Statistical Learning and Text Classification with NLTK and scikit-learn

Olivier Grisel
http://twitter.com/ogrisel

PyCON FR – 2010

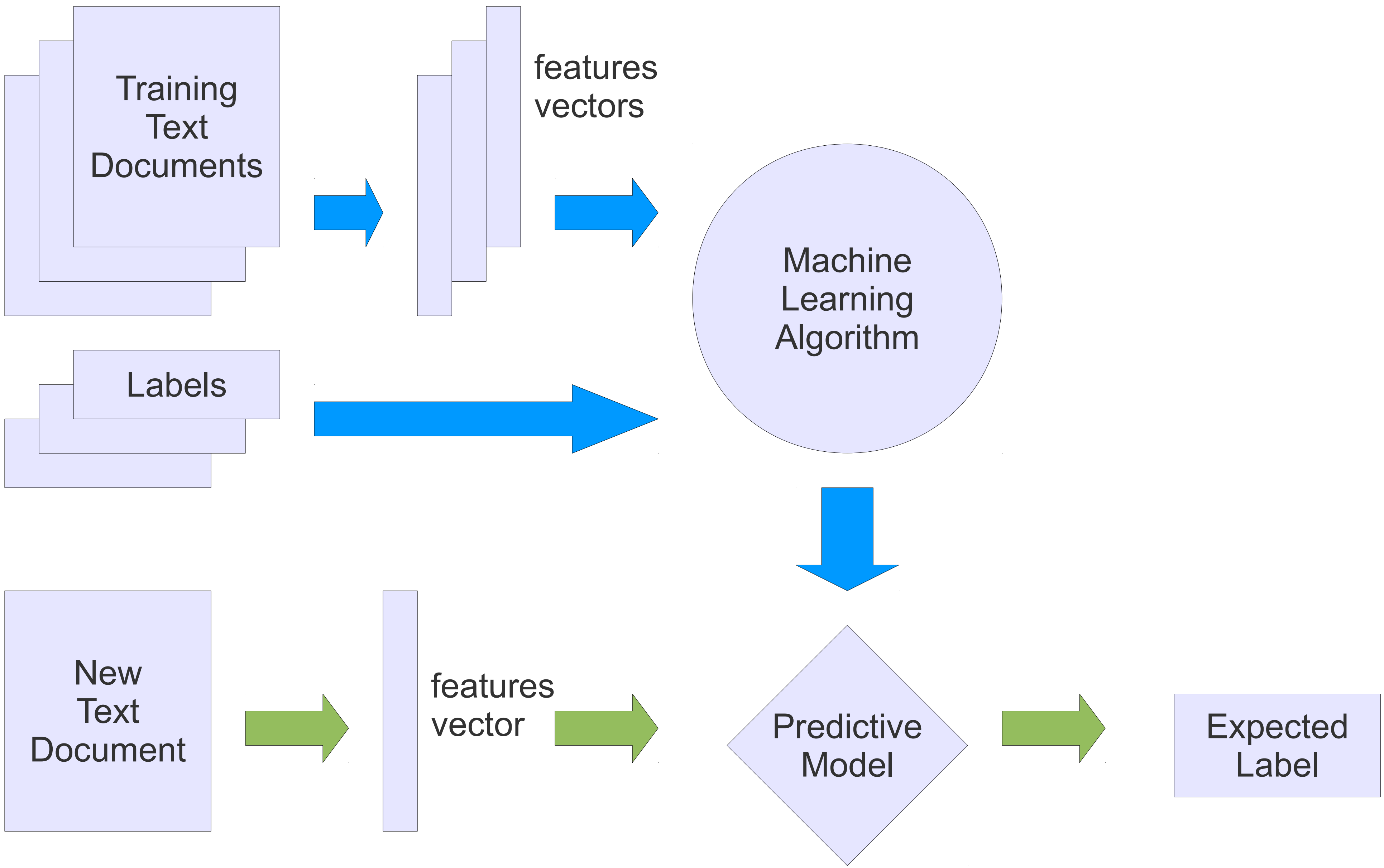
Applications of Text Classification

Task	Predicted outcome
Spam filtering	Spam, Ham
Language guessing	English, Spanish, French, ...
Sentiment Analysis for Product Reviews	Positive, Neutral, Negative
News Feed Topic Categorization	Politics, Business, Technology, Sports, ...
Pay-per-click optimal ads placement	Will yield money, Won't
Personal twitter filter	Will interest me, Won't
Malware detection in log files	Normal, Malware

Supervised Learning Overview

- Convert training data to a set of vectors of features (input) & label (output)
- Build a model based on the statistical properties of features in the training set, e.g.
  - Naïve Bayesian Classifier
  - Logistic Regression / Maxent Classifier
  - Support Vector Machines
- For each new text document to classify
  - Extract features
  - Asked model to predict the most likely outcome

Supervised Learning Summary



Typical features for text documents

- Tokenize document into list of words: **uni-grams**

```
['the', 'quick', 'brown', 'fox', 'jumps', 'over',
'the', 'lazy', 'dog']
```

- Then chose one of:
  - **Binary occurrences** of uni-grams:

    ```
    {'the': True, 'quick': True, ...}
    ```

  - **Frequencies** of uni-grams: nb times word_i / nb words in document:

    ```
    {'the': 0.22, 'quick': 0.11, ...}
    ```

  - **TF-IDF** of uni-grams (see next slides)

Better than freqs: TF-IDF

- Term Frequency

$$\mathrm{tf_{i,j}} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

- Inverse Document Frequency

$$\mathrm{idf_i} = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

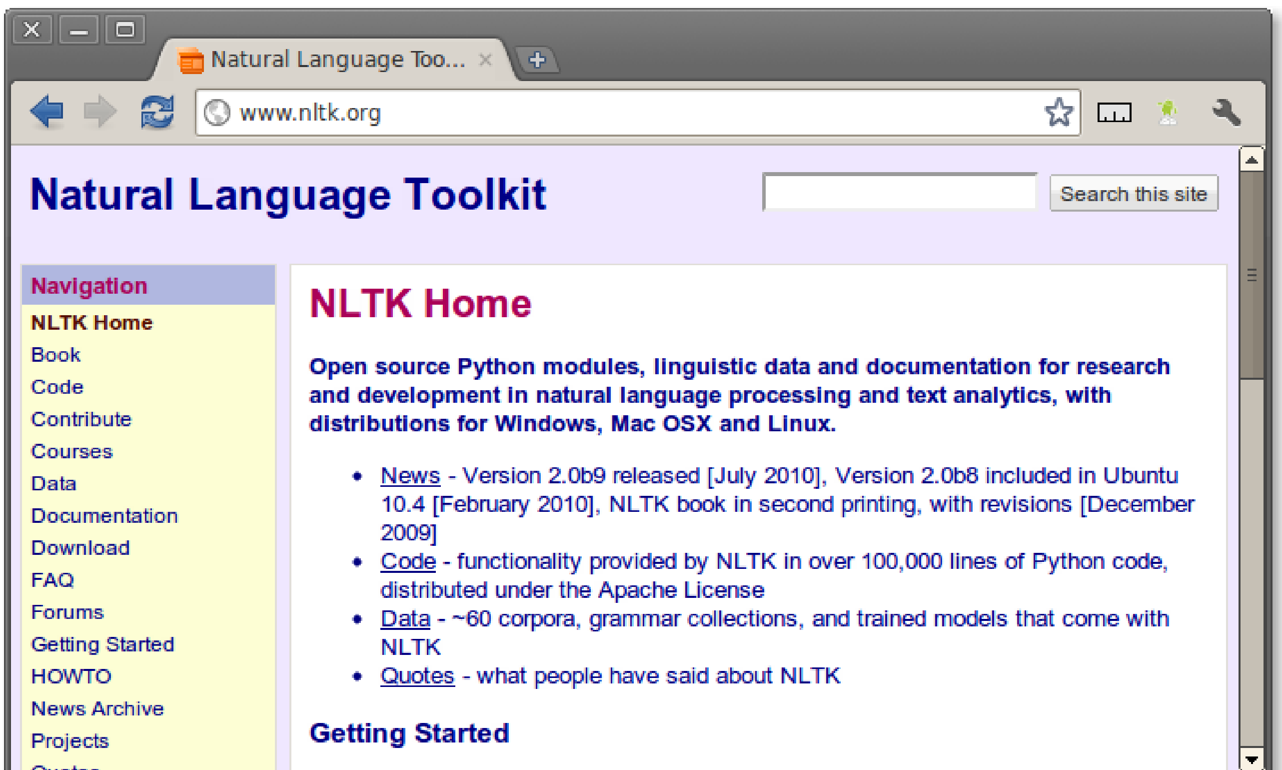
=> No real need for stop words any more, non informative words such as “the” are scaled done by IDF term

More advanced features

- Instead of uni-grams use
  - **bi-grams of words**: “New York”, “very bad”, “not good”
  - **n-grams of chars**: “the”, “ed ”, “ a ” (useful for language guessing)
- And the combine with:
  - **Binary occurrences**
  - **Frequencies**
  - **TF-IDF**

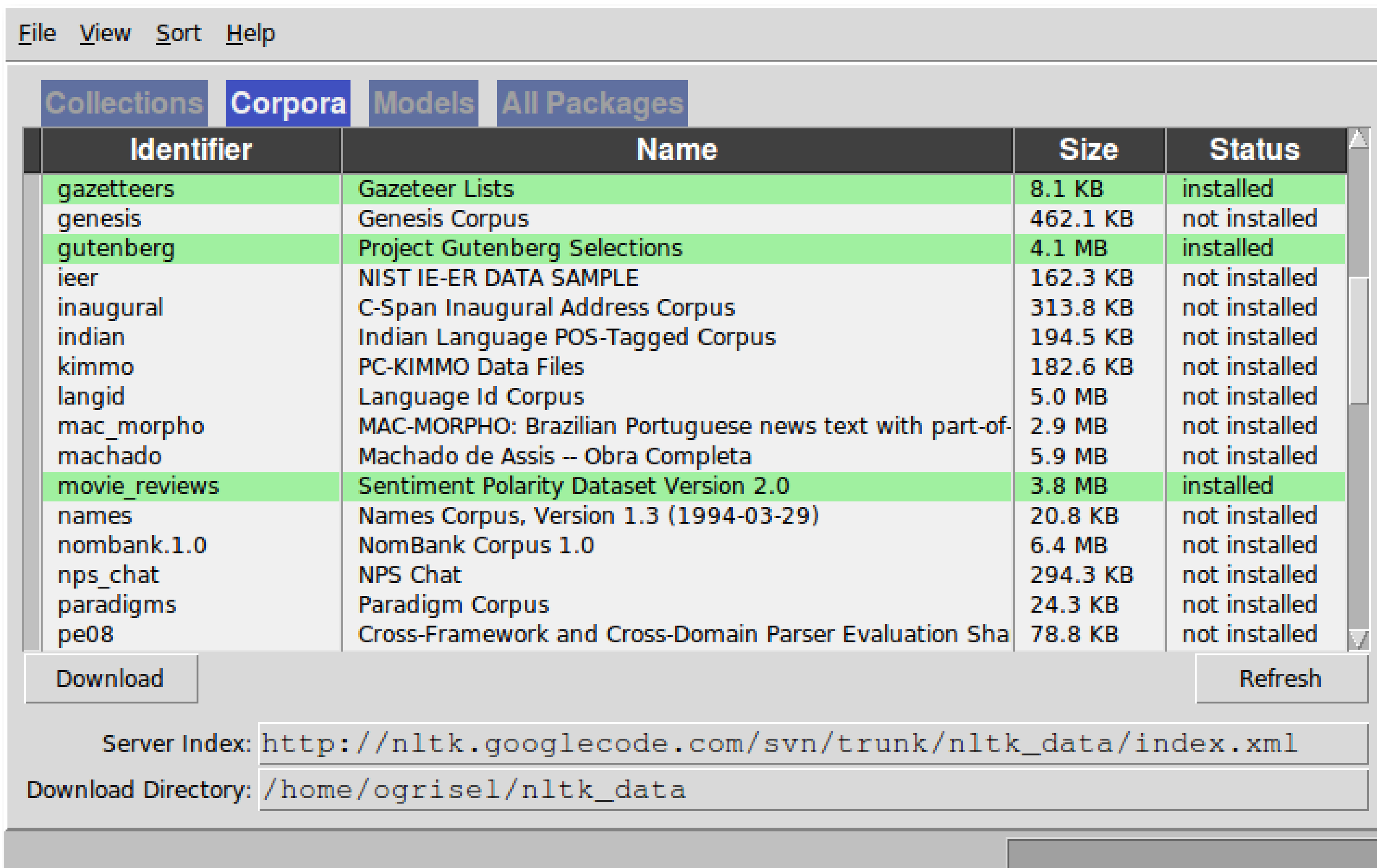
NLTK

- Code: ASL 2.0 & Book: CC-BY-NC-ND
- Tokenizers, Stemmers, Parsers, Classifiers, Clusterers, Corpus Readers



NLTK Corpus Downloader

```
>>> import nltk
>>> nltk.download()
```



Using a NLTK corpus

```
>>> from nltk.corpus import movie_reviews

>>> pos_ids = movie_reviews.fileids('pos')

>>> neg_ids = movie_reviews.fileids('neg')

>>> len(pos_ids), len(neg_ids)

1000, 1000

>>> print movie_reviews.raw(pos_ids[0])[:100]

films adapted from comic books have had plenty of success ,
whether they're about superheroes ( batm

>>> movie_reviews.words(pos_ids[0])

['films', 'adapted', 'from', 'comic', 'books', 'have', ...]
```

Common data cleanup operations

- Switch to lower case: `s.lower()`
- Remove accentuated chars:

```
import unicodedata
s = ''.join(c for c in unicodedata.normalize('NFD', s)
              if unicodedata.category(c) != 'Mn')
```

- Extract only word tokens of at least 2 chars
  - Using NLTK tokenizers & stemmers
  - Using a simple regexp:

```
re.compile(r"\b\w\w+\b", re.U).findall(s)
```

Feature Extraction with NLTK

- Simple word binary occurrence features:

```
def word_features(words):
    return dict((word, True) for word in words)
```

- Word Bigrams occurrence features:

```
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures as BAM
from itertools import chain


def bigram_word_features(words, score_fn=BAM.chi_sq, n=200):
    bigram_finder = BigramCollocationFinder.from_words(words)
    bigrams = bigram_finder.nbest(score_fn, n)
    return dict((bg, True) for bg in chain(words, bigrams))
```

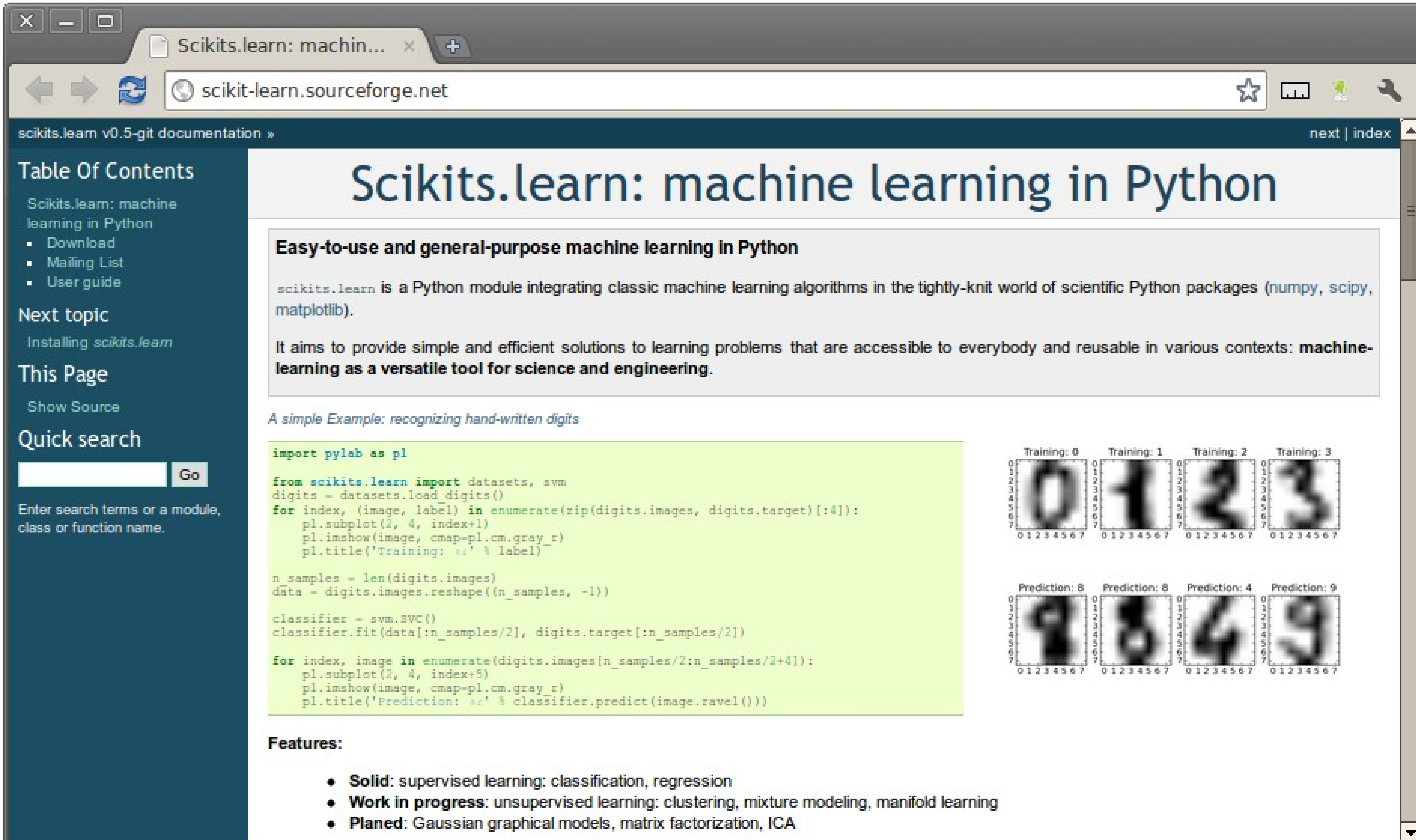
The NLTK - Naïve Bayes Classifier

```
from nltk.classify import NaiveBayesClassifier
mr = movie_reviews
neg_examples = [(features(mr.words(i)), 'neg')
                for i in neg_ids]
pos_examples = [(features(mr.words(i)), 'pos')
                for i in pos_ids]
train_set = pos_examples + neg_examples
classifier = NaiveBayesClassifier.train(train_set)

# later on a previously unseed document
predicted_label = classifier.classify(new_doc_features)
```

Most informative features

```
>>> classifier.show_most_informative_features()
        magnificent = True              pos : neg    =     15.0 : 1.0
        outstanding = True              pos : neg    =     13.6 : 1.0
          insulting = True              neg : pos    =     13.0 : 1.0
         vulnerable = True              pos : neg    =     12.3 : 1.0
          ludicrous = True              neg : pos    =     11.8 : 1.0
             avoids = True              pos : neg    =     11.7 : 1.0
        uninvolving = True              neg : pos    =     11.7 : 1.0
         astounding = True              pos : neg    =     10.3 : 1.0
        fascination = True              pos : neg    =     10.3 : 1.0
            idiotic = True              neg : pos    =      9.8 : 1.0
```

scikit-learn



Features Extraction in scikit-learn

```
from scikits.learn.features.text import *


text = u"J'ai mang\xe9 du kangourou  ce midi, c'\xe9tait pas
tr\xeas bon."

print WordNGramAnalyzer(min_n=1, max_n=2).analyze(text)

[u'ai', u'mange', u'du', u'kangourou', u'ce', u'midi',
u'etait', u'pas', u'tres', u'bon', u'ai mange', u'mange du',
u'du kangourou', u'kangourou ce', u'ce midi', u'midi etait',
u'etait pas', u'pas tres', u'tres bon']

char_ngrams = CharNGramAnalyzer(min_n=3, max_n=6)

print char_ngrams[:5] + char_ngrams[-5:]

[u"j'a", u"'ai", u'ai ', u'i m', u' ma', u's tres', u' tres
', u'tres b', u'res bo', u'es bon']
```

TF-IDF features & SVMs

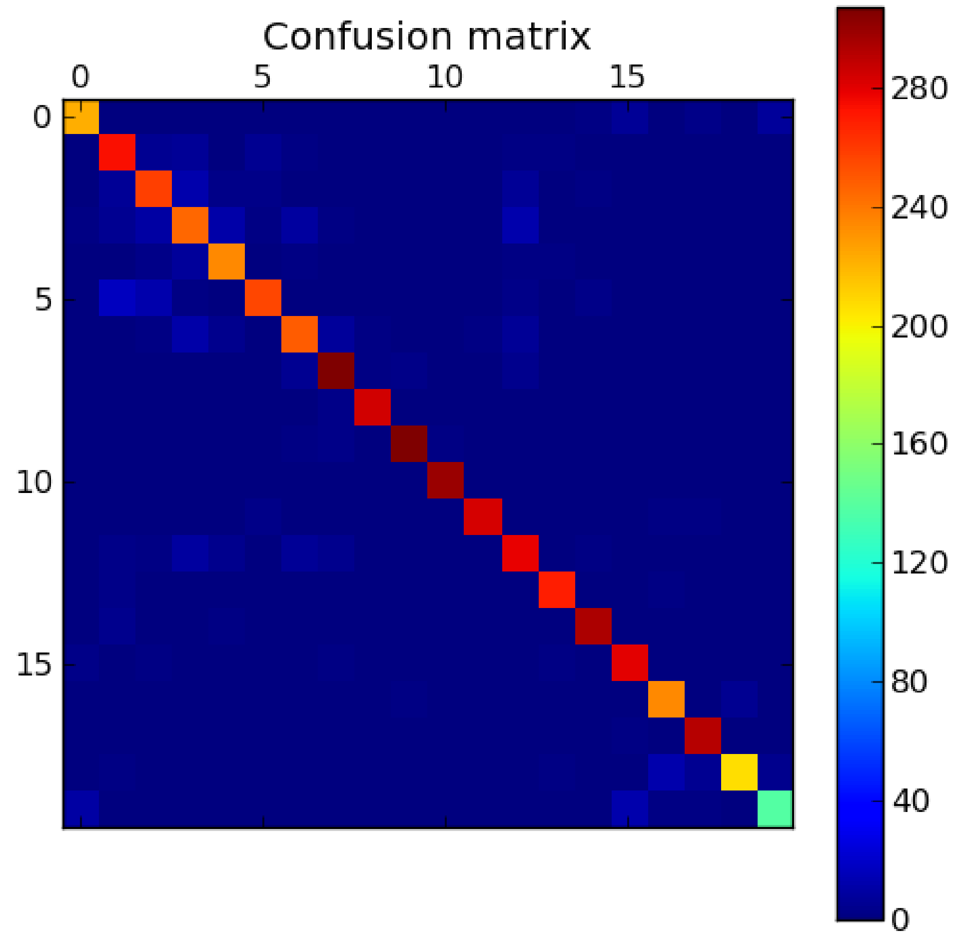
```
from scikits.learn.features.text import *
from scikits.learn.sparse.svm import LinearSVC

hv = SparseHashingVectorizer(dim=1000000, analyzer=)
hv.vectorize(list_of_documents)
features = hv.get_tfidf()
clf = SparseLinearSVC(C=10, dual=false)
clf.fit(features, labels)

# later with the same clf instance
predicted_labels = clf.predict(features_of_new_docs)
```

Typical performance results

- Naïve Bayesian Classifier with unigram occurrences on movie reviews: ~ 70%
- Same as above selecting the top 10000 most informative features only: ~ 93%
- TF-IDF unigram features + Linear SVC on 20 newsgroups ~93% (with 20 target categories)
- Language guessing with character ngram frequencies features + Linear SVC: almost perfect if document is long enough

Confusion Matrix (20 newsgroups)

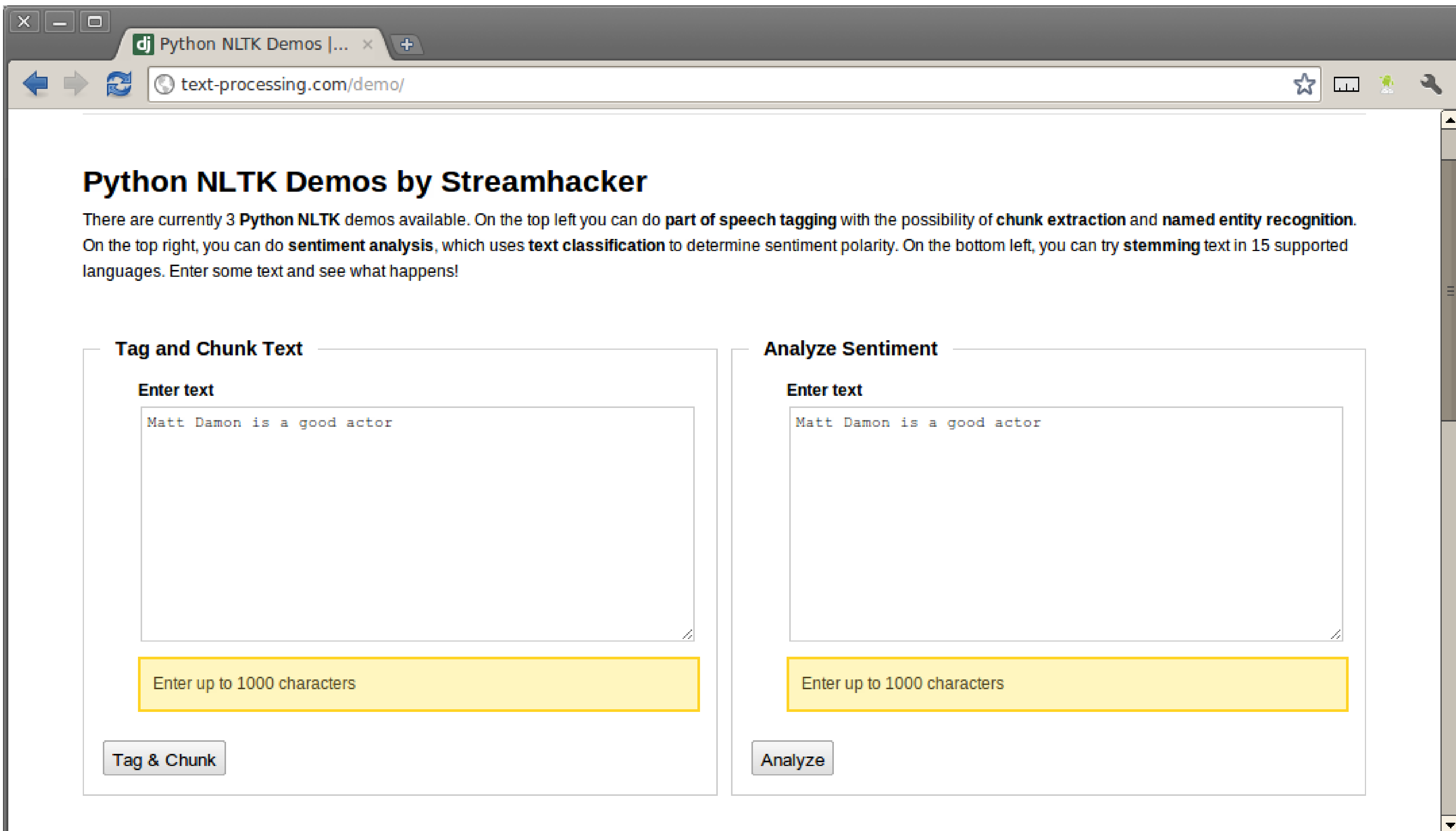
00 alt.atheism
01 comp.graphics
02 comp.os.ms-windows.misc
03 comp.sys.ibm.pc.hardware
04 comp.sys.mac.hardware
05 comp.windows.x
06 misc.forsale
07 rec.autos
08 rec.motorcycles
09 rec.sport.baseball
10 rec.sport.hockey
11 sci.crypt
12 sci.electronics
13 sci.med
14 sci.space
15 soc.religion.christian
16 talk.politics.guns
17 talk.politics.mideast
18 talk.politics.misc
19 talk.religion.misc



Handling many possible outcomes

- Example: possible outcomes are all the categories of Wikipedia (565,108)
- Document Categorization becomes Information Retrieval
- Instead of building one linear model for each outcome build a fulltext index and perform TF-IDF similarity queries
- Smart way to find the top 10 search keywords
- Use Apache Lucene / Solr MoreLikeThisQuery

NLTK – Online demos

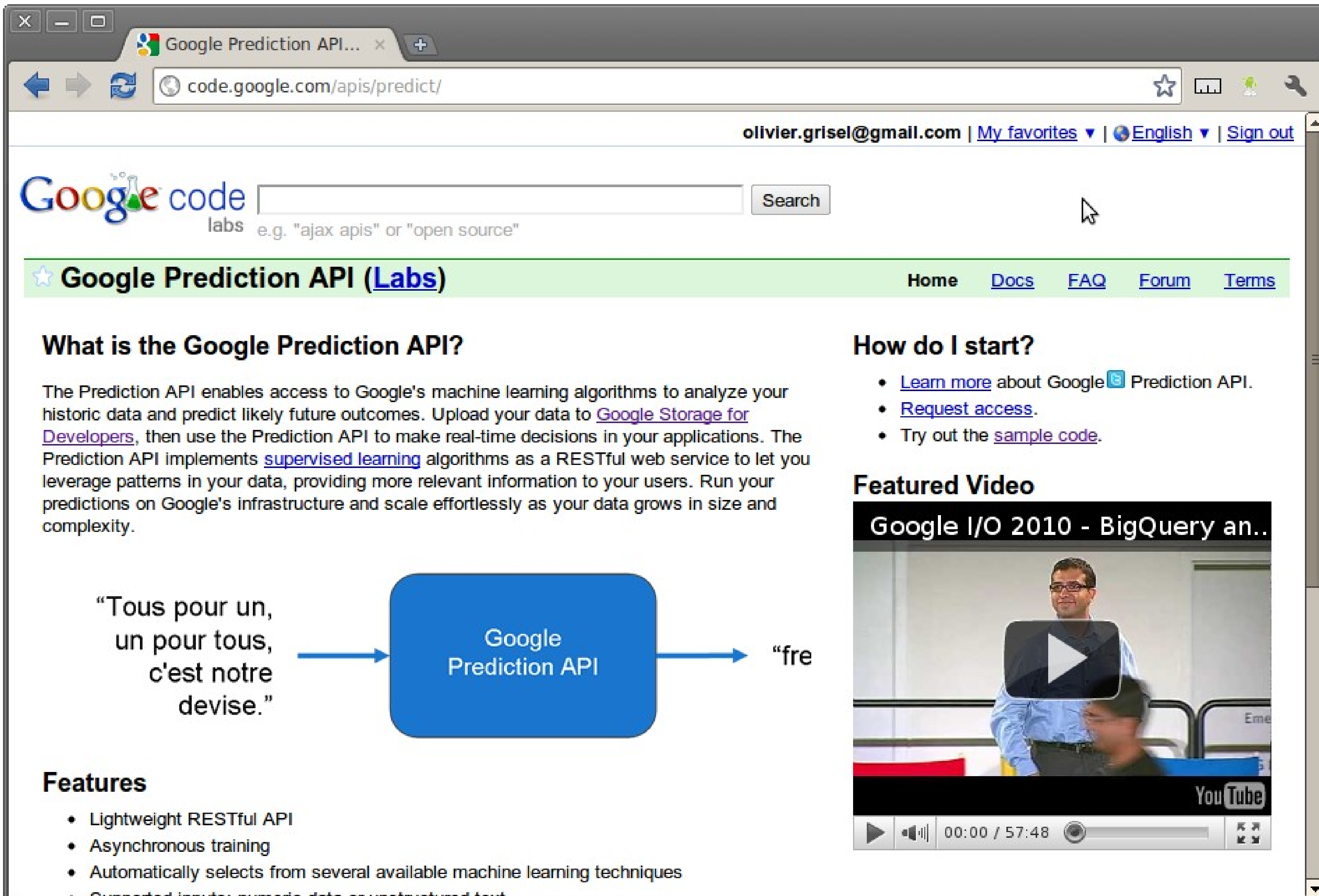


NLTK – REST APIs

```
% curl -d "text=Inception is the best movie ever" \
        http://text-processing.com/api/sentiment/
```


```
{
    "probability": {
        "neg": 0.36647424288117808,
        "pos": 0.63352575711882186
    },
    "label": "pos"
}
```

Google Prediction API



Some pointers

- http://www.nltk.org (Code & Doc & PDF Book)
- http://scikit-learn.sf.net (Doc & Examples)
  http://github.com/scikit-learn (Code)
- http://www.slideshare.net/ogrisel (These slides)

- http://streamhacker.com/(Blog on NLTK & APIs)
- http://github.com/hmason/tc (Twitter classifier – work in progress)