ENGR 102 PROGRAMMING PRACTICE

WEEK 13

Document Filtering

Document Filtering

- Your email address in the wrong hands:

  lots of unnecessary and unsolicited email messages!!!

SPAM SPAM SPAM SPAM SPAM SPAM SPAM SPAM SPAM SPAM SPAM SPAM ...

- A well-known application of document filtering is the elimination of spam.

Classification

- The algorithms are not specific to dealing with spam.
- The general problem of learning to recognize whether a document belongs in one category or another.
- ***App1***: Automatically dividing your inbox into social and work-related email, based on the contents of the messages.
- ***App2***: Identifying email messages that request information and automatically forwarding them to the most competent person to answer them.

Filtering Spam

- Early attempts to filter spam: rule-based classifiers.
  - e.g., overuse of capital letters, ...

- Spammers learned all the rules and stopped exhibiting the obvious behaviors to get around the filters.

- “What is spam”? Subjective!

- How about tailor-fitting?
  You teach me what is spam email and what isn’t spam email, and I learn for you to automate the task.

Documents and Words

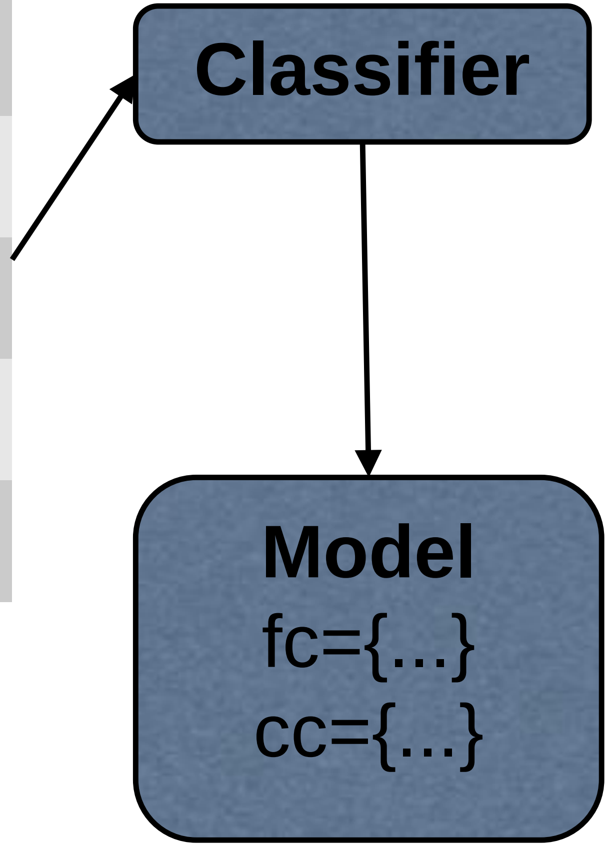
- The classifier needs features to use for classifying different items.
- Some words are more likely to appear in spam than in not-spam?
- Use words in the document as features.
- Not just individual words, however; word pairs or phrases or anything else that can be classified as absent or present in a particular document.

Extracting Features from Text

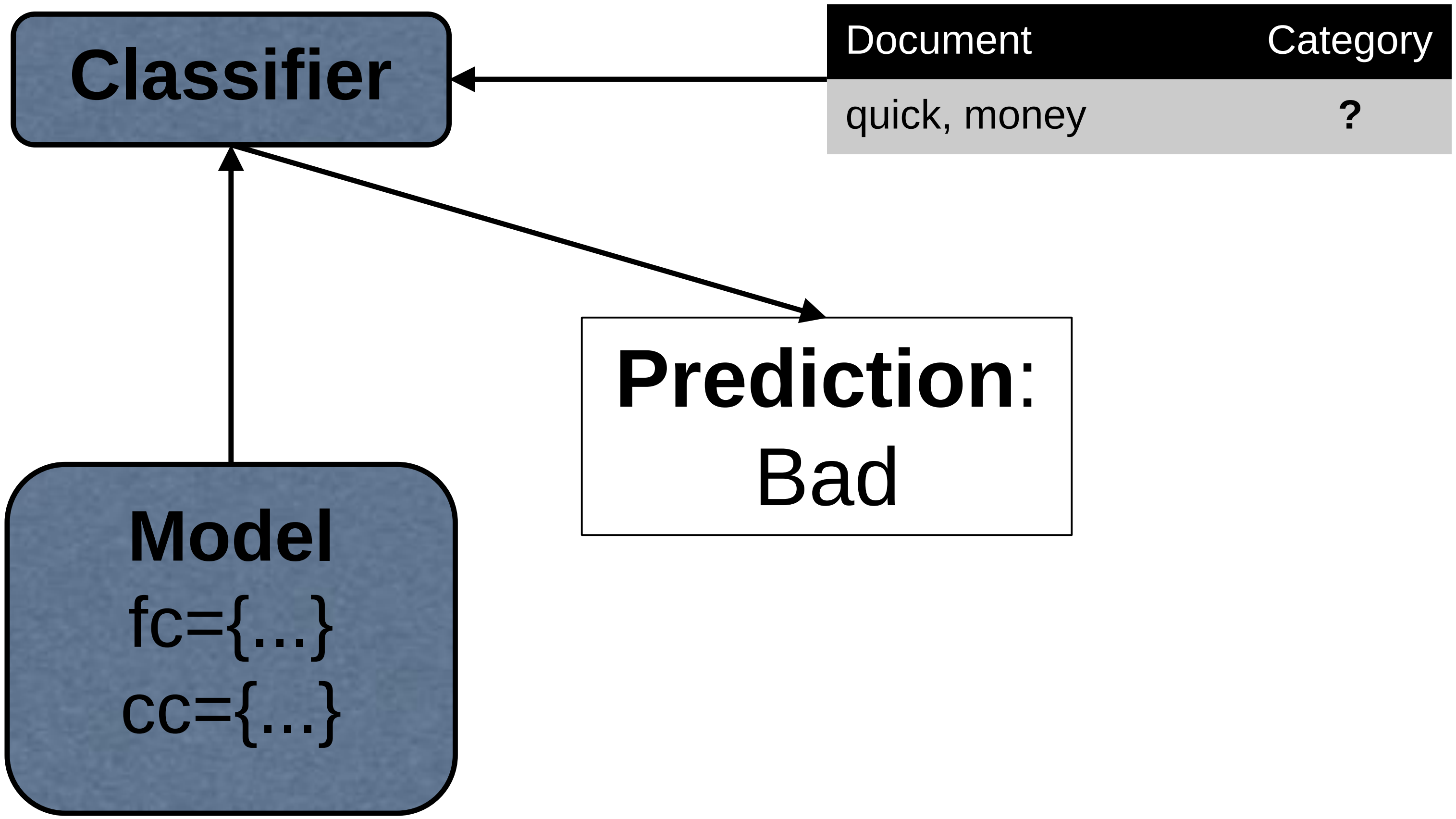
```
def getwords(doc):
    # equivalent to [^a-zA-Z0-9_]
    splitter = re.compile(r'\W+')

    # Split the words by non-alpha characters
    words = [s.lower() for s in splitter.split(doc)
               if len(s) > 2 and len(s) < 20]

    # Return the unique set of words only
    return dict([(w,1) for w in words])
```

Training

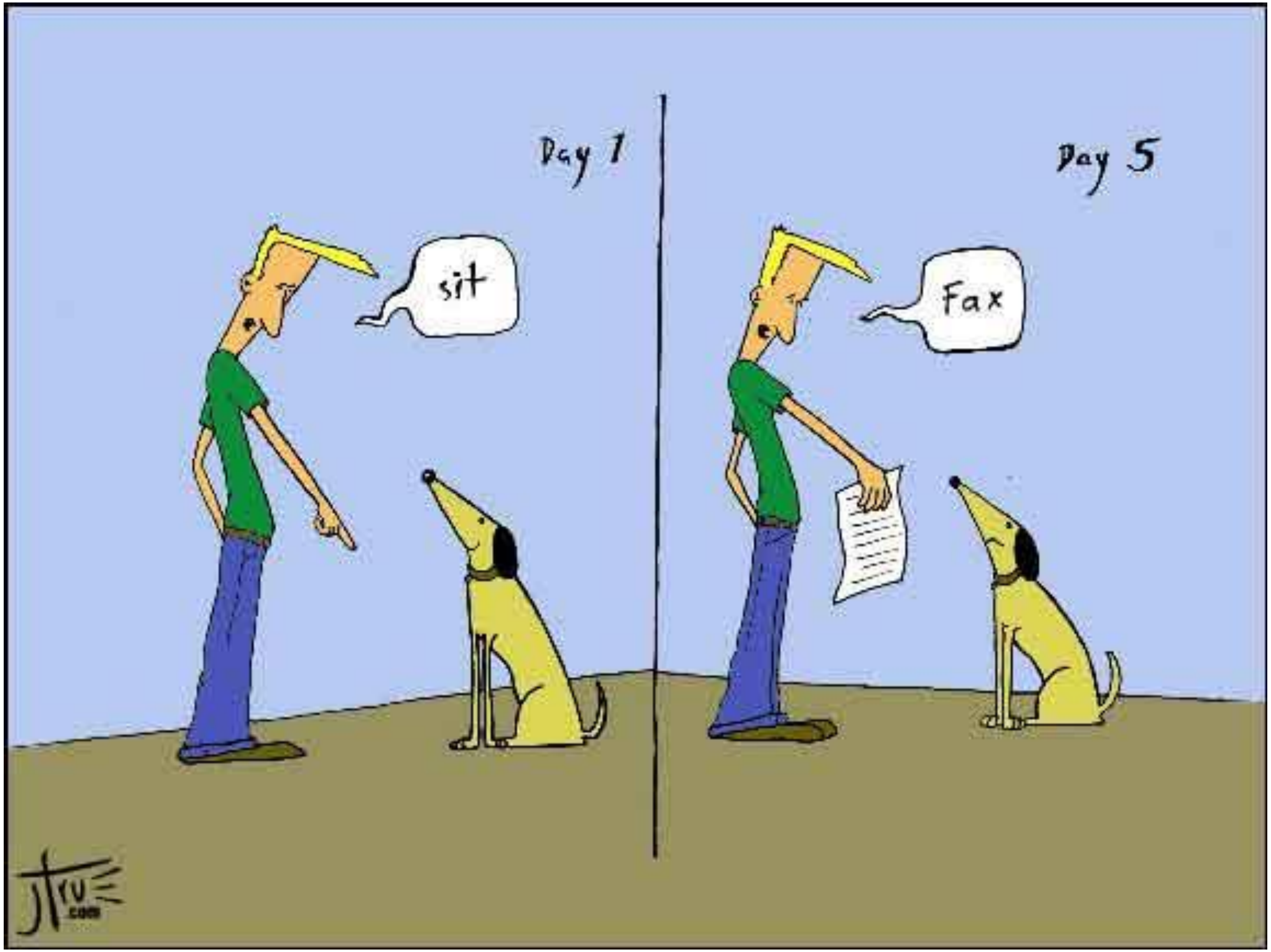
Document	Category
nobody, owns, the, water	good
the, quick, rabbit, jumps, fences	good
buy, pharmaceuticals, now	bad
make, quick, money, at, the, online, casino	bad
the, quick, brown, fox, jumps	good



Testing / Prediction



Training the Classifier



Training the Classifier

- The more examples the classifier is fed with, the better the classifier will get at making predictions.
    - example: a document and its classification
- The classifier starts off very uncertain and increase in certainty as it “learns”
    - which features are important for making a distinction.

Creating a classifier

```
class Classifier:

    def __init__(self, getfeatures):

        # Counts of feature/category combinations
        self.fc = {}

        # Counts of documents in each category
        self.cc = {}

        self.getfeatures = getfeatures
```

Training the Classifier
Model Elements

fc:

```
{
        'python': {'bad': 0, 'good': 6},
        'the':    {'bad': 3, 'good': 3}
 }
```

cc:

```
{
        'good': 4,
        'bad':  5
}
```

Creating classifier

```
# Increase the count of a feature/category pair
def incf(self,f,cat):
    self.fc.setdefault(f,{})
    self.fc[f].setdefault(cat,0)
    self.fc[f][cat] += 1

# Increase the count of a category
def incc(self,cat):
    self.cc.setdefault(cat,0)
    self.cc[cat] += 1

# The number of times a feature has appeared in a category
def fcount(self,f,cat):
    if f in self.fc and cat in self.fc[f]:
        return float(self.fc[f][cat])
    return 0.0
```

Creating classifier

```
# The number of documents in a category
def catcount(self,cat):
    if cat in self.cc:
        return float(self.cc[cat])
    return 0

# The total number of documents
def totalcount(self):
    return sum(self.cc.values())

# The list of all categories
def categories(self):
    return self.cc.keys()
```

Creating a Classifier – Train method

```
def train(self, doc, cat):
    features = self.getfeatures(doc)

    # Increment the count for every feature with this category
    for f in features:
        self.incf(f, cat)

    # Increment the count for this category
    self.incc(cat)
```

Let's check if our classifier works correctly so far!

```
import docclass
cl = docclass.classifier(docclass.getwords)
cl.train('the quick brown fox jumps over the lazy dog', 'good')
cl.train('make quick money in the online casino', 'bad')
print cl.fcount('quick', 'good')
print cl.categories()
```

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','bad')
    cl.train('make quick money at the online  casino','bad')
    cl.train('the quick brown fox jumps','good')
```

İSTANBUL ŞEHİR UNIVERSITY

Calculating probabilities

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

$$= \frac{\text{\# of documents that contains “word” in C}}{\text{the total number of documents in C}}$$

Calculating probabilities

```python
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 cat.
    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','bad')
    cl.train('make quick money at the online casino','bad')
    cl.train('the quick brown fox jumps','good')
```

Calculating probabilities Consider zero prob.

```
def fprob(self, f, cat, default_prob=0.01):
    if self.catcount(cat) == 0:
        return 0

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

Calculating probabilities

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

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 “programming”.

Naive Classifier

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

  the word "casino" appears in 80% 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) = 0.2 × 0.8 = 0.16

Naive Classifier

```
class NaiveBayes(classifier):

    def docprob(self,doc,cat):
        features = self.getfeatures(doc)
        # Multiply the probabilities of all the features together
        p = 1
        for f in features:
            p *= self.fprob(f,cat)
        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(Category | Document) =

$$\frac{\text{Pr(Document | Category) x P(Category)}}{\text{Pr(Document)}}$$

Problem

- Suppose that
  - Pr(Bad) = 0.20 (20% of the training documents are labeled as Bad)
  - Pr(Good) = 0.80 (80% of the training documents are labeled as Good)
  - Pr(Python & Casino | Bad) = 0.2 × 0.8 = 0.16
  - Pr(Python & Casino | Good) = 0.8 × 0.1 = 0.08

$$\text{Pr(Bad | Python \& Casino)} = \frac{0.20 \times 0.16}{0.20 \times 0.16 + 0.80 \times 0.08}$$

constants

$$\text{Pr(Good | Python \& Casino)} = \frac{0.80 \times 0.08}{0.20 \times 0.16 + 0.80 \times 0.08}$$

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>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.

Classify method

```
def classify(self, item):
    # Find the category with the highest probability
    max=0.0

    for cat in self.categories():
        cat_prob = self.prob(item,cat)

        if cat_prob > max:
            max = cat_prob
            best = cat

    return best
```