CSE 5523: Lecture Notes 3
Probability Models

Contents

3.1 Full joint models and sparse data problems

We can estimate probablities for new atomic events based on frequencies of these events in training (assume a probability space $\langle O, 2^O, \mathsf{P}\rangle$ and frequency space $\langle O, 2^O, \mathsf{F}\rangle$ with $O = X_1 \times X_2 \times \cdots \times X_V$):

$$\mathsf{P}(x_1, x_2, \ldots, x_V) \overset{\text{def}}{=} \frac{\mathsf{F}(x_1, x_2, \ldots, x_V)}{\mathsf{F}(O)}$$

This is called a **full joint distribution**.

But if there are too many outcomes for the training data, full joints can make arbitrary predictions.

For example, consider this (junk/not-junk) email classifier algorithm:

1. Memorize a set of training emails (random variables are for first word, second word, etc.).
2. Compare each test email to the training set (estimate probability of atomic event).
   (a) If you find an exact match, report the class from the matching training example.
   (b) If you find no exact match, call it whatever is most common in training data (not-junk).

This will very rarely find an exact match, and end up classifying virtually nothing as junk.

This is called a **sparse data problem**. It shows the importance of *generalizing* from data.

3.2 Marginals and conditionals

One way to make better use of scarse data is to consolidate training instances across conditions.

We can do this by marginalizing or summing out certain variables that don't matter as much.

We can define **marginal** distributions from the Kolmogorov axioms:

$$\mathsf{P}(x) = \sum_{y \in Y} \mathsf{P}(x, y)$$

We can then define **conditional** distributions based on these marginals:

$$\mathsf{P}(y \mid x) = \frac{\mathsf{P}(x, y)}{\mathsf{P}(x)}$$

3.3 Factored models and independence assumptions

We can consolidate data by first factoring our model, using the definition of conditional probability (you can validate this by cross-cancelling the numerators and denominators):

$$\begin{aligned}\mathsf{P}(x_1, x_2, \ldots, x_{V-1}, x_V) &= \mathsf{P}(x_1) \cdot \underbrace{\frac{\mathsf{P}(x_1, x_2)}{\mathsf{P}(x_1)}}_{} \cdot \underbrace{\frac{\mathsf{P}(x_1, x_2, x_3)}{\mathsf{P}(x_1, x_2)}}_{} \cdots \underbrace{\frac{\mathsf{P}(x_1, x_2, \ldots, x_{V-1}, x_V)}{\mathsf{P}(x_1, x_2, \ldots, x_{V-1})}}_{} \\ &= \mathsf{P}(x_1) \cdot \mathsf{P}(x_2 \mid x_1) \cdot \mathsf{P}(x_3 \mid x_1, x_2) \cdots \mathsf{P}(x_V \mid x_1, x_2, \ldots, x_{V-1})\end{aligned}$$

This is called a **chain rule decomposition**.

If we then think some variables don't influence others much, we can remove them from conditions:

$$\mathsf{P}(x \mid y, z, \ldots) \overset{\text{def}}{=} \mathsf{P}(x \mid y, \ldots)$$

These re-definitions are called **independence assumptions**.

3.4 Graphical representation

Independence assumptions in models are often visualized graphically:

- **circles** for random variables
- **arrows** for conditional dependencies (from conditioned-on to modeled variables)

So every conditional probability term is associated with a circle with a set of arrows stuck into it (one impinging arrow for each conditioned-on variable):

$$\mathsf{P}(x_1) \cdot \mathsf{P}(x_2 \mid x_1)$$

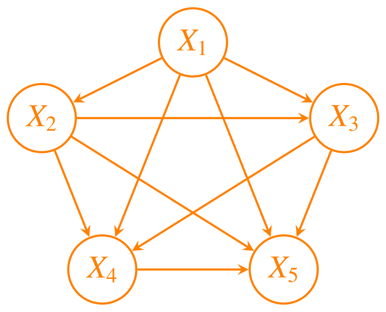
$$\begin{tikzcd} X_1 \arrow[r] & X_2 \end{tikzcd}$$

(Models that are drawn this way are also called Bayes Nets.)

For example, here is a chain rule decomposition with no independence assumptions:

$$\mathsf{P}(x_1, x_2, x_3, x_4, x_5) = \mathsf{P}(x_1) \cdot \mathsf{P}(x_2 \mid x_1) \cdot \mathsf{P}(x_3 \mid x_1, x_2) \cdot \mathsf{P}(x_4 \mid x_1, x_2, x_3) \cdot \mathsf{P}(x_5 \mid x_1, x_2, x_3, x_4)$$

It makes this evil pentagram, where X_5 depends on (receives arrows from) every other variable:



As such, the model for X_5 will have $|X_1| \cdot |X_2| \cdot |X_3| \cdot |X_4| \cdot |X_5|$ parameters — same as the full joint!

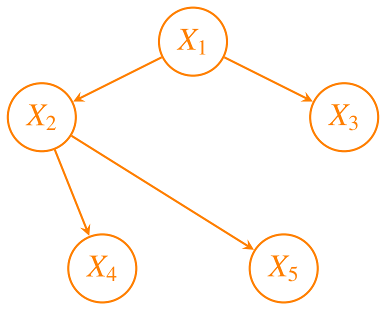
In contrast, if we add the following independence assumptions:

$$\begin{aligned} \mathsf{P}(x_3 \,|\, x_1, x_2) &\overset{\text{def}}{=} \mathsf{P}(x_3 \,|\, x_1) \\ \mathsf{P}(x_4 \,|\, x_1, x_2, x_3) &\overset{\text{def}}{=} \mathsf{P}(x_4 \,|\, x_2) \\ \mathsf{P}(x_5 \,|\, x_1, x_2, x_3, x_4) &\overset{\text{def}}{=} \mathsf{P}(x_5 \,|\, x_2) \end{aligned}$$

we get a shorter equation:

$$\mathsf{P}(x_1, x_2, x_3, x_4, x_5) \overset{\text{def}}{=} \mathsf{P}(x_1) \cdot \mathsf{P}(x_2 \,|\, x_1) \cdot \mathsf{P}(x_3 \,|\, x_1) \cdot \mathsf{P}(x_4 \,|\, x_2) \cdot \mathsf{P}(x_5 \,|\, x_2)$$

which makes a much simpler graph:



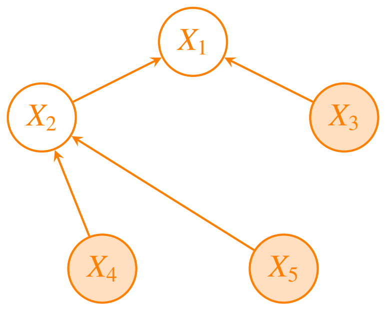
Now X_5 only has $|X_2| \cdot |X_5|$ parameters!

3.5 Generative vs discriminative models

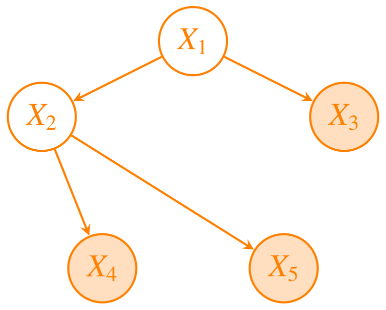
Random variables that are observed at test time are typically shaded in graphical representations.

These are called **observations**; the others are called **hidden** or **latent** variables.

When these observed variables are conditioned on, the model is called **discriminative**:



When observed variables are *not* conditioned on, the model is called **generative**:



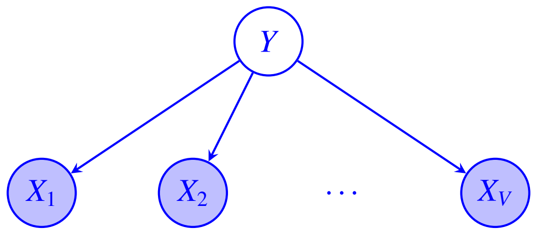
3.6 Naive Bayes model

One common model with a lot of independence assumptions is the Naive Bayes model.

The Naive Bayes model sets a hidden random variable as source and conditions observations on it (assume a probability space $\langle O, 2^O, \mathsf{P}\rangle$ with $O = Y \times X_1 \times X_2 \times \cdots \times X_V$):

$$\begin{aligned}\mathsf{P}(y, x_1, x_2, \ldots, x_V) &\overset{\text{def}}{=} \mathsf{P}(y) \cdot \mathsf{P}(x_1 \,|\, y) \cdot \mathsf{P}(x_2 \,|\, y) \cdot \cdots \cdot \mathsf{P}(x_V \,|\, y) \\ &= \mathsf{P}(y) \cdot \prod_v \mathsf{P}(x_v \,|\, y)\end{aligned}$$

Graphically it looks like this:



Independence assumptions mean observed variables train independently given hidden variable.

This strengthens data support for each parameter estimate.

3.7 Naive Bayes implementation

If you have python 3 and pandas installed on your machine, you can run the below classifier.

Paste this into ‘train.csv’ (the first row is interpreted as the column labels):

```
fruit,shape,color
apple,short,red
apple,tall,red
apple,short,green
pear,tall,green
pear,tall,green
```

Paste this into ‘test.csv’ (the first row is interpreted as the column labels):

```
shape,color
short,red
```

Paste this into ‘naivebayes.py’ (you may have to fix the single quotes):

```
import pandas

## Training...

YX = pandas.read_csv('train.csv')

F = { }
for f in YX['fruit'].unique():
  F[f] = len(YX[ YX['fruit']==f ]) / len(YX)

CgivF = { }
for f in YX['fruit'].unique():
  for c in YX['color'].unique():
    CgivF[c,f] = len(YX[ (YX['color']==c) & (YX['fruit']==f) ]) / len(YX[ YX['fruit']==f ])

SgivF = { }
for f in YX['fruit'].unique():
  for s in YX['shape'].unique():
    SgivF[s,f] = len(YX[ (YX['shape']==s) & (YX['fruit']==f) ]) / len(YX[ YX['fruit']==f ])

## Testing...

X = pandas.read_csv('test.csv')

for i,(s,c) in X.iterrows():
  for f in F:
    print( s + ' ' + c + ' ' + f + ' : ' + str(F[f] * CgivF[c,f] * SgivF[s,f]) )
```

Run ‘python naivebayes.py’; you should see this output:

```
short red pear : 0.0
short red apple : 0.26666666666666666
```