6.864 Advanced Natural Language Processing[1]

Lecture 1: Introduction

9 September 2015

[1] Instructors: Prof. Regina Barzilay, and Prof. Tommi Jaakkola.
TAs: Franck Dernoncourt, Karthik Rajagopal Narasimhan, Tianheng Wang.
Scribes: Brandon Carter, Sarah Vente.

This first lecture introduces Natural Language Processing (NLP). It presents the core concepts of NLP, some of its main applications as well as challenges, a succinct history of the field, and machine learning approaches.

Definition of Natural Language Processing (NLP)

TODO

Applications

TODO

Challenges

TODO

History

TODO

Applying Machine Learning Methods

The Determiner Placement Task

The determiner placement task is defined as follows: given a text with no determiner, add any missing determiner to the text. For example, the given text could be "*Scientists in United States have found way of turning lazy monkeys into workaholics using gene therapy.*". To simplify the task, assume that we only consider one sort of determiner, the definite article *the*. The decision is therefore binary, viz. whether *the* should precede a given noun in a given text.

For a native English speaker, performing this task is usually easy. However, there exist many grammar rules specifying determiner placement, based on linguistic knowledge such as the countability of the noun (i.e. whether it is countable or uncountable) or the plurality of the noun (i.e. whether it is singular or plural), as well as on world knowledge such as the uniqueness of reference (*the current president of the US*) or the situational associativity between nouns (*the score of*

the football game). In addition to these generic grammar rules, there exist numerous exceptions: for instance, while the definite article is required in newspaper titles (e.g. *The Times*), it shall not be used for names of magazines and journals (e.g. *Time*).

Given the amount of rules and exceptions involved in determiner placement, it would be tremendously difficult to manually encode all of this information: a symbolic system for the determiner placement task would subsequently be difficult to successfully construct.

Determiner Placement as a Supervised Classification Task

As an alternative to the symbolic approach, one could adopt a statistical approach to the determiner placement task by casting it as a supervised classification task.

A basic classifier could be trained as follows:

1. Gather a large collection of texts (e.g. a set of newspaper articles) and split it into a training set and a test set.
2. For each noun in the training set, compute the probability it follows a certain determiner, i.e. $p(\text{determiner}|\text{noun}) = \frac{\text{count(determiner, noun)}}{count(noun)}$.
3. Given a text from the test set where determiners have been removed, for each noun in the text, select the determiner with the highest probability as estimated in the previous step.

The quality of the predictions made by this classifier is not outstanding, but still surprisingly high for such a simple method: using the first 21 sections of the Wall Street Journal (WSJ) corpus as the training set, and the 23rd section as the test set, the prediction accuracy is 71.5%. This result can be used as a baseline for more elaborate classifiers.

The typical steps to train a classifier in a supervised fashion are as follows:

1. *Data collection*: Gather a large collection of texts, and split it into a training set and a test set. Because one can easily get a training set of sample sentences in English with correct determiner placement, no addition work is needed to obtain the labels: we assign a label -1 or $+1$ to each noun ($+1$ if the noun is preceded by a determiner, -1 if not).
2. *Feature extraction*: For each noun in the training and test sets, extract some features. E.g. whether the noun plural is plural (0 if no, 1 if yes), whether it is its first appearance in the text (0 if no, 1 if yes), and so on. We therefore obtain one feature vector for each noun. Figure 1 illustrates the process of feature extraction.

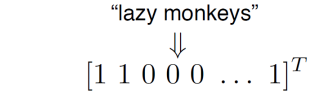

Figure 1: **Feature extraction** is the process of converting a text to a feature vector, before feeding it to a classifier. If there are n binary features, then the feature vector will be of dimension n.

3. *Training*: Train a classifier using the extracted features as well as the labels. The training step is used to find the optimal feature weight vector to improve the final prediction. Figure 2 shows a decision boundary (in simplified two-dimensional space) based on the feature vectors and weight vectors.

4. *Testing*: Assess the classifier's prediction quality on the test set. Using this machine learning approach, one can reach an accuracy above 86%.

Table 1 shows pairs of features and label that result from step 1 and 2. To train the classifier in step 3 using these features and labels, there exist many classification algorithms such as the perceptron algorithm. The perceptron algorithm iterates through the training examples and finds an optimal decision boundary to separate as many of the training points as possible: it places the decision boundary so that it optimizes the number of training points that are classified correctly, as shown in Figure 2.

Once the decision boundary is placed, one can plot any new vector into this space and determine on which side of the decision boundary it is located. A noun whose vector falls on the $+1$ side is assigned a $+1$ label, while a noun whose vector fall on the other side of the boundary is assigned a -1 label. This classification procedure therefore utilizes the training examples with known determiner placement to classify the unlabeled examples.

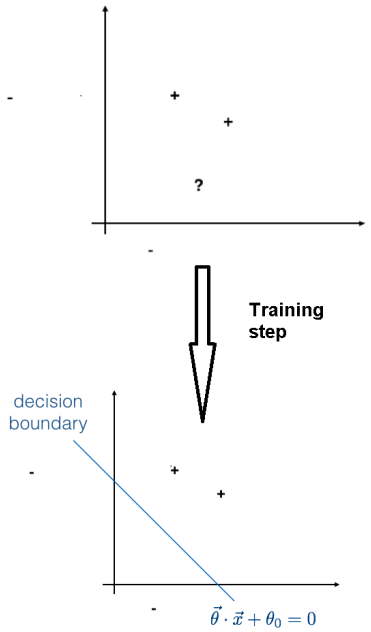

Figure 2: Learning the **decision boundary** based on the feature vectors and weight vectors. In this example (in simplified two-dimensional space), there are two positive points (with $+$ labels), and two negative points (with $-$ labels). The decision boundary correctly separates all of the training points.

	Features			
Noun	plural?	1st appearance?	unigram	label
Scientists	1	1	00100	null
United States	1	1	00010	the
lazy monkeys	1	1	00001	the
gene therapy	0	1	01000	null

Table 1: Example of feature values and labels given to the classifier during the training phase. This table results from data collection, labeling and feature extraction. The feature values for the unigram feature present in this table are much smaller than it is in reality: unigram feature values are one-hot vectors of dimension $|V|$, where V is the dictionary.

One of the most common features in NLP is word *n-grams*, or simply n-grams. A word n-gram is a sequence of n words. A unigram designates a 1-gram, a bigram is a 2-gram, and a trigram is a 3-gram. n-gram of larger sizes are less often used. An n-gram feature value is encoded as a *one-hot vector*, i.e. a vector in which all elements are 0 except one element that is 1. For example, $[0, 0, 0, 1, 0]$ is a one-hot vector. If the vocabulary is of size $|V|$, then there are $|V|$ possible unigrams, $|V|^2$ possible bigrams, and $|V|^3$ possible trigrams. This means that the unigram feature value will be a one-hot vector of dimension $|V|$, the bigram feature value will be a one-hot vector of dimension $|V|^2$, and the trigram feature value will be a one-hot vector of dimension

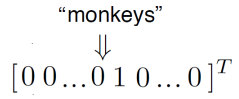

Figure 3: Extracting an unigram feature is the process of converting a word to a one-hot vector, which will be used as a feature.

$|V|^3$. Despite its simplicity, using n-gram as a feature often yields high performances: if the unigram feature is added to the classifier in the determiner placement task (i.e. the unigram feature is extracted from the noun, as shown in Figure 3), it significantly improves the accuracy of the classifier. However, n-gram features are very sparse, since $|V|$ is typically very large, often over 10^4.

Limitations of Traditional Machine Learning Approaches

This traditional machine learning approach has several issues, the two main ones being the sparsity of the feature vectors and feature engineering:

- *Sparsity*: feature vectors are typically high-dimensional and sparse (i.e. most elements are 0). Figure 4 illustrates that sparsity often results in many feature vectors not being seen when the training set is small. Figure 5 shows the impact of sparsity on accuracy of parsers for different languages.
- *Feature Engineering*: the second problem with using traditional machine learning approaches for classification is feature engineering. Early studies spent much time engineering features. Traditionally, features of compositional objects are manually-selected concatenations of atomic features, as Figure 6 illustrates. However, crafting these features by hand is difficult and time-consuming as they are combinatorial and the number of them is therefore extremely large.

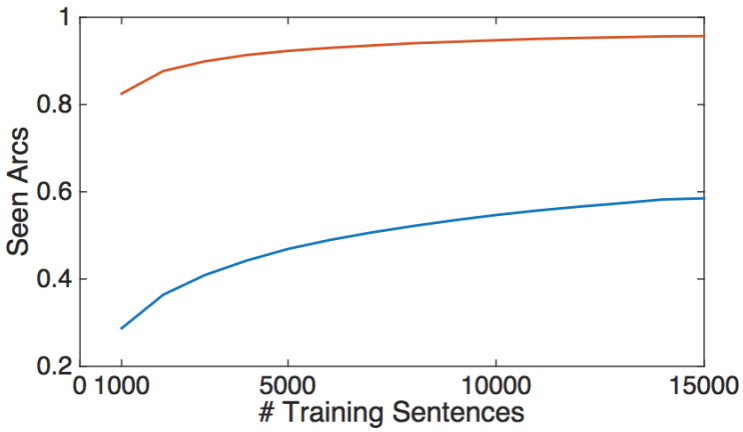

Figure 4: In this graph, the red line indicates the portion of unigrams (single words) in the unseen tests that has already been seen in training. The first 1000 training sentences contain over 80% of unigrams, and the first 10000 about 90% of them. The blue line in the graph indicates the portion of bigrams (pairs of words) in the unseen samples that has already been seen in the training samples: this number is a lot lower than that for unigrams. Thus, if we consider bigrams for classification, it becomes difficult to be able to accurately classify unseen examples when the classifier has not seen the majority of bigrams during the training phase.

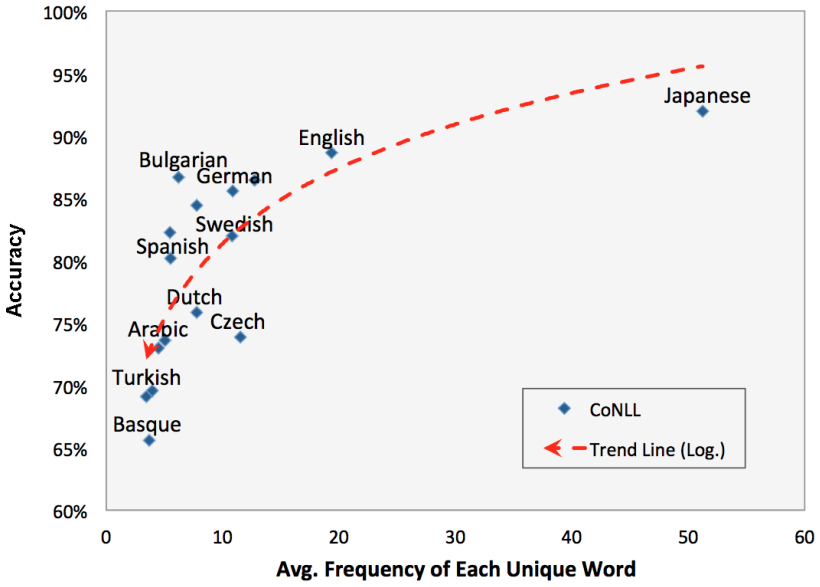

Figure 5: This graph shows the impact of sparsity on the accuracy of parsers for different languages. Japanese and English have the highest accuracies, and a good predictor of the parser accuracy is the average frequency of each unique word. This means that the parser has a much greater accuracy when it can be trained on larger subsets of the words that will appear in the test set.

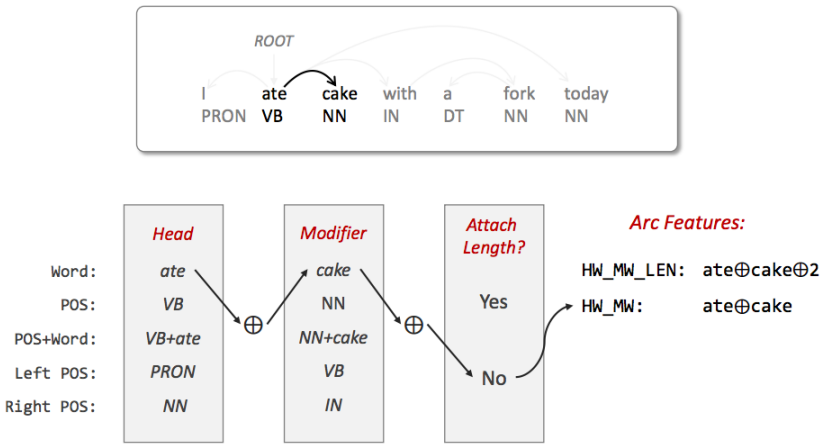

Figure 6: Selecting feature representation. Traditionally, features of compositional objects are manually-selected concatenations of atomic features.

Word Embeddings

Word embeddings are one solution to the sparsity problem. Embeddings are a set of machine learning techniques involving artificial neural networks that map discrete, one-hot vectors into low-dimensional continuous representations. They compress large sparse vectors into smaller, dense vectors, with semantically related words located close to each other in the word embedding space, as Figures 7 and 8 show. The word embedding space also contains interesting linear substructures, as Figure 9 illustrates: word embeddings are very powerful as they can capture some semantic relations between words.

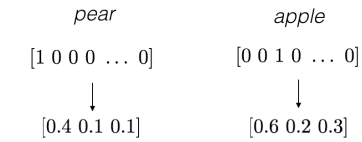

Figure 7: The words "pear" and "apple" are mapped from one-hot vectors to continuous, dense vectors of much lower dimension, where they are located close to each other.

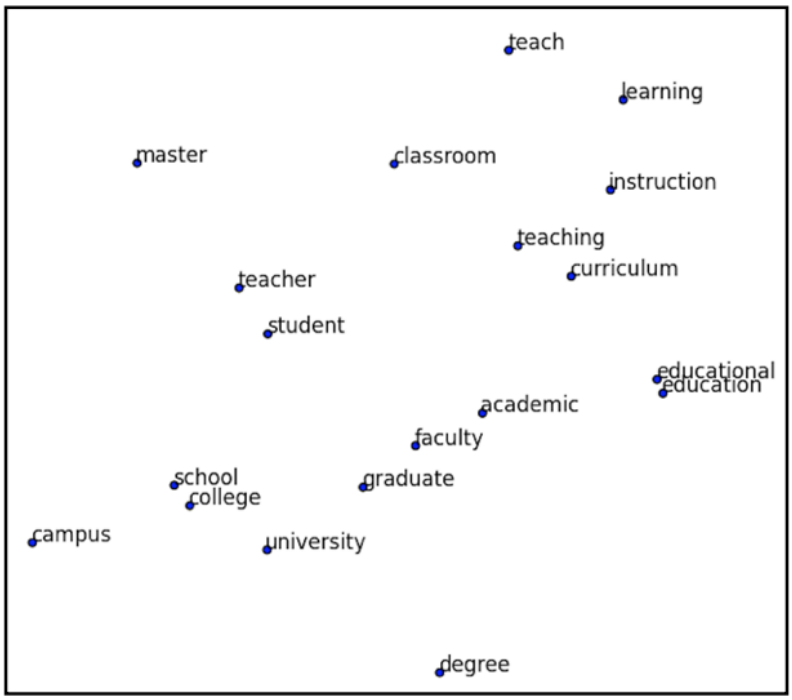

Figure 8: t-SNE visualization of word embeddings (t-SNE stands for t-Distributed Stochastic Neighbor Embedding). This plot illustrates a projection of embedded vectors into 2-D space. Semantically close words such as "school" and "college" are mapped closely to one another in the word embedding space, whereas they would appear unrelated when encoded as one-hot vectors. Source: `http://nlp.yvespeirsman.be`

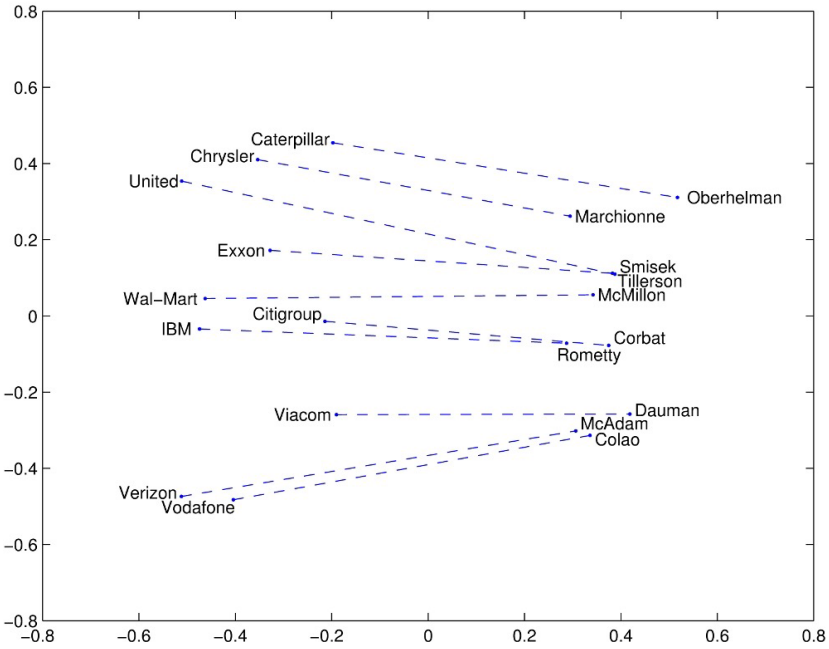

Figure 9: Linear substructures in word embeddings. In this example, if we take the embedding corresponding to the CEO of one company, subtract that company, and add another company, we will arrive at the CEO of the other company. Source: `http://nlp.stanford.edu/projects/glove`

Word embeddings are also one solution to the problem of feature engineering. Suppose we are trying to find a feature representation for the phrase "red apples." We could start with the word embedding vector for "red" and the word embedding vector for "apples". Then, we could add them, concatenate them, or combine them otherwise, as Figure 10 illustrates. Traditionally, we would just engineer features and use them for the main problem that requires these features. Using word embeddings, one can learn them jointly with the learning problem, for example by adding a hidden layer in learning problem, which allows us to learn compositional meaning in the vector space, optimize the feature representation, and then carry it forward to the final NLP task.

In general, one can tackle NLP tasks using traditional machine learning approaches, or newer techniques based on artificial neural networks. This course will cover both approaches.

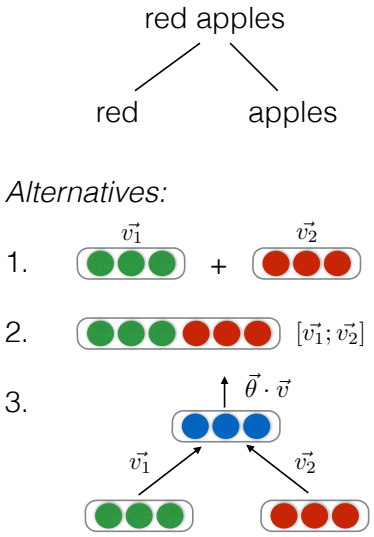

Figure 10: Three strategies amongst others to combine word embeddings.