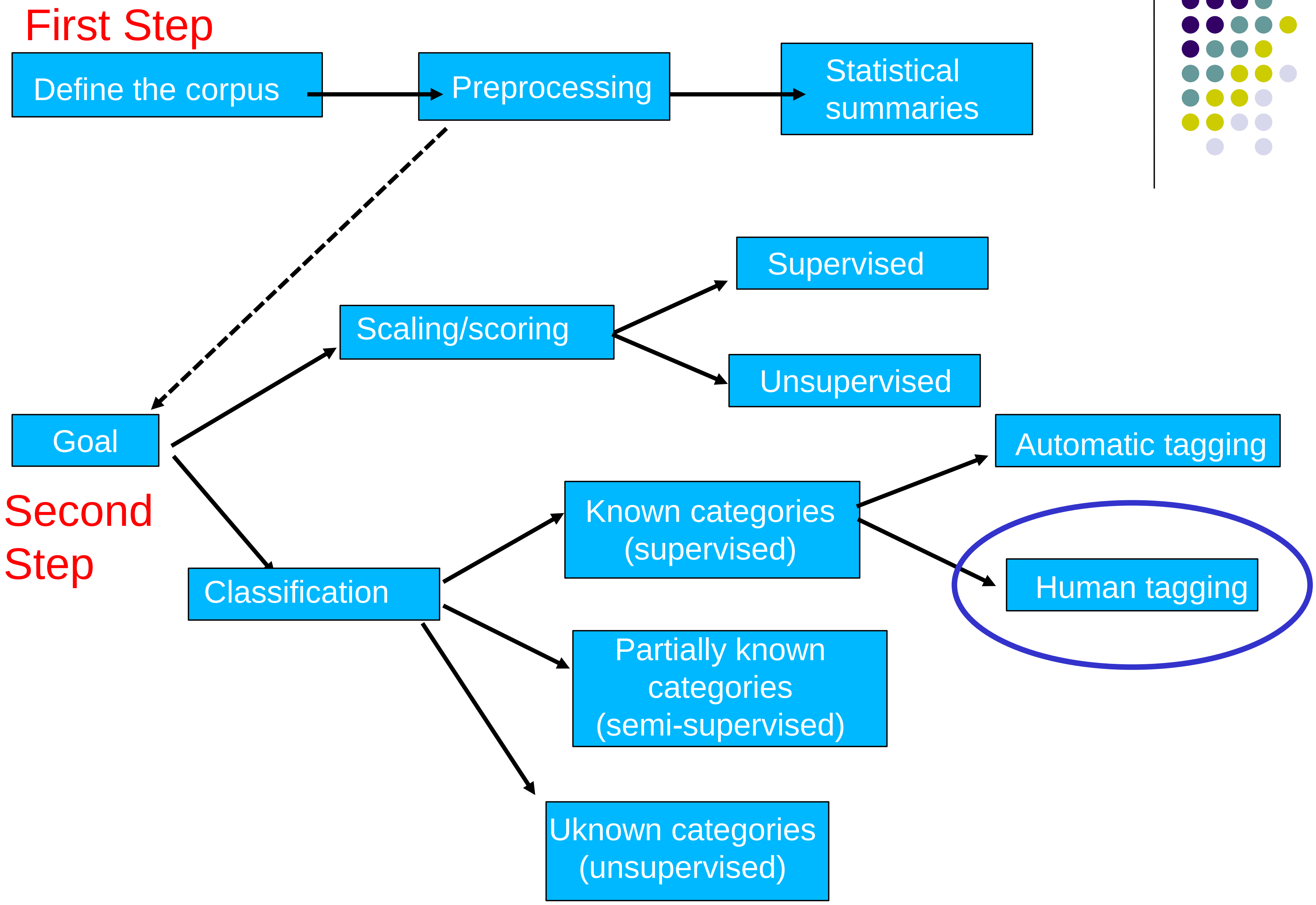
Big Data Analytics

Lecture 4/C

The importance of the training-set

UNIVERSITÄT LUZERN


First Step
Define the corpus
Preprocessing
Statistical summaries
Goal
Second Step
Scaling/scoring
Supervised
Unsupervised
Classification
Known categories (supervised)
Automatic tagging
Human tagging
Partially known categories (semi-supervised)
Uknown categories (unsupervised)

Reference

- ✓ Grimmer, Justin, and Stewart, Brandon M. (2013). Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts. *Political Analysis*, 21(3): 267-297
- ✓ Curini, Luigi, and Robert Fahey (2020). Sentiment Analysis and Social Media. In Luigi Curini and Robert Franzese (eds.), *SAGE Handbook of Research Methods is Political Science & International Relations*, London, Sage, chapter 29
- ✓ Barberá, Pablo et al. (2020) Automated Text Classification of News Articles: A Practical Guide, *Political Analysis*, DOI: 10.1017/pan.2020.8

Constructing a training set

For supervised problems, the researcher is aiming to classify documents into a set of known or assumed categories based upon rules or information that can be learned from the training set

This requires **labels** in the training set from which to infer categories in the test set

The most important step in applying a supervised learning algorithm is therefore constructing a **reliable training set**, because no statistical model can repair a poorly constructed training set!

If the training set is **poorly constructed**, the supervised algorithms will simply replicate such poorly construction and there is no way that any cross-validation exercise will be of any help here!

Constructing a training set

(1) creating and executing a coding scheme:

Best practice is to **iteratively develop coding schemes**

Initially, a **concise codebook** is written to guide coders, who then apply the codebook to an initial set of documents

For example, suppose you want to code the FB posts of politicians as either employing a *populist language* or otherwise. Accordingly, you need to advance a clear definition of populist language, with possible some examples, to show to your coders

When using the codebook, particularly at first, coders are likely to **identify ambiguities** in the coding scheme (for example: *how to classify a post that discusses about political corruption, but that praises the role of science?*)

Constructing a training set

(1) creating and executing a coding scheme:

While doing this, always define a number of exhaustive (and exclusive) categories – **no overlooked categories** should be present!

Suppose you want to classify tweets discussing about ISIS as either *positive, negative or neutral*. Then suppose that in the training-set you discover a sub-set of tweets that uses the hashtag #Isis as a way to make more viral tweets on a completely different topic. In this instance, you could include a category *off-topic* to label such tweets. In a different situation, you can also decide to include a category "*Others*", wherein classifying the texts that do not deal with any of the theoretically interesting categories you have identified for your research

ML algorithms must learn to classify also those categories!

Constructing a training set

(1) creating and executing a coding scheme:

This subsequently leads to a **revision of the codebook**, which then needs to be applied to a new set of documents to ensure that the ambiguities have been sufficiently addressed

Only after coders apply the coding scheme to documents without noticing ambiguities is a “final” scheme ready to be applied to the data set

Constructing a training set

(2) **sampling documents:**

Basically all ML methods aiming at individual classification implicitly assume that the **training set is a random sample** from the population of documents to be coded

This is because Supervised learning methods **use the relationship** between the features in the training set to classify the remaining documents in the test set (out-of-sample predictions)

Constructing a training set

(2) **sampling documents:**

This presents particular difficulty when…

…**all the data are not available at the time of coding**: either because it will be produced in the future or because it has yet to be digitized

For example, you want to classify the FB posts over an electoral campaigning, but you sample your training-set at the beginning of that campaigning

Per-se, this could be particularly problematic in dealing with any ***semantic change***, which is the difference in the meaning of language between the training and the test set

Constructing a training set

(2) **sampling documents:**

In particular, we can have **emergent discourse**, where new words and phrases, or the meanings of existing words and phrases, appear in the test set but not in the training set

…and **vanishing discourse**, where the words, phrases, and their meanings exist in the labeled/training set but not the unlabeled/test set

How to face this risk?

Keep updating the training-set (if your test-set is still to come...)!

Constructing a training set

(2) **sampling documents:**

Moreover, Supervised methods need **enough information** to learn the relationship **between words and documents in each category of a coding scheme**

Hopkins and King (2010) offer **five hundred** as a rule of thumb with one hundred documents for each class-label probably being enough

Constructing a training set

(2) **sampling documents:**

Still the number necessary will depend upon:

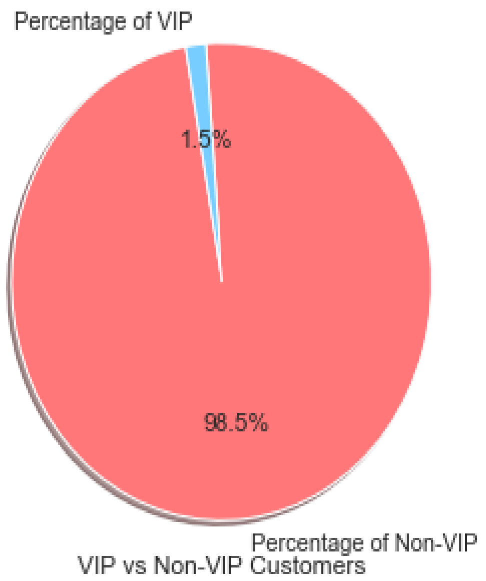
a) **the specific application of interest**. For example, as the number of categories in a coding scheme increases, the number of documents needed in the training set also increases

Constructing a training set

(2) **sampling documents:**

b) Moreover, beware of very imbalanced training-set (i.e., a training-set that contains many more samples from one class than from the rest of the classes)

Indeed, if you have a very imbalanced data set you could have a very hard day with any ML algorithm. Why?



Constructing a training set

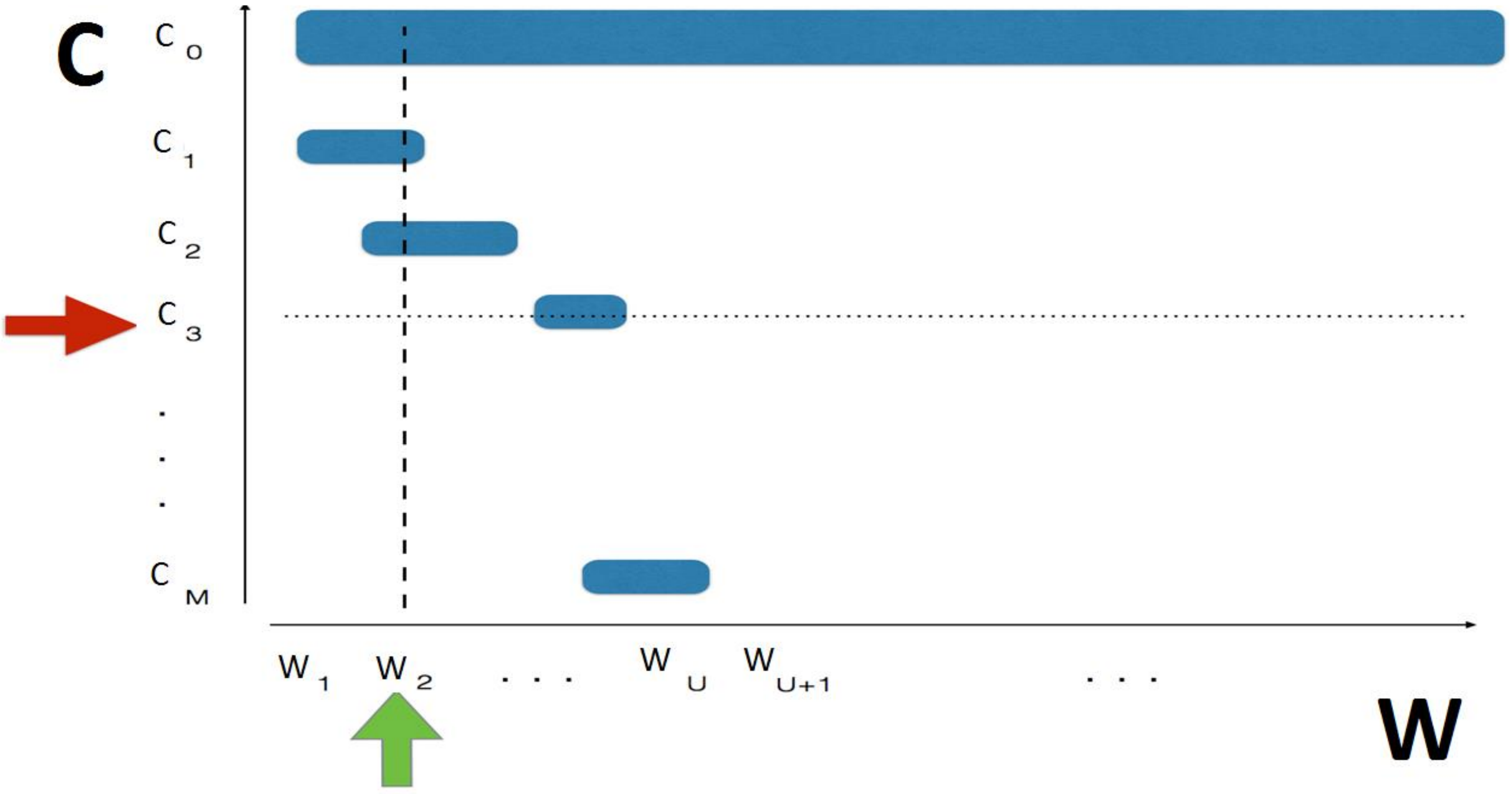
(2) **sampling documents:**

b) In this scenario, classifiers can have *good accuracy on the majority class* but very poor accuracy on *the minority class(es)* due to the influence that the larger majority class produces – i.e., the model will perform badly because the model is not trained on a sufficient amount of data representing the minority class(es) given that there is insufficient opportunity to “learn” about this category and its properties

This will greatly affect negatively your out-of-sample prediction!

Constructing a training set

The existence of a category C_k extremely frequent in a training-set can negatively affect $p(\boldsymbol{C}|\boldsymbol{W})$



The imbalanced data-set riddle

And so what to do?

When attempting to detect **small changes or rare categories**, *therefore*, increasing the probability that they are observed in the training set often means increasing the size of the training set relative to the test set to decrease the overall level of class imbalance

And if you cannot? As a second-best strategy, you can always try to **resample** the original training dataset

The imbalanced data-set riddle

Resampling is done either by *oversampling* the minority class and/or *under-sampling* the majority class until the classes are approximately equally represented

Even though both approaches address the class imbalance problem, they also suffer some drawbacks

The random undersampling method can potentially remove certain important data points (and therefore information!), and random oversampling can lead to overfitting

The imbalanced data-set riddle

Other possibility: Synthetic data generation such as…

SMOTE: Synthetic Minority Over-sampling Technique has been designed to generate new samples that are coherent with the minor class distribution

The main idea is to consider the relationships that exist between samples and create new synthetic points along the segments connecting a group of neighbors

Constructing a training set

(3) **checking human-tagging reliability:**

While labeling training data requires the use of human coders to sort texts into desired categories, human coding lacks consistency and reliability both within and across individuals, above and beyond the time and expense required to complete the task

Therefore always run an **inter-coder reliability test**!!!

Constructing a training set

(3) **checking human-tagging reliability:**

What is **inter-coder (or inter-rater) reliability**?

Intercoder reliability is the extent to which 2 different researchers agree on how to code the same content

It’s often used in content analysis when one goal of the research is for the analysis to aim for consistency and validity

Intercoder reliability ensures that when you have multiple researchers coding a set of data, that they come to the same conclusions

Constructing a training set

One common statistics used is Cohen's kappa coefficient (κ)

k is a more robust measure than simple percent agreement calculation, as it takes into account the possibility of the agreement occurring by chance

For example, if you have 2 coders, and one of them is doing a good job in coding, while the other is always choosing the class label completely at random, you are going still to get some percent agreement between the two coders

However this percent agreement would occur just by chance!

Constructing a training set

K is estimated as $(p_o-p_e)/(1- p_e)$

where $p_o$is the relative observed agreement among coders (identical to accuracy), and p_e is the hypothetical probability of chance agreement, using the observed data to calculate the probabilities of each observer randomly seeing each category

If the coders are in complete agreement then k=1. If there is no agreement among the coders other than what would be expected by chance (as given by p_e), k=0. It is possible for the statistic to be negative, which implies that the agreement is worse than random

Usually a reasonable value for k is larger than .6 (but larger than .8 would be far better)

Constructing a training set

Let’s see an example, with 2 coders, 2 categories, and 50 texts to code for each coders

	Coder B	
Coder A	Positive	Negative
Positive	20	5
Negative	10	15

Observed proportionate agreement (p_o): (20+15)/50=0.7

Is it good or bad? Wait a minute…

Constructing a training set

	Coder B	
Coder A	Positive	Negative
Positive	20	5
Negative	10	15

Which is the probability of a random agreement (p_e)?

- ✓ Coder A said “Positive" to 25 texts and “Negative" to 25 texts. Thus reader A said “Positive" 50% of the time.
- ✓ Coder B said “Positive" to 30 texts and "Negative" to 20 texts. Thus coder B said “Positive" 60% of the time

So the expected probability that both would say “Positive” at random is: 0.5*0.6=0.3

Similarly, the expected probability that both would say “Negative” at random is: 0.5*0.4=0.2

Overall random agreement probability is the probability that they agreed on either Positive or Negative, i.e. (p_e)=0.3+0.2=0.5

Constructing a training set

	Coder B	
Coder A	Positive	Negative
Positive	20	5
Negative	10	15

Applying the formula for Cohen's Kappa we get:

✓ $k=(p_o-p_e)/(1- p_e)=(0.7-0.5)/(1-0.5)=0.4$…

….therefore not such a great result!

Constructing a training set

The golden-rule for a good training-set, a brief resume:

1. Develop a good coding book!
2. Sample good your training-set!
3. Check (always) inter-coder reliability! So for example, if you have 1,000 texts in your training-set and 2 coders, always be sure that a sub-sample (say 100 hundreds) of the texts that will be coded by the coders actually overlap among themselves, so that you can run an inter-coder reliability test!