Data, Annotation, Evaluation

Jonathan May

August 24, 2022(Prepared for Fall 2022)

1 History of Methodologies

1.1 Pre-Statistical (1650s/1950s through approx. 1980s)

- Mostly about modeling specific linguistic phenomena in a small number of sentences, sometimes using code
- Linguists/highly trained coders wrote down fine-grained detailed rules to capture various aspects, e.g. ‘ “swallow” is a verb of ingestion, taking an animate subject and a physical object that is edible...’
- Very time-consuming, expensive, limited coverage (brittle), but high precision
- Academically satisfying, but not good at producing systems beyond the demo phase

1.2 Statistical

- Empirical approach: learn by observing language as it’s used “in the wild”
- Many different names:
  - Corpus Linguistics
  - Empirical NLP
  - Statistical NLP
- Central tool:
  - corpus
  - thing to count with (i.e. statistics)
  - (later on) machine learning methodologies, software/hardware for helping with scale
- Advantages
  - Generalize patterns as they actually exist (i.e. bottom-up, not top-down)

  - Little need for knowledge (just count)
  - Systems are robust and adaptable (change domain by changing corpus)
  - Systems degrade more gracefully (corner cases captured in data)
  - Evaluations are (more) meaningful

- Limitations
  - Bound by data – can't model what you can't see – "I held the book with my arm stretched out and opened my hand. It (floated away), (fell to the ground)"
  - Big Data methods fail when the data is small or wrong – sometimes you want to try to translate Oromo news to English with 50,000 words of bible when you want 10m+words of news
  - more computationally expensive (but less human-expensive) (usually a good trade-off)
  - Methods don't have the same pattern-recognition and generalization abilities of humans learning (and putting into rule-based methods) which can lead to unintuitive brittleness...even with very big LMs we can still get surprised by a sudden wrong answer a human would never make (and a rule-based model would just not even bother to try).

1.3 Corpus (pl: corpora): a collection of (natural language) text systematically gathered and organized in some manner

- Features:
  - Size
  - Balanced/domain
  - Written/Spoken
  - Raw/Annotated
  - Free/Pay
- Some Famous (text) examples:
  - Brown Corpus: 1m words balanced English text, POS tags
  - Wall Street Journal: 1m words English news text, syntax trees
  - Canadian Hansards: 10m words French/English parliamentary text, aligned at sentence level
  - Google books ngrams: 500B words
  - Wikipedia: 2B words in 4.4m articles. Well connected to other languages and some implicit markup

  - Common Crawl: attempts to crawl the web since 2008, and is regularly updated. petabytes of data, billions of pages, needs a lot of cleaning, dedup, etc to do anything useful
  - The Pile: 800GB combination of 22 high quality datasets (you may not need to do so much cleaning)
  - Dear students who read ahead: any new ones that I forgot?

1.4 How Big Does it need to be?

- We'd like to get examples of all linguistic phenomena, ideally several times so we know how likely they are to occur
- How big should a corpus be to get every possible sentence in English?
  - Every possible idea?
  - Every 5-word phrase?
  - Every word?
- None of these are possible!

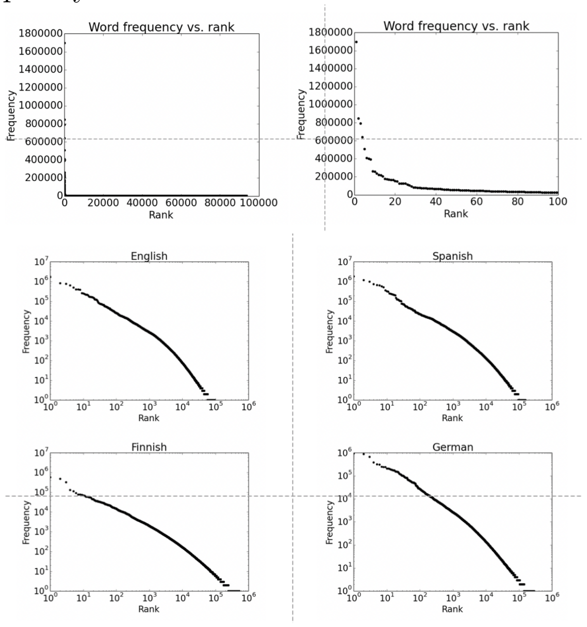
1.5 corpus processing

```
# word counts and ngram counts

# 10 most frequent words in the text
sed 's/ /\n/g' sawyr11.txt | sort | uniq -c | sort -k1nr | head
# 10 most frequent words in the text after removing blank lines
sed 's/ /\n/g' sawyr11.txt | grep -v "^$" | sort | uniq -c | sort -k1nr | head
# 10 most frequent bigrams (2 word sequences) in the text

sed 's/ /\n/g' sawyr11.txt | grep -v "^$" > ts.words
tail -n+2 ts.words > ts.2pos
paste ts.words ts.2pos | sort | uniq -c | sort -k1nr | head
# count number of words/ngrams, number of word/ngram types, number of
# 1-count word/ngram types

# words in the text without blank lines, one word per line, saved to a file
# (for convenience)
sed 's/ /\n/g' sawyr11.txt | grep -v "^$" > ts.words
# number of word tokens
wc -l ts.words
# number of word types
sort ts.words | uniq | wc -l
# number of one-count words
sort ts.words | uniq -c | awk '$1==1{print}' | wc -l
```

```
# number of two-word sequence (bigram) tokens
# (based on the answer to the number of word tokens you should know this
# without running the command...
paste ts.words <(tail -n+2 ts.words) | wc -l
# number of bigram types
paste ts.words <(tail -n+2 ts.words) | sort | uniq | wc -l
# number of one-count bigrams
paste ts.words <(tail -n+2 ts.words) | sort | uniq -c | awk '$1==1{print}' | wc -l
```

1.6 Zipf's law

Take a naturally occurring corpus (in this case, of text). Count frequency of words. Order words by frequency, to form ranks. Then the rank of a word is inversely proportional to its frequency.



For frequency f and rank k, $f \approx \frac{1}{k^a}$ for some constant a. Thus $-a \approx \frac{\log(f)}{\log(k)}$.
Consequences:

- There will always be a lot of infrequent or unseen words
- This is true at all levels of linguistic structure
- So we have to find clever ways of generalizing so we can get reasonable estimates for things we haven't seen often enough

1.7 Word Segmentation

- motivation from finnish but also for out of vocabulary issues
- morphology based

- unsupervised model-based
- unsupervised simple count-based (BPE): BPE example using "BITTER BUTLER BITE BUTTER BITER"

Segmentation will reduce the zipfian effect, since it reduces the vocabulary size. However, it will not eliminate it!

Nearly all of NLP benefits from a dispassionate analysis of how well the models we build perform the analysis or generation of NL data we are trying to have them analyze/generate. Aside from avoiding inherent bias and blindness toward a system's quality (which can lead to trouble down the road), there are circumstances where optimizing on the metric can be a good way of improving performance. But understanding what correctness and incorrectness is and how to measure these values can be tricky. Here we try to cover some of the key methods of evaluation in NLP. Evaluating these models means having labeled data for them; we'll discuss strategies for collection too.

2 Quantitative Analysis

2.1 Development, Test, Blind

One way to build your system is to look at some labeled data, write code that tries to get the right labels on that data, see what labels it's getting incorrect, refine, repeat. At the end you will have a system that does well on the corpus you looked at. But this often means you will have neglected some phenomena present **outside** your training corpus. Worse, you may engineer things so well on training that previously correctly labeled items outside of your training corpus are now mislabeled.

This is called *overfitting* and it usually means you are taking advantage of some eccentricity in your training data that does not generally hold (e.g. all sentences with an even number of words are positive sentiment).

To avoid overfitting it's a good idea to divide your data as follows: most (80-90%) is used to build your model (train corpus), a sample (10-20%) is used to periodically evaluate but not to build the model (dev corpus), and another sample (5-10%) is not looked at and only used for evaluation, very very seldomly (test corpus). You can track overfitting: plot a graph of training vs. dev performance over time; if dev starts to go down while train goes up, that's overfitting. How much to evaluate on test? It's an art, honestly. The more you do, the more you will probably overfit, and then you'll need another test corpus to verify this.

If you don't have a lot of data, you can do what's called *cross-validation*. You divide up your data and evaluate. Then you slide the dividing lines to create another *fold* of the data. Keep doing this and every chunk of data gets to be train, dev, and test. Then average the results. I think this has the tendency to lead to overfitting more quickly but it is not infrequently used.

The extreme version of cross-validation is n-fold where n is the number of items you have; this is called *leave-one-out* validation and allows the maximum amount of training data to be used (but I trust it the least).

A hybrid strategy, where cross-validation is used for train and dev, but the test set is held constant, is probably a good compromise.

The best scenario is when someone else holds on to a piece of the data for you and you only ever get to see it once, when you're totally done with your model. This corpus is called 'blind' and is usually only used in the context of shared tasks.

It's important to consider how you divide your data. Ideally data should be independent and identically distributed (IID). You might think random selection of labeled elements would be suitable. For a collection of sentiment-labeled movie reviews this is true. However, if your corpus is a set of documents, you don't want to have one sentence from a document in training and another in test. Words and phenomena tend to cluster in a document, since a document is about some topic, with topic-relevant words, and is generally written by one author, and will have a particular style. So best practice is to divide up along *document* boundaries randomly, but pay attention to the number of evaluated items that get added to each set.

Some notes on evaluation measures follow. The choice of evaluation measures is always subject to debate (look at MT metrics workshops) but here are some guidelines I like to use:

2.2 Accuracy

For strict classification, where each item receives a label from a fixed set of labels, and the distribution of labels is reasonably even (doesn't need to be all the way even, but shouldn't be 90% one class), simple accuracy $= \frac{\text{correct}}{\text{total}}$ is a perfectly good metric.

2.3 F-Measure

For cases where there is one 'background' label that predominates and a relatively few instances of a 'content' label (e.g. named entity recognition) F-measure, which combines precision and recall, is a better choice, and is calculated on all labels *except* the background label. Precision is $\frac{\text{correct}}{\text{hypothesis}}$, i.e., how much of what you predicted is correct. Recall is $\frac{\text{correct}}{\text{reference}}$, i.e. how much of what was correct did you predict. Using either one by itself can be misleading: why?

F1 is a *harmonic mean* of precision and recall. Specifically it's $2 \cdot \frac{PR}{P+R}$. In general $F_\beta = (1+\beta^2)\frac{PR}{\beta^2 P+R}$; F_2 is weighted to favor recall, and $F_{0.5}$ is weighted to favor precision. In certain circumstances one may be preferred (e.g. precision is less important if there will be a downstream selection task, while recall is less important if the task is to mine information from a very large corpus and the amount of information mined is more important than assurances it has all been mined).

What is described above can be more specifically described as *micro-averaged* F1, i.e. each non-background labeling is considered in making the tallies of what is correct, then the averages are calculated. One can also consider *macro-averaged* F1, where F1 is calculated for each kind of label (e.g. in named entity recognition, PERSON, then LOCATION) where all other labels are considered to be background. These F1s are then averaged together.

Micro-F1 is the same as accuracy when there is no background class; Macro-F1 can be done in these situations too if there is class imbalance (i.e. one dominant class is mostly correct but you want to weight each class evenly instead of each item) Example:

		Gold				
		None	Person	Location	Company	Total
Hypothesis	None	N/A	0	5	10	15
	Person	0	200	10	0	210
	Location	0	5	40	0	45
	Company	5	0	0	10	15
	Total	5	205	55	20	

Micro-averaged Precision: $\frac{200+40+10}{270} = 0.926$
Micro-averaged Recall: $\frac{200+40+10}{280} = .893$
Micro-averaged F1: $2\frac{.926 * .893}{.926 + .893} = 2 * .827/1.819 = .909$
Macro-averaged precision: $\frac{\frac{200}{210}+\frac{40}{45}+\frac{10}{15}}{3} = .836$
Macro-averaged recall: $\frac{\frac{200}{205}+\frac{40}{55}+\frac{10}{20}}{3} = .734$
Macro-averaged F1: $2\frac{.836 * .734}{.685 + .854} = 2 * .614/1.539 = .782$

2.4 Granularity

It's important to consider what item is ultimately judged correct or incorrect. Even if each word or sentence is labeled, it may make more sense for a sequence of adjacent labels to be examined to consider whether an item is correct or not.

2.5 Rank-based evaluation

Sometimes you get to return more than one label, or you get to label your items in preferential order (e.g. information retrieval). Sometimes your results are actually pretty bad but you want to convey that your algorithm is better than random (e.g. unsupervised bilingual lexicon induction). There are a few strategies for evaluating this kind of data:

Precision/Recall@N: Consider up to N ranked items. Careful – the way this metric is implemented can vary! In e.g. information retrieval, P@N means you consider N items per query and calculate precision over the N*Q total items retrieved. Howeer in e.g. bilingual lexicon induction, P@N means if *any* of the N items retrieved is correct you consider the entire item is correct, i.e. precision is calculated over Q total items retrieved. If P@N monotonically increases with N, it's the latter.

Precision@R: Especially in retrieval like cases, you can control the tradeoff between more precision and more recall but adjusting your threshhold of returning an item. This would numerically show the precision at a fixed recall. This can also be plotted to determine ideal operating points. A calculation of the *area under the curve (AUC)* is a good single number that summarizes this tradeoff (higher is better).

2.6 Edit distance/word error rate

For structured output, especially text that is generated, the idea of 'how much work does it take to fix this' is very relevant, especially if the task will be, say, a first pass before human correction (this is very common in, say, the translation service industry). Given the number of substitutions S, deletions D, and insertions I made over a text of length n, the error rate (WER or TER for task T) is:

$$WER = \frac{S + D + I}{n}$$

Variants assign different cost to different operations or operations involving different components. For example, inserting/removing determiners might only cost 0.4 of an edit, while doing so for content words could cost 2.3. These values are set experimentally.

2.7 Intrinsic Vs. Extrinsic Analysis

Intrinsic: evaluate the task on its own merits. E.g. parse F1, POS tag accuracy. Evaluation 'close' to the task. Advantage: directly evaluates the model you're building. Disadvantages: does it matter?

Extrinsic: evaluate the task as it plugs into some other task (e.g. parsing in the service of summarization). Need to hold the other technology constant. There are different levels of this (narrow = pos tagging for parsing; wide = machine translation for surgery outcome). This also brings up the question of **component** vs. **end-to-end** evaluation. Should you use a sequence of noisy components (more realistic, exposes problems of interaction, can be tough to tell what actually causes the problem) or use 'gold' data at every point in the pipeline before your tool (better for narrow debugging, doesn't give a realistic picture)? Both have their value. Don't let something out before testing end to end, though.

2.8 Human judgments

The gold standard! Just like with machines, you have to tell humans how to evaluate. And if the way you tell them to evaluate is too 'weird' you may not get results you like.

The major advantage is you can ask for judgement calls that are specifically something machines can't do. For example:

- Given one reference translation, ask them to edit a machine translation until it *means the same thing* as the reference.
- Evaluate the sentiment of a text on a n-point scale
- Is a response sarcasm?
- ...

There are a few drawbacks:

- Humans are slow compared to machines.
- Humans are expensive compared to machines.
- Humans are inconsistent.

The last one is maybe trickiest. The same human may give two different annotations. Or two humans may be internally consistent but disagree with each other. There are ways to improve agreement:

- Have several annotate and take the most frequent label (hopefully small label set)
- Have several annotate, then talk together and resolve differences
- Have several annotate, then have a super annotator resolve differences.

Ultimately, though, you want to track *inter-annotator agreement* (IAA):

Basic idea: ask n humans to annotate the same data. Check for their overlap. There are several metrics, most based on this basic equation: $\frac{P(a)=P(e)}{1-P(e)}$ for agreement a and expected agreement e. 1 for perfect agreement. 0 for chance agreement. Can be negative if agreement is worse than chance.

- Cohen's κ (only good for 2 humans): See below. What is a good value? Wide disagreement. If you have to choose, at least 0.8 (but some might say 0.4).
- Scott's π – same as Cohen's κ but take squared arithmetic means to determine $P(e)$. Less informative than Cohen's since it assumes equal distribution of responses.
- Fleiss's Kappa (generalizes to n humans)...maybe 0.5 is ok? Generalization of Scott's π.
- Krippendorff's α – more general. Also allows for missing data, partial agreement. Quite complicated and computationally intensive to calculate.

Demo:

		B	B	B	total
		pos	neut	neg	
A	pos	54	28	3	85
A	neut	31	18	23	72
A	neg	0	21	72	93
	total	85	67	98	250

A used 'pos' 85 times = .425. B also. So for pos, $P(e) = .425 \times .425 = .180$. Similarly for neut and neg, $P(e) = .425 + .077 + .146 = .403$. $P(a) = \frac{54+18+72}{250} = .576$. Then $\kappa = \frac{.576-.403}{1-.403} = .29$.

For Scott's π but instead we calculate $P(e)$ as $\frac{85+85}{500}^2 + \frac{72+67}{500}^2 + \frac{93+98}{500}^2 = .3388$ so $\kappa = \frac{.576-.3388}{1-.3388} = 3587$

Note: IAA not really helpful for, e.g., translation. There you just want to collect many different responses and use them all to evaluate.

Which to use? Often several are used together. Cohen's over Scott's, but Fleiss' if needed, α if extra mechanism needed. Statisticians might know more.

2.9 Statistical Significance

Ok, you have some results, automatic or human. But can you rely on them? Questions you can ask yourself:

- Are the judgements actually measuring something real?
- Are they something that we care about?
- Is it from the domain/genre that we care about?
- Is it from the right distribution?
- Are there enough examples that we can trust it?

The last question is something we can answer. See also section 4.4.3 of Eisenstein and the Berg-Kirkpatrick reading this narrative is taken from:

Let's say we have two classifiers, A and B. A is better than B on some test set x by $\delta(x)$. Null hypothesis H_0: A is not actually better than B. If true, how likely is it that A would be better than B on some new set x' by at least $\delta(x)$? If $P(\delta(x') > \delta(x)|H_0) < 0.05$[1] then we can say with 95% confidence that H_0 is rejected and indeed A is better than B. 0.05 is called the *p*-value.

There are a variety of methods for establishing p-value, but one easy way that works for lots of metrics and situations where we don't have limitless test data is the following bootstrap approach:

1. Draw b boostrap samples y of size n from x with replacement.
2. Let s be 0
3. for each y, if $\delta(y) > 2\delta(x)$, increment s
4. $p \approx s/b$

What's going on here? Since the samples are all drawn from the test set we in fact want to show how often A is better than expected, and it's expected to beat B by $\delta(x)$ since we already know it does. In the original presentation of this work, $b = 10^6$ which showed stable behavior of p calculation. This was done over a wide variety of task types.

3 Annotated Task Corpora

If you want to know how well you're doing on a task you should compare yourself to examples of the task done correctly. A catch-all term for this is an 'annotation' (also a 'labeling') of data. Here are some examples:

[1]Actually a random variable $\mathbf{X}$ is used, not x'

task	input	label
POS tagging	The boy ran home	DT NN VBD NN
constituency parsing	The boy ran home	(S (NP DT NN) (VP VBD NN))
dependency parsing	The boy ran home	2-det 3-nsubj 0-root 3-advmod
sentiment	The boy ran home	neutral
translation into french	The boy ran home	Le garçon a couru à la maison

Some other kinds of annotation (the scope is limitless):

- phonetic: how was a word spoken, intoned, where were pauses taken, what words were stressed
- semantic: mark words as they're used with senses, draw semantic relatedness graphs
- pragmatics/discourse: what role does each turn play (e.g. acknowledgement, request for feedback, acceptance, marker for new phase). Is a statement a thesis, antithesis, elaboration, rebuttal, justification, etc? what are the discourse units? Resolve anaphora – what do pronouns refer to? markers for attribution (something someone else claimed happened), For more look at rhetorical structure theory (RST)

4 How to Gather Annotation

The lack of data has stymied many a project but I encourage you to think of it as an opportunity, not a roadblock! Advice: "If you want to get a lot of citations, publish a corpus." (Philipp Koehn, prof. JHU. Very famous. Lots of released corpora.)

4.1 Found

Sometimes there is natively an annotation already in existence, though possibly not in quite the right format. A simple case is that text reviews of movies often come with star ratings, which can be turned into positive/negative sentiment. A more esoteric kind of found annotation would be, say, using text spans in wikipedia that also contain page links that have info boxes used for celebrities to determine the presence of a person mention in NE tagging. Such methods can be quite powerful but are also quite noisy; they are often referred to as 'silver standard' for this reason.

Beware of using a found annotation that itself was programmatically generated or that you programmatically generate from the data, this is circular reasoning and will lead you to either create a trivial data set (e.g. list of descriptions of Japanese comic characters with annotation of whether they were originally anime or manga; labels generated by finding which word comes first in the description) or one that is impossible to label (sentiment labeled data; labels are chosen by running sentiment analysis). It sounds absurd but is easier to accidentally create than you might think!

4.2 DIY

You can (and should) annotate some data yourself before trying to get other people to do it. This will help you develop your guidelines and give you a sense of how difficult the task is. However, it will still be an overestimate, because you may try to write down annotation guidelines (see below) but there will be hidden assumptions that won't come out until you try to have someone else do the same annotation you're doing

4.3 Phone a Friend

Just doing your own annotation is unwise; you could do consistent annotation but nobody would be able to follow on since you won't need to write a comprehensive standard. You also will inherently be observing any test data you produce, so your systemns will be likely to overfit. Asking someone to try to do the annotation (advisor, colleague in your lab) is a good first test of your approach. It also gives you a way to judge IAA. If they are good enough and the annotation is simple enough you can make a sufficiently sized corpus without too much effort, you can have your friend's annotations serve as a test set, and maybe they can be a coauthor of your paper.

4.4 Hire

Once you want to get serious about annotation you'll want to parallelize and increase your throughput rate and diversity of annotator. Direct hiring (we can sometimes bring in MS or undergrad student workers) has the main advantage that you can have fairly tight control over your workers' outputs, you can have them use custom tools quite easily, and you can get and give a lot of feedback. If your annotation has special skill requirements (e.g. knowledge of a particular language) this may be the best way to go. The downside is it can be tough to find dedicated annotators, since most annotation is rather tedious. It can also be expensive; you'll be competing with other employers and your team is likely to be able to choose where to work.

4.5 Crowd

Amazon Mechanical Turk and Toloka have made the job of hiring pools of annotators much easier, but there are a number of caveats when working with MTurk. First, it is not in practice as cheap as you might imagine. Ethically you should pay a living wage; we use \$15/hr as a minimum. You will get questions about this when you publish and you may get pushback. You will also get lower quality work or no takers if your rate is too low. In practice you can't pay hourly, you have to pay by the piece. This means your annotators will have much less appetite for reading very long annotation guidelines. It also means you need to calibrate your pay by getting test annotation and estimating completion time.

Another major hassle is that there will be attempts to exploit you by not doing meaningful work. Although you are allowed to not pay for work that is not done, rejecting work already done for low quality will earn you a bad reputation, which will lower the quality and number of workers. It is important to carefully vet workers by having them randomly annotate items you already have the answers to and then validating their responses against these. You can also have workers do an entire set you know the answer to, and if they do well, you can give them specific qualification to do more work. For bots/non-workers, best bet is to simply not hire them again.

If you are doing research in a lab, annotation via crowd work may be considered human subjects research and require from one or more internal review boards (IRBs). This is more of an issue for your PI than you but you should discuss it before proceeding.

Finally, your interaction with workers is more limited. The interface options are smaller and conveying subtle differences in annotation standards can be tricky.

Caveats aside, using crowd workers is actually quite useful and sometimes the only way to get annotation work done. It's a good idea to try doing some crowd work to get a feel for it!

4.6 Inter-Annotator Agreement

While annotating you'll want to get IAA, in exactly the same way as you'd get IAA for scores per above.

5 Annotation Guidelines

It is generally a good idea to write down your intended rules for how to annotate, in as plain a language as possible. Note that this need not and should not be essentially a computer program (otherwise it wouldn't be an interesting NLP task) but can often be quite detailed. The part of speech tagging guidelines for the Penn treebank, for example, are 37 pages long! Guidelines are only effective if they're followed, of course, so you have to judge how much time your annotators will put in learning how to annotate. If you've hired them, with an hourly wage, longer manuals will probably be okay. If they're doing piece-work, e.g. in mechanical turk, they aren't going to spend time reading manuals and not getting paid.

Here are some classic examples. From the (37 page) PTB POS tag guidelines:

Adjective, superlative–JJS

Adjectives with the superlative ending *-est* (as well as *worst*) are tagged as JJS. *Most* and *least* when used as adjectives, as in *the most or the least mail*, are also tagged as JJS. *Most* and *least* can also be tagged as JJS when they occur by themselves; see the entries for these words in Section 5. Adjectives with a superlative meaning but without the superlative ending *-est*, like *first*, *last* or *unsurpassed*, should simply be tagged as JJ.

Typically, part way through the annotation, tricky cases will be uncovered, then resolved, and added to the annotation standards, like was presumably done here:

```
Words that refer to languages or nations, like English or French,
can be either adjectives (JJ) or proper nouns (NNP, NNPS).

   EXAMPLES: English/JJ cuisine tends to be uninspired.
             The English/NNPS tend to be uninspired cooks.
```

```
In prenominal position, such words are almost always adjectives (JJ).
Do not be led to tag such words as proper nouns just because they occur
in idiomatic collocations.

    EXAMPLES: Chinese/JJ cabbage; Chinese/JJ cooking
              Welsh/JJ rarebit; Welsh/JJ poetry
However, note:
    EXAMPLE: an English/NP sentence
             (cf. a sentence of English/NP)
```

6 Nominal task (also HW1): text classifier

We have some book reviews. We want to know automatically if they're positive or negative.

Positive review:

```
I loved this book. The food was really good and fast (which is good
because I have a very packed schedule). Also great if you're on a
budget. The recipes have variations so I could eat different things but
not have to buy a whole new set of ingrediants.
```

Negative review:

```
I tried reading this book but found it so turgid and poorly written that
I put it down in frustration.  It reads like a translation from another
language by an academic bureacrat. The theme is interesting, the
execution poor.  Cannot recommend
```

6.1 Evaluation

We should always ask 'how do we know we are doing well at what we are trying to do?' This time the answer is fairly simple (it won't always be). We collect many reviews along with their labels (Positive/Negative, which we may have to create out of some other labels, like a numerical score).

We'll calculate simple *accuracy*: $\frac{\#\text{ correct}}{\#\text{ total}}$.

6.2 Data

We will divide our data into *train*, *development* (dev) (also sometimes called 'validation') and *test* corpora. train is used to build a model and is the largest data set (usually 80% of the data). dev is not used to train but is frequently consulted during optimization (where relevant) to avoid *overfitting* on training data. test is usually not evaluated or looked at except for when optimization is done, to ensure even less overfitting. Sometimes an even more super-secret *blind* test set is not even available to you but held off to test your final model.