Big Data Analytics

Lecture 4/B

Cross-Validation & the importance of a good training-set

UNIVERSITÄT LUZERN

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
- ✓ Cranmer, Skyler J. and Desmarais, Bruce A. (2017) What Can We Learn from Predictive Modeling?, *Political Analysis*, 25: 145-166
- ✓ Barberá, Pablo et al. (2020) Automated Text Classification of News Articles: A Practical Guide, *Political Analysis*, DOI: 10.1017/pan.2020.8

Validation


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Validation

Since the ultimate goal of supervised learning is to find generalizable patterns of association, models must be **selected based on their ability to generate good out-of-samples predictions** (i.e., good predictions on the test-set)

However, it is impossible to evaluate a model's performance on the universe of *test-set documents* (i.e., you do not know by definition their “true” class-labels after all!), so an approximate measure of performance must be devised

Validation

Supervised methods are designed to automate the hand coding of documents into categories as we have already noticed

If a method is performing well, it will therefore **directly replicate the hand coding**. If it performs poorly, it will fail to replicate the coding – instead introducing serious errors

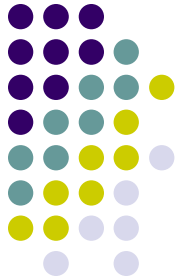
This clear objective implies a **clear standard for evaluation**: comparing the output of machine coding to the output of hand coding. From here the idea of **validation**!

Validation

The *ideal validation* procedure would divide the data into **three subsets**

1. Initial model fitting would be performed on the training-set
2. Once a final model is chosen, *a second set of hand-coded* documents - the *validation set* - would be used to assess the performance of the model
3. The final model would then be applied to the test to complete the classification

Validation



This approach to validation is difficult to apply in most settings. But **cross-validation** (also called: **K-fold validation**) can be used to replicate this ideal procedure

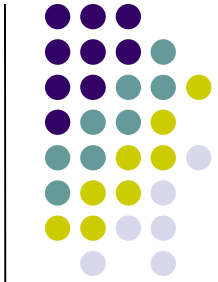
Validation

In K-fold cross-validation, the training set is randomly partitioned into some groups (say two: K1 and K2)

For each group, the first model is trained on K1, then applied to the K2 to assess performance; similarly a model is trained on the K2 and then applied to K1 to assess performance

Then you take the average across the results you get in the two scenarios

Validation



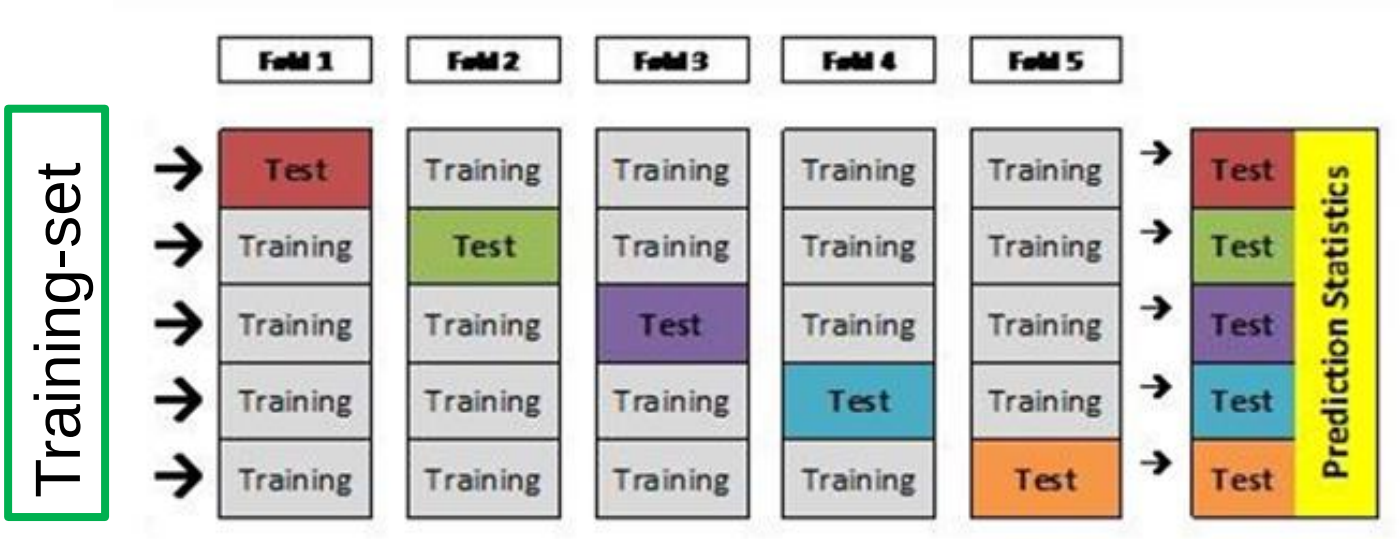
And if you want to run a K-fold cross-validation with K larger than 2?

The algorithm is as follow:

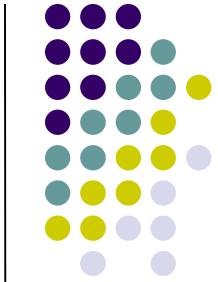
1. Randomly split the data set into k-subsets (or k-fold) (for example 5 subsets)
2. Reserve one subset and train the model on all other subsets (4 in this case)
3. Test the model on the reserved subset and record the prediction error
4. Repeat this process until each of the k subsets has served as the test set
5. Compute the average of the k recorded errors. This is called the **cross-validation error** serving as the performance metric for the model

Validation

So, for example, with K-fold cross-validation=5…



Validation



How to choose right value of k?

Lower value of k is more biased and hence undesirable. On the other hand, higher value of k is less biased, but can suffer from large variability

In practice, one typically performs k-fold cross-validation using k = 5 or k = 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance

Validation

REMEMBER: **Cross-validation** is the **only way** to control if the ML algorithm you are using is doing a good job or not (unless you are ready to believe in that by fiat)!

True: in Random Forest we can also focus on the OOB sample, still the final results are not reliable as via cross-validation (i.e., you focus only on those rows not included in the bootstrapped trees, while by doing CV you are ensured that all the rows, i.e., documents of your corpus, are included in the analysis)

Validation

Moreover, cross-validation **avoids overfitting** by focusing on **out-of-sample prediction** and **selects the best model** for the underlying data from a set of candidate models

Validation

Which statistics (or **performance metrics**) should we use to *assess model performance*?

There are several of them, but we are going to focus on three metrics for individual classifiers with text-analysis

Validation

Accuracy: proportion of correctly classified documents

While of course we want this score to be as high as possible, it can also be important to look at the two components which make up that score, known as **recall** and **precision**

Note moreover that accuracy refers to the **overall model performance**. Recall and precision refers to the performance of your model with respect to a **specific category k**

Validation

Recall or **Sensitivity** (for a category k) is a measure of what proportion of documents of a given category the algorithm correctly identified (i.e., *minimize false negatives*)

For example, if there were 10 documents of the category "positive" in the data set, and the algorithm correctly identified 8 of them, we would say that this algorithm has "recall of 0.8 for the category *positive*"

- ✓ given that a human coder labels a document as belonging to category k, what is the chance the machine identifies the document?

Validation

Precision or **Positive Predictive Value** (for a category k) on the other hand is a measure of how many of the times the algorithm identified a category were actually correct, as against how many times were false positives (i.e. *minimize false positives*)

In the previous example, where the algorithm correctly identified 8 of the 10 documents as *positive*, perhaps the algorithm also miss-identified 4 other documents as *positive* - so 8 out of its 12 *positive* classifications were correct, allowing us to say that it has a “precision of 0.667 for the category *positive*”

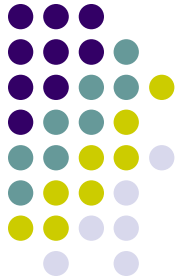
- ✓ given that the machine guesses category k, what is the probability that the machine made the right guess?

Validation

If you find market differences between the **recall and precision** (for example, with a recall rate >> precision) implies that your algorithm guesses too often that a document belongs to category k

The result is that it labels a large portion of the human coder's as *k* correctly (and so you have a **high recall rate**). But it also includes several documents that humans label differently (and so you have a **low precision**)

Validation



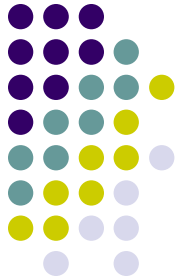
The aggregate of the recall and precision scores for a category is known as the **f1 score**

More precisely, the traditional F-measure or balanced F-score (**f1**) is the harmonic mean of precision and recall:

f1 = (2 * precision * recall) / (precision + recall)…

…where the highest level of performance (of f1) is equal to 1 and the lowest 0

Validation



The **average of the f1 scores for all the categories** is another reasonable rough measurement of the performance of the algorithm (and often more than accuracy alone as we will see below!)

However, before using the algorithm for any serious analysis work, it is always *advisable also to take a look* at the precision and recall scores for individual categories - you may find that a category you are planning to use in your analysis actually has very high rate of false-positive or false-negative identifications, which could cause serious problems for your results

Let’s an example on how to compute the statistics we discussed up to now from the ***Confusion matrix***

Performance metrics

Confusion matrix

	Actual label	
Classification (algorithm)	Black	White
Black	True Black	False Black
White	False White	True White

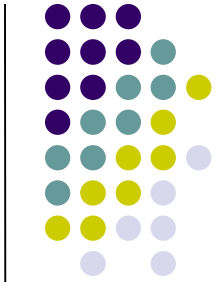
$$\text{Accuracy} = \frac{\text{TrueBlack} + \text{TrueWhite}}{\text{TrueBlack} + \text{TrueWhite} + \text{FalseBlack} + \text{FalseWhite}}$$

$$\text{Precision}_{\text{Black}} = \frac{\text{TrueB}}{\text{TrueB} + \text{FalseB}}$$ → Think horizontally

$$\text{Recall}_{\text{Black}} = \frac{\text{TrueB}}{\text{TrueB} + \text{FalseW}}$$ → Think vertically

$$f_{1\text{Black}} = \frac{2 * \text{precisionB} * \text{recallB}}{\text{precisionB} + \text{recallB}}$$

Performance metrics



	Actual label	
Classification (algorithm)	Black	White
Black	800	100
White	50	50

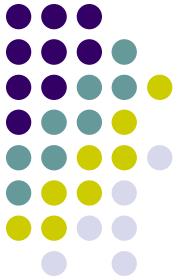
$$\text{Accuracy} = \frac{800 + 50}{800 + 50 + 100 + 50} = 0.85$$

$$\text{Precision}_{\text{Black}} = \frac{800}{800 + 100} = 0.88$$

$$\text{Recall}_{\text{Black}} = \frac{800}{800 + 50} = 0.94$$

$$f_{1\ \text{Black}} = \frac{2 * .88 * .94}{.88 + .94} = 0.91$$

Performance metrics



Confusion matrix

	Actual label	
Classification (algorithm)	Black	White
Black	True Black	False Black
White	False White	True White

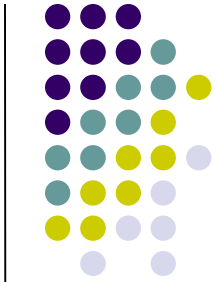
$$\text{Accuracy} = \frac{\text{TrueBlack} + \text{TrueWhite}}{\text{TrueBlack} + \text{TrueWhite} + \text{FalseBlack} + \text{FalseWhite}}$$

$$\text{Precision}_{\text{White}} = \frac{\text{TrueW}}{\text{TrueW} + \text{FalseW}} \longrightarrow \text{Think horizontally}$$

$$\text{Recall}_{\text{White}} = \frac{\text{TrueW}}{\text{TrueW} + \text{FalseB}} \longrightarrow \text{Think vertically}$$

$$f_{1\text{White}} = \frac{2 * \text{precisionW} * \text{recallW}}{\text{precisionW} + \text{recallW}}$$

Performance metrics



	Actual label	
Classification (algorithm)	Black	White
Black	800	100
White	50	50

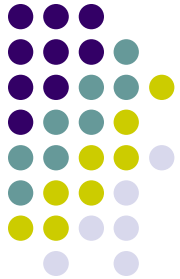
$$\text{Accuracy} = \frac{800 + 50}{800 + 50 + 100 + 50} = 0.85$$

$$\text{Precision}_{\text{White}} = \frac{50}{50 + 50} = 0.5$$

$$\text{Recall}_{\text{White}} = \frac{50}{50 + 100} = 0.33$$

$$f_{1\ \text{White}} = \frac{2 * .5 * .33}{.5 + .33} = 0.39$$

Performance metrics



In this example you are going to have a single Accuracy value=0.85

However, you could also take the **average of the F1 scores** for the classes as another measure of the performance of the algorithm

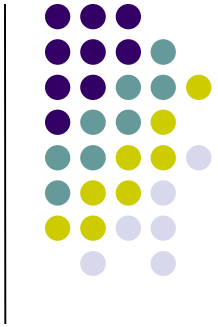
In our case: (.91+.39)/2=.65

You see the difference here between Accuracy and the averaged F1 score!

This difference is due that we are doing well with the Black class, but relative poorly with the White class

Rule-of-thumb: everytime you notice a market difference between Accuracy and F1 score, there are problems for your model!!!

Performance metrics



Let's see another example

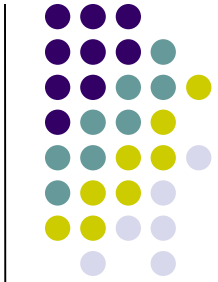
	Actual label	
Classification (algorithm)	Cats	Dogs
Cats	95	4
Dogs	1	1

$$\text{Accuracy} = \frac{95 + 1}{95 + 4 + 1 + 1} = 0.95$$

Accuray seems pretty high, but compared to a ***natural benchmark*** (i.e., to a dull algorithm that simply assigns all the observations to the most frequent class - here cats)?

In this case a random draw would produce once again 95% of Accuracy (i.e., 96/101)! Not so impressive after all…

Performance metrics



Let's see another example

	Actual label	
Classification (algorithm)	Cats	Dogs
Cats	95	4
Dogs	1	1

$$\text{Accuracy} = \frac{95 + 1}{95 + 4 + 1 + 1} = 0.95$$

Moreover, stark contrast between the two classes:

$\text{Precision}_{\text{Cats}} = 95/99 = .96$

$\text{Recall}_{\text{Cats}} = 95/96 = .99$

$F1_{\text{Cats}} = .97$

$\text{Precision}_{\text{Dogs}} = 1/2 = .5$

$\text{Recall}_{\text{Dogs}} = 1/5 = .2$

$F1_{\text{Dogs}} = .29$

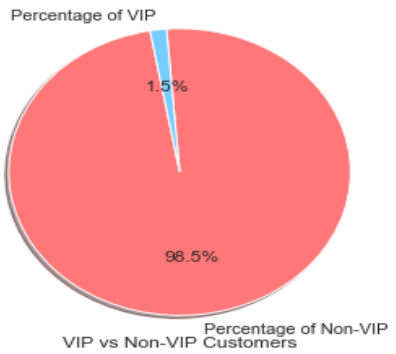
Avg. F1=.63

The imbalanced data-set riddle

In this example (as in the previous one…), Accuracy does not turn to be a reliable metric for the real performance of a classifier

This happens with a greater likelihood precisely when your **data set is highly unbalanced** (i.e., a data 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?



The imbalanced data-set riddle

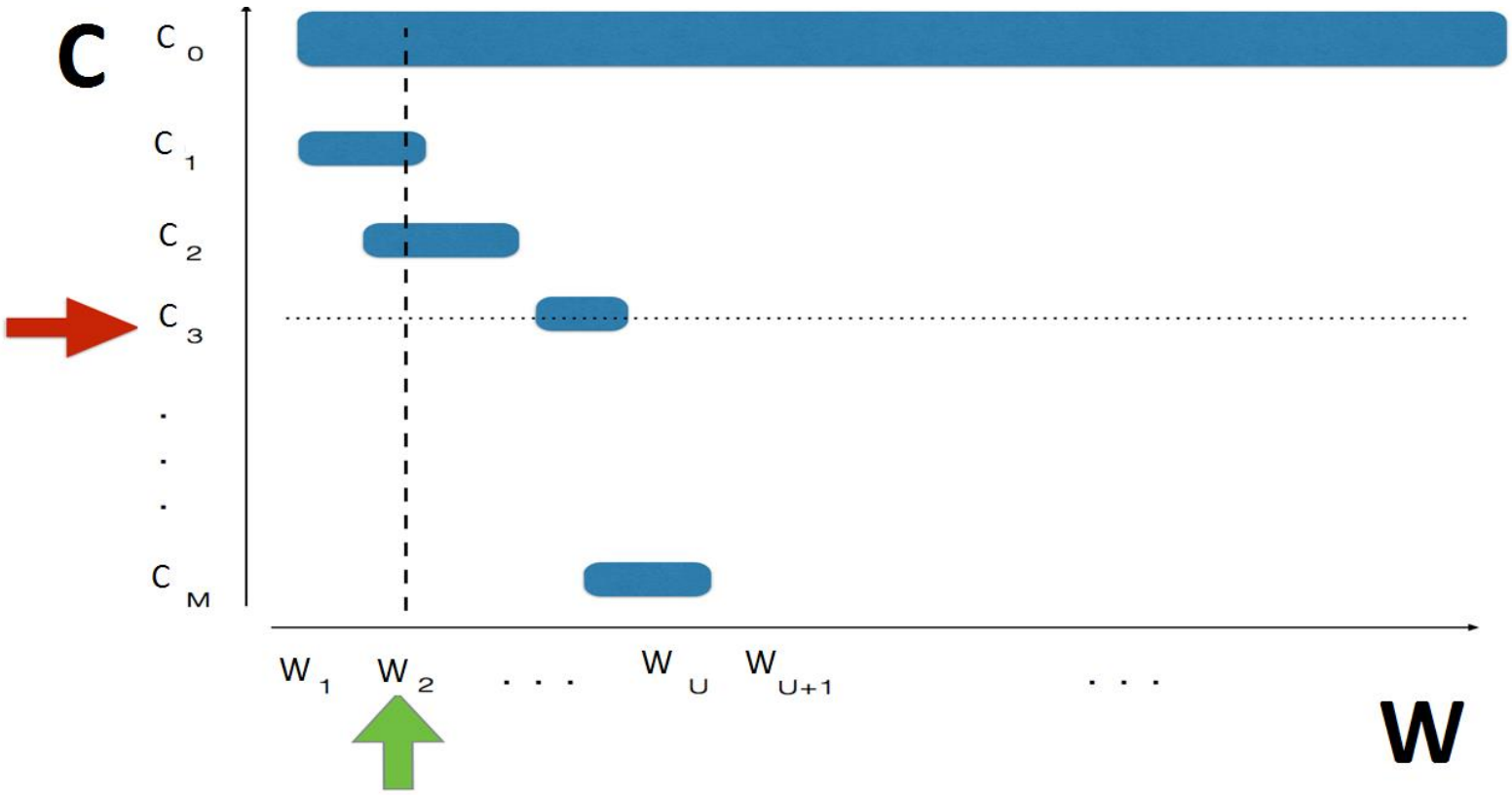
In this scenario, classifiers can have a good accuracy on the majority class but a very poor accuracy on the minority class(es) due to the influence that the larger majority class produces

That is, the model will perform badly because the model is not trained on a sufficient amount of data representing the minority class(es), so a particular classifier **might classify almost all the observations as belonging to the majority class** (as cats in our previous example)

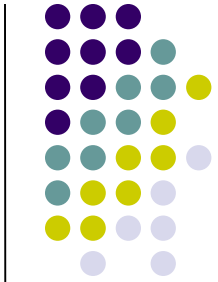
This of course will affect negatively your out-of-sample prediction!

The imbalanced data-set riddle

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

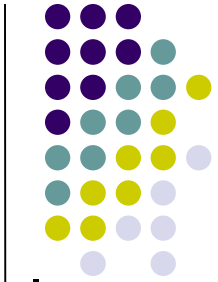


And so?

Best strategy: go back to your training-set and improve on it by collecting more texts for the minority categories 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

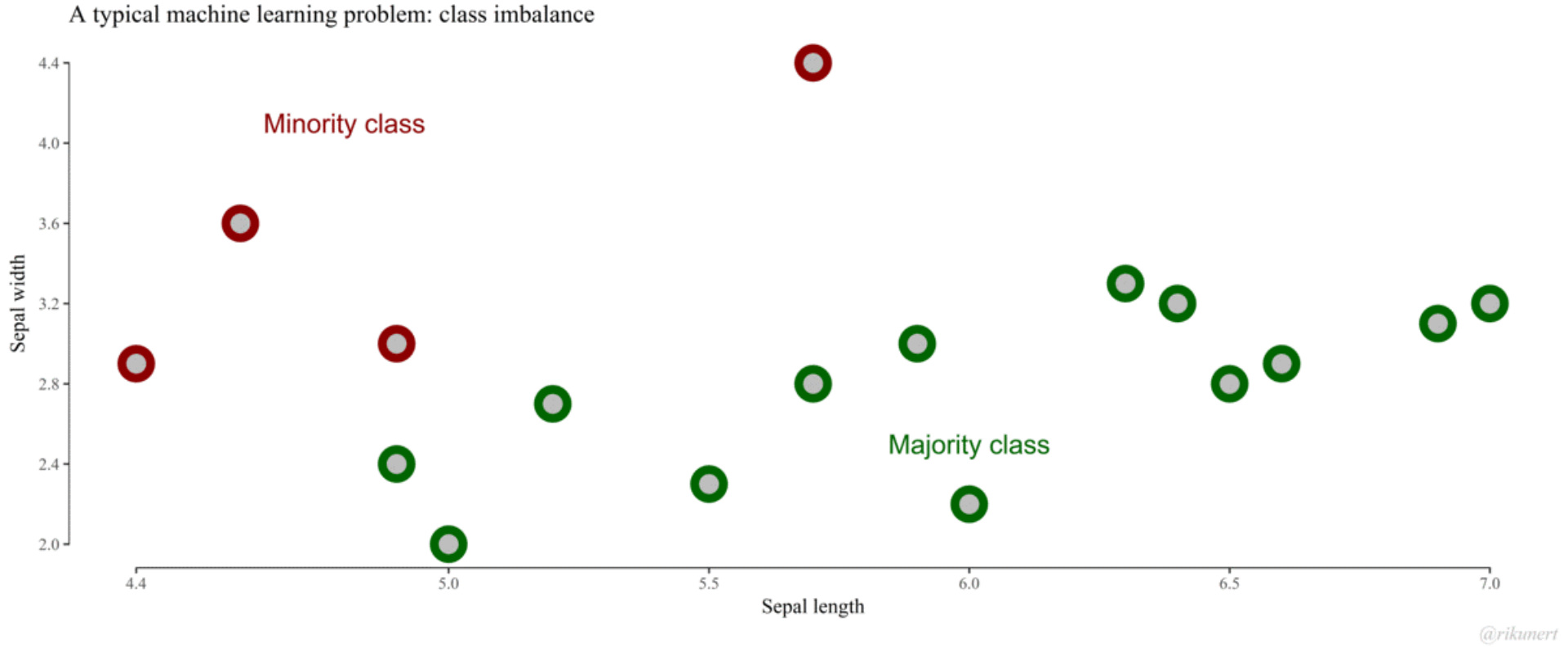
The imbalanced data-set riddle

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

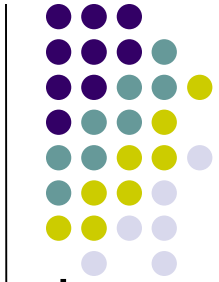
The main idea is to consider the relationships that exist between samples and create new synthetic points along the segments connecting a group of neighbors

The imbalanced data-set riddle



An example with 2 features

The imbalanced data-set riddle

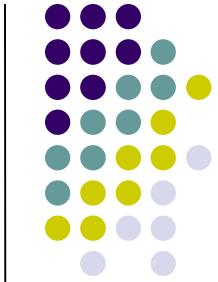


The SMOTE algorithm works directly on the DfM, by imputing values to new «rows» for each feature includeded in the DfM

An alternative to SMOTE when dealing with text classification is to use some NLP algorithm that learns the relationship between words in the texts related to a given class, and then producing some new synthetic texts related to that class

If you are interest in this topic, drop me an email to get an example!

Validation: another example with 3 categories



		Actual class		
		Cat	**Dog**	**Rabbit**
Predicted class	**Cat**	**5**	2	0
	Dog	3	**3**	2
	Rabbit	0	1	**11**

Here **Accuracy** is equal to the ratio between the sum of the diagonal (i.e., the sum of «True Positive») and the total number of observations, i.e., (5+3+11)/(5+2+3+3+2+1+11)=0.704

Validation

For each category *k* we can move from here

		Actual class		
		Cat	**Dog**	**Rabbit**
Predicted class	**Cat**	**5**	2	0
	Dog	3	**3**	2
	Rabbit	0	1	**11**

to here (example for the “cat” category)

		Actual class	
		Cat	**Non-cat**
Predicted class	**Cat**	5 True Positives	2 False Positives
	Non-cat	3 False Negatives	17 True Negatives

Validation

Then:

In the case, Precision for the cat class is: 5/(5+2)=0.71

Recall for the cat class is: 5/(5+3)=0.625

f1 for the cat class is: 2*(0.625*071)/(0.625+071)=0.66

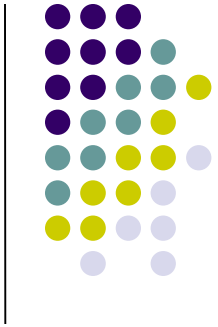
You can do the same thing for the dog and the rabbit cases, and then averaging across values to have a sense of the overall performance of your model

Validation

Depending on the application, scholars may conclude that the supervised method is able to sufficiently replicate human coders. A largely employed rule-of-thumb is getting accuracy>.85 for example (or f1>.75) (at least when you are dealing with *just 2 categories*)

Or, additional steps can be taken to improve accuracy, including trying to apply other ML algorithms or…

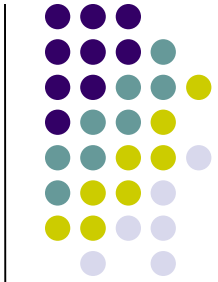
Validation



…as already underlined, most algorithms also have a range of "**hyper-parameters**" (or "*tuning parameters*") – assumptions and modifiers which are used to fine-tune the model and which can be set to different values prior to training – that can significantly impact performance (remember about the number of trees in RF)

Finding the right set of hyper-parameters for a certain task and a specific data set is also largely a case of trial and error, and it can only be done once again via cross-validation!

Validation



Some packages in R (such as `Caret` or `h2o` or the same `Quanteda` with the library `quanteda.classifiers`) provide ways to automate this task; this is known as a "grid search", allowing researchers to exhaustively search through every combination of a set of hyper-parameters to find the best performing model

This process can take a lot of time – often in the order of several hours for algorithms with complex sets of parameters – but often yields better performance than the default parameter set

Validation

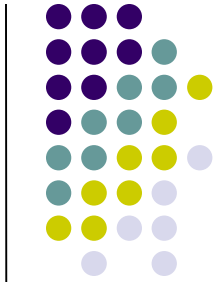
Summing up: the purpose of cross-validation is **model checking**!

Accordingly, cross-validation allows you to:

- ✓ **select among different machine-learning algorithms**...(remember the **No Free Lunch Theorem!** No machine learning algorithm is always better at predicting new, unobserved, data points universally)
- ✓ ..and to identify the **better hyperparameters setting for a given ML algorithm**

As a result always run a cross-validation before classifying the test-set to select the best ML algorithm given your corpus!

Validation: a summary



Two possible routes in this regard according to how you want to deal with the **hyper-parameters**:

First route (to success…)

a) You keep the default hyper-parameters of your ML algorithms;

b) you run a CV on each of such ML algorithms

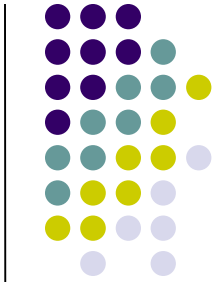
c) you select the one (or two) with the best performance on CV

d) you fine-tune the hyper-parameters on such model(s)

e) you re-run CV just on them

f) you keep the ML algorithm that performs better in the CV

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!

Constructing a training set

(1) creating and executing a coding scheme:

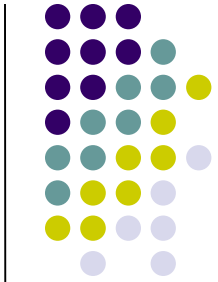
Best practice is to **iteratively develop coding schemes**

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



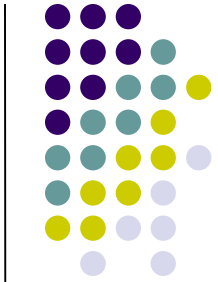
(1) creating and executing a coding scheme:

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

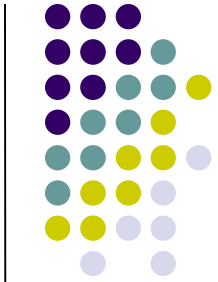


(1) creating and executing a coding scheme:

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

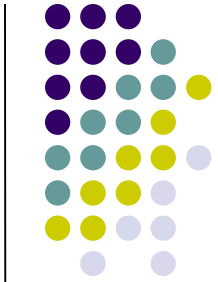


(2) **sampling documents:**

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



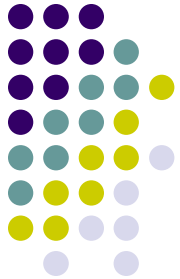
(2) **sampling documents:**

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

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 test sets

Constructing a training set



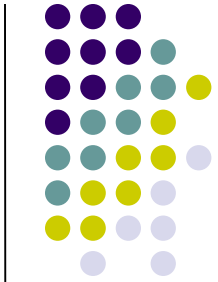
(2) **sampling documents:**

For example, 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 the training set

Constructing a training set

emergent discourse example:

Constructing a training set



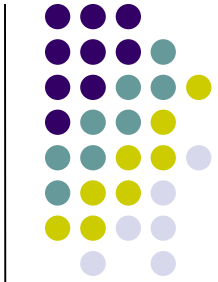
(2) **sampling documents:**

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

How to face this risk?

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

Constructing a training set

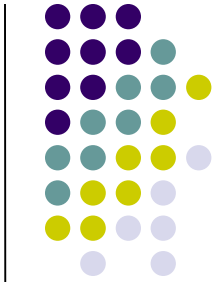


(2) **sampling documents:**

Furthermore, 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, if a category does not occur, or occurs extremely rarely, in the training set, there is insufficient opportunity to “learn” about this category and its properties, which will in turn interfere with the process of classifying test-set documents into this category correctly

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 (remember the *curse of highly imbalanced training-set*!)

That also means that you can decide to collapse two different categories into one to increase their volume (as far as these two categories are not your main theoretical focus!)

Constructing a training set

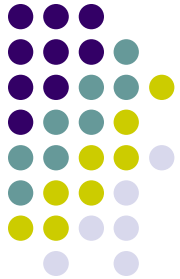
(2) **sampling documents:**

c) You also have to consider the risk of a ***lack of textual discrimination***. This happens where the language used in documents falling in the different pre-defined categories is not clearly distinguishable

Lack of textual discrimination among categories can occur because of *heterogeneity* in how authors express category-related information or a *divergence* between how authors of the documents express this information and how the analyst conceptualizes the categories

Also this factor affects the appropriate size of the training-set

Constructing a training set

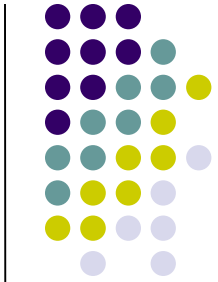


(3) **checking human-tagging reliability:**

While labeling training data often 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 text**!!!

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

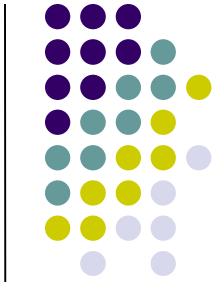
One common statistics used is Cohen's kappa coefficient (κ)

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 raters (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 raters are in complete agreement then k=1. If there is no agreement among the raters 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

✓ A good outcome? Well, not necessarily…

Constructing a training set

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

And the probability of a random agreement (p_e)?

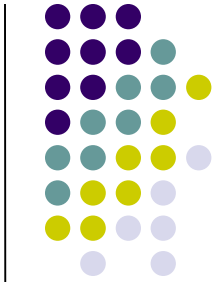
- ✓ Coder A said “Positive" to 25 texts and “Negative" to 25 texts. Thus reader A said “Positive" 50% of the time
- ✓ Coder B said “Positive" to 30 texts and "Negative" to 20 texts. Thus coder B said “Positive" 60% of the time

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

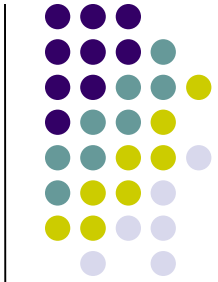
Applying the formula for Cohen's Kappa we therefore get:

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

✓ Not such a good outcome after all…

In R, you can use the `irr` package to easily compute Cohen's Kappa for a given training-set. If you are interested about it, drop me an email!

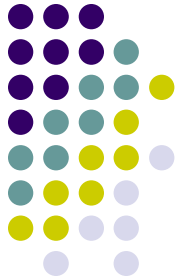
Constructing a training set



The golden-rules 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! Why not running with all the 1,000 texts the inter-coder reliability test? It would be a waste of time!

Constructing a training set



Remember: if you get a bad value for K can mean two different things:

- ✓ either one or both coders are doing a bad job
- ✓ or there is still some ambiguity in the coding book that generates disagreement between coders…

So if you get a bad inter-coder reliability statistics due to the latter reasons, what do you have to do?

Go back to the coding scheme to improve it!