Feature-based Discriminative Classifiers

Making features from text for discriminative NLP models

Christopher Manning
Classifiers

- A classifier is a function $g$ which assigns an input datum $d$ to one of $|C|$ classes, $c \in C$: $g: D \rightarrow C$

- The classes might be:
  - \{PERSON, ORGANIZATION, LOCATION, O\} for named entity recognition
  - \{politics, sports, finance, technology, arts, leisure, ...\} for news
  - \{spam, notspam\} for an email message
  - \{coreferent, not-coreferent\} for a coreference candidate mention pair
Example problem

• Classify a capitalized proper noun as a class:
  • LOCATION, DRUG, PERSON

• For a data example \(d\)
  • \(taking\) Zantac

• We work by considering each class \(c\) for the word:
  • (LOCATION, \(taking\) Zantac, )
  • (DRUG, \(taking\) Zantac, )
  • (PERSON, \(taking\) Zantac, )

• and using features to score each candidate classification
Features for a classifier

• *Features* $f$ are elementary pieces of evidence that link aspects of what we observe $d$ with a category $c$ that we want to predict.

• A feature is a function with a bounded real value: $f: C \times D \rightarrow \mathbb{R}$
  
  • Common special case in NLP:
    
    • binary features $f: C \times D \rightarrow \{0, 1\}$
Example binary features

• $f_1(c, d) \equiv [c = \text{LOCATION} \land w_{-1} = \text{“in”} \land \text{isCapitalized}(w)]$
• $f_2(c, d) \equiv [c = \text{LOCATION} \land \text{hasAccentedLatinChar}(w)]$
• $f_3(c, d) \equiv [c = \text{DRUG} \land \text{ends}(w, \text{“c”})]$

1.8 \quad -0.6

LOCATION in Arcadia \quad LOCATION in Québec \quad 0.3 \quad \text{DRUG taking Zantac} \quad \text{PERSON saw Sue}

• Models will assign to each feature a weight:
  • A positive weight votes that this configuration is likely correct
  • A negative weight votes that this configuration is likely incorrect
Binary Features

- Very commonly, a feature specifies
  1. an indicator function – a yes/no boolean matching function – of properties of the input $\Phi$ and
  2. a particular class

$$f_i(c, d) \equiv [\Phi(d) \land c = c_j] \quad \text{[Value is 0 or 1]}$$

- Each feature picks out a data subset and suggests a label for it
- The decision about a data point is based only on the values of the features active at that point.
More General Features

- Features can be more general than just binary matching:
  - Can compute a real value from input, e.g., log(word length)
  - Can match a set of values – e.g., perhaps a partial structure – across “classes”
    - This leads to structured classification, which is common in NLP, for example to match parse tree candidates, etc.
      - A discriminative can have features that match a tree with a unary S to VP
      - A coreference model can not like a cluster with different gender items
Building a Simple Discriminative Model

- We define features (indicator functions) over data points
  - Features represent sets of data points which are distinctive enough to deserve model parameters.
    - Words, but also “word contains number”, “word ends with ing”, POS, syntactic structure, relation between two phrases, etc.
- We might simply encode each $\Phi$ feature as a unique String
  - A datum will give rise to a set of Strings: the active $\Phi$ features
  - Each feature $f_i(c, d) \equiv [\Phi(d) \land c = c_j]$ gets a real number weight
- We concentrate on $\Phi$ features, but one weight for each $i$ of $f_i$
Building a Simple Discriminative Model

- Features are normally added in big batches via feature templates
  - E.g., one feature template adds $\forall i,j$ observed: $\text{lastWord}=w_i \land c = c_j$
  - Another is: $\text{nextWord}=w_i \land c = c_j$. Each may add tens of thousands of features
- A model may be specified by the set of feature templates used
- Features are often added during model development to target errors
  - Often, the easiest thing to think of are features that mark bad combinations
Linear classifiers at classification time

- Linear function from feature sets \( \{f_i\} \) to classes \( \{c\} \).
- Assign a weight \( \lambda_i \) to each feature \( f_i \).
- We consider each class for an observed datum \( d \).
- For a pair \((c,d)\), features vote with their weights:
  - \( \text{vote}(c) = \sum \lambda_i f_i(c,d) \)
- Choose the class \( c \) which maximizes \( \sum \lambda_i f_i(c,d) \)
Linear classifiers at classification time

- Linear function from feature sets \( \{ f_i \} \) to classes \( \{ c \} \).
- Assign a weight \( \lambda_i \) to each feature \( f_i \).
- We consider each class for an observed datum \( d \).
- For a pair \((c,d)\), features vote with their weights:
  - \( \text{vote}(c) = \sum \lambda_i f_i(c,d) \)

Choose the class \( c \) which maximizes \( \sum \lambda_i f_i(c,d) = \text{LOCATION} \).
Feature-based Discriminative Classifiers

Making features from text for discriminative NLP models
Feature-based softmax/maxent linear classifiers

How to put features into a classifier
Feature-Based Linear Classifiers

- Linear classifiers are a linear function from feature sets \( \{ f_i \} \) to classes \( \{ c \} \)
- At test time, we consider each class \( c \) for a datum \( d \)
  - We generate a feature set \( \{ f_i \} \) for an observed datum-class pair \((c,d)\)
  - Each feature \( f_i \) has a weight \( \lambda_i \)
  - We then score each possible class assignment: \( \text{vote}(c) = \sum \lambda_i f_i(c,d) = \lambda \cdot f \)
  - We choose the class \( c \) which maximizes \( \sum \lambda_i f_i(c,d) \)
- At training time we have observed \((c,d)\) pairs from labeled examples
  - We generate sets of features \( \{ f_i(c,d) \} \) for them
  - We use information about what features occur and don’t occur to set a weight \( \lambda_i \) for each feature
Example features

- $f_1(c, d) \equiv [c = \text{LOCATION} \land w_{-1} = \text{“in”} \land \text{isCapitalized}(w)]$
- $f_2(c, d) \equiv [c = \text{LOCATION} \land \text{hasAccentedLatinChar}(w)]$
- $f_3(c, d) \equiv [c = \text{DRUG} \land \text{ends}(w, \text{“c”})]$
Maxent models (softmax, multiclass logistic, exponential, conditional log-linear, Gibbs)

- Make a probabilistic model from the linear combination $\sum \lambda_i f_i(c,d)$

$$P(c \mid d, \lambda) = \frac{\exp \sum \lambda_i f_i(c,d)}{\sum_{c'} \exp \sum_i \lambda_i f_i(c',d)}$$

- $P(\text{LOCATION} \mid \text{in Québec}) = \frac{e^{1.8} e^{-0.6}}{e^{1.8} e^{-0.6} + e^{0.3} + e^{0}} = 0.586$
- $P(\text{DRUG} \mid \text{in Québec}) = \frac{e^{0.3}}{e^{1.8} e^{-0.6} + e^{0.3} + e^{0}} = 0.238$
- $P(\text{PERSON} \mid \text{in Québec}) = \frac{e^{0}}{e^{1.8} e^{-0.6} + e^{0.3} + e^{0}} = 0.176$

- The weights are the parameters of the probability model, combined via a “soft max” function
Feature-Based Linear Classifiers

- Maxent models:
  - Given this model form, we choose parameters \( \{\lambda_i\} \) that maximize the conditional likelihood of the data according to this model (as discussed later): \( \max_{\Lambda} P(D|C, \Lambda) \)
  - We construct not only classifications, but probability distributions over classifications.
Feature-Based Linear Classifiers

There are other (good!) ways to choose weights for features:

- Perceptron: find a currently misclassified example, and nudge weights in the direction that corrects classification.
- Margin-based methods (Support Vector Machines)
- Boosting algorithms

But these methods are not as trivial to interpret as probability distributions over classes.
How to put features into a classifier