Probabilistic classifiers

• Bayes Decision Rule minimizes expected error:

\[ \hat{c} = \arg\max_{c \in C} P(c | x) \]

• We use a generative model \( P(x, c) \) plus Bayes’ Theorem:

\[ \hat{c} = \arg\max_{c \in C} P(c | x) \]
\[ = \arg\max_{c \in C} \frac{P(x | c) P(c)}{P(x)} \]
\[ = \arg\max_{c \in C} P(x | c) P(c) \]
\[ = \arg\max_{c \in C} P(x, c) \]
Naive Bayes classifiers

- We can split $P(x, c)$ into two parts: the class prior $P(c)$, and a likelihood $P(x|c)$.

- It’s easy to get reasonable estimates of $P(c)$ from training data, but not $P(x|c)$.

- Instead, we assume that the individual features in $x$ are independent, so:
  
  $$P(x|c) = \prod_i P(x_i|c)$$

- Now the decision rule becomes:
  
  $$\hat{c} = \underset{c \in C}{\arg\max} \prod_i P(x_i|c)$$
Naive Bayes classifiers

- Naive Bayes classifiers work well even when features aren’t independent

- But, the “naive Bayes” assumption is clearly wrong – can we do without it?

- If we know all the $P(x_i|c)$’s but not their dependencies, is it possible to construct $P(x|c)$?

- Yes, in fact, there are lots of ways to do it: the problem is *ill-posed*
Maximum Entropy

• This is a general problem: how do we pick a probability distribution given possibly incomplete information?

• Our probability estimates should reflect what we know and what we don’t know: ignorance is preferable to error

• Shannon’s entropy is a measure of ignorance

• Jaynes (1957): “The least informative probability distribution maximizes the entropy $S$ subject to known constraints.”
Principle of Insufficient Reason

- Remember Bernoulli’s *Principle of Insufficient Reason*: if we have $n$ outcomes and don’t know anything else, then say each outcome has a probability of $\frac{1}{n}$.

- Suppose we have a coin (with two sides). All we know is:

  $$P(h) + P(t) = 1$$

- The entropy $H$ is:

  $$H = -(P(h) \log P(h) + P(t) \log P(t))$$
  $$= -(P(h) \log P(h) + (1 - P(h)) \log(1 - P(h)))$$

- If $P(h) = 0.5$, then $H = \log_2 1 = 1 \text{ bit}$
Principle of Insufficient Reason
Why maximize entropy? Shannon and Jaynes show that other measures run into inconsistencies.

Another argument (what Jaynes calls the “Wallis derivation”) based on a procedure for ‘fairly’ constructing a distribution given some constraints.

Divide the available probability mass into $n$ quanta, each of magnitude $\delta = \frac{1}{n}$, and randomly assign them to the $m$ possible outcomes.

If outcome $i$ gets $n_i$ quanta, then we say its probability is $p_i = n_i \delta = \frac{n_i}{n}$.

If the resulting distribution fits the known constraints, we’re done. Otherwise, we reject it and try again.
Wallis derivation

- For this to give good results, \( n \) has to be much larger than \( m \), and we might need a lot of attempts before we get a distribution that fits the constraints.

- So, instead, let's find the distribution which is most likely to come up.

- The probability of any particular assignment is given by the multinomial distribution:

\[
P(n_1, \ldots, n_m) = \binom{n}{n_1, \ldots, n_m} m^{-n} = \frac{n!}{n_1! \cdots n_m! m^{-n}}
\]

- So, the assignment which we are most likely to come up with using this fair procedure is the one that maximizes:

\[
W = \frac{n!}{n_1! \cdots n_m!}
\]
• Instead of maximizing $W$, we could equivalently maximize a monotonic increasing function of $W$, like, oh, say, $\frac{1}{n} \log W$:

\[
\frac{1}{n} \log W = \frac{1}{n} \log \frac{n!}{n_1! \cdots n_m!} = \frac{1}{n} \log \frac{n!}{np_1! \cdots np_m!} = \frac{1}{n} (\log n! - \sum_i \log np_i!)
\]
Now, we can bring in Stirling’s approximation:

\[
\log n! = \sum_{k=1}^{n} \log k \\
\approx \int_{1}^{n} \log x \, dx \\
= n \log n - n + 1 \\
\approx n \log n - n
\]
Wallis derivation

• Put them together and we get:

\[
\frac{1}{n} \log W = \frac{1}{n} \left( n \log n - n - \sum_{i} (np_i \log np_i - np_i) \right)
\]

\[
= \log n - \sum_{i} p_i \log np_i
\]

\[
= \log n - \left( \sum_{i} p_i \log n + \sum_{i} p_i \log p_i \right)
\]

\[
= \log n - \left( \sum_{i} p_i \log n + \sum_{i} p_i \log p_i \right)
\]

\[
= (1 - \sum_{i} p_i) \log n - \sum_{i} p_i \log p_i
\]

\[
= - \sum_{i} p_i \log p_i
\]
A simple example

- Suppose a fast-food restaurant sells $1.00 burgers and $2.00 chicken sandwiches. Customers, on average, pay $1.75 for lunch. What’s the probability that someone ordered a burger?

- We know:

\[ P(b) + P(c) = 1 \]

\[ ($1.00 \times P(b)) + ($2.00 \times P(c)) = $1.75 \]

- So, we can conclude:

\[ ($1.00 \times P(b)) + ($2.00 \times (1 - P(b))) = $1.75 \]

\[ P(b) = $0.25 \]
A simple example

- Now suppose this fast-food restaurant also sells $3.00 fish sandwiches. If customers pay $1.75 for lunch on average, what’s the probability that someone ordered a burger?

- We know:

  \[ P(b) + P(c) + P(f) = 1 \]
  \[ ($1.00 \times P(b)) + ($2.00 \times P(c)) + ($3.00 \times P(f)) = $1.75 \]

- Now we have three unknown probabilities and only two constraints.

- Out of the many possible ways of assigning probabilities, we want to find the one that maximizes the entropy.
A simple example

- We can use the constraints to eliminate two of the unknowns:

\[ P(c) = -2P(b) + 1.25 \]
\[ P(f) = P(b) - 0.25 \]

- Now we can apply MaxEnt:

\[
H = -P(b) \log P(b) -
(\sqrt[2]{2} \cdot P(b) + 1.25) \log\left(\sqrt[2]{2} \cdot P(b) + 1.25\right) -
(P(b) - 0.25) \log(P(b) - 0.25)
\]
A simple example

![Graph showing a curve with labels $P(f)$ and $H$.]
A simple example

To find the value of $P(b)$ which maximizes $H$, we take the derivative of $H$:

$$\frac{d}{dP(b)} H = -\log(P(b)) + 2\log(-2P(b) + 1.25) - \log(P(b) - 0.25)$$

and solve:

$$-\log(P(b)) + 2\log(-2P(b) + 1.25) - \log(P(b) - 0.25) = 0$$

$$P(b) = 0.466$$
Maximum entropy

- Simple problems can be solved analytically, but to replace naive Bayes we need a more general solution.

- We have our usual feature vector $x$, and we know the value of feature $x_i$ for every instance in the training set.

- From this, we can estimate the expected value $\hat{E}[x_i]$.

- This gives us a set of constraints:

  $\sum P(x) = 1$

  for each $x_i$: $\sum P(x) x_i = \hat{E}[x_i]$

- Of the distributions which satisfy these constraints, we need to find the one that maximizes the entropy $H(P)$. 
Constrained optimization

- This is a *constrained optimization* problem: maximize a function given a set of constraints.

- First, we restate the constraints:

  \[
  0 = \sum P(x) x_i - \hat{E}[x_i]
  \]

  \[
  0 = \sum P(x) - 1
  \]

- Next, we introduce the *Lagrangian function*:

  \[
  \mathcal{L}(P, \lambda, \gamma) = - \sum_x P(x) \log P(x) - \sum_i \lambda_i \left( \sum_x P(x) x_i - \hat{E}[x_i] \right) - \\
  \gamma \left( \sum_x P(x) - 1 \right)
  \]
Constrained optimization

- This now gives us an *unconstrained optimization* problem, which we can solve by finding the $P$ where:

$$\nabla \mathcal{L}(P, \lambda, \gamma) = 0$$

- So, we start here:

$$0 = \frac{\partial}{\partial P} \mathcal{L}(P, \lambda, \gamma)$$

$$= -(1 + \log P(x)) - \sum_i \lambda_i x_i - \gamma$$

$$\log P(x) = -\gamma - 1 - \sum_i \lambda_i x_i$$

$$P(x) = \exp (\gamma - 1) \exp \left( \sum_i \lambda_i x_i \right)$$
Constrained optimization

- Recall that:

\[
\sum_x P(x) = 1
\]

\[
= \sum_x \exp (\gamma - 1) \exp \left( \sum_i \lambda_i x_i(x) \right)
\]

\[
\exp (\gamma - 1) = \left( \sum_x \exp \left( \sum_i \lambda_i x_i \right) \right)^{-1}
\]
Constrained optimization

• Finally, substituting in \( P(x) \), we get:

\[
P(x) = \frac{1}{Z} \exp \left( \sum \lambda_i x_i \right)
\]

\[
Z = \sum_{x} \exp \left( \sum \lambda_i x_i \right)
\]

• Parameters \( \lambda_i \) are chosen so that:

\[
\sum_{x} P(x) x_i = \hat{E}[x_i]
\]

• \( Z \) is sometimes called the *partition function*
MaxEnt classifiers

- The model can be constrained by anything whose expected value is interesting (e.g., presence of a word, normalized frequency of a word)

- To apply this to classification, we need the joint distribution $P(x, c)$. So, features need to be a conjunction of a contextual predicate and a class

- We can account for the class prior $P(c)$ by including the class itself as a feature

- Feature selection can be done in the usual way.

- Setting all features for a baseline class to zero will further reduce the number of features
MaxEnt classifiers

- To build a MaxEnt classifier, we need to construct a function $f_i$ from documents to features, and then estimate $\lambda_i$ for each feature $i$ (more on that later).

- Then, to find the probability of a new document $d$ having a class label $c$, we evaluate:

$$P(d, c) = \frac{\exp \sum_i \lambda_i f_i(d, c)}{\sum_{d,c} \exp \sum_i \lambda_i f_i(d, c)}$$

- Now we have a problem: the sum in the denominator ranges over all possible documents and classes.

- One option is Monte Carlo simulation: randomly generate lots of documents according to our distribution and use them to estimate $\mathcal{Z}$. 

MaxEnt classifiers

- Instead, we can use our training data to compute an ‘empirical’ document distribution $\tilde{P}(d)$.

- Instead of these constraints:

$$\sum_d P(d, c) f_i(d, c) = \sum_d \tilde{P}(d, c) f_i(d, c)$$

we can use these constraints:

$$\sum_d \tilde{P}(d) P(c|d) f_i(d, c) = \sum_d \tilde{P}(d, c) f_i(d, c)$$

- This gives us a conditional maximum entropy model:

$$P(c|d) = \frac{\exp \sum_i \lambda_i f_i(d, c)}{\sum_c \exp \sum_i \lambda_i f_i(d, c)}$$
MaxEnt classifiers

- If we are only interested in classification, then for each document we only need to find:

\[
\hat{c} = \operatorname{arg\,max}_{c \in C} \sum_{i} \lambda_i f_i(d, c)
\]

- This (obviously) gives us a linear decision boundary

- Since we’re not summing log probabilities, there’s no clear bias for longer or shorter documents

- Also known as log-linear, Gibbs, exponential, and multinomial logit models

- Other constraints yield different distributions