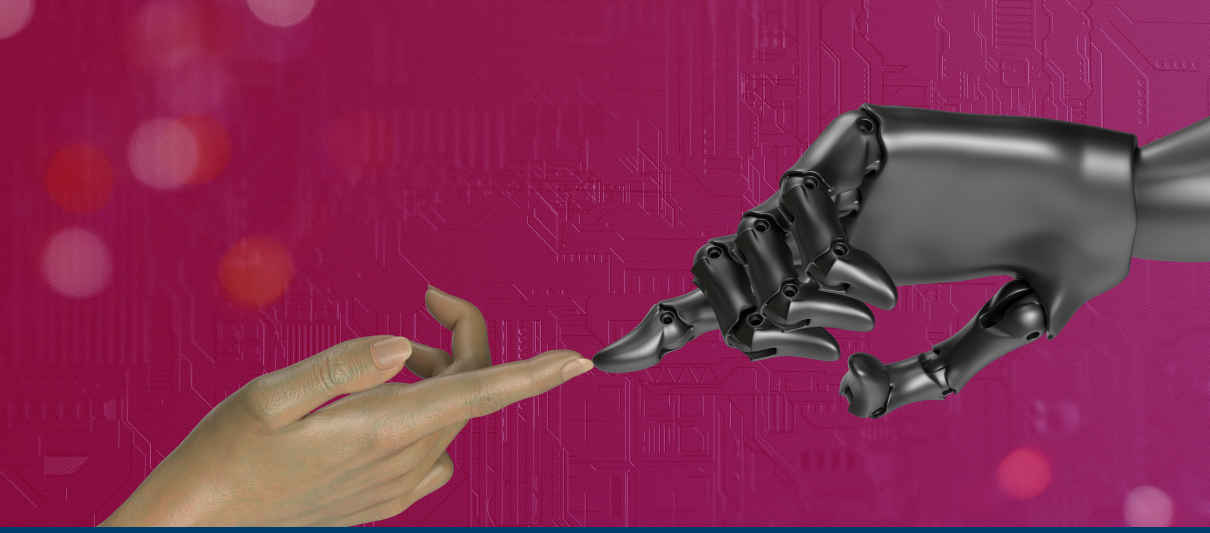

Hands-On Machine Learning

Dr. Derek Bridge | 2026/2027

UCC Coláiste na hOllscoile Corcaigh
University College Cork, Ireland

School of Computer Science and Information Technology
Scoil na Ríomheolaíochta agus na Teicneolaíochta Faisnéise

14. Text Classification

Stanford University researchers have taken 50,000 movie reviews from IMDB, labelled them as either positive or negative and made them available. I've taken 25,000 of them to use as a dataset to illustrate this lecture.

Possible Uses of Bag-Of-Words

Bag-Of-Words might be a competitive approach on tasks such as:

- **spam filtering**
- **email sorting**, e.g. announcements, personal, meetings, ...
- **sentiment analysis**, e.g. positive, negative, neutral
- **language identification**
- **topic classification**, e.g. sports, politics, travel, ...
- **inappropriate content filtering**, e.g. racism, sexism, sexually explicit content, extremism,...

Discussion

Why might a classifier of social media posts incorrectly classify some of them as inappropriate when they are not?

Advantages of Bag-Of-Words

- Simple.
- Efficient and scalable.
- Surprisingly effective at many text classification tasks.

Disadvantages of Bag-Of-Words

- Does not try to represent the semantics of the text.
- In particular, it ignores word order.
- In particular, it is unaware of synonyms.
- In particular, if we discard stop-words, then we lose the relationships that the stop-words convey.
- We may have to deal with its high dimensionality.
- To accommodate new vocabulary (e.g. from new training examples) requires that we run Text Preprocessing afresh — we end up with a different set of features (tokens).

Text Preprocessing (from previous lecture)

- Tokenization.
- Optionally, discard stop-words.
- Optionally, apply stemming or lemmatization.
- Either count vectorization or TF-IDF vectorization.

In reality...

We may need lots of additional preprocesssing, especially if the dataset has been scraped from the Web. For example:

- Some examples may use different character encodings, not UTF-8.
- They may not all be written in the same language, e.g. some in English, others in Hindi.
- There may be speling errors and tpyos.
- There may be markup (e.g. HTML) or other 'formatting' (e.g. SMTP headers, MIME headers).

High Dimensionality

- After Text Preprocessing, we can use standard Machine Learning algorithms to train classifiers, regressors, etc.
- However, n will be high – the number of features (the number of distinct tokens) — and the vectors will often be sparse.
- Reduce the number of features by:
  - discarding tokens that appear in too few documents (set a minimum df);
  - discarding tokens that appear in too many documents (set a maximum df);
  - keeping only the most frequent tokens (ordered by tf).
- Use dimensionality reduction: e.g. Singular Value Decomposition (SVD) is suitable for Bag-Of-Words, rather than PCA.

Naive Bayes Classifier

- **Naive Bayes** is a linear classifier — simple, fast, often accurate — very popular, especially for simple text classification.
- Naive Bayes is a multiclass classifier but we will present the maths for the binary case, $\mathcal{C} = \{c, c'\}$.
- Let's assume we want to classify a document, *doc*. To simplify notation, assume *doc* is a bag of words (bag of tokens).
- Compute the **likelihood** that the class is c given *doc*, $likelihood(c \mid doc)$, as follows:

$$likelihood(c \mid doc) = Prob(c) \prod_{t \in doc} Prob(t \mid c)$$

- Similarly, compute $likelihood(c' \mid doc)$
- Choose the class with the higher likelihood.

$likelihood(c \mid doc) = Prob(c) \prod_{t \in doc} Prob(t \mid c)$

- $Prob(c)$ is the **class prior** — the probability of encountering this class.
- Estimate it from the training set:

$$Prob(c) = \frac{m_c}{m}$$

where m_c is the number of training examples whose class label is c, and m is the number of training examples.

$likelihood(c \mid doc) = Prob(c) \prod_{t \in doc} Prob(t \mid c)$

- $Prob(t \mid c)$ is the **conditional probability** that token t occurs in documents of class c.
- Estimate it from the training set:

$$Prob(t \mid c) = \frac{freq_{t,c}}{freq_c}$$

where $freq_{t,c}$ is the number of occurrences of token t in training examples whose class label is c, and $freq_c$ is the number of occurrences of tokens of any kind in training examples whose class label is c.

- But for some token-class combinations, this will be zero. This will make the final likelihood values also zero.
- **Laplace smoothing**: to eliminate zeros, we add 1 to each count

$$Prob(t \mid c) = \frac{freq_{t,c} + 1}{freq_c + n}$$

where n is the size of the vocabulary (number of distinct tokens).

Example

Training set

brill	funny	lame	mint	short	tame	class label
1	2	0	0	0	0	pos
0	2	0	0	1	0	pos
0	1	0	1	0	0	pos
0	1	1	0	0	1	neg

Test example (doc)

brill	funny	lame	mint	short	tame	class
0	3	1	0	0	1	?

Is it pos?

$$Prob(\text{pos}) = 3/4$$

$$\begin{aligned} Prob(\text{funny} \mid \text{pos}) &= (5+1)/(8+6) &&= 6/14 \\ Prob(\text{lame} \mid \text{pos}) &= (0+1)/(8+6) &&= 1/14 \\ Prob(\text{tame} \mid \text{pos}) &= (0+1)/(8+6) &&= 1/14 \end{aligned}$$

$$likelihood(\text{pos} \mid doc) = 3/4 \times (6/14)^3 \times 1/14 \times 1/14 = 0.0003$$

Is it neg?

$$Prob(\text{neg}) = 1/4$$

$$\begin{aligned} Prob(\text{funny} \mid \text{neg}) &= (1+1)/(3+6) &&= 2/9 \\ Prob(\text{lame} \mid \text{neg}) &= (1+1)/(3+6) &&= 2/9 \\ Prob(\text{tame} \mid \text{neg}) &= (1+1)/(3+6) &&= 2/9 \end{aligned}$$

$$likelihood(\text{neg} \mid doc) = 1/4 \times (2/9)^3 \times 2/9 \times 2/9 = 0.0001$$

Probabilities

If you want the classifier to output probabilities, then divide the likelihood by $Prob(doc)$:

$$Prob(c \mid doc) = \frac{Prob(c) \prod_{t \in doc} Prob(t \mid c)}{Prob(doc)}$$
$$= \frac{likelihood(c \mid doc)}{likelihood(c \mid doc) + likelihood(c' \mid doc)}$$

Example

- $Prob(c \mid doc) = \frac{0.0003}{0.0003+0.0001} = 0.75$
- $Prob(c' \mid doc) = \frac{0.0001}{0.0003+0.0001} = 0.25$

Why "Bayes"?

- **Bayes' theorem** states that the posterior probability $Prob(H \mid E)$ of a hypothesis H given evidence E is given by

$$Prob(H \mid E) = \frac{Prob(H)Prob(E \mid H)}{Prob(E)}$$

- Schematically,

$$posterior\ \ probability = \frac{prior\ \ probability \times conditional\ \ probability}{evidence}$$

- In our case,

$$Prob(c \mid doc) = \frac{Prob(c)Prob(doc \mid c)}{Prob(doc)}$$

Why "Naive"?

- The Naive Bayes classifier makes the naive simplifying assumption that $Prob(doc \mid c)$ can be calculated as follows:

$$Prob(doc \mid c) = \prod_{t \in doc} Prob(t \mid c)$$

  e.g. $Prob(\{\text{funny}, \text{funny}, \text{mint}\} \mid c)$ is given by $Prob(\text{funny} \mid c) \times Prob(\text{funny} \mid c) \times Prob(\text{mint} \mid c)$.
- In other words, it assumes **conditional independence** — hence we can multiply these probabilities.
- In truth, this assumption does not hold.
- E.g. if a movie review contains the word "car", it will be more likely that it contains the word "chase".
- But, in practice, Naive Bayes is effective even though the assumption is violated.

Underflow

- The formula requires that we multiply a lot of probabilities.
- This can result in **floating point underflow** — a result so small that it cannot be represented.
- But we can 'demote' multiplication to addition by using **logarithms**:

$$\log(xy) = \log(x) + \log(y)$$

- So we use

$$loglikelihood(c \mid doc) = \log Prob(c) + \sum_{t \in doc} \log Prob(t \mid c)$$

— the class with highest log-likelihood is still the most probable.

Training

During training, we can estimate all the probabilities we might ever need from the training set and cache them.

foreach $c \in \mathcal{C}$ **do**
    logprior[c] $= \log(m_c/m)$;
    foreach $t \in$ vocabulary **do**
        logcondprob[t][c] $= \log(\frac{freq_{t,c}+1}{freq_c+n})$
    end
end

Inference

To predict the class of *doc* (which is a bag):

foreach $c \in \mathcal{C}$ **do**
    loglikelihood[c] $\leftarrow$ logprior[c];
    foreach $t \in doc$ **do**
        loglikelihood[c] += logcondprob[t][c]
    end
end
return c for which loglikelihood[c] is highest.

Some Variants of Naive Bayes

Multinomial Naive Bayes

- This is the one we have studied!
- It assumes features are frequencies.
- Hence, it works on text that has been Count Vectorized.
- (In practice, it also works on TF-IDF scores.)

Bernoulli Naive Bayes

- This assumes features are binary.
- Hence, it works on text that has been Count Vectorized with frequencies capped at 1.

Gaussian Naive Bayes

- This works on tabular datasets whose features are all numeric.
- But it does assume that the feature values within each class follow a Gaussian (Normal) distribution — a bell curve.
- E.g. features such as a person's height and weight might follow this distribution.
- In practice, it is robust enough to perform well even if the distributions are slightly skewed.
- We might make features more Gaussian-like by transforming them using, e.g. log, square root.