Automatic Feature Induction for Text Classification

Jason Rennie
jrennie@ai.mit.edu

Joint work with Tommi Jaakkola.

Definition of Spam is Personal

```
Subject: Win up to $50--Help out fellow MIT students
From: Alcohol Study <jgut@mit.edu>
Date: Sat, 07 Dec 2002 16:24:17 -0500
To: mitalcoholstudy@yahoo.com

Do you want to be entered into a lottery to win cash prizes?
If so, take a minute and fill out this simple survey at
http://web.mit.edu/jgut/www/Survey.htm.  Winners will be
contacted through another web page (/winners.htm) and will be
notified using code words given at the bottom of the survey.
Thank you for your participation.
```

Spam is constantly changing

- Unique Subject strings (e.g. [udxzc]) to avoid hash-matching
- Random From addresses (e.g. xixnsd@naver.com) to avoid replies
- Friendly Subject/body text (e.g. "Unbelievable, I'm your neighbor!") to make you read it
- "Unsubscribe" URL to confirm e-mail addresses
- Interesting use of HTML comments (e.g. "En<!–foo–>gland")

Every time we discover a feature to catch spam, the spammers will find a work-around...

The Problems with Current Spam Filters

- Most spam filters lex messages in a fixed way.
  - Look for words and/or a pre-defined set of features.
- When spammers adapt, a human must find new features to catch and add them to the system.
- Leads to endless cycle of trailing the spammers—is there a better way?

What types of features are there?

- Many of these are simple string-matches (e.g. “Dear”)
- Many others are simple regular expressions (e.g. 3 consecutive 8-bit characters in HTML comment)

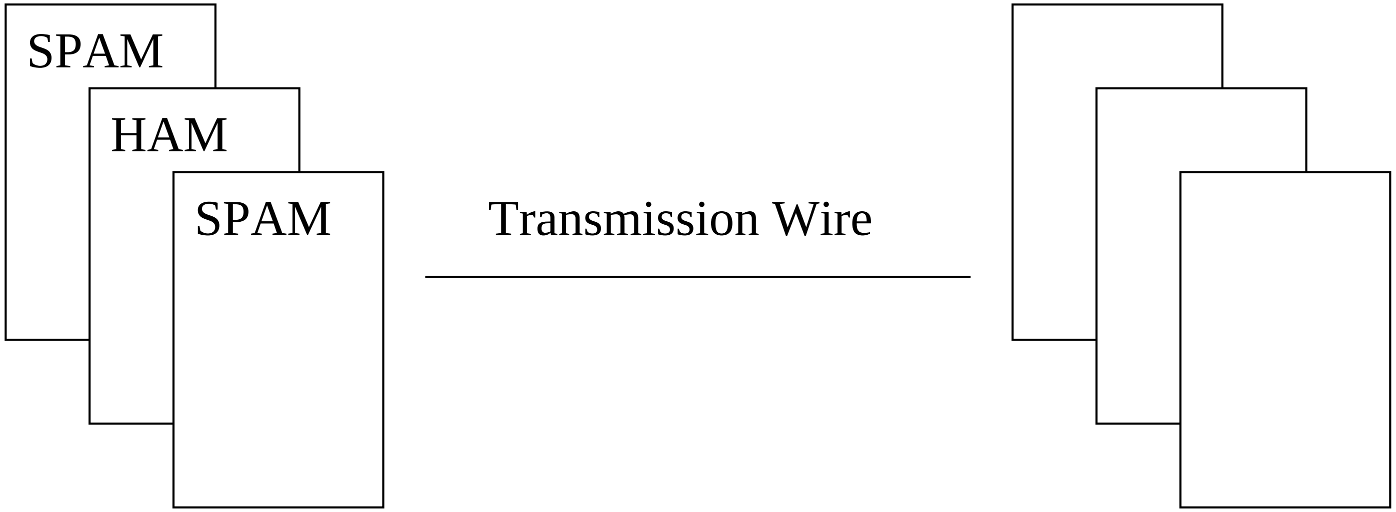
A Better Way: Feature Induction

- Learn features automatially: either fixed strings or simple regular expressions.
- Huge space of possible features. How do we handle it?

Compression

Standard compression problem:

- E-mails and labels (ham/spam) are at one end of wire.
- Copy of e-mails at other end of wire.
- What is fewest number of bits needed to transmit labels?



Minimum Description Length

- Concept introduced by Jorma Rissanen (1978).
- **Idea:** Best generalization achieved by smallest encoding of training examples.

MDL: Rule List Example

- Default encoding: each label requires one bit.
- Consider rule: **any message contaning "sex" is spam**.
- Say 65 training messages include "sex": 60 are spam, 5 are not.
- Encoding of rule requires $3 \log(26) = 14$ bits plus 1 bit $= 15$ bits.
- New encoding of labels requires average of $\mathrm{H}(\mathrm{Binom}(\frac{60}{65}, \frac{5}{65})) = 0.39$ bits per label: 25 bits.
- Improvement: $65 - (25 + 15) = 25$ bits

Simple MDL Algorithm

- Let length = (# examples)
- Let ruleSet = {}
- while (length < oldLength)
  - foreach newRule
    - calcLength({ruleSet, newRule})
  - ruleSet = {ruleSet, bestNewRule}
  - length = calcLength(ruleSet)

Rule List: Examples of Learned Features

␣x	comp.os.xwindows
␣windows	comp.os.ms-windows.misc
␣car␣	rec.autos
for␣sale	misc.forsale
␣turk	talk.politics.mideast
486	comp.sys.ibm.pc.hardware
3.1	comp.os.ms-windows.misc
␣␣$	misc.forsale
t␣condition	misc.forsale

Regular Expressions

- Cost of rule is now encoding of the regular expression.
- Potential features:
  - Toll-free telephone numbers
  - HTML comments (recall “En<!–foo–>gland”)
  - “Dear (something)”
  - URLs without “http://”

The Bad News

- Inherently tied to a classifier
- Naive implementation is *exceedingly* slow
- Not clear how to handle counts (non-binary features)

The Good News

- MDL approach learns the most appropriate, most general features for text classification
- Finds digit, punctuation, etc. features just as easily as alphabetic features
- Automatically learns *new* features to handle new types of spam (given labeled examples to learn from)
- Learning is personalized

Related Work

- Language Segmentation (de Marcken 1996)
  - Also used MDL approach.
  - Excellent way to learn parts-based decomposition of objects.
- String Kernels (Haussler 1999, Lodhi et. al. 2001)
  - Project document into feature space of substrings of length n or less, find linear decision boundary.
  - Also very computationally expensive.