Named Entity Recognition: A Short Tutorial and Sample Business Application

By Humphrey Sheil

Jan 15, 2015

Contents Print + Share This

Page 1 >

Humphrey Sheil, co-author of Sun Certified Enterprise Architect for Java EE Study Guide, 2nd Edition, demonstrates how an off the shelf Machine Learning package can be used to add significant value to vanilla Java code for language parsing, recognition and entity extraction.

This article was originally published at humphreysheil.blogspot.co.uk.

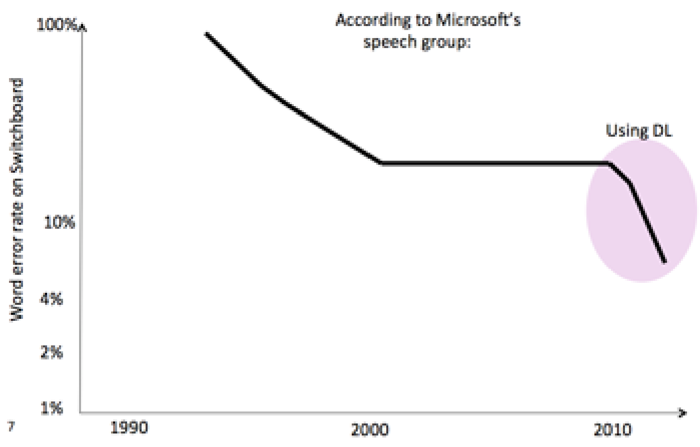
A latent theme is emerging quite quickly in mainstream business computing - the inclusion of Machine Learning to solve thorny problems in very specific problem domains. For me, Machine Learning is the use of any technique where system performance improves over time by the system either being trained or learning.

In this short article, I will quickly demonstrate how an off the shelf Machine Learning package can be used to add significant value to vanilla Java code for language parsing, recognition and entity extraction. In this example, adopting an advanced, yet easy to use, Natural Language Parser (NLP) combined with Named Entity Recognition (NER), provides a deeper, more semantic and more extensible understanding of natural text commonly encountered in a business application than any non-Machine Learning approach could hope to deliver.

Machine Learning is one of the oldest branches of Computer Science. From Rosenblatt's perceptron in 1957 (and even earlier), Machine Learning has grown up alongside other subdisciplines such as language design, compiler theory, databases and networking - the nuts and bolts that drive the web and most business systems today. But by and large, Machine Learning is not *straightforward* or *clear-cut*enough for a lot of developers and until recently, its' application to business systems was seen as not strictly necessary. For example, we know that investment banks have put significant efforts applying neural networks to market prediction and portfolio risk management and the efforts of Google and Facebook with deep learning (the third generation of neural networks) has been widely reported in the last three years, particularly for image and speech recognition. But mainstream business systems do not display the same adoption levels..

Aside: *accuracy* is important in business / real-world applications.. the picture below shows why you now have Siri / Google Now on your iOS or Android device. Until 2009 - 2010, accuracy had flat-lined for almost a decade, but the application of the next generation of artificial neural networks drove the error rates down to a usable level for millions of users (graph drawn from Yoshua Bengio's ML tutorial at KDD this year).

From the author of



Click to view larger image

Dramatic reduction in error rate on Switchboard data set post introduction of deep learning techniques.

Luckily you don't need to build a deep neural net just to apply Machine Learning to your project! Instead, let's look at a task that many applications can and should handle better mining unstructured text data to extract meaning and inference.

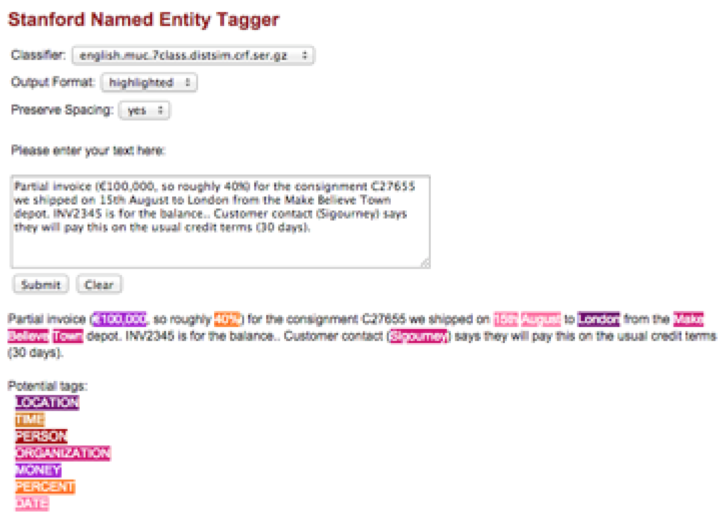
Natural language parsing is tricky. There are any number of seemingly easy sentences which demonstrate how much context we subconsciously process when we read. For example, what if someone comments on an invoice: “*Partial invoice (€100,000, so roughly 40%) for the consignment C27655 we shipped on 15th August to London from the Make Believe Town depot. INV2345 is for the balance.. Customer contact (Sigourney) says they will pay this on the usual credit terms (30 days).*”.

Extracting tokens of interest from an arbitrary String is pretty easy. Just use a StringTokenizer, use space (" ") as the separator character and you're good to go.. But code like this has a high maintenance overhead, needs a lot of work to extend and is fundamentally only as good as the time you invest into it. Think about stemming, checking for ',','.',';' characters as token separators and a whole slew more of plumbing code hoves into view.

How can Machine Learning help?

Natural Language Parsing (NLP) is a mature branch of Machine Learning. There are many NLP implementations available, the one I will use here is the CoreNLP / NER framework from the language research group at Stanford University. CoreNLP is underpinned by a robust theoretical framework, has a good API and reasonable documentation. It is slow to load though.. make sure you use a Factory + Singleton pattern combo in your code as it is thread-safe since ~2012. An online demo of a 7-class (recognises seven different things or entities) trained model is available at http://nlp.stanford.edu:8080/ner/process where you can submit your own text and see how well the classifier / tagger does. Here's a screenshot of the default model on our sample sentence:

From the author of


Stanford Named Entity Tagger

Classifier: english.muc.7class.distsim.crf.ser.gz

Output Format: highlighted

Preserve Spacing: yes

Please enter your text here:

```
Partial invoice (€100,000, so roughly 40%) for the consignment C27655
we shipped on 15th August to London from the Make Believe Town
depot. INV2345 is for the balance.. Customer contact (Sigourney) says
they will pay this on the usual credit terms (30 days).
```

Partial invoice (€100,000, so roughly 40%) for the consignment C27655 we shipped on 15th August to London from the Make Believe Town depot. INV2345 is for the balance.. Customer contact (Sigourney) says they will pay this on the usual credit terms (30 days).

Potential tags:
- LOCATION
- TIME
- PERSON
- ORGANIZATION
- MONEY
- PERCENT
- DATE

Click to view larger image

Output from a trained model without the use of a supplementing dictionary / gazette.

You will note that "Make Believe Town" is classified (incorrectly in this case) as an ORGANIZATION. Ok, so let's give this "out of the box" model a bit more knowledge about the geography our company uses to improve its' accuracy. Note: I would have preferred to use the gazette feature in Stanford NER (I felt it was a more elegant solution), but as the documentation stated, gazette terms are not set in stone, behaviour that we require here.

So let's create a simple tab-delimited text file as follows:

Make Believe Town LOCATION

(make sure you don't have any blank lines in this file - RegexNER really doesn't like them!)

Save this one line of text into a file named locations.txt and place it in a location available to your classloader at runtime. I have also assumed that you have installed the Stanford NLP models and required jar files into the same location.

Now re-run the model, but this time asking CoreNLP to add the regexner to the pipeline.. You can do this by running the code below and changing the value of the useRegexner boolean flag to examine the accuracy with and without our small dictionary.

Hey presto! Our default 7-class model now has a better understanding of our unique geography, adding more value to this data mining tool for our company (check out the output below vs the screenshot from the default model above)..

Code

```
package phoenix;

import java.util.ArrayList;
import java.util.List;
import java.util.Properties;
```

From the author of

```
import org.slf4j.LoggerFactory;

import edu.stanford.nlp.ling.CoreLabel;
import edu.stanford.nlp.ling.CoreAnnotations.NamedEntityTagAnnotation;
import edu.stanford.nlp.ling.CoreAnnotations.SentencesAnnotation;
import edu.stanford.nlp.ling.CoreAnnotations.TextAnnotation;
import edu.stanford.nlp.ling.CoreAnnotations.TokensAnnotation;
import edu.stanford.nlp.pipeline.Annotation;
import edu.stanford.nlp.pipeline.StanfordCoreNLP;
import edu.stanford.nlp.util.CoreMap;


/**
 * Some simple unit tests for the CoreNLP NER (http://nlp.stanford.edu/software/CRF-NER.
 * article.
 *
 * @author hsheil
 *
 */
public class ArticleNlpRunner {

  private static final Logger LOG = LoggerFactory.getLogger(ArticleNlpRunner.class);

  @Test
  public void basic() {
    LOG.debug("Starting Stanford NLP");

    // creates a StanfordCoreNLP object, with POS tagging, lemmatization, NER, parsing,
    Properties props = new Properties();
    boolean useRegexner = true;
    if (useRegexner) {
      props.put("annotators", "tokenize, ssplit, pos, lemma, ner, regexner");
      props.put("regexner.mapping", "locations.txt");
    } else {
      props.put("annotators", "tokenize, ssplit, pos, lemma, ner");
    }
    StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

    // // We're interested in NER for these things (jt->loc->sal)
    String[] tests =
        {
            "Partial invoice (€100,000, so roughly 40%) for the consignment C27655 we sh
        };
    List tokens = new ArrayList<>();

    for (String s : tests) {

      // run all Annotators on the passed-in text
      Annotation document = new Annotation(s);
      pipeline.annotate(document);

      // these are all the sentences in this document
      // a CoreMap is essentially a Map that uses class objects as keys and has values w
      // custom types
      List sentences = document.get(SentencesAnnotation.class);
      StringBuilder sb = new StringBuilder();

      //I don't know why I can't get this code out of the box from StanfordNLP, multi-to
```

From the author of

```
          // traversing the words in the current sentence, "O" is a sensible default to in
          // tokens to since we're not interested in unclassified / unknown things..
          String prevNeToken = "O";
          String currNeToken = "O";
          boolean newToken = true;
          for (CoreLabel token : sentence.get(TokensAnnotation.class)) {
            currNeToken = token.get(NamedEntityTagAnnotation.class);
            String word = token.get(TextAnnotation.class);
            // Strip out "O"s completely, makes code below easier to understand
            if (currNeToken.equals("O")) {
              // LOG.debug("Skipping '{}' classified as {}", word, currNeToken);
              if (!prevNeToken.equals("O") && (sb.length() > 0)) {
                handleEntity(prevNeToken, sb, tokens);
                newToken = true;
              }
              continue;
            }

            if (newToken) {
              prevNeToken = currNeToken;
              newToken = false;
              sb.append(word);
              continue;
            }

            if (currNeToken.equals(prevNeToken)) {
              sb.append(" " + word);
            } else {
              // We're done with the current entity - print it out and reset
              // TODO save this token into an appropriate ADT to return for useful process
              handleEntity(prevNeToken, sb, tokens);
              newToken = true;
            }
            prevNeToken = currNeToken;
          }
        }

        //TODO - do some cool stuff with these tokens!
        LOG.debug("We extracted {} tokens of interest from the input text", tokens.size())
      }
    }
    private void handleEntity(String inKey, StringBuilder inSb, List inTokens) {
      LOG.debug("'{}' is a {}", inSb, inKey);
      inTokens.add(new EmbeddedToken(inKey, inSb.toString()));
      inSb.setLength(0);
    }


  }
  class EmbeddedToken {

    private String name;
    private String value;

    public String getName() {
      return name;
    }
```

From the author of

```
  public EmbeddedToken(String name, String value) {
    super();
    this.name = name;
    this.value = value;
  }
}
```

Output

```
16:01:15.465 [main] DEBUG phoenix.ArticleNlpRunner - Starting Stanford NLP
Adding annotator tokenize
TokenizerAnnotator: No tokenizer type provided. Defaulting to PTBTokenizer.
Adding annotator ssplit
edu.stanford.nlp.pipeline.AnnotatorImplementations:
Adding annotator pos
Reading POS tagger model from edu/stanford/nlp/models/pos-tagger/english-left3words/engl
Adding annotator lemma
Adding annotator ner
Loading classifier from edu/stanford/nlp/models/ner/english.all.3class.distsim.crf.ser.g
Loading classifier from edu/stanford/nlp/models/ner/english.muc.7class.distsim.crf.ser.g
Loading classifier from edu/stanford/nlp/models/ner/english.conll.4class.distsim.crf.ser
sutime.binder.1.
Initializing JollyDayHoliday for sutime with classpath:edu/stanford/nlp/models/sutime/jo
Reading TokensRegex rules from edu/stanford/nlp/models/sutime/defs.sutime.txt
Reading TokensRegex rules from edu/stanford/nlp/models/sutime/english.sutime.txt
Oct 06, 2014 4:01:37 PM edu.stanford.nlp.ling.tokensregex.CoreMapExpressionExtractor app
INFO: Ignoring inactive rule: null
Oct 06, 2014 4:01:37 PM edu.stanford.nlp.ling.tokensregex.CoreMapExpressionExtractor app
INFO: Ignoring inactive rule: temporal-composite-8:ranges
Reading TokensRegex rules from edu/stanford/nlp/models/sutime/english.holidays.sutime.tx
Adding annotator regexner
TokensRegexNERAnnotator regexner: Read 1 unique entries out of 1 from locations.txt, 0 T
16:01:38.077 [main] DEBUG phoenix.ArticleNlpRunner - '$ 100,000' is a MONEY
16:01:38.080 [main] DEBUG phoenix.ArticleNlpRunner - '40 %' is a PERCENT
16:01:38.080 [main] DEBUG phoenix.ArticleNlpRunner - '15th August' is a DATE
16:01:38.080 [main] DEBUG phoenix.ArticleNlpRunner - 'London' is a LOCATION
16:01:38.080 [main] DEBUG phoenix.ArticleNlpRunner - 'Make Believe Town' is a LOCATION
16:01:38.080 [main] DEBUG phoenix.ArticleNlpRunner - 'Sigourney' is a PERSON
16:01:38.081 [main] DEBUG phoenix.ArticleNlpRunner - '30 days' is a DURATION
16:01:38.081 [main] DEBUG phoenix.ArticleNlpRunner - We extracted 7 tokens of interest f
```

There are some caveats though – your dictionary needs to be carefully selected to not overwrite the better “natural” performance of Stanford NER using its' Conditional Random Field (CRF)-inspired logic augmented with Gibbs Sampling. For example, if you have a customer company called Make Believe Town Limited (unlikely, but not impossible), then Stanford NER will mis-classify Make Believe Town Limited to Make Believe Town. However, with careful dictionary population and a good understanding of the target raw text corpus, this is still a very fruitful approach.

From the author of