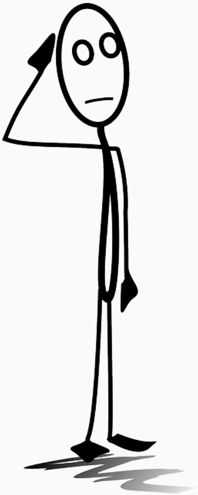
Big Data and Automated Content Analysis (6EC)
Week 7: »Processing textual data«
Monday

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

May 16, 2022

UvA RM Communication Science

Today



Everything clear from last week?

This week, we will get a general overview of working with textual data. Due to a lack of time, I will introduce you to some of the basic concepts, point you to resources, and give you a practical, hands-on introduction.

Unsupervised Machine Learning for Text Classification

Boumans and Trilling, 2016: Types of Automated Content Analysis

	Methodological approach		
	Counting and Dictionary	*Supervised Machine Learning*	*Unsupervised Machine Learning*
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

Supervised vs Unsupervised

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 `x1`, `x2`, `x3` and you want to predict `y`, which you also measured

Unsupervised machine learning

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

You might already know *some* techniques to figure out whether `x1`, `x2`,`...x_i` co-occur

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

Unsupervised Machine Learning for Text Classification

An introduction to LDA

Enter **topic modeling with Latent Dirichlet Allocation (LDA)**

LDA, what's that?

No mathematical details here, but the general idea

- There are k topics, $T_1 \dots T_k$
- Each document D_i consists of a mixture of these topics, e.g.$80\% T_1, 15\% T_2, 0\% T_3, \dots 5\% T_k$
- On the next level, each topic consists of a specific probability distribution of words
- Thus, based on the frequencies of words in D_i, one can infer its distribution of topics
- Note that LDA is a Bag-of-Words (BOW) approach

Doing a LDA in Python

We will use gensim (**Rehurek10softwareframework**) for this (make sure you have version >4.0)

Let us assume you have a list of lists of documents called `texts`:

```
print(texts[0][:115])
```

which looks something like:

```
'Stop the presses: CNN covered some actual news yesterday when it reported on the story of
↪ medical kidnapping victim Alyssa Gilderhus at the Mayo Clinic. But was it actually
↪ InfoWars and FreeMartyG which publicly shamed CNN into doing this real journalism? Cue the
↪ Mission Impossible theme music for this one...\n\nThis mission, as we accepted it, began
↪ more than a year ago during the baby Charlie Gard medical kidnapping scandal in the UK and
↪ we thought that it had ended with an apparently unsuccessf'
```

Preprocessing

Your preprocessing steps and feature engineering decisions *largely* affect your topics

1. You can apply 'manual' preprocessing steps ...
2. ...In isolation or combination with for example `tfidf` transformations

```
texts_clean = [text.lower() for text in texts]
texts_clean=[" ".join(text.split()) for text in texts_clean]  #remove dubble spaces
texts_clean = ["".join([l for l in text if l not in punctuation]) for text in texts_clean]
↪  #remove punctuation
texts_clean[0][:500]
```

which looks something like:

```
'stop the presses cnn covered some actual news yesterday when it reported on the story of
↪  medical kidnapping victim alyssa gilderhus at the mayo clinic but was it actually infowars
```

Preprocessing

Without *stopword removal*, *tfidf* transformation and/or *pruning*, you topics will not be very informative.

```
mystopwords = set(stopwords.words('english')) # use default NLTK stopword list; alternatively:
# mystopwords = set(open('mystopwordfile.txt').readlines())  #read stopword list from a textfile with one stopword per line
texts_clean = [" ".join(word for word in text.split() if word not in mystopwords) for text in texts_clean]
texts_clean[0][:500]
```

which looks something like:

```
'stop presses cnn covered actual news yesterday reported story medical kidnapping victim alyssa gilderhus mayo clinic actually infowars freemartyg publicly shamed cnn real journalism cue mission impossible theme music one mission accepted began year ago baby charlie gard medical kidnapping scandal uk thought ended apparently unsuccessful april fools joke cnn sure many recall charlie gard infant rare form otherwise notsorare condition mitochondrial disease story went viral made international news '
```

Tokenization

`gensim` expects a list of words (hence: `tokenize` your corpus)

```
tokenized_texts_clean = [TreebankWordTokenizer().tokenize(text) for text in texts_clean ] #
↪  tokenize texts; convert all strings to a list of tokens
tokenized_texts_clean[0][:500]
```

which looks something like:

```
['stop',
'presses',
'cnn',
'covered',
'actual',
'news',
'yesterday',
'reported',
'story',
..
```

LDA implementation

LDA implementation

```python
raw_m1 = tokenized_texts_clean

# assign a token_id to each word
id2word_m1 = corpora.Dictionary(raw_m1)
# represent each text by (token_id, token_count) tuples
ldacorpus_m1 = [id2word_m1.doc2bow(text) for text in raw_m1]

#estimate the model
lda_m1 = models.LdaModel(ldacorpus_m1, id2word=id2word_m1, num_topics=10)
lda_m1.print_topics()
```

```
[(0, '0.015*"trump" + 0.012*"said" + 0.006*"president" + 0.006*"people" + 0.004*"cnn" +
↪  0.004*"us" + 0.004*"house" + 0.004*"news" + 0.003*"also" + 0.003*"twitter"'),
(1,'0.010*"said" + 0.008*"trump" + 0.004*"one" + 0.004*"people" + 0.004*"us" +
↪  0.004*"president" + 0.004*"would" + 0.003*"media" + 0.003*"also" + 0.003*"new"'),
(2, '0.011*"trump" + 0.009*"said" + 0.007*"president" + 0.005*"would" + 0.004*"people" +
↪  0.004*"us" + 0.003*"also" + 0.003*"like" + 0.003*"news" + 0.003*"state"'),
(3, '0.010*"trump" + 0.006*"president" + 0.005*"said" + 0.004*"would" + 0.004*"us" +
```

Visualization with pyldavis

```
import pyLDAvis
import pyLDAvis.gensim_models as gensimvis
# first estiate gensim model, then:
vis_data = gensimvis.prepare(lda_m1,ldacorpus_m1,id2word_m1)
pyLDAvis.display(vis_data)
```

Unsupervised Machine Learning for Text Classification

Choosing the best (or a good) topic model

Choosing the best (or a good) topic model

- There is no single best solution (e.g., do you want more coarse of fine-grained topics?)
- Non-deterministic
- Very sensitive to preprocessing choices
- Interplay of both metrics and (qualitative) interpretability

See for more elaborate guidance:

Maier, D., Waldherr, A., Miltner, P., Wiedemann, G., Niekler, A., Keinert, A., . . . Adam, S. (2018). Applying LDA Topic Modeling in Communication Research: Toward a Valid and Reliable Methodology. *Communication Methods and Measures, 12*(2–3), 93–118. doi:10.1080/19312458.2018.1430754

Evaluation metrics

qualitative: human judgement

Observation and interpretation based: observe the top N words in your topic, and evaluate the quality of the coherence of the topic. Can you identify words that do not belong to a topic?

quantitative: coherence

- mean coherence of the whole model: attempts to quantify the interpretability
- coherence per topic: allows to get topics that are most likely to be coherently interpreted (`.top_topics()`)

So, how do we do this?

- Estimate multiple models, store the metrics for each model, and then compare them (numerically, or by plotting)
- Idea: We select some candidate models, and then look whether they can be interpreted.
- But what can we tune?

Choosing k: How many topics do we want?

- Typical values: $10 < k < 200$
- Too low: losing nuance, so broad it becomes meaningless
- Too high: picks up tiny pecularities instead of finding general patterns
- There is no inherent ordering of topics
- We can throw away or merge topics later, so if out of $k = 50$ topics 5 are not interpretable and a couple of others overlap, it still may be a good model

Choosing α: how sparse should the document-topic distribution θ be?

- The higher α, the more topics per document
- Default: $1/k$
- But: We can explicitly change it, or – really cool – even learn α from the data (`alpha = "auto"`)

```
mylda =LdaModel(corpus=tfidfcorpus[ldacorpus], id2word=id2word, num_topics=50, alpha='auto',
    passes=10)
```

Unsupervised Machine Learning for Text Classification

Using topic models

Using topic models

You got your model – what now?

1. Assign topic scores to documents
2. Label topics
3. Merge topics, throw away boilerplate topics and similar (manually, or aided by cluster analysis)
4. Compare topics between, e.g., outlets
5. or do some time-series analysis.

Example: Tsur, O., Calacci, D., & Lazer, D. (2015). A Frame of Mind: Using Statistical Models for Detection of Framing and Agenda Setting Campaigns. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (pp. 1629–1638).

Try it out yourself..

- Work through the example notebook on LDA: https://github.com/uvacw/teaching-bdaca/blob/main/6ec-course/week07/exercises/topic-modelling.ipynb
- *Other resources*: https://github.com/uvacw/teaching-bdaca/blob/main/12ec-course/week10/lda.ipynb https://github.com/annekroon/gesis-machine-learning/blob/main/day3/excercise-afternoon/lda.ipynb

Supervised Machine Learning for Text Classification

Supervised Machine Learning for Text Classification

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. tf$\times$idf) ($\Rightarrow$BOW-representation)

Supervised Machine Learning for Text Classification

From regression to classification

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

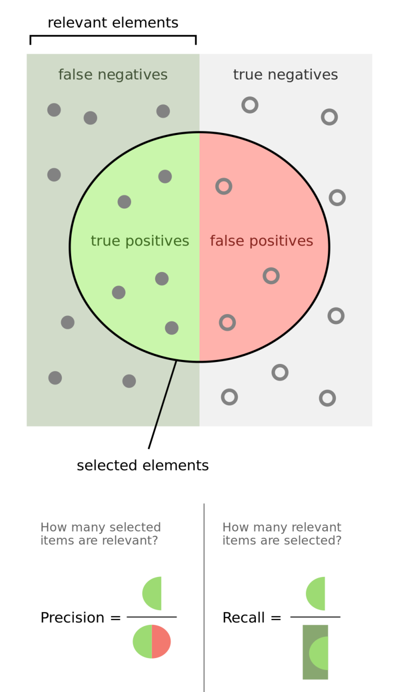
(quite confusingly, even if we use a logistic regression for the latter)

Classification tasks

For many computational approaches, we are actually not that interested in predicting a continous 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?

http://commons.wikimedia.org/wiki/File:Precisionrecall.svg



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 classfication ("positive"/"negative"), but it doesn't really matter whether there are two or more labels)

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

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.)

Supervised Machine Learning for Text Classification

Diving into SML

Supervised Machine Learning for Text Classification

An implementation

An implementation

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

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

And a second list with an identical structure:

```
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

```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer
from sklearn import metrics

# This is just an efficient way of computing word counts
vectorizer = CountVectorizer(stop_words='english')
train_features = vectorizer.fit_transform([r[0] for r in reviews])
test_features = vectorizer.transform([r[0] for r in test])

# Fit a naive bayes model to the training data.
nb = MultinomialNB()
nb.fit(train_features, [r[1] for r in reviews])

# Now we can use the model to predict classifications for our test features.
predictions = nb.predict(test_features)
actual=[r[1] for r in test]

print("Precision: {0}".format(metrics.precision_score(actual, predictions, pos_label=1, labels = [-1,1])))
```

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

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

This returns, as you would expect and hope:

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

But we can do even better

We can use different vectorizers and different classifiers.

Supervised Machine Learning for Text Classification

Classifiers

Different classifiers

Typical options in a nutshell:

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

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
```

Summing up

What *is* our fitted classifier again?

Essentially, just a formula

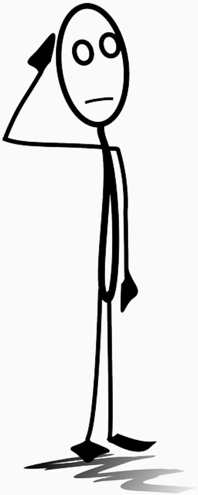
$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n)}}$$

where β_0 is an intercept[1], β_1 a coefficient for the frequency (or tf·idf score) of some word, β_2 a coefficient some other word.

If our fitted *vectorizer* contains 5,000 words, we thus have 5,001 coefficients.

~~(for logistic regression in this case, but same argu~~ment applies to other classifiers as well)

[1] Machine Learning people sometimes call the intercept "bias" (yes, I know, that's confusing)



But isn't that then essentially very much like a dictionary, except that the words have different weights?

In some sense, yes.

- But we don't pretend that we can construct the dictionary *a priori*.
- It's specifically tailored to our use-case.
- The weights are *really* essential here.

We *could* print all coefficients-word pairs, but probably it's enough to just look at those with the largest absolute value:

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 earthly and celestial eminent domains in the american west withholds the fairytale literalness that marked its german predecessor in the ad hoc genre of

ELI5

- Inspecting *all* coefficients of a ML model usually doesn't make much sense
- But that does not mean that we cannot understand how the model makes its predictions
- We can look at the most important coefficients
- We can look which words in a given text contributed most to its classfication

But have we solved all problems of dictionaries?

No.

For instance, the negation and/or intensifier problem.

Possible approaches

- *n*-grams as features
- preprocessing (?)
- deep learning
- ...

$\Rightarrow$ **But ultimately, it's just an empirical question how big the problem is!**

Summing up

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

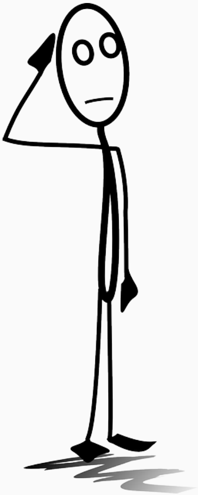
- **It does not matter *how* you create `y` and `X`!**
- Getting data into the right shape can be as much work (or more) as training the classifier itself

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)

Looking forward: Beyond classic SML

Note that classic SML is still based on word frequencies with weights (and hence cannot solve all problems we started off with). State-of-the art approaches like deep learning and transformers address this issue – but that's for another time.



Any questions?

Next steps

Thursday: time for your individual questions about the final project.

I prepared exercises to work on *during* the Thursday meeting (alone or in teams):
https://github.com/uvacw/teaching-bdaca/blob/main/6ec-course/week07/exercises/

References

Boukes, M., van de Velde, B., Araujo, T., & Vliegenthart, R. (2020). What's the Tone? Easy Doesn't Do It: Analyzing Performance and Agreement Between Off-the-Shelf Sentiment Analysis Tools. *Communication Methods and Measures*, *14*(2), 83–104. https://doi.org/10.1080/19312458.2019.1671966

Boumans, J. W., & Trilling, D. (2016). Taking stock of the toolkit: An overview of relevant autmated content analysis approaches and techniques for digital journalism scholars. *Digital Journalism*, *4*(1), 8–23. https://doi.org/10.1080/21670811.2015.1096598

Burscher, B., Odijk, D., Vliegenthart, R., de Rijke, M., & de Vreese, C. H. (2014). Teaching the computer to code frames in news: Comparing two supervised machine learning approaches to frame analysis. *Communication Methods and Measures*, *8*(3), 190–206. https://doi.org/10.1080/19312458.2014.937527

Burscher, B., Vliegenthart, R., & De Vreese, C. H. (2015). Using supervised machine learning to code policy issues: Can classifiers generalize across contexts? *The ANNALS of the American Academy of Political and Social Science*, *659*(1), 122–131. https://doi.org/10.1177/0002716215569441

Hopkins, D. J., & King, G. (2010). A method of automated nonparametric content analysis for social science. *American Journal of Political Science*, *54*(1), 229–247.

Hutto, C. J., & Gilbert, E. (2014). Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Eighth International AAAI Conference on Weblogs and Social Media*.

Pennebaker, J. W., Booth, R. J., & Francis, M. E. (2007). Linguistic Inquiry and Word Count: LIWC.

Thelwall, M., Buckley, K., & Paltoglou, G. (2012). Sentiment strength detection for the social web. *Journal of the American Society for Information Science and Technology*, *63*(1), 163–173. https://doi.org/10.1002/asi.21662

van Atteveldt, W., van der Velden, M. A., & Boukes, M. (2021). The Validity of Sentiment Analysis:Comparing Manual Annotation, Crowd-Coding, Dictionary Approaches, and Machine Learning Algorithms. *Communication Methods and Measures*, *00*(00), 1–20. https://doi.org/10.1080/19312458.2020.1869198

Vermeer, S., Araujo, T., Bernritter, S. F., & van Noort, G. (2019). Seeing the wood for the trees: How machine learning can help firms in identifying relevant electronic word-of-mouth in social media. *International Journal of Research in Marketing*, *36*(3), 492–508. https://doi.org/10.1016/j.ijresmar.2019.01.010