W4995 Applied Machine Learning
Fall 2021

Lecture 10

Dr. Vijay Pappu

Announcements

- Midterm grades released
- HW4 will be posted this week
- Next class is the last class
- Project final deliverable on 12/15

In today’s lecture, we will cover…

- Working with text data
- Topic models
- Word & Document embeddings

Working with Text Data

Working With Text Data

- We worked with *structured* data so far:
  - Categorical
    - Ordered
    - Unordered
  - Numerical
    - Integer, Float etc.
- Text data is primarily *unstructured*
  - *Audio, image data ...*

Working With Text Data

"One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me.

The first thing that struck me about Oz was its brutality and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word.

It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements are never far away.

I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance...OZ doesn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become comfortable with what is uncomfortable viewing....thats if you can get in touch with your darker side."

"A wonderful little production.

The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece.

The actors are extremely well chosen- Michael Sheen not only ""has got all the polari"" but he has all the voices down pat too! You can truly see the seamless editing guided by the references to Williams' diary entries, not only is it well worth the watching but it is a terrificly written and performed piece. A masterful production about one of the great master's of comedy and his life.

The realism really comes home with the little things: the fantasy of the guard which, rather than use the traditional 'dream' techniques remains solid then disappears. It plays on our knowledge and our senses, particularly with the scenes concerning Orton and Halliwell and the sets (particularly of their flat with Halliwell's murals decorating every surface) are terribly well done."

Working With Text Data

"One of the other reviewers has mentioned that after watching just 1 **Oz** episode you'll be hooked. They are right, as this is exactly what happened with me.**

The first thing that struck me about Oz was **its brutality and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word.

It is called **OZ** as that is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements are never far away.

I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance...OZ doesn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become comfortable with what is uncomfortable viewing....**thats** if you can get in touch with your darker side."

"A wonderful little production.

The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire piece.

The actors are extremely well chosen- Michael Sheen not only ""has got all the polari"" but he has all the voices down pat too! You can truly see the seamless editing guided by the references to Williams' diary entries, not only is it well worth the watching but it is a **terrificly** written and performed piece. A masterful production about one of the great master's of comedy and his life.

The realism really comes home with the little things: the fantasy of the guard which, rather than use the traditional 'dream' techniques remains solid then disappears. It plays on our knowledge and our senses, particularly with the scenes concerning Orton and Halliwell and the sets (particularly of their flat with Halliwell's murals decorating every surface) are terribly well done."

Working With Text Data

	fullName	country	politicalGroup	id	nationalPoliticalGroup
0	Magdalena ADAMOWICZ	Poland	Group of the European People's Party (Christia...	197490	Independent
1	Asim ADEMOV	Bulgaria	Group of the European People's Party (Christia...	189525	Citizens for European Development of Bulgaria
2	Isabella ADINOLFI	Italy	Group of the European People's Party (Christia...	124831	Forza Italia
3	Matteo ADINOLFI	Italy	Identity and Democracy Group	197826	Lega
4	Alex AGIUS SALIBA	Malta	Group of the Progressive Alliance of Socialist...	197403	Partit Laburista

Working With Text Data - Preprocessing

- Remove special characters
- Bag of Words
- Stop Words, Infrequent Words
- Stemming & Lemmatization
- TF-IDF
- N-grams

Working With Text Data - Libraries

- sklearn
- NLTK
- SpaCy
- Gensim

Working With Text Data - Bag Of Words

- A simple approach to *vectorize* free text
- Every document is *tokenized*.
- A *vocabulary* is built over all documents over all words.
- The length of your feature vector is the size of the vocabulary.
- For each word in the vocabulary, a value is given based on the number of times the word appears in the string.

Working With Text Data - Bag Of Words

sklearn.feature_extraction.text.CountVectorizer

class sklearn.feature_extraction.text.CountVectorizer(*, *input='content'*, *encoding='utf-8'*, *decode_error='strict'*, *strip_accents=None*, *lowercase=True*, *preprocessor=None*, *tokenizer=None*, *stop_words=None*, *token_pattern='(?u)\b\w\w+\b'*, *ngram_range=(1, 1)*, *analyzer='word'*, *max_df=1.0*, *min_df=1*, *max_features=None*, *vocabulary=None*, *binary=False*, *dtype=<class 'numpy.int64'>*) [source]

```
from sklearn.feature_extraction.text import CountVectorizer
got = ["Winter is coming",
       "Chaos isn't a pit. Chaos is a ladder"]
vector = CountVectorizer()
vector.fit(got)
print(vector.get_feature_names())
vector.transform(got)
```

```
['chaos', 'coming', 'is', 'isn', 'ladder', 'pit', 'winter']
<2x7 sparse matrix of type '<class 'numpy.int64'>'
        with 8 stored elements in Compressed Sparse Row format>
```

Working With Text Data - Bag Of Words

```
print(vector.transform(got))
```

```
  (0, 1)        1
  (0, 2)        1
  (0, 6)        1
  (1, 0)        2
  (1, 2)        1
  (1, 3)        1
  (1, 4)        1
  (1, 5)        1
```

```
vector.transform(got).toarray()
```

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

Working With Text Data - Bag Of Words

```
vector_transformed = vector.transform(got)
print(vector.inverse_transform(vector_transformed[0]))
print(vector.inverse_transform(vector_transformed[1]))
```

```
[array(['coming', 'is', 'winter'], dtype='<U6')]
[array(['chaos', 'is', 'isn', 'ladder', 'pit'], dtype='<U6')]
```

Working With Text Data - Bag Of Words

```
imdb_data = pd.read_csv("imdb_dataset.csv")
imdb_data.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

Working With Text Data - Bag Of Words

```
imdb_data.sentiment.value_counts()
```

```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

Working With Text Data - Bag Of Words

```
reviews = list(imdb_data.review)
reviews = [review.replace("<br />", " ")
           for review in reviews]
sentiment = imdb_data.sentiment
reviews[1]
```

```
'A wonderful little production.   The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, an
d sometimes discomforting, sense of realism to the entire piece.   The actors are extremely well chosen- Michael Sheen not only
"has got all the polari" but he has all the voices down pat too! You can truly see the seamless editing guided by the references
to Williams\' diary entries, not only is it well worth the watching but it is a terrificly written and performed piece. A master
ful production about one of the great master\'s of comedy and his life.   The realism really comes home with the little things:
the fantasy of the guard which, rather than use the traditional \'dream\' techniques remains solid then disappears. It plays on
our knowledge and our senses, particularly with the scenes concerning Orton and Halliwell and the sets (particularly of their fl
at with Halliwell\'s murals decorating every surface) are terribly well done.'
```

Working With Text Data - Bag Of Words

```
dev_text, test_text, dev_y, test_y = train_test_split(reviews, sentiment, test_size=0.2,
                                                      random_state=42)
print(dev_y.value_counts())
print(test_y.value_counts())
```

```
negative    20039
positive    19961
Name: sentiment, dtype: int64
positive    5039
negative    4961
Name: sentiment, dtype: int64
```

Working With Text Data - Bag Of Words

```
vector = CountVectorizer()
dev_X = vector.fit_transform(dev_text)
test_X = vector.transform(test_text)
dev_X
```

```
<40000x93003 sparse matrix of type '<class 'numpy.longlong'>'
        with 5455841 stored elements in Compressed Sparse Row format>
```

```
test_X
```

```
<10000x93003 sparse matrix of type '<class 'numpy.int64'>'
        with 1361185 stored elements in Compressed Sparse Row format>
```

Working With Text Data - Bag Of Words

```
feature_names = vector.get_feature_names()
print(feature_names[:10])
print(feature_names[10000:10020])
print(feature_names[::5000])
```

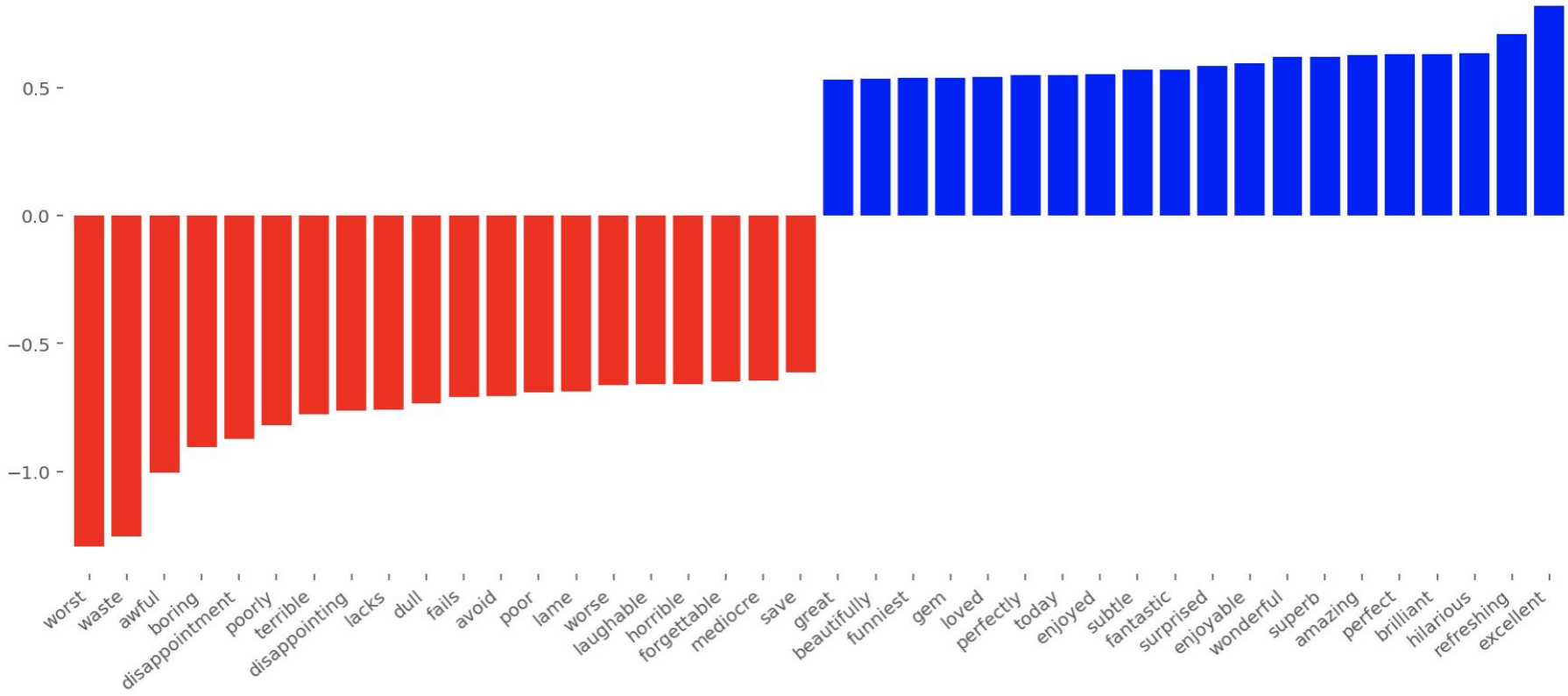
```
['00', '000', '00000000000', '00000001', '00001', '00015', '000dm', '000s', '001', '003830']
['blush', 'blushed', 'blushes', 'blushing', 'blushy', 'bluster', 'blustering', 'blusters', 'blustery', 'blut', 'bluth', 'bluto',
'blvd', 'bly', 'blystone', 'blyth', 'blythe', 'blythen', 'blyton', 'blóðbönd']
['00', 'aqil', 'blush', 'chevening', 'cursed', 'drovers', 'fernack', 'gout', 'hypocrite', 'kentuckian', 'magnavolt', 'msted', 'p
andering', 'psammead', 'rosenberg', 'siodmak', 'supervillainy', 'tumba', 'weixler']
```

Working With Text Data - Bag Of Words

```
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
0.9013
```

Working With Text Data - Bag Of Words



Working With Text Data - Stop Words

```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
print(len(ENGLISH_STOP_WORDS))
print(list(ENGLISH_STOP_WORDS))
```

```
318
['been', 'has', 'its', 'therein', 'to', 'per', 'every', 'who', 'below', 'many', 'back', 'latterly', 'ours', 'in', 'perhaps', 'so
mehow', 'ltd', 'be', 'nobody', 'some', 'seem', 'afterwards', 'side', 'empty', 'out', 'same', 'against', 'both', 'sincere', 'wher
eby', 'how', 'his', 'amoungst', 'con', 'not', 'yet', 'seeming', 'can', 'of', 'for', 'noone', 'cant', 'get', 'fill', 'after', 'th
is', 'thru', 'via', 'the', 'themselves', 'everyone', 'three', 'whose', 'ever', 'everywhere', 'twelve', 'detail', 'system', 'ours
elves', 'either', 'without', 'could', 'serious', 'couldnt', 'sixty', 'because', 'alone', 'eight', 'etc', 'until', 're', 'have',
'anything', 'will', 'that', 'due', 'and', 'toward', 'at', 'become', 'indeed', 'should', 'thin', 'first', 'onto', 'whom', 'betwee
n', 'six', 'made', 'forty', 'found', 'mostly', 'herself', 'yours', 'front', 'nor', 'our', 'sometimes', 'together', 'such', 'anot
her', 'hundred', 'above', 'is', 'any', 'never', 'seems', 'former', 'cannot', 'enough', 'much', 'am', 'latter', 'but', 'besides',
'ten', 'except', 'your', 'still', 'two', 'them', 'by', 'name', 'always', 'bottom', 'so', 'through', 'part', 'why', 'himself', 'w
hereafter', 'meanwhile', 'anyway', 'somewhere', 'other', 'though', 'over', 'might', 'off', 'fifteen', 'an', 'along', 'yourself',
'being', 'are', 'had', 'about', 'move', 'formerly', 'none', 'under', 'around', 'these', 'when', 'fire', 'beforehand', 'what', 'e
verything', 'describe', 'whereas', 'than', 'within', 'neither', 'keep', 'across', 'or', 'ie', 'take', 'else', 'do', 'yourselve
s', 'inc', 'fifty', 'since', 'thence', 'call', 'each', 'nevertheless', 'top', 'almost', 'becoming', 'during', 'also', 'those',
'whatever', 'seemed', 'third', 'him', 'towards', 'must', 'whoever', 'cry', 'thereupon', 'were', 'next', 'thick', 'their', 'eleve
n', 'myself', 'most', 'bill', 'something', 'hereby', 'give', 'whereupon', 'see', 'often', 'me', 'down', 'whether', 'hers', 'las
t', 'there', 'find', 'may', 'wherein', 'few', 'among', 'mill', 'if', 'itself', 'full', 'others', 'whither', 'own', 'well', 'sh
e', 'very', 'we', 'he', 'hence', 'anyhow', 'thereby', 'from', 'then', 'five', 'eg', 'thereafter', 'upon', 'behind', 'even', 'the
refore', 'too', 'again', 'was', 'which', 'as', 'up', 'where', 'you', 'my', 'otherwise', 'further', 'de', 'done', 'all', 'here',
'nowhere', 'nine', 'wherever', 'whence', 'became', 'already', 'twenty', 'four', 'least', 'show', 'i', 'anyone', 'interest', 'whi
le', 'into', 'herein', 'less', 'whole', 'hasnt', 'namely', 'moreover', 'amongst', 'more', 'someone', 'throughout', 'becomes', 'o
n', 'put', 'thus', 'hereafter', 'nothing', 'amount', 'go', 'un', 'sometime', 'co', 'please', 'anywhere', 'however', 'once', 'no
w', 'beyond', 'hereupon', 'a', 'several', 'would', 'before', 'her', 'rather', 'elsewhere', 'whenever', 'although', 'us', 'no',
'one', 'they', 'beside', 'with', 'mine', 'only', 'it']
```

Working With Text Data - Stop Words

```
vector_stop_words = CountVectorizer(stop_words='english')
dev_X = vector_stop_words.fit_transform(dev_text)
test_X = vector_stop_words.transform(test_text)
dev_X
```

```
<40000x92692 sparse matrix of type '<class 'numpy.longlong'>'
        with 3543198 stored elements in Compressed Sparse Row format>
```

Working With Text Data - Stop Words

```
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
0.8922
```

Working With Text Data - Infrequent Words

```
vector_infrequent_words = CountVectorizer(min_df=4)
dev_X = vector_infrequent_words.fit_transform(dev_text)
test_X = vector_infrequent_words.transform(test_text)
print(dev_X.shape)
```

```
(40000, 37693)
```

Working With Text Data - Infrequent Words

```
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
0.9013
```

Working With Text Data - Stemming & Lemmatization

- Lemmatization and stemming are special cases of normalization.
- Stemming and lemmatization both reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.
- **Stemming** just removes or stems the last few characters of a word, often leading to incorrect meanings and spelling (Caring ➜ Car)
- **Lemmatization** considers the context and converts the word to its meaningful base form, which is called Lemma (Caring ➜ Care)

Working With Text Data - Stemming & Lemmatization

```
from nltk.tokenize import sent_tokenize, word_tokenize
def stemSentence(sentence):
    token_words=word_tokenize(sentence)
    stem_sentence=[porter.stem(word) for word in token_words]
    return " ".join(stem_sentence)
```

```
porter = PorterStemmer()
reviews_stem = [stemSentence(review) for review in reviews]
```

Working With Text Data - Stemming & Lemmatization

```
reviews_stem[0]
```

```
"one of the other review ha mention that after watch just 1 oz episod you 'll be hook . they are right , as thi is exactli what happen with me . the
first thing that struck me about oz wa it brutal and unflinch scene of violenc , which set in right from the word go . trust me , thi is not a show
for the faint heart or timid . thi show pull no punch with regard to drug , sex or violenc . it is hardcor , in the classic use of the word . it is
call oz as that is the nicknam given to the oswald maximum secur state penitentari . it focus mainli on emerald citi , an experiment section of the
prison where all the cell have glass front and face inward , so privaci is not high on the agenda . em citi is home to mani .. aryan , muslim , gang
sta , latino , christian , italian , irish and more .... so scuffl , death stare , dodgi deal and shadi agreement are never far away . i would say t
he main appeal of the show is due to the fact that it goe where other show would n't dare . forget pretti pictur paint for mainstream audienc , forg
et charm , forget romanc ... oz doe n't mess around . the first episod i ever saw struck me as so nasti it wa surreal , i could n't say i wa readi f
or it , but as i watch more , i develop a tast for oz , and got accustom to the high level of graphic violenc . not just violenc , but injustic ( cr
ook guard who 'll be sold out for a nickel , inmat who 'll kill on order and get away with it , well manner , middl class inmat be turn into prison
bitch due to their lack of street skill or prison experi ) watch oz , you may becom comfort with what is uncomfort view .... that if you can get in
touch with your darker side ."
```

```
reviews[0]
```

```
"One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened w
ith me.  The first thing that struck me about Oz was its brutality and unflinching scenes of violence, which set in right from the word GO. Trust m
e, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the cla
ssic use of the word.  It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald
City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is
home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreem
ents are never far away.  I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pic
tures painted for mainstream audiences, forget charm, forget romance...OZ doesn't mess around. The first episode I ever saw struck me as so nasty it
was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic
violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well
mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become
comfortable with what is uncomfortable viewing....thats if you can get in touch with your darker side."
```

Working With Text Data - Stemming & Lemmatization

```
dev_text, test_text, dev_y, test_y = train_test_split(reviews_stem, sentiment, test_size=0.2,
                                                      random_state=42)
print(dev_y.value_counts())
print(test_y.value_counts())
```

```
negative    20039
positive    19961
Name: sentiment, dtype: int64
positive    5039
negative    4961
Name: sentiment, dtype: int64
```

```
vector = CountVectorizer()
dev_X = vector.fit_transform(dev_text)
test_X = vector.transform(test_text)
dev_X
```

```
<40000x69379 sparse matrix of type '<class 'numpy.longlong'>'
        with 5251401 stored elements in Compressed Sparse Row format>
```

```
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
0.8944
```

Working With Text Data - Stemming & Lemmatization

```
vector_stop_words = CountVectorizer(stop_words='english')
dev_X = vector_stop_words.fit_transform(dev_text)
test_X = vector_stop_words.transform(test_text)
dev_X
```

```
<40000x69097 sparse matrix of type '<class 'numpy.longlong'>'
        with 3562905 stored elements in Compressed Sparse Row format>
```

```
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
0.8868
```

Working With Text Data - Stemming & Lemmatization

```
def lemma_sentence(sentence):
    token_words=word_tokenize(sentence)
    lemmatize_sentence=[lemmatizer.lemmatize(word) for word in token_words]
    return " ".join(lemmatize_sentence)
```

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
reviews_lemma = [lemma_sentence(review) for review in reviews]
```

Working With Text Data - Stemming & Lemmatization

```
reviews[0]
```

```
"One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened w
ith me.  The first thing that struck me about Oz was its brutality and unflinching scenes of violence, which set in right from the word GO. Trust m
e, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the cla
ssic use of the word.  It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald
City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is
home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreem
ents are never far away.  I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pic
tures painted for mainstream audiences, forget charm, forget romance...OZ doesn't mess around. The first episode I ever saw struck me as so nasty it
was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic
violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well
mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become
comfortable with what is uncomfortable viewing....thats if you can get in touch with your darker side."
```

```
reviews_lemma[0]
```

```
"One of the other reviewer ha mentioned that after watching just 1 Oz episode you 'll be hooked . They are right , a this is exactly what happened w
ith me . The first thing that struck me about Oz wa it brutality and unflinching scene of violence , which set in right from the word GO . Trust me
, this is not a show for the faint hearted or timid . This show pull no punch with regard to drug , sex or violence . Its is hardcore , in the class
ic use of the word . It is called OZ a that is the nickname given to the Oswald Maximum Security State Penitentary . It focus mainly on Emerald City
, an experimental section of the prison where all the cell have glass front and face inwards , so privacy is not high on the agenda . Em City is hom
e to many .. Aryans , Muslims , gangsta , Latinos , Christians , Italians , Irish and more .... so scuffle , death stare , dodgy dealing and shady a
greement are never far away . I would say the main appeal of the show is due to the fact that it go where other show would n't dare . Forget pretty
picture painted for mainstream audience , forget charm , forget romance ... OZ doe n't mess around . The first episode I ever saw struck me a so nas
ty it wa surreal , I could n't say I wa ready for it , but a I watched more , I developed a taste for Oz , and got accustomed to the high level of g
raphic violence . Not just violence , but injustice ( crooked guard who 'll be sold out for a nickel , inmate who 'll kill on order and get away wit
h it , well mannered , middle class inmate being turned into prison bitch due to their lack of street skill or prison experience ) Watching Oz , you
may become comfortable with what is uncomfortable viewing .... thats if you can get in touch with your darker side ."
```

Working With Text Data - Stemming & Lemmatization

```
dev_text, test_text, dev_y, test_y = train_test_split(reviews_lemma, sentiment, test_size=0.2,
                                                      random_state=42)
print(dev_y.value_counts())
print(test_y.value_counts())
vector = CountVectorizer()
dev_X = vector.fit_transform(dev_text)
test_X = vector.transform(test_text)
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
negative    20039
positive    19961
Name: sentiment, dtype: int64
positive    5039
negative    4961
Name: sentiment, dtype: int64
0.8977
```

```
dev_X.shape
```

```
(40000, 88084)
```

Working With Text Data - Stemming & Lemmatization

```
vector = CountVectorizer(stop_words='english')
dev_X = vector.fit_transform(dev_text)
test_X = vector.transform(test_text)
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
0.8882
```

```
dev_X.shape
```

```
(40000, 87774)
```

Working With Text Data - TF-IDF

- TF-IDF can be seen as an alternative to using soft stop words in a sentence.
- The idea here is to down-weight things that are very common.

$$tf - idf\,(t,d) = tf\,(t,d) \cdot idf\,(t)$$

$$idf\,(t) = \log\left(\frac{1+n_d}{1+df\,(d,t)}\right)+1$$

$$n_d - \text{ total number of documents}$$

$$df\,(d,t) - \text{ number of documents containing term } t$$

Working With Text Data - TF-IDF

- TF-IDF can be seen as an alternative to using soft stop words in a sentence.
- The idea here is to down-weight things that are very common.

$$tf - idf\,(t,d) = tf\,(t,d) \cdot idf\,(t)$$

$$idf\,(t) = \log\left(\frac{1+n_d}{1+df\,(d,t)}\right)+1$$

$$n_d - \text{ total number of documents}$$

$$df\,(d,t) - \text{ number of documents containing term } t$$

Working With Text Data - TF-IDF

```
dev_text, test_text, dev_y, test_y = train_test_split(reviews, sentiment, test_size=0.2,
                                                      random_state=42)
print(dev_y.value_counts())
print(test_y.value_counts())
vector = TfidfVectorizer()
dev_X = vector.fit_transform(dev_text)
test_X = vector.transform(test_text)
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
negative    20039
positive    19961
Name: sentiment, dtype: int64
positive    5039
negative    4961
Name: sentiment, dtype: int64
0.9053
```

Slightly better than BOW

```
dev_X.shape
```

```
(40000, 93002)
```

Working With Text Data - TF-IDF

```
dev_text, test_text, dev_y, test_y = train_test_split(reviews, sentiment, test_size=0.2,
                                                      random_state=42)
print(dev_y.value_counts())
print(test_y.value_counts())
vector = TfidfVectorizer(stop_words='english')
dev_X = vector.fit_transform(dev_text)
test_X = vector.transform(test_text)
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
negative    20039
positive    19961
Name: sentiment, dtype: int64
positive    5039
negative    4961
Name: sentiment, dtype: int64
0.8972
```

```
dev_X.shape
```

```
(40000, 92691)
```

Working With Text Data - Putting everything together

Stemming

TF-IDF

```
porter = PorterStemmer()
reviews_stem = [stemSentence(review) for review in reviews]
dev_text, test_text, dev_y, test_y = train_test_split(reviews_stem, sentiment, test_size=0.2,
                                                      random_state=42)
print(dev_y.value_counts())
print(test_y.value_counts())
vector = TfidfVectorizer(min_df=4)
dev_X = vector.fit_transform(dev_text)
test_X = vector.transform(test_text)
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
negative    20039
positive    19961
Name: sentiment, dtype: int64
positive    5039
negative    4961
Name: sentiment, dtype: int64
```

```
0.8977
```

```
dev_X.shape
```

```
(40000, 26018)
```

Working With Text Data - N-grams

- Bag of words limitation
  - Removes order
- Surrounding words provides some context
  - “like” v.s. “Don’t like”
- Unigrams looks at single words (what we have done so far)
- Bigrams looks at two words at a time
- Trigrams ➜ three words at a time

Working With Text Data - N-grams

```
got = ["Winter is coming",
       "Chaos isn't a pit. Chaos is a ladder"]
vector_unigram = CountVectorizer(ngram_range =(1, 1))
vector_unigram.fit(got)
print(vector_unigram.get_feature_names())
```

```
['chaos', 'coming', 'is', 'isn', 'ladder', 'pit', 'winter']
```

```
got = ["Winter is coming",
       "Chaos isn't a pit. Chaos is a ladder"]
vector_unigram = CountVectorizer(ngram_range =(2, 2))
vector_unigram.fit(got)
print(vector_unigram.get_feature_names())
```

```
['chaos is', 'chaos isn', 'is coming', 'is ladder', 'isn pit', 'pit chaos', 'winter is']
```

```
got = ["Winter is coming",
       "Chaos isn't a pit. Chaos is a ladder"]
vector_unigram = CountVectorizer(ngram_range =(1, 2))
vector_unigram.fit(got)
print(vector_unigram.get_feature_names())
```

```
['chaos', 'chaos is', 'chaos isn', 'coming', 'is', 'is coming', 'is ladder', 'isn', 'isn
pit', 'ladder', 'pit', 'pit chaos', 'winter', 'winter is']
```

Working With Text Data - N-grams

```
vector_unigram = CountVectorizer(ngram_range =(1, 1))
vector_bigram = CountVectorizer(ngram_range =(2, 2))
vector_trigram = CountVectorizer(ngram_range =(3, 3))
vector_4gram = CountVectorizer(ngram_range =(4, 4))
print(f"unigram - vocabulary size:", len(vector_unigram.fit(reviews).vocabulary_))
print(f"bigram - vocabulary size:", len(vector_bigram.fit(reviews).vocabulary_))
print(f"trigram - vocabulary size:", len(vector_trigram.fit(reviews).vocabulary_))
print(f"4-grams - vocabulary size:", len(vector_4gram.fit(reviews).vocabulary_))
```

```
unigram - vocabulary size: 101895
bigram - vocabulary size: 2380722
trigram - vocabulary size: 6620959
4-grams - vocabulary size: 9343847
```

Working With Text Data - N-grams

```
cv = CountVectorizer(ngram_range=(1, 2), min_df=4)
cv.fit(reviews)
print("(1, 2), min_df=4: ", len(cv.vocabulary_))
cv = CountVectorizer(ngram_range=(1, 2), min_df=4,
                     stop_words="english")
cv.fit(reviews)
print("(1, 2), stopwords, min_df=4: ", len(cv.vocabulary_))
```

```
(1, 2), min_df=4:  334679
(1, 2), stopwords, min_df=4:  202905
```

Working With Text Data - N-grams

```
cv4 = CountVectorizer(ngram_range=(4, 4), min_df=4)
cv4.fit(reviews)
cv4sw = CountVectorizer(ngram_range=(4, 4), min_df=4,
                        stop_words="english")
cv4sw.fit(reviews)
print(len(cv4.get_feature_names()))
print(len(cv4sw.get_feature_names()))
```

```
117804
2107
```

Working With Text Data - N-grams

```
print(list(cv4sw.vocabulary_.keys())[:20])
```

```
['looking forward watching film', 'high school film project', 'wrong place wrong time',
'really does look like', 'america funniest home videos', 'le conseguenze dell amore', 've
seen movie times', 'streets new york city', 'albert finney tom courtenay', 'best picture
best director', 'probably worst movie seen', 'worst movie seen life', 'movie complete was
te time', 'karen sarah michelle gellar', 'definitely worst movie seen', 'worst movie seen
seen', 'just plain bad bad', 'bad bad bad bad', 'bad bad bad movie', 'wait wait wait wai
t']
```

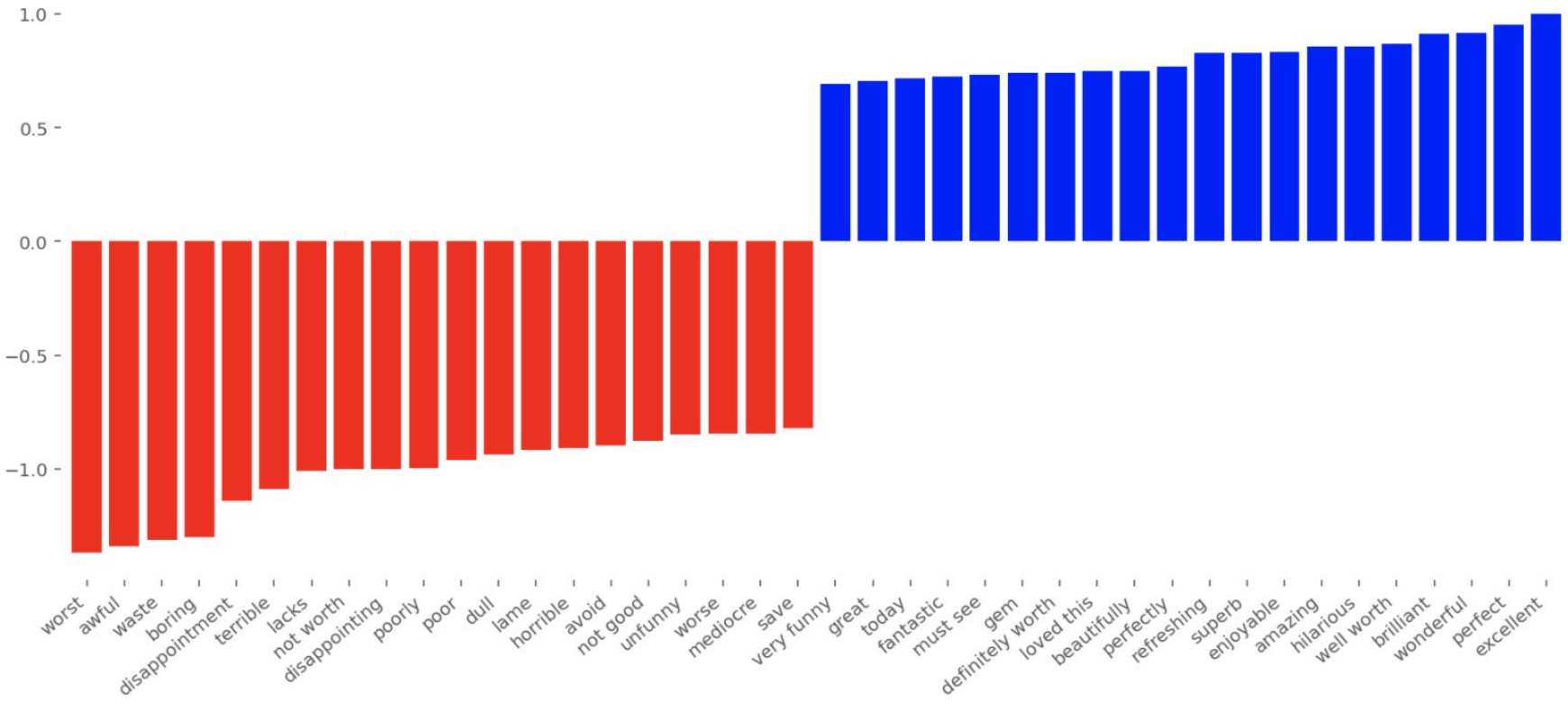
Working With Text Data - N-grams

```
dev_text, test_text, dev_y, test_y = train_test_split(reviews, sentiment, test_size=0.2,
                                                      random_state=42)
print(dev_y.value_counts())
print(test_y.value_counts())
vector = CountVectorizer(ngram_range=(1, 2), min_df=4)
dev_X = vector.fit_transform(dev_text)
test_X = vector.transform(test_text)
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
```

```
negative    20039
positive    19961
Name: sentiment, dtype: int64
positive    5039
negative    4961
Name: sentiment, dtype: int64
```

```
0.9089
```

Working With Text Data - N-grams

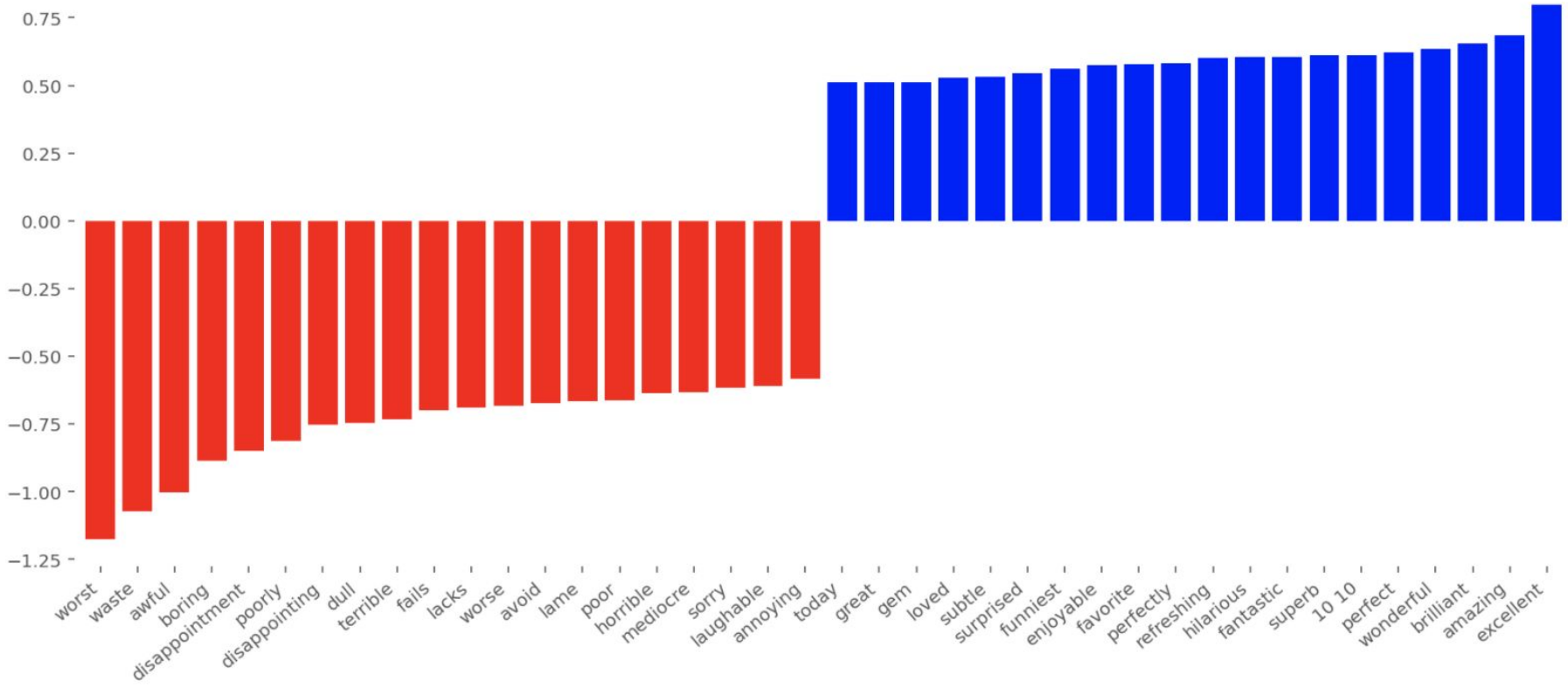


Working With Text Data - N-grams

```
dev_text, test_text, dev_y, test_y = train_test_split(reviews, sentiment, test_size=0.2,
                                                      random_state=42)
print(dev_y.value_counts())
print(test_y.value_counts())
cv4sw = CountVectorizer(ngram_range=(1, 4), min_df=4,
                        stop_words="english")
dev_X = cv4sw.fit_transform(dev_text)
test_X = cv4sw.transform(test_text)
lr = LogisticRegressionCV().fit(dev_X, dev_y)
lr.score(test_X, test_y)
print(len(cv4sw.vocabulary_))
```

```
negative    20039
positive    19961
Name: sentiment, dtype: int64
positive    5039
negative    4961
Name: sentiment, dtype: int64
178475
```

Working With Text Data - N-grams



Working With Text Data - What else can we do?

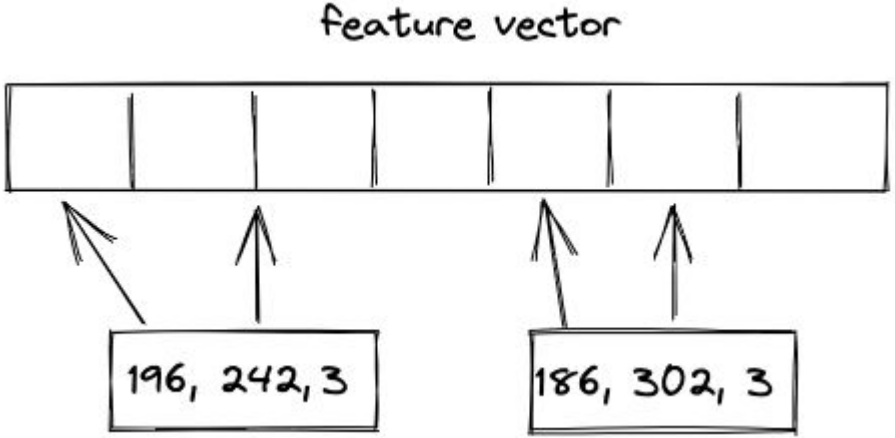
- Document length
- Positive & Negative words
- Special formatting (ALL CAPS, punctuations, etc.)

Working With Text Data - Large Scale Tokenization

- Sometimes when the vocabulary is large, creating Bag-Of-Words may not be the efficient way to tokenize
- Given the sparsity, we could instead use a hash function for hashing the string
- We would not need to store a vocabulary
- Especially efficient when dealing with data like Twitter streams
- Collisions are not necessarily an issue for model accuracy

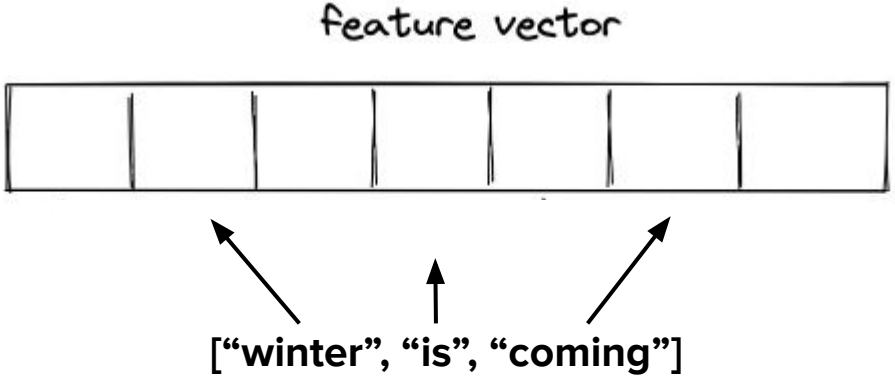
Learning with Sparse Data - Feature Hashing (**from Lecture6**)

- Feature hashing is a fast and space-efficient way to vectorize features i.e. converting features to indices in a vector/array
- The conversion is typically achieved using hash functions (murmur3, MD5, SHA1, SHA2, etc.)



Learning with Sparse Data - Feature Hashing

- Feature hashing is a fast and space-efficient way to vectorize features i.e. converting features to indices in a vector/array
- The conversion is typically achieved using hash functions (murmur3, MD5, SHA1, SHA2, etc.)



Working With Text Data - Large Scale Tokenization

- **Pros:**
  - Fast & memory efficient
  - Works well with large datasets (like Twitter streams)

- **Cons:**
  - Hard to interpret & debug

Working With Text Data - Large Scale Tokenization

```
from sklearn.feature_extraction.text import HashingVectorizer
dev_text, test_text, dev_y, test_y = train_test_split(reviews, sentiment, test_size=0.2,
                                                      random_state=42)
print(dev_y.value_counts())
print(test_y.value_counts())
hv = HashingVectorizer()
dev_X = hv.fit_transform(dev_text)
test_X = hv.transform(test_text)
lr = LogisticRegressionCV().fit(dev_X, dev_y)
print(lr.score(test_X, test_y))
```

```
negative    20039
positive    19961
Name: sentiment, dtype: int64
positive    5039
negative    4961
Name: sentiment, dtype: int64
0.8983
```

```
dev_X.shape
```

```
(40000, 1048576)
```

Working With Text Data - Limitations Of Current Approach

- Word semantics not captured
- Synonymous words are not represented
- Creates long feature vectors with difficulty to reason

Questions?

Let’s take a 10 min break!

Topic Models

Topic Models

- Each document is assumed to be mixture of topics
- Topics are considered to be mixture of words
- Topic models group documents and words simultaneously
- Unsupervised learning problem

Topic Models

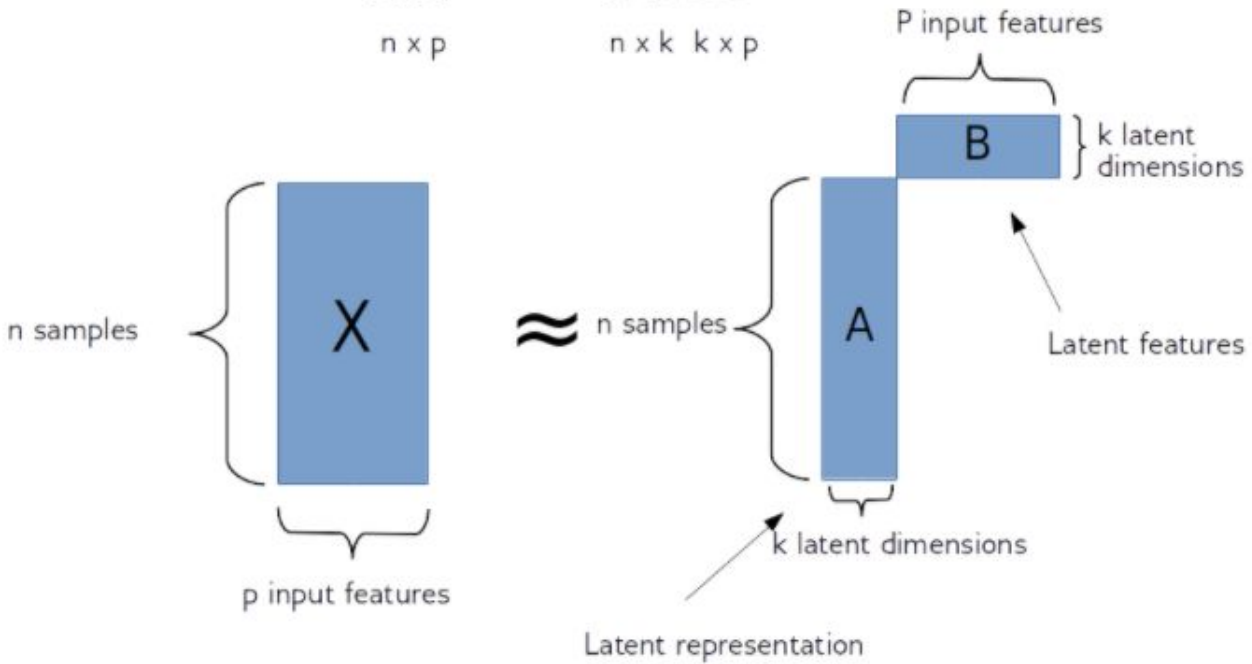
- Latent Semantic Analysis (LSA)
- Non-Matrix Factorization (NMF)
- Latent Dirichlet Allocation (LDA)

Latent Semantic Analysis (LSA)

Topic Models - Matrix Factorization

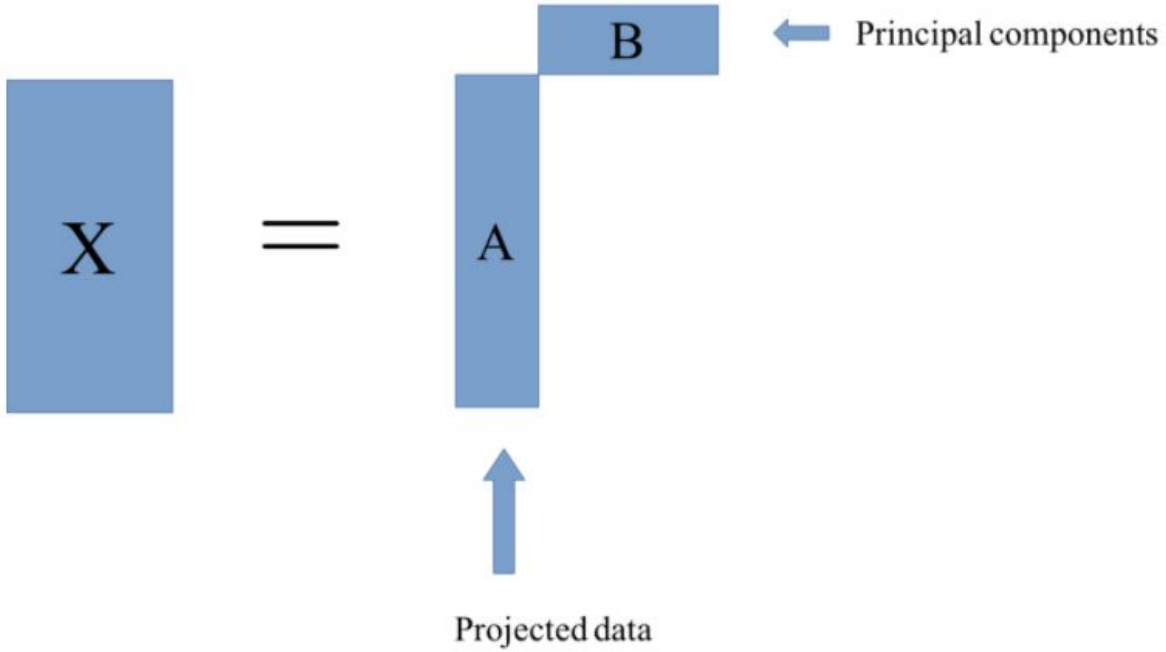
$$X \approx AB$$

n x p n x k k x p



https://amueller.github.io/COMS4995-s20/slides/aml-16-topic-models/#p6

Topic Models - Principal Component Analysis (PCA)



https://amueller.github.io/COMS4995-s20/slides/aml-16-topic-models/#p7

Topic Models - Latent Semantic Analysis (LSA)

- Dimensionality reduction technique
- Modified version of PCA
    - Does not subtract mean
- Truncated Singular Value Decomposition (SVD)
- Convex optimization

Topic Models - Latent Semantic Analysis (LSA)

```
from sklearn.feature_extraction.text import CountVectorizer
vector = CountVectorizer(stop_words="english", min_df=4)
dev_X = vector.fit_transform(dev_text)
dev_X.shape
```

```
(40000, 37385)
```

Topic Models - Latent Semantic Analysis (LSA)

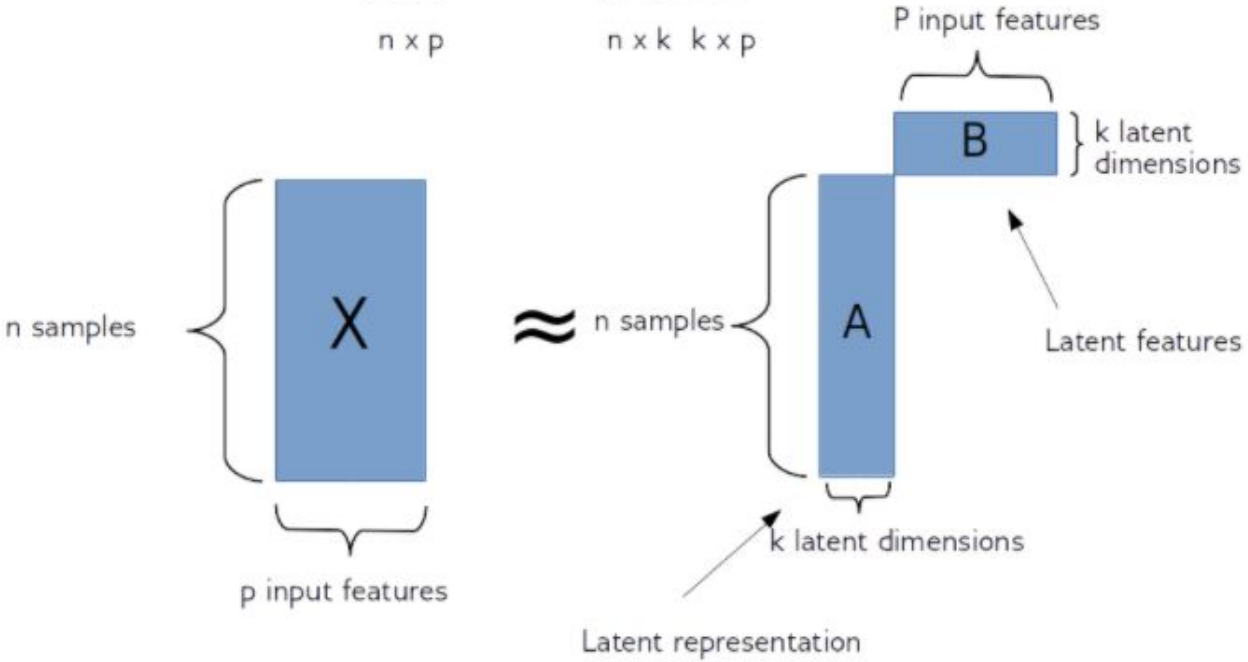
```
from sklearn.decomposition import TruncatedSVD
lsa = TruncatedSVD(n_components=100)
X_lsa = lsa.fit_transform(dev_X)
print(lsa.components_.shape)
print(X_lsa.shape)
```

```
(100, 37385)
(40000, 100)
```

Topic Models - Latent Semantic Analysis (LSA)

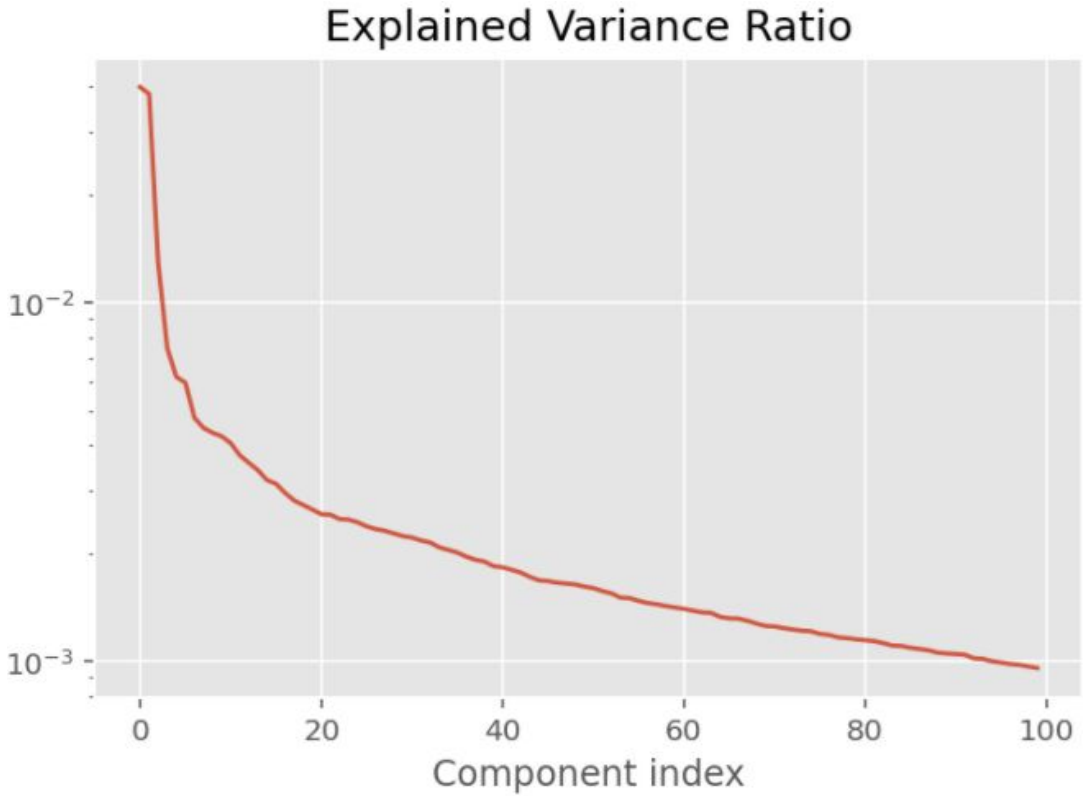
$$X \approx AB$$

n x p n x k k x p

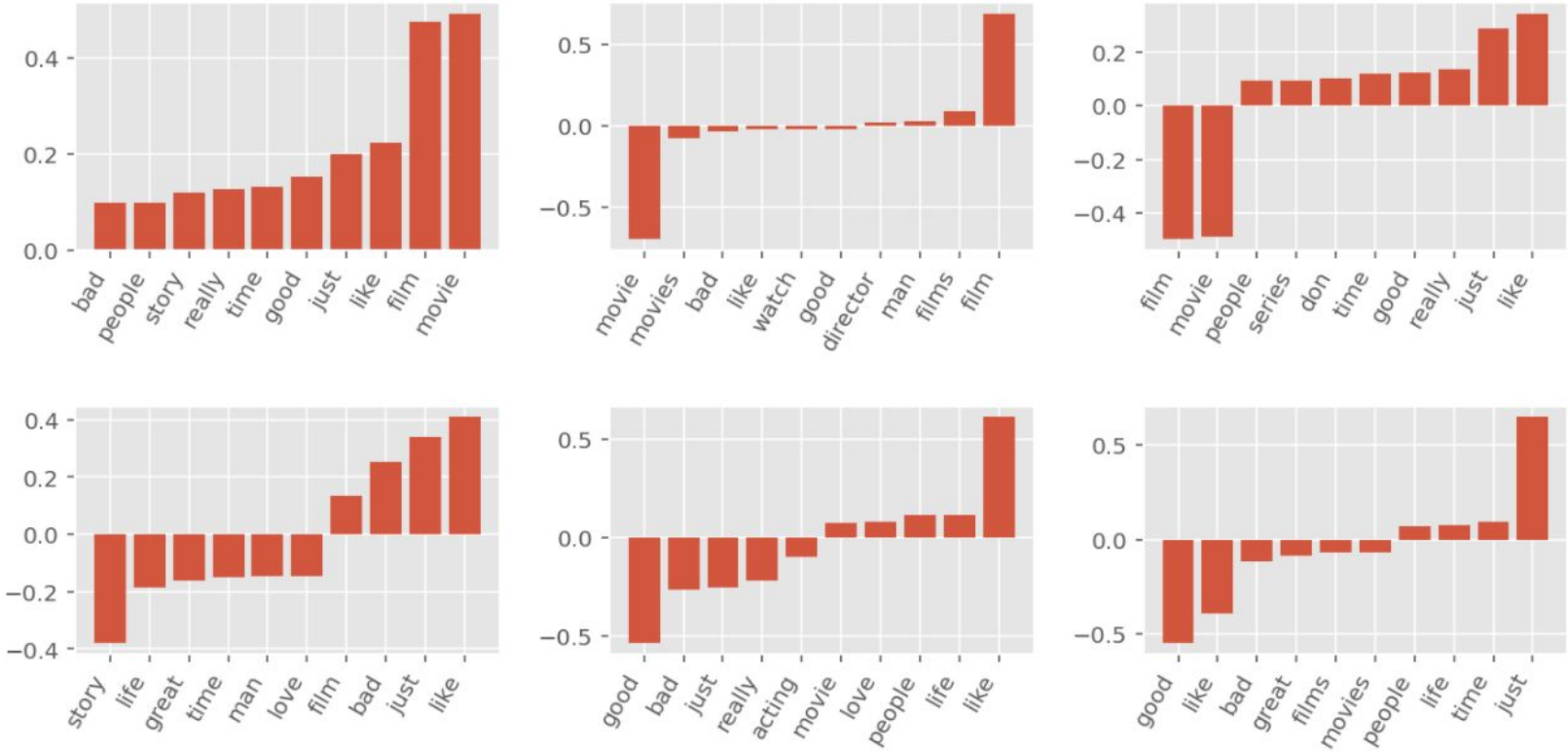


n → 40000
p → 37385
K → 100

Topic Models - Latent Semantic Analysis (LSA)



Topic Models - Latent Semantic Analysis (LSA)

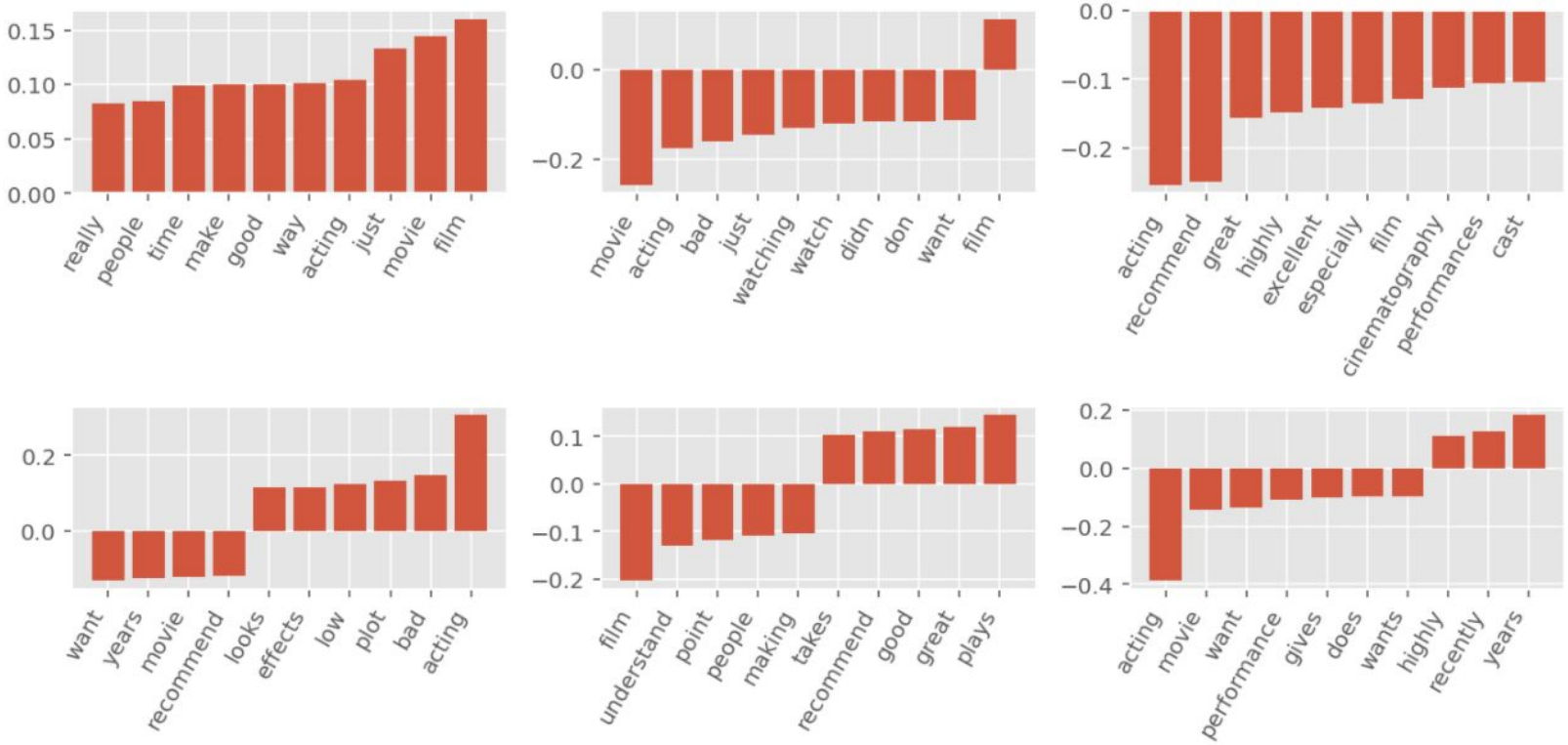


Topic Models - Latent Semantic Analysis (LSA)

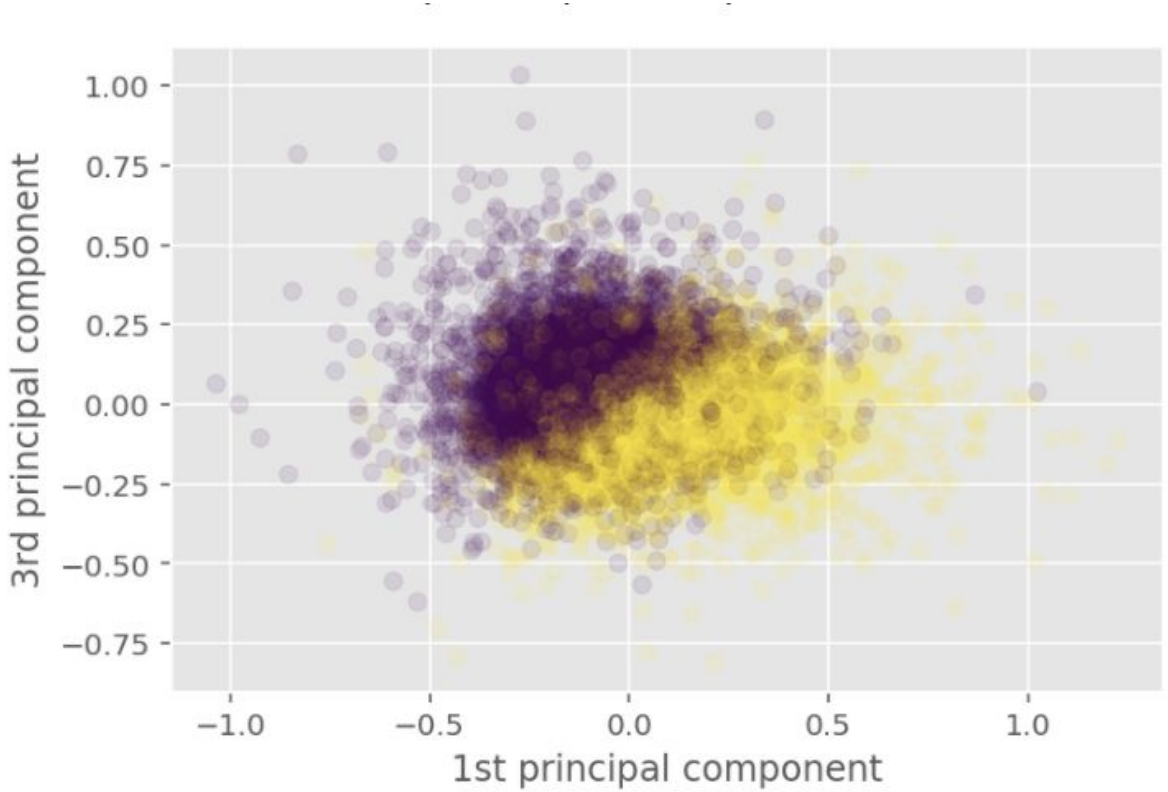
```
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
dev_X_scaled = scaler.fit_transform(dev_X)
lsa_scaled = TruncatedSVD(n_components=100)
dev_X_lsa_scaled = lsa_scaled.fit_transform(dev_X_scaled)
print(dev_X_lsa_scaled.shape)
```

```
(40000, 100)
```

Topic Models - Latent Semantic Analysis (LSA)

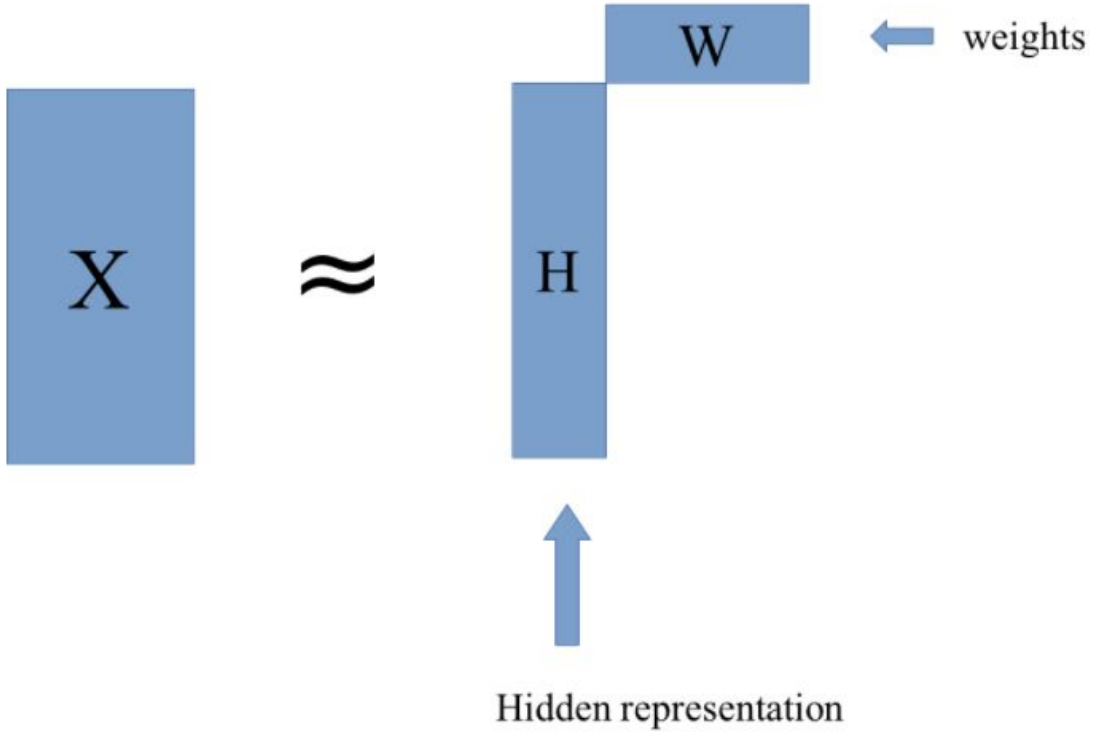


Topic Models - Latent Semantic Analysis (LSA)



Non-Matrix Factorization (NMF)

Topic Models - Non-Matrix Factorization (NMF)

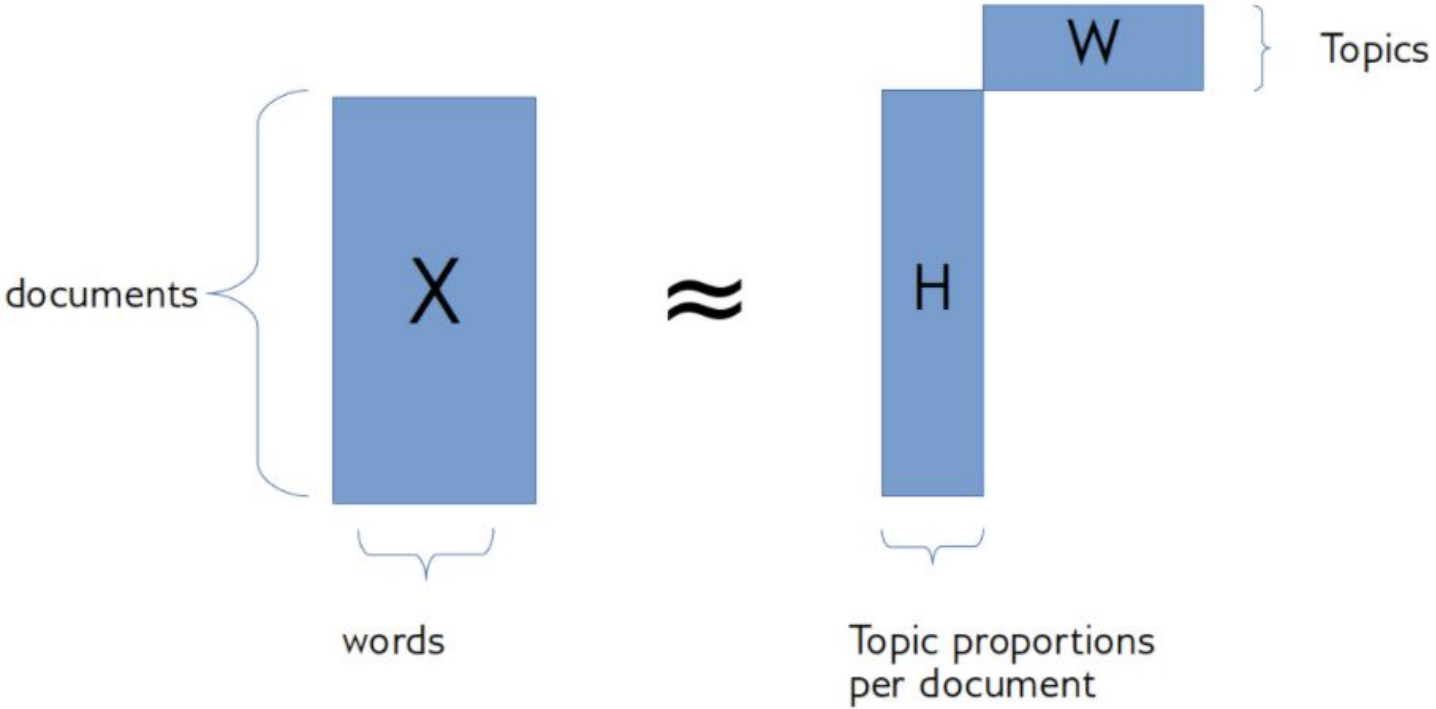


https://amueller.github.io/COMS4995-s20/slides/aml-16-topic-models/#p15

Topic Models - Non-Matrix Factorization (NMF)

$$\min_{W \in \mathbb{R}^{p \times r}, H \in \mathbb{R}^{r \times n}} ||X - WH||_F^2 \quad \text{such that} \quad W \geq 0 \text{ and } H \geq 0.$$

Topic Models - Non-Matrix Factorization (NMF)



https://amueller.github.io/COMS4995-s20/slides/aml-16-topic-models/#p19

Topic Models - Non-Matrix Factorization (NMF)

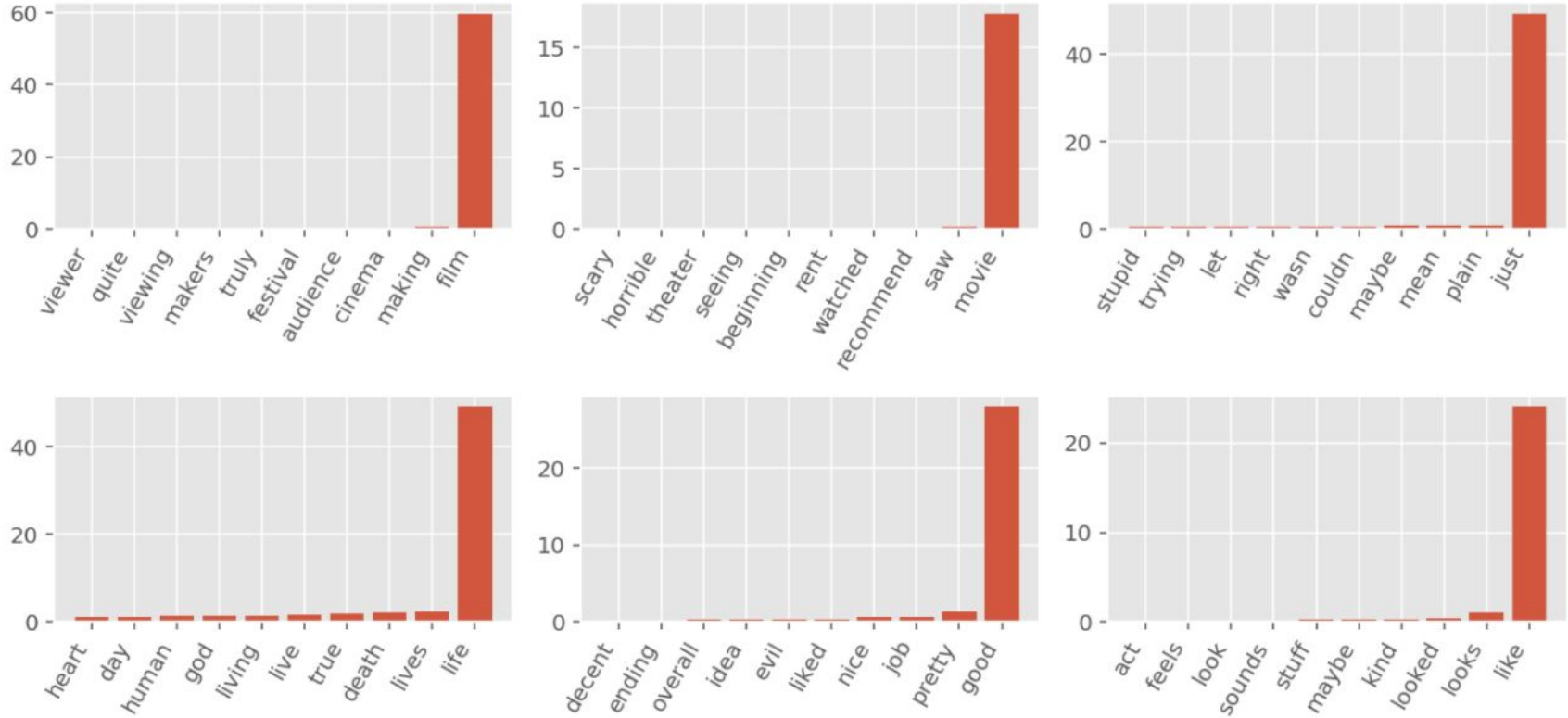
```
from sklearn.decomposition import NMF
nmf_scale = NMF(n_components=100, verbose=10, tol=0.01)
nmf_scale.fit(dev_X)
```

(without scaling)

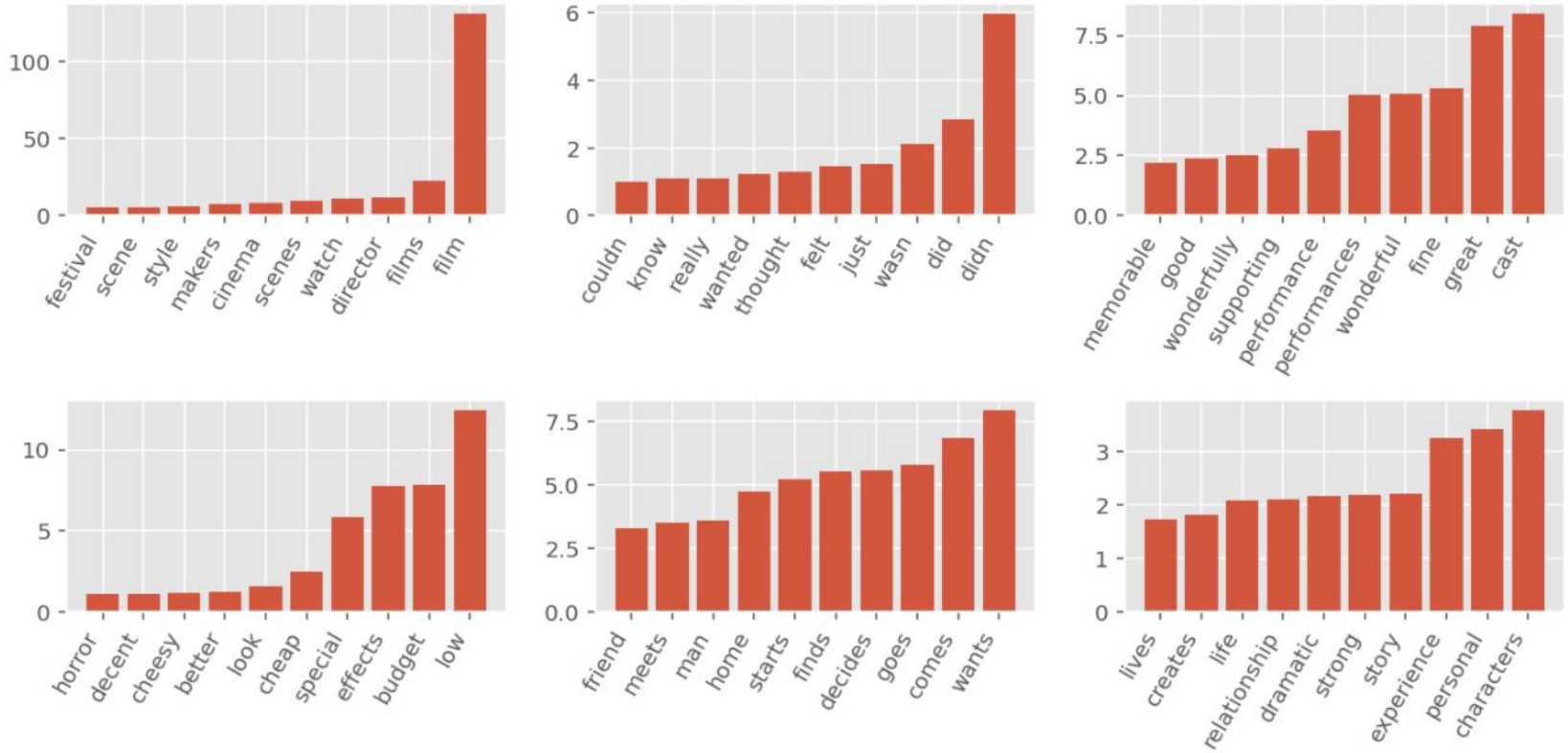
```
from sklearn.decomposition import NMF
nmf_scale = NMF(n_components=100, verbose=10, tol=0.01)
nmf_scale.fit(dev_X_scaled)
```

(with scaling)

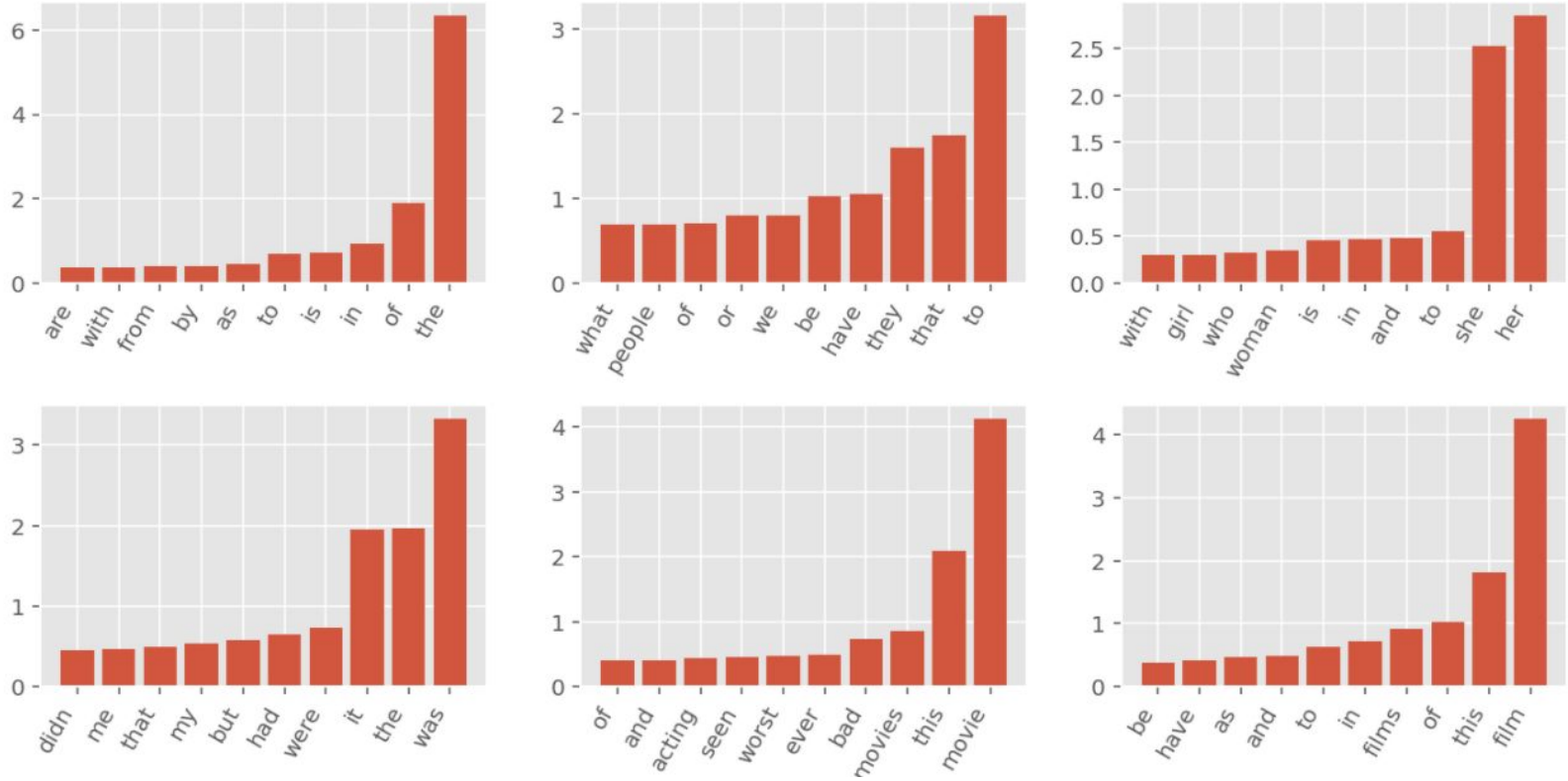
Topic Models - Non-Matrix Factorization (No Scaling)



Topic Models - Non-Matrix Factorization (Scaling)

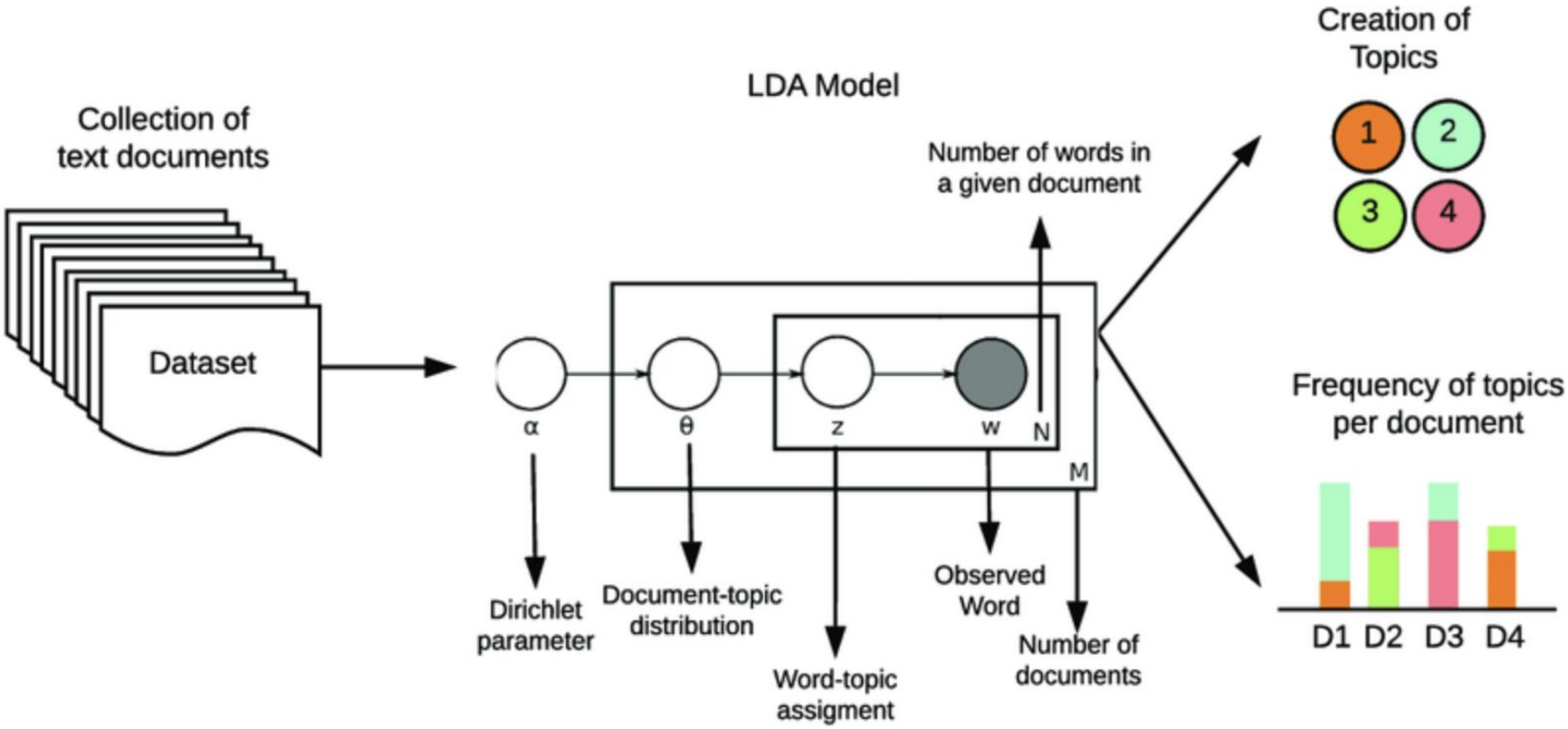


Topic Models - Non-Matrix Factorization with TF-IDF



Latent Dirichlet Allocation (LDA)

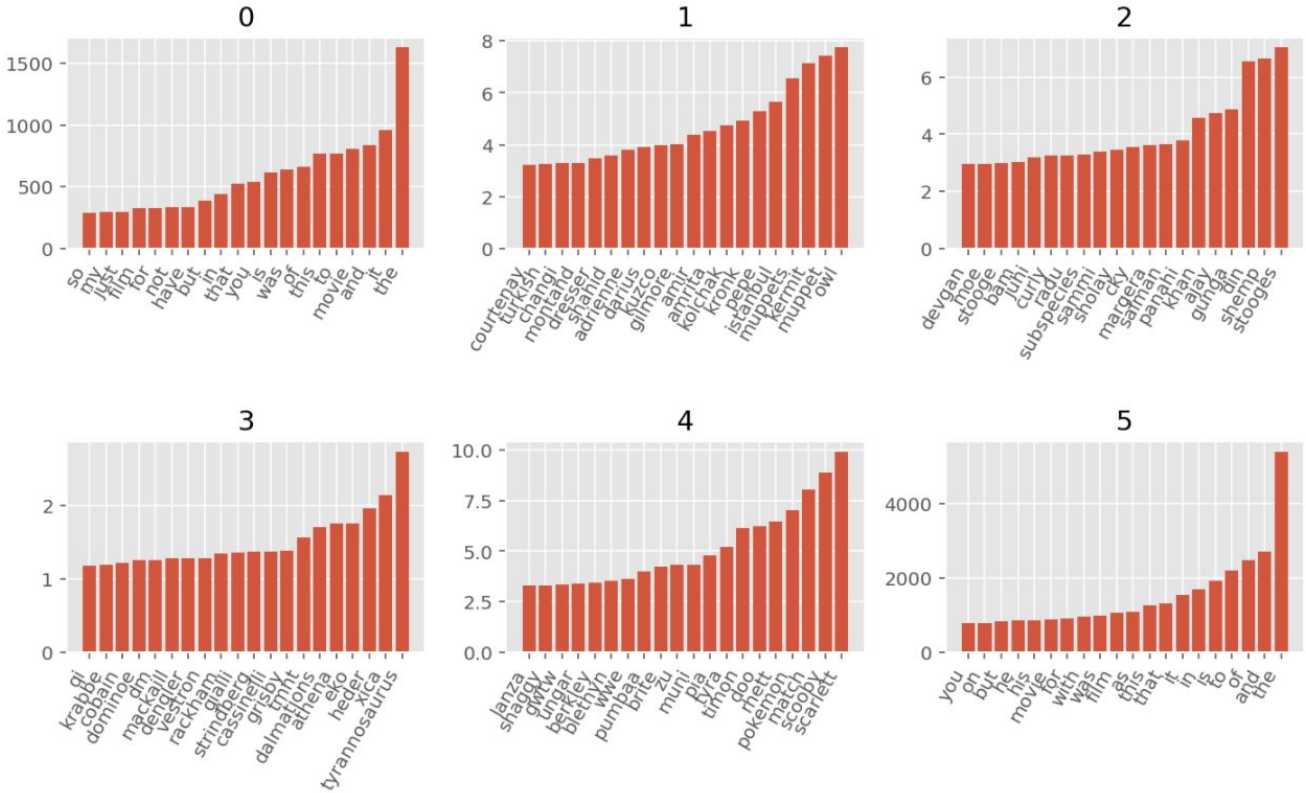
Topic Models - Latent Dirichlet Allocation (LDA)



Topic Models - Latent Dirichlet Allocation (LDA)

```
from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components=10, learning_method="batch")
X_lda = lda.fit_transform(dev_X)
```

Topic Models - Latent Dirichlet Allocation (LDA)



Topic Models - Guided LDA

- Guided LDA disambiguate topics by being able to provide seed words for topics
- guidedlda python package

```
1.  (politics): Barack Obama, elections, PM, Narendra Modi,
2.  (Sports): Cricket team, world cup, FIFA
3.  (Science): Einstein, Nobel Prize, Physics, Medicine, biological
4.  (Space/Tech): SpaceX, Tesla, Google, Apple, iPhone
```

```
1.  (Seed politics): Barack Obama, elections, PM, Narendra Modi,
2.  (Seed Sports): Cricket team, world cup, FIFA
3.  (Seed Science): Einstein, Nobel Prize, Physics, Medicine, biological
4.  (Seed Space): SpaceX, Nasa, Solar Eclipse
5.  (Seed Tech): Tesla, Google, Apple, iPhone
```

https://www.freecodecamp.org/news/how-we-changed-unsupervised-lda-to-semi-supervised-guidedlda-e36a95f3a164/

Word & Document Embeddings

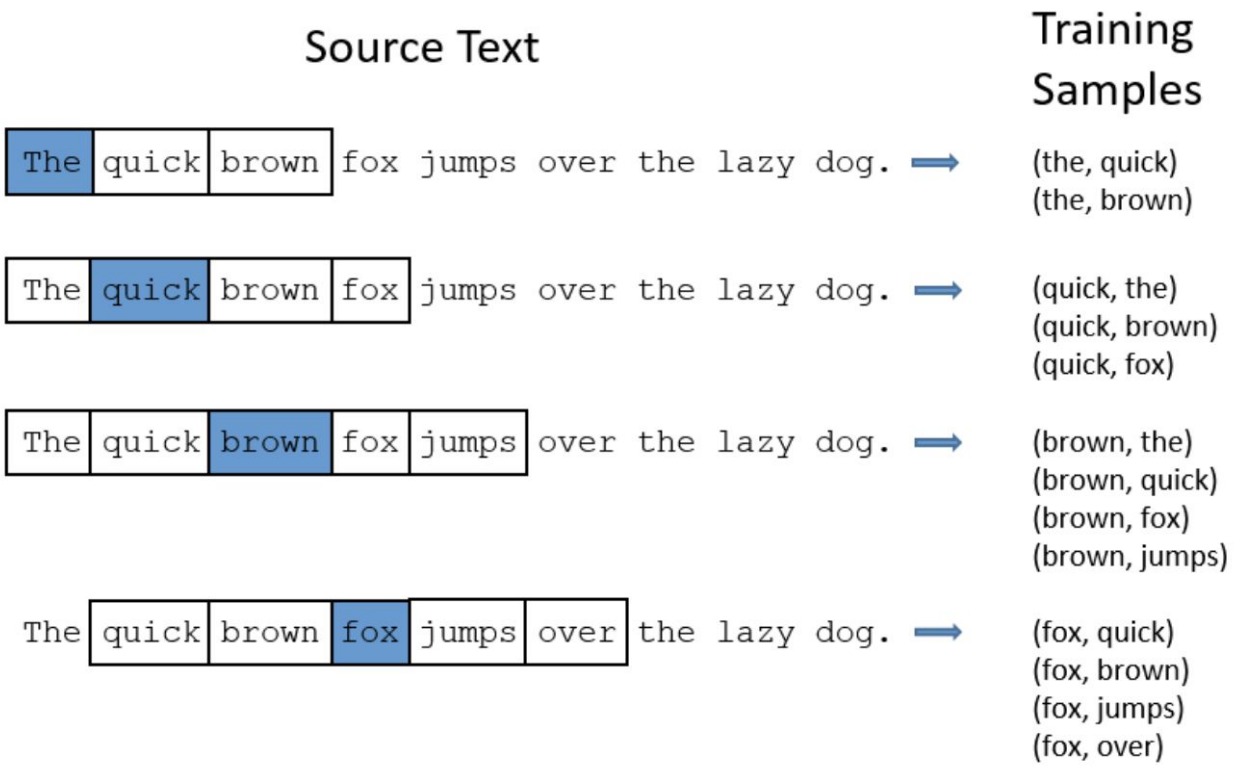
Word Embeddings

- So far, we have tried learning representations for the documents
- We will try learning representations (embeddings) for words
- We will learn a general representation using a large corpus
- The input would still be a one-hot representation (as in BoW)
- We use will an auxiliary task to learn these general representations
- These models are generally called “Word2Vec” class of models

Word2Vec - Skip-Gram Models

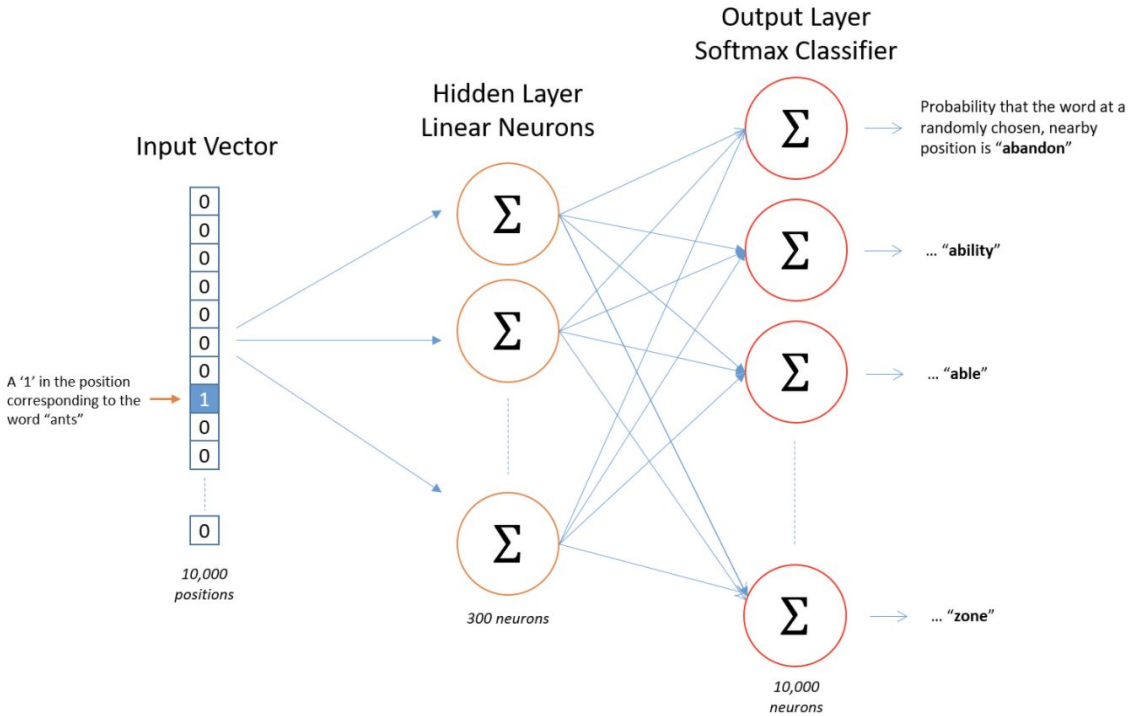
- Supervised learning problem
  - Predict the surrounding word, given a word
- Documents provide a lot of training examples
- Task is to learn the representation and **not** to achieve model accuracy

Word2Vec - Skip-Gram Models



http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Word2Vec - Skip-Gram Models

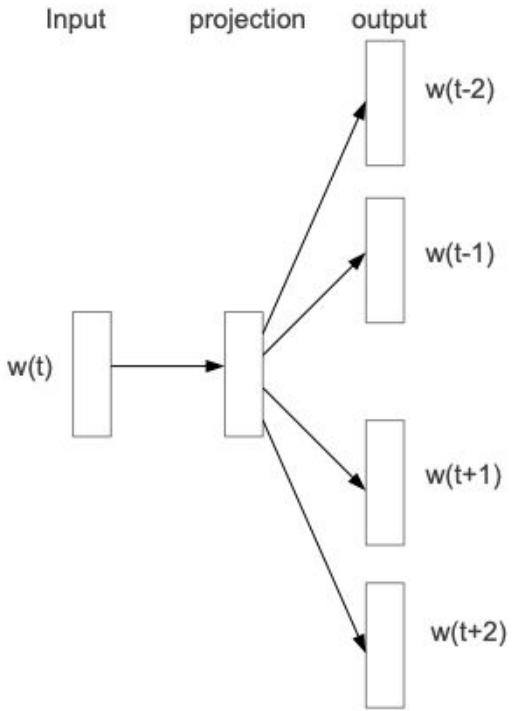


http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Word2Vec - Skip-Gram Models

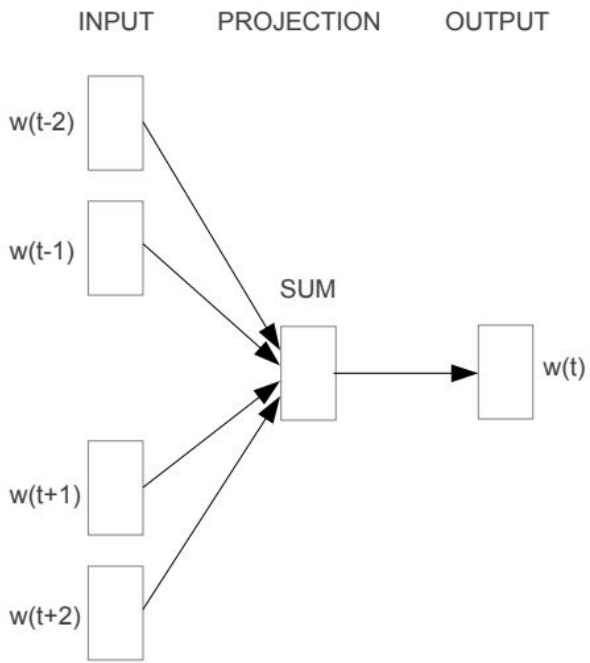
$$Max \quad \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log p(w_{t+j}|w_t)$$

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top} v_{w_I}\right)}$$



https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

Word2Vec - CBOW Models



https://arxiv.org/pdf/1301.3781.pdf

Word2Vec - Training

- Large number of weights to be determined:
  - 2*10000*300 = 6MM weights!! (window size = 1)
- Improvements to address training
  - Treating common words/phrases as single words
  - Subsampling frequent words
  - Negative sampling

https://arxiv.org/pdf/1301.3781.pdf

Word2Vec - Implementation

- Gensim
- SpaCy
- Tensorflow (train your model)
- Fasttext

https://arxiv.org/pdf/1301.3781.pdf

Word2Vec - Example

```
import spacy
nlp = spacy.load("en_core_web_lg")
doc = nlp("Winter is coming")
for token in doc:
    print(token.text, token.lemma_, token.pos_, token.tag_,
          token.dep_, token.shape_, token.is_alpha, token.is_stop)
```

```
Winter winter NOUN NN nsubj Xxxxx True False
is be AUX VBZ aux xx True True
coming come VERB VBG ROOT xxxx True False
```

Word2Vec - Example

```
doc = nlp("Winter is coming!")
for token in doc:
    print(token.text, token.has_vector, token.vector.shape)
```

```
Winter True (300,)
is True (300,)
coming True (300,)
! True (300,)
```

```
nlp.vocab.vectors.shape
```

```
(684830, 300)
```

Word2Vec - Cosine Similarity

$$\text{similarity}(v, w) = \cos(\theta) = \frac{v^T w}{\|v\| \|w\|}$$

Word2Vec - Example

```
queries = [w for w in nlp.vocab if w.is_lower and w.prob >= -15]
def most_similar(word, count=10):
    by_similarity = sorted(queries, key=lambda w: word.similarity(w), reverse=True)
    return [w.orth_ for w in by_similarity[:count]]
most_similar(nlp("throne"))
```

```
['throne',
 'reign',
 'king',
 'kingdom',
 'prince',
 'ascended',
 'emperor',
 'kings',
 'heir',
 'monarch']
```

Word2Vec - Example

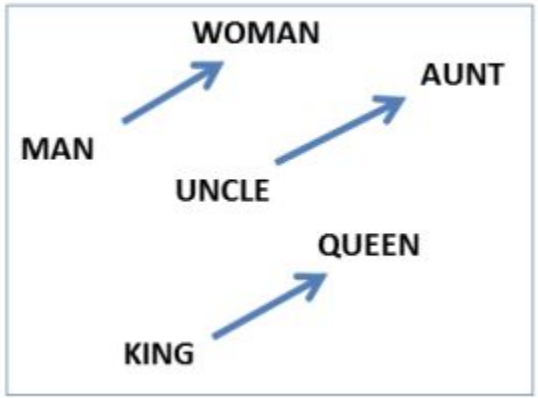
```
most_similar(nlp("game"))
```

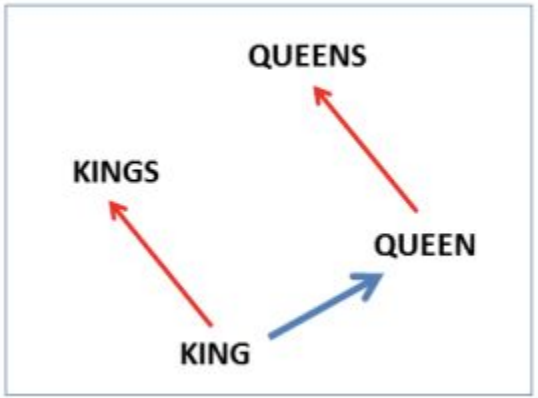
```
['game',
 'games',
 'play',
 'players',
 'playing',
 'multiplayer',
 'played',
 'gameplay',
 'player',
 'tournament']
```

```
most_similar(nlp("machine learning"))
```

```
['machine',
 'learning',
 'machines',
 'learn',
 'computer',
 'skills',
 'teaching',
 'instruction',
 'knowledge',
 'basic']
```

Word2Vec - Analogues & Relationships

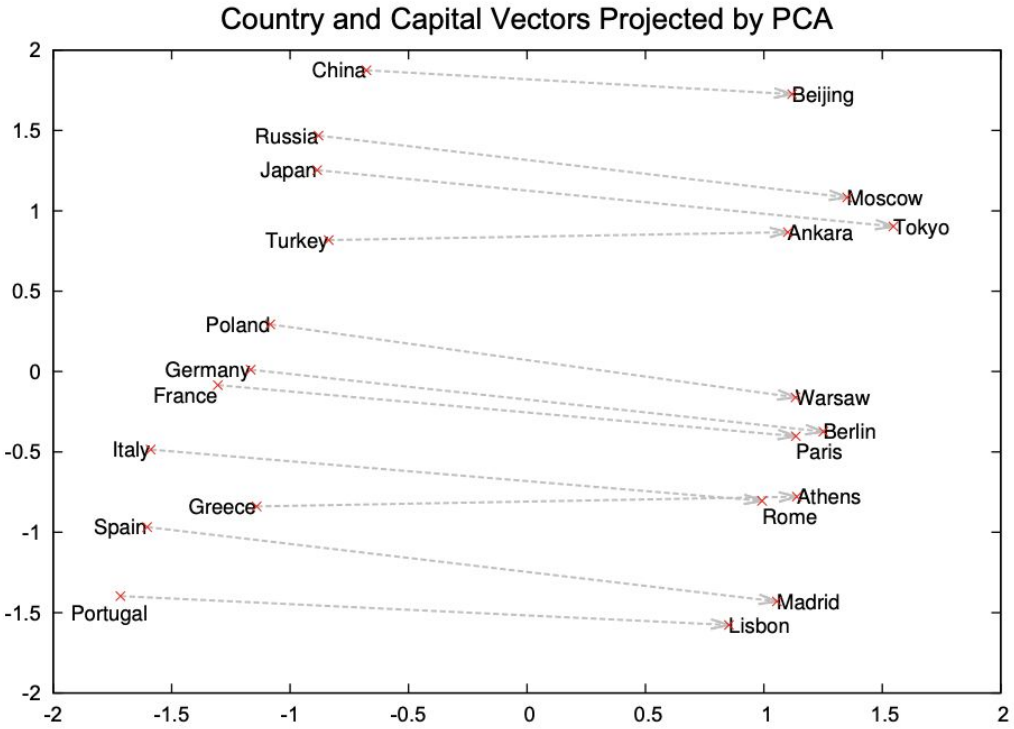




"King is to Kings as Queen is to ?"

https://aclanthology.org/N13-1090.pdf

Word2Vec - Analogues & Relationships



https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

Word2Vec - Relations

$$a : b :: c :?$$

$$d = \arg\max_i \frac{(\text{vec}(b) - \text{vec}(a) + \text{vec}(c))^T vec_i}{\|\text{vec}(b) - \text{vec}(a) + \text{vec}(c)\| \|\text{vec}_i\|}$$

Word2Vec - Relations

City 1 : State containing City 1 : : City 2 : State containing City 2

Input	Result Produced
Chicago : Illinois : : Houston	Texas
Chicago : Illinois : : Philadelphia	Pennsylvania
Chicago : Illinois : : Phoenix	Arizona
Chicago : Illinois : : Dallas	Texas
Chicago : Illinois : : Jacksonville	Florida
Chicago : Illinois : : Indianapolis	Indiana
Chicago : Illinois : : Austin	Texas
Chicago : Illinois : : Detroit	Michigan
Chicago : Illinois : : Memphis	Tennessee
Chicago : Illinois : : Boston	Massachusetts

https://cs224d.stanford.edu/lecture_notes/notes2.pdf

Word2Vec Models - Bias

$$\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{computer programmer}} - \overrightarrow{\text{homemaker}}.$$

Extreme *she* occupations

1. homemaker	2. nurse	3. receptionist
4. librarian	5. socialite	6. hairdresser
7. nanny	8. bookkeeper	9. stylist
10. housekeeper	11. interior designer	12. guidance counselor

Extreme *he* occupations

1. maestro	2. skipper	3. protege
4. philosopher	5. captain	6. architect
7. financier	8. warrior	9. broadcaster
10. magician	11. figher pilot	12. boss

Gender stereotype *she-he* analogies.

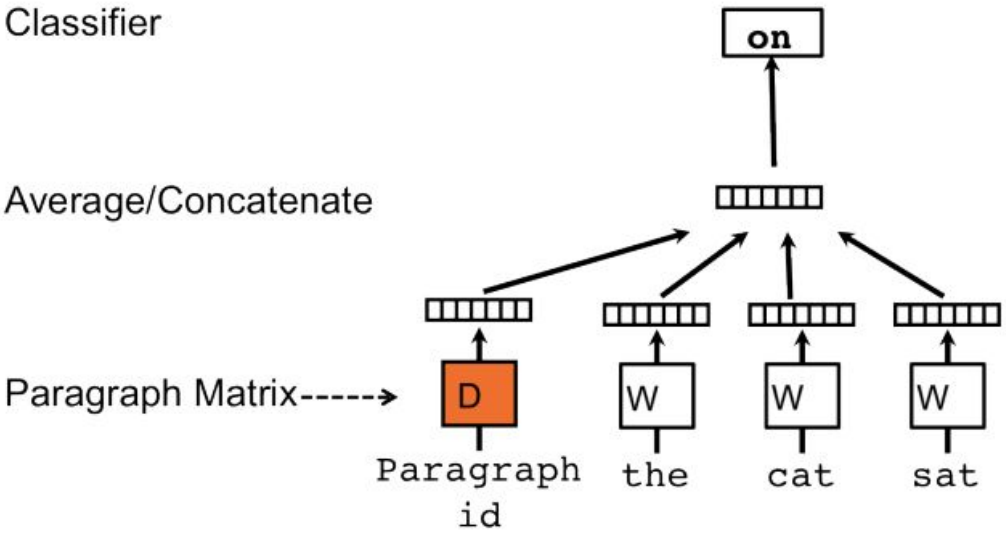
sewing-carpentry	register-nurse-physician	housewife-shopkeeper
nurse-surgeon	interior designer-architect	softball-baseball
blond-burly	feminism-conservatism	cosmetics-pharmaceuticals
giggle-chuckle	vocalist-guitarist	petite-lanky
sassy-snappy	diva-superstar	charming-affable
volleyball-football	cupcakes-pizzas	hairdresser-barber

Gender appropriate *she-he* analogies.

queen-king	sister-brother	mother-father
waitress-waiter	ovarian cancer-prostate cancer	convent-monastery

https://arxiv.org/pdf/1607.06520.pdf

Doc2Vec Models



https://cs.stanford.edu/~quoc/paragraph_vector.pdf

Questions?