.org/](https://pytorch.org/)
  - Tensorflow (Google)
  - MXNet (Amazon)
    - [https://aws.amazon.com/mxnet/](https://aws.amazon.com/mxnet/)
Wrapping up

- Midterm on Thu. ➔ Details next slide

- Regular expressions and FSA
  - *Language and Computers*, Ch.4 Searching
    - 4.4 Searching semi-structured data with regular expressions
    - 4.41 Syntax of regular expressions
  - NLTK 3.4 Regular expressions
    - [https://www.nltk.org/book/ch03.html#sec-regular-expressions-word-patterns](https://www.nltk.org/book/ch03.html#sec-regular-expressions-word-patterns)
  - J&M Regular expressions
    - [https://web.stanford.edu/~jurafsky/slp3/2.pdf](https://web.stanford.edu/~jurafsky/slp3/2.pdf)
Midterm exam: what to expect

- **10/12 (Thursday)**
  - 75 minutes.
  - At LMC's PC Lab (G17 CL) ← NOT our usual classroom!

- **Exam format:**
  - Closed book. All pencil-and-paper.
  - Topical questions: "what is/discuss/analyze/find out/calculate..."
  - Bring your calculator! →

  - Front and back.
  - Hand-written only.