ENGR 212: Programming Practice

Week 12

İSTANBUL ŞEHİR UNIVERSITY

Calculating probabilities

- We have counts for how often email messages appear in each category (after training).
- The probability that a word is in a particular category C:

$$= \frac{\text{\# of times “word” appears in a document in C}}{\text{the total number of documents in C}}$$

Calculating probabilities

```
def fprob(self,f,cat):
    if self.catcount(cat) == 0:
        return 0
    # The total number of times this feature appeared in
    # this category divided by the total number of items
    # in this category
    return self.fcount(f,cat)/self.catcount(cat)
```

Conditional probability

- This is called and usually written as Pr(A | B) and read as "the probability of A given B."
- If the word "quick" appears in 2 out of a total of 3 documents classified as good,

  then:

  there's a probability of Pr(quick | good)=0.666 that a **good document** will <u>contain that word</u>.

Conditional probability - Example run

```
>>>import docclass
>>>cl=docclass.classifier(docclass.getwords)
>>>docclass.sampletrain(cl)
>>>cl.fprob('quick','good')
```

Zero counts

- In the sample training data, the word “online” only appears in one document and is classified as bad.
- Since the word “online” is in one bad document and no good ones, the probability that it will appear in the good category is now 0.
- This is a bit extreme, since “online” might be a perfectly neutral word that just happens to appear first in a bad document.

Creating a Classifier – Sample Train method

```
def sampletrain(cl):
    cl.train('Nobody owns the water.','good')
    cl.train('the quick rabbit jumps fences','good')
    cl.train('buy pharmaceuticals now online','bad')
    cl.train('make quick money at the online casino','bad')
    cl.train('the quick brown fox jumps','good')
```

Calculating probabilities

$$= \frac{w \times prob_{init} + c \times prob_{est}}{w + c}$$

Calculating probabilities

- In the "online" example, the weighted probability for the word "online" starts at 0.5 for all categories.
- After the classifier is trained with one bad document and finds that "online" fits into the bad category, its probability becomes 0.75 (not 1) for bad and 0.25 for good (not 0).

  (weight * $prob_{init}$ + count * $prob_{est}$) / (count + weight)
  = (1*0.5+1*1.0)/(1.0 + 1.0)
  = 0.75
  where weight = 1

- What would be the prob. after training with 4 bad documents all which contain the word "online"?

Calculating probabilities

```
def weightedprob(self,f,cat,prf,weight=1.0,ap=0.5):
     # Calculate current probability
     basicprob = prf(f,cat)

     # Count the number of times this feature has
     # appeared in all categories
     totals = sum([self.fcount(f,c) for c in
                                    self.categories()])

     # Calculate the weighted average
     bp = ((weight * ap) + (totals * basicprob)) /
          (weight + totals)
     return bp
```

Calculating probabilities

```
>>> import docclass
>>> cl=docclass.classifier(docclass.getwords)
>>> docclass.sampletrain(cl)
>>> print cl.weightedprob(online','good',cl.fprob)
>>> docclass.sampletrain(cl)
>>> print cl.weightedprob(‘online','good',cl.fprob)
```

Zero counts

- The assumed probability of 0.5 was chosen simply because it is halfway between 0 and 1.
- Can we use probabilities from other people's already-trained spam filters as the assumed probabilities?
  - The filter will get personalized more and more with new training data.
  - The filter is better able to handle words that it has come across very infrequently.

Combining probabilities

- We know the probability of a document in a category containing a particular word.
- We need a way to combine the individual word probabilities to get the probability that an entire document belongs in a given category.

Naive classifier: we assume that the probabilities being combined are independent of each other.

Naive Classifier

- **Assumption**: the probability of one word in the document being in a specific category is unrelated to the probability of the other words being in that category.
- This is actually a false assumption!
    - Documents containing the word “casino” are much more likely to contain the word “money” than documents containing “Python programming”.

Naive Classifier

- You *can't actually use the probability* created by the naïve Bayesian classifier as the actual probability that a document belongs in a category.
- However, you can **compare** the results for different categories and see which one has the highest probability.

Naive Classifier

- Suppose that:
  the word "Python" appears in 20 percent of your ***bad*** *documents: Pr(Python | Bad) = 0.2*

  the word "casino" appears in 80 percent of your ***bad*** *documents: Pr(Casino | Bad) = 0.8*
- The independent probability of "Python" and "casino" appearing together in a ***bad*** document:

  Pr(Python & Casino | Bad) = $0.8 \times 0.2 = 0.16$

Naive Classifier

```
class naivebayes(classifier):
  def docprob(self,item,cat):
    features=self.getfeatures(item)

    # Multiply the probabilities of all the features together
    p=1
    for f in features: p*=self.weightedprob(f,cat,self.fprob)
    return p
```

Naive Classifier

- We know how to calculate Pr(Document | Category).
- In order to classify documents, we need **Pr(Category | Document)**
- In other words, given a specific document, what's the probability that it fits into this category?
- A British mathematician named Thomas Bayes figured out how to do this about 250 years ago.

Bayes' Theorem

- Pr(A | B) = Pr(B | A) x Pr(A) / Pr(B)
- Therefore,

$$\Pr(\text{Category} \mid \text{Document}) = \frac{\Pr(\text{Document} \mid \text{Category}) \times P(\text{Category})}{\Pr(\text{Document})}$$

Bayes' Theorem

- Pr(Category): the number of documents in the category divided by the total number of documents.
- Pr(Document | Category) → docprob(…)
  - Pr(Python & Casino | Bad) = 0.8 × 0.2 = 0.16
- Pr(Document) is independent of category. And, it will only scale the results by the exact same amount. So we will ignore this term.
  - <u>Remember that we are interested in **ranking** class probabilities rather than using their actual numeric values.</u>

Naive Classifier

```
def prob(self,item,cat):
  catprob=self.catcount(cat)/self.totalcount( )
  docprob=self.docprob(item,cat)
  return docprob*catprob
```

Naive Classifier

```
>>> import docclass
>>> cl = docclass.naivebayes(docclass.getwords)
>>> docclass.sampletrain(cl)
>>> print cl.prob('quick rabbit','good')
>>> print cl.prob('quick rabbit','bad')
```

Assignment to a class

- How to decide in which category a new item belongs?
- Calculate the probability for each category, and choose the category with the best probability.
- For some applications, a marginally high probability may be enough to determine the class.
- For other applications, you have to be overly confident for making any assignment.

Being overly confident?

- For spam filtering:
  - The probability **for bad** would have to be 3 times higher than the probability for good.
  - The threshold **for good** could be set to 1, so anything would be good if the probability were at all better than for the bad category.
  - Any message where the probability for bad is higher, but not 3 times higher, would be classified as **unknown**.

Choosing a category

```
def __init__(self,getfeatures):
  classifier.__init__(self,getfeatures)
  self.thresholds={}

def setthreshold(self,cat,t):
  self.thresholds[cat]=t

def getthreshold(self,cat):
  if cat not in self.thresholds: return 1.0
  return self.thresholds[cat]
```

Classify method

```
def classify(self,item,default=None):
  probs={}
  # Find the category with the highest probability
  max=0.0
  for cat in self.categories( ):
    probs[cat]=self.prob(item,cat)
    if probs[cat]>max:
      max=probs[cat]
      best=cat

  # Make sure the probability exceeds threshold*next best
  for cat in probs:
    if cat==best: continue
    if probs[cat]*self.getthreshold(best)>probs[best]: return default
  return best
```

Testing Naïve Bayes Classifier

```
>>> import docclass
>>> cl=docclass.naivebayes(docclass.getwords)
>>> docclass.sampletrain(cl)
>>> print cl.classify('quick rabbit',default='unknown')
>>> print cl.classify('quick money',default='unknown')
>>> cl.setthreshold('bad',3.0)
>>> print cl.classify('quick money',default='unknown')
```