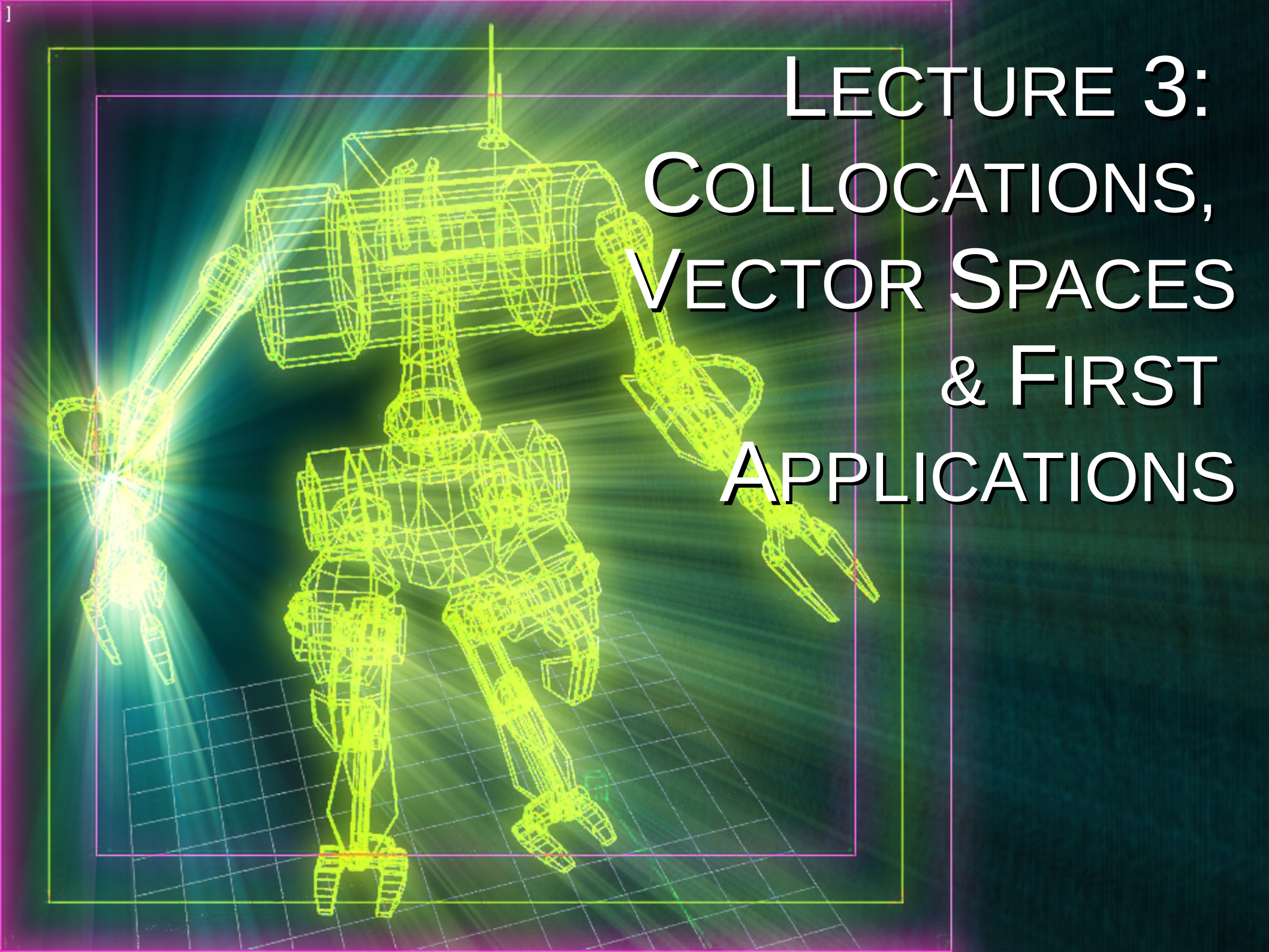


Lecture 3: Collocations, Vector Spaces & First Applications

Pruning the vocabulary

- **Pruning the vocabulary** can be done by several typical rules.
- 1. Known stop word lists. nltk includes one:

```
# download the stopword list if you do not have it already
nltk.download('stopwords')
print(nltk.corpus.stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
"you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself',
'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers',
'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs',
'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being',
'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an',
'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of',
'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down',
'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then',
'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both',
'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not',
'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will',
'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o',
're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven',
"haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't",
'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Pruning the vocabulary

- 2. Overly frequent words (either overall frequency or document frequency). E.g. leave out the top 1% most frequent words.
- 3. Overly infrequent words (either overall frequency or document frequency). E.g. leave out the bottom 35% most infrequent words.
  - In particular: numerous words will occur **only once in the entire collection.** They can help search for a particular document but do not help model content over documents.
  - In 20 Newsgroups, out of 298705 words in the unified vocabulary, 183327 (61%) occur only once.
- 4. Overly short words (e.g. single-character words)
- 5. Overly long words over 20 characters
- 6. Words that contain unwanted characters, e.g. numbers or special characters (but accented characters are ok)

Pruning the vocabulary

- In Python:

```
#%% Vocabulary pruning
nltkstopwords=nltk.corpus.stopwords.words('english')
pruningdecisions=numpy.zeros((len(unifiedvocabulary),1))
for k in range(len(unifiedvocabulary)):
    # Rule 1: check the nltk stop word list
    if (unifiedvocabulary[k] in nltkstopwords):
        pruningdecisions[k]=1
    # Rule 2: if the word is in the top 1% of frequent words
    if (k in highest_totaloccurrences_indices[\
        0:int(numpy.floor(len(unifiedvocabulary)*0.01))]):
        pruningdecisions[k]=1
    # Rule 3: if the word is in the bottom 65% of frequent words
    if (k in highest_totaloccurrences_indices[(int(numpy.floor(\
        len(unifiedvocabulary)*0.35))):len(unifiedvocabulary)]):
        pruningdecisions[k]=1
    # Rule 4: if the word is too short
    if len(unifiedvocabulary[k])<2:
        pruningdecisions[k]=1
    # Rule 5: if the word is too long
    if len(unifiedvocabulary[k])>20:
        pruningdecisions[k]=1
    # Rule 6: if the word has unwanted characters
    # (here for simplicity only a-z allowed)
    if unifiedvocabulary[k].isalpha()==False:
        pruningdecisions[k]=1
```

- These rules prune the vast majority (250626 of 298705) words in Twenty Newsgroups!

Pruning the vocabulary

- In Python:

```
#%% Get indices of documents to remaining words
oldtopruned=[]
tempind=-1
for k in range(len(unifiedvocabulary)):
    if pruningdecisions[k]==0:
        tempind=tempind+1
        oldtopruned.append(tempind)
    else:
        oldtopruned.append(-1)

#%% Create pruned texts

mycrawled_prunedtexts=[]
myindices_in_prunedvocabulary=[]
for k in range(len(mycrawled_lemmatizedtexts)):
    print(k)
    temp_newindices=[]
    temp_newdoc=[]
    for l in range(len(mycrawled_lemmatizedtexts[k])):
        temp_oldindex=myindices_in_unifiedvocabulary[k][l]
        temp_newindex=oldtopruned[temp_oldindex]
        if temp_newindex!=-1:
            temp_newindices.append(temp_newindex)
            temp_newdoc.append(unifiedvocabulary[temp_oldindex])
    mycrawled_prunedtexts.append(temp_newdoc)
    myindices_in_prunedvocabulary.append(temp_newindices)
```

Pruning the vocabulary

- In Python:

```
#%% Get indices of documents to remaining words
oldtopruned=[]
tempind=-1
for k in range(len(unifiedvocabulary)):
    if pruningdecisions[k]==0:
        tempind=tempind+1
        oldtopruned.append(tempind)
    else:
        oldtopruned.append(-1)

#%% Create pruned texts

mycrawled_prunedtexts=[]
myindices_in_prunedvocabulary=[]
for k in range(len(mycrawled_lemmatizedtexts)):
    print(k)
    temp_newindices=[]
    temp_newdoc=[]
    for l in range(len(mycrawled_lemmatizedtexts[k])):
        temp_oldindex=myindices_in_unifiedvocabulary[k][l]
        temp_newindex=oldtopruned[temp_oldindex]
        if temp_newindex!=-1:
            temp_newindices.append(temp_newindex)
            temp_newdoc.append(unifiedvocabulary[temp_oldindex])
    mycrawled_prunedtexts.append(temp_newdoc)
    myindices_in_prunedvocabulary.append(temp_newindices)
```

Pruning the vocabulary

- In Python:

```
#%% Inspect remaining frequent words
# Sort remaining words by largest total (or mean) occurrence count
remainingindices=numpy.squeeze(numpy.where(pruningdecisions==0)[0])
remainingvocabulary=unifiedvocabulary[remainingindices]
remainingvocabulary_totaloccurrencecounts= \
    unifiedvocabulary_totaloccurrencecounts[remainingindices]
remaining_highest_totaloccurrences_indices= \
    numpy.argsort(-1*remainingvocabulary_totaloccurrencecounts,axis=0)
print(numpy.squeeze(remainingvocabulary[remaining_highest_totaloccurrences_indices[1:500]]))
print(numpy.squeeze(remainingvocabulary_totaloccurrencecounts[ \
    remaining_highest_totaloccurrences_indices[1:500]]))
```

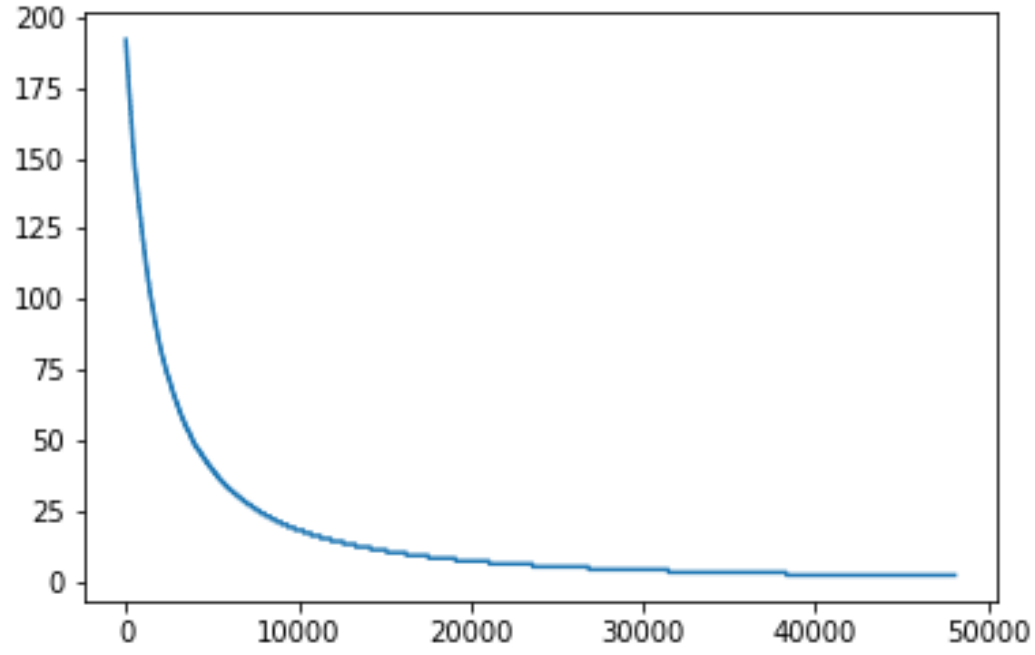
Pruning the vocabulary

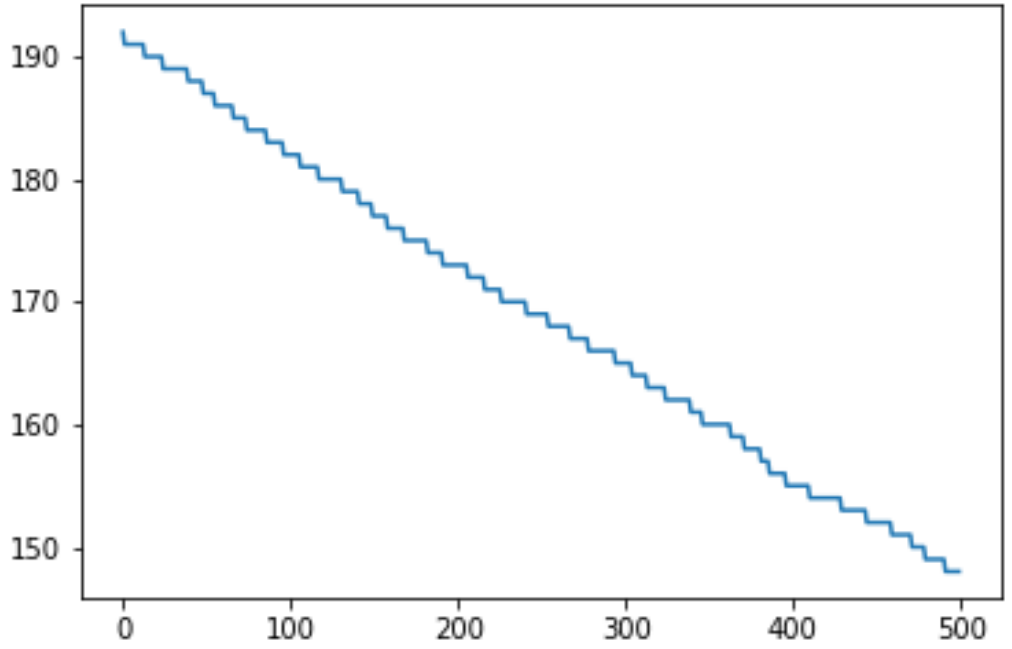
- Remaining top most frequent words and their frequencies after pruning:

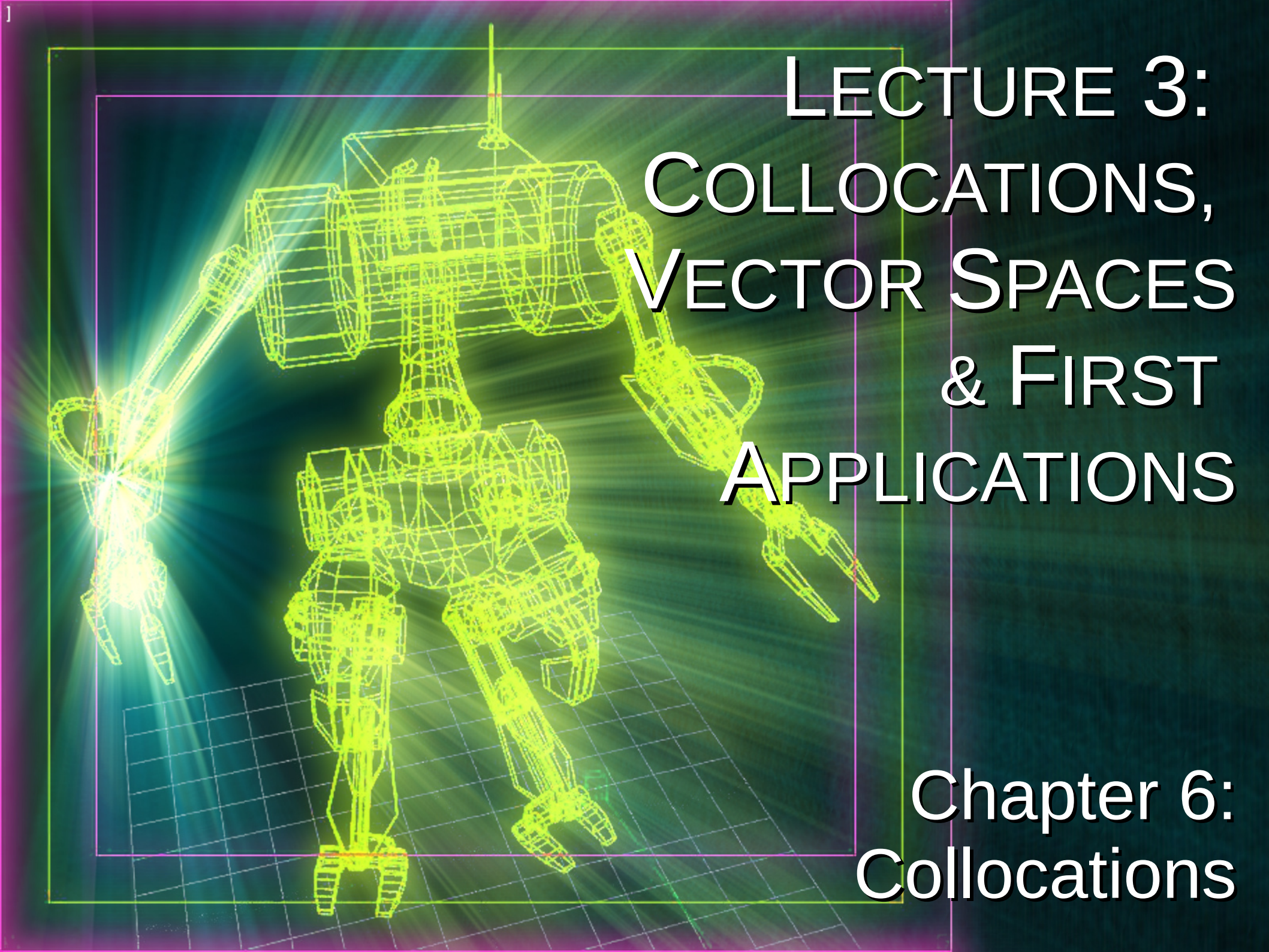
```
['cnn' 'telescope' 'linux' 'stereo' 'february' 'metal' 'effectively'
 'manufacture' 'todd' 'union' 'quantum' 'capture' 'global' 'baalke' 'era'
 'blind' 'pound' 'outlet' 'francis' 'destruction' 'french' 'barry'
 'albert' 'dry' 'athens' 'header' 'janet' 'warwick' 'pack' 'eight' 'bruin'
 'propaganda' 'dyer' 'behanna' 'brett' 'nelson' 'pool' 'promote'
 'offensive' 'hudson' 'chi' 'crazy' 'mt' 'wa' 'practical' 'fourth'
 'captain' 'combination' 'flat' 'firm' 'xview' 'constant' 'lucky' 'clh'
 'whenever' 'ra' 'favorite' 'sabre' 'fat' 'prime' 'patent' 'ethnic'
 'extreme' 'harvard' 'cub' 'rsa' 'gon' 'august' 'strnlght' 'cook' 'enable'
 'inning' 'dynamic' 'numerous' 'svga' 'advanced' 'vlb' 'gift' 'saint'
 'intelligent' 'bitnet' 'brief' 'kaldis' 'contradiction' 'strategy'
 'annoy' 'oracle' 'justification' 'clinical' 'conservative' 'rational'
 'smart' 'mainly' 'miracle' 'skin' 'royal' 'saturday' 'subscribe' 'dear']
[191. 191. 191. 191. 191. 191. 191. 191. 191. 191. 191. 191. 190. 190.
 190. 190. 190. 190. 190. 190. 190. 190. 189. 189. 189. 189. 189.
 189. 189. 189. 189. 189. 189. 189. 189. 188. 188. 188. 188.
 188. 188. 188. 188. 188. 187. 187. 187. 187. 187. 187. 187. 186. 186.
 186. 186. 186. 186. 186. 186. 186. 186. 186. 185. 185. 185. 185. 185.
 185. 185. 185. 184. 184. 184. 184. 184. 184. 184. 184. 184. 184. 184.
 184. 183. 183. 183. 183. 183. 183. 183. 183. 183. 183. 182. 182. 182.
 182.]
```

Pruning the vocabulary

- Frequency vs rank after pruning (all and top-500 words):







Lecture 3: Collocations, Vector Spaces & First Applications

Chapter 6: Collocations

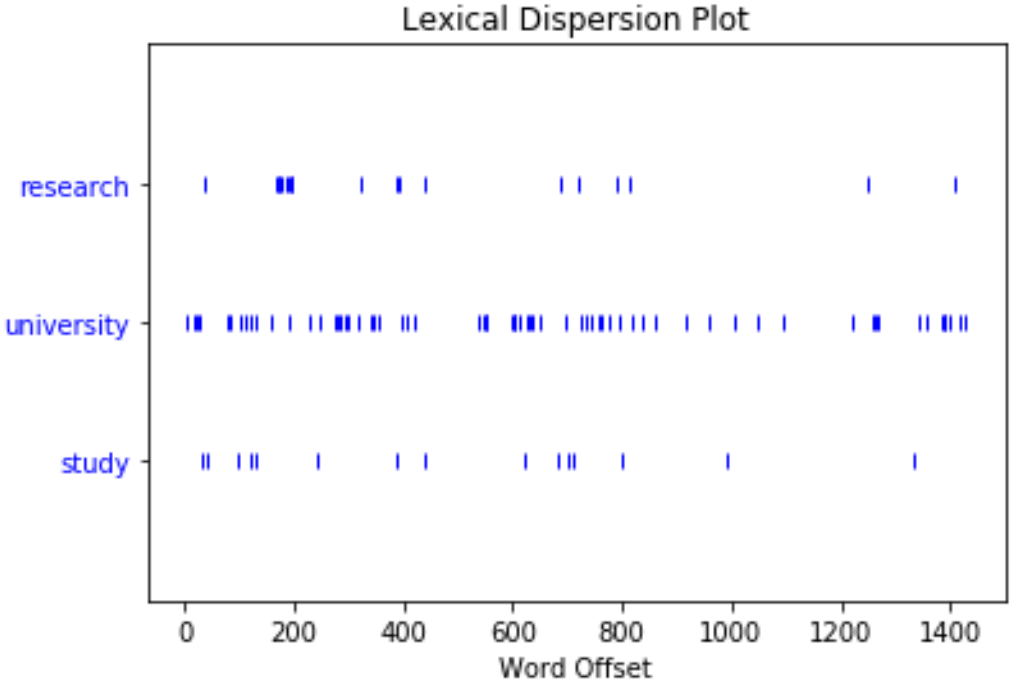
Dispersion over text

- NLTK's lexical dispersion plot allows inspecting where in a text a word occurs:

```
mycrawled_lemmatizedtexts[1].dispersion_plot(['research','university','stud
y'])
```

- In a research paper or news article, words near the beginning of the article might represent a summary, words near the middle might be details, and words near the end might be conclusions or implications

dispersion plot of words 'research', 'university', and 'study' on the front page of Tampere University



Collocations

- **Collocations** are expressions of at least two words
  - Noun phrases: *high horse*, *letter of recommendation*
  - Phrasal verbs: *to come up with*
  - Idiomatic expressions (stock phrases): *movers and shakers*, *water under the bridge*, *cutting corners*, *bite the bullet*, *under the weather*
    - *Usage is specific: miss the boat = be too late for something* (but not miss the plane, car, taxi etc.)
- Somewhat overlapping with **terminological phrases**: e.g. *machine learning*, *support vector machine*, etc.
- Collocations are not fully **compositional**: their meaning cannot be predicted from meanings of their parts.
  - E.g. *break the ice = become comfortable, get over initial unfamiliarity* does not involve ice, or breaking some object
  - Some cases are partly non-compositional, e.g. *best practices* has additional meaning beyond the plain meaning of its parts

Collocations

- Benefits of being aware of collocations:
  - Models that create natural language should be able to create collocations
  - Models for parsing natural language should recognize collocations
  - Methods that study statistical variation of language use in corpuses should be able to study variation in use of known collocations, and to identify previously unknown collocations

Collocations

- NLTK's **concordance view** allows inspecting collocations of an individual word:

```
mycrawled_lemmatizedtexts[0].concordance('research')
```

```
Displaying 14 of 14 matches:
xploratory data analysis ( smile ) research group . i be member of the tampere
group . i be member of the tampere research center in information and system a
of the probabilistic machine learn research group ; i be previously an academy
group ; i be previously an academy research fellow at the same department . i
ellence in computational inference research coin , helsinki institute for info
pascal2 network of excellence . my research interest include probabilistic gen
 letter , journal of machine learn research , neural network , pattern analysi
mining-driven analysis of previous research . journal of business research , 1
ous research . journal of business research , 106:46-59 , january 2020 . ( acc
ization . journal of machine learn research , 11:451-490 , 2010 . ( abstract ,
eding of digra 2020 , digital game research association conference , accept fo
ernational acm sigir conference on research and development in information ret
. ( pdf at publisher page ) recent research talk this be a partial list of rec
k this be a partial list of recent research talk i have give . jaakko peltonen
```

Identifying collocations

- Counting occurrence frequencies
  - If a set of words occurs together sufficiently frequently, they might denote a special meaning
  - However, naive counting might just discover e.g. word pairs that co-occur because of English syntax (*it is*, *I am* etc.)
  - A filter that only allows parts-of-speech that are likely to constitute phrases can help. Suggested filter (Justeson and Katz, 1995):
    "adjective noun", "noun noun", "adjective adjective noun", "adjective noun noun", "noun adjective noun", "noun noun noun", "noun preposition noun"

Identifying collocations

- Collocated words might appear with other words inbetween:
  - Did you hear the **bells ring**? (Nursery Rhymes of England)
  - ... that **rings** the **bell**. (Bunny Brown and his Sister Sue and Their Shetland Pony)
  - These are the **bells** that did not **ring** (The Bells of Nagasaki)
  - ... splendid **bells**, made expressly to **ring** (The Magic World)
  - the generators are required to **ring** all of the **bells** in the series (Cyclopedia of Telephony 1)
  - A **ring** at the front-door **bell** is heard. (The Bicyclers and Three Other Farces)
  - Then the White Bear gave her a silver **bell**, and when she wanted anything she had only to **ring** it (Fairy Ring)
- Despite this one could determine that ring is more likely to be associated with bell than e.g. tap the bell, clang the bell or carry the bell.

Identifying collocations

- One could compute the mean and variance of the position difference between two words (e.g. difference from each appearance of **bell** to the nearest appearance of **ring**)
  - Did you hear the **bells ring**? (*Nursery Rhymes of England*) diff: 1
  - ... that **rings** the **bell**. (*Bunny Brown and his Sister Sue and Their Shetland Pony*) diff: -2
  - These are the **bells** that did not **ring** (*The Bells of Nagasaki*) diff: 4
  - ... splendid **bells**, made expressly to **ring** (*The Magic World*) diff: 4
  - the generators are required to **ring** all of the **bells** in the series (*Cyclopedia of Telephony 1*) diff: 4
  - A **ring** at the front-door **bell** is heard. (*The Bicyclers and Three Other Farces*) diff: -4
  - Then the White Bear gave her a silver **bell**, and when she wanted anything she had only to **ring** it (*Fairy Ring*) diff: 10
- Words whose mean distance (mean absolute distance) is low are likely to occur close together
- This takes into account both how often words co-occur together how close they are placed when they co-occur

Identifying collocations

- Practicalities: One should only compute distance to nearest occurrence of a word pair.
  - For example: "**This** kind of situation **is** a perfect example of what kinds of consequences **this** decision **is** going to cause" should yield two this--is pairs, not four.
- Words can occur on both sides, e.g. "**is this** true" (this--is distance: -1) and "**this is** true" (this--is distance: 1). They should not sum to distance 0!
  - One could count occurrences in different order separately
  - Sometimes the difference in meaning is small ("Did he solve it? He did solve it. Solve it he did.")
  - Simplified version: count totals of absolute distances and signed distances separately
- One can restrict word-pair occurrences to a maximum distance (window size)

Identifying collocations

- In Python:first we need indices after vocabulary pruning from document texts to the new pruned vocabulary

```
#%% Get indices of documents to remaining words
oldtopruned=[]
tempind=-1
for k in range(len(unifiedvocabulary)):
    if pruningdecisions[k]==0:
        tempind=tempind+1
        oldtopruned.append(tempind)
    else:
        oldtopruned.append(-1)
#%% Create pruned texts
mycrawled_prunedtexts=[]
myindices_in_prunedvocabulary=[]
for k in range(len(mycrawled_lemmatizedtexts)):
    print(k)
    temp_newindices=[]
    temp_newdoc=[]
    for l in range(len(mycrawled_lemmatizedtexts[k])):
        temp_oldindex=myindices_in_unifiedvocabulary[k][l]
        temp_newindex=oldtopruned[temp_oldindex]
        if temp_newindex!=-1:
            temp_newindices.append(temp_newindex)
            temp_newdoc.append(unifiedvocabulary[temp_oldindex])
    mycrawled_prunedtexts.append(temp_newdoc)
    myindices_in_prunedvocabulary.append(temp_newindices)
```

Identifying collocations

- We will count numbers of distance-pair occurrences, sums of distances, absolute distances, and squares
- We need to use sparse matrices (storing only nonzero values) from the **scipy** library to avoid memory problems, because the vocabulary size (=number of rows and columns) is large
- It is also quite slow to compute in Python because for-loops take a lot of time - hours on my laptop for 20000 documents of 20 Newsgroups!

```
#%% Compute statistics of word distances
# Compute counts and subs of distances and squared distances
import scipy
distanceoccurrences=scipy.sparse.lil_matrix(\
    (len(remainingvocabulary),len(remainingvocabulary)))
sumdistances=scipy.sparse.lil_matrix(\
    (len(remainingvocabulary),len(remainingvocabulary)))
sumabsdistances=scipy.sparse.lil_matrix(\
    (len(remainingvocabulary),len(remainingvocabulary)))
sumdistancesquares=scipy.sparse.lil_matrix(\
    (len(remainingvocabulary),len(remainingvocabulary)))
```

Identifying collocations

- Efficient counting: instead of going through the texts separately for each word pair, go through texts only once, counting all pairs as they occur

```
for l in range(len(mycrawled_lemmatizedtexts)):
    latestoccurrencepositions=scipy.sparse.lil_matrix(\
        (len(remainingvocabulary),len(remainingvocabulary)))
    # Loop through all word positions m of document l
    for m in range(len(mycrawled_prunedtexts[l])):
        # Get the vocabulary index of the current word in position m
        currentword=myindices_in_prunedvocabulary[l][m]
        # Loop through previous words, counting back up to 10 words from current word
        windowsize=min(m,10)
        for n in range(windowsize):
            # Get the vocabulary index of the previous word in position m-n-1
            previousword=myindices_in_prunedvocabulary[l][m-n-1]
            # Is this the fist time we have encountered this word while
            # counting back from the word at m? Then it is the closest pair.
            if latestoccurrencepositions[currentword,previousword]<m:
                # Store the occurrence of this word pair with the word at m as the 1st word
                distanceoccurrences[currentword,previousword]=\
                    distanceoccurrences[currentword,previousword]+1
                sumdistances[currentword,previousword]=sumdistances[\
                    currentword,previousword]+((m-n-1)-m)
                sumabsdistances[currentword,previousword]=\
                    sumabsdistances[currentword,previousword]+abs((m-n-1)-m)
                sumdistancesquares[currentword,previousword]=\
                    sumdistancesquares[currentword,previousword]+((m-n-1)-m)**2
                # Store the occurrence of this word pair with the word at n as the 1st word
                distanceoccurrences[previousword,currentword]=\
                    distanceoccurrences[previousword,currentword]+1
                sumdistances[previousword,currentword]=sumdistances[\
                    previousword,currentword]+(m-(m-n-1))
                sumabsdistances[previousword,currentword]=\
                    sumabsdistances[currentword,previousword]+abs(m-(m-n-1))
                sumdistancesquares[previousword,currentword]=\
                    sumdistancesquares[previousword,currentword]+(m-(m-n-1))**2
                # Mark that we found this pair while counting down from m,
                # so we do not count more distant occurrences of the pair
                latestoccurrencepositions[currentword,previousword]=m
                latestoccurrencepositions[previousword,currentword]=m
```

Identifying collocations

- After counting we can derive the statistics (slow version):

```
# Compute distribution statistics based on the counts
n_vocab=len(remainingvocabulary)
distancemeans=scipy.sparse.lil_matrix((n_vocab,n_vocab))
absdistancemeans=scipy.sparse.lil_matrix((n_vocab,n_vocab))
distancevariances=scipy.sparse.lil_matrix((n_vocab,n_vocab))
absdistancevariances=scipy.sparse.lil_matrix((n_vocab,n_vocab))
for m in range(n_vocab):
    print(m)
    for n in range(n_vocab):
        n_occurrences=distanceoccurrences[m,n]
        if n_occurrences>1:
            # Estimate mean of m-n distance
            distancemeans[m,n]=sumdistances[m,n]/n_occurrences
            absdistancemeans[m,n]=sumabsdistances[m,n]/n_occurrences
            # Estimate variance of m-n distance
            distancevariances[m,n]=\
                sumdistancesquares[m,n]/(n_occurrences-1) \
                - (n_occurrences/(n_occurrences-1))\
                *(distancemeans[m,n]**2)
            absdistancevariances[m,n]=\
                sumdistancesquares[m,n]/(n_occurrences-1) \
                - (n_occurrences/(n_occurrences-1))\
                *(absdistancemeans[m,n]**2)
```

Identifying collocations

- After counting we can derive the statistics (faster version):

```
# Compute distribution statistics based on the counts
n_vocab=len(remainingvocabulary)
distancemeans=scipy.sparse.lil_matrix((n_vocab,n_vocab))
absdistancemeans=scipy.sparse.lil_matrix((n_vocab,n_vocab))
distancevariances=scipy.sparse.lil_matrix((n_vocab,n_vocab))
absdistancevariances=scipy.sparse.lil_matrix((n_vocab,n_vocab))
for m in range(n_vocab):
    print(m)
    # Find the column indices that have at least two occurrences
    tempindices=numpy.nonzero(distanceoccurrences[m,:]>1)[1]
    # The occurrence vector needs to be a non-sparse data type
    tempoccurrences=distanceoccurrences[m,tempindices].todense()
    # Estimate mean of m-n distance
    distancemeans[m,tempindices]=numpy.squeeze(\
        numpy.array(sumdistances[m,tempindices]/tempoccurrences))
    absdistancemeans[m,tempindices]=numpy.squeeze(\
        numpy.array(sumabsdistances[m,tempindices]/tempoccurrences))
    # Estimate variance of m-n distance
    meanterm=distancemeans[m,tempindices].todense()
    meanterm=numpy.multiply(meanterm,meanterm)
    meanterm=numpy.multiply(tempoccurrences/(tempoccurrences-1),meanterm)
    distancevariances[m,tempindices]=numpy.squeeze(\
        numpy.array(sumdistancesquares[m,tempindices]/(tempoccurrences-1) \
        - meanterm))
    meanterm=absdistancemeans[m,tempindices].todense()
    meanterm=numpy.multiply(meanterm,meanterm)
    meanterm=numpy.multiply(tempoccurrences/(tempoccurrences-1),meanterm)
    absdistancevariances[m,tempindices]=numpy.squeeze(\
        numpy.array(sumdistancesquares[m,tempindices]/(tempoccurrences-1) \
        - meanterm))
```

Identifying collocations

- After counting we can derive the statistics:

```
# Compute overall distance distribution
overalldistancecount=numpy.sum(distanceoccurrences)
overalldistancesum=numpy.sum(sumdistances)
overallabsdistancesum=numpy.sum(sumabsdistances)
overalldistancesquaresum=numpy.sum(sumdistancesquares)
overalldistancemean=overalldistancesum/
overalldistancecount
overallabsdistancemean=overallabsdistancesum/
overalldistancecount
overalldistancevariance=overalldistancesquaresum/
(overalldistancecount-1)\
    -overalldistancecount/(overalldistancecount-
1)*overalldistancemean
overallabsdistancevariance=overalldistancesquaresum/
(overalldistancecount-1)\
    -overalldistancecount/(overalldistancecount-
1)*overallabsdistancemean
```

Identifying collocations

- With the statistics, we can e.g. sort them and print most closely-occurring word pairs:

```
def findwordindex(wordstring):
    for k in range(len(remainingvocabulary)):
        if remainingvocabulary[k]==wordstring:
            return(k)
    return(-1)
# Find the chosen word and words that occurred with it at least 2 times
mywordindex=findwordindex('developer')
tempindices=numpy.nonzero(distanceoccurrences[mywordindex,:]>1)[1]
# Sort the pairs by lowest mean absolute distance
lowest_meandistances_indices=numpy.argsort(numpy.squeeze(numpy.array(\
    absdistancemeans[mywordindex,tempindices].todense())),axis=0)
# Print the top-50 lowest-distance pairs
for k in range(50):
    otherwordindex=tempindices[lowest_meandistances_indices[k]]
    # Print word pairs, absolute distances and distances (mean+std)
    print((remainingvocabulary[mywordindex],\
        remainingvocabulary[otherwordindex],\
        absdistancemeans[mywordindex,otherwordindex],\
        numpy.sqrt(absdistancevariances[\
        mywordindex,otherwordindex]),\
        distancemeans[mywordindex,otherwordindex],\
        numpy.sqrt(distancevariances[mywordindex,otherwordindex])))
```

Identifying collocations

- In order to check whether the mean distance of two specific words is "low"
  - the distribution of their distances should be compared to a more general distribution of distances between words
  - the comparison can be done by a statistical hypothesis test
  - in order to compare two distance distributions, we need to estimate also other statistics of the distributions: if we assume the distributions are Gaussian, the comparison can be done if we estimate their mean and variance/standard deviation
  - The distribution of distances for any specific pair of words can be compared against the general distribution of distances across all pairs of words. If the specific pair has a statistically significantly smaller distance than in general, we can call it a collocation.

Identifying collocations

- Two-sample unpaired t-test, test statistic:

test statistic (Behrens-Welch test statistic)

$$d = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

Once you have computed the statistic d and the degrees of freedom df, the p-value can be computed. In Python:

```
import scipy.stats
pvalue = 1 - stats.t.cdf(d,df=df)
```

degrees of freedom

$$df = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{(s_1^2/n_1)^2}{n_1 - 1} + \frac{(s_2^2/n_2)^2}{n_2 - 1}}$$

sample variance in data set 1

$$s_1^2 = \frac{1}{n_1 - 1} \sum_{i=1}^{n_1} (x_i - \bar{x}_1)^2$$

sample variance in data set 2

$$s_1^2 = \frac{1}{n_1 - 1} \sum_{i=1}^{n_1} (x_i - \bar{x}_1)^2$$

Identifying collocations

- **Computing p-values from statistics, Python code:**

```
#%% Compute t-test pvalues comparing abs distance distributions
absdistancepvalues=scipy.sparse.lil_matrix((n_vocab,n_vocab))

for m in range(n_vocab):
    # Find pairs of word m
    tempindices=numpy.nonzero(distanceoccurrences[m,:]>1)[1]
    # For computation we need to transform these to non-sparse vectors
    meanterm=absdistancemeans[m,tempindices].todense()
    varianceterm=absdistancevariances[m,tempindices].todense()
    occurrenceterm=distanceoccurrences[m,tempindices].todense()

    # Compute the t-test statistic for each pair
    tempstatistic=(meanterm-overallabsdistancemean)/ \
        numpy.sqrt(varianceterm/occurrenceterm+ \
        overallabsdistancevariance/overalldistancecount)
    # Compute the t-test degrees of freedom for each pair
    tempdf=(numpy.power(varianceterm/occurrenceterm+\
        overallabsdistancevariance/overalldistancecount,2))/ \
        ( (numpy.power(varianceterm/occurrenceterm,2))/(occurrenceterm-1)+ \
        ((overallabsdistancevariance/overalldistancecount)**2)/ \
        (overalldistancecount-1) )
    # Compute the t-test p-value for each pair
    temppvalue=scipy.stats.t.cdf(tempstatistic,tempdf)
    # Store the t-test p-value for each pair
    absdistancepvalues[m,tempindices]=numpy.squeeze(numpy.array(temppvalue))
```

Identifying collocations

- **Printing notable collocations (here: lowest p-values, with at least 10 occurrences), Python code:**

```
#%% Sort word pairs of a particular word by minimum mean absolute distance
def findwordindex(wordstring):
    for k in range(len(remainingvocabulary)):
        if remainingvocabulary[k]==wordstring:
            return(k)
    return(-1)

def printtopcollocations(wordstring):
    # Find the chosen word and words that occurred with it at least 2 times
    mywordindex=findwordindex(wordstring)
    if mywordindex==-1:
        print('Word not found: '+wordstring)
        return
    # Require at least 10 pair occurrences
    minpairoccurrences=10
    tempindices=numpy.nonzero(distanceoccurrences[mywordindex,:]>minpairoccurrences)[1]

    # Sort the pairs by lowest pvalue
    lowest_meandistances_indices=numpy.argsort(numpy.squeeze(\
        numpy.array(absdistancepvalues[mywordindex,tempindices].todense())),axis=0)
    # Print the top-50 lowest-distance pairs
    print('\nLowest p-values\n')
    for k in range(min(50,len(lowest_meandistances_indices))):
        otherwordindex=tempindices[lowest_meandistances_indices[k]]
        # Print the words, their absolute distances (mean+std) and distances (mean+std)
        print('{!s}--{!s}: {:d} occurrences, absdist: {:.1f} +- {:.1f}, offset: {:.1f} +- {:.1f}, pvalue: {:f}'.format(\
                remainingvocabulary[mywordindex],\
                remainingvocabulary[otherwordindex],\
                int(distanceoccurrences[mywordindex,otherwordindex]),\
                absdistancemeans[mywordindex,otherwordindex],\
                numpy.sqrt(absdistancevariances[mywordindex,otherwordindex]),\
                distancemeans[mywordindex,otherwordindex],\
                numpy.sqrt(distancevariances[mywordindex,otherwordindex]),\
                absdistancepvalues[mywordindex,otherwordindex]))
```

Identifying collocations

- **20 Newsgroups, collocations with 'linux': lowest p-values with at least 10 occurrences:**

linux--gifs: 19 occurrences, absdist: 2.6 +- 1.6, offset: -1.4 +- 2.8, pvalue: 0.000002
linux--linux: 146 occurrences, absdist: 4.0 +- 2.9, offset: 0.0 +- 5.0, pvalue: 0.000013
linux--ho: 15 occurrences, absdist: 3.0 +- 1.5, offset: -2.6 +- 2.1, pvalue: 0.000045
linux--kernel: 14 occurrences, absdist: 3.6 +- 1.4, offset: -0.4 +- 4.0, pvalue: 0.001138
linux--mach: 14 occurrences, absdist: 2.6 +- 2.9, offset: -1.5 +- 3.7, pvalue: 0.003926
linux--monochrome: 16 occurrences, absdist: 2.8 +- 3.0, offset: 0.8 +- 4.1, pvalue: 0.004776
linux--compiler: 14 occurrences, absdist: 4.1 +- 1.2, offset: 0.4 +- 4.4, pvalue: 0.006065
linux--xwindows: 11 occurrences, absdist: 2.9 +- 2.3, offset: 0.9 +- 3.7, pvalue: 0.006317
linux--gcc: 11 occurrences, absdist: 3.0 +- 2.5, offset: -0.8 +- 3.9, pvalue: 0.011365
linux--vgalib: 11 occurrences, absdist: 3.5 +- 2.0, offset: 1.4 +- 4.0, pvalue: 0.016419
linux--bsd: 24 occurrences, absdist: 4.0 +- 3.1, offset: -2.0 +- 4.7, pvalue: 0.056995
linux--recovery: 12 occurrences, absdist: 3.9 +- 3.1, offset: -3.2 +- 3.8, pvalue: 0.114691
linux--redistribute: 13 occurrences, absdist: 4.1 +- 3.0, offset: -2.8 +- 4.3, pvalue: 0.131025
linux--nextstep: 14 occurrences, absdist: 4.4 +- 2.3, offset: 2.1 +- 4.7, pvalue: 0.169185
linux--cview: 17 occurrences, absdist: 4.6 +- 2.3, offset: -4.6 +- 2.3, pvalue: 0.213928
linux--athena: 17 occurrences, absdist: 5.0 +- 2.9, offset: -2.1 +- 5.5, pvalue: 0.471118
linux--mgr: 13 occurrences, absdist: 5.2 +- 3.3, offset: -0.9 +- 6.3, pvalue: 0.576205
linux--svga: 22 occurrences, absdist: 5.5 +- 2.8, offset: 1.8 +- 5.9, pvalue: 0.748705
linux--dump: 12 occurrences, absdist: 5.9 +- 3.3, offset: -3.4 +- 6.0, pvalue: 0.807363
linux--ghhwang: 19 occurrences, absdist: 5.5 +- 2.3, offset: 5.4 +- 2.5, pvalue: 0.813153
linux--risc: 37 occurrences, absdist: 5.8 +- 3.7, offset: 4.6 +- 5.2, pvalue: 0.892655
linux--gum: 16 occurrences, absdist: 5.8 +- 2.2, offset: -5.8 +- 2.2, pvalue: 0.904095
linux--bryan: 11 occurrences, absdist: 6.5 +- 2.9, offset: -5.7 +- 4.2, pvalue: 0.931647
linux--naplps: 16 occurrences, absdist: 5.9 +- 2.2, offset: -1.3 +- 6.4, pvalue: 0.933202
linux--bambi: 11 occurrences, absdist: 6.7 +- 2.8, offset: -3.8 +- 6.4, pvalue: 0.962565
linux--accelerate: 11 occurrences, absdist: 6.6 +- 2.1, offset: -3.5 +- 6.3, pvalue: 0.984028
linux--sean: 16 occurrences, absdist: 6.8 +- 2.2, offset: -6.8 +- 2.2, pvalue: 0.996780
linux--woodworth: 12 occurrences, absdist: 8.1 +- 1.8, offset: -8.1 +- 1.8, pvalue: 0.999948

Identifying collocations

- **20 Newsgroups, collocations with 'strategy': lowest p-values with at least 10 occurrences:**

  strategy--tactic: 13 occurrences, absdist: 1.2 +- 0.6, offset: 0.7 +- 1.1, pvalue: 0.000000

  strategy--august: 18 occurrences, absdist: 4.5 +- 0.5, offset: -0.5 +- 4.6, pvalue: 0.000143

  strategy--veteran: 13 occurrences, absdist: 2.2 +- 2.3, offset: -1.3 +- 3.0, pvalue: 0.000438

  strategy--economics: 12 occurrences, absdist: 2.9 +- 2.1, offset: -0.9 +- 3.6, pvalue: 0.002457

  strategy--lp: 18 occurrences, absdist: 3.5 +- 2.6, offset: 2.5 +- 3.6, pvalue: 0.010177

  strategy--strategy: 62 occurrences, absdist: 5.5 +- 3.1, offset: 0.0 +- 6.3, pvalue: 0.865763

  strategy--republican: 11 occurrences, absdist: 6.2 +- 2.6, offset: 2.7 +- 6.4, pvalue: 0.912987

  strategy--presidential: 18 occurrences, absdist: 7.0 +- 2.1, offset: 7.0 +- 2.1, pvalue: 0.999554

  strategy--bylaw: 18 occurrences, absdist: 7.5 +- 2.6, offset: 7.5 +- 2.6, pvalue: 0.999574

  strategy--outreach: 11 occurrences, absdist: 8.5 +- 1.2, offset: -6.2 +- 6.3, pvalue: 0.999999

Identifying collocations

- **20 Newsgroups, collocations with 'strategy': lowest p-values with at least 10 occurrences:**

  mathematics--adelaide: 12 occurrences, absdist: 1.0 +- 0.0, offset: 1.0 +- 0.0, pvalue: 0.000000
  mathematics--applied: 16 occurrences, absdist: 1.0 +- 0.0, offset: -1.0 +- 0.0, pvalue: 0.000000
  mathematics--kempmp: 21 occurrences, absdist: 3.0 +- 0, offset: 3.0 +- 0, pvalue: 0.000000
  mathematics--pure: 12 occurrences, absdist: 2.0 +- 0.0, offset: -2.0 +- 0.0, pvalue: 0.000000
  mathematics--pihatie: 21 occurrences, absdist: 1.0 +- 0.0, offset: 1.0 +- 0.0, pvalue: 0.000000
  mathematics--petri: 23 occurrences, absdist: 2.1 +- 0.5, offset: -1.5 +- 1.6, pvalue: 0.000000
  mathematics--des: 18 occurrences, absdist: 1.5 +- 0.5, offset: -0.5 +- 1.5, pvalue: 0.000000
  mathematics--pihko: 22 occurrences, absdist: 1.2 +- 0.9, offset: -0.7 +- 1.3, pvalue: 0.000000
  mathematics--abstract: 13 occurrences, absdist: 1.4 +- 1.0, offset: -1.4 +- 1.0, pvalue: 0.000000
  mathematics--oulu: 23 occurrences, absdist: 2.6 +- 2.0, offset: 1.0 +- 3.2, pvalue: 0.000004
  mathematics--mathematics: 28 occurrences, absdist: 3.6 +- 2.6, offset: 0.0 +- 4.5, pvalue: 0.002588
  mathematics--discrete: 13 occurrences, absdist: 2.8 +- 2.8, offset: -1.2 +- 3.8, pvalue: 0.005887
  mathematics--askew: 16 occurrences, absdist: 4.7 +- 1.2, offset: -2.1 +- 4.5, pvalue: 0.131287
  mathematics--mathematical: 11 occurrences, absdist: 4.1 +- 3.1, offset: -1.9 +- 4.9, pvalue: 0.165169
  mathematics--methodology: 11 occurrences, absdist: 4.9 +- 2.4, offset: 0.0 +- 5.7, pvalue: 0.424917
  mathematics--jaskew: 16 occurrences, absdist: 5.2 +- 1.3, offset: -3.1 +- 4.5, pvalue: 0.656355
  mathematics--spam: 11 occurrences, absdist: 7.2 +- 0.8, offset: -7.2 +- 0.8, pvalue: 0.999999

Identifying collocations

- **20 Newsgroups, collocations with 'honor': lowest p-values with at least 10 occurrences:**

  honor--apologize: 13 occurrences, absdist: 2.4 +- 2.3, offset: 0.2 +- 3.3, pvalue: 0.000552

  honor--honor: 64 occurrences, absdist: 4.2 +- 2.4, offset: 0.0 +- 4.9, pvalue: 0.004860

  honor--esteem: 13 occurrences, absdist: 3.1 +- 2.4, offset: -2.5 +- 3.1, pvalue: 0.006324

  honor--husband: 12 occurrences, absdist: 3.4 +- 2.0, offset: 1.8 +- 3.7, pvalue: 0.008604

  honor--markbr: 12 occurrences, absdist: 4.7 +- 2.6, offset: -0.7 +- 5.5, pvalue: 0.309596

  honor--fame: 12 occurrences, absdist: 4.8 +- 2.8, offset: -0.8 +- 5.7, pvalue: 0.397042

  honor--wolfe: 14 occurrences, absdist: 5.3 +- 3.2, offset: -1.9 +- 6.0, pvalue: 0.606737

  honor--sabbath: 11 occurrences, absdist: 6.8 +- 3.0, offset: 1.4 +- 7.6, pvalue: 0.960609

Identifying collocations

- **20 Newsgroups, collocations with 'democracy': lowest p-values with at least 10 occurrences:**

democracy--democracy: 88 occurrences, absdist: 3.0 +- 2.1, offset: 0.0 +- 3.7, pvalue: 0.000000
democracy--pluralism: 21 occurrences, absdist: 2.0 +- 0.8, offset: 0.0 +- 2.2, pvalue: 0.000000
democracy--pluralistic: 28 occurrences, absdist: 2.5 +- 1.7, offset: -1.0 +- 2.9, pvalue: 0.000000
democracy--reaffirm: 14 occurrences, absdist: 1.9 +- 1.0, offset: 0.9 +- 2.0, pvalue: 0.000000
democracy--ideal: 22 occurrences, absdist: 2.9 +- 1.7, offset: -1.0 +- 3.3, pvalue: 0.000003
democracy--commitment: 30 occurrences, absdist: 3.6 +- 1.9, offset: -0.7 +- 4.0, pvalue: 0.000068
democracy--conviction: 18 occurrences, absdist: 3.1 +- 2.2, offset: 0.4 +- 3.8, pvalue: 0.000666
democracy--essential: 21 occurrences, absdist: 3.7 +- 2.6, offset: -3.0 +- 3.3, pvalue: 0.011023
democracy--elect: 15 occurrences, absdist: 3.9 +- 2.1, offset: -0.8 +- 4.4, pvalue: 0.023198
democracy--massive: 16 occurrences, absdist: 4.6 +- 2.0, offset: -2.7 +- 4.3, pvalue: 0.171778
democracy--dimension: 14 occurrences, absdist: 4.4 +- 2.5, offset: 2.4 +- 4.6, pvalue: 0.187232
democracy--copenhagen: 28 occurrences, absdist: 5.2 +- 3.2, offset: -5.2 +- 3.2, pvalue: 0.628684
democracy--allegation: 15 occurrences, absdist: 5.3 +- 2.4, offset: -5.3 +- 2.4, pvalue: 0.673325
democracy--cultural: 13 occurrences, absdist: 5.5 +- 1.7, offset: 1.9 +- 5.6, pvalue: 0.804235
democracy--invite: 15 occurrences, absdist: 6.1 +- 2.0, offset: -5.1 +- 3.9, pvalue: 0.964261
democracy--democratic: 40 occurrences, absdist: 6.5 +- 3.1, offset: 0.4 +- 7.2, pvalue: 0.996382
democracy--commoner: 14 occurrences, absdist: 7.0 +- 2.1, offset: -7.0 +- 2.1, pvalue: 0.998092
democracy--provision: 15 occurrences, absdist: 7.0 +- 2.1, offset: 7.0 +- 2.1, pvalue: 0.998511
democracy--vast: 12 occurrences, absdist: 6.5 +- 1.2, offset: -4.8 +- 4.7, pvalue: 0.999021
democracy--barry: 14 occurrences, absdist: 8.0 +- 2.1, offset: -8.0 +- 2.1, pvalue: 0.999930

Identifying collocations

- **20 Newsgroups, collocations with 'researcher': lowest p-values with at least 10 occurrences:**

  researcher--obesity: 12 occurrences, absdist: 1.5 +- 0.9, offset: 0.5 +- 1.7, pvalue: 0.000000

  researcher--subliminal: 12 occurrences, absdist: 3.7 +- 2.6, offset: -1.3 +- 4.4, pvalue: 0.046366

  researcher--flash: 16 occurrences, absdist: 4.1 +- 2.6, offset: -2.1 +- 4.5, pvalue: 0.075936

  researcher--hiv: 42 occurrences, absdist: 4.7 +- 3.1, offset: 1.9 +- 5.3, pvalue: 0.210442

  researcher--hicnet: 13 occurrences, absdist: 4.5 +- 3.2, offset: -3.2 +- 4.6, pvalue: 0.284178

  researcher--infected: 11 occurrences, absdist: 5.0 +- 2.6, offset: -3.4 +- 4.7, pvalue: 0.474643

  researcher--researcher: 50 occurrences, absdist: 5.2 +- 2.7, offset: 0.0 +- 5.9, pvalue: 0.611248

  researcher--rebound: 12 occurrences, absdist: 5.3 +- 2.1, offset: 1.3 +- 5.8, pvalue: 0.677744

  researcher--retrieve: 11 occurrences, absdist: 6.1 +- 2.9, offset: 1.2 +- 6.9, pvalue: 0.867973

Identifying collocations

- **20 Newsgroups, collocations with 'horse': lowest p-values with at least 10 occurrences:**

  horse--rein: 29 occurrences, absdist: 2.4 +- 1.7, offset: -0.4 +- 2.9, pvalue: 0.000000

  horse--horse: 68 occurrences, absdist: 3.4 +- 2.3, offset: 0.0 +- 4.2, pvalue: 0.000000

  horse--neck: 21 occurrences, absdist: 2.6 +- 2.5, offset: 0.0 +- 3.6, pvalue: 0.000101

  horse--steer: 16 occurrences, absdist: 3.2 +- 3.0, offset: -1.8 +- 4.1, pvalue: 0.014349

  horse--app: 13 occurrences, absdist: 3.5 +- 3.4, offset: 2.5 +- 4.3, pvalue: 0.068916

  horse--mining: 15 occurrences, absdist: 4.1 +- 2.3, offset: 1.9 +- 4.5, pvalue: 0.071825

  horse--egreen: 19 occurrences, absdist: 5.8 +- 2.7, offset: -2.1 +- 6.2, pvalue: 0.877384

  horse--cruncher: 17 occurrences, absdist: 6.7 +- 2.8, offset: -2.8 +- 6.9, pvalue: 0.986531

Identifying collocations

- **20 Newsgroups, collocations with 'global': lowest p-values with at least 10 occurrences:**

  global--dynamic: 11 occurrences, absdist: 1.5 +- 0.7, offset: 0.5 +- 1.7, pvalue: 0.000000

  global--warming: 13 occurrences, absdist: 2.2 +- 2.6, offset: 0.2 +- 3.5, pvalue: 0.001088

  global--climate: 15 occurrences, absdist: 3.1 +- 2.9, offset: 0.1 +- 4.4, pvalue: 0.012272

  global--growth: 12 occurrences, absdist: 3.7 +- 2.4, offset: 0.0 +- 4.5, pvalue: 0.034828

  global--global: 60 occurrences, absdist: 4.5 +- 3.0, offset: 0.0 +- 5.5, pvalue: 0.091471

  global--booster: 13 occurrences, absdist: 4.7 +- 2.8, offset: 0.5 +- 5.6, pvalue: 0.323183

  global--martian: 11 occurrences, absdist: 5.9 +- 2.1, offset: -1.4 +- 6.4, pvalue: 0.900191

  global--titan: 12 occurrences, absdist: 6.3 +- 3.1, offset: -4.0 +- 6.0, pvalue: 0.911109

  global--infrastructure: 19 occurrences, absdist: 6.7 +- 2.8, offset: 4.1 +- 6.1, pvalue: 0.989592

In brief: regular expressions

- **Regular expressions** are a language of specifying searches in text for matches of desired terms, allowing constraints on where the matches can occur and what the surrounding text must be.
  - They are occasionally useful e.g. for extracting complicated collocations as features from text: for example "for X years" where X is any number
- Examples of regular expression patterns:
  - where match occurrences of 'where' with this exact capitalization. Can be part of words like 'somewhere' or 'whereupon'
  - [Ww]here match occurrences of 'where' or 'Where'. Can be part of words.
  - \\b[Ww]here\\b must have a word boundary (double backslash needed in Python)
  - [^a-zA-Z][Ww]here[a-zA-Z] cannot have a letter on the left side, must have a letter on the right side

In brief: regular expressions

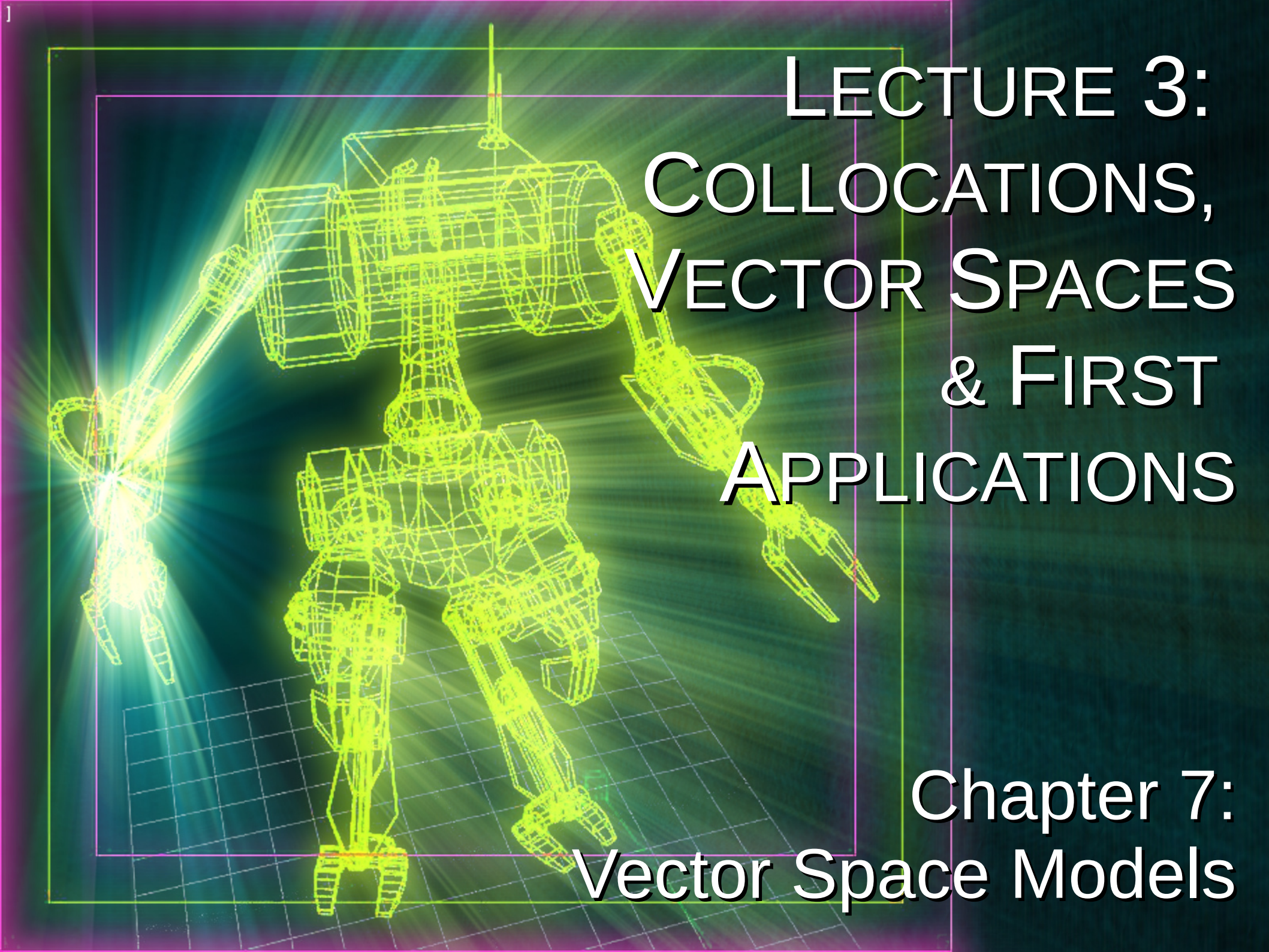
- In Python: regular expressions can be searched from text strings using the **re** library

```
import re
# Define the sentence
sentence='For 20 years I have worked for 50 others, and
I am sick of it, and for this reason, from today onward
I work only for 1 person, myself.'
# Define and compile the pattern
pattern='[Ff]or [0-9]'
pattern=re.compile(pattern)
# Find all matches and print each match and its span
# (start and end character indices of the match)
allmatches=re.finditer(pattern,sentence)
for tempmatch in allmatches:
    print(tempmatch.group(),tempmatch.span())
Out:
For 2 (0, 5)
for 5 (27, 32)
for 1 (114, 119)
```

In brief: regular expressions

- Another example:

```
import re
# Define the sentence
sentence='I want to go somewhere where the usual
things I see everyday are nowhere to be seen.'
# Define and compile the pattern
pattern='\\b[Ww]here\\b'
pattern=re.compile(pattern)
# Find all matches and print each match and its
span
# (start and end character indices of the match)
allmatches=re.finditer(pattern,sentence)
for tempmatch in allmatches:
    print(tempmatch.group(),tempmatch.span())
Out:
where (23, 28)
```



Lecture 3: Collocations, Vector Spaces & First Applications

Chapter 7: Vector Space Models

Document representation

- When we are interested in analyzing a (large) set of documents (a corpus) by statistical and computational methods, we need some way of **representing the content of each document**
  - The documents of course represent themselves perfectly, but it is not straightforward to compare two complex documents
  - The representation should be compact, so that it summarizes the content of the document for computational purposes. (The summary does not need to be human-readable language.)
  - The representation should allow comparison of similarities and differences between documents
  - The representations are usually only an overall level description that leaves out almost all detail: it is not possible to construct the original document content from such a representation.
- Document representations are often created as **vectors of feature values**: such vectors live in a **vector space**

Vector space models

- The vector space model is widely used for tasks like document classification, document clustering, and information retrieval
- It is conceptually very simple: essentially create various features about document content
  - Often done by counting number of appearance of different terms: each feature is the count of a particular term.
  - Simplest: count number of occurrences of individual words
  - More complicated: count also number of identified meaningful collocations
- Documents having similar vectors are supposed to be semantically similar

Vector space models

- If we have two documents in a D-dimensional vector space, with vectors $\boldsymbol{x}_1, \boldsymbol{x}_2$, their **dissimilarity** can be computed for example by their **Euclidean distance**

$$d(\boldsymbol{x}_1, \boldsymbol{x}_2) = \|\boldsymbol{x}_1 - \boldsymbol{x}_2\| = \sqrt{\sum_{d=1}^{D} (x_{1,d} - x_{2,d})^2}$$

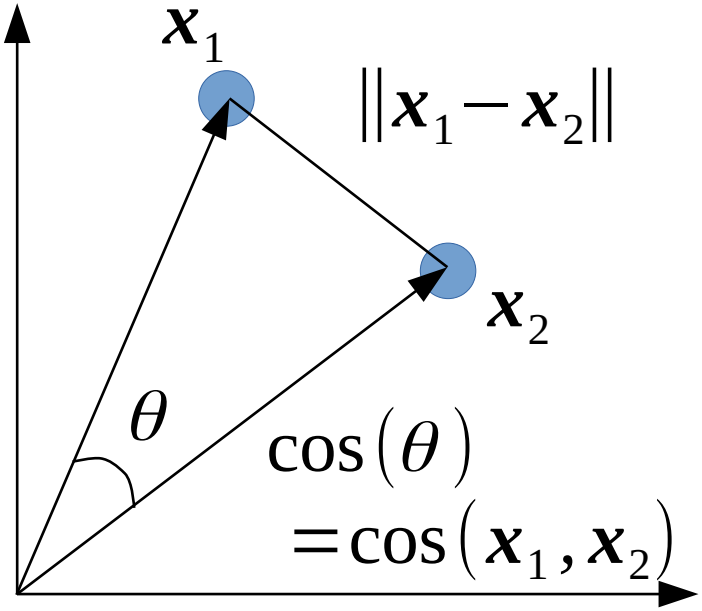
- The similarity of the documents can be computed for example by the **angle** between their vectors, or more conveniently by the cosine of the angle (**cosine similarity**):

$$\cos(\boldsymbol{x}_1, \boldsymbol{x}_2) = \frac{\boldsymbol{x}_1^T \boldsymbol{x}_2}{\|\boldsymbol{x}_1\| \cdot \|\boldsymbol{x}_2\|} = \frac{\sum_{d=1}^{D} x_{1,d} x_{2,d}}{\sqrt{\sum_{d=1}^{D} x_{1,d}^2} \cdot \sqrt{\sum_{d=1}^{D} x_{2,d}^2}}$$

Vector space models

- Euclidean distance and cosine similarity are related to one another:

$$d^2(\boldsymbol{x}_1,\boldsymbol{x}_2)=\|\boldsymbol{x}_1-\boldsymbol{x}_2\|^2$$
$$=\|\boldsymbol{x}_1\|^2+\|\boldsymbol{x}_2\|^2-2\,\boldsymbol{x}_1^T\boldsymbol{x}_2$$
$$=\|\boldsymbol{x}_1\|^2+\|\boldsymbol{x}_2\|^2-2\|\boldsymbol{x}_1\|\cdot\|\boldsymbol{x}_2\|\cdot\cos(\boldsymbol{x}_1,\boldsymbol{x}_2)$$



- In particular, if the document vectors have been normalized to be length 1:

$$d^2(\boldsymbol{x}_1,\boldsymbol{x}_2)=2-2\cdot\cos(\boldsymbol{x}_1,\boldsymbol{x}_2)=2-2\,\boldsymbol{x}_1^T\boldsymbol{x}_2$$

  then bigger cosine means smaller distance and vice versa.

- Cosine (inner product) uses the origin (zero vector) as a point of reference for the angle
  - Origin is often meaningful as a reference, but sometimes it is useful to use a different reference point (e.g. center of a document corpus)

Vector space models

- Distances and similarities in vector space models only consider the features extracted from text documents, but not their order of appearance in the document.
- **Edit distance** is a distance measure between two strings of text that is not based on a vector-space representation
  - it counts numbers of editing operations (insertions, deletions, changes) needed to transform one string into the other
  - More about that later

Other document features

- **Length** of the document:
  - Number of pages
  - Number of chapters, sections and subsections
  - Number of paragraphs
  - Number of sentences
  - Number of words
  - Number of characters
- **Metadata**:
  - List of authors, affiliations, addresses, emails, websites, account IDs, contribution types of different authors
  - Date of creation, date of last file modification, date of submission, date of revision, date of publication, date of last access
  - Venue of publication (publisher, website, channel, book/journal, issue, pages, URL, DOI)

Other document features

- **Connectivity**:
  - Number of outgoing hyperlinks
  - Number of incoming hyperlinks
  - Number of tagged persons in a tweet, number of hashtags
  - Number of citations to other publications
  - Number of citations from other publications to this one
  - Number of supplementary materials
  - Number of other-media versions of the same document (video, audiobook, PDF, HTML)
  - Number of direct parents (or ancestor documents) in a thread of replies
  - Number of replies to (or descendants of) this document

Other document features

- **Popularity**:
  - Number of views/reads
  - Number of shares/retweets/recommendations
  - Number of purchases
  - Status as trending, growth over time
- **Sentiment** - derived feature of document content
  - Positivity/negativity
  - Sadness, fear, anger, disgust, awe, horror, love, lust, etc.
  - Imperativity, stating, questioning
  - Agreement/disagreement, rebuttal
  - Review score if the document is a review
- **Reception**:
  - Number of upvotes and downvotes
  - Review ratings of the document
  - Sentiment of replies

Document representation

- **Representation 1: binary vector.** For each term v in a vocabulary of V terms, check whether the word occurs in document i

$$\boldsymbol{x}_i = [x_{i,1}, \ldots, x_{i,V}]$$
$$= [1('aardvark' \in document\ i), \ldots, 1('zebra' \in document\ i)]$$

- **Representation 2: count vector.** For each term v in a vocabulary of V terms, count how many times the word occurs in document i

$$\boldsymbol{x}_i = [x_{i,1}, \ldots, x_{i,V}]$$
$$= [count('aardvark' \in document\ i), \ldots, count('zebra' \in document\ i)]$$

Weighting and normalization

- In document vectors, features with the biggest values tend to affect distances and cosine similarities the most.
- Longer documents will have vectors with greater counts.
- Considering only the counts of a term (word) in a document itself does not take into account the distribution of the term in a collection
  - If the word is common in all documents, it is not surprising that it occurs a lot in the current document
- To improve the performance of document representation in different tasks, it is typical to adjust document vectors by
  - **weighting** of the features: multiplying feature values by feature importances.
  - **normalization of the features**: trying to make all features have a similar range/amount of variation over a collection
  - **normalization** of the document vectors

Weighting and normalization

- **Feature normalization 1: per-feature z-score.** The z-score is compares a feature value to its distribution in a collection: it is the number of standard deviations the value is higher/lower than the mean value.

$$z_{iv} = \frac{x_{iv} - \mu_v}{\sigma_v}$$

where $\mu_v = (1/T) \sum_{i=1}^{T} x_{iv}$ is the mean of feature v

and $\sigma_v = \sqrt{\frac{1}{T-1} \sum_{i=1}^{T} (x_{iv} - \mu_v)^2}$ is the standard deviation of feature v in the collection

Weighting and normalization

- **Document normalization 1: unit vector norm.** This normalization makes the document vector have length (norm) 1.

$$\boldsymbol{x}_i' = \frac{\boldsymbol{x}_i}{\|\boldsymbol{x}_i\|} = \left[\frac{x_{i1}}{\sqrt{\sum_{v=1}^{V} x_{iv}^2}}, \ldots, \frac{x_{iV}}{\sqrt{\sum_{v=1}^{V} x_{iv}^2}} \right]$$

- **Document normalization 2: proportional counts.** If the feature values are counts of terms, this normalization changes them to proportions of all terms in the document.

$$\boldsymbol{x}_i' = \frac{\boldsymbol{x}_i}{\sum_{v=1}^{V} x_{iv}} = \left[\frac{x_{i1}}{\sum_{v=1}^{V} x_{iv}}, \ldots, \frac{x_{iV}}{\sum_{v=1}^{V} x_{iv}} \right]$$

TF-IDF

- Considering only the counts of a term (word) in a document itself does not take into account the distribution of the term in a collection
  - If the word is common in all documents, it is not surprising that it occurs a lot in the current document
- **Term frequency - inverse document frequency** (**TF-IDF**) is a weighting method popular in natural language processing and information retrieval
  - TF-IDF is not just term weighting, but a more general transformation of features
  - Idea: upweight terms that appear a lot in the current document, but downweight terms that appear in a large part of the document collection

TF-IDF

- TF-IDF formula:

$$\boldsymbol{x}_i' = \left[tf(1,i) \cdot idf(1), \ldots, tf(V,i) \cdot idf(V) \right]$$

  for each feature (term) v, compute the product of
  - a term frequency value $tf(v,i)$ of the term in document i
  - an inverse document frequency value $idf(v)$ of the term in the collection
- There are multiple variant equations for the tf and idf values
- Term frequency variants:
  1. Boolean: $tf(v,i) = \delta(v \in document\ i)$ one if v is in i, zero otherwise
  2. Raw count: $tf(v,i) = count(v \in document\ i)$
  3. Length-normalized frequency: $tf(v,i) = \frac{count(v \in document\ i)}{number\ of\ terms \in i}$
  4. Logarithm of the count: $tf(v,i) = \log\left(1 + count(v \in document\ i)\right)$
  5. Count relative to most frequent term: $tf(v,i) = \alpha + (1-\alpha)\frac{count(v \in document\ i)}{max_{u=1}^{V} count(u \in document\ i)}$

TF-IDF

- Inverse document frequency variants:
  1. Unary/boolean: $idf(v) = 1$ for all v in the collection
  2. Logarithmic inverse document frequency: $idf(v) = \log\left(\frac{T}{\sum_{i=1}^{T} \delta(v \in document\ i)}\right)$
  3. Smoothed version to avoid zero and infinite values: $idf(v) = 1 + \log\left(\frac{T}{1 + \sum_{i=1}^{T} \delta(v \in document\ i)}\right)$
  4. Version proportional to most common term: $idf(v) = \log\left(\frac{max_{u=1}^{V} \sum_{i=1}^{T} \delta(u \in document\ i)}{1 + \sum_{i=1}^{T} \delta(v \in document\ i)}\right)$
  5. Version proportional to documents without the term $idf(v) = \log\left(\frac{T - \sum_{i=1}^{T} \delta(v \in document\ i)}{\sum_{i=1}^{T} \delta(v \in document\ i)}\right)$

TF-IDF

- In Python:

```
#%% Create TF-IDF vectors
n_docs=len(mycrawled_prunedtexts)
n_vocab=len(remainingvocabulary)
# Matrix of term frequencies
tfmatrix=scipy.sparse.lil_matrix((n_docs,n_vocab))
# Row vector of document frequencies
dfvector=scipy.sparse.lil_matrix((1,n_vocab))
# Loop over documents
for k in range(n_docs):
    # Row vector of which words occurred in this document
    temp_dfvector=scipy.sparse.lil_matrix((1,n_vocab))
    # Loop over words
    for l in range(len(mycrawled_prunedtexts[k])):
        # Add current word to term-frequency count and document-count
        currentword=myindices_in_prunedvocabulary[k][l]
        tfmatrix[k,currentword]=tfmatrix[k,currentword]+1
        temp_dfvector[0,currentword]=1
    # Add which words occurred in this document to overall document counts
    dfvector=dfvector+temp_dfvector
# Use the count statistics to compute the tf-idf matrix
tfidfmatrix=scipy.sparse.lil_matrix((n_docs,n_vocab))
# Let's use raw term count, and smoothed logarithmic idf
idfvector=numpy.log(1+((dfvector+1)**-1)*n_docs)
for k in range(n_docs):
    # Combine the tf and idf terms
    tfidfmatrix[k,:]=tfmatrix[k,:]*idfvector
```