

Big Data & Automated Content Analysis

Week 8 – Monday: »Supervised Approaches to Text Analysis «

Anne Kroon
a.c.kroon@uva.nl
@annekroon

17 May 2021

Today

Machine Learning

Predicting things

You have done it before!

From regression to classification

Classifiers

Vectorizers

Classification

SML

An implementation

A note on the input data

Looking back & forward

Machine Learning

A familiar picture by now

	Methodological approach		
	<i>Counting and Dictionary</i>	<i>Supervised Machine Learning</i>	<i>Unsupervised Machine Learning</i>
Typical research interests and content features	visibility analysis sentiment analysis subjectivity analysis	frames topics gender bias	frames topics
Common statistical procedures	string comparisons counting	support vector machines naive Bayes	principal component analysis cluster analysis latent dirichlet allocation semantic network analysis
<div> <div>deductive</div> <div>inductive</div> </div>			

Some terminology

Supervised machine learning

You have a dataset with both predictor and outcome (independent and dependent variables; features and labels) — a *labeled* dataset. Think of regression: You measured x_1 , x_2 , x_3 and you want to predict y , which you also measured

Unsupervised machine learning

You have no labels. (You did not measure y)

























Again, you already know some techniques to find out how x_1 , x_2, \dots, x_i co-occur from other courses:

- Principal Component Analysis (PCA) and Singular Value Decomposition (SVD)
- Cluster analysis
- Topic modelling

Let's distinguish four use cases. . .

1. Finding similar variables (dimensionality reduction) – unsupervised
2. Finding similar cases (clustering) – unsupervised
3. Predicting a continuous variable (regression) – supervised
4. Predicting group membership (classification) – supervised

	x1	x2	x3	x4	x5	y
case1	■	■	■	■	■	■
case2	■	■	■	■	■	■
case3	■	■	■	■	■	■
case4	■	■	■	■	■	■

	x1	x2	x3	x4	x5	(y)
case1						
case2						
case3						
case4						

Dimensionality reduction: finding similar variables (features)

	x1	x2	x3	x4	x5	(y)
case1	■	■	■	■	■	(■)
case2	■	■	■	■	■	(■)
case3	■	■	■	■	■	(■)
case4	■	■	■	■	■	(■)

Clustering: finding similar cases

	x1	x2	x3	x4	x5	→	y
case1	■	■	■	■	■	→	■
case2	■	■	■	■	■	→	■
case3	■	■	■	■	■	→	■
case4	■	■	■	■	■	→	■
new case	■	■	■	■	■	→	?

Regression and classification: learn how to predict y.

Note, again, that the ■ signs can be *anything*. For us, often word counts or *tf*·*idf* scores (x) and, for supervised approaches, a topic, a sentiment, or similar (y).

But it could also be pixel colors or clicks on links or anything else.

	x1	x2	x3	x4	x5	y
case1	■	■	■	■	■	■
case2	■	■	■	■	■	■
case3	■	■	■	■	■	■
case4	■	■	■	■	■	■

Predicting things

Predicting things

You have done it before!

You have done it before!

Regression

1. Based on your data, you estimate some regression equation
$$y_i = \alpha + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i$$
2. Even if you have some *new unseen data*, you can estimate your expected outcome \hat{y} !
3. Example: You estimated a regression equation where y is newspaper reading in days/week:
$$y = -.8 + .4 \times man + .08 \times age$$
4. You could now calculate \hat{y} for a man of 20 years and a woman of 40 years – *even if no such person exists in your dataset*:

$$\hat{y}_{man20} = -.8 + .4 \times 1 + .08 \times 20 = 1.2$$

$$\hat{y}_{woman40} = -.8 + .4 \times 0 + .08 \times 40 = 2.4$$

This is
Supervised Machine Learning!

...but...

- We will only use *half* (or another fraction) of our data to estimate the model, so that we can use the other half to check if our predictions match the manual coding (“labeled data”, “annotated data” in SML-lingo)
 - e.g., 2000 labeled cases, 1000 for training, 1000 for testing — if successful, run on 100,000 unlabeled cases
- We use many more independent variables (“features”)
- Typically, IVs are word frequencies (often weighted, e.g. $\text{tf} \times \text{idf}$) (\Rightarrow BOW-representation)

Predicting things

From regression to classification

Machine Learning
oooooooo

Predicting things
o
ooo
o●ooo
oo
ooo

Classification
oooooooooooooooo

SML
ooooo
ooooooo
oooo

Looking back & forward
ooo
oooooooooooooooo

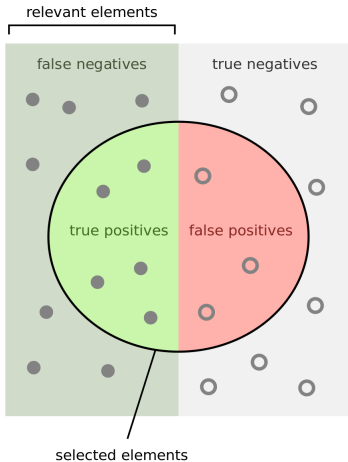
In the machine learning world, predicting some continuous value is referred to as a **regression** task. If we want to predict a binary or categorical variable, we call it a **classification** task.

(quite confusingly, even if we use a logistic regression for the latter)

Classification tasks

For many computational approaches, we are actually not that interested in predicting a continuous value. Typical questions include:

- Is this article about topic A, B, C, D, or E?
- Is this review positive or negative?
- Does this text contain frame F?
- Is this satire?
- Is this misinformation?
- Given past behavior, can I predict the next click?



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Some measures

- Accuracy
- Recall
- Precision
- $F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$
- AUC (Area under curve)
[0, 1], 0.5 = random guessing

Different classification algorithms

- It is an empirical question which one works best
- We typically try several ones and select the best
- (remember: we have a test dataset that we did *not* use to train the model, so that we can assess how well it predicts the test labels based on the test features)

(to make it easier, imagine a binary classification ("positive"/"negative"), but it doesn't really matter whether there are two or more labels)

Predicting things

Classifiers

Different classifiers

Typical options in a nutshell:

- Naïve Bayes
- Logistic Regression
- Support Vector Machine (SVM/SVC)
- Random forests

Predicting things

Vectorizers

Different vectorizers

1. CountVectorizer (=simple word counts)
2. TfidfVectorizer (word counts (“term frequency”) weighted by number of documents in which the word occurs at all (“inverse document frequency”))

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t$$

Which one would you (not) use for which purpose?

NB with Count

	precision	recall
positive reviews:	0.87	0.77
negative reviews:	0.79	0.88

NB with TfIdf

	precision	recall
positive reviews:	0.87	0.78
negative reviews:	0.80	0.88

LogReg with Count

	precision	recall
positive reviews:	0.87	0.85
negative reviews:	0.85	0.87

LogReg with TfIdf

	precision	recall
positive reviews:	0.89	0.88

Classification

Let's consider three tasks

For a given text (say, a news article, a press release, a review),
determine the

sentiment e.g., [positive|neutral|negative]

topic e.g., [sports|economy|politics|entertainment|other]

frames e.g., [economic|human|moral|conflict], or
non-exclusive: economic = [0|1], human = [0|1], ...



Imagine using a dictionary-based (list of keywords, list of regular expressions, or similar) approach to these tasks. How does the design (length, inclusiveness, etc.) of this list influence precision and recall?

Dictionary-based approaches for text classification

good for

- distinct, manifest things (names of organizations, pronouns, swearwords (?), ...)
- little room for interpretation/misunderstandings etc.
- “must-be-explainable-to-a-five-year-old”

bad for

- latent constructs and concepts
- implicit things

Hence, *not* state-of-the-art for

- topics
- frames
- sentiment

Machine Learning
oooooooo

Predicting things
o
oooo
ooooo
ooooo
oo
ooo

Classification
oooo●oooooooo

SML
oooooo
oooooo
oooooo
ooo

Looking back & forward
ooo
oooooooooooooooo

From dictionary approaches to SML

- Early days of sentiment analysis: list of positive words, list of negative words, count what occurs most
- You can even *buy* lists of words that are meant to measure constructs like “positive emotions” or even “analytic” or “authentic” language use from a psychologist (LIWC, Pennebaker2007)



What do you think? Can this even work

Bag-of-words dictionary approaches to sentiment analysis

con

- simplistic assumptions
- e.g., intensifiers cannot be interpreted (“really” in “really good” or “really bad”)
- or, even more important, negations.

Improving the BOW approach

Example: Sentistrength (Thelwall2012)

- $-5 \dots -1$ and $+1 \dots +5$ instead of positive/negative
- spelling correction
- “booster word list” for strengthening/weakening the effect of the following word
- interpreting repeated letters (“baaaaaad”), CAPITALS and !!!
- idioms, negation

VADER by Hutto2014 works in a similar way. Even though this is much less naïve than LIWC, for instance, the problem remains: Can we construct a dictionary that, *irrespective of the context*, gives us a meaningful estimate of sentiment?

Machine Learning

oooooooo

Predicting things

o
oooo
ooooo
ooooo
oo
ooo

Classification

oooooooo●oooo

SML

ooooo
oooooo
ooooooo
ooo

Looking back & forward

ooo
oooooooooooooooo

Such an *off-the-shelf* dictionary does not
(and probably cannot) exist.

Boukes2020: Sentiment analysis of economic news

Table A1. Correlations between sentiment scores using different methods for headlines (above) and full texts (below).

	Headline							
	Manual coding	Recession	D & B	LIWC	SentiStrength	Pattern	Polyglot	DANEW
Manual coding	1.00 ***							
Recession	-	-						
Damstra and Boukes (2018)	0.16 ***	-	1.00 ***					
LIWC	0.30 ***	-	0.16 ***	1.00 ***				
SentiStrength	0.24 ***	-	0.08 **	0.26 ***	1.00 ***			
Pattern	0.22 ***	-	0.00	0.30 ***	0.22 ***	1.00 ***		
Polyglot	0.30 ***	-	0.19 ***	0.32 ***	0.37 ***	0.26 ***	1.00 ***	
DANEW	0.24 ***	-	0.04	0.43 ***	0.33 ***	0.23 ***	0.32 ***	1.00 ***
	Full text							
	Manual coding	Recession	D & B	LIWC	SentiStrength	Pattern	Polyglot	DANEW
Manual coding	1.00 ***							
Recession	-0.06 *	1.00 ***						
Damstra and Boukes (2018)	0.27 ***	-0.16 ***	1.00 ***					
LIWC	0.39 ***	0.02	0.27 ***	1.00 ***				
SentiStrength	0.17 ***	-0.01	0.10 ***	0.18 ***	1.00 ***			
Pattern	0.13 ***	-0.02	0.04	0.28 ***	0.12 ***	1.00 ***		
Polyglot	0.26 ***	0.05	0.17 ***	0.41 ***	0.21 ***	0.30 ***	1.00 ***	
DANEW	0.15 ***	0.06 *	0.05	0.36 ***	0.18 ***	0.29 ***	0.37 ***	1.00 ***

The word "recession" did not occur in headlines of our sample, as such, no correlation coefficient is available for the recession classifier; *** $p < .001$, ** $p < .010$, * $p < .05$.

Boukes2020: Sentiment analysis of economic news

- Dictionaries have low agreement with each other, and also with human coders
- Even their own dictionary didn't agree
- **This is not because these dictionaries are particularly bad!.** Main point: For such a complex and context-dependent task, a dictionary is just not the right tool.

VanAtteveldt2021: Extending Boukes2020 with SML

“manual coding (using undergraduate students) yields the best results

[...] A good second place is taken by crowd coding [...]

[...] machine learning performs worse than both students' manual coding and crowd coding. Reaching $\alpha = 0.50$ for deep learning (CNN) and slightly worse for classical machine learning (SVM; $\alpha = 0.41$, NB; $\alpha = 0.40$), machine learning still performs significantly better than chance. However, since these results are lower than generally accepted levels of inter-coder reliability [...]

Finally, [...] dictionaries [...] perform worse than the machine learning results and much worse than manual annotation [...]
[and] approximate chance agreement”

Vermeer2019: Satisfaction with brands

Category	Technique	Accuracy	Precision	Recall
Satisfaction (<i>N</i> = 854)				
Sentiment analysis	LIWC	0.05	0.06	0.04
	P	0.04	0.04	0.04
	SN	0.07	0.07	0.08
Dictionary-based	D	0.15	0.30	0.10
Machine learning	BNB	0.38	0.44	0.34
	MNB	0.32	0.67	0.21
	LR	0.51	0.38	0.76
	SGD	0.49	0.38	0.69
	SVM	0.52	0.41	0.63
	PA	0.50	0.40	0.68
Neutral (<i>N</i> = 760)				
Sentiment analysis	LIWC	0.13	0.16	0.10
	P	0.13	0.13	0.14
	SN	0.19	0.16	0.22
Dictionary-based	D	0.14	0.35	0.09
Machine learning	BNB	0.28	0.25	0.32
	MNB	0.15	0.34	0.10
	LR	0.37	0.25	0.74
	SGD	0.33	0.23	0.60
	SVM	0.36	0.24	0.69
	PA	0.34	0.24	0.60
Dissatisfaction (<i>N</i> = 267)				
Sentiment analysis	LIWC	0.20	0.15	0.29
	P	0.19	0.12	0.40
	SN	0.22	0.14	0.54
Dictionary-based	D	0.09	0.41	0.05
Machine learning	BNB	0.26	0.20	0.40
	MNB	0.25	0.48	0.16
	LR	0.35	0.23	0.77
	SGD	0.39	0.32	0.48
	SVM	0.04	0.02	1.00
	PA	0.35	0.23	0.71

Note. LIWC Linguistic Inquiry and Word Count; P Pattern; SN Sentiment Net; D Dictionary-based; BN Bernoulli Naïve Bayes; MNB Multinomial Naïve Bayes; LR Logistic Regression; SGD Stochastic Gradient Descent; SVM Support Vector

Machine Learning
oooooooo

Predicting things
o
oooo
ooooo
ooooo
oo
ooo

Classification
oooooooooooooooo●

SML
oooooo
oooooooo
oooo

Looking back & forward
ooo
oooooooooooooooo

SML is no panacea, but the most promising approach to analyzing large quantities of texts. Don't believe off-the-shelf packages that claim to do the work for you. (For small datasets, just do it by hand.)

SML

SML to code frames and topics

Some work by Burscher2014 and Burscher2015

- Humans can code generic frames (human-interest, economic, ...)
- Humans can code topics from a pre-defined list
- **But it is very hard to formulate an explicit rule**
(as in: code as 'Human Interest' if regular expression R is matched)

⇒ This is where you need supervised machine learning!

TABLE 4
Classification Accuracy of Frames in Sources Outside the Training Set

	<i>VK/NRC</i> <i>→ Tel</i>	<i>VK/TEL</i> <i>→ NRC</i>	<i>NRC/TEL</i> <i>→ VK</i>
Conflict	.69	.74	.75
Economic Cons.	.88	.86	.86
Human Interest	.69	.71	.67
Morality	.97	.90	.89

Note. VK = Volkskrant, NRC = NRC/Handelsblad, TEL = Telegraaf

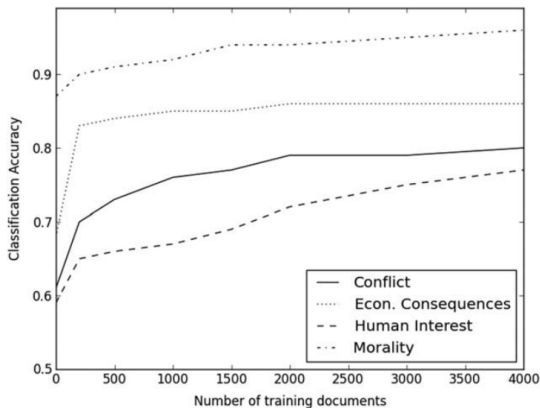


FIGURE 1 Relationship between classification accuracy and number of training documents.

FIGURE 1

Learning Curves for the Classification of News Articles and PQs

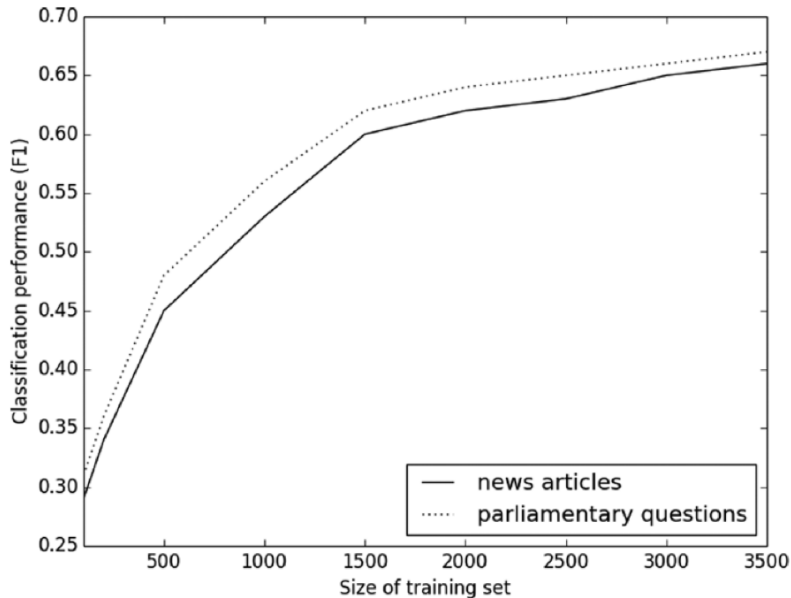


TABLE 1

F1 Scores for SML-Based Issue Coding in News Articles and PQs

Issue	News Articles			PQs	
		All Words	Lead Only		All Words
Features	N	F1	F1	N	F1
Macroeconomics	413	.54	.63	172	.46
Civil rights and minority issues	327	.34	.28	192	.53
Health	444	.70	.71	520	.81
Agriculture	114	.72	.76	159	.66
Labor and employment	217	.43	.49	174	.58
Education	188	.79	.71	229	.78
Environment	152	.34	.44	237	.59
Energy	81	.35	.59	67	.66
Immigration and integration	150	.50	.57	239	.78
Transportation	416	.58	.67	306	.81
Law and crime	1198	.70	.69	685	.77
Social welfare	115	.33	.34	214	.54
Community development and housing	113	.45	.44	136	.72
Banking, finance, and commerce	622	.62	.67	188	.58
Defense	393	.59	.55	196	.71
Science, technology, and communication	426	.64	.59	57	.53
International affairs and foreign aid	1,106	.70	.64	352	.65
Government operations	1,301	.71	.72	276	.48
Other issue	3,322	.84	.80	360	.51
Total	11,089	.71	.68	4,759	.69

NOTE: The F1 score is equal to the harmonic mean of recall and precision. Recall is the fraction of relevant documents that are retrieved, and precision is the fraction of retrieved documents that are relevant.

What does this mean for our research?

It we have 2,000 documents with manually coded frames and topics. . .

- we can use them to train a SML classifier
- which can code an unlimited number of new documents
- with an acceptable accuracy (at least for some of them)

Some easier tasks even need only 500 training documents, see [Hopkins2010](#).

SML

An implementation

An implementation

Let's say we have a list of tuples with movie reviews and their rating:

```
1 reviews=[("This is a great movie",1),("Bad movie",-1), ... ...]
```

And a second list with an identical structure:

```
1 test=[("Not that good",-1),("Nice film",1), ... ...]
```

Both are drawn from the same population, it is pure chance whether a specific review is on the one list or the other.

Based on an example from <http://blog.dataquest.io/blog/naive-bayes-movies/>

Training a A Naïve Bayes Classifier

```
1 from sklearn.naive_bayes import MultinomialNB
2 from sklearn.feature_extraction.text import CountVectorizer
3 from sklearn import metrics
4
5 # This is just an efficient way of computing word counts
6 vectorizer = CountVectorizer(stop_words='english')
7 train_features = vectorizer.fit_transform([r[0] for r in reviews])
8 test_features = vectorizer.transform([r[0] for r in test])
9
10 # Fit a naive bayes model to the training data.
11 nb = MultinomialNB()
12 nb.fit(train_features, [r[1] for r in reviews])
13
14 # Now we can use the model to predict classifications for our test
    features.
15 predictions = nb.predict(test_features)
16 actual=[r[1] for r in test]
17
18 print("Precision: {}".format(metrics.precision_score(actual,
```

And it works!

Using 50,000 IMDB movies that are classified as either negative or positive,

- I created a list with 25,000 training tuples and another one with 25,000 test tuples and
- trained a classifier
- with precision and recall values $> .80$

Dataset obtained from <http://ai.stanford.edu/~amaas/data/sentiment>, Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. *49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)*

Playing around with new data

```
1 newdata=vectorizer.transform(["What a crappy movie! It sucks!", "This is  
   awesome. I liked this movie a lot, fantastic actors","I would not  
   recomment it to anyone.", "Enjoyed it a lot"])  
2 predictions = nb.predict(newdata)  
3 print(predictions)
```

This returns, as you would expect and hope:

```
1 [-1  1 -1  1]
```

Machine Learning
oooooooo

Predicting things
o
ooo
oooo
ooooo
oo
ooo

Classification
oooooooooooooooo

SML
ooooo
ooooo●o
ooo

Looking back & forward
oo
oooooooooooooooo

But we can do even better

We can use different vectorizers and different classifiers.

Machine Learning
oooooooo

Predicting things
o
oooo
ooooo
oo

Classification
oooooooooooooooo

SML
ooooo
ooooo●
ooo

Looking back & forward
ooo
oooooooooooooooo

ELI5

```
In [98]: import eli5  
eli5.show_weights(pipe, top=10)
```

Out[98]: **y=1** top features

Weight?	Feature
+9.043	great
+8.487	excellent
+6.908	perfect
... 37662 more positive ...	
... 37178 more negative ...	
-6.507	worse
-7.347	poor
-8.341	boring
-8.944	waste
-8.976	bad
-9.152	awful
-12.749	worst

```
In [111]: eli5.show_prediction(clf, test[0][0],vec=vec)
```

Out[111]: **y=1** (probability **0.844**, score **1.689**) top features

Contribution?	Feature
+1.920	Highlighted in text (sum)
-0.232	<BIAS>

it is a **rare** and **fine** spectacle, an allegory of death and transfiguration that is **neither** preachy nor **mawkish**. a work of **mature** and courageous insight, northfork avoids arthouse distinction by refusing to belong to a kind. **unlike** the most memorable and accomplished film to impose an **obvious** comparison, **wim wenders'** 1987 wings of desire (der himmel über berlin), it sustains an ambivalence in a narrative spectrum spanning from the **mundane** to the supernatural. this story of earthy and celestial eminent domains in the **american west** withholds the **fairytale** literalness that marked its **german** predecessor in the **ad hoc** genre of angels shedding their wings with obsequious sentimentalism. its celestial transcendence, be it **inspired** by doleful faith or **impelled** by a fever dream, never

SML

A note on the input data

The input scikit-learn expects

A training dataset consisting of:

1. an array (e.g., a list) of labels (`y_train`)
2. a corresponding array (e.g., a list) of feature vectors (`X_train`)

A test dataset consisting of:

1. an array (e.g., a list) of labels (`y_test`)
2. a corresponding array (e.g., a list) of feature vectors (`X_test`)

The feature vectors can be created via a *vectorizer*, but could in principle also just be lists themselves.

We use a lowercase `y` because it is a onedimensional vector, and an uppercase `X` because it is a two-dimensional matrix.

The input scikit-learn expects

- It does not matter *how* you create y and X !
- Getting data into the right shape can be as much work (or more) as training the classifier itself

Typical techniques:

- Reading text files from folders into lists of strings (looping over folder contents)
- Reading from csv file either directly into lists (csv module) or via pandas
- List comprehension to restructure or process data
- Potentially, you need to split into train and test dataset yourself (with slicing, or with scikit-learn itself)



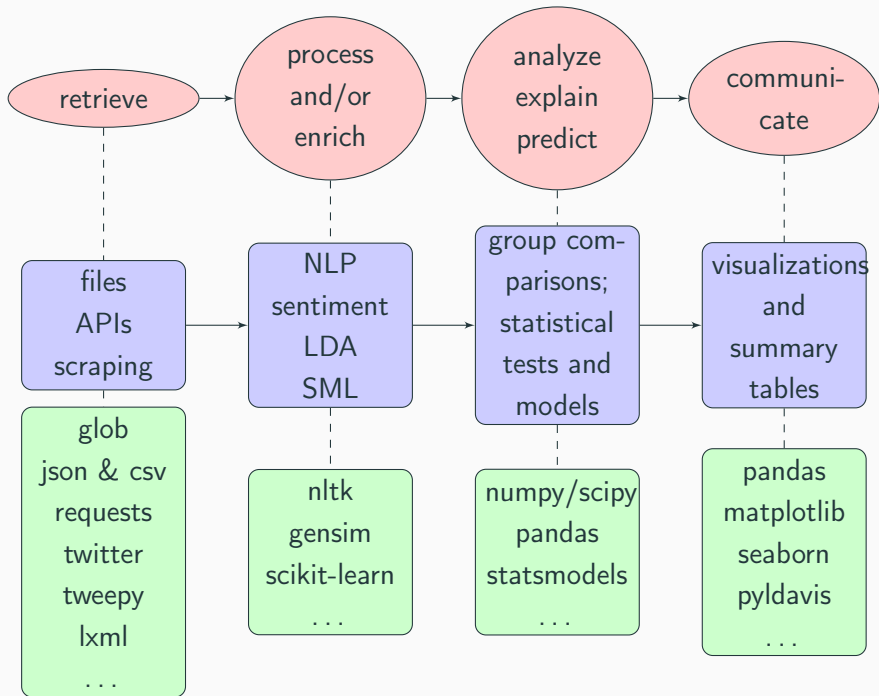
Any questions?

Looking back & forward

Steps of a CSS project

We learned techniques for:

- retrieving data
- processing data
- analyzing data
- visualising data



Looking back & forward

A good workflow

Machine Learning

oooooooo

Predicting things

o
oooo
ooooo
ooooo
oo
ooo

Classification

oooooooooooooooo

SML

ooooo
oooooo
oooooo
ooo

Looking back & forward

ooo
o●oooooooooooo

A good workflow

Machine Learning
oooooooo

Predicting things
o
ooo
oooo
ooooo
oo
ooo

Classification
oooooooooooooooo

SML
ooooo
ooooo
oooooo
ooo

Looking back & forward
ooo
oo●oooooooooooo

The big picture

Start with pen and paper

1. Draw the Big Picture
2. Then work out what components you need

Develop components separately

One script for downloading the data, one script for analyzing

- Avoids waste of resources (e.g., unnecessary downloading multiple times)
- Makes it easier to re-use your code or apply it to other data

Start small, then scale up

- Take your plan (see above) and solve *one* problem at a time (e.g., parsing a review page; or getting the URLs of all review pages)
- (for instance, by using functions [next slides])

Machine Learning
oooooooo

Predicting things
o
oooo
ooooo
ooooo
oo
ooo

Classification
oooooooooooooooo

SML
oooooo
ooooooo
ooooooo
oooo

Looking back & forward
ooo
oooo●oooooooo

Develop components separately

If you copy-paste code, you are doing something wrong

- Write loops!
- If something takes more than a couple of lines, write a function!

Copy-paste approach (ugly, error-prone, hard to scale up)

```
1 allreviews = []
2
3 response = requests.get('http://xxxxx')
4 tree = fromstring(response.text)
5 reviewelements = tree.xpath('//div[@class="review"]')
6 reviews = [e.text for e in reviewelements]
7 allreviews.extend(reviews)
8
9 response = requests.get('http://yyyyy')
10 tree = fromstring(response.text)
11 reviewelements = tree.xpath('//div[@class="review"]')
12 reviews = [e.text for e in reviewelements]
13 allreviews.extend(reviews)
```

Better: for-loop

(easier to read, less error-prone, easier to scale up (e.g., more URLs, read URLs from a file or existing list)))

```
1 allreviews = []
2
3 urls = ['http://xxxxx', 'http://yyyyy']
4
5 for url in urls:
6     response = requests.get(url)
7     tree = fromstring(response.text)
8     reviewelements = tree.xpath('//div[@class="review"]')
9     reviews = [e.text for e in reviewelements]
10    allreviews.extend(reviews)
```

Even better: for-loop with functions

(main loop is easier to read, function can be re-used in multiple contexts)

```
1 def getreviews(url):
2     response = requests.get(url)
3     tree = fromstring(response.text)
4     reviewelements = tree.xpath('//div[@class="review"]')
5     return [e.text for e in reviewelements]
6
7
8 urls = ['http://xxxxx', 'http://yyyyy']
9
10 allreviews = []
11
12 for url in urls:
13     allreviews.extend(getreviews(url))
```

Scaling up

Until now, we did not look too much into aspects like code style, re-usability, scalability

- Use functions to make code more readable and re-usable
- Avoid re-calculating values
- Think about how to minimize memory usage (e.g., generators)
- Do not hard-code values, file names, etc., but take them as arguments

Make it robust

You cannot foresee every possible problem.

Most important: Make sure your program does not fail and loose all data just because something goes wrong at case 997/1000.

- Use try/except to explicitly tell the program how to handle errors
- Write data to files (or database) in between
- Use `assert len(x) == len(y)` for sanity checks

Practice yourself!

- Reproduce examples from the book for SML on the IMDB data (11.2, 11.3, 11.4) (check `week08/exercises/codefrombook.py` on github if you do not want to type over the code)
- Play around with different options! Can you tweak the models and make them even better? Take a look back at week 7 when we compared different vectorizers as well!

Thursday meeting

- Sign up on canvas in case you have questions about your final project.

