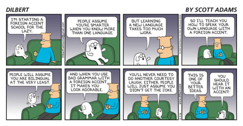
nlp-intro

March 18, 2017

Natural Language Processing
an introduction using python
Casey Kennington, PhD
Boise Code Camp 2017


DILBERT
BY SCOTT ADAMS
I'M STARTING A FOREIGN ACCENT SCHOOL FOR THE LAZY.
PEOPLE ASSUME YOU'RE SMARTER WHEN YOU KNOW MORE THAN ONE LANGUAGE.
BUT LEARNING A NEW LANGUAGE TAKES TOO MUCH WORK.
SO I'LL TEACH YOU HOW TO SPEAK YOUR OWN LANGUAGE WITH A FOREIGN ACCENT.
PEOPLE WILL ASSUME YOU ARE BILINGUAL AT THE VERY LEAST.
AND WHEN YOU USE BAD GRAMMAR WITH A FOREIGN ACCENT, IT MAKES YOU LOOK ADORABLE.
YOU'LL NEVER NEED TO DO ANOTHER COURTESY LAUGH, EITHER. PEOPLE WILL JUST ASSUME YOU DIDN'T GET THE JOKE.
THIS IS ONE OF YOUR BETTER IDEAS.
YOU SHOULD HEAR IT WITH AN ACCENT!

title

0.0.1 What is NLP?

- Answer: machine learning applied to language data and tasks

What is Machine Learning?

- Answer: *f(x)*
- Programmers do a lot of things. Foremost is writing rules into functions (programs, classes, etc.). Machine learning is when machines write those functions themselves.
- ML is algorithms that learn functions.

0.0.2 How can machines learn?

- Answer: data
- Answer: probability and information theory (i.e., statistics)
- Answer: mathematical models
- Answer: for NLP, knowledge about linguistics

Discussion: There are many ways to put data, probabilitiy/information theory, mathematical models, and linguistic knowledge together. You've likely heard of classifers which can take various information and yield some kind of discrete label. You've heard of artificial neural networks and deep learning which can take continuous data, select the information that is useful, and produce either a discrete label or some kind of high-dimensional representation within the layers. NLP has a lot of ties to artificial intelligence; in fact, the more interesting unsolved problems of AI, such as language understanding and spoken interaction, are largely NLP tasks.

0.0.3 Applications

- Automatic Speech Recognition (ASR)
- Text to Speech Synthesis (TTS)
- Dialogue Systems / Digital Personal Assistants (e.g., Siri, Cortana . . .)
- Chatbots
- Search / Information Retrieval
- Text Analytics
- Sentiment Analysis
- Machine Translation
- Spam Filtering
- Spell / Grammar Checking
- Summarization (e.g., of meetings or news)
- Automatic image description generation
- ....

1 Example: Predicting a Name's Type

Scenario: Suppose you work for a company that is trying to keep track of all movies ever made. So you write a program (using NLP techniques) that can extract names from texts like Wikipedia. For example:

- Music Through the Years
- Walking on the Wild Side

But you also find that many of the names don't belong to movies. For example:

- Dihistine Expectorant 131
- Carlos Saavedra Lamas
- American Retirement Corp.

In fact, you find that many of the names you extract are things like people's names, drug/medicine names, as well as names of companies.

Task: use a classifier to determine the type of a name. So you gather as much data as you can and you go through a lot of names and label them. As you do so, you don't really notice any patterns in determining what names belong to which type. So you can't just write a function with some rules. Instead, you try some ML/NLP.

```
In [39]: import pandas as pd

In [40]: data = pd.read_csv('pnp-train.txt',names=['type','name'], delimiter='\t', encoding='ISO

In [41]: data[:10]

Out[41]:      type                             name
         0    drug                          Dilotab
         1   movie  Beastie Boys: Live in Glasgow
         2  person          Michelle Ford-Eriksson
```

```
3   place                   Ramsbury
4   place            Market Bosworth
5    drug      Cyanide Antidote Package
6  person               Bill Johnson
7   place                   Ettalong
8   movie            The Suicide Club
9   place                    Pézenas
```

Using this data, we can try using the words in the names to predict the type. First, we'll want to normalize the text a little bit by making everything lower case.

```
In [4]: data['split_names'] = data['name'].map(lambda x: x.strip().lower().split())
```

```
In [5]: data[:10]
```

```
Out[5]:     type                         name                    split_names
        0    drug                      Dilotab                      [dilotab]
        1   movie  Beastie Boys: Live in Glasgow  [beastie, boys:, live, in, glasgow]
        2  person        Michelle Ford-Eriksson     [michelle, ford-eriksson]
        3   place                     Ramsbury                     [ramsbury]
        4   place              Market Bosworth             [market, bosworth]
        5    drug        Cyanide Antidote Package  [cyanide, antidote, package]
        6  person                 Bill Johnson                [bill, johnson]
        7   place                     Ettalong                     [ettalong]
        8   movie              The Suicide Club            [the, suicide, club]
        9   place                      Pézenas                      [pézenas]
```

```
In [6]: import nltk
        from nltk.classify.naivebayes import NaiveBayesClassifier
        import collections as c
```

1.1 Training Step using a Naive Bayes Classifier

```
In [7]: # train
        data['feats'] = data['split_names'].map(lambda x: c.Counter(x))
        train_data = list(zip(data['feats'], data['type']))

        classifier = NaiveBayesClassifier.train(train_data)
```

1.2 Evaluation Step

```
In [8]: # evaluate
        nltk.classify.util.accuracy(classifier, train_data)
```

```
Out[8]: 0.9761439931431837
```

1.2.1 Our classifier can predict the correct name type over 97% of the time!

Actually, it can't.

1.3 It is a cardinal sin in ML / NLP to eavaluate on your training data.

Fortuntely, we've separated some data to use for evaluation.

```
In [9]: test = pd.read_csv('pnp-test.txt',names=['type','name'], delimiter='\t', encoding='ISO-8
        test['split_names'] = test['name'].map(lambda x: x.strip().lower().split())
        test['feats'] = test['split_names'].map(lambda x: c.Counter(x))
        test_data = list(zip(test['feats'], test['type']))
```

```
In [10]: # evaluate
         nltk.classify.util.accuracy(classifier, test_data)
```

```
Out[10]: 0.6693333333333333
```

1.3.1 That's a more reliable measure of how your classifier will function given new names.

Discussion

- Important in ML/NLP is generalizability (the opposite of over-training)
- Always split up your data into train/dev/test. Use train/dev while you are programming, working out tweaks, munging data, adjusting your classifier, then when you are done try out your classifier on the test data.
- A large portion of ML/NLP is dealing with data.

1.4 What's in a classifier?

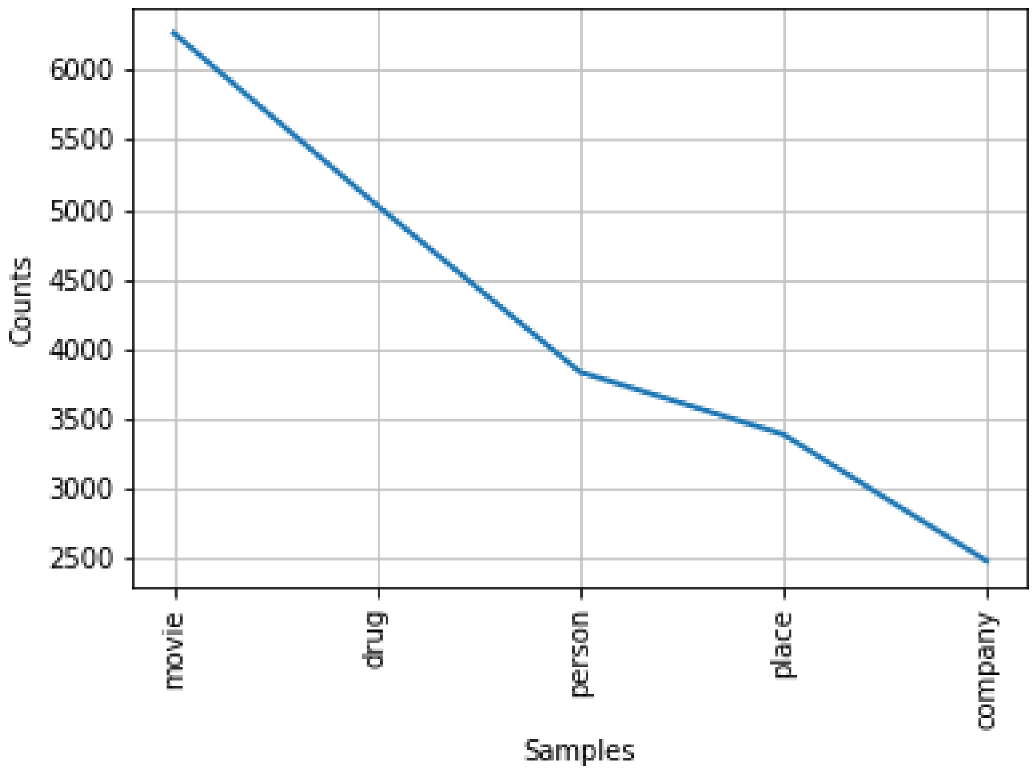
1.4.1 Let's take a closer look at how the Naive Bayes Classifier works

What kind of information can we get from our data?

- We know the types
- We know the names
- We know which names belong to which type

We can use probability theory to help us. Are some types more common than others? We can easily plot this information.

```
In [11]: %matplotlib inline
         import nltk

         nltk.FreqDist(data['type']).plot()
```



Let's use this frequency information to our advantage! We can write a simple probability function that can return the probability of a type. It determines that probability by counting the relative frequencies of the types.

2 P(T)

```
In [12]: types = set(data['type'])

         type_count = c.Counter(data['type'])

         def Ptype(T=''):
             return type_count[T] / float(len(data))
```

What about the words? The frequency of the words should be taken into account in our classifier. We can write a simple probability function that returns the probability of a word. It determines that probability by counting up the relative frequencies of the words.

3 P(W)

```
In [13]: all_words = [word for name in data['split_names'] for word in name]
In [14]: word_count = c.Counter(all_words)
         total_word_count = sum(word_count.values())

         def Pword(W=''):
             if W not in word_count: return 0.000001 # what's this? "stupid smoothing" for when
             return word_count[W] / total_word_count
```

3.0.1 What about both words and types? We should somehow model the relationship between words and types. We can write a conditional probability function that counts up the words for each individual type.

4 P(W | T)

```
In [15]: counters = {}
         total_counts = {}
         for t in types:
             sub = data[data['type'] == t]
             counters[t] = c.Counter(w for line in sub['split_names'] for w in line)
             total_counts[t] = sum(counters[t].values())

         def Pwt(W='', T=''):
             if W not in counters[T]: return 0.000001 # again, it we see novel words
             return counters[T][W] / total_counts[T]
```

4.0.1 But what we really want is to predict the type given the words, right? How can we model that given what we have?

Answer: Bayes' Rule

4.0.2

$$P(T|W) = \frac{P(W|T)P(T)}{P(W)}$$

```
In [16]: # Bayes' Rule

         def Ptw(T='', W=''):
             return Pwt(W=W, T=T) * Ptype(T=T) / Pword(W=W)
```

```
In [43]: for t in types:
             print(t, Pwt(W='the', T=t))
```

```
person 1e-06
movie 0.05773911242297136
company 0.007047134935304991
drug 1e-06
place 0.0004970178926441351
```

4.0.3 But Bayes' Rule only cares about individual words. How do we predict the type given a name which is made up of multiple words?

Answer: multiply.

4.1

$$P(T|W_1...W_n) = \prod_w P(T|W = w)$$

4.1.1 This constitutes the Naive Bayes Classifier.

```
In [17]: import numpy as np

         def P(T='',N=''):
             return np.prod([Ptw(T=T, W=w) for w in N])
```

4.1.2 Evaluating our classifier.

Evaluate by taking each name and asking your classifier what it thinks the probability of the type is given the name. Take the type that got the highest probability as your guess and see if your classifier was right using the testing data.

```
In [18]: cor = 0.0
         for typ,name in zip(test['type'],test['split_names']):
             probs = [(t, P(T=t, N=name)) for t in types]
             m = ('',0.0)
             for t,p in probs:
                 if p > m[1]: m = (t,p)
             if typ == m[0]: cor +=1.0

         cor / float(len(test))
```

```
Out[18]: 0.6792380952380952
```

4.1.3 Which is slightly higher than the NLTK implementation of the Naive Bayes Classifier.

4.1.4 Discussion:

- Naive Bayes is a very very simple classifier that makes a lot of *independence assumptions*. For example, it assumes that the words have no relation to each other (but this is clearly wrong!).

5 Analyzing Sentiment

What is sentiment analysis? Given some text, try to determine if the sentiment of the writer of that text was positive, negative,or neutral. This is fairly easy for humans to do, but we want to automate this because we want to know if something (e.g., our restaurant, movie, book, or other product) is being talked about in real-time on social media, and, more importantly, what people think about our product. Knowing if a tweet or a post is positive or negative can help us see how our product is being received and how we can improve it.

Example: sentiment about election candidates in Belgium: http://www.clips.ua.ac.be/pages/pattern-examples-elections

- some data: http://help.sentiment140.com/for-students

Columns:

```
0 - the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)
1 - the id of the tweet (2087)
2 - the date of the tweet (Sat May 16 23:58:44 UTC 2009)
3 - the query (lyx). If there is no query, then this value is NO_QUERY.
4 - the user that tweeted (robotickilldozr)
5 - the text of the tweet (Lyx is cool)
```

```
In [44]: cols = ['polarity','id', 'date', 'query', 'user', 'tweet']

         data = pd.read_csv('sentiment.train.csv',names=cols, encoding='ISO-8859-1')
         print('length of data {}'.format(len(data)))
         test = pd.read_csv('sentiment.test.csv',names=cols, encoding='ISO-8859-1')
         print('length of test {}'.format(len(test)))
         data = pd.concat([data,test])
         print('length of both {}'.format(len(data)))
```

```
length of data 1600000
length of test 498
length of both 1600498
```

```
In [20]: data=data.sample(frac=0.1,random_state=200) # this is a lot of data, while we develop l
         data = data.drop(['id', 'date', 'query', 'user'], axis=1)
```

```
In [21]: data[:5]
```

```
Out[21]:         polarity                                              tweet
         888312         4  Breaky burrito at Whole Foods is a good way to...
         516573         0  i'm out! gonna check my facebook. please!!!!!!...
         970735         4  yay just won mac msf in petticoat on ebay and ...
         862961         4                      @shezDOPEx3 i love you more
         122643         0            @mahdi Maybe the problem is from my ISP
```

```
In [22]: c.Counter(data.polarity)
```

```
Out[22]: Counter({0: 80028, 2: 12, 4: 80010})
```

We have a fairly even spread between positive (0) and negative tweets (4). I want to get rid of the neutral tweets.

```
In [23]: data = data[data['polarity'] != 2] # remove anything where polarity is 2
```

```
In [24]: c.Counter(data.polarity)
```

```
Out[24]: Counter({0: 80028, 4: 80010})
```

Now I want to represent the polarity as a string neg or pos.

```
In [46]: data['polarity'] = data.polarity.map(lambda x: 'neg' if x == 0 else 'pos')

In [47]: data[:10]

Out[47]:   polarity          id                          date     query  \
         0      neg  1467810369  Mon Apr 06 22:19:45 PDT 2009  NO_QUERY
         1      neg  1467810672  Mon Apr 06 22:19:49 PDT 2009  NO_QUERY
         2      neg  1467810917  Mon Apr 06 22:19:53 PDT 2009  NO_QUERY
         3      neg  1467811184  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY
         4      neg  1467811193  Mon Apr 06 22:19:57 PDT 2009  NO_QUERY
         5      neg  1467811372  Mon Apr 06 22:20:00 PDT 2009  NO_QUERY
         6      neg  1467811592  Mon Apr 06 22:20:03 PDT 2009  NO_QUERY
         7      neg  1467811594  Mon Apr 06 22:20:03 PDT 2009  NO_QUERY
         8      neg  1467811795  Mon Apr 06 22:20:05 PDT 2009  NO_QUERY
         9      neg  1467812025  Mon Apr 06 22:20:09 PDT 2009  NO_QUERY

                      user                                              tweet
         0  _TheSpecialOne_  @switchfoot http://twitpic.com/2y1zl - Awww, t...
         1    scotthamilton  is upset that he can't update his Facebook by ...
         2         mattycus  @Kenichan I dived many times for the ball. Man...
         3          ElleCTF    my whole body feels itchy and like its on fire
         4           Karoli  @nationwideclass no, it's not behaving at all...
         5         joy_wolf                      @Kwesidei not the whole crew
         6          mybirch                                        Need a hug
         7             coZZ  @LOLTrish hey  long time no see! Yes.. Rains a...
         8  2Hood4Hollywood               @Tatiana_K nope they didn't have it
         9          mimismo                          @twittera que me muera ?
```

split the data into training, dev, and test data

```
In [26]: temp=data.sample(frac=0.2,random_state=200)
         train=data.drop(temp.index)
         dev=temp.sample(frac=0.5,random_state=200)
         test=temp.drop(dev.index)

         train.shape, dev.shape, test.shape

Out[26]: ((128029, 2), (16004, 2), (16004, 2))

In [27]: '''
             Given a dataframe with a 'tweet' and 'polarity' column, this will prepare the featu
         '''

         def prep_data(df):
             df['split_tweet'] = df.tweet.map(lambda x: x.strip().lower().split())
             df['feats'] = df['split_tweet'].map(lambda x: c.Counter(x))
             return list(zip(df['feats'], df['polarity']))
```

```
In [28]: # train
         train_data = prep_data(train)
         classifier = NaiveBayesClassifier.train(train_data)

/home/casey/.local/lib/python3.5/site-packages/ipykernel/__main__.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
/home/casey/.local/lib/python3.5/site-packages/ipykernel/__main__.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
```

```
In [29]: dev_data = prep_data(dev)
         nltk.classify.util.accuracy(classifier, dev_data)

Out[29]: 0.735628592851787
```

Question:

- This is good, right?

Answer:

- It all depends on your **baseline**. Since the neg/pos frequency is about 50/50, then we know our classifier is doing well when we are above that baseline. This is called the *most common* baseline (i.e., what is the accuracy of a random classifier that simple chooses the most common class label/type?) Right now we are about 26% above that baseline, so not bad.
- Another kind of baseline is the *random* baseline which is basically 1/len(types) which in this case is 2. So the random baseline here is the same as the most common baseline beacuse it's a binary problem. If we had 4 possible types, then the random baseline would be 25%, but the most common baseline would be the relative frequency of the most common type in the training data.

A better baseline is actually what we have above, which is our > 73% accuracy score. This is called a "bag of words" baseline. It uses the simplest features: words. Let's work from here to make our classifier work better.

5.1 How can we improve above the baseline?

Answers:

- Use a different classifier / model (or invent your own)
- Get more data
- Fix your noisy/messy data
- Apply some knowledge about language (feature engineering)

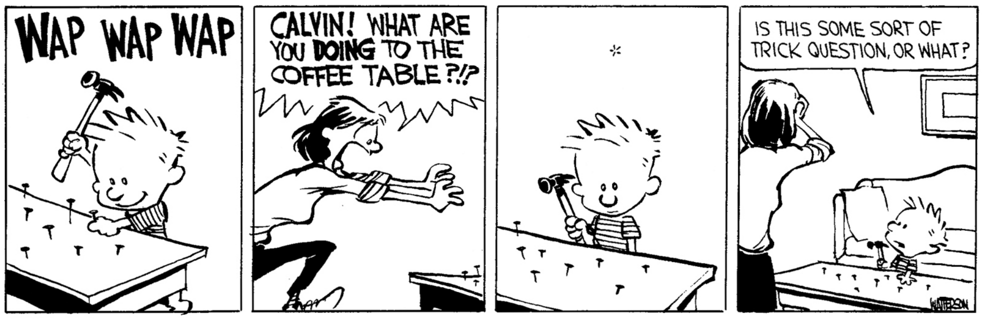
- some ideas: 01 recipes_exploratory_analysis from https://github.com/bonzanini/nlp-tutorial

```
In [30]: from nltk.classify import MaxentClassifier

In [31]: me_classifier = MaxentClassifier.train(train_data,max_iter=5)

  ==> Training (5 iterations)

      Iteration    Log Likelihood    Accuracy
      ---------------------------------------
             1          -0.69315        0.500
             2          -0.58556        0.884
             3          -0.51267        0.903
             4          -0.46010        0.911
         Final          -0.42025        0.918


In [32]: nltk.classify.util.accuracy(me_classifier, dev_data)

Out[32]: 0.7614346413396651

In [ ]:

In [34]: import nltk
         import string
         from nltk.tokenize import TweetTokenizer

         tknzr = TweetTokenizer(strip_handles=True, reduce_len=True) #something better than just

         def prep_data(df):
             df['split_tweet'] = df.tweet.map(lambda x: tknzr.tokenize(x))
             df['feats'] = df.split_tweet.map(lambda x: c.Counter(x))
             return list(zip(df['feats'], df['polarity']))

In [35]: train_data = prep_data(train)
         dev_data = prep_data(dev)

/home/casey/.local/lib/python3.5/site-packages/ipykernel/__main__.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
/home/casey/.local/lib/python3.5/site-packages/ipykernel/__main__.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#
```

```
In [36]: classifier = NaiveBayesClassifier.train(train_data)
         nltk.classify.util.accuracy(classifier, dev_data)

Out[36]: 0.7509997500624844

In [37]: me_classifier = MaxentClassifier.train(train_data,max_iter=5)
         nltk.classify.util.accuracy(me_classifier, dev_data)

  ==> Training (5 iterations)

      Iteration    Log Likelihood    Accuracy
      ---------------------------------------
             1          -0.69315        0.500
             2          -0.61275        0.835
             3          -0.55441        0.852
             4          -0.51052        0.858
         Final          -0.47640        0.862

Out[37]: 0.7704948762809297
```

5.2 Concluding Remarks

- Getting into ML/NLP isn't impossible, but it takes effort. Spend time learning about probability and information theory, linear algebra, programming, libraries and APIS (e.g., pandas, nltk, and scikit-learn in Python), and data manipulation.
- Be sure you are honest in your evaluations: never evaluate on training data!
- There are many dichotomties to ML/NLP:
- supervised vs unsupervised learning
- generative vs discriminative models
- continuous vs discrete models
- classification vs regression
- ...
- There are some nice online tutorials and courses at Boise State University that one can take.


WAP WAP WAP
CALVIN! WHAT ARE YOU DOING TO THE COFFEE TABLE ?!?
IS THIS SOME SORT OF TRICK QUESTION, OR WHAT?
WATTERSON

title

```
In [ ]:

In [ ]:
```