Machine Learning for Language Processing (L101)

Ann Copestake

Computer Laboratory
University of Cambridge

October 2017

The story so far

Models:

1. Naive Bayes (see also Manning et al, Chapter 13)
2. HMMs
3. Discriminative classifiers: MaxEnt, MEMM

Methodology:

1. Task/data/model; shared tasks.
2. Smoothing. Feature engineering.
3. Token annotation schemes. Regularization (in passing). Precision/recall balance.

Model power

- Why don't we just use the more powerful models and forget about the simpler ones?
- Speed etc
- Also: danger of overfitting: powerful models can pick up unintended effects in training data.
- Suggested additional reading: Manning et al §14.6: the bias-variance tradeoff.
- Especially important to be careful with deep learning models: very sensitive to artifacts; blackbox.
- Not at all clear we have methodology right yet: sensitivity to classes of artifact?

Model power

- Why don't we just use the more powerful models and forget about the simpler ones?
- Speed etc
- Also: danger of overfitting: powerful models can pick up unintended effects in training data.
- Suggested additional reading: Manning et al §14.6: the bias-variance tradeoff.
- Especially important to be careful with deep learning models: very sensitive to artifacts; blackbox.
- Not at all clear we have methodology right yet: sensitivity to classes of artifact?

Where we're headed

4 Today: focus is linear and non-linear classifiers. (Note: Naive Bayes is a generative linear classifier, we just don't normally describe it that way . . .)

5 Clustering. Topic models, LDA: latent variables, Dirichlet, hyperparameters.

6 Gibbs sampling, RBMs, intro to deep learning. Preparation: L90 notes for lecture 8 and 9 (8 online now, 9 ready early next week).

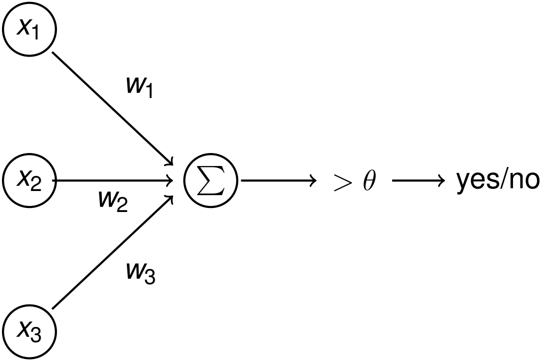
7 LSTMs. Compositional distributional semantics.

8 Current issues

Rest of today's lecture

- Perceptron: an early neurally-inspired linear classifier
  - Gradient descent training
  - Limitations of perceptrons
  - Perceptrons in NLP
- Kernel methods: non-linear decisions from linear classifiers
- An NLP example

Perceptron



Dot product of an input vector $\vec{x}$ and a weight vector $\vec{w}$, compared to a threshold θ

Perceptron

- The perceptron was one of the first neural network architectures (Rosenblatt 1962)
- Cognitively inspired — but nobody knew much about how real neurons worked then ...
- Multilayer perceptron is not a perceptron ...
- perceptron algorithm for learning — suitable for classification where linearly separable.
- Many variants: kernel perceptron, voted perceptron (which is competitive with techniques such as SVMs).
- In NLP, mainly for parse selection (alternative to MaxEnt).
- Description here based on Manning and Schütze: see Stephen Clark's notes for perceptron applied to tagging.

Perceptron learning algorithm

- Simple example of gradient descent (also know as hill climbing, gradient ascent).
- Move the prediction in the direction of the training data via the steepest gradient (i.e., derivative).
- Theory fairly complex, implementation simple (and fast!).
- Will converge if problem is linearly separable, but:
  - boundary may flip back and forth — not always clear in training if it will converge or if problem non-linear
  - results depend on training data order, boundaries non-optimal

Perceptron learning algorithm

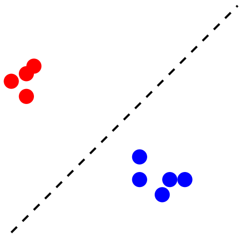
θ threshold, $\vec{w}$ weights, $\vec{x}_j$ (numerical) feature vector

decision($\vec{x}_j, \vec{w}, \theta$) is **yes** if $\vec{w} \cdot \vec{x}_j > \theta$ else **no**

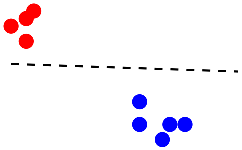
initialize $\vec{w}$ and θ to 0

while not converged **do**
  for each element $\vec{x}_j$ in training set **do**
    d := decision($\vec{x}_j, \vec{w}, \theta$)
    if trueclass$(\vec{x}_j) = d$ **then continue**
    elseif trueclass$(\vec{x}_j) =$ **yes then** $\theta := \theta - 1$
      $\vec{w} := \vec{w} + \vec{x}_j$
    elseif trueclass$(\vec{x}_j) =$ **no then** $\theta := \theta + 1$
      $\vec{w} := \vec{w} - \vec{x}_j$
    fi

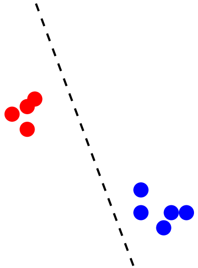
Perceptron boundaries



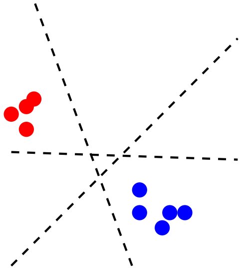
Perceptron boundaries



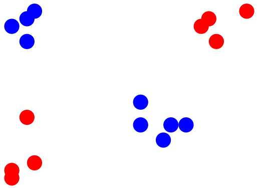
Perceptron boundaries



Perceptron boundaries



Perceptron decisions: XOR



Perceptron issues

- Multiple possible boundaries between linearly separable data points: other approaches just find one boundary (e.g., SVMs)
- Some classes not linearly separable: this issue was partly responsible for killing NN work in the 1960s.
- BUT: very fast, fast to train, so can use with more data than other methods for a given amount of CPU time.
- Originally: specialist perceptron hardware.

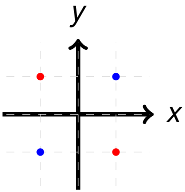
Perceptrons in NLP

- Introduced to NLP by Collins in 2002 (voted perceptron).
- Tagging (see Steve Clark's notes from two years ago), named entity recognition but primarily used for parse ranking.
- Can be used in conjunction with kernels. e.g., parse ranking: features are all subtrees of parse tree (so exponential number): use tree kernels.
- Kernels allow perceptrons and other methods to be used for problems that are not linearly separable.

Kernel methods

- Roughly: a kernel is a function which allows features to be mapped to an inner product in a higher-dimensional (possibly infinite) feature space.
- A valid kernel is defined by any symmetric finitely positive semi-definite function (psd: definition Manning et al, p305).
- Hence, if we prove a function has these properties, then we have a kernel: no need to explicitly represent the mapping.
- Various similarity measurements are kernels, including cosine similarity and Jensen-Shannon divergence.

XOR and Kernels



- Add a third dimension, xy: blue $xy = 1$, red $xy = -1$, so linearly separable.
- More generally, use a quadratic kernel:

$$K(\vec{u}, \vec{v}) = (1 + \vec{u}^T \vec{v})^2$$

- or more generally still, a polynomial kernel.

Why kernel methods?

- Allow linear classifiers to learn non-linear classification functions.
- Allow structured objects (trees, strings, sets etc) to be classified by vectorial methods (convert to real numbers, fixed length).
- Multiple kernels may be combined to give a new kernel: usually better performance than treating them individually.
- Can be used in conjunction with a variety of ML methods: e.g., perceptron (first used by Aizerman et al 1964).
- SVMs use kernels.

String kernel example

Feature space from character pairs: 1 if contiguous in word, λ if one intervening character, λ^2 if two intervening characters etc

	c-a	c-t	c-r	a-r	r-t	b-a	b-r
$\phi(cat)$	1	λ	0	0	0	0	0
$\phi(cart)$	1	λ^2	λ	1	1	0	0
$\phi(bar)$	0	0	0	1	0	1	λ

$$k(\text{cat}, \text{cart}) = 1 + \lambda^3$$
$$k(\text{cat}, \text{bar}) = 0$$
$$k(\text{cart}, \text{bar}) = 1$$

From Mark Gales' L101 slides 2010/11

Compound noun relations

- *cheese knife*: knife for cutting cheese
- *steel knife*: knife made of steel
- *kitchen knife*: knife characteristically used in the kitchen

Very limited syntactic/phonological cues in English, so assume parser gives: N1(x), N2(y), compound(x,y).

Language-specific restrictions

German compounds with non-compound translations:

Arzttermin	*doctor appointment	doctor's appointment
Terminvorschlag	* date proposal	proposed date
Terminvereinbarung	* date arrangement	arrangement of a date
Januarhälfte	* January half	half of January
Frühlingsanfang	* spring beginning	beginning of spring

Data-driven approaches to compound relation learning

- Find paraphrases by looking for explicit relationships in corpora: e.g., knife made of steel
  (Lauer: prepositions, Lapata: verbal compounds)
- treat as a supervised classification problem:
  1. human annotation of compounds: e.g., steel knife annotated with BE
  2. use distributional techniques to compare unseen to seen examples.

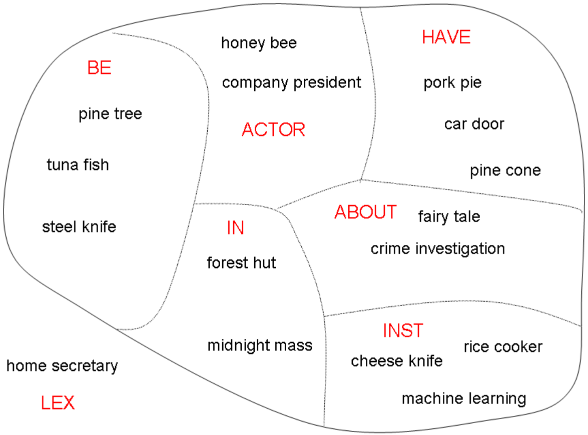
  Girju et al, Turner, Ó Séaghdha (2008) among others.

Relation schemes for learning experiments: Ó Séaghdha (2007)

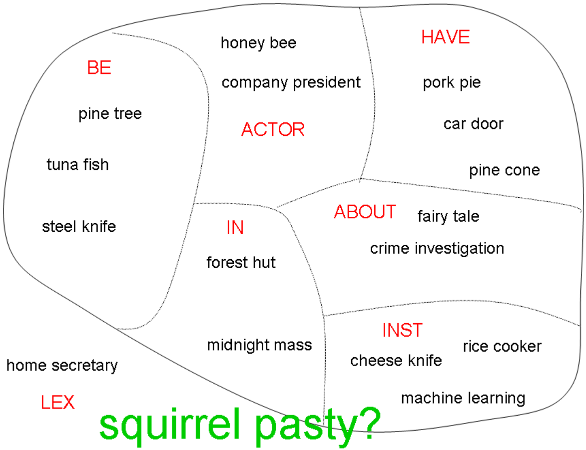
BE, HAVE, INST, ACTOR, IN, ABOUT: (with subclasses)
LEX: lexicalised, REL: weird, MISTAG: not a noun compound.

- Relation scheme based on Levi (1978)
- Considerable experimentation to define a usable scheme: some classes very rare (therefore not annotated reliably).
- Annotation of 1400 examples from BNC by two trained annotators, using extensive guidelines.
- Reasonable interannotator agreement (IAA).

Compound noun relation learning



Compound noun relation learning



Squirrels and pasties

Compound noun relation learning: Ó Séaghdha, 2008

- Use distributional methods: count vectors, acquired from subset of parsed British National Corpus and from Google 5-gram corpus.
- Distributions normalised to give probabilities.
- Apply distributional similarity to the compound phrase (note difference between compound noun and adjective-noun combination).
- Treating compounds as single words?
  Distributional vector for pork pie compared with vector for squirrel pasty?

Compound noun relation learning: Ó Séaghdha, 2008

- Two similarity methods that do work:
  1. Constituent similarity: compounds x1 x2 and y1 y2, compare x1 vs y1 and x2 vs y2.
     squirrel vs pork, pasty vs pie
  2. Relational similarity: compare **sentences** with x1 and x2 vs sentences with y1 and y2.
     squirrel is very tasty, especially in a pasty vs
     pies are filled with tasty pork
- Comparison using kernel methods: including combined constituent and relational similarity kernels.
- Best accuracy: about 65% (only slightly lower than agreement between annotators).
- Same system successfully used for a SEMEVAL task: classifying relationships between unconnected words in a sentence.

Kernel methods vs deep learning

- Deep learning is now potentially an alternative to kernels for structured input.
- Deep learning is perhaps theoretically more interesting when you don't know much about the structure of the problem (because less feature engineering, can potentially learn structure) but sometimes very difficult to apply to NLP problems.
- Kernel methods can be fast: Ó Séaghdha's linear kernels took 45 minutes to train on Google 5-gram with a slow CPU.
- Various hybrid methods are being proposed.