Web-Scale Named Entity Recognition

Casey Whitelaw
Google
whitelaw@google.com

Alex Kehlenbeck
Google
apk@google.com

Nemanja Petrovic
Google
nemanja@google.com

Lyle Ungar
University of Pennsylvania
ungar@cis.upenn.edu

ABSTRACT

Automatic recognition of named entities such as people, places, organizations, books, and movies across the entire web presents a number of challenges, both of scale and scope. Data for training general named entity recognizers is difficult to come by, and efficient machine learning methods are required once we have found hundreds of millions of labeled observations. We present an implemented system that addresses these issues, including a method for automatically generating training data, and a multi-class online classification training method that learns to recognize not only high level categories such as place and person, but also more fine-grained categories such as soccer players, birds, and universities. The resulting system gives precision and recall performance comparable to that obtained for more limited entity types in much more structured domains such as company recognition in newswire, even though web documents often lack consistent capitalization and grammatical sentence construction.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Text analysis*; I.7.m [**Document and Text Processing**]: [Miscellaneous]

General Terms

Algorithms, Experimentation

Keywords

named entity recognition, text mining, web mining

1. INTRODUCTION

There has been extensive work on recognizing named entities such as people, places, and organizations from corpora such as newswire or scientific literature. Many sophisticated methods have been used, with some reasonable success. This paper shows how to extend this work to tag a much richer set of entities, including geographical locations, movies, books and many sub-categories of people (soccer players, priests, politicians) on the web. The size and heterogeneity of the web introduces a number of problems not present in cleaner corpora such as newswire, not the least of which is obtaining good training data.

Like most named entity classification systems, we use a supervised model to predict the entity type as a function of features of a "mention," such as context it is found in. We have a preexisting set of entity types that we would like to identify, and it is useful to have fixed labels for each of these types. Supervised models require high-quality annotated training data. For some languages, there exist annotated corpora that are widely used for training named entity recognition models, such as the MUC, ACE, and CoNLL data sets. These are unsuited to our application for a number of reasons. First, the set of entity types is small; only person, location, organization, and miscellaneous are annotated. We want to classify many more fine-grained types than this. Second, the data is drawn from newswire corpora, which does not look like the open web. To build models that will work on the open web, we need training data from web pages, in their full diversity, including tables and lists. Worse, since there are millions of potentially useful features (such as words) we need millions of labeled training examples. In fact, we used almost a half billion training instances extracted from tens of millions of web pages, with millions of different features.

Dramatically extending both the scope (number of types of entities) and the scale (number of documents to be tagged) requires a number of issues to be addressed.

- *Training data generation* Given the large number of different formats of web pages, and the many different entities types that we wish to predict, manual annotation of training data is not feasible. We thus require something that almost sounds like an oxymoron: automated ways of generating training data. We start from a seed set of entities and relations, and learn templates to extract more labeled mentions.
- *Feature generation* What features should be used for the type predictors? We combine multiple "base" entity type predictors with a set features of the mention

CIKM'08, October 26–30, 2008, Napa Valley, California, USA.

being tagged and its context: the web page and domain it is on, and the tokens in and near it.

- *Machine learning* We generate a training set of hundreds of millions of mentions labeled with dozens of types and millions of potential features. Estimating models requires some attention to efficiency. The models learned need to be small and fast enough to fit in memory, so that they can be run over the entire web. We describe an approach to feature selection and online training of a classifier which works well in this setting. Accurately predicting entity subtypes requires using multi-label prediction methods that recognize the correlation structure between type labels.

- *Resolution* The machine learning algorithm makes predictions for individual mentions on a page; Final predictions require resolving conflicts of overlapping annotations.

Our end goal was to annotate all mentions of entities of hundreds of types on the entire web. Our entity types fall in a hierarchy, with the highest level containing the more commonly tagged types like people, places and organizations, and a longer list of less common, but still important types such as movies, books, plants, animals, and vehicles. Types such as people are further divided into subtypes such as politician, actor and athlete, which are then further subdivided into categories such as football player. In our current structure, the sibling categories are mutually exclusive; one can be both a person and an athlete (a subtype) but not a politician and an athlete (a sibling type).

For this paper we picked a set of 32 type labels (person, place, company, school, plant, animal, etc.), and built and evaluated a system to label the English portion of the web. The design, however, is consistent with our goal of labeling hundreds of entity types in dozens of languages. Eventually, we also plan to resolve the mentions to determine which place or person is referred to by a specific mention of "Paris" or "Washington"; for now we address the first part of this question, determining if the "Paris" in question is a city, book, song, person, god, etc.

2. TRAINING SET GENERATION

The hardest part of constructing a named entity classifier using supervised learning is the generation of the training data. Manual annotation of terms in web pages was judged to be too expensive, particularly since the goal was to produce a methodology that generalizes relatively easily to many languages other than English. Automated methods of generating training data were therefor sought.

Several data generation techniques were implemented and combined. Although it would be nice to have a single clean method of data generation, we found that in practice combining multiple methods gave superior performance. The goal is to find "trusted mentions," mentions of a name of an entity where we know its type. We use the assumption that a name in a single document always refers to the same entity throughout our classification system, including during training. Thus, since all occurrences of the same entity name in the same document will have the same entity type, and since our training data all comes from the web, we can represent our training data as a set of (name, type, url) tuples.

We start with a seed list which consists of (name, type) pairs. Names may be repeated; I.e., they may be of more than one type. Compared to most other approaches, we use a relatively large seed list. Compared to annotated text, seed lists are relatively easy to find or extract. Web sites such as Wikipedia, and specialist sites such as IMDB, are a rich source of (name, type) information; seed lists can also be learned or bootstrapped from other sources such as query data [5] or the web [18, 13]. The requirements for this seed list are relatively weak: we do not assume that capitalization is correct. We recover capitalization in the trusted mentions that we identify. Names may be ambiguous. Lists may be noisy. Larger lists are generally better. The methods presented below for identifying trusted mentions of known names can also be used to find trusted mentions of new names, but this is beyond the scope of this paper.

Given a seed set of entities and their types, and the pages they are mentioned on, we augment this data in two interacting ways: (1) by extracting high precision templates of the words surrounding the mentions and (2) by propagating name type information across web links.

2.1 Entity extraction with Factzor

Key to the success of our system was the use of a method we call "factzor" for unsupervised extraction of facts from the web. Factzor finds templates that can be used to identify high quality entity mentions of known type. Factzor starts with a seed set of entities of known type and (optionally) relations among them. As mentioned above, we use entities and facts extracted from sources such as Wikipedia and IMDB. We then search the web for occurrences of the names of entities. Recurring patterns ("templates") in the text around the names are extracted, filtered, (optionally) generalized, and then used to extract more facts or, in our case, entity mentions of known type.

2.1.1 Mention extraction via lists

First, we identify candidate lists in web documents. For simplicity, we consider only two kinds of lists: rows or columns in HTML tables, and bulleted HTML lists. We consider the complete plain-text contents of a cell (<TD>) or a list element (<LI>) to be a candidate name. We chose HTML-formatted lists for a few reasons. First, they are very common, especially HTML tables, which are widely used both in hand-written and auto-generated web pages. There are many kinds of entities which appear in auto-generated web pages, backed by databases, which almost never occur in hand-written comma-separated lists. These include entity types such as song titles and product names. Secondly, they provide clear delineation of names without any text processing, which makes internationalization much simpler. Third, we observed that they were generally longer than comma-separated lists, and so are more productive.

Once a candidate list has been identified, we look for all elements that exactly match (modulo capitalization) names from the seed list. For lists with enough matches (based on a manually-defined threshold of both number of matches, and proportion of matches, in order to avoid matching very-long lists with only a few elements, which could be random), we calculate its type distribution. This is a type-hierarchy-aware measurement of the distribution of entity types. If there is a single dominant type, the list is deemed to be consistently-typed. All mentions of compatible known

names in the list are stored as trusted mentions of that type (and its parent types).

Note that lists do not have to be perfectly consistent; this allows for noise and incompleteness in the seed list.

2.1.2 Mention extraction via templates

A list on a web page provides a context in which type disambiguation can be done easily and with high reliability. Another approach is to look across pages for contexts which are highly predictive of type. One common source of such contexts are template-based web sites; here, the idea of cross-page contexts is directly analogous to a list in an automatically-generated table on a page. Within one web site (usually delineated by a single domain name), there are many templates which are type specific. (See Table 1 for examples.) The other type of context is a more "natural" textual context which is highly predictive of type; these are not domain-specific, but would be expected to be applicable to any text. We used a template-based learning approach to discover both kinds of templates.

The key steps in our template-based extraction are shown in Figure 1 and described below in more detail.

1. Find seed terms on the web by exact string match.
2. Extract context of three tokens on each side of each term found.
3. Generate all possible shorter templates by trimming.
4. Filter templates based on specificity, frequency and diversity: Keep templates with at least 80
5. Match templates against the web to find trusted mentions.

First, we find every occurrence of each seed name on every page in the corpus. For each occurrence, a candidate template is created, using a naive fixed context of three tokens on either side. In addition to the sequence of tokens, each template stores metadata about its provenance: its entity type, the URL upon which it was found, and exactly which seed name was found. This metadata is aggregated across the corpus to provide occurrence statistics for each template.

At this stage, there are many long candidate templates, which are generally very sparse. Templates with such a large context (six tokens) will usually be too long to collect meaningful statistics. The desired qualities of a template depend upon the exact task (more below), but can be expressed as:

- *specificity:* How tightly this template is associated with a specific entity type, as measured by the conditional probability, $P(type|context)$.
- *frequency* How often this template occurs.
- *source diversity:* How many different sources (hosts, domains, pages, books, ...) a template was found on. The wider a template's source diversity, the more likely it is to be found again on other, different sources.
- *name diversity:* The more different names appear in a template, the more likely it is that other good names will appear in it. Conversely, many high-frequency, low-diversity templates, such as boilerplates do not generalize well.

Table 1: Sample Factzor Templates

running back [athlete], who
(c) 2006 [organization] All Rights
episodes of " [tv program] , "
Prime Minister [prime minister], who
, and [drummer] on drums
To celebrate [holiday], we
won the [award] for best

- *template makeup:* The number of characters, number of words, and number of words on each side.

In this paper, we are interested in finding occurrences of known names in trusted contexts for use as training data. For this task, specificity is the most important single quality; we would like to be quite confident that an occurrence is correctly typed. Templates are used as a filter, rather than as an extraction mechanism. In this context, even a relatively infrequently-observed template (e.g. only occurring tens or a hundred times) is worth using.

To find a good set of templates, we begin with the huge, sparse set of templates gathered from all occurrences of known names. From each template, we generate each possible shorter template that fulfills our template-makeup restrictions. Occurrence metadata for each of these trimmed templates is aggregated from all original templates, and then the templates are filtered on the basis of specificity and diversity. This process starts with over a billion sparse templates, and after selection yields several million shorter, denser, yet still highly type-specific templates.

Finding consistently-typed mentions this way is computationally cheap. All that is required is the same name-finding and template-generating procedures as were used to generate the templates in the first place. This time, the generated templates are checked against a whitelist. The list can be very large (millions of templates), but the list is easily handled; We just store the templates as strings in a hash set.

A variety of template forms can be used, ranging from exact literal matches on tokens to powerful regular expressions. We match templates literally, requiring an exact match of all tokens to a known template. We define tokens very liberally, to include, for example, punctuation. For example, the following template recognizes people

. [person] was born on

Table 1 shows some typical templates. Using exact matches, without gaps, has the advantage that one need not determine in advance the exact boundaries of the mentions, so one can learn without a list of terms. Of course, it has the disadvantage that templates are highly specific, and do not generalize as well as they might. Note that *[person]* in the above example is delimited by the period (from the preceding sentence) and the verb *"was"*. Similar features are often used in supervised term extractors using models such as conditional random fields.

The resulting templates are then aggregated and filtered. "Good" templates have several features: They occur frequently, they have a diversity of different entities in them, and they have high specificity for entities of the desired type. For example, in the templates

. [XXX] was born in and *was born in [YYY].*

XXX is almost always a person, while YYY could be a place, a time period such as a year, month or season, or even some other non-entity phrase (*"a hurry"*).

The factzor approach has several advantages. Simultaneously looking for many entity types provides good measures of which templates are specific or non-specific. If one only knows about people, all entities showing up in templates are either people or unknown, rather than being known to be locations, animals or movies. Using a literal sequence of tokens provides patterns that are language independent; they can be used in Chinese as well as English, and they work on HTML tables as well as prose. Patterns can either be generic to the entire web, or specific to a given domain, like IMDB or Wikipedia.

We start with a library of some 5 million entity names of known type. Searching the web for occurrences of these entities yields about 4 billion matches, where each match represents a raw template. Keeping templates that are one, two, or three tokens in length to either side of the entity yields 16 billion unique templates. We keep frequent (greater than 50 occurrences), diverse (at least 50 different names), and specific (over 80%) templates. We allow templates to contain as few as four tokens in total, with at least one token on each side of the name. Further, we require that templates have at least eight characters in total, to exclude templates that are completely composed of punctuation. This yields around half a million site-specific templates, which are kept and used to find trusted mentions.

Since our goal is to provide a training set for a machine learning model, not to recognize all mentions on the web using the templates, high precision and good diversity are important, but high recall is not needed. Combining mentions from tables and templates, in total we found 65 million distinct trusted mentions. These mentions have an accuracy, as judged by inspecting a small sample of them, of about 99.9%; One error was found in a random sample of over 1,000 mentions.

Looking for occurrences of the selected templates on the web gives new mentions of both known and previously unknown entities. For our purpose, the key point is that a seed list of entities, many with names which can refer to entities of different type, have been converted into high precision (but low recall) mentions labeled with their correct type. These are then extended, as described below, and used as training data for our type prediction models.

2.2 Training Set Extension

The initial set of "trusted mentions" is extended in two ways. First, it is assumed that all occurrences of a name on a page are of the same type, so the type label is extended to all mentions of the entity on the same page. Secondly, given a set of pages that are contain entities of known type, e.g. pages that contain a trusted mention from factzor, we find pages that point to those pages or are pointed at by HTML links in those pages. More formally: we propagate each seed mention with name N and type T along every inlink and outlink to pages P'. Assign type T to all mentions of N on each of those P'. If there is a conflict – P' was one link away from two different seed pages for N, but with different types T on each – then drop N as a name of known type for P', otherwise keep it. This gives a much richer set of labeled mentions. Of course, not all of the extended labels generated are correct, but we show below empirically that using this expanded set improves performance.

In summary, the use of factzor as described in Section 2.1 produced approximately 65 million trusted mentions. We found an additional 8 million trusted mentions by searching for names that had four or more occurrences of known associated facts (e.g., date of birth, nationality, name of wife, etc.) in a large (500 token) window around them. By assuming a single sense per page, and trusting all mentions of the same name on the same page as a trusted mention, this is expanded to 216 million. By propagating one page through the web graph, and keeping unambiguous pages, our total training set was increased to 475 million trusted mentions. These served as our training set of mentions labeled with types.

Even though the trusted mentions found using factzor need not be "typical" of all entity mentions on the web, extending that set using one-sense-per-discourse and link-propagation results in a much more diverse training data set. Thus by matching a single, formulaic occurrence of an entity name (e.g., in a page header), we can easily obtain many more mentions (e.g. in people's comments/reviews of a movie).

2.3 Non-entities

We also needed to recognize which potential entities were, in fact, not entities. Automatic extraction methods invariably find some potential entities that should not be labeled as entities. Phrases such as "Privacy Policy" or "Click here" end up fitting some extraction template and being added as people or companies. For the final labeling of the web, we generate candidate entities to label by taking the union of a person recognizer, a place recognizer, and pattern matching against all entities in our database. Although these incorrectly extracted entities constitute a very small fraction of all the extracted entities, they have a disproportionate effect on the mentions selected for labeling, since they are often very frequent. Similarly, a movie called "September" or "2000" or a video called "Athletic Department" leads to an enormous number of spurious mentions being labeled. We thus filter out entity names which are all digits or are a single dictionary word. We also build a separate model for predicting which putative mentions are, in fact, not entities at all. Training data for "non-entities" is generated by looking for mentions which are composed entirely of dictionary names. To avoid labeling vast numbers of books and movies as "not entities", we keep as entities those mentions whose capitalization on the web is found to reliably (greater that 50% of the time) match the capitalization of the entity as in the database.

2.4 Feature Generation

For each mention of a term on a web page, a variety of features were generated and used to train a maximum margin classifier.

The web page containing each mention was tokenized, and its URL and domain were extracted. The domain of the URL on which the mention occurs was used as a feature, as were the tokens in the mention itself, and each of the three tokens immediately before and after the term. These features are position-specific; a colon immediately to the left of a mention is not the same as a colon one token further left. Some URLs had been labeled for other purposes as

falling into a number of approximate categories similar to those used in the open directory project (www.dmoz.org) or the Yahoo directory (dir.yahoo.com); these categories were also used as features.

A set of base entity-predictors were also run over the page. These included a person tagger, which uses a large dictionary of first and last names and a pattern matcher, a place tagger, which similarly uses a large dictionary of roads, cities, counties, states and countries, and a set of more generic annotators that do pattern matching against lists of companies, movies, and books extracted from sources such as IMDB and Wikipedia. (These latter entities were the same ones used as seeds for factzor.) The base predictors are far from being error free. For example, the name recognizer predicts that "Beverly Hills" will be a person, but that "Hitler" and "Jun Sun" are not people. (The name recognizer assumes that names have at least two tokens, and rejects mentions composed entirely of dictionary words, "Jun" and "sun" are both in the dictionary, since "Jun" is a common abbreviation of the month "June". A given mention being classified may have either an exact match or a partial overlap with one or more of the base type predictors, all of which are used as predictors. For example, schools and companies often have partial overlap with places. The relative proportions of predictions of each type on the web page of the mention and overall on all pages in its domain were also used as features.

We also checked for the frequency of each token in the mention in each of the word lists. For example, the term "Toyota Corporation of Japan" would score matches against both the list of Companies, where "Toyota" and "Japan" show up several times and "Corporation" shows up many times, and against lists of books, movies, etc.

One concern was whether including such specific features as the name of the term and the web site it was taken from would lead to overfitting, and hence reduce prediction accuracy on out-of-sample pages. In practice, it turns out that such overfitting does not occur, and models learned with a combination of both specific and general features work well.

3. MACHINE LEARNING

Training a model with dozens of categories, hundreds of millions of observations, and millions of features requires a particularly efficient algorithm. We used a two step method, first doing univariate screening, and then using an online perceptron-style maximum margin method to estimate model parameters. In order to capitalize on the common model structure across different entity types and subtypes, an error correcting output code multi-label classification method (described below) was used. Both screening and model estimation were designed to run in the map-reduce [11] data parallel architecture used at google and in open source software such as Hadoop (http://wiki.apache.org/lucene-hadoop/).

3.1 Feature Screening

Since the features used include tokens in and near the mentions being typed, we have many millions of features. Feature selection is critical, as much for efficiency as to avoid overfitting. Even given a highly efficient online learning method, estimating dozens of models with millions of features can be prohibitively slow.

Our goal is not to include all features that are statistically significant, but instead to include as many of the *best* features as we can afford. Doing feature selection during model estimation was found to be relatively slow compared to screening, so we first selected the "best" features, and then estimated models using the selected feature set. Because the major cost of including a feature in a model is reading the data from disk, we store the same set of features for all models.

The benefit of a feature is a function of both how frequently it occurs (extremely rare features are not helpful), and how highly correlated it is with any of the type labels. These properties are well captured by a t-statistic on feature/label correlation:

$$t_{jk} \sim (\mathbf{x_j}'\mathbf{y_k} - \bar{\mathbf{x}}_j\bar{\mathbf{y}}_k)/|\mathbf{x}_j|_2|\mathbf{y}_k - \bar{\mathbf{y}}_k|_2 \quad (1)$$

$\mathbf{x}_j$ and $\mathbf{y}_k$ are vectors of n observations of a feature j and of a label k. We subtracted off the mean frequencies of each label $\mathbf{y}_k$ for better resolution. If $t_{jk} > \theta$ for any label k, then the feature j is retained in the model. The threshold θ is chosen to give as large a training set size as can reasonably be handled, since that still involves screening out features which are statistically significant.

Screening is easily done in a map-reduce environment by randomly distributing the observations over many computers and computing components of the t-statistic on each machine for each subset of the observations. For example, the inner product $\mathbf{x}'\mathbf{y}$ is easily computed by computing the inner products of each subset of the observations on separate machines (the "map" phase) and then summing the resulting partial sums (the "reduce" phase).

Screening trimmed down the multiple millions of features down to 300,000 features. Typical observations had between 10 and 100 nonzero features.

3.2 Perceptron Algorithm

Models were estimated using the "passive aggressive" version of an additive perceptron model [9]. The goal is to select the coefficients $\mathbf{c}_k$ in the models for each label k

$$y_k^{pred} = sgn(\mathbf{c_k} \cdot \mathbf{x}) \quad (2)$$

to minimize the hinge loss prediction error:

$$\sum_{t,k} max(0, 1 - y_k^t \mathbf{c_k} \cdot \mathbf{x^t}) \quad (3)$$

where the true label y_k is either 1 or -1, and the summation ranges over the n observations (superscript t) and the K labels (subscript k).

The idea behind the perceptron algorithm is simple: if the prediction is correct, no change is made; otherwise, the coefficients are changed by adding just enough of the new observation x to make the new prediction be correct. I.e., the update is:

$$\mathbf{c_k} := \mathbf{c_k} + \alpha y \mathbf{x} \quad (4)$$

where $\alpha = max(0, 1 - y_k \mathbf{c_k} \cdot \mathbf{x}/|\mathbf{x}|^2)$ is the loss (if any) for the point $\mathbf{x}$ divided by the norm of $\mathbf{x}$ squared, and can be derived from a dual formulation[9]. (For notational clarity, we have dropped the superscript t indicating the observation.)

In order to get sufficient speed, we again used the map-reduce architecture [11]. The training data are randomly divided into different subsets ("shards"), and a separate model

Table 2: Type labels used

airport, animal, athlete, bird, chemical, chemical compound, company, dinosaur, event, festival, fish, game, mathematician, music recording, organization, person, place, plant, political party, politician, product, record label, school, scientist, soccer player, spider, team, tree, vehicle, video game, winery, wrestler, non-entity

is learned in each shard (the "map" phase). The coefficients of these models are then averaged (the "reduce" phase). Since the models are linear, averaging the coefficients of the different models gives the average of the model scores, giving an average of the log-odds ratios of the predictions.

3.3 Use of Class Hierarchy

We initially tried predicting each entity class label as a separate classification (i.e., separately predict for each type label whether or not it should be there). However, such individual classifications fail to take advantage of the fact that there are strong correlations (both positive and negative) between the coefficients in the different models. Features which positively predict one class will necessarily (but more weakly) negatively predict non-overlapping classes, and will positively predict classes that are supersets or subsets.

More subtly, correctly handling the hierarchical structure of the classes is particularly important to make accurate predictions on small subclasses. In the experiments we ran, we used just over thirty entity classes, which are very imbalanced; classes at the top of the hierarchy of classes are large (e.g., people), but those at the leaves are often much smaller (e.g., professional wrestlers), and we were interested in also doing fine-grained classification well. Classifying each category vs. all others worked reasonably well for the large classes, but the approach was particularly bad at classifying small classes, especially subclasses. For independent predictions, the model for each subclass was trained with positive examples being subclass instances and negative examples being examples of conflicting types, i.e., anything not on the same path in the hierarchy. Thus there was no "actor vs. person" model. In this setup, "person" always has a larger prior (more training examples) than "actor", and thus has roughly the same set of negative training instances (modulo sub-types of person, which are negative examples for actor). The result is that when independent models are used, subclasses were predicted less often than they should have been.

To overcome this problem, we took advantage of the type hierarchy to decompose the problem into multiple classifications. We train a model for each parent type (an internal node in the hierarchy), determining the subtype. We changed the structure of the hierarchy so that there was a leaf node corresponding to each type; for a parent type such as "person", this meant adding a new child called e.g. "person-leaf" below "person". Classification starts at the root and continues until a leaf type is chosen.

The hierarchical model is also more statistically efficient. Since subtypes of "person" all behave primarily like "person" but have their own specific differences, an "actor vs. all" and a "politician vs. all" model are both mostly learning a "person vs. all" model, with fewer degrees of freedom left for representing actor-specific features, etc. Learning a single model for all people, and then learning differences between subtypes works much better.

Several other researchers have also looked at hierarchical classification of documents, generally in a setting where documents are forced into the leaves of a hierarchy. (See. e.g., [6, 15, 26] and the references therein.) The goal of most such hierarchical classification has been speed, although it has also been used to reduce the number of features selections [21].

Using a hierarchical model also had the advantage that it decomposed the classification into smaller matrices, so the codes were more efficient; it gave more sensible partitions of the classification space. In practice, hierarchical models were faster to run since fewer models were run, and most models had more ignored labels. Using a hierarchy also made it *much* more likely to actually classify entities into a fine-grained subtype, even if it was relatively small, since it was easier for the subtypes to be relatively balanced (e.g., fish vs. dinosaur vs. mammal) than (e.g., fish vs. everything else). The net result was increased accuracy everywhere, but hugely increased accuracy for subtypes. (For example, accuracies on fish increase from a precision/recall of 50.0%/0.03% to 43.9%/61.9%; See the Results section for details.)

3.4 Error Correcting Output Codes

Regardless of whether we were doing hierarchical classification or not, we have a multi-class classification problem with many highly imbalanced classes. We used Error Correcting Output Codes (ECOC) to improve classification performance[12]. Error correcting output codes use a code vector to represent each of the labels. This can be represented as a table of 0s and 1s where the rows are labeled with the type labels and the columns are the new bit values for their output codes. The new codes are chosen maximize the minimum Hamming distance between any pair of codes (maximizing the "error correcting" capability), while minimizing the correlation between columns in the new representation (avoiding redundancy in the code.) For problems with up to five classes, all codings can be generated exhaustively, Since our problem is larger, we generated many codings randomly and selected the best.

For example, we might generate the following coding, where columns correspond to the classes we learn models for (ten in this example):

```
person:        1111100000
scientist:     1010101010
physicist:     1101100011
chemist:       0000100110
...
```

Classifications are learned by building separate models to predict each of the columns (bits of the code) We get a prediction (0 or 1) for each of those 10 classes, which we pack into a binary vector with 10 elements. We then compute the hamming distance between that prediction vector and the binary vectors in the coding matrix. The type label with the smallest hamming distance is then selected. So if our 10 classifiers gave the binary vector 1110101010 (which is only one edit away from scientist above), then we would predict the instance as "scientist" (and, by inference also "person" because person is a supertype).

We compared a simple ECOC as described above, which

just treated all types as equivalent for the purposes of generating the ECOC code and hence in which each output-code was allowed to be (and typically was) a mixture of all types, with a hierarchy-aware version. The hierarchy-aware ECOC, which gave better performance, used a different output-code at each level of the hierarchy. For a simple hierarchy of person with subtypes scientist and celebrity where scientists are physicists or chemists, and celebrities are actors or musicians, we would thus generate three different output codes: one for the person-LEAF vs. scientist vs. celebrity classification, one for the scientist-LEAF vs. physicist vs. chemist classification, and one for the celebrity-LEAF vs. actor vs. musician classification.

Visually, that means the coding matrix becomes block-diagonal:

```
person-LEAF:       0000________
scientist:         1100________
celebrity:         1010________
scientist-LEAF:    ____0011____
physicist:         ____1010____
chemist:           ____0110____
celebrity-LEAF:    ________1111
actor:             ________0001
musician:          ________1010
```

where _ means "ignore instance"; that is, model number 12, the last one, will not be trained on instances labeled just as "person" because the top row has a _ at the end.

When classifying, we start at the top of the hierarchy – with person – and perform the person-LEAF vs. scientist vs. celebrity classification using hamming distance just as we did above. If person-LEAF wins, stop. Otherwise, if e.g. scientist wins, recurse down and perform the scientist-LEAF vs. physicist vs. chemist classification. Etc.

We used similar length codes in both cases, so memory usage was about the same, but with the hierarchical code both training time and classification time were improved because there were 0's in the coding matrix (meaning we could ignore many instance-model pairs during training) and because at classification time we walked down the hierarchy so we didn't have to run every classifier on every instance.

Using a standard ECOC encoding reduced the error significantly, and using a hierarchical output coding gave further benefit. We used a 100-code ECOC model for a 33-label problem. This meant a three times increase in the size of the memory used for storing models during classification, and thus a three-fold decrease in classification speed, but was waorth it for the increased accuracy.

3.5 Prediction Resolution

After training, we have a model for each entity type, each of which gives a binary prediction. Prior to thresholding it, in fact, gives a number which typically lies between -1 and 1, which conveys useful information about strength of belief in the prediction.

Since there are potentially multiple predictions for each mention, and there are often multiple mentions with the same name on each page, these individual mention/type predictions can be combined to give a final prediction for each mention. Often one mention will contain another, such as "Philadelphia Electric Company". In this case, we select the longest annotation and ignore the ones that are contained within it. In the case of partially overlapping annotations, if they share the same types, we take their union. ("New York" as place overlapping "York City" as place becomes a single annotation for "New York City" as place.) If they overlap partially but do not share the same types, we do not attempt to resolve the conflict.

We also experimented with voting methods that take advantage of the fact that usually all mentions sharing the same name on a page are of the same type. None gave a large difference in performance, but we did choose to label all mentions of a term on a page with the most frequent label.

Table 3: Test Set Precision and Recall of the hierarchy-aware ECOC model on the "basic" trusted mentions and the set expanded by being "link-propagated". Tested on "basic" held-out mentions and on the " ambiguous" subset of them.

train on	on basic prec. / recall	on ambiguous prec. / recall
basic	94.7 / 93.7	85.7 / 88.2
link-propagated	93.3 / 95.7	82.7 / 89.6

4. RESULTS

For evaluation, we divided the entity mentions for which we had "trusted mentions" randomly into two groups, training set names and testing set names. Different training and testing sets were constructed for the two sets of mentions by either using trusted mentions as extracted by factzor, or by extending the set of data by using link propagation, as described above.

Precision and recall as a function of the training and testing sets used are given in table 3. In all cases, the results shown are for test set mentions that have no overlap with the training set. Extending the training set by link propagation ("lp") gives a two percentage point increase in recall at a cost of a 1.4 point decrease in precision. Results restricting ourselves to only the ambiguous terms (those with more than one possible contradictory type label, as determined by their presence on name lists of given type or as predicted by the person name predictor or the geographic location predictor) are qualitatively similar but, of course, have lower accuracies. For the basic data set, 18% of the 216 million instances were ambiguous, while for the set extended with link propagation, 32% of 475 million instances were ambiguous.

The bottom line is that there is not significant overfitting, and that we get around 94% overall accuracy, and around 85% accuracy on words which are ambiguous as to their type.

We also examined the benefit of using the type hierarchy and the error correcting output codes. Table 4 compares the following models: *independent models* (one model per type, highest score wins), *simple ECOC* (ECOC with no hierarchy-awareness) *hierarchy aware* (no ECOC, but hierarchy-aware, with one model per type, but only trained on sibling instances) and *hierarchy-aware ECOC*, a combination of the two preceding methods, which we found to give the best performance, on both type prediction (only for mentions which are entities), and on predicting whether a mention will be an entity or not.

Both ECOC versions have much better typed performance

Table 4: Test Set Precision and Recall of different models

Training Method	on basic	on ambiguous
for entity type prediction	**prec. / recall**	**prec. / recall**
independent models	84.9 / 86.2	49.6 / 52.0
simple ECOC	86.5 / 87.4	76.5 / 76.8
hierarchy-aware	83.6 / 84.9	44.4 / 46.3
hierarchy-aware ECOC	84.3 / 85.1	73.5 / 75.0
for entity/non-entity classification		
independent models	93.2 / 20.2	97.4 / 5.9
simple ECOC	78.0 / 37.0	61.5 / 55.9
hierarchy-aware	95.3 / 21.0	94.9 / 14.9
hierarchy-aware ECOC	68.5 / 40.1	67.8 / 40.3

Table 5: Test Set Precision and Recall for different types; The effect of using hierarchy-aware ECOC

type	independent prec. / recall	hierarchy-aware ECOC prec. / recall
company	48.6 / 6.9	42.6 / 62.4
fish	50.0 / 0.03	43.9 / 61.9
person	98.0 / 49.4	95.2 / 90.8
place	89.0 / 58.5	86.0 / 86.3

than the independent models on the ambiguous names, which is the more difficult and interesting set, and vastly better recall on the entity/non-entity predictions. The benefit of the hierarchy-aware ECOC over the non-hierarchical ECOC is less clear. However, the significant reduction in the compute time needed when using the hierarchy was sufficient to convince us to use it. Table 5 gives some example improvements on specific types from the independent models to the hierarchy-aware ECOC.

4.1 Feature benefit

The *benefit* of each feature was estimated by the percentage of non-zero occurrences of the feature times the magnitude of the feature's coefficient divided by the standard deviation of the non-zero feature values. The benefit of a feature class is the sum of the benefits of the features in that class.

Inspecting the most beneficial features in predicting type labels (see table 6), we see that document category is often useful: movies show up on pages labeled "entertainment" or "movie" (as do people). Case signatures also have a strong signal. School names tend to be a sequence of three capitalized letters, companies tend not to be one or two words capitalized (relative to other types), but do tend to be a single word with internal capitalization (e.g. DealTime or JetBlue).

In general, being labeled with one type tends to indicate that a mention is not of another type. I.e., mentions labeled "company" or "place" are mostly not movies. Note that the weights on these relationships are asymmetric; the probability of a person being labeled a place is not the same as the probability of a place being labeled a person. It is this correlation structure, in part, that the ECOC training is benefiting from.

Table 6: Examples of the most beneficial features (positive and negative) for feature class for predicting different labels

Type label	Sign	Feature class	Feature value
school	+	document category	vehicle licensing
school	+	document category	automotive
school	+	left 0 tokens	Unified
school	+	right 2 tokens	-
school	+	case signature	Aa Aa Aa
school	-	type-labeler	album
school	-	type-labeler	movie
school	-	type-labeler	company
school	-	type-labeler	place
school	-	person-labeler	person
company	+	document category	business
company	+	document category	news
company	+	left 0 tokens	friend
company	+	left 1 token	a
company	-	case signature	Aa Aa
company	-	case signature	Aa
company	-	document category	computers & electronics
movie	+	document category	entertainment
movie	+	right 2 tokens	)
movie	+	document category	movies
movie	-	type-labeler (pm)	album
movie	-	labels in domain	album
movie	-	type-labeler	company
movie	-	type-labeler	place
movie	-	case signature	Aa
movie	-	case signature	Aa a

5. RELATED WORK

Named entity recognition has been widely studied, mostly using supervised learning methods to label person, places and organizations in news [17, 7, 27], or genes and diseases in scientific abstracts [16]. (See [23] for a recent survey.) These methods have relied on extensive (and expensive) hand-labeling of mentions of the entities. This labeled data, and a set of automatically extracted features [22, 20], is then used to train models such as maximum entropy [3], or recently, conditional random fields. There are also many commercial named entity recognition (NER) systems, such as AeroText, Rosette Entity Extractor (REX), ClearForest, Inxight, PalyAnalyst, and SRA NetOwl. These typically use significant numbers of hand-coded rules, which help them to get reasonable performance for limited numbers of entity types on well-circumscribed corpora, such as news articles. Open source NER systems, such as MinorThird and Annie, tend to be more statistically based, but have been trained on very specific entity types and corpora such as gene names and diseases in medline abstracts, teams and players in soccer game descriptions, or corporations and people in financial reports.

More recently, there has been some work on unsupervised entity extraction from the web, typically with a goal of fact extraction [1, 18, 25, 13, 2, 5, 25]. (Such work actually goes back over a decade[4].) A variety of different templates forms have been used, including regular expressions and Hidden Markov Models (HMMs) [14]. Factzor is at the extreme limit of this spectrum, as it uses millions of very simple templates (exact literal matches), which are rapidly matched in parallel, in contrast to more complex templates consisting of regular expressions or HMMs. Our goal here also differs from the above work, as we use *factzor* for generating training data for a supervised learning method, rather than for extracting facts.

There have been several previous methods for incorporating unlabeled data in training a named entity classification model. Collins and Singer [8] use unlabeled data directly through co-training; they rely upon POS-tagging and parsing to identify training examples and features. Kim et al. [19] start with a larger seed list than Collins and Singer, and learn common contextual features that occur unambiguously with a single category. They then use these contextual features to find more entities to use as training examples. They report a relatively small increase in training set size from 3,899 to 4,504 instances in a corpus of 37,831 candidates. Also, they explicitly exclude homonyms, so all identified entities are of known type, which would not be useful for the web, where (in our experiments) between roughly one third of the entity mentions are ambiguous as to type.

Recent work on unsupervised information extraction from the web (a.k.a. "Machine Reading") [18, 13, 2, 25] is similar in many ways to our factzor system for finding entity mentions. However, our goal is not to extract lists of names of a given type as Know-It-All [18] does, but rather to get trusted mentions to serve as training data to build an accurate, high coverage classifier to predict the type of all names on the web. Thus, although we share some techniques like list extraction and pattern learning with Know-It-All, we differ both in the specific methods used (partly due to our need for scalability) and in our use of link-propagation to extend the sets of mentions.

In related work, [24] extract lists of entity names (they call the lists 'gazetteers'), and then use heuristics to determine the type of a given mention (e.g., is "May" a month or a person.) They get accuracies of 70% to 80% on MUC evaluations. We have found that such heuristics, although sometimes very helpful, do not work on the more fine-grained distinctions such as sub-types of people. The company resulting from the above work, YooName, classifies nine major entity types. Some of their categories (e.g. vehicles, art and media) have useful subtypes. They do not, however, find subcategories of person by profession.

The related problem of named entity disambiguation has been attacked by using data extracted from Wikipedia. [10] extract 1.4 million entities, finding 540,000 pairs, 38 million mentions, and 139,000 "category tags," and disambiguate mentions on Wikipedia and in news articles. We address a somewhat different problem, looking over an order of magnitude more entities (we use 22 million names), and a much larger and broader corpus (the web). We find, for example, many people, organizations, schools, addresses, and companies not mentioned in Wikipedia, and our goal is to label mentions of these entities with a relatively small set (hundreds) of well-defined type-tags, suitable for displaying in a user interface showing, for example, grouping by type entities related to a given entity. The 139,000 Wikipedia category tags are useful as features for disambiguation, but are far too fine-grained, overlapping, and sloppy for the user interface we envision. The mentions we find show up in a wide variety of local contexts, including titles, lists, and tables, as well as running text. The capitalization on the web is fairly erratic. Words are often in all caps ("GERMANY") or lower-cased ("germany"); we find and label these mentions in spite of their "incorrect" capitalization. Typical web pages are also much less dense in entities than Wikipedia pages.

6. CONCLUSIONS

The classic named entity recognition problem studied in academia assumes a large, high quality set of labeled entities. This view of the world works well for relatively small numbers of entity types (people, places, organization, genes and diseases) in relatively uniform, high quality corpora. It does not generalize well to hundreds of entity types in dozens of languages in multiple types of corpora, such as are found on the web.

The key to broad coverage entity recognition is the semi-automatic generation of training data, and the simultaneous labeling of many different entity types. The first insight is that there are many good quality lists and databases of entities which have type annotations. Wikipedia is available in many languages. Starting from these entities and their type lists, we can find templates that identify reliable mentions of many entities of many different types. Simultaneously learning templates for all different types provides a major benefit, as it allows us to assess the quality of the proposed templates; the different types provide negative examples that would otherwise be missing. Using language-independent template formats allows us to extract data from lists and tables as well as from free text, and to do so in many languages. Since we are fitting over the entire web, we use specific templates and exact string matching, rather than regular expressions, which would be more general, but slower. Using literal templates has the further advantage that the boundaries of new terms can be identified, since the template delimits the mentions.

Once a large (hundreds of millions) set of labeled mentions is extracted, predictive models can be learned. These models use a wealth of contextual features to disambiguate names that have multiple types. Since many models with different feature sets and parameter settings were estimated during development, fast model estimation was key. In an online learning algorithm, the slowest step is reading the data. This is easily parallelized across many machines in a map-reduce data-parallel architecture. Furthermore, since the models for all types share the same features, the features need only be read once per mention.

Simultaneously tagging many types of entities simultaneously presents additional opportunity to gain predictive power. The predictions of different entity types are by no means uncorrelated (a wrestler is a person, and a city is not). Making simultaneous predictions of all labels using error correcting output codes (ECOC) gives a significant improvement in prediction accuracy. Since terms like "Beverly Hills" might be tagged both as a person and as a location by our "base taggers", the classification model learns how to use other contextual features to discriminate between the labels.

The combination of using lists and templates to extract a high quality training set of labeled "trusted mentions", extending this set to other mentions on the same page and (if they are unambiguous) on linked pages, and online learning using hierarchy-aware error correcting output codes gives a broad coverage named entity recognizer which works well on dozens of entities across the entire English web. We are currently extending this work to entity resolution, where entities are not only typed (This "Paris" is a person), but also resolved (This "Paris" is "Paris Hilton".)

7. REFERENCES

[1] E. Agichtein. *Extracting Relations from Large Text Corpora*. PhD. Thesis, CS, Columbia, University, 2005.

[2] Michele Banko, Michael J Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *IJCAI*, 2007.

[3] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proc. 6th Workshop on Very Large Corpora (VLC)*. Association for Computational Linguistics, 1998.

[4] Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB*, 1998.

[5] Razvan C. Bunescu and Marius Pasca. Using encyclopedic knowledge for named entity disambiguation. In *EACL, Trento, Italy*, 2006.

[6] Michelangelo Ceci and Donato Malerba. Classifying web documents in a hierarchy of categories: a comprehensive study. *J. Intell. Inf. Syst.*, 28(1):37–78, 2007.

[7] N. Chinchor, L. Hirschman, and D. Lewis. Evaluating message understanding systems: An analysis of the third message understanding conference (muc-3). *Computational Linguistics*, 3(19):409–449, 1994.

[8] Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *EMNLP/VLC-99*, 1999.

[9] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[10] S. Cucerzan. Large scale named entity disambiguation based on wikipedia data. In *The EMNLP-CoNLL Joint Conference. Prague*, 2007.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, 2004.

[12] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 1995.

[13] Doug Downey, Matthew Broadhead, and Oren Etzioni. Locating complex named entities in web text. In *IJCAI*, 2007.

[14] Doug Downey, Steven Schoenmackers, and Oren Etzioni. Sparse information extraction: Unsupervised language models to the rescue. In *ACL-07*, 2007.

[15] Susan T. Dumais and Hao Chen. Hierarchical classification of Web content. In *SIGIR-00*, pages 256–263. ACM Press, New York, US, 2000.

[16] A. Yeh et al. Biocreative task 1a: Gene mention finding evaluation. *BMC Bioinformatics*, 6, 2005.

[17] D. Miller et al. Named entity extraction from broadcast news. In *Proceedings of DARPA Broadcast News Workshop*. Herndon, VA, 1999.

[18] O. Etzioni. et al. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134, 2005.

[19] Jae-Ho Kim, In-Ho Kang, and Key-Sun Choi. Unsupervised named entity classification models and their ensembles. In *Proc. 19th Intl. Conf. on Computational linguistics*, 2002.

[20] Dan Klein, Joseph Smarr, Huy Nguyen, and Christopher D. Manning. Named entity recognition with character-level models. In *Proceedings the Seventh Conference on Natural Language Learning*, 2003.

[21] D. Koller and M. Sahami. Hierarchically classifying documents using very few words. In *ICML*, pages 170–178, 1997.

[22] James Mayfield, Paul McNamee, and Christine D. Piatko. Named entity recognition using hundreds of thousands of features. In *CoNLL*, 2003.

[23] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26, 2007.

[24] David Nadeau, Peter D. Turney, and Stan Matwin. Unsupervised named entity recognition: Generating gazetteers and resolving ambiguity. In *Proc. Canadian Conference on Artificial Intelligence*, 2006.

[25] B. Rosenfeld and R. Feldman. Using corpus statistics on entities to improve semi-supervised relation extraction from the web. In *ACL-07*, 2007.

[26] Aixin Sun and Ee-Peng Lim. Hierarchical text classification and evaluation. In *ICDM '01*, pages 521–528, 2001.

[27] Erik F. Tjong, Kim Sang, and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *CoNLL*, 2003.