2017 Intro to NLP

Presented by Michael Beilman
July 25th, 2017

https://github.com/beilmanmich/nlp_101

Introduction

Unstructured Data, What's NLP? Requirements/Resources, Why Python? Learning outcomes and objectives

Python 101

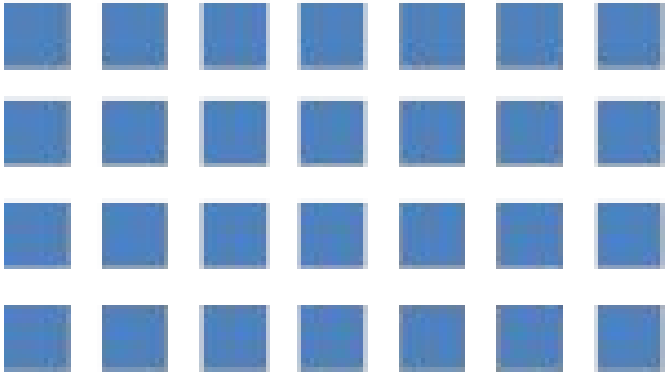
Quick foundations: Jupyter, Lists, Dictionaries, For Loops, Functions

NLP with Python 101

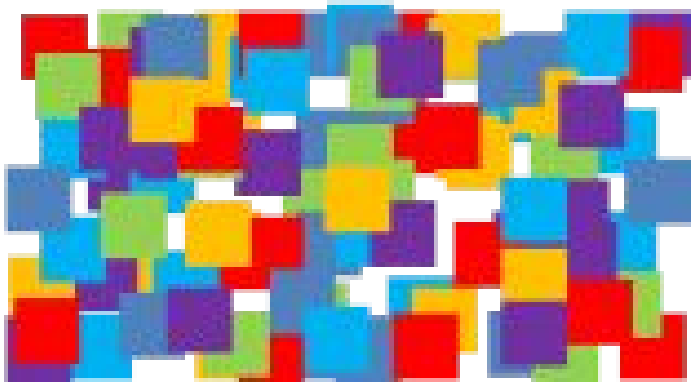
A few quick examples

USE CASES

Structured Data

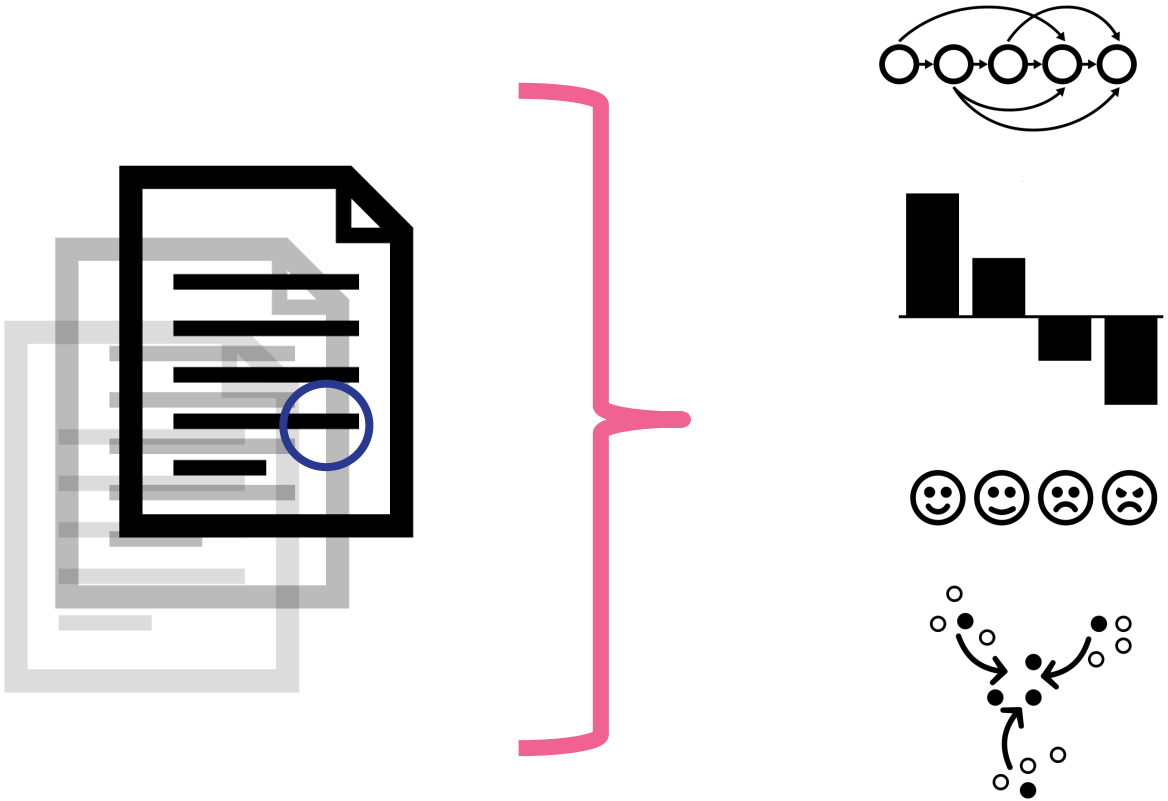


Unstructured Data



According to a 2006 Merrill Lynch study, **~80% of all potentially usable business information** may originate in unstructured form

Natural Language Processing - NLP



Common Uses

Natural Language Processing

- Automatic text summarization
- Sentiment analysis
- Topic extraction
- Named entity recognition
- Parts-of-speech tagging
- Relationship extraction
- Stemming
- Text mining
- Machine translation
- Automated question answering

Recent Examples

It’s everywhere

- Amazon Alexa, Google, Siri
- Chatbots (verbal/written)
- Marketing - twitter sentiment (smart KPIs), churn analysis
- Advanced surveys
- Information retrieval: e-litigation, Google search, autocomplete
- Many examples across social science research: predicting crime

Resources & Requirements

Python

Following installation directions from Continuum Anaconda:

https://docs.continuum.io/anaconda/install/

NLTK

Natural language toolkit, installation:

http://www.nltk.org/

CLI:

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

Jupyter Notebooks

Allows an interactive "sandbox" environment:

http://jupyter.readthedocs.io/en/latest/install.html

```
$ cd /nlp_tutorial
$ ipython notebook
```

What & Why Python?

- It’s really a preference (part of Data Science stack)
- Python is an open source general scripting language
- Python Advantages:
  - Readable
  - Fast for prototypes
  - Rich text & list support
  - Lots of great NLP-related libraries
    - NLTK, TextBlob, spaCy, scipy bundle, pandas, itertools
  - Great parsing libraries:
    - BeautifulSoup; Scrapy, Tweepy
    - Regex, Itertools
  - Data type relative non-issue (json, xml, html, etc.)
  - Jupyter Notebooks
  - Free (Open Source)
  - Large community

Learning Objectives

Appreciation

By the end of this overview we will have an appreciation for Python, and the vast potential of text-based unstructured data. Data driven decision making in a new light.

Introduction

Python 101

Quick foundations: Jupyter, Lists, Dictionaries, For Loops, Functions

NLP with Python

USE CASES

Change to directory

```
[Michaels-MacBook-Pro-2:~ michaelbeilman$ cd Projects/nlp_tutorial
[Michaels-MacBook-Pro-2:nlp_tutorial michaelbeilman$ ipython notebook
```

Launch Jupyter Notebook

localhost:8888/tree?token=d2e954b15d91ea9ac05065713f9ae11e8b2e7d

jupyter

Logout

Files | Running | Clusters

Select items to perform actions on them.

Upload | New

- 1. Quick Python Refresher
- 2. NLTK and the Basics
- 3. Tokenization, Tagging, Chunking
- 4. Custom Sources
- 5. Projects
- Untitled Folder
- README.md

LISTS

Linked to GitHub

A Python list stores comma separated values. In our cases these values will be strings, and numbers.

```
In [1]: mylist = ["a","b","c"]

In [2]: mylist
Out[2]: ['a', 'b', 'c']

In [3]: mylist2 = [1,2,3,4,5]
        mylist2
Out[3]: [1, 2, 3, 4, 5]
```

Each item in the list has a position or index. By using a list index you can get back individual list item.

```
In [4]: mylist[0]

Out[4]: 'a'

In [5]: mylist2[0]

Out[5]: 1
```

Remember that in programming, counting starts at 0, so to get the first item, we would call index 0.

The first number tells us where to start while the second tells us where to end and is exclusive. If we don't enter the fist number, we will get back the first x items, where x is the second index number we provide.

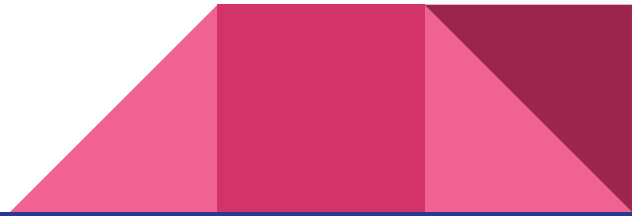
```
In [7]: mylist[:2]
Out[7]: ['a', 'b']

In [8]: mylist2[:3]
Out[8]: [1, 2, 3]

In [9]: mylist[-2:]
Out[9]: ['b', 'c']
```

Python can also easily measure the shape of stored lists, important for NLP

```
In [10]: len(mylist)

Out[10]: 3

In [11]: len(mylist2)

Out[11]: 5
```

DICTIONARIES

Linked to GitHub



A Python dictionary stores key-value pairs.

In [1]:
```
d = {
    'Python': 'programming',
    'English': "natural",
    'French': 'natrual',
    'Ruby' : 'programming',
    'Javascript' : 'programming'
}
```

In [2]:
```
d
```

Out[2]:
```
{'English': 'natural',
 'French': 'natrual',
 'Javascript': 'programming',
 'Python': 'programming',
 'Ruby': 'programming'}
```

In [3]:
```
type(d)
```

Out[3]:
```
dict
```

QUICK QUIZ!

What is the *value* for the *key* “**Python**”? How about the *key* “**English**”?

```
Out[2]: {'English': 'natural',
         'French': 'natrual',
         'Javascript': 'programming',
         'Python': 'programming',
         'Ruby': 'programming'}
```

QUICK QUIZ!

What is the *value* for the *key* "**Python**"? How about the *key* "**English**"?

```
Out[2]: {'English': 'natural',
         'French': 'natrual',
         'Javascript': 'programming',
         'Python': 'programming',
         'Ruby': 'programming'}
```

```
In [4]: d["Python"]

Out[4]: 'programming'
```

```
In [5]: d["English"]

Out[5]: 'natural'
```

We can add new entries to a dictionary.

```
In [6]: d['Scala'] = 'programming'

In [7]: d

Out[7]: {'English': 'natural',
         'French': 'natrual',
         'Javascript': 'programming',
         'Python': 'programming',
         'Ruby': 'programming',
         'Scala': 'programming'}
```

Values can also be numbers.

```
In [8]: d["languages known"] = 3

In [9]: d

Out[9]: {'English': 'natural',
         'French': 'natrual',
         'Javascript': 'programming',
         'Python': 'programming',
         'Ruby': 'programming',
         'Scala': 'programming',
         'languages known': 3}
```

Dictionaries -- **Key:** and **:Value** pairs...

```
{'English': 'natural',
 'French': 'natrual',
 'Javascript': 'programming',
 'Python': 'programming',
 'Ruby': 'programming',
 'Scala': 'programming',
 'languages known': 3}
```

```
In [10]: d.keys()
```

```
Out[10]: ['Scala',
          'Python',
          'Javascript',
          'French',
          'English',
          'Ruby',
          'languages known']
```

Dictionaries -- **Key:** and **:Value** pairs...

```
{'English': 'natural',
 'French': 'natrual',
 'Javascript': 'programming',
 'Python': 'programming',
 'Ruby': 'programming',
 'Scala': 'programming',
 'languages known': 3}
```

```
In [11]: d.values()
```

```
Out[11]: ['programming',
          'programming',
          'programming',
          'natrual',
          'natural',
          'programming',
          3]
```

And similar to lists, Pythonic comprehension of size is natural

```
{'English': 'natural',
 'French': 'natrual',
 'Javascript': 'programming',
 'Python': 'programming',
 'Ruby': 'programming',
 'Scala': 'programming',
 'languages known': 3}
```

```
In [12]: len(d)

Out[12]: 7
```

LOOPS & CONDITIONALS

Linked to GitHub

```
In [1]: mylist = ["Python","Ruby","Javascript","HTML"]
```

```
In [2]: mylist
Out[2]: ['Python', 'Ruby', 'Javascript', 'HTML']
```

We can iterate in Python through lists (and dictionaries) like this.

```
In [3]: for item in mylist:
            print item

Python
Ruby
Javascript
HTML
```

An alternative loop...

```
In [4]: [item for item in mylist]
Out[4]: ['Python', 'Ruby', 'Javascript', 'HTML']
```

Loops help us perform repeated actions, often across items (from a list/dict)...

```
In [5]: for item in mylist:
            print item + " is a fun programming language."

Python is a fun programming language.
Ruby is a fun programming language.
Javascript is a fun programming language.
HTML is a fun programming language.
```

They can also be used to quickly store information.

```
In [6]: newlist = [item + " is a fun programming language." for item in mylist]

In [7]: newlist

Out[7]: ['Python is a fun programming language.',
         'Ruby is a fun programming language.',
         'Javascript is a fun programming language.',
         'HTML is a fun programming language.']
```

We can use if statements to look for special conditions.

In [8]:
```
x = 10
```

In [9]:
```
if x > 5:
    print "It looks like x is greater than 5"
```

```
It looks like x is greater than 5
```

In [10]:
```
if x > 5 and x < 20:
    print "Hello"
```

```
Hello
```

In [11]:
```
number_list = [1,2,3,4,5,6,7,8,9,10]
```

We can use for loops to execute multiple conditions...

In [12]:
```
for number in number_list:
    if number%2 == 0:
        print str(number) + " is even"
    elif number == 7:
        print str(number) + " is the best number!"
    else:
        print str(number) + " is odd"
```

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
6 is even
7 is the best number!
8 is even
9 is odd
10 is even
```

FUNCTIONS

Linked to GitHub

We can define a function by giving it a name, and telling the function any values we plan on passing along.

```
In [1]: def my_function(something):
            return something

In [2]: my_function("hello")
Out[2]: 'hello'

In [3]: my_function(2)
Out[3]: 2
```

We can implement more complex functions

```
In [4]: def information(word):
            return "Word: " + str(word) + ", Length: " + str(len(word))

In [5]: information("hello")
Out[5]: 'Word: hello, Length: 5'

In [6]: information("language")
Out[6]: 'Word: language, Length: 8'
```

Introduction

Python 101

NLP with Python 101

A few quick examples

USE CASES

NLTK

Natural Language Toolkit
*the entire kitchen sink

spaCy

Industrial strength NLP
*more practical

Counting Text
Frequency Dist
Bigrams
Regular Expressions

Tokenization
Tagging
Normalization
Chunking

Text Files
HTML
URL
CSV

Sentiment Analysis
Text Classification
Term Frequency - Index Frequency (TFIDF)
Lemmetization

NLTK Quick Example

Analyzing presidential inaugural speeches

Linked to GitHub

Let's analyze Presidential Inaugural Speeches...

```
In [32]: import nltk

In [33]: from nltk.corpus import inaugural

In [34]: inaugural.fileids()

Out[34]: [u'1789-Washington.txt',
          u'1793-Washington.txt',
          u'1797-Adams.txt',
          u'1801-Jefferson.txt',
          u'1805-Jefferson.txt',
          u'1809-Madison.txt',
          u'1813-Madison.txt',
          u'1817-Monroe.txt',
          u'1989-Bush.txt',
          u'1993-Clinton.txt',
          u'1997-Clinton.txt',
          u'2001-Bush.txt',
          u'2005-Bush.txt',
          u'2009-Obama.txt']
```

Let's loop through the speeches and calculate total words...

In [35]:
```
for speech in inaugural.fileids():
    words_total = len(inaugural.words(speech))
    print words_total, speech
```

```
1538 1789-Washington.txt
147 1793-Washington.txt
2585 1797-Adams.txt
1935 1801-Jefferson.txt
2713 1989-Bush.txt
1855 1993-Clinton.txt
2462 1997-Clinton.txt
1825 2001-Bush.txt
2376 2005-Bush.txt
2726 2009-Obama.txt
```

Let's store this data to a variable name...

```
In [36]: speech_len = [(len(inaugural.words(speech)), speech) for speech in inaugural.fileids()]
```

```
In [37]: speech_len

Out[37]: [(1538, u'1789-Washington.txt'),
          (147, u'1793-Washington.txt'),
          (2585, u'1797-Adams.txt'),
          (1935, u'1801-Jefferson.txt'),
          (2384, u'1805-Jefferson.txt'),
          (1265, u'1809-Madison.txt'),
          (1304, u'1813-Madison.txt'),
```

We can now easily calculate a **max** and a **min**, any guesses?

We can now easily calculate a **max** and a **min**, any guesses?

```
In [38]: max(speech_len)

Out[38]: (9165, u'1841-Harrison.txt')
```

We can now easily calculate a **max** and a **min**, any guesses?

```
In [38]: max(speech_len)
Out[38]: (9165, u'1841-Harrison.txt')

In [39]: min(speech_len)
Out[39]: (147, u'1793-Washington.txt')
```

Now let's write a simple for loop to calculate **words per sentence**

In [47]:
```
for speech in inaugural.fileids():
    words_total = len(inaugural.words(speech))
    sents_total = len(inaugural.sents(speech))
    print words_total/sents_total, speech
```

```
64 1789-Washington.txt
36 1793-Washington.txt
69 1797-Adams.txt
46 1801-Jefferson.txt
52 1805-Jefferson.txt
60 1809-Madison.txt
```

Now just a few steps to get our data into a Padas DataFrame

```
In [52]: import pandas as pd
```

```
In [53]: data = pd.DataFrame([int(speech[:4]),
                              len(inaugural.words(speech))/len(inaugural.sents(speech))]
                             for speech in inaugural.fileids())
```

```
In [54]: data.head()
```

Out[54]:

	0	1
0	1789	64
1	1793	36
2	1797	69
3	1801	46

GREAT! Our data is in matrix notation...

...let's add column headers...

```
In [55]: data.columns = ["year","average wps"]
```

```
In [56]: data.head(10)
```

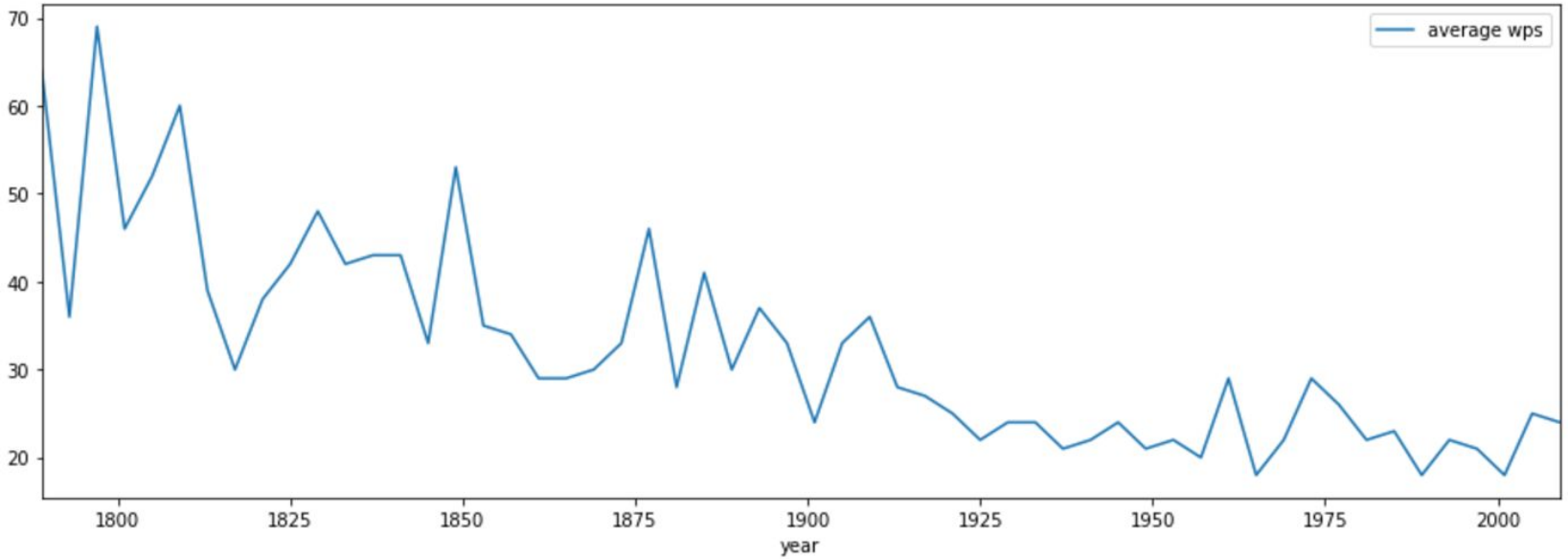
Out[56]:

	year	average wps
0	1789	64
1	1793	36
2	1797	69
3	1801	46

...let's graph and look at trends...

```
In [46]: import matplotlib
         %matplotlib inline
         data.plot("year", figsize=(15,5))
```

```
Out[46]: <matplotlib.axes._subplots.AxesSubplot at 0x114140090>
```



SO WHY IS THIS USEFUL??

NLP techniques based off *relatively* simple formulas

Automated Readability Index (ARI) =

$$4.71\left(\frac{\text{characters}}{\text{words}}\right)+0.5\left(\frac{\text{words}}{\text{sentences}}\right)-21.43$$

Flesch-Kincaid Readability =

$$206.835-1.015\left(\frac{\text{total words}}{\text{total sentences}}\right)-84.6\left(\frac{\text{total syllables}}{\text{total words}}\right)$$

Dale-Chall Readability =

$$0.1579\left(\frac{\text{difficult words}}{\text{words}}\times 100\right)+0.0496\left(\frac{\text{words}}{\text{sentences}}\right)$$

And many, many more… Coleman-Liau, Gunning-Fog, LIX, etc.

Linked to GitHub

Advanced ML builds on these foundations!

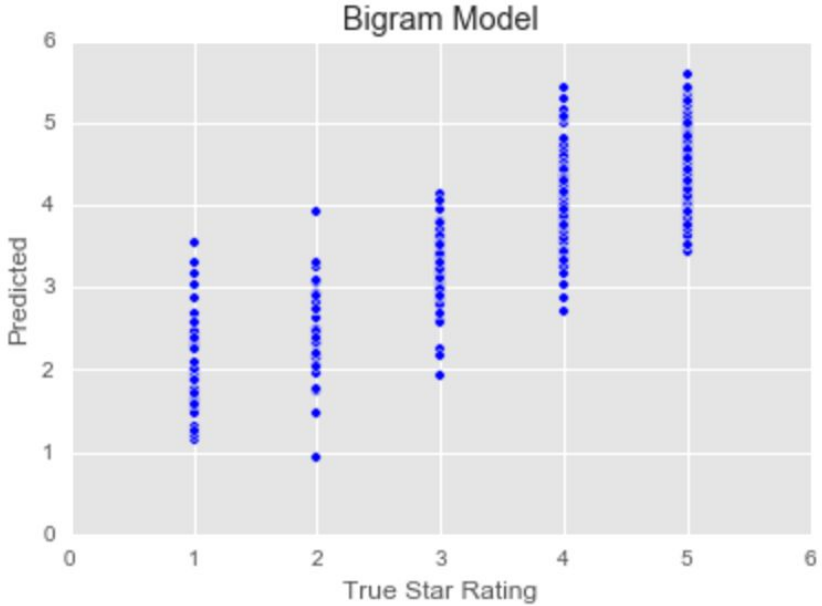
In [7]:
```
if runit:
    hv_bi = HashingVectorizer(tokenizer=LemmaTokenizer(),norm='l2',ngram_range=(2,2),stop_words=nl
tk.corpus.stopwords.words('english'))
    hv_bi_counts = hv_bi.fit_transform(sampled_df['text'])
    joblib.dump(hv_bi,'hash_vectorizer_bi.pkl')
    del(hv_bi_counts)
    #fit model
    tic=timeit.default_timer()
    bigram_pipeline = Pipeline([
    ('vect', hv_bi),
    ('lm', linear_model.SGDRegressor(n_iter=6000,alpha=.00001,penalty='l2')),
    ])
    bigram_pipeline.fit(text_train,stars_train)
    toc=timeit.default_timer()
    print toc - tic
    joblib.dump(bigram_pipeline,'bigram_pipeline.pkl')
else:
    hv_bi = joblib.load('hash_vectorizer_bi.pkl')
    bigram_pipeline=joblib.load('bigram_pipeline.pkl')
```

...how accurately can we predict Yelp “star” ratings based on review text alone?

...text carries predictive power!

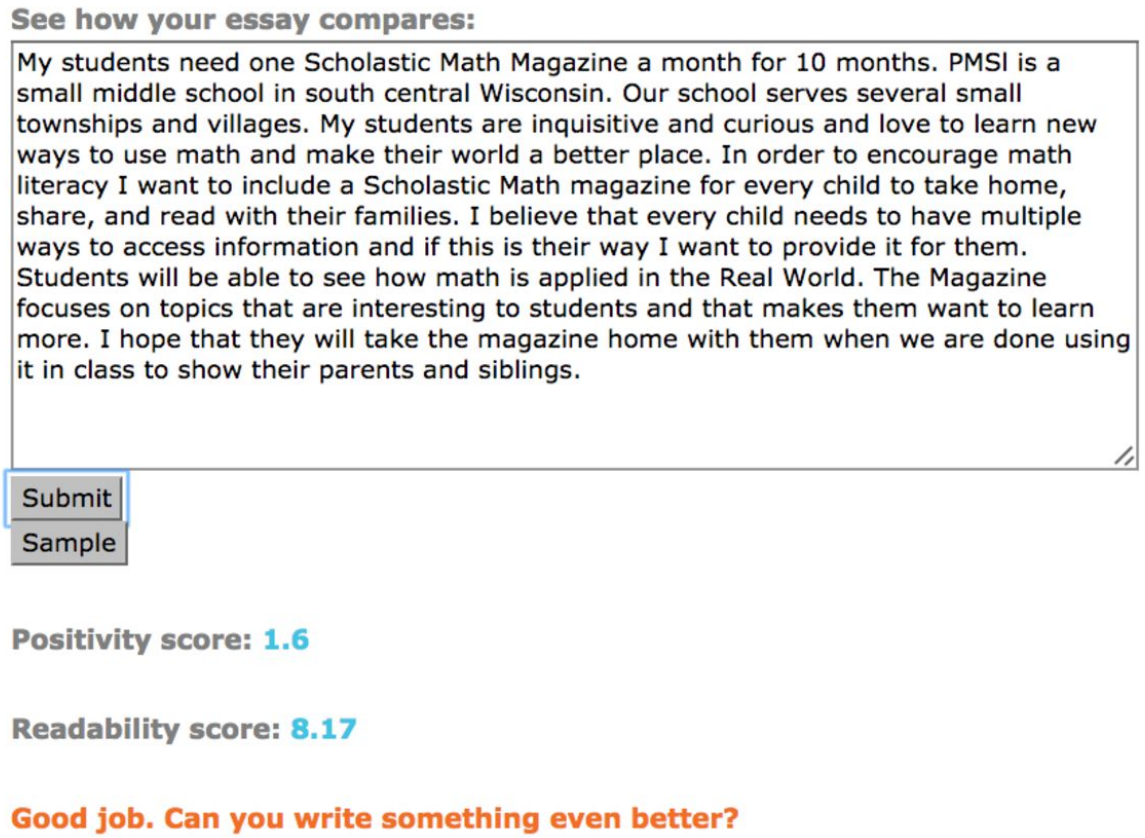
```
preds = bigram_pipeline.predict(text_test)
print 'Bigram RMSE: ', mean_squared_error(stars_test,preds)**0.5
print 'Bigram MAE: ', mean_absolute_error(stars_test,preds)
print 'Bigram R^2: ', r2_score(stars_test,preds)
```

```
Bigram RMSE:  0.649430728128
Bigram MAE:  0.506666155077
Bigram R^2:  0.745359651404
```



Linked to GitHub

WebApp example, DonorsChoose Essay NLP



Linked to GitHub

Introduction

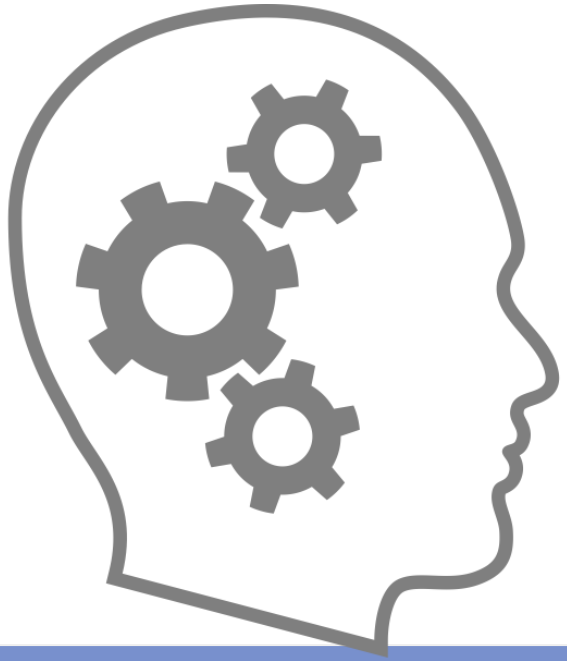
Python 101

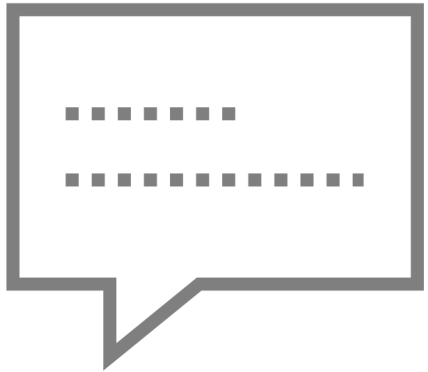
NLP with Python 101

Project Gutenberg examples

USE CASES

USE CASES





USE CASES

- **Pre policy “smart” survey - interlaying free/open response with categorical responses**
- **Social media - monitoring pulse to policy, reaction to new initiatives**
- **Other feedback loops could feed some of the PEAK metrics (i.e. “voice of the customer” survey)**
- **Town Hall / Open Forum - public record analysis**
- **Personalization - allowing testing for PocketGov, 311, other apps**
- **Increased realization of bill collection/payment?**

USE CASES

- **Analyze existing and proposed policies for keywords / topics to create summaries / rules / heuristics**
- **An added level of analysis to complex interventions (helps further inform causality)**
- **Assist in cutting backlogs, creating priority queues (e.g. DC Formapalooza)**
- **Potentially increase efficiencies in certain data/cloud storage**
- **Potentially reduce paperwork**
- **Fraud detection**
- **Churn Analysis**

Additional Resources

- GitHub NLP 101 (including Python 101) Tutorial
- GitHub NLP WebApp
- GitHub Yelp NLP Prediction
- GitHub NLP 102 Tutorial
- General Python Resources:
  - Recommended Python 101 - Google Developers, Learn Python the Hard Way, DataCamp (and others!)
  - Anaconda Data Science Stack: https://docs.continuum.io/anaconda/install/
  - NLTK: http://www.nltk.org/
  - Jupyter Notebooks: http://jupyter.readthedocs.io/en/latest/install.html
  - StackOverflow, GitHub, KDnuggets, DataTau
- Deloitte University, SmartCities (good overview of NLP & SmartCities)