Week 4: Classifying Texts into Known Categories

LSE MY459: Quantitative Text Analysis
https://lse-my459.github.io/

Ryan Hübert

Slides last updated: 10 February 2025

Outline for today

➜ Supervised learning overview
➜ Creating a labeled set and evaluating its reliability
➜ Classifier performance metrics
➜ Examples of classifiers: Naive Bayes
➜ Examples of classifiers: Regularised regression
➜ Classification after prediction
➜ Building a robust classifier
➜ Application: civility on Twitter

Supervised learning overview

Classifying into known categories

Our goal today: classify documents into known categories

We already (sort of) did this last week using ADMs

- ➜ Advantage of ADMs: not corpus-specific, so cost to apply to a new corpus is trivial
- ➜ Disadvantage of ADMs: not corpus-specific, so performance on a new corpus is unknown (**domain shift**)

Supervised learning is another approach:

- ➜ "Generalisation" of ADMs where associations between features and categories are learned from data (and *not* from dictionary)
- ➜ By construction, they will outperform dictionary methods for *classification tasks*, as long as "training sample" is large enough

ADMs versus supervised learning

Barberá et al. (2021) codes a set of >4,000 articles, showing ML generally does better than ADMs coding sentiment

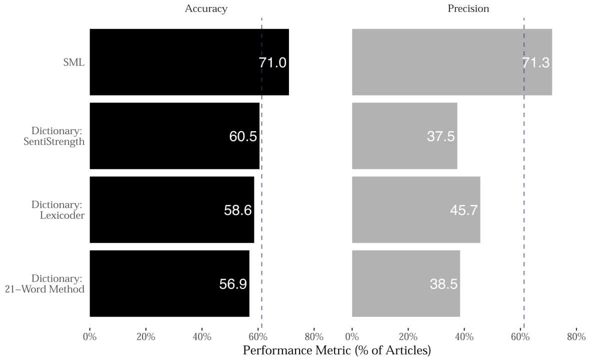


Figure 3. Performance of SML and Dictionary Classifiers—Accuracy and Precision.
Note: Accuracy (percent correctly classified) and precision (percent of positive predictions that are correct) for the ground truth dataset coded by ten CrowdFlower coders. The dashed vertical lines indicate the baseline level of accuracy or precision (on any category) if the modal category is always predicted. The corpus used in the analysis is based on the keyword search of *The New York Times* 1980–2011 (see the text for details).

Source: http://doi.org/10.1017/pan.2020.8

Backing up: why do this?

Classification with pre-defined categories is about *measurement*

A typical use case:

- Have large collection of documents that we want to partition into pre-defined categories
- Some documents are (or can be) **labeled**
- Many others are **unlabeled**

Supervised learning is using the labeled data to build a model that we can use to label the unlabeled data

- Idea: methods "supervised" by the existing labels

Can be useful for dealing with missing data (labels) in existing data and predicting categories in future data

Supervised v. unsupervised methods

What if you want to classify documents but don't know categories?

This is what **unsupervised learning** is about

- ➜ Identify similarities in documents based on statistical patterns, without requiring any "supervision" (human annotations)

Unsupervised learning is largely (but not always) about *discovery* and not *measurement* (next two weeks. . .)

Useful way to remember distinction:

- ➜ Supervised methods involve statistical models with "outcome" variables, like regressions
- ➜ Unsupervised methods do not have "outcome" variables

Classification v. scaling methods

Today our focus is on classification into discrete categories

But, this is just one way to use these methods

We can also use them to locate things (e.g. documents) on continous scales representing latent traits

This is known as **scaling**

More on this later, but keep in mind that you can “classify” onto continuous scales, even though it’s not our focus today

Several issues we need to cover

Suppose we have a corpus containing documents we want to classify

1. How do we create (or find) a set of labeled documents?
   - ➜ What are the categories of interest and what is the best way to label a subset of documents?
2. How do we do the classification?
   - ➜ What classifier do we use?
   - ➜ How do we ensure "robust" classification?
3. How do we validate whether our classifier worked well?
   - ➜ What quantitative performance metrics can we use to evaluate?

Basic workflow

1. Gather documents
2. Create a training set by, e.g., hand labeling a random sample
3. Preprocess documents
4. Transform the documents (to a dfm)
5. Choose a classifier and specify parameters
   - ➜ A **model** that you will estimate
6. Train the model
   - ➜ Output is an *estimated* model
7. Evaluate model performance
8. Use estimated model to generate *predicted* classifications for unlabeled documents

Creating a labeled set and evaluating its reliability

How do we obtain a labeled set?

1. **External sources of labels**
   - ➜ Documents authored by same people, like Federalist papers
   - ➜ Party labels for election manifestos
   - ➜ Legislative proposals by think tanks
2. **Expert labels**
   - ➜ "Canonical" dataset in Comparative Manifesto Project
   - ➜ In most projects, undergraduate students whose expertise comes from careful training
3. **Crowd-sourced labels**
   - ➜ **Wisdom of crowds**: aggregated judgments of non-experts converge to judgments of experts at much lower cost
   - ➜ Easy to implement with CrowdFlower or MTurk

Evaluating the quality of a labeled set

Before using labeled documents for classification, you *must* assess the quality of the labeled set

Considerations for a hand-coding from Neuendorf (2017):

- ➜ *Objectivity-intersubjectivity*: shared understanding among coders
- ➜ *A priori design*: "codebook" developed in advance of coding
- ➜ *Reliability*: different coders yield same codings
- ➜ *Validity*: coding captures what we intend it to
- ➜ *Genralisability*: can be used in new contexts
- ➜ *Replicability*: coding should be able to be replicated

One I'll add: is your labeled set a *random sample*?

See:
Krippendorff, Klaus. 2019. *Content Analysis: An Introduction to Its Methodology* 4th ed. SAGE Publications, Inc. https://doi.org/10.4135/9781071878781
Neuendorf, Kimberly A. 2017. *The Content Analysis Guidebook* 2nd ed. SAGE Publications, Inc. https://doi.org/10.4135/9781071802878

Evaluating the quality of a labeled set

If you want to hand code a labeled set, GRS offers a good procedure

A key point: have multiple people label each document and then check the **inter-rater reliability** (or **inter-coder reliability**)

- **Percent agreement** is very simple and straight forward
- **Correlation**, usually Pearson's r, but can also be Spearman's rho or Kendall's tau-b (correlation for ordinal data)
- **Agreement measures** take into account not only observed agreement, but also *agreement that would have occurred by chance*
  - **Cohen's kappa** is most common
  - **Krippendorf's alpha** is a generalization of Cohen's kappa

For all: 1 is perfect agreement

Classifier performance metrics

Quantifying the performance of classifiers

Classifiers *predict* which class each document is in

Classifiers are build using labeled data where we already know the classifications

- ➜ We think of the already existing labels as the “gold standard” or simply the “truth”

To evaluate performance we ask: how good of a job did the classifier do predicting classes, when compared to the gold standard/truth?

To build ideas, consider a simple example:

- ➜ A classification task with three classes, A, B and C
- ➜ 100 “labeled” documents (i.e., we know the class)
- ➜ A classifier that predicted each document’s class

Confusion matrix

A **confusion matrix** contains counts of documents that are (1) truly in each class and (2) predicted to be in each class by the classifier

Predicted class

True class		A	B	C	
	A	**35**	10	5	**50**
	B	6	**24**	9	**39**
	C	8	7	**16**	**31**
		49	**41**	**30**	**120**

With this matrix, you can calculate several metrics

Standard performance metrics

Accuracy: how correct is the classifier's identifications overall?

$$\text{Accuracy} = \frac{\text{Correctly classified}}{\text{Total number classified}}$$

How correct is my classifier's classifications of a particular class?

➜ **Precision** (for a class k): what proportion of observations classified in class k are truly in that class?

$$\text{Precision}_k = \frac{\text{Correctly classified as } k}{\text{Total number classified as } k}$$

➜ **Recall** (for a class): what proportion of observations in the class are correctly classified by the classifier?

$$\text{Recall} = \frac{\text{Correctly classified as } k}{\text{Total number truly in class } k}$$

Standard performance metrics

For a class k, you can also construct a composite measure of precision and recall (technically, the harmonic mean)

$$\text{F1}_k = 2 \times \frac{\text{Precision}_k \times \text{Recall}_k}{\text{Precision}_k + \text{Recall}_k}$$

And since precision and recall are defined on a class-by-class basis, you can also create an F1 that is an average across classes

➜ You have to decide how to average for your specific purpose
See https://en.wikipedia.org/wiki/F-score#Extension_to_multi-class_classification

Warning: lots of people report precision, recall and F1 without explicitly saying which class – use context clues to figure it out

Confusion matrix

Predicted class

True class	A	B	C	
A	**35**	10	5	**50**
B	6	**24**	9	**39**
C	8	7	**16**	**31**
	49	**41**	**30**	**120**

What is the accuracy of this classifier?

$$\text{Accuracy} = \frac{35 + 24 + 16}{120} = 0.625$$

In other words: what proportion of the documents were classified correctly by the classifier?

Confusion matrix

Predicted class

		A	B	C	
True class	A	**35**	10	5	**50**
	B	6	**24**	9	**39**
	C	8	7	**16**	**31**
		49	**41**	**30**	**120**

What is the precision of this classifier for class A?

$$\text{Precision}_A = \frac{35}{35+6+8} = 0.714$$

In other words: of the documents that my classifier classified as A, what proportion were actually in class A?

Confusion matrix

		Predicted class			
		A	B	C	
True class	A	**35**	10	5	**50**
	B	6	**24**	9	**39**
	C	8	7	**16**	**31**
		49	**41**	**30**	**120**

What is the recall of this classifier for class A?

$$\text{Recall}_A = \frac{35}{35 + 10 + 5} = 0.700$$

In other words: of the documents that are in class A, what proportion were classified correctly by the classifier?

Confusion matrix

		Predicted class			
		A	B	C	
True class	A	**35**	10	5	**50**
	B	6	**24**	9	**39**
	C	8	7	**16**	**31**
		49	**41**	**30**	**120**

Can calculate precision and recall for the other classes too:

$$\text{Precision}_B = \frac{24}{10+24+7} = 0.585 \qquad \text{Recall}_B = \frac{24}{6+24+9} = 0.615$$

$$\text{Precision}_C = \frac{16}{5+9+16} = 0.533 \qquad \text{Recall}_C = \frac{16}{8+7+16} = 0.516$$

Which performance metric do I use?

Deciding which metric to focus on depends on your task

Example from GRS: spam filter

- If you want to make sure that messages sent to junk folder are actually spam: *precision*
- If you want to make sure that all spam ends up in the junk folder: *recall*

For an email provider, precision is (probably) more important

Report multiple metrics as they give different information

- Accuracy is often misleading. . .

Why you should report multiple statistics: spam

True class	Predicted class: Not Spam	Predicted class: Spam	
Not Spam	**920**	60	**980**
Spam	20	**20**	**40**
	940	**80**	**1020**

$$\text{Accuracy} = \frac{940}{1020} = 0.92$$

$$\text{Recall}_S = \frac{20}{20+20} = 0.50$$

$$\text{Precision}_S = \frac{20}{20+60} = 0.25$$

92% are accurately classified, but 50% of spam messages are getting through the filter and 75% of messages in the junk mail folder *aren't spam*(!)

Why you should report multiple statistics: illegal content

Predicted class

True class	Legal	Illegal	
Legal	**850**	50	**900**
Illegal	100	**50**	**150**
	950	**100**	**1050**

$$\text{Accuracy} = \frac{900}{1050} = 0.86$$

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

$$\text{Precision}_I = \frac{50}{50 + 50} = 0.50$$

86% are accurately classified, but 66% of illegal content is getting through the filter and 50% of blocked content isn't illegal

Binary classification

Binary classification tasks are common, e.g.

- ➜ Spam versus not spam
- ➜ Illegal content versus legal content
- ➜ Republican speech versus Democratic speech

We have some special (but confusing) terminology for such scenarios

		Predicted class	
		Positive	Negative
True class	Positive	True Positives	False Negatives (Type II error)
	Negative	False Positives (Type I error)	True Negatives

The confusing part: what's "positive" and what's "negative"?

Examples of classifiers: Naive Bayes

Doing the classification

So far, we've talked a lot about classifier performance

In some sense, we were getting ahead of ourselves

Now we ask: *how do we actually perform the classification?*

There are a set of algorithms/models/classifiers that are designed for this task

- ➜ They all estimate the *probability* a document i is in class k
- ➜ Once you get these estimated probabilities, you need to decide how to do the classification

The example set up

Suppose we have N documents, each of which we want to classify into one of K "classes"

- Each document i's class is notated by y_i
- Documents with known (already labeled) classes are used to **train** a classifier
- We'll "dummy" the class variable, so that for every document i, $\boldsymbol{\pi}_i$ is a one-hot encoding where $\pi_{ik} = 1$ if and only if document i is in class k

What is our goal?

Given document D_i, we want to predict its class

For each class k, we want to know $\Pr(\pi_{ik} = 1 | D_i)$

We can't observe this, but using Bayes' theorem, we can write:

$$\Pr(\pi_{ik} = 1 | D_i) = \frac{\Pr(\pi_{ik} = 1) \Pr(D_i | \pi_{ik} = 1)}{\Pr(D_i)}$$

Since the denominator is just a scaling factor, we can simplify:

$$\Pr(\pi_{ik} = 1 | D_i) \propto \underbrace{\Pr(\pi_{ik} = 1)}_{\text{class prevalence}} \overbrace{\Pr(D_i | \pi_{ik} = 1)}^{\substack{\text{class-specific} \\ \text{language model}}}$$

Specifying a model

We said that $\Pr(D_i|\pi_{ik} = 1)$ is a class specific language model

But to make any progress, we need to say *what* model

Lots of directions we could go, but with a DFM and bag of words, we can use a multinomial model (like before!)

- ➜ Document D_i is represented by $\mathbf{W}_i$ (vector of feature counts)
- ➜ Document D_i has $M_i = \sum_j W_{ij}$ tokens (sum across columns)

Since the model depends on the category π_{ik}, we can write

$$\mathbf{W}_i|(\pi_{ik} = 1) \sim \text{Multinomial}(M_i, \boldsymbol{\mu}_k)$$

Specifying a model

And since we use the multinomial, we know (from Wikipedia!):

$$\Pr(\mathbf{W}_i | \pi_{ik} = 1) = \frac{M_i!}{\prod_{j=1}^{J} (W_{ij}!)} \prod_{j=1}^{J} \left(\mu_{kj}^{W_{ij}} \right)$$

We can, again, drop the scaling factor to simplify:

$$\Pr(\mathbf{W}_i | \pi_{ik} = 1) \propto \prod_{j=1}^{J} \left(\mu_{kj}^{W_{ij}} \right)$$

Specifying a model

Now recall what we're interested in:

$$\Pr(\pi_{ik} = 1 | D_i) \propto \Pr(\pi_{ik} = 1) \Pr(D_i | \pi_{ik} = 1)$$

Given we assumed the multinomial model, we can substitute:

$$\Pr(\pi_{ik} = 1 | D_i) \propto \Pr(\pi_{ik} = 1) \prod_{j=1}^{J} \left(\mu_{kj}^{W_{ij}} \right)$$

And finally, let's use δ_k to indicate the prevalence of class k:

$$\Pr(\pi_{ik} = 1 | D_i) \propto \delta_k \prod_{j=1}^{J} \left(\mu_{kj}^{W_{ij}} \right)$$

Naive Bayes classifier

This is the full model we plan to estimate and use as our classifier

$$\Pr(\pi_{ik} = 1 | D_i) \propto \delta_k \prod_{j=1}^{J} \left(\mu_{kj}^{W_{ij}} \right)$$

This approach is known as **Naive Bayes** because it uses a language model (i.e., a distribution) that assumes that tokens in each document are drawn independently conditional on class

This is, again, downstream of bag of words

Naive Bayes classifier

Some caveats:

1. The assumptions of Naive Bayes are obviously wrong
   - ➜ The simplicity of the model makes it easier to estimate, but also more limited in its ability to provide accurate classifications
   - ➜ One issue (see GRS): because it ignores correlations between words in documents, it can yield incorrect estimates of the proportion of documents in a class
   - ➜ But it still largely works for classifying individual documents
2. Don't need to use multinomial; any distribution satisfying conditional independence can be a Naive Bayes classifier

Estimating a Naive Bayes classifier

Remember what we're interested in estimating:

$$\Pr(\pi_{ik} = 1 | D_i) \propto \delta_k \prod_{j=1}^{J} \left(\mu_{kj}^{W_{ij}} \right)$$

There are two quantities that we can estimate: $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$

- ➜ $\boldsymbol{\delta}$ is a vector of all the class prevalences (the δ_k parameters)
- ➜ $\boldsymbol{\mu}$ is a matrix containing all the individual μ_{kj} parameters

We can calculate all these things with our sample (data)

How?

Estimating a Naive Bayes classifier

What proportion of documents of all documents are in class k?

$$\hat{\delta}_k = \frac{\sum_i \pi_{ik}}{N}$$

(Note: I use slightly different notation that GRS)

For each token j in class k, what proportion of all tokens used in class k documents were token j?

$$\hat{\mu}_{kj} = \frac{\sum_i \pi_{ik} W_{ij}}{\sum_j \sum_i \pi_{ik} W_{ij}}$$

(Adding a J-length Laplace smoother $\boldsymbol{\alpha}$ to deal with sparsity:)

$$\hat{\mu}_{kj} = \frac{\alpha_j + \sum_i \pi_{ik} W_{ij}}{\sum_j \left(\alpha_j + \sum_i \pi_{ik} W_{ij}\right)}$$

Putting everything together

For a document i, we can estimate the model with

$$\widehat{\Pr}(\pi_{ik} = 1 | \mathbf{W}_i) \propto \hat{\delta}_j \prod_j \hat{\mu}_{kj}^{W_{ij}}$$

This will yield a quantity that is proportional to the probability that a document $\mathbf{W}_i$ is in class k

➜ To make it an actual probability, multiply it by $\frac{M_i!}{\prod_{j=1}^{J}(W_{ij}!)}$

An example: inauguration speeches

An example of a (reduced) DFM from the inaugural address corpus

Suppose we want to classify whether a speech was written by Trump using data from all speeches since 2000

➜ Question: *why* would we want to do this?

	pi.ik	america	us	nation
2001-Bush	0	11	11	14
2005-Bush	0	20	3	13
2009-Obama	0	10	23	15
2013-Obama	0	9	21	9
2017-Trump	1	19	2	13
2021-Biden	0	20	27	16

Right away, we can calculate

$$\hat{\delta}_{\mathcal{T}} = \frac{1}{6} \approx 0.167 \quad \text{and} \quad \hat{\delta}_{\mathcal{O}} = \frac{5}{6} \approx 0.833$$

An example: inauguration speeches

Collapsing the DFM into the two categories:

```
          pi.ik     america          us      nation
Not.Trump     0          70          85          67
Trump         1          19           2          13
```

Let's calculate the probability Trump uses word `america` ($\hat{\mu}_{\mathcal{T}\texttt{a}}$):

$$\hat{\mu}_{\mathcal{T}\texttt{a}} = \frac{\alpha_j + 19}{(\alpha_j + 19) + (\alpha_j + 2) + (\alpha_j + 13)}$$

Adding $\alpha_j = 1$ as a smoother

$$\hat{\mu}_{\mathcal{T}\texttt{a}} = \frac{20}{37} \approx 0.541$$

An example: inauguration speeches

Collapsing the DFM into the two categories:

```
          pi.ik   america      us   nation
Not.Trump     0        70      85       67
Trump         1        19       2       13
```

Let's calculate the probability other presidents use word `america` ($\hat{\mu}_{\mathcal{O}\texttt{a}}$):

$$\hat{\mu}_{\mathcal{O}\texttt{a}} = \frac{\alpha_j + 70}{(\alpha_j + 70) + (\alpha_j + 85) + (\alpha_j + 67)}$$

Adding $\alpha_j = 1$ as a smoother

$$\hat{\mu}_{\mathcal{O}\texttt{a}} = \frac{71}{225} \approx 0.316$$

An example: inauguration speeches

Repeat for each feature to get all estimated model parameters

	$\hat{\mu}_{k\mathrm{a}}$	$\hat{\mu}_{k\mathrm{u}}$	$\hat{\mu}_{k\mathrm{n}}$	$\hat{\delta}_k$
Not Trump	0.316	0.382	0.302	0.833
Trump	0.541	0.081	0.378	0.167

Now, let's predict the category for the first Bush speech using the model parameters

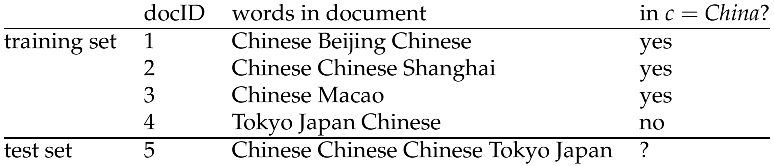
```
             pi.ik    america         us      nation
2001-Bush        0         11         11          14
```

$$\widehat{\Pr}(\text{Trump}|\mathbf{W}_{\text{B2001}}) \propto 0.167 \times 0.541^{11} \times 0.081^{11} \times 0.378^{14} = 2.355282 \times 10^{-22}$$

$$\widehat{\Pr}(\text{Not Trump}|\mathbf{W}_{\text{B2001}}) \propto 0.833 \times 0.316^{11} \times 0.382^{11} \times 0.302^{14} = 3.472943 \times 10^{-18}$$

Another example

▶ **Table 13.1** Data for parameter estimation examples.



	docID	words in document	in $c = China$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

Source: Manning, Christopher D., Raghavan, Prabhakar, and Schütze, Hinrich. 2008. *Introduction to information retrieval*. Cambridge University Press. LSE Library link

Another example

Example 13.1: For the example in Table 13.1, the multinomial parameters we need to classify the test document are the priors $\hat{P}(c) = 3/4$ and $\hat{P}(\overline{c}) = 1/4$ and the following conditional probabilities:

$$\begin{aligned}
\hat{P}(\text{Chinese}|c) &= (5+1)/(8+6) = 6/14 = 3/7 \\
\hat{P}(\text{Tokyo}|c) = \hat{P}(\text{Japan}|c) &= (0+1)/(8+6) = 1/14 \\
\hat{P}(\text{Chinese}|\overline{c}) &= (1+1)/(3+6) = 2/9 \\
\hat{P}(\text{Tokyo}|\overline{c}) = \hat{P}(\text{Japan}|\overline{c}) &= (1+1)/(3+6) = 2/9
\end{aligned}$$

The denominators are $(8+6)$ and $(3+6)$ because the lengths of $text_c$ and $text_{\overline{c}}$ are 8 and 3, respectively, and because the constant B in Equation (13.7) is 6 as the vocabulary consists of six terms.

We then get:

$$\begin{aligned}
\hat{P}(c|d_5) &\propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003. \\
\hat{P}(\overline{c}|d_5) &\propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001.
\end{aligned}$$

Thus, the classifier assigns the test document to c = *China*. The reason for this classification decision is that the three occurrences of the positive indicator Chinese in d_5 outweigh the occurrences of the two negative indicators Japan and Tokyo.

Examples of classifiers: Regularised regression

Regularised regression

Another common classification algorithm is linear regression, which chooses the $\boldsymbol{\mu}$ that minimizes the **residual sum of squared errors**:

$$\text{RSS} = \sum_{i=1}^{N} (y_i - \mathbf{W}_i \boldsymbol{\mu})^2$$

If $\boldsymbol{\mu} = (\mu_0, \mu_1, ..., \mu_J)$, we can write this as:

$$\sum_{i=1}^{N} \left(y_i - \mu_0 - \sum_{j=1}^{J} \mu_j W_{ij} \right)^2$$

Note: We're implicitly assuming a binary categorisation

But is it possible to do this for DFMs?

- ➜ If $J > N$, then OLS does not have a unique solution
- ➜ Even with $N > J$, OLS has low bias/high variance (**overfitting**)

Regularised regression

To deal with these issues, we can add a **penalty for model complexity** (e.g., too many features!), such that we now minimize:

$$\sum_{i=1}^{N}\left(y_i-\mu_0-\sum_{j=1}^{J}\mu_j W_{ij}\right)^2+\lambda\sum_{j=1}^{J}\mu_j^2$$ **ridge regression**

or

$$\sum_{i=1}^{N}\left(y_i-\mu_0-\sum_{j=1}^{J}\mu_j W_{ij}\right)^2+\lambda\sum_{j=1}^{J}|\mu_j|$$ **lasso regression**

λ is the **penalty parameter**, which is a "hyperparameter" to be chosen via cross-validation

➜ The lasso and ridge penalties are often referred to as L1 and L2 penalties, respectively

➜ **Elastic net** regression takes a weighted average of L1 and L2

Regularised regression

Estimation gives us $\hat{\boldsymbol{\mu}}$

We don't care about these estimates per se – our goal is to classify into categories

For each document i, we can calculate $\hat{y}_i$, yielding something like the *probability* that document i is classified as 1:

$$\hat{y}_i = \hat{\mu}_0 + \sum_{j=1}^{J} \hat{\mu}_j W_{ij}$$

➜ Not exactly a probability — can be outside $[0, 1]$

Just like Naive Bayes, regression gives us an estimate of (something like) a probability that a document is in a class

Lasso vs. ridge regression

How does the penalty work?

- ➜ Algorithm is trying to *minimise* RSS but λ is increasing it
- ➜ The optimal way to minimise RSS, given the penalty: push down coefficients that create large penalties
- ➜ If $\lambda = 0$, both are equivalent to standard OLS

Both types of regression penalise large regression coefficients

- ➜ The idea: we're worried about some features causing our model predictions to swing wildly (too much variance)
- ➜ Features with huge effects are pushed toward zero

Lasso regression pushes some coefficients to zero, but ridge does not

- ➜ Lasso can be used for feature selection and to identify a regression when $J > N$

Advantages and disadvantages of regularised regression

Advantages of regularised regression:

- ➜ Easy to interpret
- ➜ Works well when features are each independently informative
- ➜ Works well when data is limited

Disadvantages:

- ➜ Does not work well when features are, e.g., jointly informative
- ➜ Often outperformed by more complex methods

Adaptive basis function methods: classifiers that adaptively learn feature importance (tree methods, boosting, neural networks)

- ➜ These topics are covered in more detail in MY474

Classification after prediction

Classification after prediction

Classifiers give predictions about probabilities, not classifications

How do you determine document i's category using the predictions?

One intuitive option (for binary classification):

- ➜ Classify document i in a class k if the classifier's predicted probability for class k is greater than 0.5

But using 0.5 doesn't usually work well—there's no magic threshold

- ➜ Depends on your task and on your data

For example: if you are trying to classify unbalanced classes (e.g., illegal content), 0.5 threshold is not going to work

- ➜ All your predicted probabilities will be very low or very high

Classification after prediction

A common approach for “balanced” classes (i.e., similarly sized):

Choose threshold t that maximises difference between **True Positive Rate (TPR)** and **False Positive Rate (FPR)**

- True Positive Rate (TPR): using threshold t, calculate recall for the documents classified as “positive”
- False Positive Rate (FPR): using threshold t, calculate recall for the documents classified as “negative” and then subtract from 1

Some boring terminology issues:

- For binary classification, many people say “recall” to mean the recall for the positive classification (i.e., TPR)
- TPR is also known as **sensitivity**
- Recall for negative classified documents is known as **specificity**

Building a robust classifier

Choosing your classifier

Which classifier should you use?

Obviously, you want to use a classifier that performs the "best" (e.g., high accuracy, recall and precision)

You have to choose which classifier to use:

- ➜ Which "algorithm"? E.g., Naive Bayes, regularised regression, etc.
- ➜ What **hyperparameters** do you want to set? E.g., which variables to include in your regression or which language model to use for Naive Bayes

People refer to this as "tuning" a classifier

It's tempting to try a lot of options and pick the best performer—but beware. . . .

Overfitting

Classifiers are optimised to maximise performance statistics within the **training sample** used to do the classification

- ➜ In otherwords: they are optimised for **in-sample performance**

But generally we want to apply our models to new data

- ➜ For example, we want to classify a large number of documents into categories, but only have a small labeled sample

Overfitting occurs when a classifier builds a model that is too closely tailored for the sample used for training

- ➜ Overfitting means a classifier won't perform well on new data

Overfitting

Contrived example:

	by	man	upon
Hamilton	859	102	374
Jay	82	0	1
Madison	474	17	7

Suppose a classifier generates this estimated model:

- ➜ A document is written by Hamilton if and only if it contains 859 uses of the word "by"; written by Jay if and only if it contains 82 uses of the word "by"; and written by Madison if and only if it contains 474 uses of the word "by"

This model will perform *perfectly* in the training set: but is it a good model for use in other data? It has **overfit** to the sample!

- ➜ Roughly speaking, it is taking the sampling noise too seriously

Overfitting

Can happen when the model you want to estimate is “too complex”

- ➜ Machines are very good at finding patterns, even spurious ones
- ➜ Giving your machine a lot of flexibility (by letting it fit a very complex model) allows it more room to find patterns
- ➜ But it’s finding patterns specific to the training set, so its performance **in-sample** will be better than its performance **out-of-sample**
- ➜ You need to discipline the machine because you want to use the model for out-of-sample classification!

Can also happen with quirks in the data, e.g. zeroes

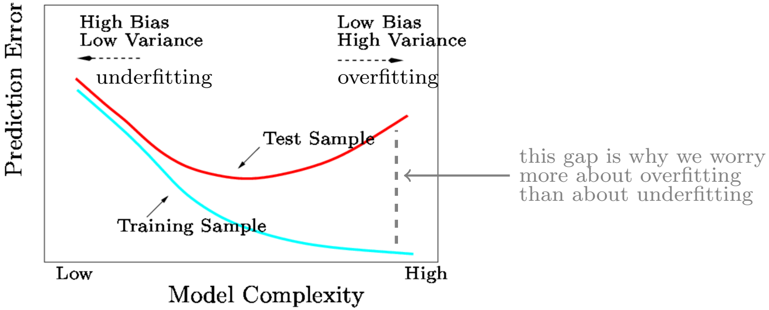
The bias-variance tradeoff

We make models complex to reduce bias across samples

➜ Recall: this is why people add controls to regressions!

But model complexity increases variance across samples

In general, there is a trade-off between bias and variance



In-sample versus out-of-sample performance

You can calculate accuracy, precision and recall in sample

- ➜ This gives a rough sense of how good your classifier is *within the training set*
- ➜ This is what we did for the hypothetical examples above
- ➜ When you look at performance metrics of an OLS regression (e.g., goodness of fit), this is usually in-sample

But, if you do this with the intention of trying to improve or otherwise change your classifier, you're probably fueling overfitting

Instead: you should check the performance of your classifier in a **validation (or test) set**

- ➜ This consists of a random sample of your labeled data that has not been used to train the model

Uses of validation sets

Validation sets are useful for two things:

1. Evaluating how well your model would do on untouched data
   - ➜ Remember: in-sample performance measures will always make a model look better than it will be out-of-sample
2. Allowing you to choose among different classifiers
   - ➜ I.e., choosing your algorithm and/or hyperparameters

The basic process for choosing the best classifier:

1. choose a set of different classifiers you want to try
2. train each classifier on the training set and evaluate performance on validation set
3. select the classifier with the best *validation set performance*
4. use that classifier on *entire training dataset* to generate prediction model you can use to label unlabeled data

Validation set performance isn't magic

Overfitting is a risk when classifiers are adjusted/tuned based on performance metrics of previously fitted models

Classifiers are doing this "under the hood" when estimating models

- ➜ To some extent, you can't fully avoid "overfitting" to a training set (that's why performance in sample will always be better)

But, if you iteratively train, evaluate performance in a validation set then tune, you can also cause overfitting (yikes)

TLDR: for each classifier you are evaluating, you get to use your training set *once* and your validation set *once*

Cross-validation

Standard approach to prevent overfitting: train classifiers on a training set, validate on a validation set, choose best classifier

- This is wasting data — you didn't use all your data to train, nor did you validate your model on all your data

What if you do the following instead

- Train on the training set and validate on the validation set then *switch* to train on the validation set and validate on the training set
- Pick best performing classifier based on its average performance across these two different iterations

Now you've trained *and* validated the model on all the data

This idea is called **cross-validation**

Cross-validation

The magic of cross-validation is more apparent when you split the data into more than just two sets (i.e., training and validation)

Instead, partition your data into K randomly created **folds** (often 10), then you have K training sets and K validation sets

How does it work? For each classifier you're considering:

- For each fold k, you treat all the folds *except* k as a training set, running classifier on it and evaluating performance on the held-out fold k
- Repeat for each of the remaining $K - 1$ folds
- Average together the performance metrics across the folds

As K becomes bigger: (1) computation costs increase, but (2) each fold's training set is larger and more representative of full dataset

Cross-validation

You can still create overfitting if you use cross validation, then tune your model and use cross validation again (esp. with same folds)

The principle remains unchanged: you are at risk of overfitting when you tune your model in response to measured performance

- ➜ E.g., when you iteratively train on the same training set in response to performance statistics from previous rounds of training

Choosing a robust classifier

Choose a set of different “candidate” classifiers

For each one:

- use your labeled dataset to fit a model via K-fold cross-validation
- average together the performance metrics across the K folds

Select the classifier that performs the best

Rerun that classifier on your full labeled dataset to get a model you can apply to out-of-sample predictive tasks

- Out of sample performance metrics can be extracted from the cross-validation process

Application: civility on Twitter

Example:

Theocharis et al. (2016) ask: why don't politicians take full advantage of interactive nature of social media?

Two hypotheses:

- ➜ H1: Politicians make broadcasting rather than engaging use of Twitter
- ➜ H2: Engaging style of tweeting is positively related to impolite or uncivil responses

Source: https://doi.org/10.1111/jcom.12259

Data collection and case selection

Data: European Election Study 2014, Social Media Study

- ➜ List of all candidates with Twitter accounts in 28 EU countries
- ➜ 2,482 out of 15,527 identified MEP candidates (16%)
- ➜ Collaboration with TNS Opinion to collect all tweets by candidates *and* tweets mentioning candidates (tweets, retweets, @-replies), May 5th to June 1st 2014.

Case selection: expected variation in politeness/civility

	Received bailout	Did not receive bailout
High support for EU	Spain (55.4%)	Germany (68.5%)
Low support for EU	Greece (43.8%)	UK (41.4%)

(% indicate proportion of country that considers the EU to be "a good thing")

Data collection and case selection

Data coverage by country

Country	Lists	Candidates	on Twitter	Tweets
Germany	9	501	123 (25%)	86,777
Greece	9	359	99 (28%)	18,709
Spain	11	648	221 (34%)	463,937
UK	28	733	304 (41%)	273,886

Coding tweets

Labeled data: random sample of approximately 7,000 tweets from each country, each labeled by undergraduate students

1. **Politeness**
   - ➜ Polite: tweet adheres to politeness standards.
   - ➜ Impolite: ill-mannered, disrespectful, offensive language. . .
2. **Communication style**
   - ➜ Broadcasting: statement, expression of opinion
   - ➜ Engaging: directed to someone else/another user
3. **Political content: moral and democracy**
   - ➜ Tweets refer to: freedom and human rights, traditional morality, law and order, social harmony, democracy. . .

Incivility = impoliteness + moral and democracy

Coding tweets

Coding process: summary statistics

	Germany	Greece	Spain	UK
Coded by 1/by 2	2947/2819	2787/2955	3490/1952	3189/3296
Total coded	5766	5742	5442	6485
Impolite	399	1050	121	328
Polite	5367	4692	5321	6157
% Agreement	92	80	93	95
Krippendorf/Maxwell	0.30/0.85	0.26/0.60	0.17/0.87	0.54/0.90
Broadcasting	2755	2883	1771	1557
Engaging	3011	2859	3671	4928
% Agreement	79	85	84	85
Krippendorf/Maxwell	0.58/0.59	0.70/0.70	0.66/0.69	0.62/0.70
Moral/Dem.	265	204	437	531
Other	5501	5538	5005	5954
% Agreement	95	97	96	90
Krippendorf/Maxwell	0.50/0.91	0.53/0.93	0.41/0.92	0.39/0.81

Machine learning classification of tweets

Coded tweets as training dataset for a machine learning classifier:

1. **Text preprocessing**: lowercase, remove stopwords and punctuation (except # and @), transliterating to ASCII, stem, tokenize into unigrams and bigrams. Keep tokens in 2+ tweets but <90%.
2. **Train classifier:** logistic regression with L2 regularization (ridge regression), one per language and variable
3. **Evaluate classifier:** compute accuracy using 5-fold cross-validation

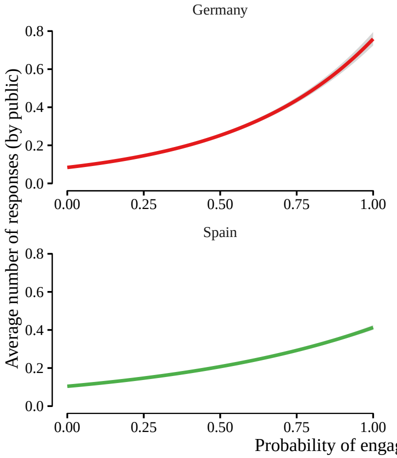
Machine learning classification of tweets

Classifier performance (5-fold cross-validation)

		UK	Spain	Greece	Germany
Communication Style	Accuracy	0.821	0.775	0.863	0.806
	Precision	0.837	0.795	0.838	0.818
	Recall	0.946	0.890	0.894	0.832
Polite vs. impolite	Accuracy	0.954	0.976	0.821	0.935
	Precision	0.955	0.977	0.849	0.938
	Recall	0.998	1.000	0.953	0.997
Morality and Democracy	Accuracy	0.895	0.913	0.957	0.922
	Precision	0.734	0.665	0.851	0.770
	Recall	0.206	0.166	0.080	0.061

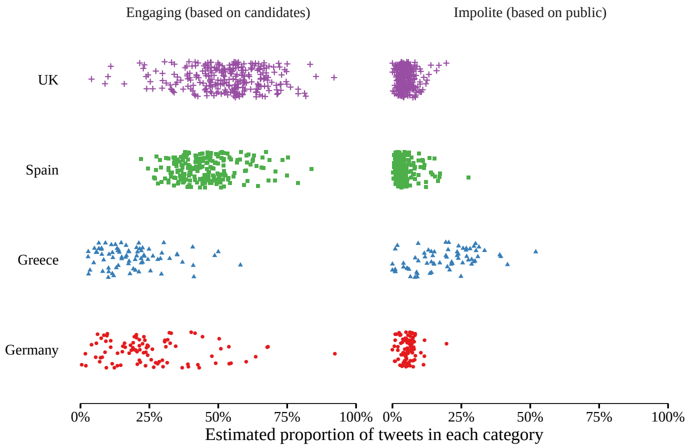
Predictive validity

Citizens are more likely to respond to candidates when they adopt an engaging style



Results: H1

Proportion of engaging tweets sent and impolite tweets received, by candidate and country



Results: H2

Is engaging style positively related to impolite responses?

Three levels of analysis:

1. **Across candidates**: candidates who send more engaging tweets receive more impolite responses.
2. **Within candidates, over time**: the number of impolite responses increases during the campaign for candidates who send more engaging tweets
3. **Across tweets**: tweets that are classified as engaging tend to receive more impolite responses