Trolling Detection with NLTK and Sklearn

https://github.com/rafaharo/trolling_detection

Rafael Haro | rharo [at] apache

Senior Software Engineer @ Athento

PyConES 2015

@rafa_haro

About Me

apache stanbol™

athento
understanding documents

Apache ManifoldCF™

What are we going to see today?

- Why NLP is so important?
- An Example Problem: Trolling Detection
- Python and NLP
- Some Theory: Document Representation Models
- Let's catch those Trolls!

Why Is NLP So Important?

http://www.wired.com/insights/2014/02/growing-importance-natural-language-processing/

DiGiorno Pizza
@DiGiornoPizza

Follow

A million apologies. Did not read what the hashtag was about before posting.

Reply Retweet Favorite More

RETWEETS 132 FAVORITES 143

11:15 PM - 8 Sep 2014

Public Comments · Moderator View Settings

ProPhoto

Add a comment...

Post to Facebook Posting as ProPhoto Blogs (Change) Comment

Arnaud Douglas Ardans II · Subscribe · Gresham, Oregon
I wish there was a PPB Community site. Share stories, customization help, etc. :-) Would love to see that.
2 · Like · Reply · Moderate · Follow Post · Saturday at 2:49am

Janet Wolbarst · Fine Art & Portrait Photographer at Janet Wolbarst Photography
do we need to do anything if we are using ProPhoto4?
Like · Reply · Moderate · Follow Post · March 8 at 12:10pm

Jared Henderson · Owner/Lead Developer at ProPhoto Blogs
Nope, not unless you're using the Tweaks plugin
Like · Reply · March 8 at 12:17pm

Facebook social plugin

Related Topics: Scam - Identity Theft Advice - Phishing - Report Internet Scam

Notice Links

« Back to Inbox | Archive | Report Spam | Delete | More actions...

Billing Update Inbox

staff334@ebay.com to aroyalflush, www.simvia show details 3:49 pm (6 minutes ago) Reply

Below is the result of your feedback form. It was submitted by
staff334@ebay.com (staff334@ebay.com) on Friday, November 10, 2006 at 18:49:51
---

h6:

Dear eBay customer,
we are sorry to inform you that we are having
problem's with the billing information on your
account. We would appreciate it if you would go to
our website and fill out the proper information that
we require to keep your account active,Please Update
your account information by visiting our updates web
site

http://get-me.to/ebaybillingupdate

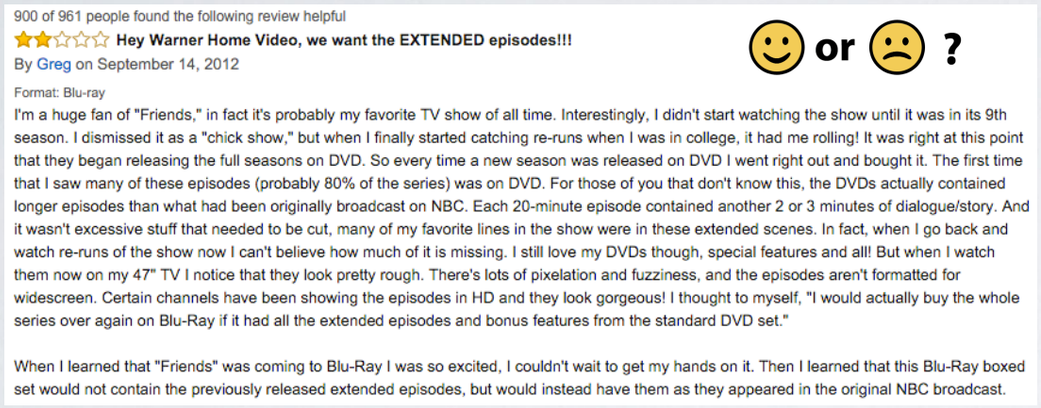
eBay Billing Center
Rep ID. 32A
Thank you for your business.
The eBay Staff.

User Reviews

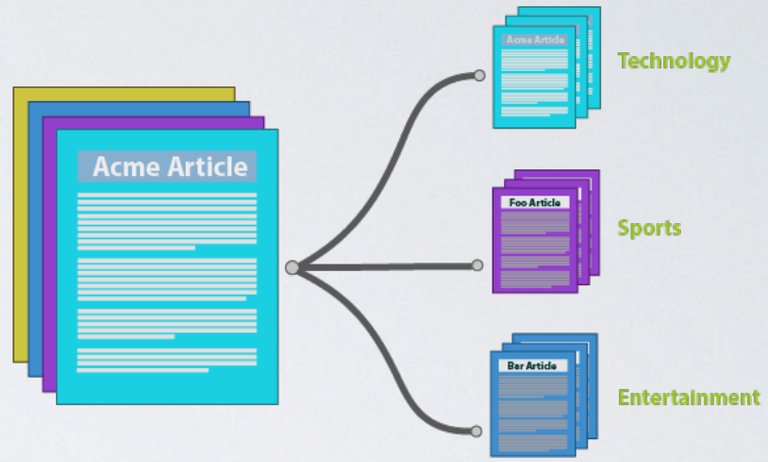
10 November 2003 | by nowhozdisiz (new york) – See all my reviews

I thought this movie was cute when I was about 10, but now that I look at it i realize how stupid the whole thing is. there is not an educated person alive that can honestly say that they could walk back into the middle ages and save an entire kingdom all by themselves, theyd get there ass kicked. First of all, if this kid managed to survive the first 5 minutes without getting an arrow put through him or beheaded by a broadsword, and he managed to avoid being taken prisoner and made a gay soliders plaything, he would most likely die of many diseases that his body wasn't accustomed to. Also, even you do have a magical place to plug in a stereo you play rock and roll for guards at a middle age castle, youll be labled as a heritic and executed. but dying aside, the middle ages just werent that cool, disease and warfare were everywhere, people lived in disgusting conditions that even mother theresa would run away from, and there was no place for the weak, you either fought or you died and if you werent tough, you were probably killed. But enough about that, what i have to say is for young kids that enjoy the "kids outsmarting the stupid grownups" genre, like home alone (that ones off the hook though, still entertains me even at 19 and will continue to) go see it, but for older people that know what the middle ages were really like, dont bother.

NLP Applications



Opinion Mining



Text Categorization

Information Extraction

Virtual Assistants

An Example: Trolling Detection

- kaggle's 2012 competition (https://www.kaggle.com/c/detecting-insults-in-social-commentary)
- Predict whether a comment posted during a public discussion is considered insulting to one of the participants.
- Some participants' solutions:
  - http://blog.kaggle.com/2012/09/26/impermium-andreas-blog/
  - https://github.com/andreiolariu/kaggle-insults

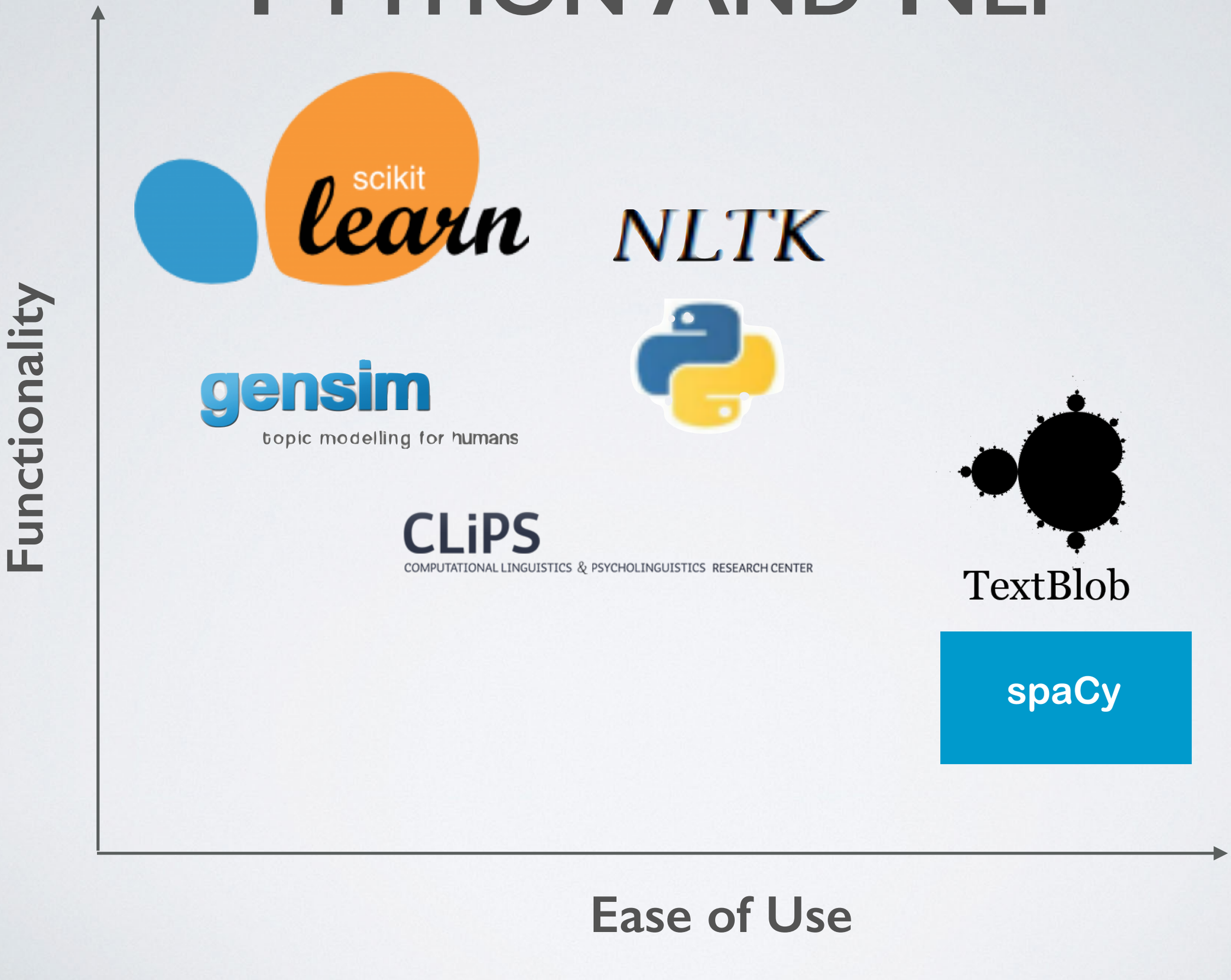
Dataset Format

Insult,Date,Comment
1,20120618192155Z,""You fuck your dad.""
0,20120528192215Z,""i really don't understand your point.\xa0 It seems that you are mixing apples and oranges.""
0,20120503012628Z,""Yeah and where are you now?""
1,,""shut the fuck up. you and the rest of your faggot friends should be burned at the stake""
1,20120502173553Z,""Either you are fake or extremely stupid...maybe both...""
1,20120620160512Z,""That you are an idiot who understands neither taxation nor women's health.""
0,20120620015140Z,""@jdstorm dont wish him injury but it happened on its OWN and i DOUBT he's injured, he looked embarrassed to me""
0,20120530044519Z,""Be careful,Jimbo.OG has a fork with your name on it.""
0,,""@tonnyb Or they just don't pay attention ""
0,,""@barrettmarson Huh? Her income was $21,912. Her spending was $5,470. My math puts that at 24.9%.""
1,20120611090207Z,""FOR SOME REASON U SOUND RETARDED. LOL. DAMN. WHERE U BEEN NEGRO""
1,20120320162532Z,""You with the 'racist' screen name\n\nYou are a PieceOfShit.....""
0,,""Oh! & cheating as well...1966 cup was a cheat fest, & you still dare to post that comment..shame on you lol""
1,20120320075347Z,""your such a dickhead...""
1,20120320203947Z,""Your a retard go post your head up your #%&*""
0,20120612051612Z,""@EephusBlue\xa0Makes you want to say "Mike MacWHOgal?"""
0,20120611234235Z,""http://www.youtube.com/watch?v=tLYLLPHKRU4""
0,20120502215304Z,""And you know they've burned holes in all the carpeting.""
0,20120503031721Z,""you are a land creature. You would drown....""

Competition Guidelines

- We are looking for comments that are intended to be insulting to a person who is a part of the larger blog/forum conversation.
- We are NOT looking for insults directed to non-participants (such as celebrities, public figures etc.).
- Insults could contain profanity, racial slurs, or other offensive language. But often times, they do not.
- Comments which contain profanity or racial slurs, but are not necessarily insulting to another person are considered not insulting.
- The insulting nature of the comment should be obvious, and not subtle.

Python And Nlp



Document Representation Models

- Bag of Words: Each Document is Represented as a list of numbers counting the number of times each word of the corpus appears in the document

```
(1) John likes to watch movies. Mary likes movies too.
```

```
(2) John also likes to watch football games.
```

```
[
    "John",
    "likes",
    "to",
    "watch",
    "movies",
    "also",
    "football",
    "games",
    "Mary",
    "too"
]
```

Vocabulary

```
(1) [1, 2, 1, 1, 2, 0, 0, 0, 1, 1]
(2) [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]
```

Documents' Vectors

Source: **Wikipedia**

Our Corpus Representation

```
def corpus_stats(collection):
    import nltk
    import pprint
    words = tokenize_collection(collection, lowercase=True, stopwords='english', min_length=3)
    text = nltk.Text(word.lower() for word in words)
    print "Number of Words: " + str(len(text))
    print "Number of unique words: " + str(len(set(text)))
    dist = nltk.FreqDist(text)
    pp = pprint.PrettyPrinter(indent=4)
    pp.pprint(dist.most_common(20))
    print dist['stupid']
```

```
Number of Words: 69134
Number of unique words: 15080
[   (u'like', 725),
    (u'...', 551),
    (u'people', 426),
    (u'get', 410),
    (u'would', 375),
    (u'one', 371),
    (u'know', 328),
    (u'think', 305),
    (u'fuck', 252),
    (u'://', 245),
    (u'http', 239),
    (u'right', 225),
    (u'time', 222),
    (u'make', 212),
    (u'good', 210),
    (u'see', 209),
    (u'....', 208),
    (u'really', 195),
    (u'back', 191),
    (u'even', 189)]
155
```

NLTK

Vector Space Model: TF-IDF

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

Vector Space Model: TF-IDF

```
def tf_idf_stats(collection, num=20):
    from sklearn.feature_extraction.text import TfidfVectorizer
    tfidf = TfidfVectorizer(analyzer='word', min_df=4, stop_words = 'english')
    matrix = tfidf.fit_transform(collection)
    dense = matrix.todense()
    features_names = tfidf.get_feature_names()
    print "\nNumber of Features: " + str(len(features_names))
    for index, row in enumerate(dense[0:num]):
        print "\nScores for comment: " + str(index)
        comment = row.tolist()[0]
        scores = [pair for pair in zip(range(0, len(comment)), comment) if pair[1] > 0]
        sorted_scores = sorted(scores, key=lambda t: t[1] * -1)
        for phrase, score in [(features_names[word_id], score) for (word_id, score) in sorted_scores][:num]:
            print('{0: <20} {1}'.format(phrase, score))
        print "\n"
```

```
Number of Features: 2746

Scores for comment: 0
dad                  0.855948362082
fuck                 0.517061313047



Scores for comment: 1
oranges              0.544506685284
apples               0.533299076937
understand           0.373548678543
point                0.358020658348
really               0.300104678424
don                  0.247637515659
```

Using Tf-IDF: Document Similarity

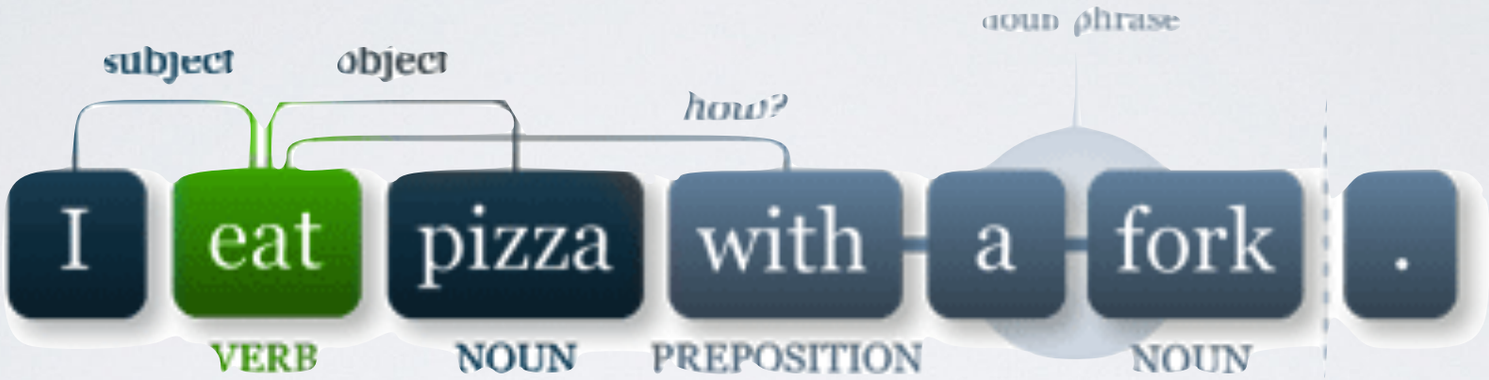
```python
def collection_text_similarity(collection, tokenizer=None, ngram_range=(1, 1)):
    """
    This function calculate the cosine based similarity between each pair of documents in the collection
    First, the TF-IDF transformation of the documents is performed to later calculate the similarity
    matrix by calculating the cosine between the vectors of tokens

    :param collection: Collection of documents (list of strings)
    :type collection: list
    :param tokenizer: [Optional] Tokenizer to be applied to the texts
    :type tokenizer: sandbox.document_processing.categorizer.tokenizer.Tokenizer
    :param ngram_range: [Optional] Ngrams Range
    :type ngram_range: list
    :return: Collection Similarity Matrix
    :rtype:
    """
    from sklearn.feature_extraction.text import TfidfVectorizer
    vectorizer = TfidfVectorizer(decode_error='replace',
                                 strip_accents='unicode',
                                 lowercase=True,
                                 tokenizer=tokenizer, ngram_range=ngram_range)
    tfidf = vectorizer.fit_transform(collection)
    return (tfidf * tfidf.T).A


def text_similarity(textA, textB, tokenizer=None, ngram_range=(1, 1)):
    narray = collection_text_similarity([textA, textB], tokenizer, ngram_range)
    return narray[0][1]
```

```python
similarity = collection_text_similarity(['You sound stupid LOL',
                                         'You sound retarded LOL',
                                         'You are very nice'])
print similarity
```

```
[[ 1.          0.60089819  0.12042206]
 [ 0.60089819  1.          0.12042206]
 [ 0.12042206  0.12042206  1.        ]]
```

scikit learn

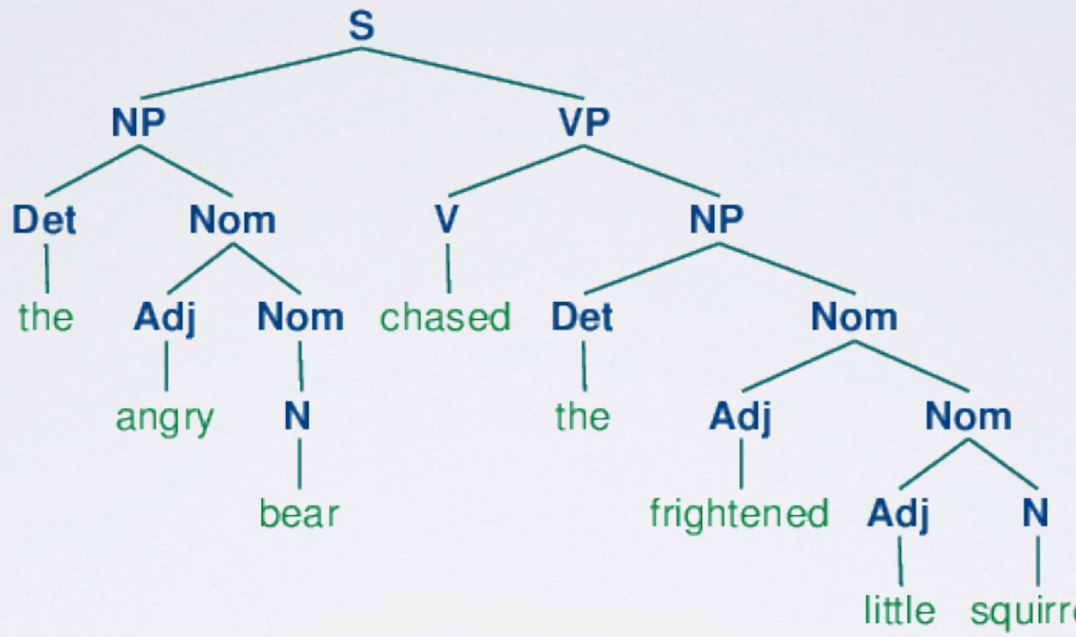
POS-Tagging




```
>>> text = word_tokenize("They refuse to permit us to obtain the refuse permit")
>>> nltk.pos_tag(text)
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit', 'VB'), ('us', 'PRP'),
('to', 'TO'), ('obtain', 'VB'), ('the', 'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```

NLTK

Sentence Parsing

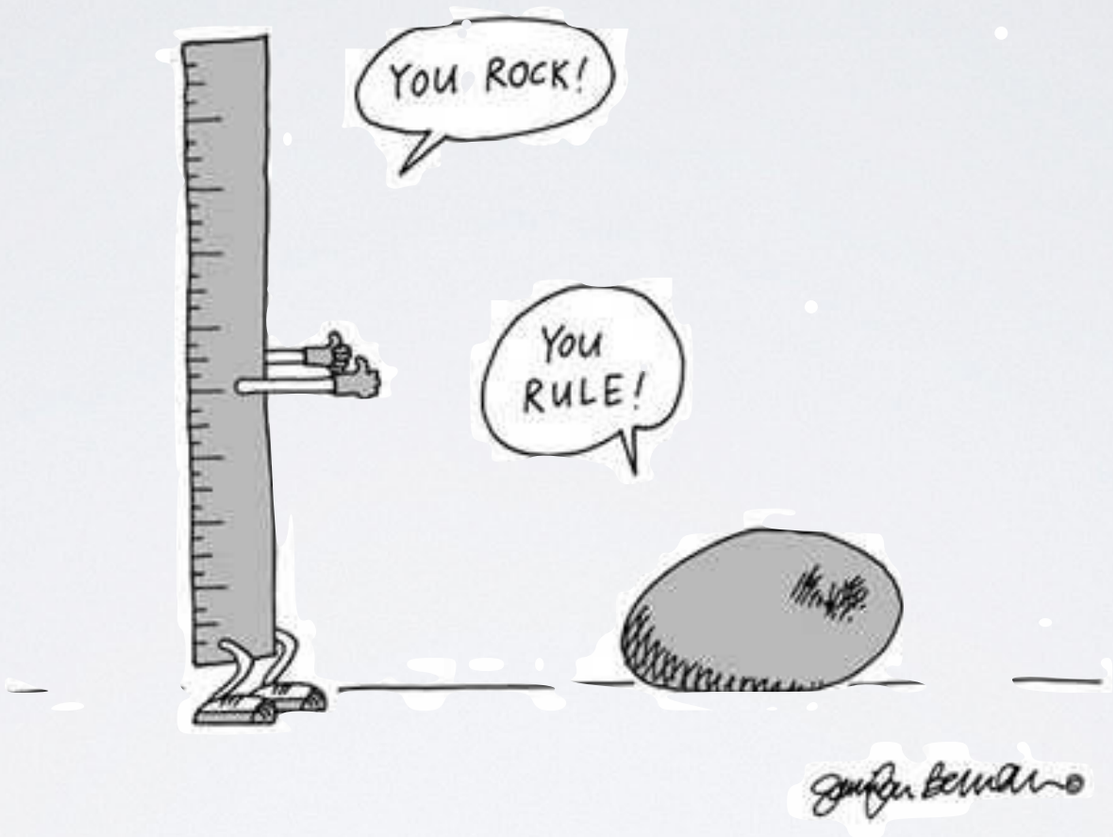



```
>>> from pattern.en import parse
>>> from pattern.en import pprint
>>>
>>> pprint(parse('I ate pizza.', relations=True, lemmata=True))

    WORD   TAG    CHUNK   ROLE   ID     PNP    LEMMA
       I   PRP    NP      SBJ    1      -      i
     ate   VBP    VP      -      1      -      eat
   pizza   NN     NP      OBJ    1      -      pizza
       .   .      -       -      -      -      .
```

gensim

topic modelling for humans

Semantics: Collocations



The **context** is essential, because the context is key for **disambiguation**

SEMANTICS: COLLOCATIONS (II)

```python
def extract_top_bigrams_collocations(collection, num=10, frequencyThreshold=3, windows_size=2, filter_word=None):
    """
    This methods extracts, for each document collection, the top N bigram collocations. Bigram collocations
    are pairs of words which commonly co-occur. With a windows_size < 2, only bigrams formed by consecutive words
    will be taken into account. In that case, the result is consistent with a list of 2-words expressions that
    frequently appear in the collection. For windows_size > 2, all pairs of words within a windows of windows_size
    words will be considered. In that case, the result is consistent with a list of 2 related words that frequently
    co-occur together and therefore commonly have a semantic relationship between them
    """
    from nltk import collocations
    words = tokenize_collection(collection)
    bigram_measures = collocations.BigramAssocMeasures()
    if windows_size > 2:
        finder = collocations.BigramCollocationFinder.from_words(words,windows_size)
    else:
        finder = collocations.BigramCollocationFinder.from_words(words)
    finder.apply_freq_filter(frequencyThreshold)
    if filter_word:
        finder.apply_ngram_filter(lambda *w: filter_word not in w)
    return finder.nbest(bigram_measures.chi_sq, num)
```

```python
collocations = extract_top_bigrams_collocations(comments, 10, windows_size=2, filter_word='sound')
print collocations
```

```
[(u'sound', u'like'), (u'you', u'sound'), (u'them', u'sound'), (u'sound', u'fakeinsult')]
```

NLTK

SEMANTICS: WORD2VEC

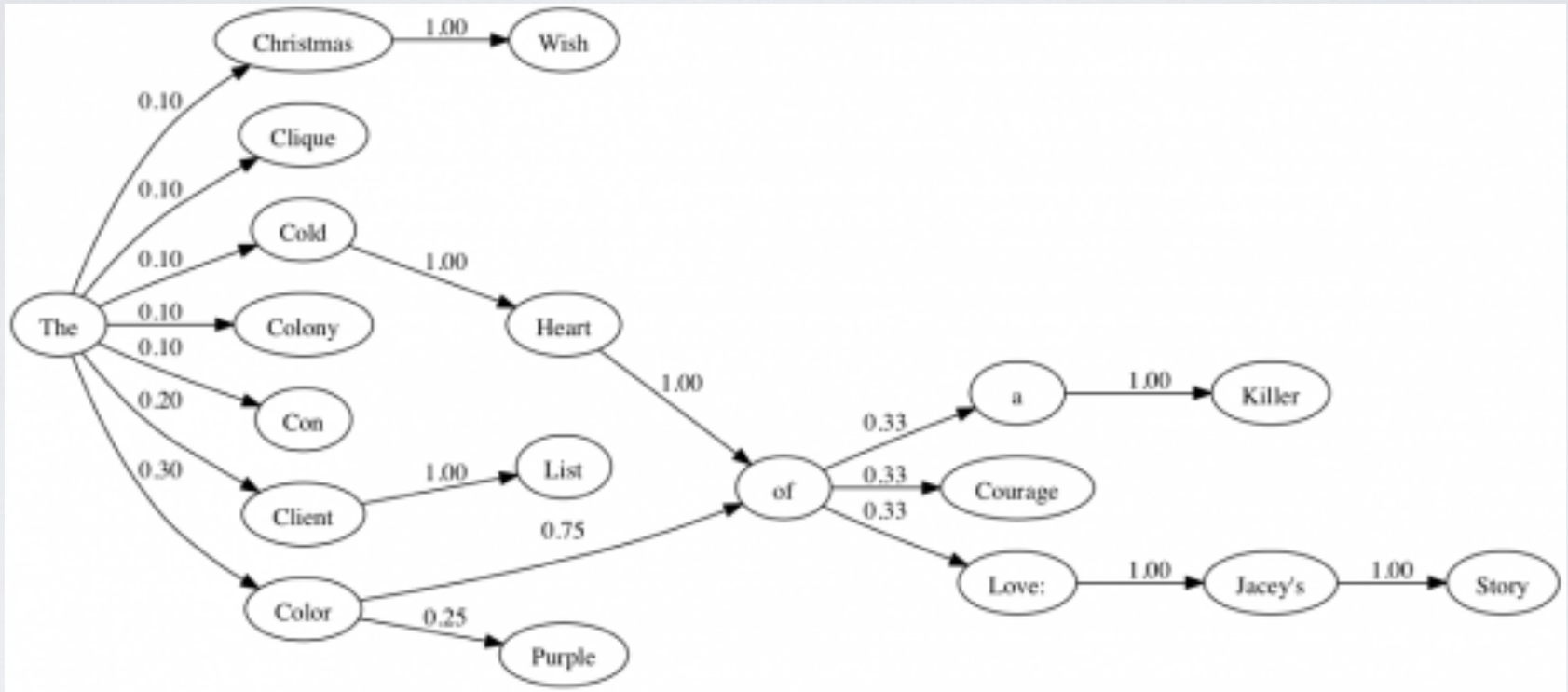
- **Word2Vec**: document representation using Deep Learning
- Each word is represented by a vector of distances with other words (different distance functions can be used)
- Each document is represented then by a vector of vectors

```
>>> model = Word2Vec(sentences, size=100, window=5, min_count=5, workers=4)
```

```
>>> model.most_similar(positive=['woman', 'king'], negative=['man'])
[('queen', 0.50882536), ...]

>>> model.doesnt_match("breakfast cereal dinner lunch".split())
'cereal'

>>> model.similarity('woman', 'man')
0.73723527

>>> model['computer']  # raw numpy vector of a word
array([-0.00449447, -0.00310097,  0.02421786, ...], dtype=float32)
```

gensim
topic modelling for humans

Language Models




```
def language_model(collection):
    from nltk import ConditionalProbDist
    from nltk import ConditionalFreqDist
    from nltk import bigrams
    from nltk import MLEProbDist
    words = tokenize_collection(collection)
    freq_model = ConditionalFreqDist(bigrams(words))
    prob_model = ConditionalProbDist(freq_model, MLEProbDist)
    return prob_model
```

NLTK

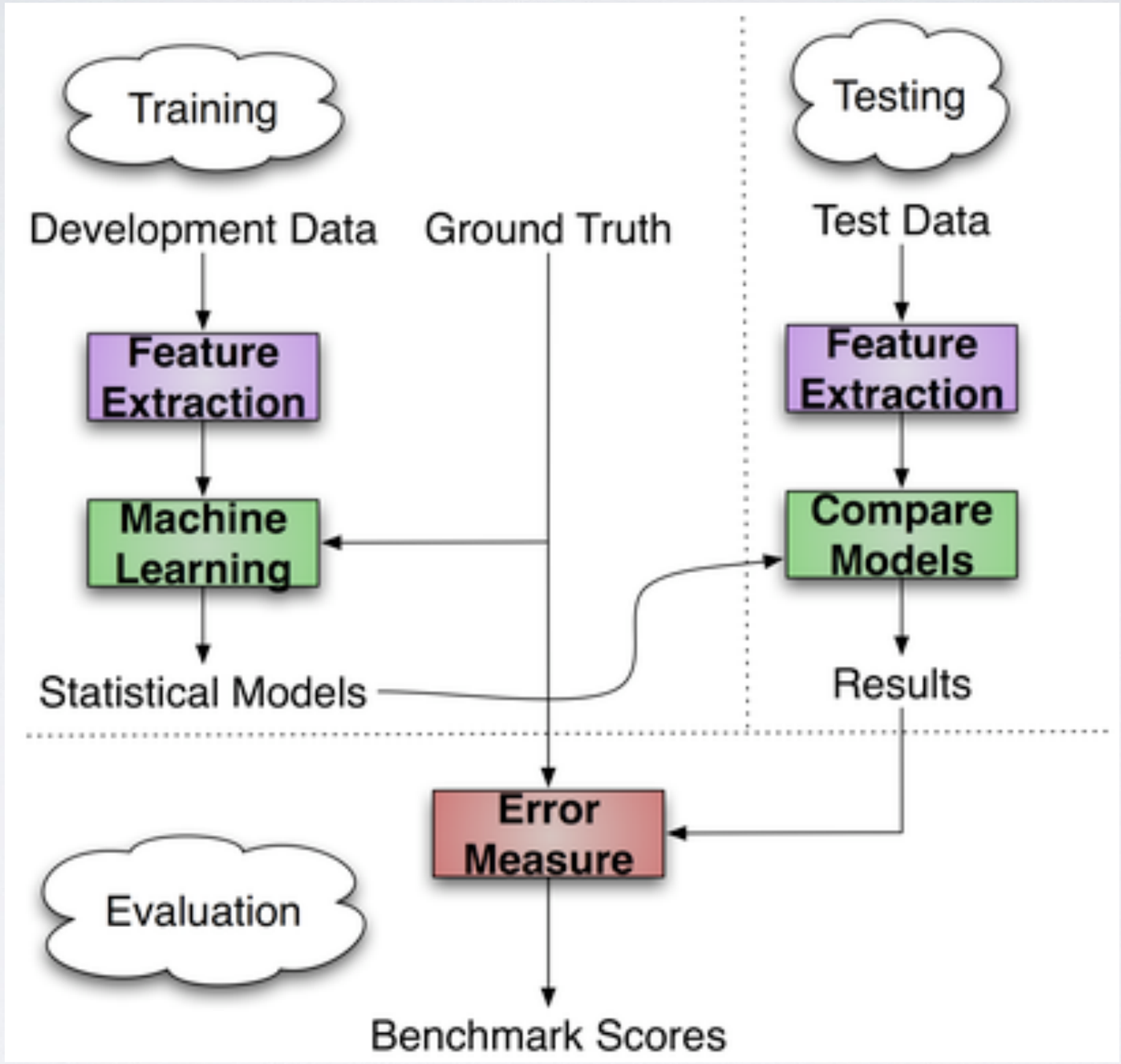
Language Models (II)

```
model = language_model(comments)

print "\nSamples: "
print model["sound"].samples()
print "\n"
```

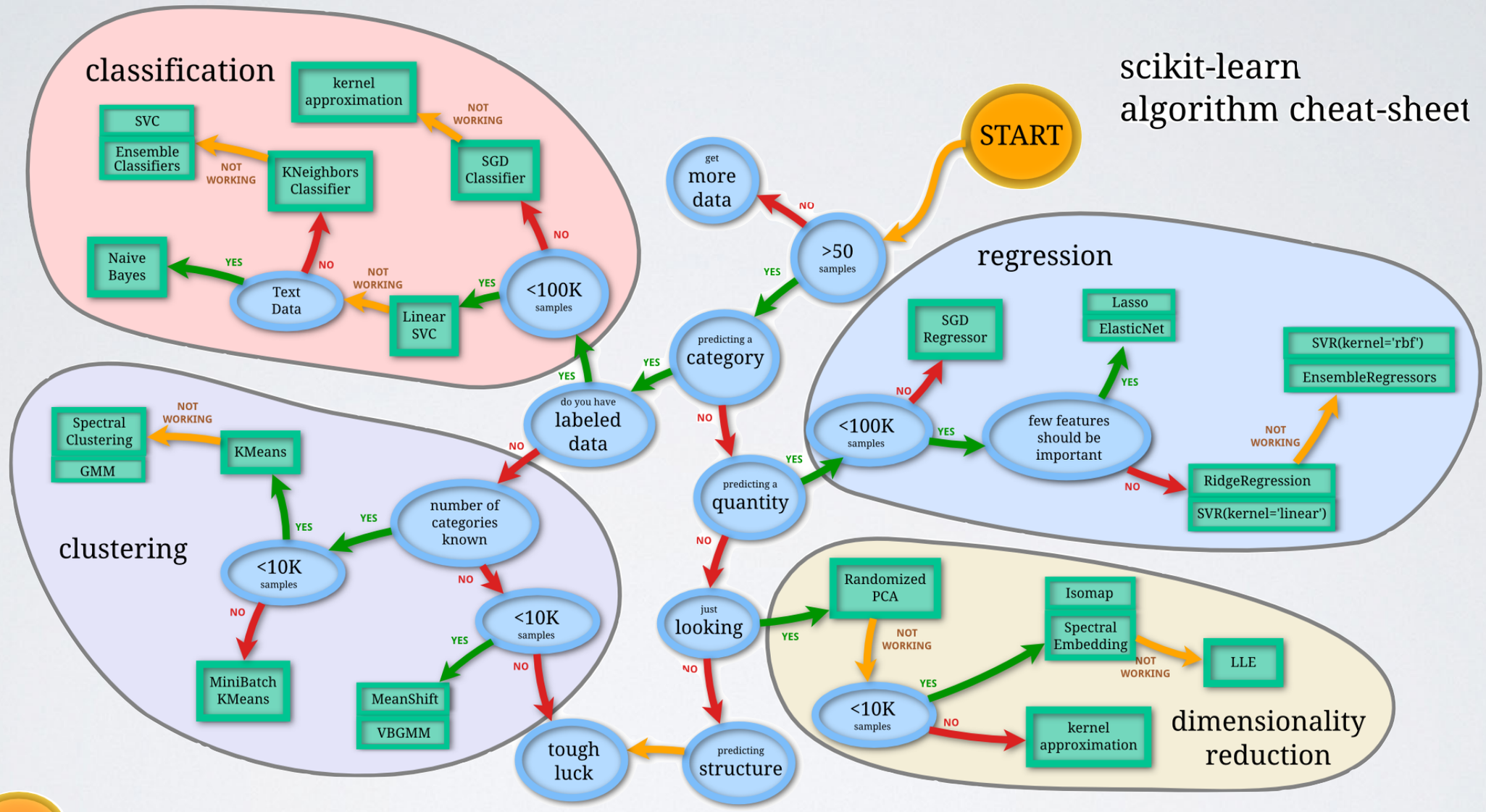
```
Samples:
[   u'all',
    u'good',
    u'like',
    u'makes',
    u'make',
    u'fakeinsult',
    u'little',
    u'judgement',
    u'brain',
    u'ignorant',
    u'are',
    u'female',
    u'familia',
    u'intelligent']
```

NLTK

Let's Catch Those Trolls!!



How To Select Estimators



RESOURCES

- **Lexicons**: BadWords list and Sentiment Analysis Lexicon

- **Libraries**: NLTK, gensim, enchant, re, Scikit-Learn

Naive Classifier

```python
def naive_classifier(comment, badwords, fake=False):
    if not fake:
        if any(badword in comment.lower() for badword in badwords):
            return 1
        else:
            return 0
    else:
        if "fakeinsult" in comment:
            return 1
        else:
            return 0
```

Naive Model Result

Accuracy: 0.67170381564

	precision	recall	f1-score	support
NoInsult	0.88	0.64	0.74	1954
Insult	0.43	0.76	0.55	693
avg / total	0.76	0.67	0.69	2647

TF-IDF Based Classifier

```
def train_basic(categories, comments):
    from sklearn.pipeline import Pipeline
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.linear_model import SGDClassifier

    text_clf = Pipeline([('vect', TfidfVectorizer(lowercase=True, ngram_range=(1, 3), analyzer="word", min_df=3)),
                         ('clf', SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, n_iter=5, random_state=42))])
    text_clf = text_clf.fit(comments, categories)
    return text_clf
```

```
TF-IDF Model Result

Accuracy: 0.817907064601

             precision    recall  f1-score   support

   NoInsult       0.81      0.99      0.89      1954
     Insult       0.90      0.34      0.50       693

avg / total       0.83      0.82      0.79      2647
```

TF-IDF Based Classifier (II)

```
def replace_badwords(comment, badwords):
    comment = comment.lower()
    for badword in badwords:
        if badword is not 'fakeinsult':
            comment = comment.replace(badword, " fakeinsult ")
    return comment
```

```
TF-IDF Model Result

Accuracy: 0.836796373253

             precision    recall  f1-score   support

   NoInsult       0.83      0.98      0.90      1954
     Insult       0.89      0.43      0.58       693

avg / total       0.85      0.84      0.81      2647
```

Word2Vec Classifier

```python
def word2vec_model(documents, n_dim=1000):
    from gensim.models import Word2Vec
    model = Word2Vec(documents, size=n_dim, window=5, min_count=3, workers=4)
    return model
```

```python
def buildWordVector(w2vmodel, text, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0.
    for word in text:
        try:
            sorted_vec = np.sort(w2vmodel[word])
            vec += sorted_vec.reshape((1, size))
            count += 1.
        except KeyError:
            continue
    if count != 0:
        vec /= count
    return vec

def w2vectorize(collection, model, n_dim):
    from sklearn.preprocessing import scale
    vecs = np.concatenate([buildWordVector(model, z, n_dim) for z in collection])
    vecs = scale(vecs)
    return vecs

def train_word2vec(categories, comments, n_dim):
    from feature_extraction import tokenize_document
    from feature_extraction import word2vec_model
    from sklearn.linear_model import SGDClassifier
    documents = [tokenize_document(document) for document in comments]
    model = word2vec_model(documents, n_dim)
    train_vecs = w2vectorize(documents, model, n_dim)
    classifier = SGDClassifier(loss='log', penalty='l1')
    classifier.fit(train_vecs, categories)

    return model, classifier
```

Word2Vec Model Result

Accuracy: 0.760105780128

	precision	recall	f1-score	support
NoInsult	0.83	0.85	0.84	1954
Insult	0.54	0.51	0.53	693
avg / total	0.76	0.76	0.76	2647

Custom Classifier: Features By Comment

- **n_words**: Number of words
- **n_chars**: Number of characters
- **n_dwords**: Number of words recognized by an english dictionary
- **sent_pos**: Average word's Positive score
- **n_you_are**: number of 'you are…insult' regex matchs
- **n_you**: number of 'you are|sound' regex matchs
- **n_bad**: number of bad words
- **exclamation**: number of exclamations
- **addressing**: number of user's references using '@'
- **allcaps_ratio**: Ratio of Uppercase words
- **bad_ratio**: Ratio of Bad Words
- **dic_ratio**: Ratio of dictionary recognized words

Custom Classifier (II)

```
def transform(self, documents):
    import enchant
    import re
    d = enchant.Dict("en_US")
    from feature_extraction import tokenize_document
    tokenized_documents = [tokenize_document(document) for document in documents]
    n_words = [len(c.split()) for c in documents]
    #n_words = [len(document) for document in tokenized_documents]
    n_chars = [len(c) for c in documents]
    n_dwords = [sum(1 for word in document if d.check(word)) for document in tokenized_documents]

    n_you_re = [len(re.findall(self.__you_re, document)) for document in documents]
    n_you = [len(re.findall(self.__you, document)) for document in documents]

    # number of uppercase words
    allcaps = [np.sum([w.isupper() for w in comment.split()])
           for comment in documents]
    # longest word
    #max_word_len = [np.max([len(w) for w in c.split()]) for c in documents]
    # average word length
    #mean_word_len = [np.mean([len(w) for w in c.split()])
    #                                    for c in documents]
    # number badwords:
    n_bad = [np.sum([c.lower().count(w) for w in self.__badwords]) for c in documents]
    exclamation = [c.count("!") for c in documents]
    addressing = [c.count("@") for c in documents]

    allcaps_ratio = np.array(allcaps) / np.array(n_words, dtype=np.float)
    bad_ratio = np.array(n_bad) / np.array(n_words, dtype=np.float)
    dic_ratio = np.array(n_dwords) / np.array(n_words, dtype=np.float)

    return np.array([n_words, n_chars, n_dwords, n_you_re, n_you, exclamation, allcaps,
                     addressing, bad_ratio, n_bad, allcaps_ratio, dic_ratio]).T
```

Custom Classifier (III)

```
def train_custom(categories, comments, badwords):
    from sklearn.pipeline import Pipeline
    from sklearn.svm import LinearSVC

    text_clf = Pipeline([('vect', CustomTransformer(badwords)),
                         ('clf', LinearSVC(random_state=42))])
    text_clf = text_clf.fit(comments, categories)
    return text_clf
```

```
Custom Model Result

Accuracy: 0.825840574235

             precision    recall  f1-score   support

   NoInsult       0.84      0.94      0.89      1954
     Insult       0.75      0.50      0.60       693

avg / total       0.82      0.83      0.81      2647
```

Ensembling: Voting Classifier

```
def train_assembling_average(categories, comments, badwords):
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.pipeline import Pipeline
    from sklearn.linear_model import SGDClassifier
    from sklearn.ensemble import VotingClassifier

    text_clf = Pipeline([('vect', TfidfVectorizer(lowercase=True, ngram_range=(1, 3), analyzer="word", min_df=3)),
                         ('clf', SGDClassifier(loss='log', penalty='l2', alpha=1e-3, n_iter=5, random_state=42))])

    custom = CustomTransformer(badwords)
    clf = Pipeline([('vect', custom),
                    ('clf', SGDClassifier(loss='log', penalty='l2', alpha=1e-3, n_iter=5, random_state=42))])

    final_classifier = VotingClassifier(estimators=[('text', text_clf), ('custom', clf)],
                                        voting='soft', weights=[3,1])
final_classifier = final_classifier.fit(comments, categories)
return final_classifier
```

```
Accuracy: 0.837551945599

             precision    recall  f1-score   support

   NoInsult       0.89      0.89      0.89      1954
     Insult       0.69      0.69      0.69       693

avg / total       0.84      0.84      0.84      2647
```

ENSEMBLING: FEATURE UNION

```
def train_assembling(categories, comments, badwords):
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.linear_model import LogisticRegression
    from sklearn.pipeline import Pipeline, FeatureUnion
    from sklearn.feature_selection import SelectPercentile, chi2

    select = SelectPercentile(score_func=chi2, percentile=70)
    countvect_word = TfidfVectorizer(lowercase=True, ngram_range=(1, 3), analyzer="word", binary=False, min_df=3)
    custom = CustomTransformer(badwords)
    union = FeatureUnion([("custom", custom),("words", countvect_word)])
    clf = LogisticRegression(tol=1e-8, penalty='l2', C=4)
    classifier = Pipeline([('vect', union), ('select', select), ('clf', clf)])
    classifier = classifier.fit(comments, categories)
    return classifier
```

```
Feature Ensemble Model Result

Accuracy: 0.85266339252

             precision    recall  f1-score   support

   NoInsult       0.87      0.95      0.90      1954
     Insult       0.80      0.59      0.68       693

avg / total       0.85      0.85      0.84      2647
```

Conclusions

- High Precision over a (relatively) small train dataset
- Recall must be improved, probably by improving the lexicons or finding better features for insults

"If a machine is expected to be infallible, it cannot also be intelligent."

– **Alan Turing**