Foundations of Natural Language Processing
Lecture 10
Text Classification / Logistic Regression

Ivan Titov
(some slides from Alex Lascarides and Sharon Goldwater)

6 February 2024

Last time: Naive Bayes

- Given document x and set of categories C (say, spam/not-spam), we want to assign x to the most probable category $\hat{c}$.

$$\begin{aligned}\hat{c} &= \operatorname*{argmax}_{c \in C} P(c|x) \\ &= \operatorname*{argmax}_{c \in C} P(x|c)P(c)\end{aligned}$$

- The **naive Bayes assumption**: features are conditionally independent given the class.

$$P(f_1, f_2, \ldots f_n|c) \approx P(f_1|c)P(f_2|c) \ldots P(f_n|c)$$

- That is, the prob. of a word occurring depends **only** on the class.

Advantages of Naive Bayes

- Very easy to implement
- Very fast to train, and to classify new documents (good for huge datasets).
- Doesn't require as much training data as some other methods (good for small datasets).
- Usually works reasonably well
- This should be your baseline method for any classification task

Problems with Naive Bayes

- Naive Bayes assumption is naive!
- Consider categories TRAVEL, FINANCE, SPORT.
- Are the following features independent given the category?

beach, sun, ski, snow, pitch, palm, football, relax, ocean

Problems with Naive Bayes

- Naive Bayes assumption is naive!
- Consider categories TRAVEL, FINANCE, SPORT.
- Are the following features independent given the category?

beach, sun, ski, snow, pitch, palm, football, relax, ocean

  - No! Ex: Given TRAVEL, seeing beach makes sun more likely, but ski less likely.
  - Defining finer-grained categories might help (beach travel vs ski travel), but we don't usually want to.

Non-independent features

- Features are not usually independent given the class
- Adding multiple feature types (e.g., words and morphemes) often leads to even stronger correlations between features
- Accuracy of classifier can sometimes still be ok, but it will be highly **overconfident** in its decisions.
  - Ex: NB sees 5 features that all point to class 1, treats them as five independent sources of evidence.
  - Like asking 5 friends for an opinion when some got theirs from each other.

A less naive approach

- Although Naive Bayes is a good starting point, often we have enough training data for a better model (and not so much that slower performance is a problem).

- We may be able to get better performance using loads of features and a model that doesn't assume features are conditionally independent.

- Namely, a **Maximum Entropy model**.

MaxEnt classifiers

- Used widely in many different fields, under many different names
- Most commonly, **multinomial logistic regression**
  - *multinomial* if more than two possible classes
  - otherwise (or if lazy) just *logistic regression*
- Also called: log-linear model, one-layer neural network, single neuron classifier, etc ...
- The mathematical formulation here (and in the text) looks slightly different from standard presentations of mult. logistic regresssion, but is ultimately equivalent.

Naive Bayes vs MaxEnt

- Like Naive Bayes, MaxEnt assigns a document x to class $\hat{c}$, where
$$\hat{c} = \operatorname*{argmax}_{c \in C} P(c|x)$$

- Unlike Naive Bayes, we do not apply Bayes' Rule. Instead, we model $P(c|x)$ directly.

MaxEnt is a *discriminative* model

- It is trained to **discriminate** correct vs. incorrect values of c, given input x. That's all it can do.

- Naive Bayes can also **generate** data: sample a class from $P(c)$, then sample words from $P(x|c)$. So, we call it a **generative** model.

Discriminative models more broadly

- Trained to **discriminate** correct vs. wrong values of c, given input x.
- Need not be probabilistic.
- Examples: artificial neural networks, decision trees, nearest neighbor methods, support vector machines
- Here, we consider only one method: Maximum Entropy (MaxEnt) models, which *are* probabilistic.

Example: classify by topic

- Given a web page document, which topic does it belong to?
  - $\vec{x}$ are the words in the document, plus info about headers and links.
  - c is the latent class. Assume three possibilities:

$c =$	class
1	TRAVEL
2	SPORT
3	FINANCE

Feature functions

- Like Naive Bayes, MaxEnt models use **features** we think will be useful for classification.

- However, features are treated differently in the two models:
  - NB: features are **directly observed** (e.g., words in doc): no difference between features and data.
  - MaxEnt: we will use $\vec{x}$ to represent the observed data. Features are **functions** that depend on both observations $\vec{x}$ and class c.

This way of treating features in MaxEnt is standard in NLP; in ML it's often explained differently.

MaxEnt feature example

- If we have three classes, our features will always come in groups of three. For example, we could have three binary features:

$$
\begin{aligned}
f_1: &\quad \texttt{contains('ski')} \;\&\; c = 1 \\
f_2: &\quad \texttt{contains('ski')} \;\&\; c = 2 \\
f_3: &\quad \texttt{contains('ski')} \;\&\; c = 3
\end{aligned}
$$

  - training docs from class 1 that contain ski will have f_1 active;
  - training docs from class 2 that contain ski will have f_2 active;
  - etc.

- Each feature f_i has a real-valued **weight** w_i (learned in training).

Classification with MaxEnt

Choose the class that has highest probability according to

$$P(c|\vec{x}) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(\vec{x}, c)\right)$$

where the normalization constant $Z = \sum_{c'} \exp(\sum_i w_i f_i(\vec{x}, c'))$

- Inside brackets is just a dot product: $\vec{w} \cdot \vec{f}$.
- And $P(c|\vec{x})$ is a **monotonic function** of this dot product.
- So, we will end up choosing the class for which $\vec{w} \cdot \vec{f}$ is highest.

Classification example

$$
\begin{array}{lll}
f_1: & \texttt{contains('ski')}\ \&\ c=1 & w_1 = 1.2 \\
f_2: & \texttt{contains('ski')}\ \&\ c=2 & w_2 = 2.3 \\
f_3: & \texttt{contains('ski')}\ \&\ c=3 & w_3 = -0.5 \\
f_4: & \texttt{link\_to('expedia.com')}\ \&\ c=1 & w_4 = 4.6 \\
f_5: & \texttt{link\_to('expedia.com')}\ \&\ c=2 & w_5 = -0.2 \\
f_6: & \texttt{link\_to('expedia.com')}\ \&\ c=3 & w_6 = 0.5 \\
f_7: & \texttt{num\_links}\ \&\ c=1 & w_7 = 0.0 \\
f_8: & \texttt{num\_links}\ \&\ c=2 & w_8 = 0.2 \\
f_9: & \texttt{num\_links}\ \&\ c=3 & w_9 = -0.1
\end{array}
$$

- f_7, f_8, f_9 are **numeric** features that count outgoing links.

Classification example

- Suppose our test document contains ski and 6 outgoing links.
- We don't know c for this doc, so we try out each possible value.
  - Travel: $\sum_i w_i f_i(\vec{x}, c = 1) = 1.2 + (0.0)(6) = 1.2$.
  - Sport: $\sum_i w_i f_i(\vec{x}, c = 2) = 2.3 + (0.2)(6) = 3.5$.
  - Finance: $\sum_i w_i f_i(\vec{x}, c = 3) = -0.5 + (-0.1)(6) = -1.1$.
- We'd need to do further work to compute the probability of each class, but we know already that Sport will be the most probable.

Feature templates

- In practice, features are usually defined using **templates**

  `contains(`w`)` $\&$ c
  `header_contains(`w`)` $\&$ c
  `header_contains(`w`) & link_in_header` $\&$ c

  - instantiate with all possible words w and classes c
  - usually filter out features occurring very few times

- NLP tasks often have a few templates, but 1000s or 10000s of features

Training the model

- Given annotated data, choose weights that make the labels most probable under the model.

- That is, given items $x^{(1)} \dots x^{(N)}$ with labels $c^{(1)} \dots c^{(N)}$, choose

$$\hat{w} = \operatorname*{argmax}_{\vec{w}} \sum_j \log P(c^{(j)} | x^{(j)})$$

Training the model

- Given annotated data, choose weights that make the class labels most probable under the model.

- That is, given examples $x^{(1)} \ldots x^{(N)}$ with labels $c^{(1)} \ldots c^{(N)}$, choose

$$\hat{w} = \operatorname*{argmax}_{\vec{w}} \sum_j \log P(c^{(j)} | x^{(j)})$$

- called **conditional maximum likelihood estimation** (CMLE)

- Like MLE, CMLE will overfit, so we use tricks (**regularization**) to avoid that.

MaxEnt training: gradient descent

Random initialization (e.g., from a Gaussian distribution)

$$\vec{w} \leftarrow \text{Random}()$$

repeat

$$\vec{w} \leftarrow \vec{w} + \eta \cdot \nabla_w \sum_{j=1}^{N} \log P(c^{(j)} | x^{(j)})$$

until Converged()

Learning rate: a scalar regulating how much you update on every example

Common strategy: finish when the performance on the development set stops improving (or after a fixed number of iterations)

MaxEnt training: mini-batch gradient descent

Choosing a "**batch**": Indexes of a random subset of examples (e.g., choose 10 random examples)

$$
\begin{aligned}
&\vec{w} \leftarrow \text{Random}() \\
&\textbf{repeat} \\
&\quad B \leftarrow \text{RandomSubset}([1, \ldots, N]) \\
&\quad \vec{w} \leftarrow \vec{w} + \eta \cdot \nabla_w \textstyle\sum_{j \in B} \log P(c^{(j)} | x^{(j)}) \\
&\textbf{until } \text{Converged}()
\end{aligned}
$$

Sum only over examples in the current batch

How does the gradient look like?

$$P(c|\vec{x}) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(\vec{x}, c)\right)$$

$$Z = \sum_{c'} \exp\left(\sum_i w_i f_i(\vec{x}, c')\right)$$

Let us consider a single component of the gradient, corresponding to a feature

$$f_l : \quad \ldots \,\&\, c = k$$

E.g., the feature could be

$$f_l : \ \texttt{contains('ski')} \ \&\ c = k$$

$$\frac{d}{dw_l} \log P(c^{(j)}|\vec{x}^{(j)}) = ?$$

How does the gradient look like?

$$P(c|\vec{x}) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(\vec{x}, c)\right)$$

$$Z = \sum_{c'} \exp\left(\sum_i w_i f_i(\vec{x}, c')\right)$$

Let us consider a single component of the gradient, corresponding to a feature

$$f_l : \quad \ldots \,\&\, c = k$$

$$\frac{d}{dw_l} \log P(c^{(j)}|\vec{x}^{(j)}) =$$

$$= \frac{d}{dw_l} \log\left(\exp\left(\sum_i w_i f_i(\vec{x}^{(j)}, c^{(j)})\right)\right) - \frac{d \log Z}{dw_l}$$

How does the gradient look like?

$$P(c|\vec{x}) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(\vec{x}, c)\right)$$

$$Z = \sum_{c'} \exp\left(\sum_i w_i f_i(\vec{x}, c')\right)$$

Let us consider a single component of the gradient, corresponding to a feature

$$f_l : \quad \ldots \& \; c = k$$

$$\frac{d}{dw_l} \log P(c^{(j)}|\vec{x}^{(j)}) =$$

$$= \frac{d}{dw_l} \cancel{\log}\left(\cancel{\exp}\left(\sum_i w_i f_i(\vec{x}^{(j)}, c^{(j)})\right)\right) - \frac{d \log Z}{dw_l}$$

How does the gradient look like?

$$P(c|\vec{x}) = \frac{1}{Z} \exp\left(\sum_i w_i f_i(\vec{x}, c)\right)$$

$$Z = \sum_{c'} \exp\left(\sum_i w_i f_i(\vec{x}, c')\right)$$

Let us consider a single component of the gradient, corresponding to a feature

$$f_l : \ \ldots \& \ c = {\color{red}k}$$

$$\frac{d}{dw_l} \log P(c^{(j)}|\vec{x}^{(j)}) =$$

$$= \frac{d}{dw_l} \cancel{\log}\left(\cancel{\exp}\left(\sum_i w_i f_i(\vec{x}^{(j)}, c^{(j)})\right)\right) - \frac{d \log Z}{dw_l} = {\color{red}(\mathbf{I}) - (\mathbf{II})}$$

First term (I)

$$\mathbf{(I)} = \frac{d}{dw_l}\left(\sum_i w_i f_i(\vec{x}^{(j)}, c^{(j)})\right) = f_l(\vec{x}^{(j)}, c^{(j)})$$

First term (I)

$$(\mathbf{I}) = \frac{d}{dw_l}\left(\sum_i w_i f_i(\vec{x}^{(j)}, c^{(j)})\right) = f_l(\vec{x}^{(j)}, c^{(j)})$$

Recall that we consider the feature which is only active for the class k: $f_l : \ \ldots \& \ c = k$

First term (I)

$$(\mathbf{I}) = \frac{d}{dw_l}\left(\sum_i w_i f_i(\vec{x}^{(j)}, c^{(j)})\right) = f_l(\vec{x}^{(j)}, c^{(j)})$$

Recall that we consider the feature which is only active for the class k: $f_l : \quad \ldots \& \; c = k$

$$= [k = c^{(j)}] \cdot f_l(\vec{x}^{(j)}, k)$$

[.] is the Iverson bracket:

$$[S] \equiv \begin{cases} 0 & \text{if } S \text{ is false} \\ 1 & \text{if } S \text{ is true,} \end{cases}$$

Second term (II)

$$\textbf{(II)} = \frac{d \log Z}{dw_l} = \frac{d}{dw_l} \log\left(\sum_{c'} \exp\left(\sum_i w_i f_i(\vec{x}^{(j)}, c')\right)\right)$$

$$Z = \sum_{c'} \exp\left(\sum_i w_i f_i(\vec{x}, c')\right)$$

Second term (II)

$$\textbf{(II)} = \frac{d \log Z}{d w_l} = \frac{d}{d w_l} \log \left(\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right) \right)$$

$$= \frac{\frac{d}{d w_l} \left(\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right) \right)}{\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right)}$$

$$\frac{d}{dx} \log(x) = 1/x$$

Second term (II)

$$\mathbf{(II)} = \frac{d \log Z}{d w_l} = \frac{d}{d w_l} \log \left(\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right) \right)$$

$$= \frac{\frac{d}{d w_l} \left(\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right) \right)}{\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right)}$$

$$= \frac{\frac{d}{d w_l} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right)}{Z}$$

$$Z = \sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}, c') \right)$$

Second term (II)

$$(\mathbf{II}) = \frac{d \log Z}{d w_l} = \frac{d}{d w_l} \log \left(\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right) \right)$$

$$= \frac{\frac{d}{dw_l} \left(\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right) \right)}{\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right)}$$

$$= \frac{\frac{d}{dw_l} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right)}{Z}$$

$$= \frac{\exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right) \frac{d}{dw_l} \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right)}{Z}$$

$$\frac{d}{dx} \exp(x) = \exp(x)$$

Second term (II)

$$\textbf{(II)} = \frac{d \log Z}{d w_l} = \frac{d}{d w_l} \log \left(\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right) \right)$$

$$= \frac{\frac{d}{d w_l} \left(\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right) \right)}{\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right)}$$

$$= \frac{\frac{d}{d w_l} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right)}{Z}$$

$$= \frac{\exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right) \frac{d}{d w_l} \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right)}{Z}$$

$$= P(c = k | x^{(j)}) \cdot \frac{d}{d w_l} \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right)$$

$$P(c|\vec{x}) = \frac{1}{Z} \exp \left(\sum_i w_i f_i(\vec{x}, c) \right)$$

Second term (II)

$$(\mathbf{II}) = \frac{d \log Z}{d w_l} = \frac{d}{d w_l} \log \left(\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right) \right)$$

$$= \frac{\frac{d}{d w_l} \left(\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right) \right)}{\sum_{c'} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, c') \right)}$$

$$= \frac{\frac{d}{d w_l} \exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right)}{Z}$$

$$= \frac{\exp \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right) \frac{d}{d w_l} \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right)}{Z}$$

$$= P(c = k | x^{(j)}) \cdot \frac{d}{d w_l} \left(\sum_i w_i f_i(\vec{x}^{(j)}, k) \right)$$

$$= P(c = k | x^{(j)}) \cdot f_l(\vec{x}^{(j)}, k)$$

Expectation of the feature, under the model distribution

Bringing everything together

$$\frac{d}{dw_l} \log P(c^{(j)}|\vec{x}^{(j)}) = \mathbf{(I)} - \mathbf{(II)}$$

$$= [k = c^{(j)}] \cdot f_l(\vec{x}^{(j)}, k) - P(c = k|x^{(j)}) \cdot f_l(\vec{x}^{(j)}, k)$$

Bringing everything together

$$\frac{d}{dw_l} \log P(c^{(j)}|\vec{x}^{(j)}) = \textbf{(I)} - \textbf{(II)}$$

$$= [k = c^{(j)}] \cdot f_l(\vec{x}^{(j)}, k) - P(c = k|x^{(j)}) \cdot f_l(\vec{x}^{(j)}, k)$$

$$= \left([k = c^{(j)}] - P(c = k|x^{(j)})\right) f_l(\vec{x}^{(j)}, k)$$

Close to zero if the classifier confidently predicts the correct class

$$P(c = k|x^{(j)}) \approx \begin{cases} 1 & \text{if } k = c^{(j)} \\ 0 & \text{otherwise} \end{cases}$$

If the classifier is already confident, gradient is close to 0 and no learning is happening

Relation to Naive Bayes

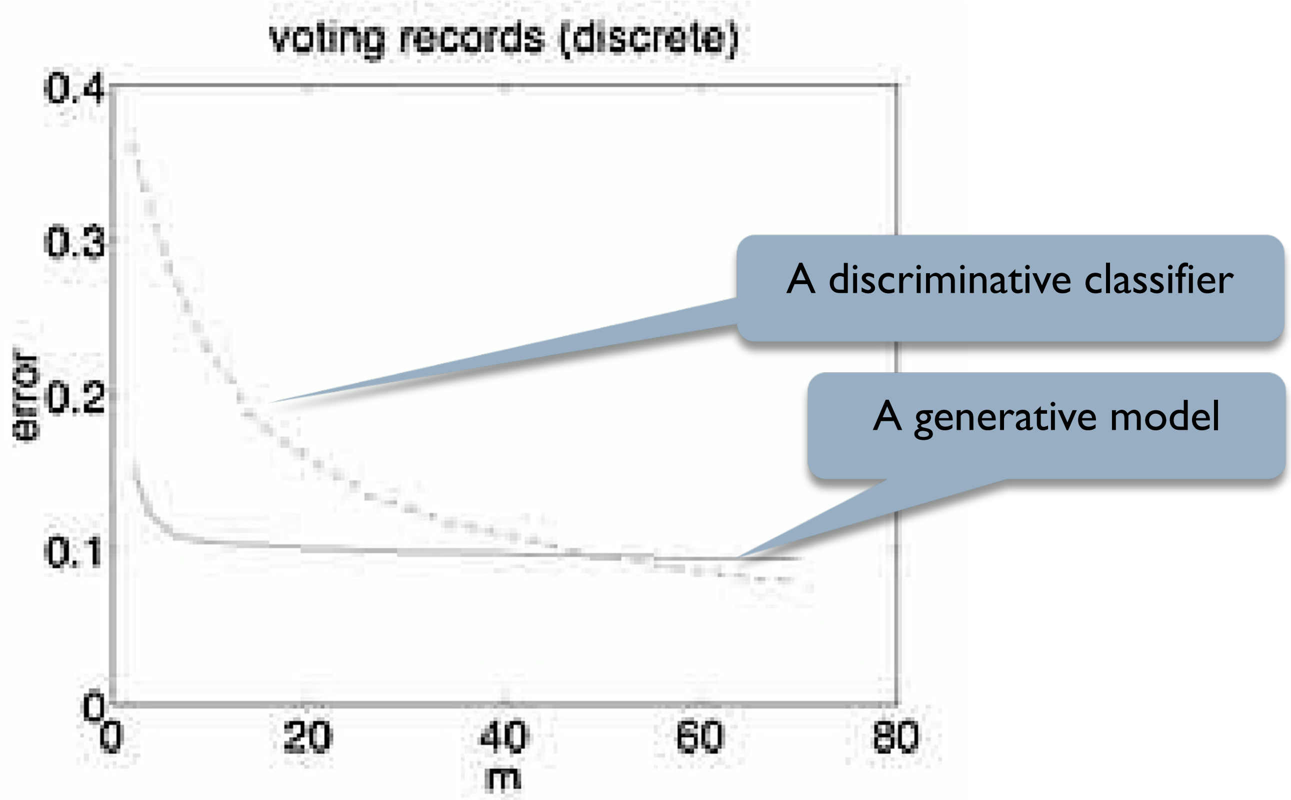
$$
\begin{array}{lll}
f_1: & \texttt{contains('ski')} \ \& \ c=1 & w_1 = \log \hat{P}(\texttt{'ski'}|c=1) \\
f_2: & \texttt{contains('ski')} \ \& \ c=2 & w_2 = \log \hat{P}(\texttt{'ski'}|c=2) \\
f_3: & \texttt{contains('ski')} \ \& \ c=3 & w_3 = \log \hat{P}(\texttt{'ski'}|c=3) \\
f_4: & \texttt{contains('beach')} \ \& \ c=1 & w_4 = \log \hat{P}(\texttt{'beach'}|c=1) \\
f_5: & \texttt{contains('beach')} \ \& \ c=2 & w_5 = \log \hat{P}(\texttt{'beach'}|c=2) \\
f_6: & \texttt{contains('beach')} \ \& \ c=3 & w_6 = \log \hat{P}(\texttt{'beach'}|c=3) \\
f_7: & c=1 & w_7 = \log \hat{P}(c=1) \\
f_8: & c=2 & w_8 = \log \hat{P}(c=2) \\
f_9: & c=3 & w_9 = \log \hat{P}(c=3)
\end{array}
$$

- Recall, Naive Bayes is also a linear classifier, and can be expressed in the same form

- Should the features be actually independent (will never happen), they would converge to the same solution as the amount of training data increases

NB vs MaxEnt dependence on dataset size

- Theoretical results: generative classifiers converge faster with training set size to their optimal error [Ng & Jordan, NeurIPS 2001]
- Empirical:



Predicting Democrat vs Republican, based on voting records

train examples

The downside to MaxEnt models

- Supervised MLE in generative models is easy: compute counts and normalize.
- Supervised CMLE in MaxEnt model not so easy
  - requires multiple iterations over the data to gradually improve weights (using gradient ascent).
  - each iteration computes $P(c^{(j)}|x^{(j)})$ for all j, and each possible $c^{(j)}$.
  - this can be time-consuming, especially if there are a large number of classes and/or thousands of features to extract from each training example.

Robustness: MaxEnt and Naive Bayes

- Imagine that in training there is one very frequent predictive feature
  - E.g., in training setiment data contained emoticons but not at test time
- The model can quickly learn to rely on this feature
  - model is confident on examples with emoticons
  - the gradient on these examples gets close to zero
  - the model does not learn other features
- In MaxEnt, **a feature weight will depend on the presence of other predictive features**
- Naive Bayes will rely on all features
  - The weight of a feature is not affected by how predictive other features are
- This makes NB more robust that (**basic**) MaxEnt when test data is (distributionally) different from train data

Summary

- Two methods for text classification: Naive Bayes, MaxEnt
- Make different independence assumptions, have different training requirements.
- Both are easily available in standard ML toolkits.
  - But you now also know how to implement them!
- Both require some work to figure out what features are good to use.
  - Later in the class, we will see how to alleviate the need for feature engineering