Naïve Bayes

CENG 499

Introduction to Data Science

Erdoğan Doğdu

Classification

Example: Spam Filter

- Input: an email
- Output: spam/ham

- Setup:
  - Get a large collection of example emails, each labeled “spam” or “ham”
  - Note: someone has to hand label all this data!
  - Want to learn to predict labels of new, future emails

- Features: The attributes used to make the ham / spam decision
  - Words: FREE!
  - Text Patterns: $dd, CAPS
  - Non-text: SenderInContacts
  - …

Dear Sir.

First, I must solicit your confidence in this transaction, this is by virture of its nature as being utterly confidencial and top secret. …

TO BE REMOVED FROM FUTURE MAILINGS, SIMPLY REPLY TO THIS MESSAGE AND PUT "REMOVE" IN THE SUBJECT.

99 MILLION EMAIL ADDRESSES FOR ONLY $99

Ok, Iknow this is blatantly OT but I'm beginning to go insane. Had an old Dell Dimension XPS sitting in the corner and decided to put it to use, I know it was working pre being stuck in the corner, but when I plugged it in, hit the power nothing happened.

Example: Digit Recognition

- Input: images / pixel grids
- Output: a digit 0-9

- Setup:
  - Get a large collection of example images, each labeled with a digit
  - Note: someone has to hand label all this data!
  - Want to learn to predict labels of new, future digit images

- Features: The attributes used to make the digit decision
  - Pixels: (6,8)=ON
  - Shape Patterns: NumComponents, AspectRatio, NumLoops
  - …


0


1


2

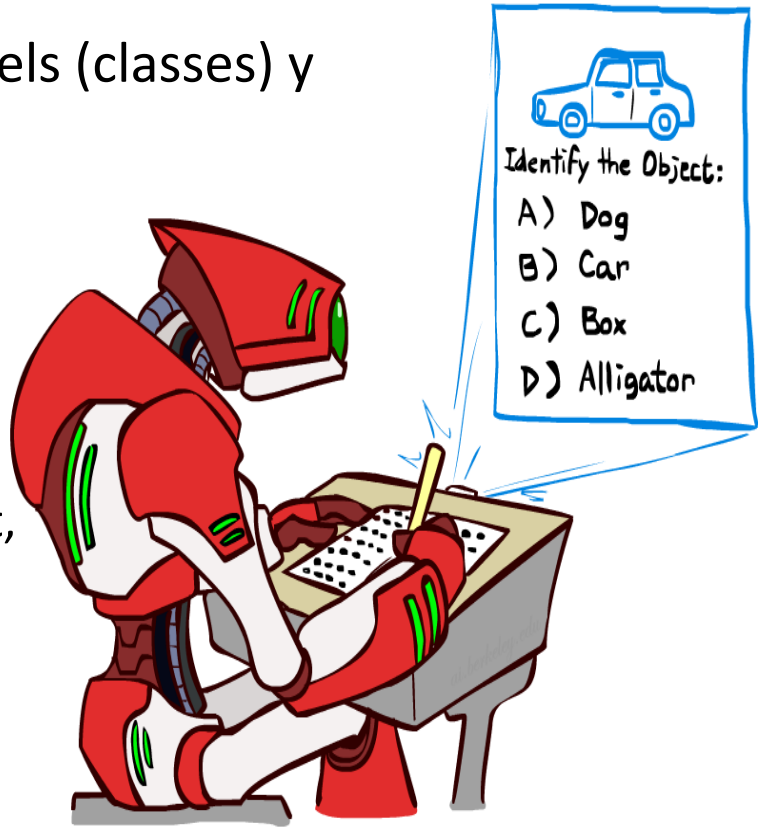

1


??

Other Classification Tasks

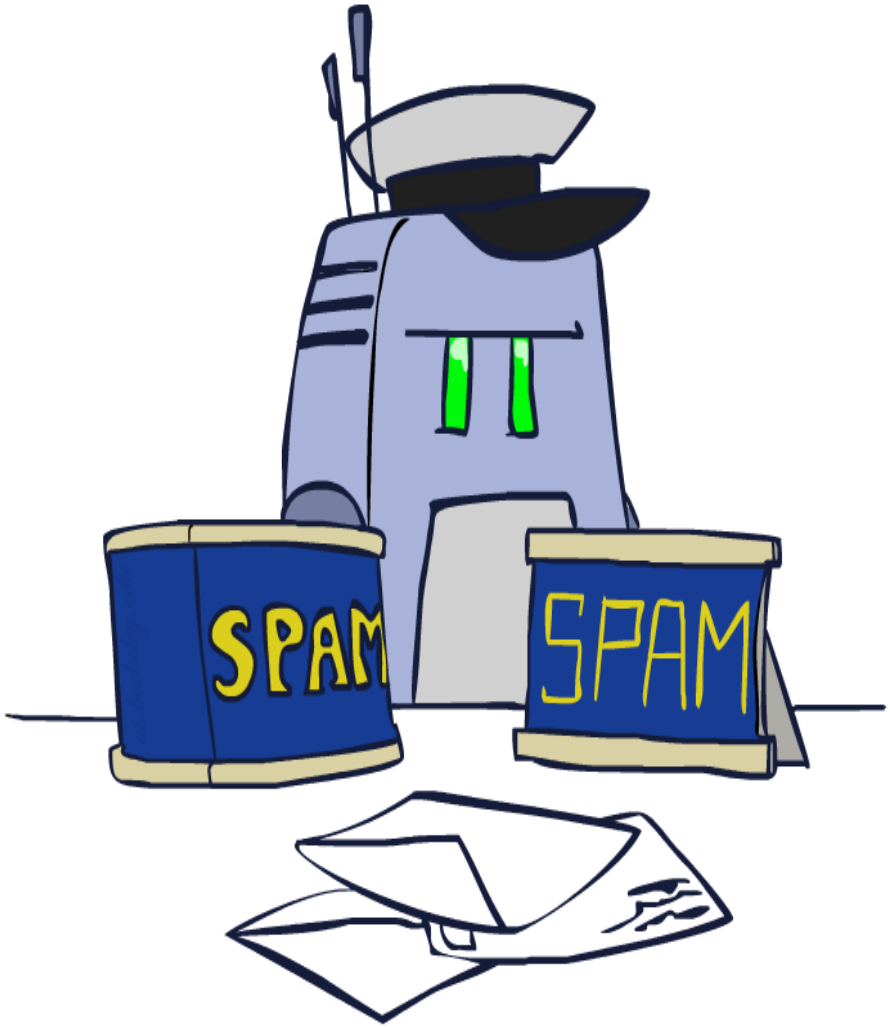
- Classification: given inputs x, predict labels (classes) y

- Examples:
  - Spam detection (input: document, classes: spam / ham)
  - OCR (input: images, classes: characters)
  - Medical diagnosis (input: symptoms, classes: diseases)
  - Automatic essay grading (input: document, classes: grades)
  - Fraud detection (input: account activity, classes: fraud / no fraud)
  - Customer service email routing
  - … many more



- Classification is an important commercial technology!

Model-Based Classification



Model-Based Classification

- Model-based approach
  - Build a model (e.g. Bayes' net) where both the label and features are random variables
  - Instantiate any observed features
  - Query for the distribution of the label conditioned on the features

- Challenges
  - What structure should the BN have?
  - How should we learn its parameters?

Spam Filter

- Problem: Filter out spam messages
- Events:
  - S: Message is spam
  - V: Message contains the word *Free*
- *Bayes' Theorem says:* the probability that the message is spam conditional on containing the word *Free* is:

$$P(S \mid V) = \frac{P(V \mid S)P(S)}{P(V \mid S)P(S) + P(V \mid \neg S)P(\neg S)}$$

Spam Filter

- Assume any message is equally likely to be spam or not-spam: P(S) = P(¬S) = 0.5

$$P(S \mid V) = \frac{P(V \mid S)}{P(V \mid S) + P(V \mid \neg S)}$$

- Example: if 50% of spam messages have the word *Free*, but only 1% of non-spam messages do, then the probability that any given *Free*-containing email is spam is:

$$0.5 / 0.5 + 0.01 = 98\%$$

Spam Filter

- A vocabulary of many words w_1, ..., w_n
- $P(X_i|S)$: the probability that a spam message contains the i^{th} word
- $P(X_i|\neg S)$: the probability that a nonspam message contains the i^{th} word.
- $P(X_1=x_1, \ldots, X_n=x_n|S)=(X_1=x_1|S)x \ldots x (X_n=x_n|S)$

Spam Filter

- Vocabulary = {‘free’,’rolex’)
- Assume half of spam messages include ‘free account’ and the other half of spam messages include ‘authentic rolex’
- Naive Bayes estimate that a spam message contains both “free” and “rolex” is:
  $P(X_1=1, X_2=1|S)=(X_1=1|S) \times (X_2=1|S) = .5 \times .5 = .25$

Spam Filter

- The probability that a message is spam:

$$P(S \mid X = x) = \frac{P(X = x \mid S)}{P(X = x \mid S) + P(X = x \mid \neg S)}$$

by multiply together the individual probability estimates for each vocabulary word.

- May cause "underflow" problem
  - *Instead of* $p_1 * p_2 * \ldots * p_n$
  - $\exp(\log p_1 + \cdots + \log p_n)$

Spam Filter

- Estimates for $P(X_i|S)$ and $P(X_i|\neg S)$
  - probabilities that a spam message (or nonspam message) contains the word w_i
  - the fraction of spam messages (in the training set) containing word w_i
  - Problem: Some words may not appear in spam
    - P("data"|S) = 0
    - Smoothing by *pseudocount k*
    - $P(X_i|S)$ = (*k* + number of spams containing w_i) / (2*k* + number of spams)
    - if "data" occurs in 0/98 spam documents, and if *k* is 1, we estimate P("data"|S) as 1/100 = 0.01

Implementation

```
def tokenize(message):
    message = message.lower()                       # convert to lowercase
    all_words = re.findall("[a-z0-9']+", message)   # extract the words
    return set(all_words)                           # remove duplicates
```

```
def count_words(training_set):
    """training set consists of pairs (message, is_spam)"""
    counts = defaultdict(lambda: [0, 0])
    for message, is_spam in training_set:
        for word in tokenize(message):
            counts[word][0 if is_spam else 1] += 1
    return counts
```

Implementation

```
def word_probabilities(counts, total_spams, total_non_spams, k=0.5):
    """turn the word_counts into a list of triplets
    w, p(w | spam) and p(w | ~spam)"""
    return [(w,
             (spam + k) / (total_spams + 2 * k),
             (non_spam + k) / (total_non_spams + 2 * k))
             for w, (spam, non_spam) in counts.iteritems()]
```

```
def spam_probability(word_probs, message):
    message_words = tokenize(message)
    log_prob_if_spam = log_prob_if_not_spam = 0.0

    # iterate through each word in our vocabulary
    for word, prob_if_spam, prob_if_not_spam in word_probs:

        # if *word* appears in the message,
        # add the log probability of seeing it
        if word in message_words:
            log_prob_if_spam += math.log(prob_if_spam)
            log_prob_if_not_spam += math.log(prob_if_not_spam)

        # if *word* doesn't appear in the message
        # add the log probability of _not_ seeing it
        # which is log(1 - probability of seeing it)
        else:
            log_prob_if_spam += math.log(1.0 - prob_if_spam)
            log_prob_if_not_spam += math.log(1.0 - prob_if_not_spam)

    prob_if_spam = math.exp(log_prob_if_spam)
    prob_if_not_spam = math.exp(log_prob_if_not_spam)
    return prob_if_spam / (prob_if_spam + prob_if_not_spam)
```

```
class NaiveBayesClassifier:

    def __init__(self, k=0.5):
        self.k = k
        self.word_probs = []

    def train(self, training_set):

        # count spam and non-spam messages
        num_spams = len([is_spam
                         for message, is_spam in training_set
                         if is_spam])
        num_non_spams = len(training_set) - num_spams

        # run training data through our "pipeline"
        word_counts = count_words(training_set)
        self.word_probs = word_probabilities(word_counts,
                                             num_spams,
                                             num_non_spams,
                                             self.k)

    def classify(self, message):
        return spam_probability(self.word_probs, message)
```