

# Word Embeddings for Fun and Profit



<http://rare-technologies.com/>

Lev Konstantinovskiy

Community Manager at Gensim

@teagermylk

**Part 1. Word2vec.**

**Part 2. Document classification.**

***Notebook at: <http://small.cat/dry>***

## About



Graduate school drop-out in Algebraic Geometry

Worked in Trading IT

Graduate of Galvanize Data Science Bootcamp in San Francisco

Community manager and consultant at RaRe Technologies.

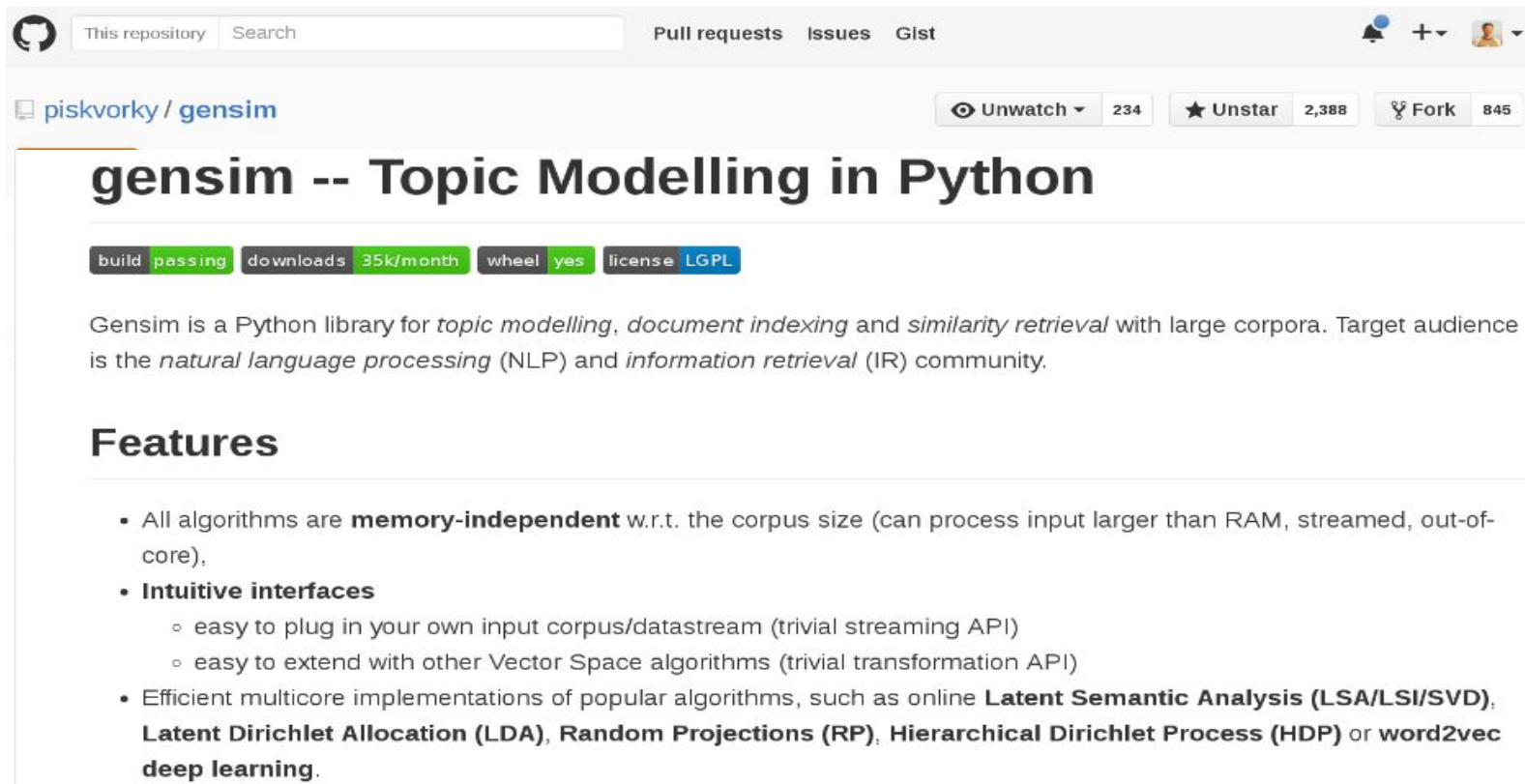
Lev Konstantinovskiy

NLP consulting and open-source development of Natural Language Processing packages in Python.

@teagermylk

<https://github.com/tmylk>

# Word2vec and Topic Modelling in Python



The screenshot shows the GitHub repository page for `piskvorky/gensim`. At the top, there's a navigation bar with the GitHub logo, a search bar, and links for Pull requests, Issues, and Gist. Below this, the repository name `piskvorky/gensim` is displayed, along with statistics: 234 watchers, 2,388 stars, and 845 forks. The repository title is `gensim -- Topic Modelling in Python`. Below the title, there are badges for build status (passing), downloads (35k/month), wheel support (yes), and license (LGPL). The description states: "Gensim is a Python library for *topic modelling*, *document indexing* and *similarity retrieval* with large corpora. Target audience is the *natural language processing* (NLP) and *information retrieval* (IR) community." The **Features** section lists: All algorithms are **memory-independent** w.r.t. the corpus size; **Intuitive interfaces** (easy to plug in and extend); and Efficient multicore implementations of popular algorithms like **Latent Semantic Analysis (LSA/LSI/SVD)**, **Latent Dirichlet Allocation (LDA)**, **Random Projections (RP)**, **Hierarchical Dirichlet Process (HDP)**, and **word2vec deep learning**.

This repository Search

Pull requests Issues Gist

piskvorky / gensim

Unwatch 234 Unstar 2,388 Fork 845

## gensim -- Topic Modelling in Python

build passing downloads 35k/month wheel yes license LGPL

Gensim is a Python library for *topic modelling*, *document indexing* and *similarity retrieval* with large corpora. Target audience is the *natural language processing* (NLP) and *information retrieval* (IR) community.

## Features

- All algorithms are **memory-independent** w.r.t. the corpus size (can process input larger than RAM, streamed, out-of-core),
- **Intuitive interfaces**
  - easy to plug in your own input corpus/datastream (trivial streaming API)
  - easy to extend with other Vector Space algorithms (trivial transformation API)
- Efficient multicore implementations of popular algorithms, such as online **Latent Semantic Analysis (LSA/LSI/SVD)**, **Latent Dirichlet Allocation (LDA)**, **Random Projections (RP)**, **Hierarchical Dirichlet Process (HDP)** or **word2vec deep learning**.

# Why would I care? Because it is magic!

## Translating restaurants via concepts



**Harris'**  
Steakhouse in  
Downtown area

$$\vec{v}(\text{Harris}') + \vec{v}(\text{patio})$$



**Patio at  
Las  
Sendas**  
Steakhouse  
with amazing  
patio

$$\vec{v}(\text{Harris}') + \vec{v}(\text{jazz})$$



**Broadway  
Jazz Club**  
Steakhouse  
with live jazz

$$\vec{v}(\text{Harris}') + \vec{v}(\text{scenic})$$



**Celestial  
Steakhous  
e**  
Steakhouse  
with a view



## Word embeddings can be used for:

- automated text tagging (*this talk*)
- recommendation engines
- synonyms and search query expansion
- machine translation
- plain feature engineering

**Notebook at: <http://small.cat/dry>**

# The business problem to be solved in Part 2

You run a movie studio.

Every day you receive thousands of proposals for movies to make.

Need to send them to the right department for consideration!

One department per genre.

Need to ***classify plots by genre.***

# What is a word embedding?

‘Word embedding’ = ‘word vectors’ = ‘distributed representations’

It is a **dense** representation of words in a **low-dimensional vector space**.

## One-hot representation:

king = [1 0 0 0.. 0 0 0 0 0]

queen = [0 1 0 0 0 0 0 0 0]

book = [0 0 1 0 0 0 0 0 0]

## Distributed representation:

king = [0.9457, 0.5774, 0.2224]



# How to come up with an embedding?

word2vec relies on the “Distributional hypothesis”:

*“You shall know a word by the company it keeps”*

-J. R. Firth 1957

government debt problems turning into banking crises as has happened in  
saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

# Usual procedure

1. Initialise random vectors
2. Pick an objective function.
3. Do gradient descent.

For the theory, take Richard Sochers's class

## Details of Word2Vec

- Predict surrounding words in a window of length  $m$  of every word.
- Objective function: Maximize the log probability of any context word given the current center word:

- $$J(\theta) = \frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} | w_t)$$

- Where  $\theta$  represents all variables we optimize

# word2vec algorithm

*“The fox jumped **over** the lazy dog”*

Maximize the likelihood of seeing the *context words* given the word **over**.

$P(\text{the}|\text{over})$

$P(\text{fox}|\text{over})$

$P(\text{jumped}|\text{over})$

$P(\text{the}|\text{over})$

$P(\text{lazy}|\text{over})$

$P(\text{dog}|\text{over})$

# Probability should depend on the word vectors.

$P(\text{fox}|\text{over})$



$P(v_{\text{fox}} | v_{\text{over}})$

*A twist: two vectors for every word*

Should depend on whether it's the input or the output.

$$P(v_{\text{OUT}} | v_{\text{IN}})$$

*“The fox jumped **over** the lazy dog”*



$v_{\text{IN}}$

*Twist:* two vectors for every word

Should depend on whether it's the input or the output.

$$P(\mathbf{v}_{\text{OUT}} | \mathbf{v}_{\text{IN}}) = P(\mathbf{v}_{\text{THE}} | \mathbf{v}_{\text{OVER}})$$

*“The fox jumped **over** the lazy dog”*

  
 $\mathbf{v}_{\text{OUT}}$

  
 $\mathbf{v}_{\text{IN}}$

*Twist:* two vectors for every word

Should depend on whether it's the input or the output.

$$P(v_{\text{OUT}} | v_{\text{IN}})$$

*“The fox jumped **over** the lazy dog”*

  
 $v_{\text{OUT}}$   
 $v_{\text{IN}}$

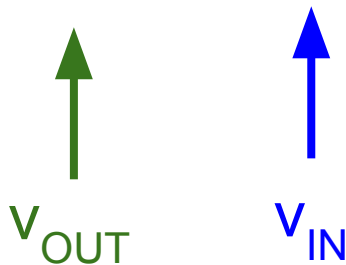


*Twist:* two vectors for every word

Should depend on whether it's the input or the output.

$$P(v_{\text{OUT}} | v_{\text{IN}})$$

*“The fox jumped **over** the lazy dog”*

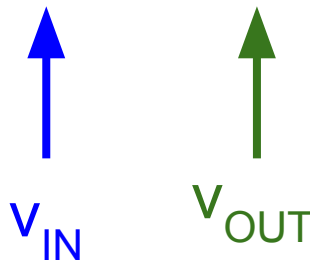


*Twist:* two vectors for every word

Should depend on whether it's the input or the output.

$$P(v_{\text{OUT}} | v_{\text{IN}})$$

*“The fox jumped **over** the lazy dog”*



*Twist:* two vectors for every word

Should depend on whether it's the input or the output.

$$P(v_{\text{OUT}} | v_{\text{IN}})$$

*“The fox jumped **over** the lazy dog”*

  
 $v_{\text{IN}}$

  
 $v_{\text{OUT}}$

*Twist:* two vectors for every word

Should depend on whether it's the input or the output.

$$P(v_{\text{OUT}} | v_{\text{IN}})$$

*“The fox jumped **over** the lazy dog”*



$v_{\text{IN}}$



$v_{\text{OUT}}$

*Twist:* two vectors for every word

Should depend on whether it's the input or the output.

$$P(v_{\text{OUT}} | v_{\text{IN}})$$

*“The fox jumped over **the** lazy dog”*

  
 $v_{\text{OUT}}$

  
 $v_{\text{IN}}$

*Twist:* two vectors for every word

Should depend on whether it's the input or the output.

$$P(v_{\text{OUT}} | v_{\text{IN}})$$

*“The fox jumped over **the** lazy dog”*

  
 $v_{\text{OUT}}$

  
 $v_{\text{IN}}$

# How to define $P(v_{\text{OUT}}|v_{\text{IN}})$ ? First, define similarity.

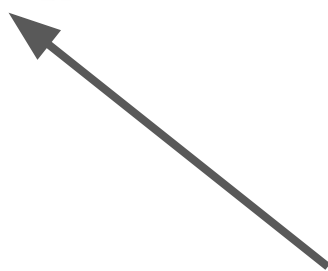
How similar are two vectors?

*Just dot product for unit length vectors*

$$V_{\text{OUT}}^* V_{\text{IN}}$$

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

Get a probability in  $[0,1]$  out of similarity in  $[-1, 1]$

$$\textit{softmax} = \frac{\exp(v_{in} \cdot v_{out})}{\sum_{k \in V} \exp(v_{in} \cdot v_k)} = P(v_{out}|v_{in})$$


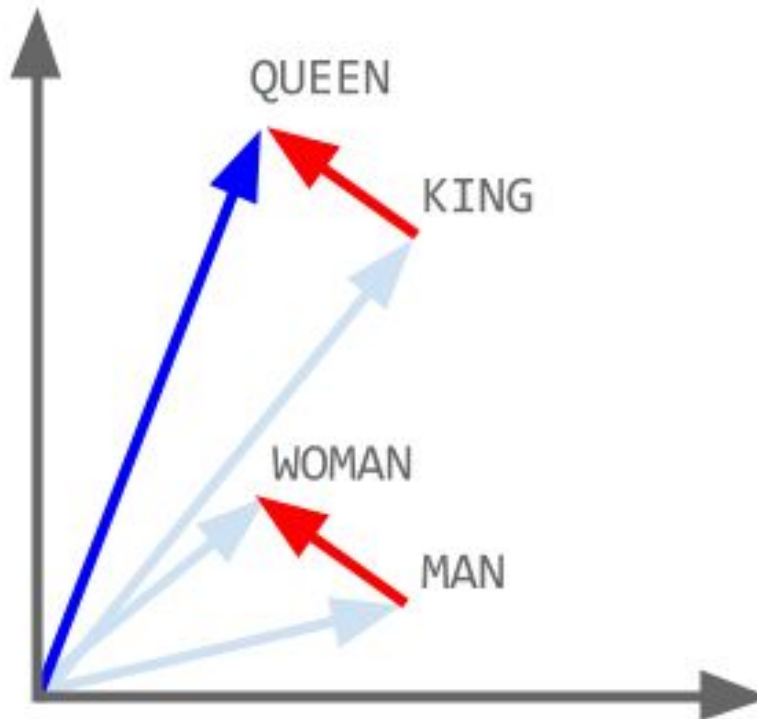
Normalization term over all out words



Word2vec is great!

Vector arithmetic

So  $\text{king} - \text{man} + \text{woman} = \text{queen!}$



Slide from @chrisemoodly <http://www.slideshare.net/ChristopherMoody3/word2vec-lda-and-introducing-a-new-hybrid-algorithm-lda2vec>

# Consistent directions

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

# Word2vec is big victory of **unsupervised** learning

Google ran word2vec on 100billion of unlabelled documents.

Then shared their trained model.

Thanks to Google for cutting our training time to zero!. :)

# Questions?

**Repo at: <http://small.cat/dry>**

## Part 2. Document Classification

**Repo at: <http://small.cat/dry>**

# What is the genre of this plot?

*In a future world devastated by disease, a convict is sent back in time to gather information about the man-made virus that wiped out most of the human population on the planet.*

# Of course it is SCI-FI

*In a future world devastated by disease, a convict is sent back in time to gather information about the man-made virus that wiped out most of the human population on the planet.*



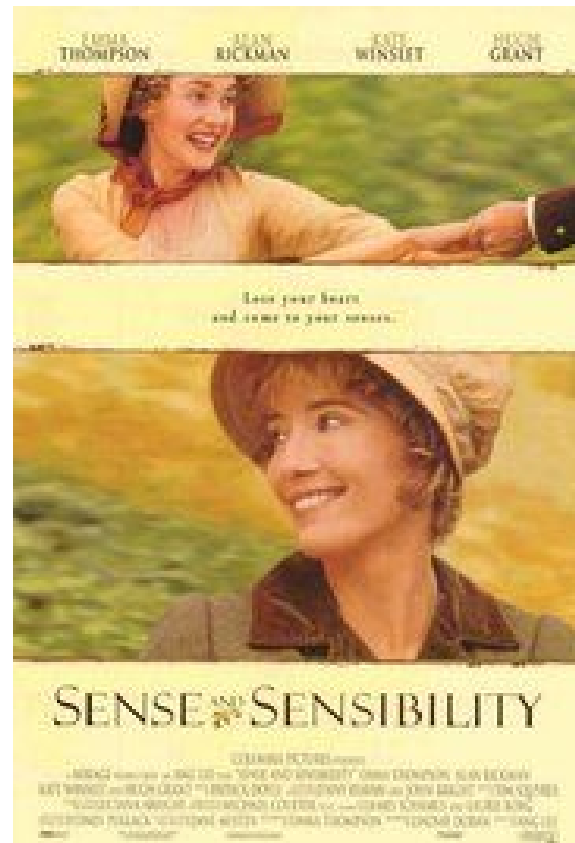
# What is the genre of this one?

*Mrs. Dashwood and her three daughters are left in straitened circumstances. When Elinor forms an attachment for the wealthy Edward Ferrars, his family disapproves and separates them. And though Mrs. Jennings tries to match the worthy (and rich) Colonel Brandon to her, Marianne finds the dashing and fiery John Willoughby more to her taste.*



# ROMANCE

When Mr. Dashwood dies, he must leave the bulk of his estate to the son by his first marriage, which leaves his second wife and their three daughters (Elinor, Marianne, and Margaret) in straitened circumstances. They are taken in by a kindly cousin, but their lack of fortune affects the marriageability of both practical Elinor and romantic Marianne. When Elinor forms an attachment for the wealthy Edward Ferrars, his family disapproves and separates them. And though Mrs. Jennings tries to match the worthy (and rich) Colonel Brandon to her, Marianne finds the dashing and fiery John Willoughby more to her taste. Both relationships are sorely tried.



***The text is very different so should be some signal there***

## sci-fi

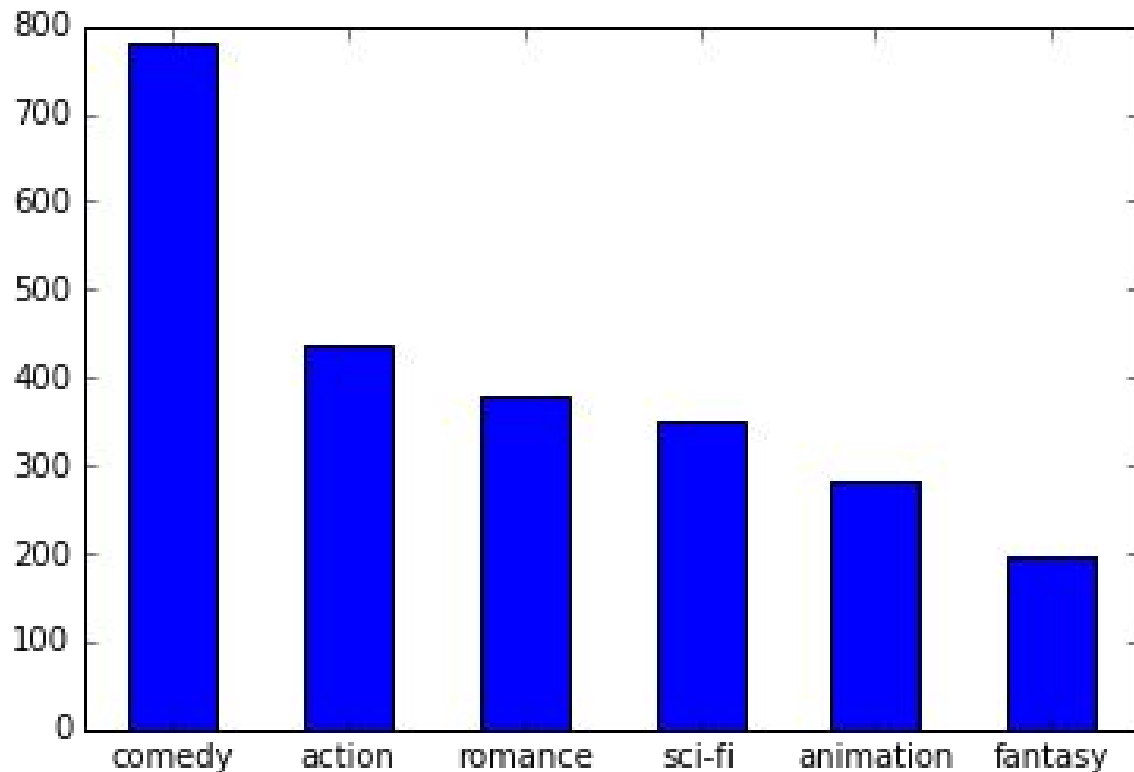
*In a future world devastated by disease, a convict is sent back in time to gather information about the man-made virus that wiped out most of the human population on the planet.*

## romance

*Mrs. Dashwood and her three daughters are left in straitened circumstances. When Elinor forms an attachment for the wealthy Edward Ferrars, his family disapproves and separates them. And though Mrs. Jennings tries to match the worthy (and rich) Colonel Brandon to her...*

**Corpus:** 2k movie plots. 170k words.

## Unbalanced classes



**Disclaimer: First run, “out of the box” performance. No tuning.**

# Simple baseline: 46%

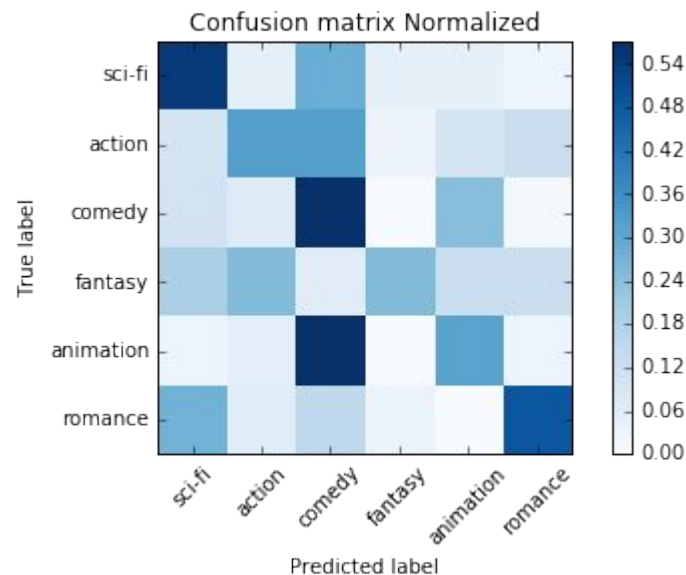
Embedding	Classifier	Accuracy	Train time,s	Predict time, s
Baseline tf-idf bag of words	Logistic	46	2	1

TF-IDF just counting words in a document, adjusting for doc length, word frequency and word-doc frequency

```
TfidfVectorizer(  
    min_df=2,  
    tokenizer=nlk.word_tokenize,  
    preprocessor=None, stop_words='english')
```

*Counting words, char n-grams*

*are similar: 42, 44.*



Let's look at more advanced techniques

First, let's load Google's Word2vec model

# Word2vec to... A Document Classifier

We need some features to power our favourite classifier (logistic regression/KNN)

We have vectors for words but need vectors for documents.

*How to create a **document** classifier out of a set of **word vectors**?*

*For KNN, how similar is one **sequence** of words to another **sequence** of words?*

# Averaging word vectors aka 'Naive document vector'

Just add word vectors together!

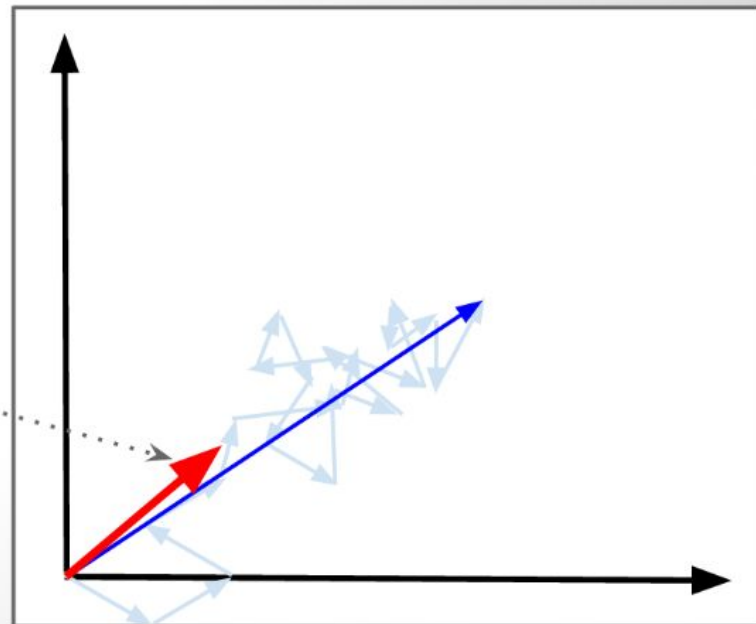
All words in a book

**'A tale of two cities'**

Should add up to

**'class-struggle'**

*Class-struggle*





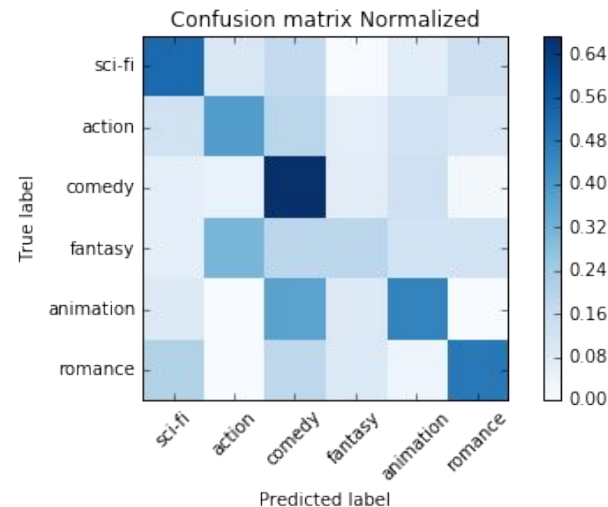
**Disclaimer: First run, “out of the box” performance. No tuning.**

## Averaging word vectors: 52 % accuracy

Embedding	Classifier	Accuracy	Train time,s	Predict time 250 docs, s
Averaging word vectors from pre-trained Google News	Logistic	52	2 (thanks to Google!)	1

```
# loading 1.5 GB archive into memory
wv = Word2Vec.load_word2vec_format(
    "GoogleNews-vectors-negative300.bin.gz")
X_train_naive_dv =
    naived2v_conversion_np(wv,flat_list_train)

logreg = linear_model.LogisticRegression(n_jobs=-1, C=1e5)
logreg = logreg.fit(X_train_naive_dv, train_data['tag'])
```



Why did summarisation not work?

# Introducing **Doc2vec**

Tag is ‘a word that is in every context in the doc’

$$P(v_{\text{OUT}} | v_{\text{IN}}, \text{COMEDY}) = P(v_{\text{FOX}} | v_{\text{OVER}}, v_{\text{COMEDY}})$$

*“The fox jumped **over** the lazy dog. (COMEDY)”*

  
 $v_{\text{OUT}}$

  
 $v_{\text{IN}}$

# Introducing **Doc2vec**

Tag is ‘a word that is in every context in the doc’

$$P(v_{\text{OUT}} | v_{\text{IN}}, \text{COMEDY}) = P(v_{\text{JUMPED}} | v_{\text{OVER}}, v_{\text{COMEDY}})$$

*“The fox jumped **over** the lazy dog. (COMEDY)”*



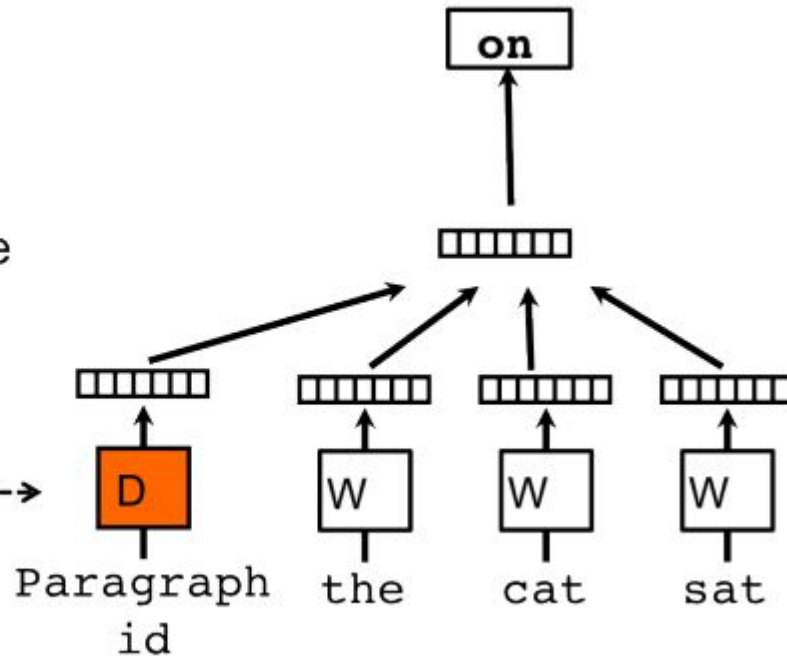
# Doc2vec DM

Tag is 'a word that is in every context in the doc'

Classifier

Average/Concatenate

Paragraph Matrix----->



Let's run some code

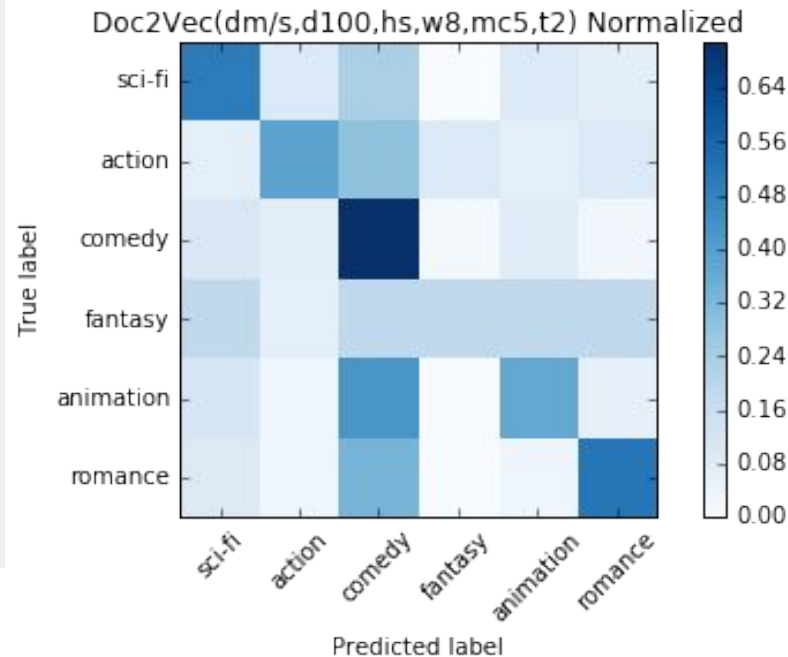
**Disclaimer: First run, “out of the box” performance. No tuning.**

Doc2vec: 52 %

Embedding	Classifier	Accuracy	Train time,s	Predict time, s
Doc2vec	Logistic	52	25	1

```
# simple gensim doc2vec api
model = Doc2Vec(
    trainsent, workers=2, size=100, iter=20,
    dm=1)

train_targets, train_regressors = zip(*[
    (doc.tags[0],
     model.infer_vector(doc.words, steps=20))
    for doc in trainsent])
```



# Closest words to sci-fi

```
model.most_similar([mdm_alt.docvecs['sci-fi']])  
[[('alien', 0.4514704942703247),  
 ('express', 0.4008052945137024),  
 ('space', 0.40043187141418457),  
 ('planet', 0.3805035352706909),  
 ('ant', 0.37011784315109253),  
 ('ferocious', 0.36217403411865234),  
 ('ship', 0.35579410195350647),  
 ('hole', 0.3422626256942749)  
]
```



# Closest words romance

```
model.most_similar([mdm_alt.docvecs['romance']])  
  
[('say', 0.38082122802734375),  
 ('skill', 0.3159002363681793),  
 ('leads', 0.3063559830188751),  
 ('local', 0.3018215596675873),  
 ('millionaire', 0.2863730788230896),  
 ('located', 0.28458985686302185),  
 ('hood', 0.2830425798892975),  
 ('heir', 0.2802196145057678),  
 ('died', 0.27215155959129333),  
 ('indians', 0.26776593923568726)]
```

# Closest genres to romance

```
model.docvecs.most_similar('romance')
```

```
[('fantasy', -0.09007323533296585),  
 ('sci-fi', -0.0983937606215477),  
 ('animation', -0.13281254470348358),  
 ('comedy', -0.1537310779094696),  
 ('action', -0.16746415197849274)]
```

# Bayesian Inversion

1. Train a word2vec model only on comedy plots.
2. Repeat for each genre. Get 6 models.
3. Take a plot and see which model fits it best

# Bayesian Inversion

Which class does this plot fit in?

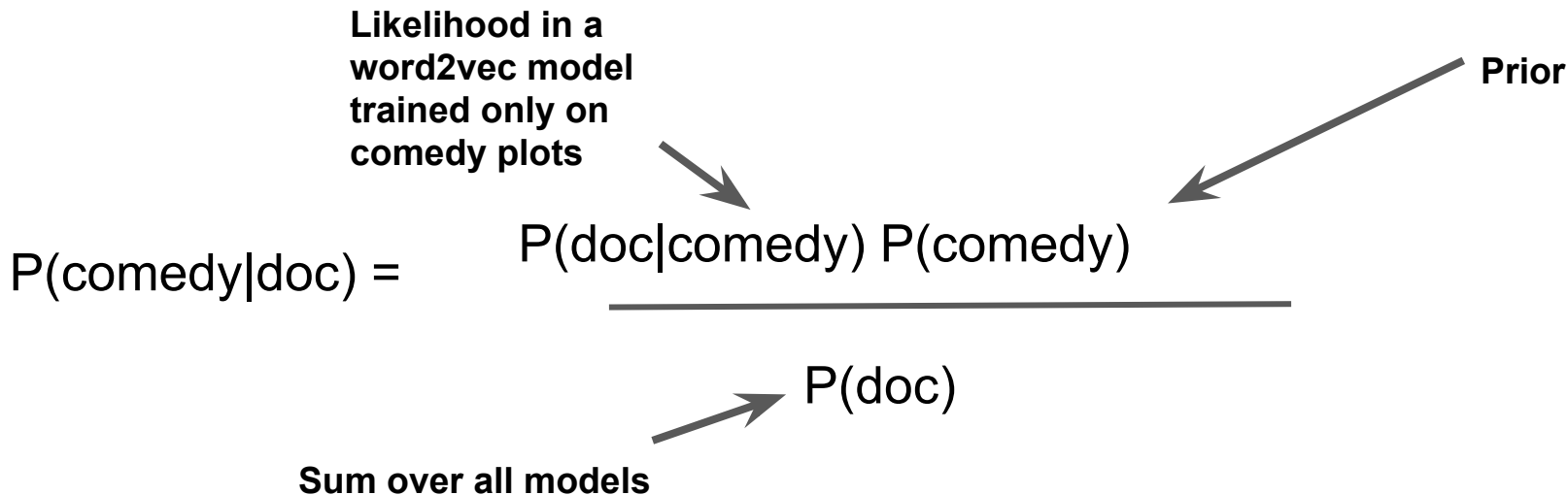
## *Bayes Rule*

Likelihood in a word2vec model trained only on comedy plots

Prior

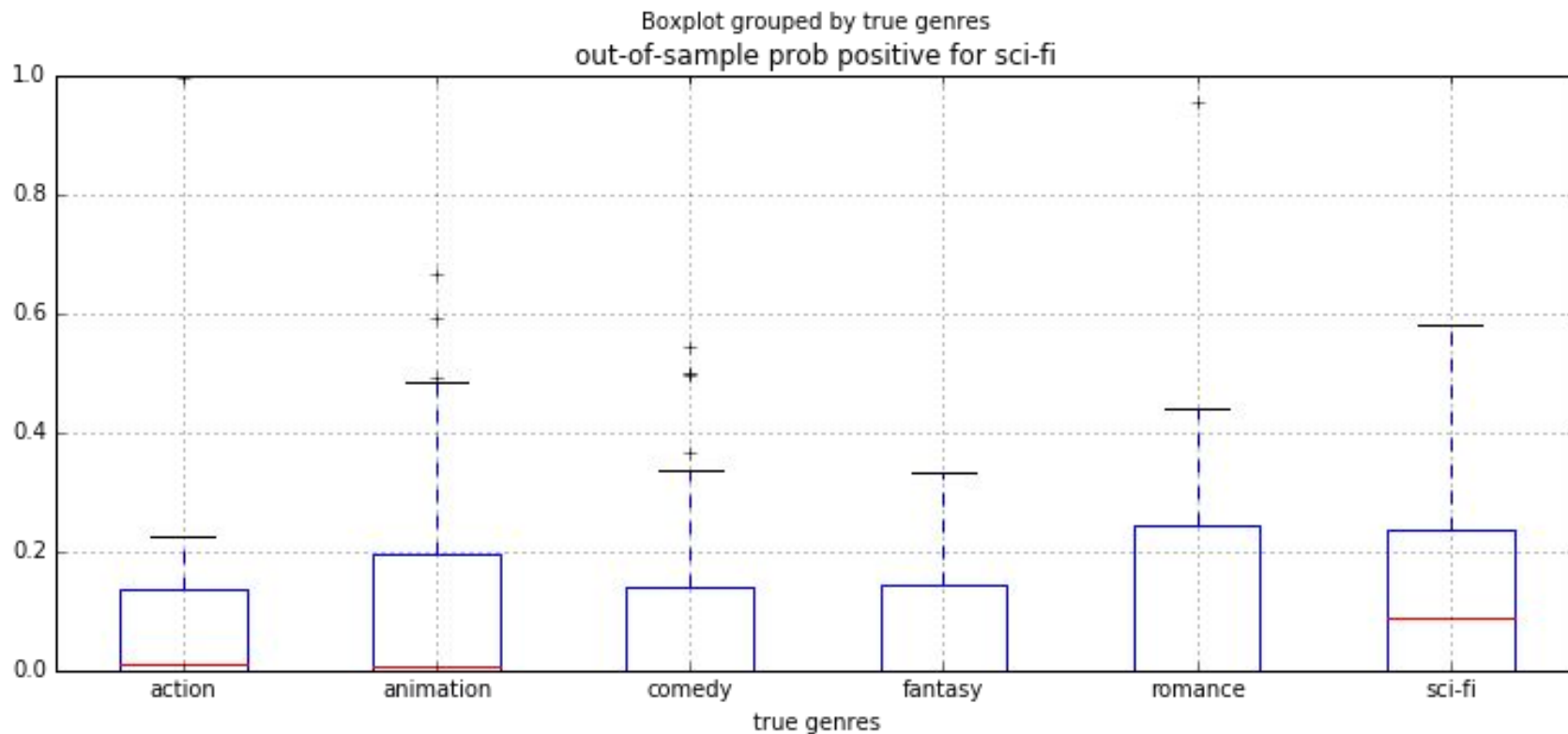
$$P(\text{comedy}|\text{doc}) = \frac{P(\text{doc}|\text{comedy}) P(\text{comedy})}{\text{Sum over all models } P(\text{doc})}$$

Sum over all models



Let's run some code

# Probabilities assigned by sci-fi model to plots of each genre



**Disclaimer: First run, “out of the box” performance. No tuning.**

# Bayesian Inversion: only 30%.

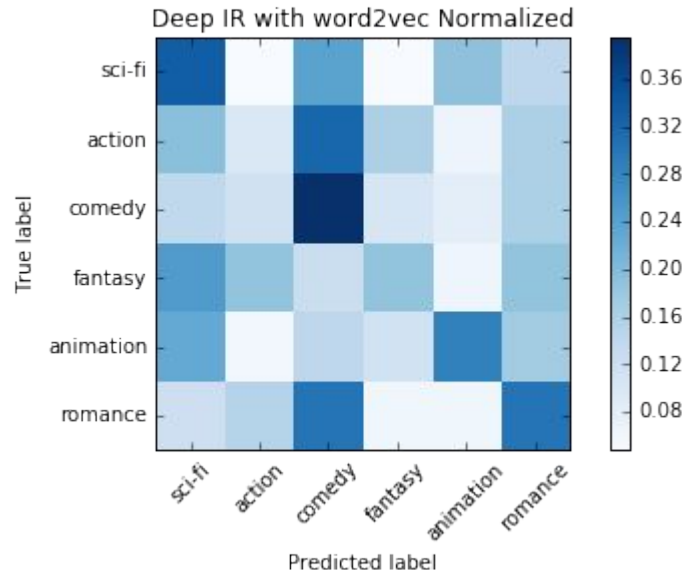
Embedding	Classifier	Accuracy	Train time,s	Predict time, s
Bayesian Inversion	Max likelihood	30	120	1

***Usually quite good but corpus too small***

***Train:*** six different word2vec models. Use sentences from plots with tag X.

***Predict:*** Average all sentences in a review to get a likelihood.

*1000 iterations in word2vec training*



# Word Mover's distance

## From Word Embeddings To Document Distances

**Matt J. Kusner**

**Yu Sun**

**Nicholas I. Kolkin**

**Kilian Q. Weinberger**

Washington University in St. Louis, 1 Brookings Dr., St. Louis, MO 63130

MKUSNER@WUSTL.EDU

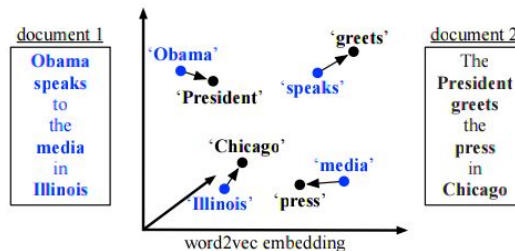
YUSUN@WUSTL.EDU

N.KOLKIN@WUSTL.EDU

KILIAN@WUSTL.EDU

### Abstract

We present the Word Mover's Distance (WMD), a novel distance function between text documents. Our work is based on recent results in word embeddings that learn semantically meaningful representations for words from local co-occurrences in sentences. The WMD distance measures the dissimilarity between two text doc-



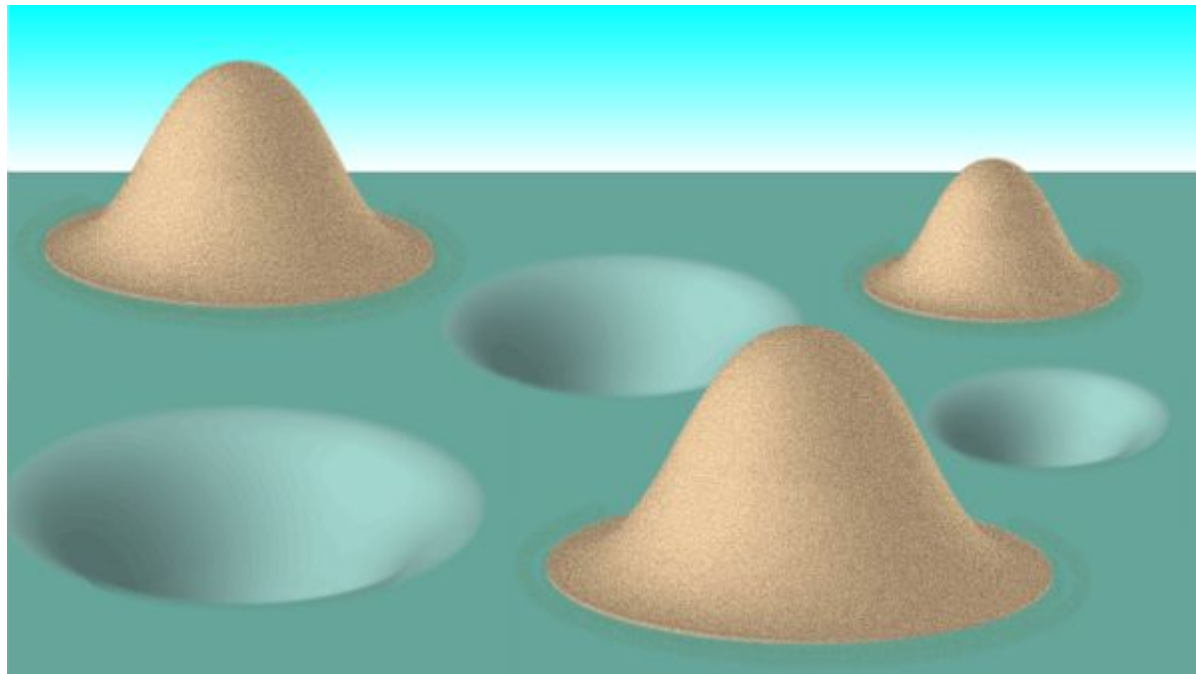
<http://jmlr.org/proceedings/papers/v37/kusnerb15.pdf>

<https://github.com/mkusner/wmd>



# Earth Mover's Distance

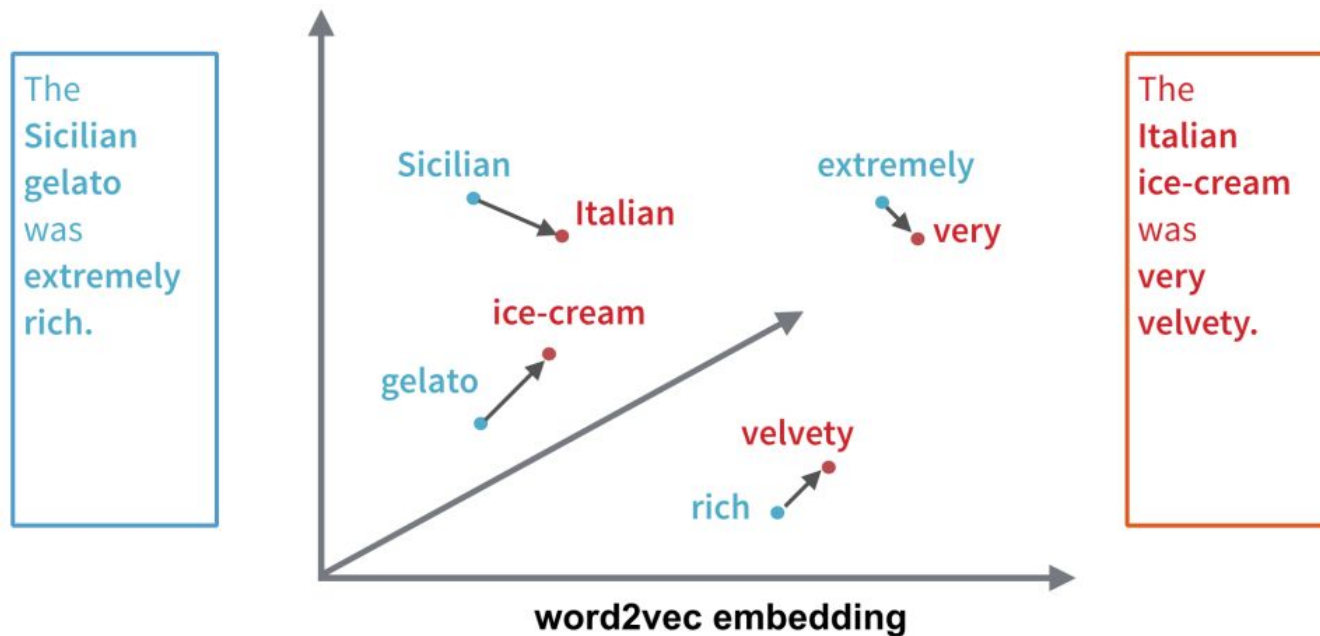
How do you best move piles of sand to fill up holes of the same total volume?



Stated by Monge in 1781. Solved by Kantorovich in

[Image: APS/Alan Stonebraker]

# Word Mover's distance



<http://tech.opentable.com/2015/08/11/navigating-themes-in-restaurant-reviews-with-word-movers-distance/>

# Distance in gensim

```
from gensim.models import Word2Vec

sentence_gelato = 'The Sicilian gelato was extremely rich'
sentence_icecream = 'The Italian ice-cream was very velvety'
distance = model.wmdistance(sentence_gelato, sentence_icecream) # 1.0175

sentence_orange = 'Oranges are my favorite fruit'
model.wmdistancef(sentence_gelato, sentence_orange) # 1.3363
```

**Disclaimer: First run, “out of the box” performance. No tuning.**

# Word Mover's Distance: 42 %

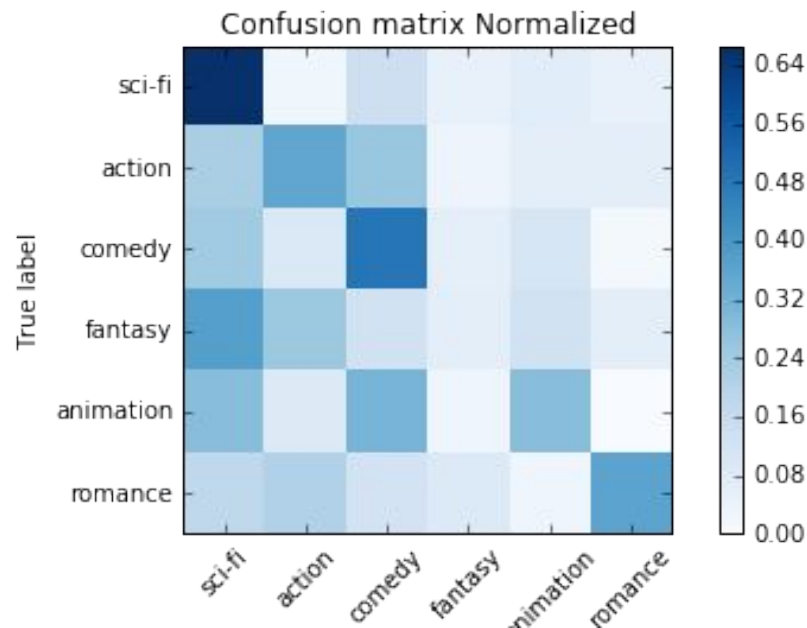
Embedding	Classifier	Accuracy	Train time,s	Predict time, s
Word Movers Distance on Google News	KNN	42	1 (thanks to Google!)	1800

```
WordMoversKNN (
```

```
    n_neighbors=1, W_embed=vv.
```

```
    syn0norm,
```

```
    n_jobs=7)
```



Credit: sklearn api by @vene

<http://vene.ro/blog/word-movers-distance-in-python.html>

# Conclusion

# Rough, first run, “out of the box” performance. No tuning.

Embedding	Classifier	Accuracy	Train time,s	Predict time on 250 docs, s
Baseline tf-idf bag of words	Logistic	47	2	1
Averaging word vectors from pre-trained Google News	Logistic	<b>52</b>	2 (thanks to Google!)	1
Doc2vec	Logistic	<b>52</b>	25	1
Bayesian Inversion	Max likelihood	30	120	1
Word Movers Distance on Google News	KNN	42	<b>1</b> (thanks to Google!)	1800

*No neural network magic out of the box:(*

Simple baselines are not much worse than fancy methods.

# Which model is easiest to tune and debug?

*“What caused this error?”*

*“What is wrong with comedy genre?”*

*“What can I do to fix this class of errors?”*



# Rough, first run, “out of the box” performance. No tuning.

Embedding	Classifier	Accuracy	Train time,s	Predict time, s	Debug/tune
Baseline tf-idf bag of words	Logistic	47	2	1	<b>Easy</b>
Averaging word vectors from pre-trained Google News	Logistic	52	2 (thanks to Google!)	1	Hard
Doc2vec	Logistic	52	25	1	Hard
Bayesian Inversion	Max likelihood	30	120	1	Hard
Word Movers Distance on Google News	KNN	42	1 (thanks to Google!)	1800	Hard

Easiest to understand

Easiest to debug

***TF-IDF Bag of words, of course!***

# There is a place for word embeddings

- Small improvement matters (maybe you want your paper accepted)
- Lots of data
- Extra features to existing ensembles
- Need directionality (analogies)

# Thanks!

***Come and learn about word2vec with us!***

Lev Konstantinovskiy

[github.com/tmylk](https://github.com/tmylk)

@teagermylk

Events coming up:

- Tutorials sprint at PyCon in Portland, Oregon 2-6 June
- Doc classification at PyData New York, 25 May

