Using **NLP & Django** to build a **movie suggestion site** and **twitterbot**

Vince Salvino
salvino@coderedcorp.com
Djangocon US 2016

Slides Available Online

www.coderedcorp.com/resources

About CIFF

- The Cleveland International Film Festival (CIFF) is a two-week long event featuring hundreds of foreign, independent, and new films making their debut on the silver screen.
- Was held in April 2016. This year marked CIFF's 40th anniversary.
- Very large and important annual event for Cleveland, OH. This year featured over 400 films and shorts.
- I myself am not a film buff, but do appreciate a well told story.
- Shout out to "Morris from America" – a great film I saw this year.
- * I am not affiliated with CIFF.

About the Project

- Our team built a movie recommendation engine for the film fest. This was a non-official/hobby project, so we worked with publicly available data.
- Built a Django project that: scrapes the film data from a website, builds a film similarity index using Django models and natural language processing, and automatically shows suggested films via a website and twitterbot (ciff.coderedcorp.com and @CIFFbot).
- Project was built using public data and open source software.
  - Film data scraped from www.clevelandfilm.org.
  - Natural language processing done with NLTK.
  - Twitter connectivity done using Twitter API (twython) and cron jobs to schedule tweets.
- Project was implemented in 2 days and is 100% Python, excluding use of cron.

About this Talk

1. Put on our search engine hats and scrape data from a website.
2. Make custom Django management commands.
3. Explore a few basic concepts in natural language processing.
4. Explore functionality in the NLTK.
5. Using the Twitter API to make a dumb twitterbot
6. Creating a cron job that invokes a Django command.
7. Implement a simple whoosh/haystack search in Django (time permitting).

First Things First

- Look at the CIFF website.
- Identify what data we need and how to represent it.
- Make models.

```
Movie                    Showtime
```

1. Scraping the Data

`urllib`

- Make a request and fetch the page.

`BeautifulSoup`

- “the browser” – manipulate and parse the HTML doc’s markup.

Code tip #1

2. Django Management Command

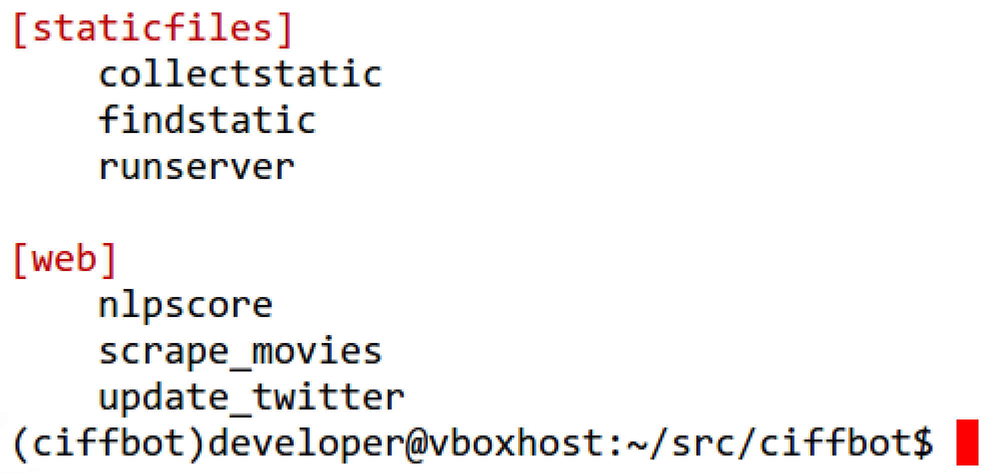
- https://docs.djangoproject.com/en/1.9/howto/custom-management-commands/
- It’s really this easy:

```
from django.core.management.base import BaseCommand
from web.util.crawler import scrape_movies

class Command(BaseCommand):
    help = 'Scrape film data from clevelandfilm.org'
    can_import_settings = True

    def handle(self, *args, **options):
        scrape_movies()
```

2. Django Management Command

Code tip #1

```
python manage.py help
```



```
[staticfiles]
    collectstatic
    findstatic
    runserver

[web]
    nlpscore
    scrape_movies
    update_twitter
(ciffbot)developer@vboxhost:~/src/ciffbot$
```

3. Natural Language Processing

TF-IDF

- “Term Frequency, Inverse Document Frequency”.
- One of the most simple ways to determine document similarity.
- Break down the doc into individual words, throw away the stopwords (common words such as: the, a, an, and, is, etc.), and then looks to see which docs have highest number of words in common.
- Effective, but not very smart.

3. Natural Language Processing

TF-IDF

“I went to the bank to deposit money”.

“I slid down the bank by the lake”.

TF-IDF says these are similar.

But in reality we know that “bank” has completely different meanings in both contexts.

3. Natural Language Processing

Word Sense Disambiguation

- The meaning of the word is determined based on the context, not just the spelling alone.
- First break each document into sentences, and then analyze each word of each sentence.
- Once we have determined the meaning of each word within the context of each sentence, look for lemmas to that meaning.
- Lemma is an abstract term that defines the true meaning of a word before you have spoken or written the word, but have an idea in your head. You can think of lemmas as synonyms.

3. Natural Language Processing

Word Sense Disambiguation

“I went to the bank to deposit money”

BANK: meaning: a financial institution that accepts deposits and channels the money into lending activities.

BANK: lemmas: bank, banking company, financial institution

“I slid down the bank by the lake”

BANK: meaning: sloping land (especially the slope beside a body of water).

BANK: lemmas: slope, curve, side, edge, shore

3. Natural Language Processing

Sentiment Analysis

- Determine “feeling” of the text.
- Typically this is “positive” or “negative”.
- By combining with word sense disambiguation, sentiment analysis can be used to infer additional advanced sentiments
  - A negative sentence from a customer about finance might indicate frustration or confusion.
  - A negative product review might indicate disapproval.
  - A negative comment on politics might indicate harsher feelings such as disgust.

Code tip #2

4. NLTK (and scikit)

TF-IDF

```
import nltk
import regex
from nltk.corpus import stopwords
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.stem.porter import PorterStemmer
from nltk.wsd import lesk
from sklearn.feature_extraction.text import TfidfVectorizer
from web.models import Movie, Comparator
```

```
filmtexts = []
for film in films:
    # change to lowercase and remove punctuation
    filmtexts.append(_clean(_remove_author(film.description)))
# Create TF-IDF metrics
vect = TfidfVectorizer(tokenizer=_tfidf_tokenize)
tfidf = vect.fit_transform(filmtexts)

return (tfidf * tfidf.T).A
```

http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

4. NLTK

Word Sense Disambiguation

Code is a little more complicated…

http://www.nltk.org/howto/wsd.html

4. NLTK

Sentiment Analysis

- Used VADER sentiment analyzer
- Was trained on dataset of 10,000 tweets and 10,000 movie reviews from rotten tomatoes.
- Each one of these tweets and reviews was labeled by a human as being "positive" or "negative".
- Rates on a scale from -1.0 to +1.0 representing negativity or positivity.
- http://www.nltk.org/howto/sentiment.html

Code tip #3

4. NLTK

Sentiment Analysis

```
import nltk
import regex
from nltk.corpus import stopwords
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.stem.porter import PorterStemmer
from nltk.wsd import lesk
from sklearn.feature_extraction.text import TfidfVectorizer
from web.models import Movie, Comparator
```

```
results = []
sid = SentimentIntensityAnalyzer()
counter = 0
for film in films:
    results.append(
        sid.polarity_scores(
            _remove_author(film.description).lower())['compound']
    )
    print("Sentiment {0}".format(counter))
    counter += 1
return results
```

4. NLTK

Crunch the numbers and compare results:

- Our crawler pulled in 436 films from the clevelandfilm.org website.
- Every word in every sentence of every film was broken down, analyzed, and then compared to every other film.
- In computer science lingo, this means O(n2-n) comparison, or in human terms: 189,660 different comparisons.
- Use of “Comparator” model to store the comparisons between “Movie” models.

5. Twitter API

- https://apps.twitter.com/
- Create an app.
- Since you are owner of app, the app will have access to your account.
- We will not be building in an OAuth / grant permission process
  - For that, check out python-social-auth.
  - We will just use our API keys directly.
- We will use Twython to access make Twitter API calls.
  - http://twython.readthedocs.io/en/stable/usage/basic_usage.html#updating-status

5. Twitter API

```
from django.conf import settings
from django.utils import timezone
from web.models import Showtime, Comparator
from twython import Twython
from datetime import timedelta
```

Code tip #4

```
now = timezone.now().replace(second=0, microsecond=0)
minutes_future = (now + timedelta(minutes=5)).replace(second=0, microsecond=0)

now_starting_showtimes = Showtime.objects.filter(start_time__range=(now, minutes_future),
                                                  has_now_playing_tweet=False)

if now_starting_showtimes != []:
    twitter = Twython(
            settings.TWITTER_CONSUMER_KEY,
            settings.TWITTER_CONSUMER_SECRET,
            settings.TWITTER_ACCESS_TOKEN,
            settings.TWITTER_ACCESS_SECRET
            )
    for showtime in now_starting_showtimes:
        status = '"{0}" now playing at #CIFF40. See my analysis https://ciff.coderedcorp.com{1}'\
                 .format(str(showtime.movie), showtime.movie.get_absolute_url())
        twitter.update_status(status=status)
        showtime.has_now_playing_tweet = True
        showtime.save()
        print(showtime)
```

5. Twitter API (simple example)

Code tip #4

```
def djangocon_tweet():
    twitter = Twython(
        settings.TWITTER_CONSUMER_KEY,
        settings.TWITTER_CONSUMER_SECRET,
        settings.TWITTER_ACCESS_TOKEN,
        settings.TWITTER_ACCESS_SECRET
    )
    status = "Check out the talk about me at #djangocon https://2016.djangocon.us/schedule/presentation/30/"
    twitter.update_status(status=status)
    print("Tweeted!")
```

Code tip #5

6. Schedule Tweets with Cron

- It's as easy as creating another management command!
- Then call the management command from cron.
- Make sure cron command runs in appropriate virtualenv.

```
*/5 * * * *

username

cd /var/www/ciffbot &&
   /virtualenvs/ciffbot/bin/python3
      /var/www/ciffbot/manage.py update_twitter >
   /var/log/ciffbot/ciffbot.log 2>&1
```

7. Searching the Models

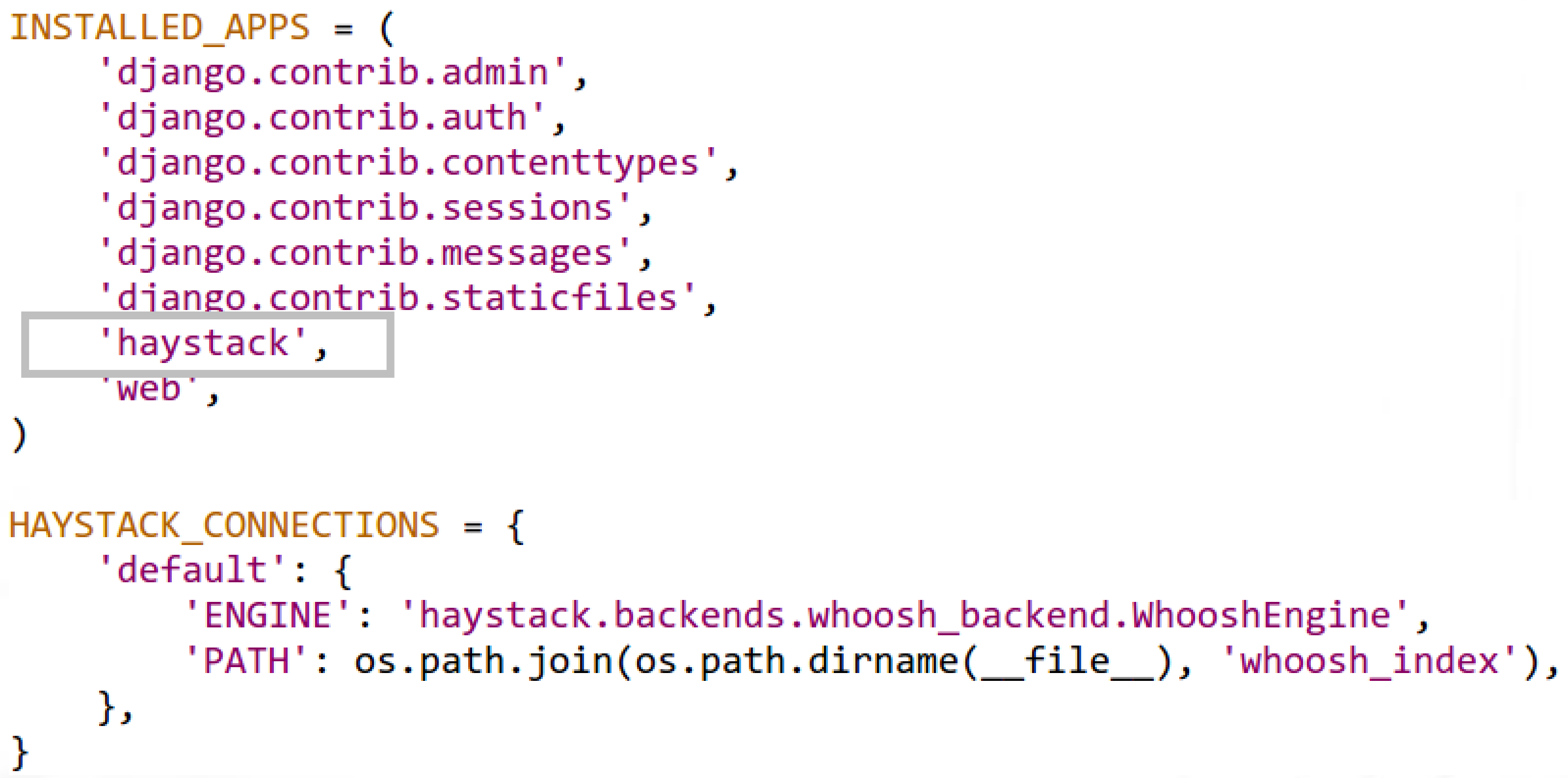
Haystack/Whoosh

- One of the simplest ways to search Django models
- Haystack acts as a search wrapper or API
- Whoosh acts as the search backend.
- Think haystack = Django ORM, whoosh = SQLite
- http://haystacksearch.org/

Code tip #6

7. Searching the Models

Settings



```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'haystack',
    'web',
)

HAYSTACK_CONNECTIONS = {
    'default': {
        'ENGINE': 'haystack.backends.whoosh_backend.WhooshEngine',
        'PATH': os.path.join(os.path.dirname(__file__), 'whoosh_index'),
    },
}
```

7. Searching the Models

Code tip #6

search_indexes.py

```
from haystack import indexes
from web.models import Movie

class MovieIndex(indexes.SearchIndex, indexes.Indexable):
    text = indexes.CharField(document=True, use_template=True)
    name = indexes.CharField(model_attr='name')
    description = indexes.CharField(model_attr='description')

    def get_model(self):
        return Movie

    def index_queryset(self, using=None):
        return self.get_model().objects.all()
```

Code tip #6

7. Searching the Models

views.py

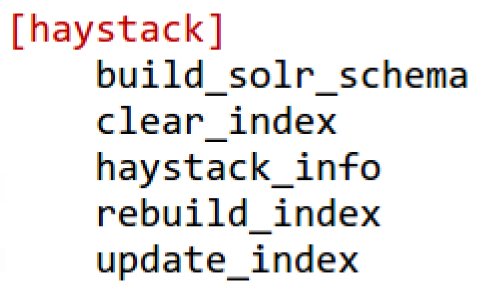
```
from django.shortcuts import render
from haystack.forms import SearchForm
from haystack.query import SearchQuerySet

def search(request):
    search_term = request.GET['q']
    movies = []
    if search_term != '':
        results = SearchQuerySet().auto_query(search_term)
        for result in results:
          movies.append(Movie.objects.get(id=result.pk))
    else:
        movies = []

    return render(request, 'web/search.html', {'movies': movies})
```

Code tip #6

7. Searching the Models

- Now just build the index.

```
python manage.py update_index
```



```
[haystack]
    build_solr_schema
    clear_index
    haystack_info
    rebuild_index
    update_index
```

Thank You!

salvino@coderedcorp.com

Twitter: @vincesalvino