Text classification

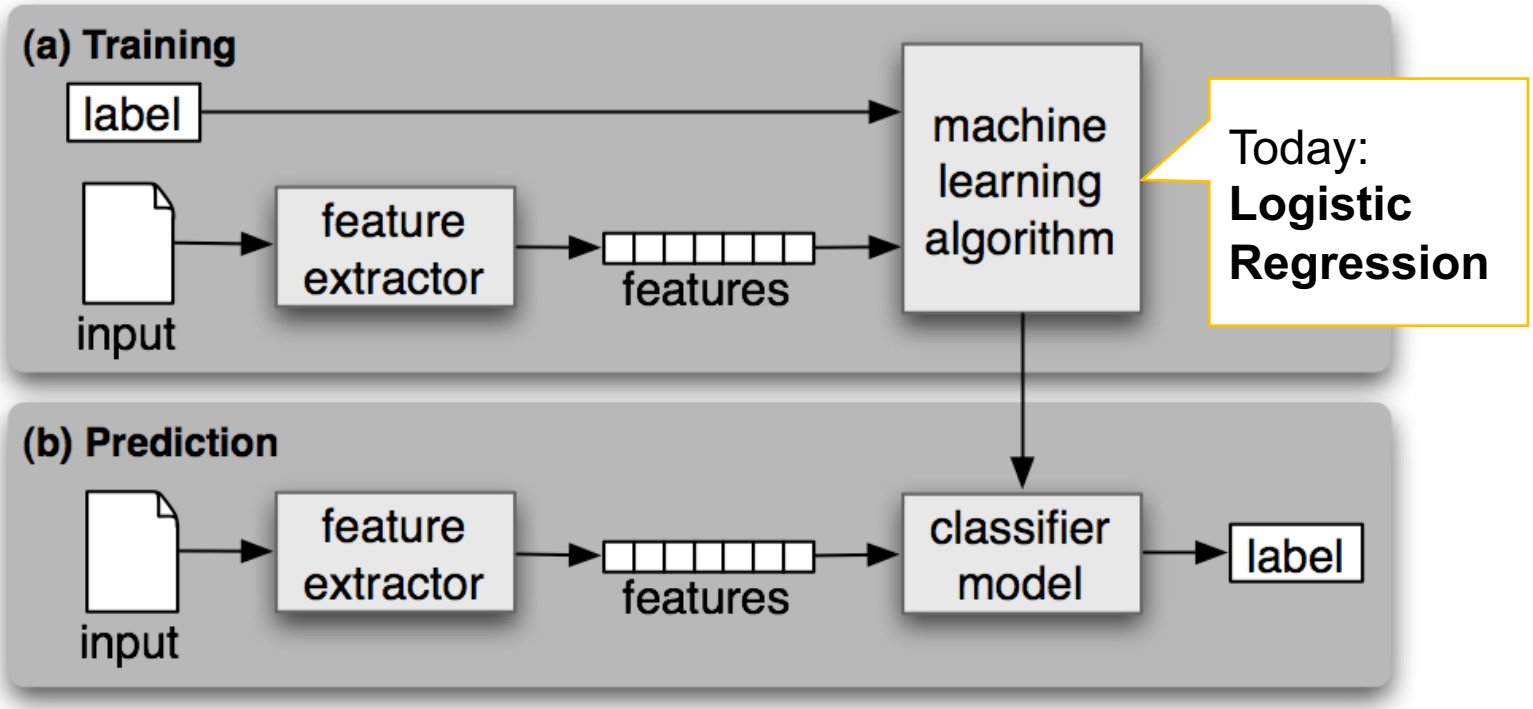
IN4080
Natural Language Processing

Yves Scherrer

Logistic regression

also known as Maximum Entropy Classifier

Supervised classification



Supervised classification

Take an instance $x \in X$ and associate it with a label $y \in Y$.

- General prediction function:

$$\hat{y} = \arg\max_{y \in Y} S(x, y)$$

  - $\hat{y}$: predicted class according to a model
  - S is some scoring function and depends on the type of classification algorithm

- Prediction function for probabilistic models:

$$\hat{y} = \arg\max_{y \in Y} P(y|x)$$

Can we estimate this probability directly, without applying Bayes' theorem?

Logistic regression

Agenda:

- Describe the instances as feature vectors
- Define the prediction function
- Define the training process

Notation:

- The ith instance/document: $x_i \in X$
- The jth label/class: $y_j \in Y$
- The feature vector corresponding to x_i: $\boldsymbol{f}^{(i)}$
- The weight vector corresponding to y_j: $\boldsymbol{w}^{(j)}$

Bag-of-words feature vectors

Each document x_i is represented by a vector $\boldsymbol{f}^{(i)}$ of word occurrences.

Documents	
x_1	it was pathetic the worst part was the boxing scenes
x_2	no plot twists or great scenes
x_3	and satire and great plot twists
x_4	great scenes great film

	$\boldsymbol{f}^{(1)}$	$\boldsymbol{f}^{(2)}$	$\boldsymbol{f}^{(3)}$	$\boldsymbol{f}^{(4)}$
and	0	0	2	0
boxing	1	0	0	0
film	0	0	0	1
great	0	1	1	2
it	1	0	0	0
no	0	1	0	0
or	0	1	0	0
part	1	0	0	0
pathetic	1	0	0	0
plot	0	1	1	0
…				

Bag-of-words feature vectors

Each document x_i is represented by a vector $\boldsymbol{f}^{(i)}$ of word occurrences.

	$\boldsymbol{f}^{(1)}$	$\boldsymbol{f}^{(2)}$	$\boldsymbol{f}^{(3)}$	$\boldsymbol{f}^{(4)}$
and	0	0	2	0
boxing	1	0	0	0
film	0	0	0	1
great	0	1	1	2
it	1	0	0	0
no	0	1	0	0
or	0	1	0	0
part	1	0	0	0
pathetic	1	0	0	0
plot	0	1	1	0
…				

Each row represents a **feature**. In a bag-of-words model, the features are individual words.

This example uses **count features**, but they may also be **binary** (presence or absence).

Weight vectors

Each class y_j is represented by a **weight vector** $\boldsymbol{w}^{(j)}$ that tells us how indicative of that class each word is.

Classes
y_{pos}
y_{neg}

These weight values are fictive. We'll see later how exactly they're learned.

	w^{pos}	w^{neg}
and	0.3	1.2
boxing	0.4	2.4
film	-0.2	-1.5
great	1.8	-0.7
it	1.1	0.1
no	-0.7	0.6
or	-1.9	-0.3
part	2.8	0.4
pathetic	1.2	3.1
plot	0.2	0.3
…		

Prediction scores

For each combination of document x_i and class y_j, we can compute a prediction score.

- Weighted sum of feature counts:

$$S(x_i, y_j) = \sum_{k=0}^{n} f_k^{(i)} \cdot w_k^{(j)}$$

This corresponds to the **dot product**:

$$S(x_i, y_j) = \boldsymbol{f}^{(i)} \cdot \boldsymbol{w}^{(j)}$$

Prediction scores

	$f^{(1)}$	$f^{(2)}$	$f^{(3)}$	$f^{(4)}$
and	0	0	2	0
boxing	1	0	0	0
film	0	0	0	1
great	0	1	1	2
it	1	0	0	0
no	0	1	0	0
or	0	1	0	0
part	1	0	0	0
pathetic	1	0	0	0
plot	0	1	1	0
…				

$$S(x_1, y_{pos}) = 5.5$$
$$S(x_1, y_{neg}) = 6.0$$

$$S(x_2, y_{pos}) = -0.6$$
$$S(x_2, y_{neg}) = -0.1$$

$$S(x_3, y_{pos}) = 2.6$$
$$S(x_3, y_{neg}) = 2.0$$

$$S(x_4, y_{pos}) = 3.4$$
$$S(x_4, y_{neg}) = -2.9$$

	w^{pos}	w^{neg}
and	0.3	1.2
boxing	0.4	2.4
film	-0.2	-1.5
great	1.8	-0.7
it	1.1	0.1
no	-0.7	0.6
or	-1.9	-0.3
part	2.8	0.4
pathetic	1.2	3.1
plot	0.2	0.3
…		

These scores don't look like probabilities…

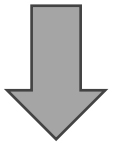
From scores to probabilities

How can we convert a tuple of numeric scores into a probability distribution?

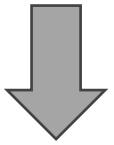
1. Make sure each score is 0 or positive
   - Exponentiation
   - $\exp(x) = e^x = 2.7183^x$
2. Make sure that the sum of scores is 1
   - Normalization

This operation is called **softmax**.

$$[s_1, s_2, \dots, s_n]$$



$$[\exp(s_1), \exp(s_2), \dots, \exp(s_n)]$$



$$\left[\frac{\exp(s_1)}{\sum_{i=1}^{n}\exp(s_i)}, \frac{\exp(s_2)}{\sum_{i=1}^{n}\exp(s_i)}, \dots, \frac{\exp(s_n)}{\sum_{i=1}^{n}\exp(s_i)}\right]$$

Prediction probabilities

- Document x_1:
  - $[5.5, 6.0] \Longrightarrow [245, 403] \Longrightarrow [0.38, 0.62]$
- Document x_2:
  - $[-0.6, -0.1] \Longrightarrow [0.55, 0.91] \Longrightarrow [0.38, 0.62]$
- Document x_3:
  - $[2.6, 2.0] \Longrightarrow [13.5, 7.4] \Longrightarrow [0.65, 0.35]$
- Document x_4:
  - $[3.4, -2.9] \Longrightarrow [29.96, 0.06] \Longrightarrow [0.998, 0.002]$

Logistic regression

Done:

- Describe the instances as feature vectors
  - Relatively simple with bag-of-words features
- Define the prediction function
  - $P(y_j|x_i) = \underset{y^{(k)} \in Y}{\text{softmax}}(\boldsymbol{f}^{(i)} \cdot \boldsymbol{w}^{(j)}) = \frac{\exp(\boldsymbol{f}^{(i)} \cdot \boldsymbol{w}^{(j)})}{\sum_{y^{(k)} \in Y} \exp(\boldsymbol{f}^{(i)} \cdot \boldsymbol{w}^{(k)})}$

To do:

- Define the training process

Logistic regression training

Typically, a LR model is trained one instance at a time:

- Take one instance x_i of the training set
- Predict label $\hat{y}$ (and their probabilities) using the current model
  - Model = set of weight vectors
  - Typically, all weight vectors are initialized to 0
- Modify the parameters of the model
  - Taking into account the probability of the predicted label
- Continue “until tired” (J. Eisenstein)

Logistic regression training

- The training process starts with a default model, which is then continuously adjusted and improved
- There is no natural end point
  - You may want to see every training instance at least once (one **epoch**), but you're not required to
  - There are heuristics to figure out when it is a good moment to stop
- Results will vary depending on the model initialization and the order of presenting the instances

Weight vector updates

Extreme cases in a two-class setting:

- Prediction is correct ($\hat{y} = y$) with $P(y|x_k) = 1$
  - This is perfect, don't change anything!
- Prediction is wrong ($\hat{y} \neq y$) with $P(y|x_k) = 0$
  - This is the worst possible outcome!
  - The weight vector of the correct label y is updated by adding the feature values:

$$\boldsymbol{w}^{(y)} \leftarrow \boldsymbol{w}^{(y)} + \boldsymbol{f}^{(k)}$$

  - The weight vector of the predicted (but wrong) label $\hat{y}$ is updated by subtracting the feature values:

$$w^{(\hat{y})} \leftarrow w^{(\hat{y})} - \boldsymbol{f}^{(k)}$$

Weight vector updates

Generalization:

- Update weight vector of correct class y:

$$\boldsymbol{w}^{(y)} \leftarrow \boldsymbol{w}^{(y)} + (1 - P(y|x_k)) \cdot \boldsymbol{f}^{(k)}$$

Perfect case: $1 - 1 = 0$

Worst case: $1 - 0 = 1$

- Update weight vectors of all other classes $z \neq y$:

$$\boldsymbol{w}^{(z)} \leftarrow \boldsymbol{w}^{(z)} - P(z|x_k) \cdot \boldsymbol{f}^{(k)}$$

Perfect case: 0

Worst case: 1 for $z = \hat{y}$, 0 for $z \neq \hat{y}$

Logistic regression update rule

- We always update parameters for all classes
  - Except in specific cases where $P(y|x_k) = 1$
- If the classifier assigns a high probability to the correct class y, we update $\boldsymbol{w}^{(y)}$ only a little
  - We also update the other weight vectors only a little
- If the classifier assigns a high probability to an incorrect class z, we strongly update $\boldsymbol{w}^{(z)}$
  - We also strongly update $\boldsymbol{w}^{(y)}$

Logistic regression update rule

The update intensity can be further modulated with an additional parameter λ, the **learning rate**:

- Update weight vector of correct class y:

$$\boldsymbol{w}^{(y)} \leftarrow \boldsymbol{w}^{(y)} + \lambda \cdot (1 - P(y|x_k)) \cdot \boldsymbol{f}^{(k)}$$

- Update weight vectors of all other classes $z \neq y$:

$$\boldsymbol{w}^{(z)} \leftarrow \boldsymbol{w}^{(z)} - \lambda \cdot P(z|x_k) \cdot \boldsymbol{f}^{(k)}$$

Exercise

You are given a logistic regression model for three classes POS, NTR and NEG with weight vectors initialized to zero.

The first training example has the feature vector $\boldsymbol{f^1} = [1, 0, 0, 1, 1]$ and is of class NEG.

- Perform the update of the three weight vectors, using a learning rate of $\lambda = 0.1$
- If you had to classify $\boldsymbol{f^1}$ after the update, would it be classified correctly? What would be its classification probability?

When should we stop training?

- After a predefined number of epochs
- When the classification performance on the validation set stops improving
  - The validation set can also be used e.g. to find the optimal learning rate.
- The probabilities inform us how confident the model is about its predictions
  - We can compute the overall probability of the training data and stop when it reaches a predefined threshold (i.e., **minimize training loss**)

Logistic regression

- Describe the instances as feature vectors
- Define the prediction function
  - $P(y_j|x_i) = \underset{y^{(k)} \in Y}{\text{softmax}}(\boldsymbol{f}^{(i)} \cdot \boldsymbol{w}^{(j)}) = \frac{\exp(\boldsymbol{f}^{(i)} \cdot \boldsymbol{w}^{(j)})}{\sum_{y^{(k)} \in Y} \exp(\boldsymbol{f}^{(i)} \cdot \boldsymbol{w}^{(k)})}$

Note: If we're only interested in the predicted label, we don't have to compute the softmax.

- Define the update rule for training
  - Correct class y:

$$\boldsymbol{w}^{(y)} \leftarrow \boldsymbol{w}^{(y)} + \lambda \cdot (1 - P(y|x_k)) \cdot \boldsymbol{f}^{(k)}$$

  - All other classes $z \neq y$:

$$\boldsymbol{w}^{(z)} \leftarrow \boldsymbol{w}^{(z)} - \lambda \cdot P(z|x_k) \cdot \boldsymbol{f}^{(k)}$$

Here, we need the actual probabilities.

Probabilities – useful or not?

When the goal is just prediction (argmax), any numeric scoring function is fine.

But an algorithm that offers probabilities over labels is useful if:

- we want to interpret its decisions, i.e., understand why it reached the conclusions it did
- we want to know how confident the algorithm was or compute statistical tests on its decisions
- we want the training process to be guided by these confidence values
- its output is fed as input to some other system

Linear classifiers: Pros and cons

	Pros	Cons
Naive Bayes	• Easy to implement, fast to train • Probabilistic • Good for small data sets	• Often poor accuracy • Limited choice of features to avoid correlation
Logistic regression	• Probabilistic (interpretable) • Typically more accurate than Naive Bayes	• Can overtrain on correctly labeled examples

Readings:

- J&M appendix B: Naïve Bayes
- J&M chapter 4: Logistic regression
  - Note that several subsections are repeated

Vector semantics

Bag-of-words feature vectors

Each document x_i is represented by a vector $\boldsymbol{f}^{(i)}$ of word occurrences.

Documents	
x_1	it was pathetic the worst part was the boxing scenes
x_2	no plot twists or great scenes
x_3	and satire and great plot twists
x_4	great scenes great film

	$\boldsymbol{f}^{(1)}$	$\boldsymbol{f}^{(2)}$	$\boldsymbol{f}^{(3)}$	$\boldsymbol{f}^{(4)}$
and	0	0	2	0
boxing	1	0	0	0
film	0	0	0	1
great	0	1	1	2
it	1	0	0	0
no	0	1	0	0
or	0	1	0	0
part	1	0	0	0
pathetic	1	0	0	0
plot	0	1	1	0
…				

For text classification, we used the **column vectors**. But what can the **row vectors** tell us?

Bag-of-words feature vectors

	d_1	d_2	d_3	d_4
and	0	0	2	0
boxing	1	0	0	0
film	0	0	0	1
great	0	1	1	2
it	1	0	0	0
no	0	1	0	0
or	0	1	0	0
part	1	0	0	0
pathetic	1	0	0	0
plot	0	1	1	0
…				

- Row vectors represent "meaning profiles" of individual words
  - Words with similar meanings will have similar vectors
- This is called a **term-document matrix**
  - Note: We don't assume here that the documents are labeled

Term-document matrices

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Figure 6.2 The term-document matrix for four words in four Shakespeare plays. Each cell contains the number of times the (row) word occurs in the (column) document.

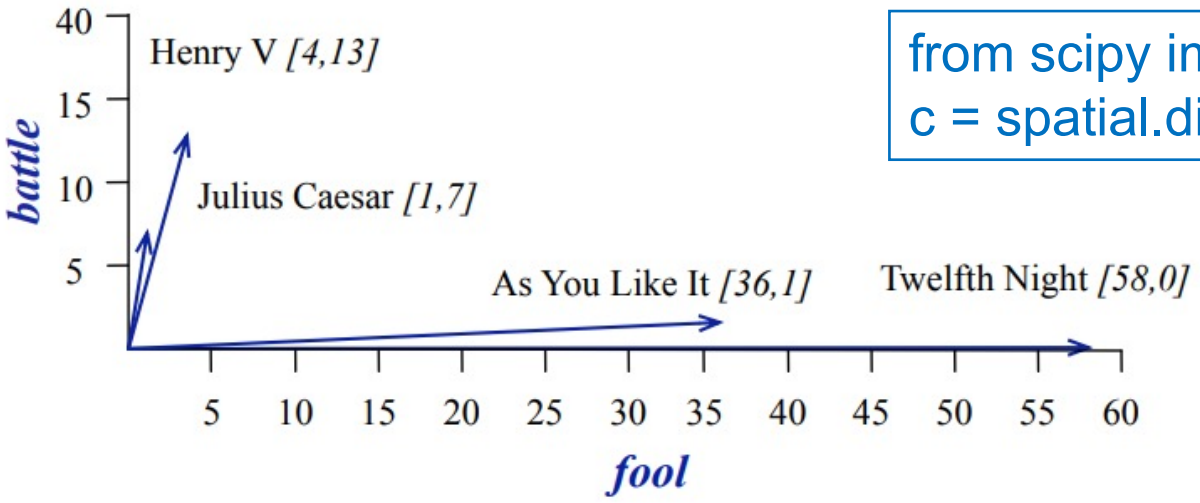
What can we do with such a matrix?

- Compute similarity between words
  - *fool* and *wit* are similar
- Compute similarity between documents
  - *As You Like It* and *Twelfth Night* are similar (comedies)
  - *Julius Caesar* and *Henry V* are similar (historical dramas)

Cosine similarity

Cosine similarity represents the **angle** between two vectors:

$$\cos(\boldsymbol{v}, \boldsymbol{w}) = \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{|\boldsymbol{v}| \cdot |\boldsymbol{w}|} = \frac{\sum_{i=1}^{N} v_i \cdot w_i}{\sqrt{\sum_{i=1}^{n} v_i^2} \cdot \sqrt{\sum_{i=1}^{n} w_i^2}}$$




```
from scipy import spatial
c = spatial.distance.cosine(v, w)
```

Term-document matrices

What can we do with such a matrix?

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- Compute similarity between words
  - *fool* and *wit* are similar
  - $cos(\text{fool}, \text{wit}) = cos([36,58,1,4], [20,15,2,3]) = 0.93$
  - $cos(\text{fool}, \text{battle}) = cos([36,58,1,4], [1,0,7,13]) = 0.09$
- Compute similarity between documents
  - *As You Like It* and *Twelfth Night* are similar (comedies)
  - *Julius Caesar* and *Henry V* are similar (historical dramas)
  - $cos(\text{AYLI}, \text{TN}) = cos([1,114,36,20], [0,80,58,15]) = 0.95$
  - $cos(\text{JC}, \text{HV}) = cos([7,62,1,2], [13,89,4,3]) = 0.69$
  - $cos(\text{TN}, \text{JC}) = cos([0,80,58,15], [7,62,1,2]) = 0.81$

Term-document matrices

What can we do with such a matrix?

	As You Like It	Twelfth Night	Julius Caesar	Henry V	Q: good fool
battle	1	0	7	13	0
good	114	80	62	89	1
fool	36	58	1	4	1
wit	20	15	2	3	0

- Compute similarity between words
- Compute similarity between documents
- **Information retrieval:**
  - Encode the **query** as an additional document
  - Find documents that are most similar to the query

Term-document matrices

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

Each word is associated with a vector that describes its meaning:

- *battle* is “the kind of word that occurs in *Julius Caesar* and *Henry V*”
- *fool* is “the kind of word that occurs in comedies, especially *Twelfth Night*”

Count weighting

Are all words equally important?

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- Intuition: A word occurring in a large proportion of documents is not a good discriminator
  - *good* does not contribute much to distinguishing the documents
  - The importance of a word should be inversely proportional to the number of documents it occurs in

TF-IDF count weighting

- TF – term frequency:
  - $tf_{t,d}$ is the frequency of term t in document d
- DF – document frequency:
  - df_t is the number of documents containing term t
- IDF – inverse document frequency:
  - $idf_t = \frac{1}{df_t}$
  - Normalize by the collection size: $\frac{N}{df_t}$
  - By convention, take the log: $\log \frac{N}{df_t}$
- TF-IDF: $tf_{t,d} \cdot \log \frac{N}{df_t}$

Implementation: replace Scikit-Learn *CountVectorizer* by *TfIdfVectorizer*

TF-IDF count weighting

Raw counts:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

TF-IDF weights:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

TF-IDF did its job!

Distributional word representations

The distributional hypothesis:

- The meaning of a word can be captured by the contexts in which it occurs

Documents? Sentences?

- Example (Nida 1975):

  A bottle of tesgüino is on the table.
  Tesgüino makes you drunk.
  Everybody likes tesgüino.
  We make tesgüino out of corn.

J.P. Firth, 1957:

"You shall know a word by the company it keeps."

Vector semantics

Core idea:

- Each word is represented as a point in a (multidimensional) semantic space
  - **Word vectors, word embeddings**
- The points are inferred from the distributions of word neighbors/contexts in text, according to the distributional hypothesis
- Similar/related words are close to each other in this space

Vector semantics

How do we get there?

- Word-document (or term-document) matrices based on co-occurrence counts
  - Count weighting with tf-idf
- Word-context matrices based on co-occurrence counts
  - Dimensionality reduction (SVD, LSA)
- An alternative approach: word2vec
  - Not today…

Word-context matrices

- One row per term/word
- One column per context term/word
  - The rows and columns may be the same, but do not have to
- Values represent counts of words **within the context** of another word

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Word-context matrices

What is the context of a word?

- The same sentence
  - Add a pinch of sugar to the cherries and boil for 10 minutes.
  - In the digital age, information is the global currency.
- n words to the left and to the right
  - Add a pinch of [sugar to the cherries and boil for] 10 minutes.
  - In [the digital age, information is the global] currency.

$n = 3$

- Often, stopwords are not counted
  - Add a [pinch of sugar to the cherries and boil for 10] minutes.
  - In the [digital age, information is the global currency.]

$n = 2$

Word-context matrices

What can we do with such a matrix?

	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

- Compute cosine similarity between words
  - *Cherry* and *strawberry* are similar
  - *Digital* and *information* are similar
- Compute cosine similarity between context words (columns)
  - Similar outcome as for rows, but not usually done

Dimensionality reduction

- Word-context vectors are sparse
  - Most values are 0
  - Most words never co-occur with *strawberry*
  - In particular when choosing small context sizes

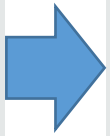
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

- Drawbacks:
  - Inefficient
  - Lack generalization capabilities

Dimensionality reduction

- Reduce the number of columns to a fixed size m (typically in the range $50 \ldots 500$)
  - e.g. using Singular Value Decomposition (SVD)
- The new columns represent abstract properties, not words:

	leash	walk	run	owner	leg	bark
dog	3	5	1	5	4	2
cat	0	3	3	1	5	0
lion	0	3	2	0	1	0
light	0	0	0	0	0	0
bark	1	0	0	2	1	0
car	0	0	4	3	0	0

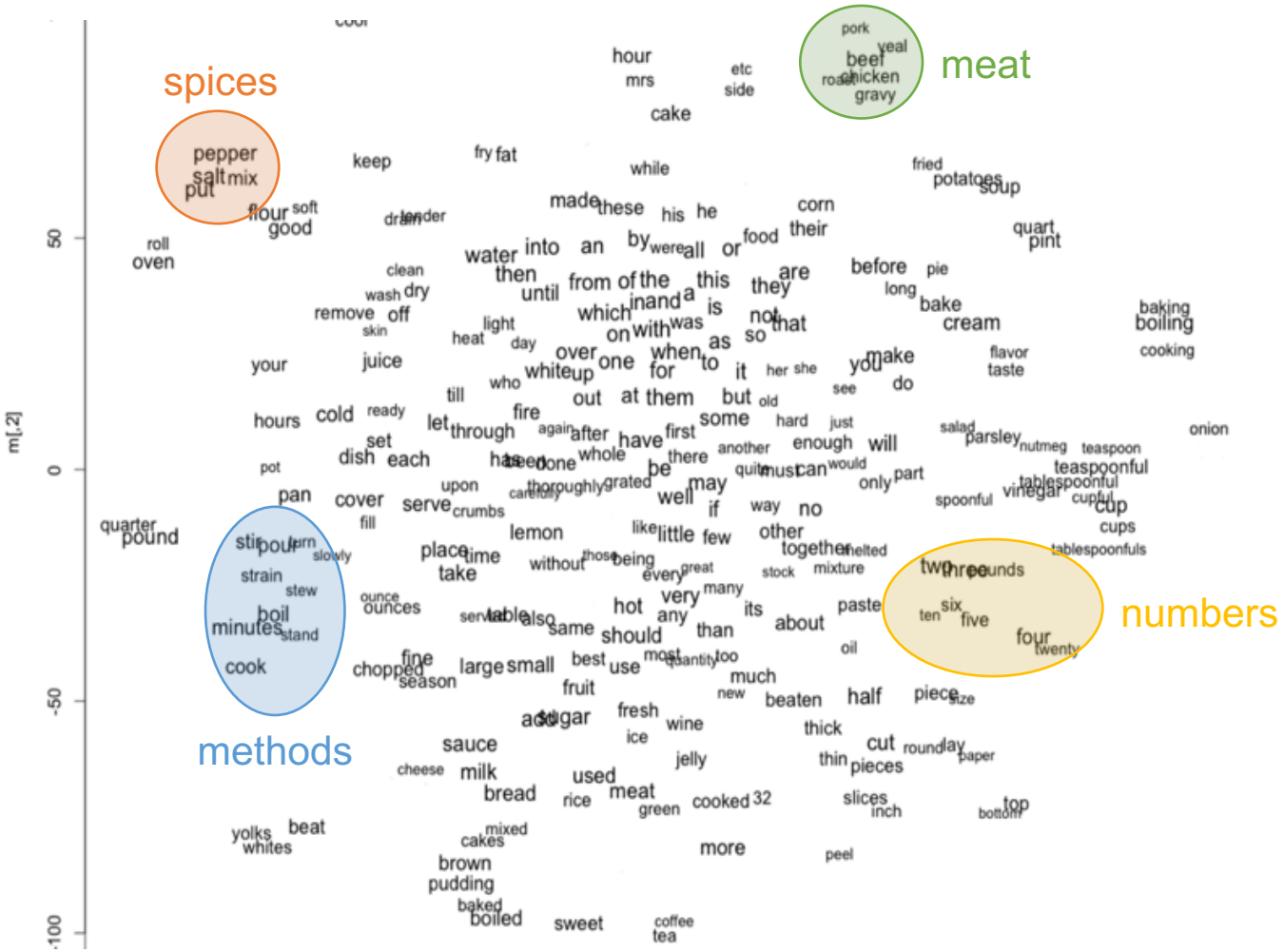


	u_A	u_B	u_C	u_D
dog	0.2	0.3	0	0.7
cat	0.7	0.2	0	0.8
lion	0.9	0.1	0	0.8
light	0	0	0.7	0
bark	0.3	0.9	0.1	0.6
car	0	0.3	0.6	0

Dimensionality reduction

- The combination of word-context vectors with SVD dimensionality reduction is known as LSA (*latent semantic analysis* – Deerwester et al. 1990)
- Other combinations are possible, e.g. using PCA (principal component analysis)
- We typically use 100-500 target dimensions but can use as few as 2 for visualization

Vector semantics



https://www.adityathakker.com

Word embeddings

- There is a direct way to obtain dense word-context vectors: **word2vec**
  - Details a bit later in this course
- Contextualized word embeddings:
  - Produce one vector per *word token*, instead of one vector per *word type*
  - E.g. **BERT**
- The Python module *gensim* is really useful for all kinds of word-vector-related experiments!

Vector semantics

What can we do with document vectors?

- Compute similarity between documents
- Find the most relevant documents to a query
- Use as feature vectors for classification tasks

What can we do with word vectors?

- Compute similarity between words
- Group similar words together
- Answer specific research questions: analogy, semantic change, bias detection
- Use as feature vectors for classification tasks

Readings

- J&M chapter 5:
  Vector semantics and embeddings
  - Except 5.5 (word2vec)