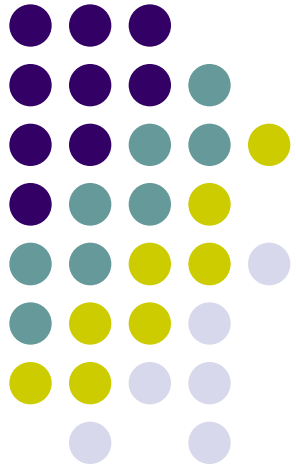
Applied Scaling & Classification Techniques in Political Science

Lecture 6

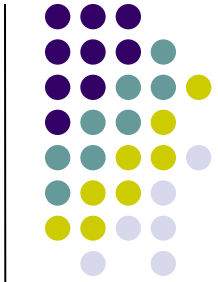
Dictionaries and Supervised classification methods



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

Classification methods



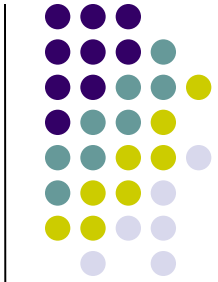
Classifying Documents into Known Categories

Assigning texts to **some known categories** is the most common use of content analysis methods in political science

For example, researchers may ask if local news coverage is positive or negative, if legislation is about the environment or some other issue area, if international statements are belligerent or peaceful, etc.

In each instance, the goal is to infer **either the category of each document**, the **overall distribution of documents across categories**, or **both**

Classification methods



Human-based methods for making these inferences are both time and resource intensive

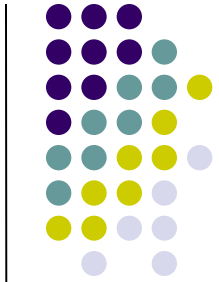
Automated methods can mitigate the cost of assigning documents to categories

Two broad groups of methods are available:

Dictionary methods use the relative frequency of key words to measure the presence of each category in texts

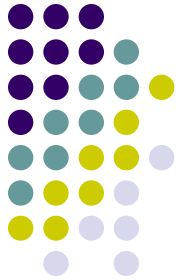
Supervised learning methods replicate the familiar manual coding task, but with a machine. **First**, human coders are used to classify a subset of documents into a predetermined categorization scheme. **Then**, this training set is used to train an automated method, which then classifies the remaining documents

Dictionary methods



Dictionaries use the **relative rate** at which key words appear in a text to **classify documents into categories** or **to measure the extent to which** documents belong to particular categories

Dictionary methods

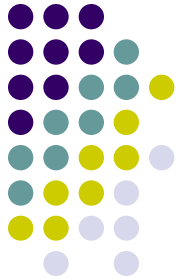


Suppose the goal is to measure the **tone** (also called the "**sentiment"**) in newspaper articles: whether articles convey information positively or negatively about a given topic

A **dictionary to measure sentiment** is a list of words that are either dichotomously classified as positive («good», «fantastic», etc.) or negative («bad», «horrible», etc.) or contain more continuous measures of their content

You can then use that dictionary to identify **the tone of a document**: either positive or negative according to the relative number of words in that document identified by the dictionary as positive or negative ones

Dictionary methods



Formally, each word *m* (m=1,….M) will have associated score s_m

For the simplest measures, s_m =-1 if the word is associated with a negative sentiment and s_m =+1 if associated with a positive sentiment

If $N_i = \sum_{m=1}^{M} W_{im}$ words (included in the dictionary) are used in document *i*, then dictionary methods can measure the sentiment for any document t_i as:

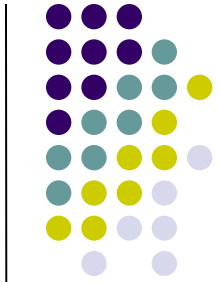
$$t_i = \sum_{m=1}^{M} \frac{s_m W_{im}}{N_i}$$

Dictionary methods

Scholars often use t_i as an approximately continuous measure of document sentiment, but it also can be used to classify documents into **sentiment categories** if a decision rule is assumed along with the dictionary method

Perhaps the simplest coding rule would assign all documents with $t_i > 0$ to a positive sentiment category and $t_i < 0$ to a negative sentiment; and if $t_i = 0$? Either neutral category or NC

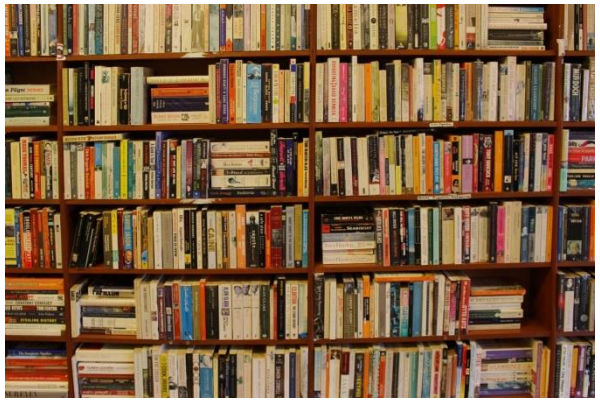
Dictionary methods

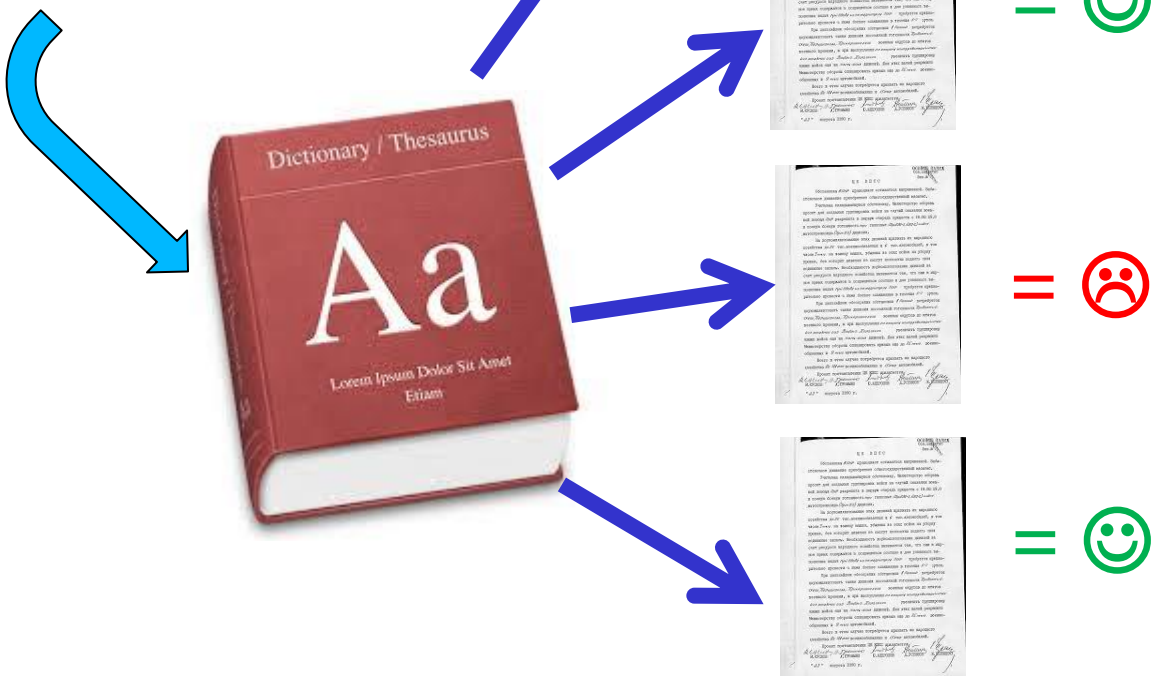


Sentiment analysis is just one type of analysis a dictionary method can perform

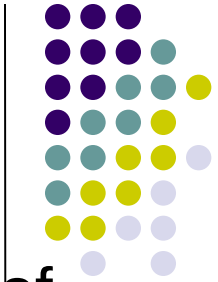
The general idea of dictionaries make them relatively easy and cheap to apply across a variety of problems: identify **words that separate categories** (for example policy categories) and measure **how often those words occur in texts**

Dictionary methods



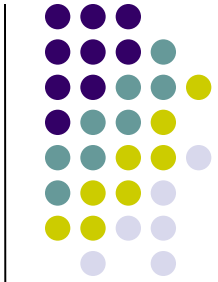


Dictionary methods



Using a dictionary can therefore **minimize** the amount of labor needed to classify documents (and this is attractive!) but…

Dictionary methods



…the challenges of using dictionary methods:

For dictionary methods to work well, the scores attached to words must closely align with how the words are used in a particular context

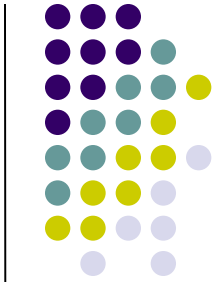
If a dictionary is developed for a **specific application**, then this assumption should be easy to justify

But when dictionaries are **created in one substantive area and then applied to another**, serious errors can occur

Language do **change across topics**!

For example, a word like `cancer` may have a positive connotation in a health-care company, but negative in many other contexts

Dictionary methods



Moreover, dictionary methods (including Natural Language Processing and Computational Linguistic approaches) work pretty well when you study texts that use a **standardized language** (i.e., legal text!).

In other contexts, things become more complex…

Dictionary methods

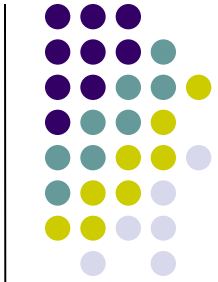
…language in fact evolves continuously: one cannot code all possible semantic rules (double meaning sentences, specific jargons, neologisms, irony) unless reading the posts!!!

Dictionary methods

On the other side, counting the number of positive and negative terms in a sentence may lead to **paradoxical effects**

Dictionary methods

Dictionary methods

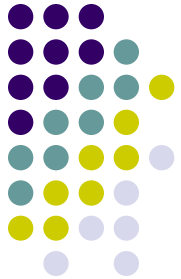


Dictionaries, therefore, should be used with **substantial caution**, or at least coupled with **explicit validation**

The problem is that quite often measures from dictionaries are **rarely validated**. Rather, standard practice in using dictionaries is to assume the measures created from a dictionary are correct and then apply them to the problem

The consequence of **domain specificity and lack of validation** is that most analyses based on dictionaries are built on shaky foundations

Supervised Learning (classification) Methods



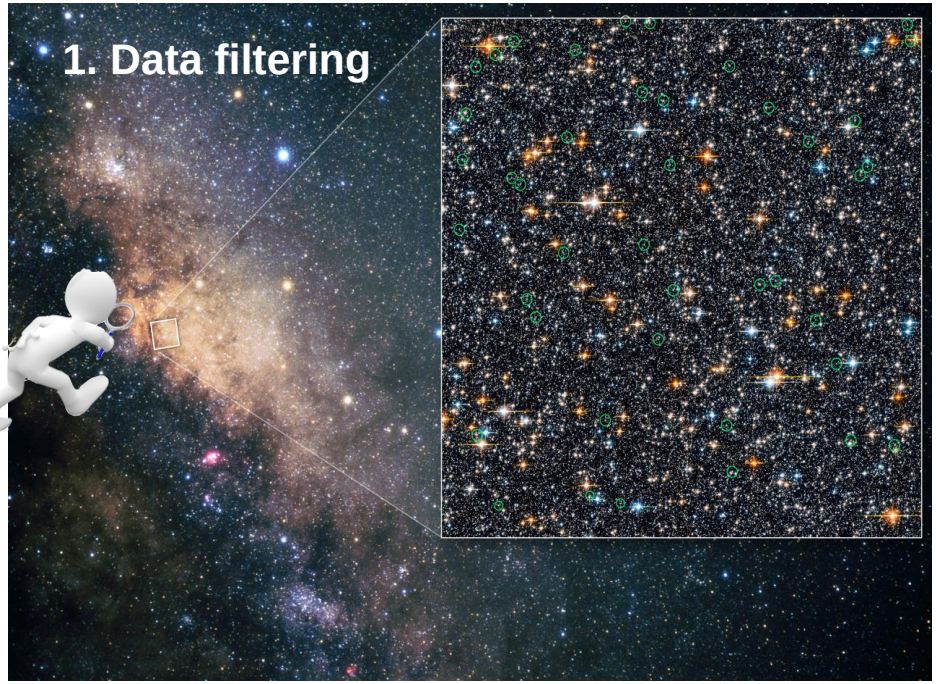
Supervised learning methods provide a useful alternative method for assigning documents to predetermined categories

The idea of supervised learning is simple: human coders categorize a set of documents (the "**training-set**") by hand

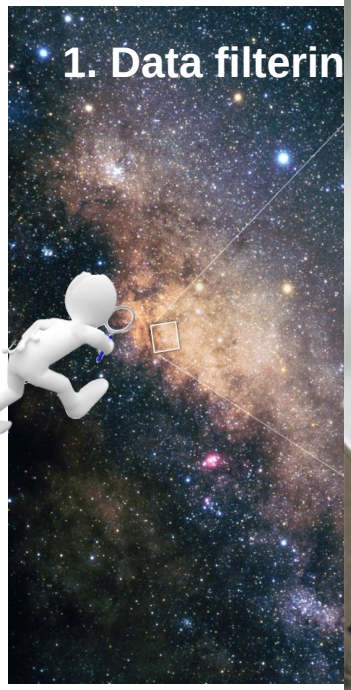
The algorithm "learns" how to sort the documents into categories using the **training set and words**

Then, it classifies the remaining set of document not classified by hand (the "**test-set**") using the characteristics of the documents to place them into the categories

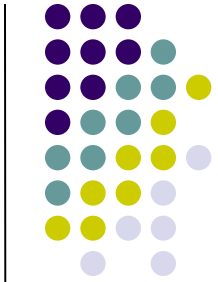
A three-steps procedure



A three-steps procedure



Supervised Learning Methods



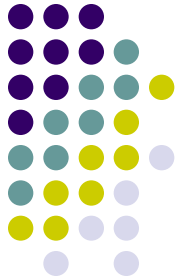
This approach to classification has **three major advantages** over dictionary methods:

First, it is **necessarily domain specific** and therefore avoids the problems of applying dictionaries outside of their intended area of use

Applying supervised learning methods **requires scholars to develop coding rules for the particular quantities of interest** under study when working on the training-set

This also forces scholars to develop coherent definitions of concepts for particular applications, which leads to clarity in what researchers are measuring and studying

Supervised Learning Methods

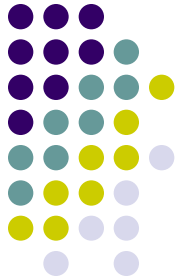


This approach to classification has **three major advantages** over dictionary methods:

Second, human involvement is crucial to understand the correct meaning of a text (double meaning sentences, specific jargons, neologisms, irony)

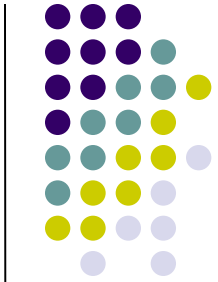
Third, supervised learning methods are much easier to validate, with clear statistics that summarize model performance (see below)

Constructing a training set



The most important step in applying a supervised learning algorithm is constructing a **reliable training set**, because no statistical model can repair a poorly constructed training set, and well coded documents can hide faults in simple models!

Constructing a training set



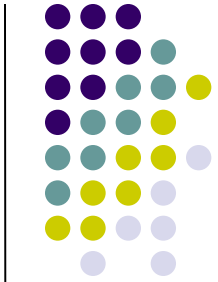
(1) creating and executing a coding scheme:

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

When using the codebook, particularly at first, coders are likely to **identify ambiguities** in the coding scheme or overlooked categories

Constructing a training set

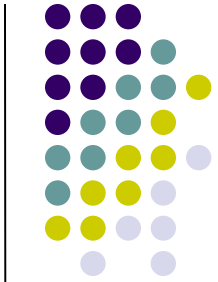


(1) creating and executing a coding scheme:

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



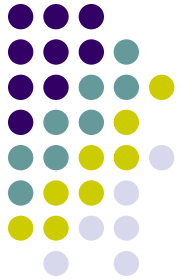
(2) **sampling documents:**

Almost all (but not all…) classification methods 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

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

Constructing a training set



(2) **sampling documents:**

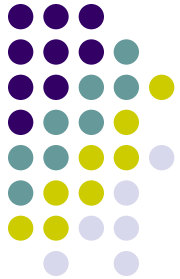
Moreover, Supervised methods need **enough information** to learn the relationship **between words and documents in each category of a coding scheme**

Training sets also need enough documents to apply supervised methods accurately

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

Still the number necessary will depend upon **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

Applying a supervised learning model



After hand classification is complete, the hand-labeled documents are used to train the supervised learning methods to learn about the test set – either:

a) **classifying the individual documents into categories**

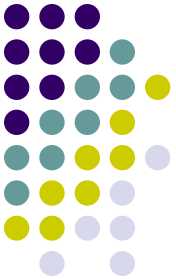
b) **measuring the proportion of documents in each category**

The methods to do this classification are diverse, though they share a common structure that usefully unifies the methods

Today we are going to discuss just about a)

In two weeks we will discuss about b)

Applying a supervised learning model

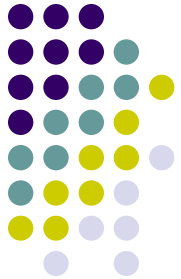


Suppose there are N_{train} documents (i=1,…, N_{train}) in our training set and that we have defined K categories (k=1,…K) for our classification, such as positive, negative, neutral in the case of a sentiment analysis

Each document i's category is then represented by $Y_i \in (C_1, C_2, .., C_K)$ and the entire training-set is represented as $\mathbf{Y}_{train}=(Y_1,\ldots,Y_{Ntrain})$

$\mathbf{W}_{train}$ is the term-document matrix for N_{train}

Applying a supervised learning model



Each supervised learning algorithm assumes that there is some (unobserved) function that describes the relationship between the words and the labels:

$$\boldsymbol{Y}_{train} = \mathrm{f}(\boldsymbol{W}_{train})$$

Each algorithm attempts to learn this relationship by estimating the "true" function *f* with $\hat{f}$

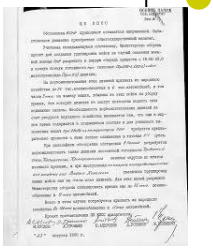
$\hat{f}$ is then used to infer properties of the test set, $\widehat{\boldsymbol{Y}_{test}}$ - **either** each document's category **or** the overall distribution of categories - using the test set's words $\boldsymbol{W}_{test}$,

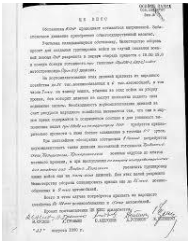
$$\widehat{\boldsymbol{Y}_{test}} = \hat{f}(\boldsymbol{W}_{test})$$

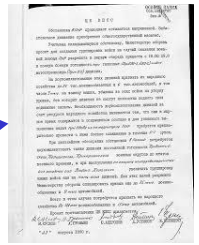
Individual methods

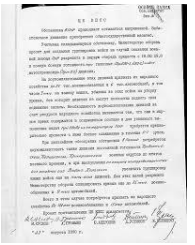
By **machine learning** we can identify all those techniques that involve *individual* classification of the data in the *test set* given a pre-coded *training set*

Supervised learning: individual classification









Individual methods

Several different possible machine learning algorithms out there (including neural network, deep learning, etc.)

Here we will offer an intuitive introduction to three algorithms: Naïve Bayes classifier; Random forest; Support Vector Machine

Naïve Bayes classifier: the algorithm allows us to predict a class, given a set of features using (Bayes) probability theorem

Naïve Bayes classifier

The model has a simple, but powerful, approach to learning the relationship between words and categories

The training set is used to learn about the distribution of words for documents from category *k*

This distribution is then used to classify each of the documents in the test set

The goal is to infer the probability that document *i* belongs to category *k* given word profile $\mathbf{W}_i$

Applying Bayes's rule: $p(C_k|\boldsymbol{W}_i) = \mathrm{p}(C_k) * p(\boldsymbol{W}_i|C_k)$,

where we drop $p(\boldsymbol{W}_i)$ from the denominator since it is a constant across the different categories

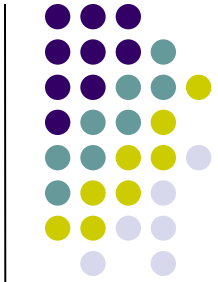
In plain English: $p(C_k|\boldsymbol{W}_i)$=posterior; $\mathrm{p}(C_k)$=prior; $p(\boldsymbol{W}_i|C_k)$= likelihood; $p(\boldsymbol{W}_i)$=evidence (the word profile we observe)

Naïve Bayes classifier

An example: let's say we have a training-set on 1000 pieces of fruit. The fruit being a Banana, Orange or some Other fruit. We know **3 features of each fruit**, whether it's long or not, sweet or not and yellow or not:

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	3	150	300	300
Other	100	150	50	200
Total	**503**	**650**	**800**	**1000**

Naïve Bayes classifier



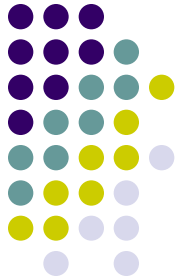
Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	3	150	300	300
Other	100	150	50	200
Total	**503**	**650**	**800**	**1000**

From the table we know that, in our training-set: 50% of the fruits are bananas; 30% are oranges; 20% are other fruits

Based on our training set we can also say the following:

Out of 500 bananas 400 (0.8) are Long, 350 (0.7) are Sweet and 450 (0.9) are Yellow; Out of 300 oranges 3 are Long (0.01), 150 (0.5) are Sweet and 300 (1) are Yellow; From the remaining 200 fruits, 100 (0.5) are Long, 150 (0.75) are Sweet and 50 (0.25) are Yellow

Naïve Bayes classifier



Now let's say we're given the features of a piece of fruit and we need to predict the class

If we're told that the **additional fruit is Long, Sweet and Yellow**, we can classify it using the Bayes formula and subbing in the values for each outcome, whether it's a Banana, an Orange or Other Fruit

The one with the highest probability (score) being the **winner class**!

Naïve Bayes classifier

p(Banana|Long, Sweet, Yellow)=p(Long|Banana=0.8)*p(Sweet|Banana=0.7)*p(Yellow|Banana=0.9)*p(Banana=0.5)=**0.252**

p(Orange|Long, Sweet, Yellow)=p(Long|Orange=0.01)*p(Sweet|Orange=0.5)*p(Yellow|Orange=1)*p(Orange=0.3)=**0.0015**

p(Other|Long, Sweet, Yellow)=p(Long|Other=0.5)*p(Sweet|Other=0.75)*p(Yellow|Other=0.25)*p(Other=0.2)=**0.01875**

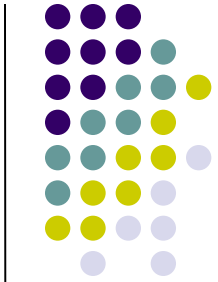
In this case, based on the **highest score**, we can classify this Long, Sweet and Yellow fruit as a Banana

Naïve Bayes classifier

Why Naïve?

Cause it assumes that every feature being classified is **independent** of the value of any other feature: in the previous example each of the three features (Long, Sweet, Yellow) are considered to *contribute independently* to the probability that the fruit is a Banana, *regardless of any correlations* between features

Naïve Bayes classifier



Why Naïve?

Features, however, aren't always independent!

However, although the model is *clearly wrong* – quite often features are not conditionally independent - it has proven to be a useful classifier for a diverse set of tasks

The same is true for text analysis: assuming that words are generated independently must be wrong given that the use of words is highly correlated in any data set. However, Naïve Bayes classifier can still be useful (remember the First Principle of text-analysis!)

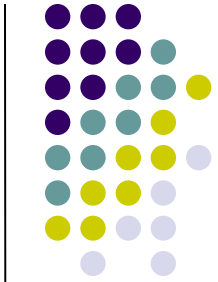
Naïve Bayes classifier

Naïve Bayes classifier algorithm is simple, fast but shows high performance for text classification

It is also an advantage that the algorithm show good performance with relatively small training dataset

Be aware however of using Naïve Bayes classifier when you have **high level of sparsity** (i.e., lots of cells in your matrix are filled with 0) in your term-document-matrix!

Random Forest classifier



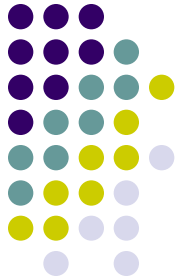
To understand random forest, let's start with what we mean by a Decision tree model

A **decision tree** is a **set of rules** used to classify data into categories

It looks at the variables in a data set, determines which are most important, and then comes up with a **tree of decisions which best partitions the data**

The tree is created by splitting data up by variables and then counting to see how many are in each bucket after each split

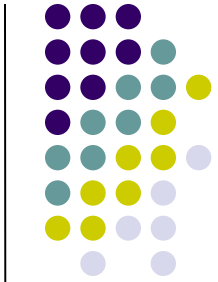
Random Forest classifier



At their core, decision trees models involve two basic steps:

1. First, they divide the covariate space into *B* nonoverlapping and exhaustive regions, R1, R2, . . . , RB, that are **relatively homogeneous** with respect to the outcome *y*
2. Second, they make a prediction for all observations that fall within region *Rb*

Random Forest classifier



The key idea is that the procedure to create decision trees is **recursive**. For a set (**S**) of observations, the following algorithm is applied:

1. If every observation in **S** is the **same class** or if **S** is very small, the tree becomes an **endpoint**, labeled with the *most frequent class*
2. If **S** is too large and it contains more than one class, find the **best rule** based **on one feature** to split it into subsets, one for each class (different rules can be used in this respect. The aim is always the same: the "best" branching rule is the one that results in the **most information gain**)

If you had to go to step 2, apply step 1 to each new subset. If your subsets need to go to step 2, apply step 1 to the sub-subsets, etc. When everything is split up appropriately (into buckets that are very small or entirely one class), you have a set of rules that look like a tree!

Random Forest classifier

Example: Given only the gender and weight of a person, can we predict whether they are Japanese or American (our 2 classes/categories)?

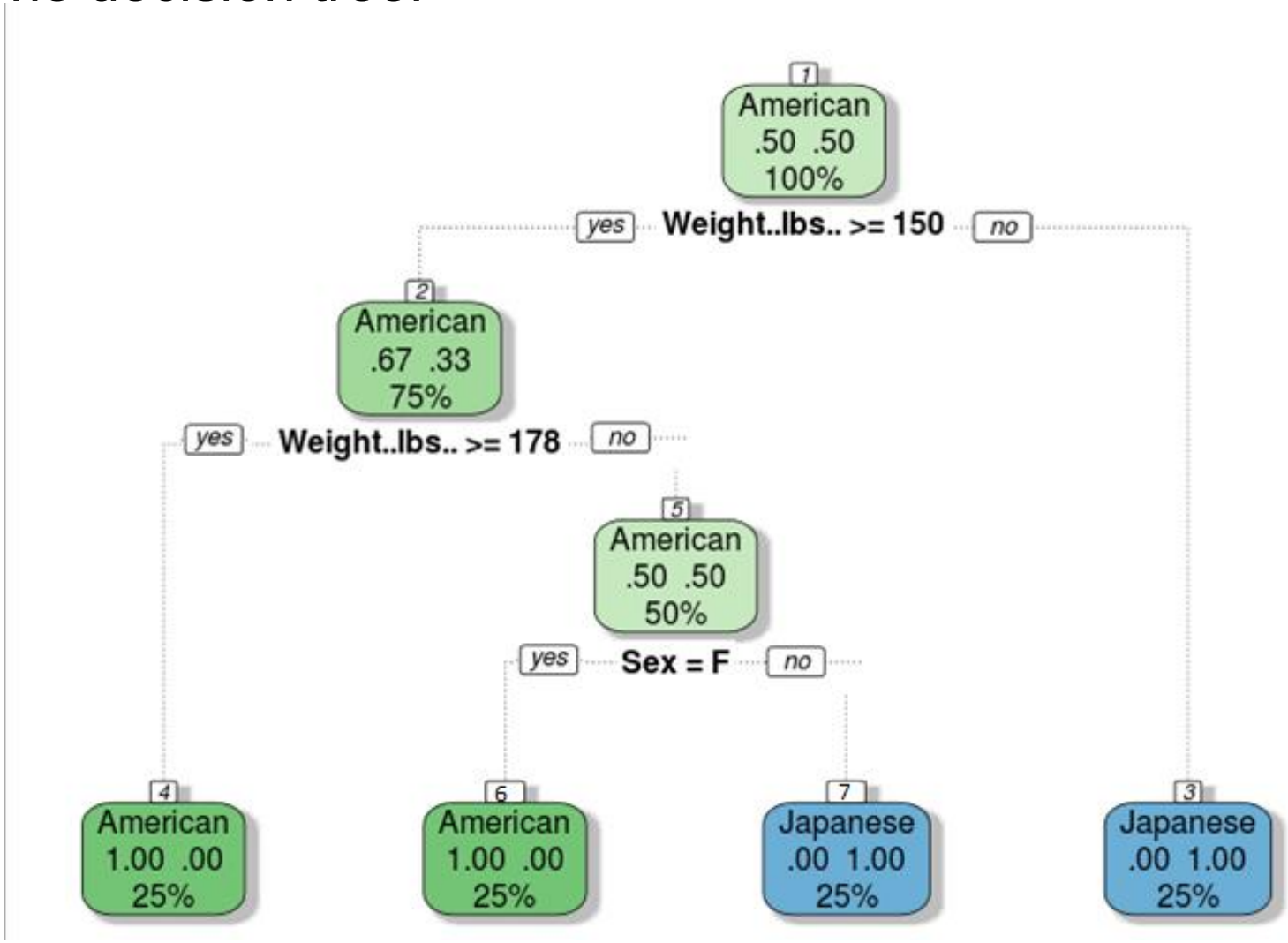
Our training set:

Weight (lbs.)/Sex/Nationality

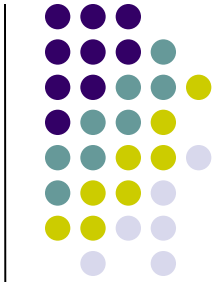
195 M American
190 M American
160 F American
165 F American
165 M Japanese
160 M Japanese
130 F Japanese
140 F Japanese

Random Forest classifier

The decision tree:



Random Forest classifier



In this example the tree can **perfectly explain** the data

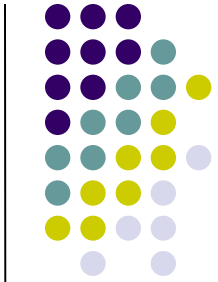
In the real world, there is overlap: there are fat Japanese people and skinny Americans

In this case, the model would be optimized to make the largest possible number of correct predictions

Trees are usually "pruned" to avoid **overfitting**. The pruning algorithm removes final nodes so that the model is a little more general

Overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably

Random Forest classifier



Overfitting is a risk always present everytime you run a machine learning algorithm!!!

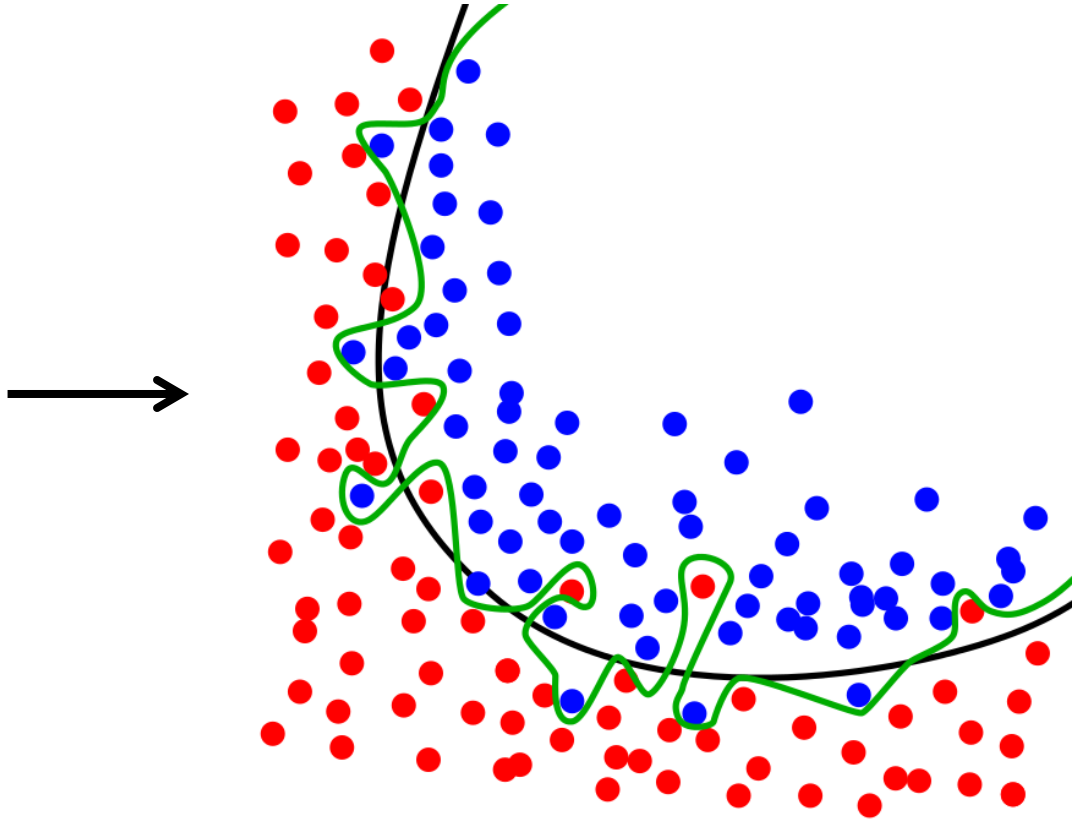
The essence of overfitting is to have unknowingly extracted some of the residual variation (i.e. the noise) as if that variation represented underlying model structure

An overfitted model is a statistical model that contains more parameters than can be justified by the data

Random Forest classifier

Two examples of overfitting

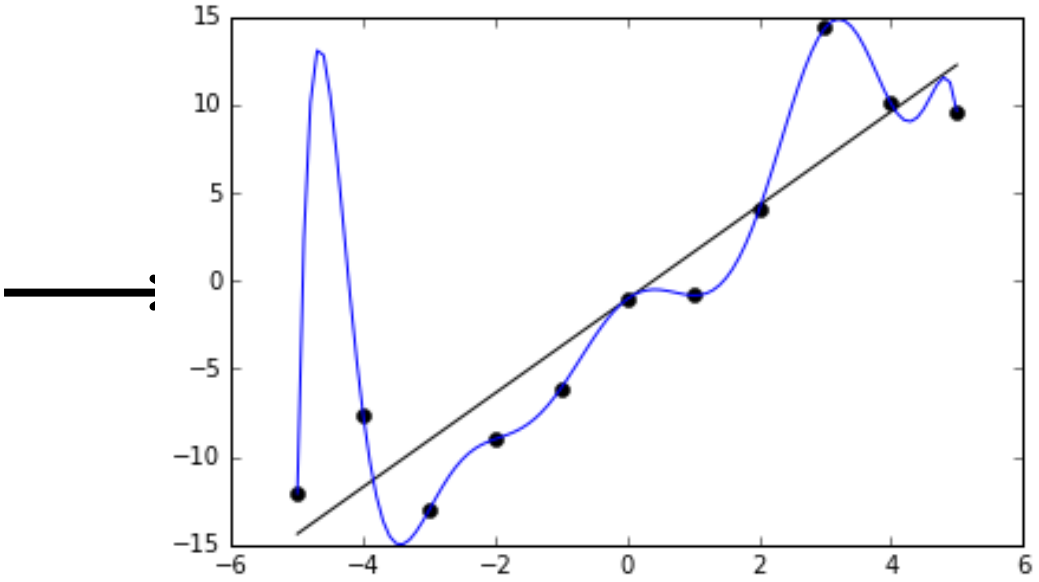
While the green line best follows the training data, it is too dependent on that data and it is likely to have a higher error rate on **new unseen data**, compared to the **black** line



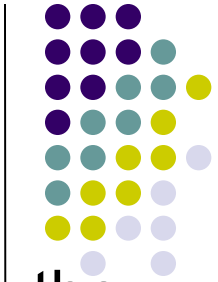
Random Forest classifier

Two examples of overfitting

Although the polynomial function (the blue line) is a perfect fit, the linear function can be expected to generalize better beyond the fitted data!



Random Forest classifier



"Pruning" however is not always an advisable solution to the problem of overfitting!

First, the algorithmic approach to building a tree leaves us with no means for assessing uncertainty in our estimates

Second, the sequential nature of the recursive splitting algorithm means that the structure of the tree is often highly sensitive to small changes in the observations included

So what to do?

Random Forest classifier

Back to random forest (RF) now! RF is like a **bootstrapping algorithm with Decision tree model**!

Say, we have 1000 observation in the complete population with 10 variables

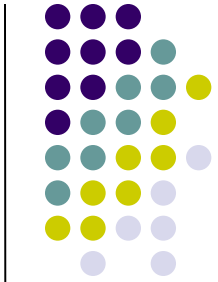
RF tries to build multiple Decision tree models **with different sample and different initial variables**

More formally, RF takes multiple simple random samples of the same data set (with replacement), of size equal to that of the original data set. It then fits a tree (with no pruning) to each bootstrapped sample

Final prediction is a function of each prediction in each random sample. This final prediction can simply be **the mean of each prediction**

All this, of course, minimizes the risk of overfitting!

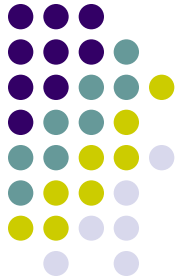
Support Vector Machine (SVM)



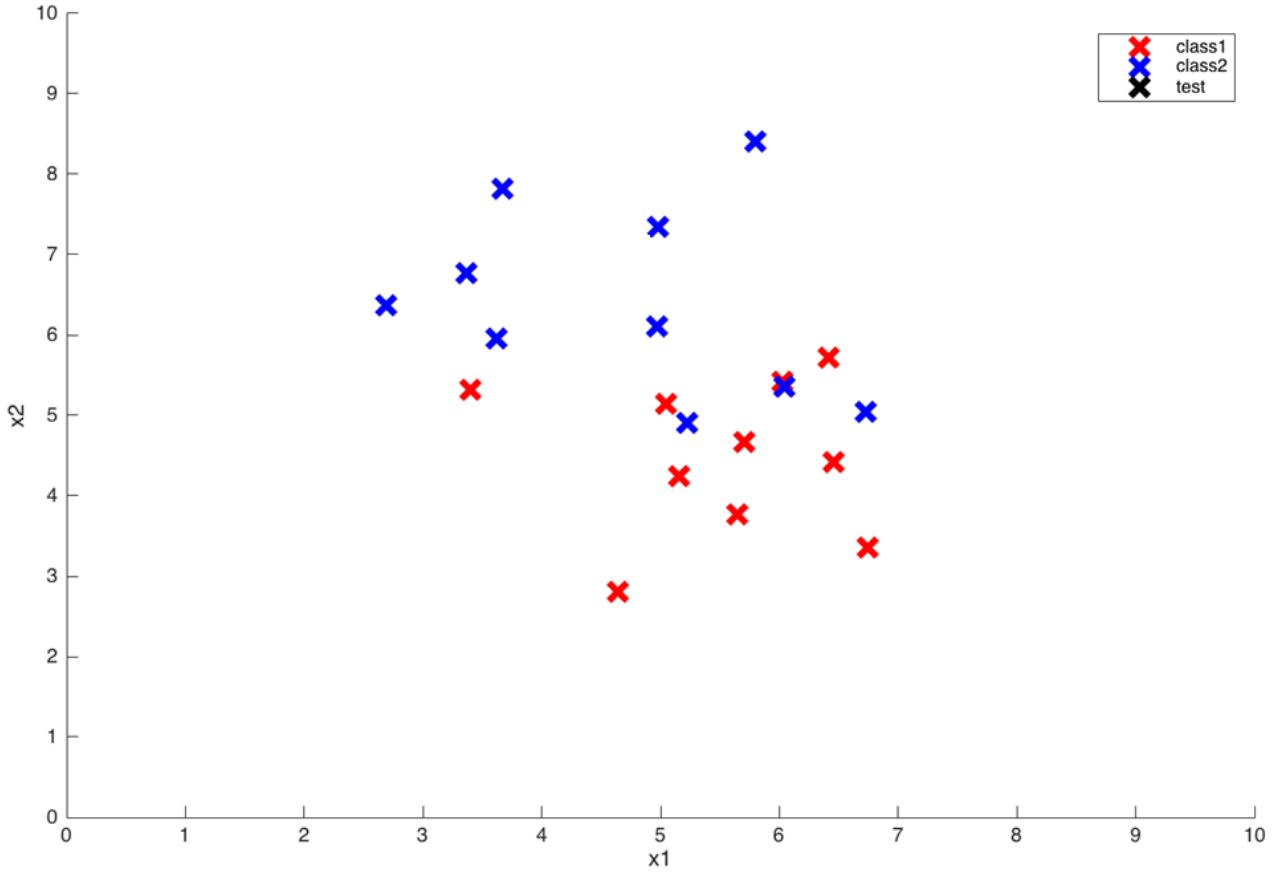
SVM is a generalization of **Nearest Neighbor** (NN) algorithm

NN is a very simple algorithm. You are given a training data consisting of *m* training documents $\{(\boldsymbol{x}^1, y^1), (\boldsymbol{x}^2, y^2), \ldots (\boldsymbol{x}^m, y^m)\}$, where x is a vector of possible variables and y^i is the class label (the category) of i^{th}

Support Vector Machine (SVM)

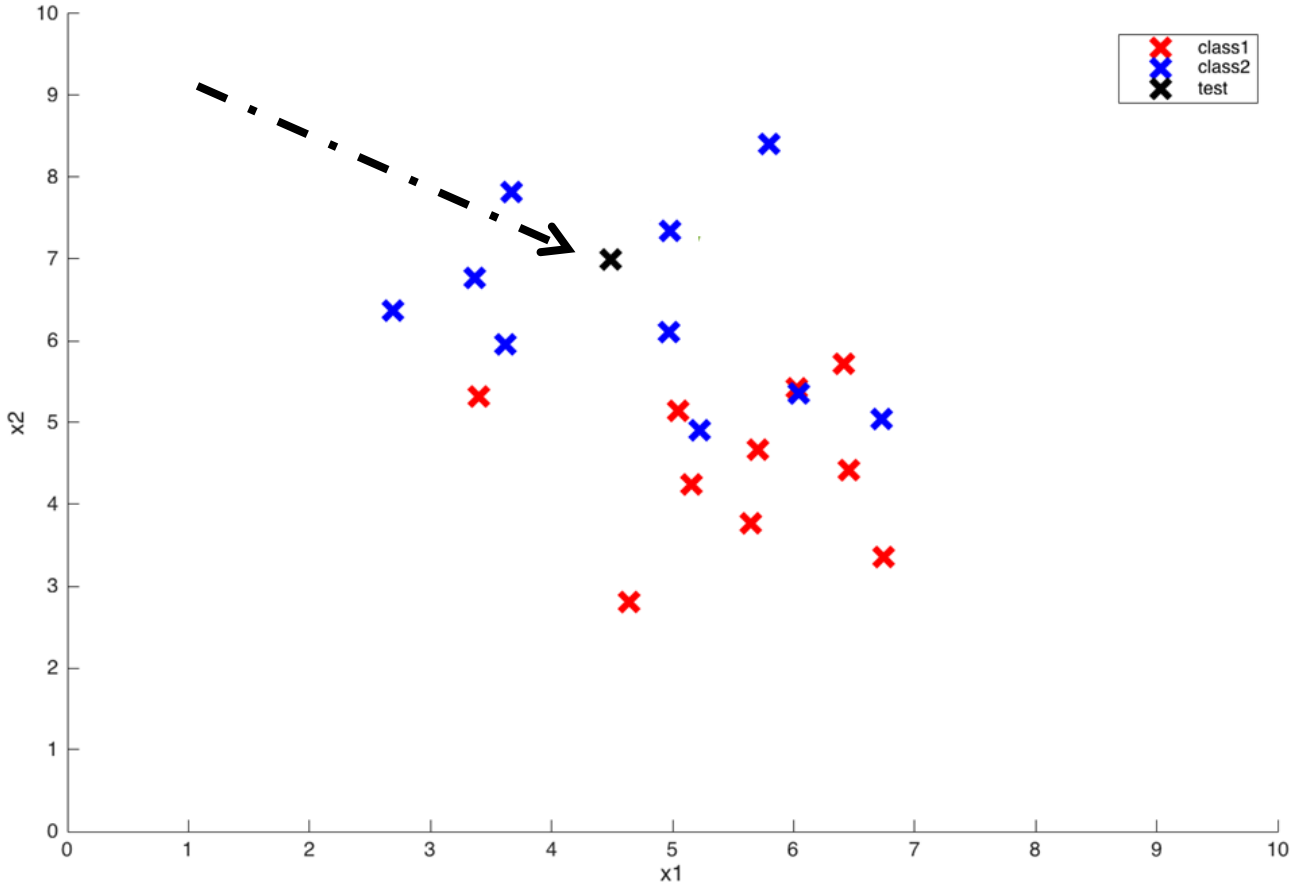


For example, in the figure here, with two Xs, each document can be either label 1 (blue points), or label −1 (red points)

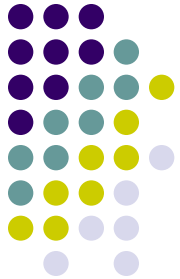


Support Vector Machine (SVM)

Now you are **given a test point** (the black x below), and you have to predict its class, whether it belongs to red class or blue class



Support Vector Machine (SVM)

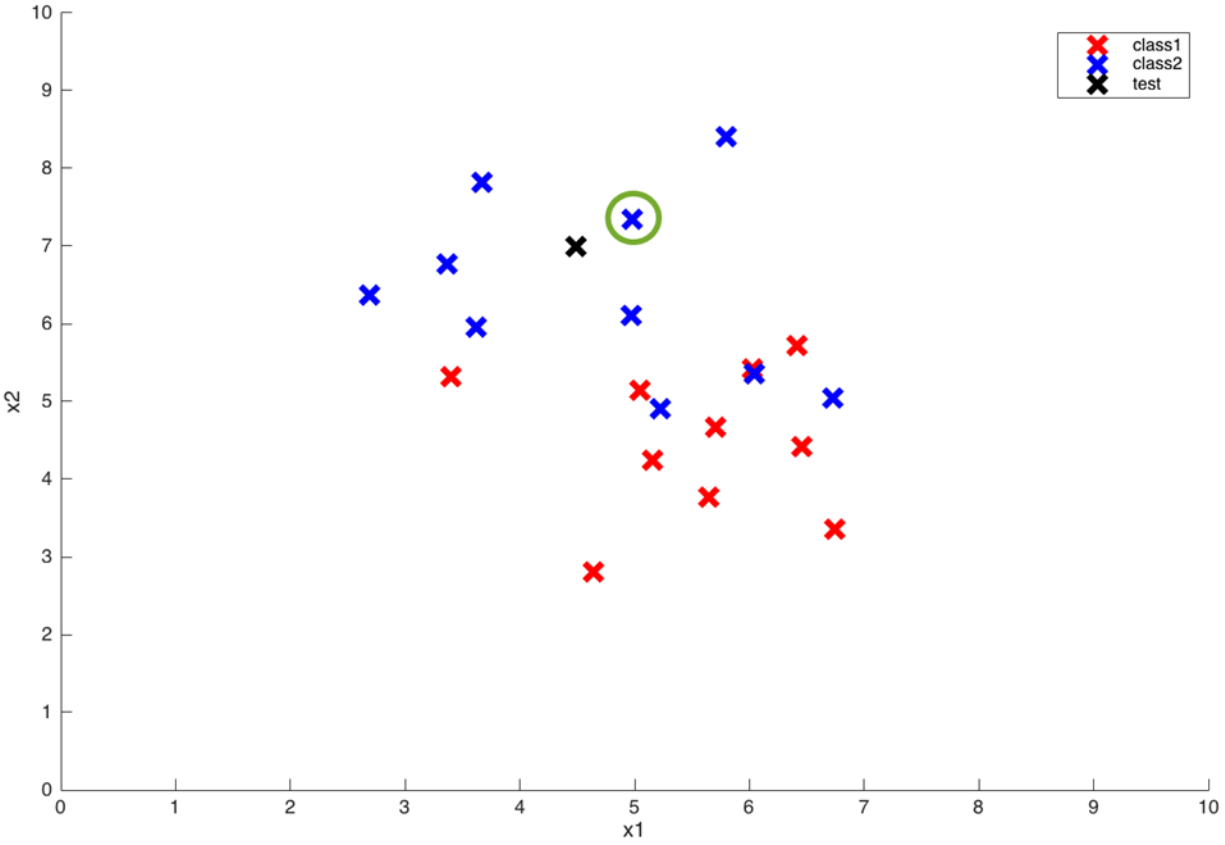


In NN algorithm, you find the nearest training point to this test point (by measuring the distance of test point from **every** training point), and the class predicted of test point is same as of nearest training point

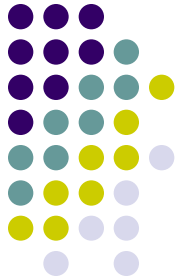
Lower the distance, higher the **similarity** between two points

Support Vector Machine (SVM)

For example, the circled point is closest to test point, and hence class of test point is blue



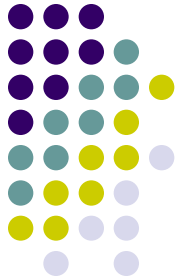
Support Vector Machine (SVM)



Two observations from NN algorithm:

- ✓ We don’t do any computation alone with training points. Only when a test point comes, we compute similarity from **every** training point. This is a big disadvantage of NN algorithm. Consider having millions of training points, and for every test point, we have to calculate millions of similarities from test point. We calculate similarities from the training points which are very far from test points, which are not really required
- ✓ We don’t give any importance to other training points except the nearest one

Support Vector Machine (SVM)



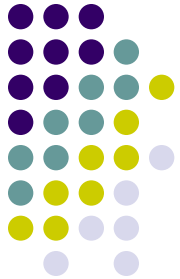
SVM remove each of these problems:

Instead of finding similarity from every training point of any test point, we calculate similarity from only a **subset** of training points, which we compute in the **training phase**

These selected training points are called **support vectors,** since only these points will support our decision of selecting the class of a test point

Our hope is that our training phase finds as few as support vectors so that we have to compute fewer number of similarities

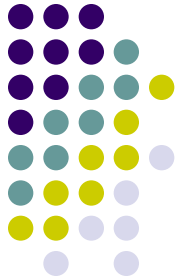
Support Vector Machine (SVM)



Moreover, once we have selected support vectors, we assign a **weight** to each support vector, which basically tells how much importance we want to give to that support vector while making our decision

So unlike NN, we don’t give importance to only a single training point (document), instead we give each support vector a separate importance

Support Vector Machine (SVM)



But how to find a **support vector**?

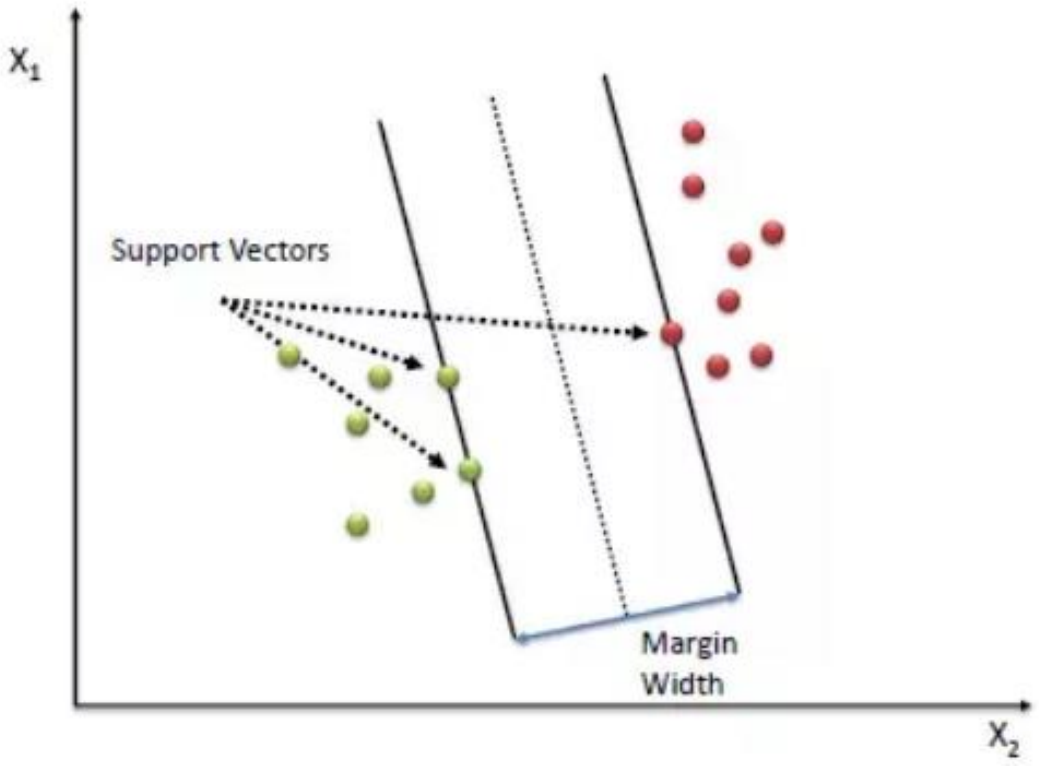
It involves finding the hyperplane (line in 2D, plane in 3D and hyperplane in higher dimensions. More formally, a hyperplane is n-1 dimensional subspace of an n-dimensional space) that **best separates two classes of points with the maximum margin** (i.e., we try to find that separating hyperplane from which distance of closest training points is maximum)

Essentially, it is a **constrained optimization problem** where the **margin is maximized** subject to the constraint that it **perfectly classifies the data**

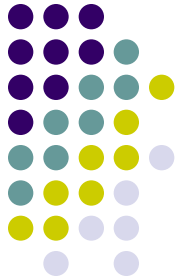
Intuitively, this works well because the "maximum-margin" line allows for noise and is most tolerant to mistakes on either side

Support Vector Machine (SVM)

Suppose that we want to split the below red circles from the green ones by drawing a line. Notice that there are an infinite number of lines that will accomplish this task, but only one "maximum-margin" line



Support Vector Machine (SVM)



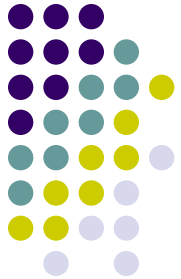
The data points that kind of "support" this hyperplane on either sides (i.e., the closest training points to the line) are called the "**support vectors**“

Support Vectors are simply the co-ordinates of individual observation

In the figure, there are only 3 support vectors, so at the test time, we will compute similarity test point on only these 3 support vectors

That's all a SVM really is: we draw a straight line through our data down the middle to separate it into two classes

Support Vector Machine (SVM)



An what about the “**weights**”?

Let's say the SWM, in a 2-dimensional setting, would find only one feature useful for separating the data (say x1 or x2), then the hyperplane would be **orthogonal to that axis**

In this instance, the absolute size of the coefficient relative to the other ones gives an indication of how important the feature was for the separation

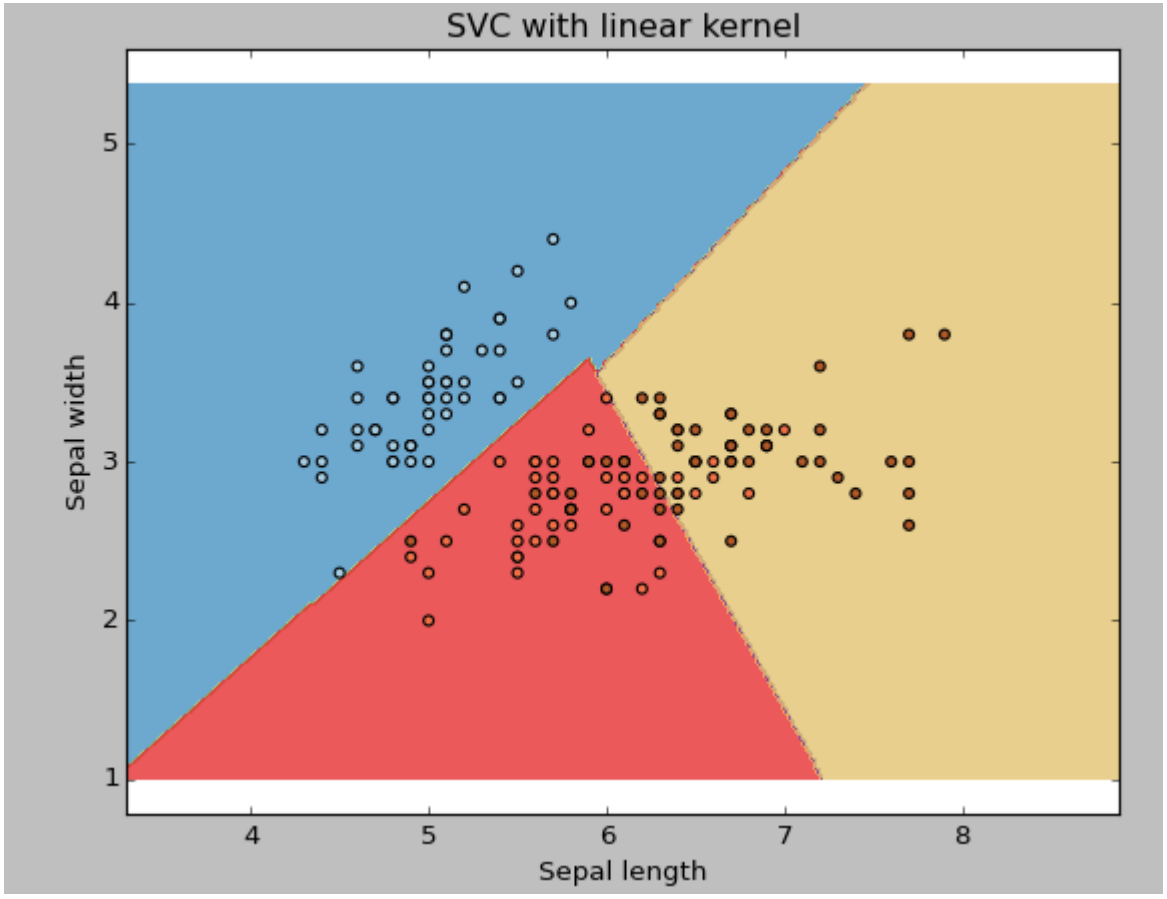
For example if only the first coordinate is used for separation, the weights will be of the form (x,0) where x is some non zero number and then $|x|>0$

In the previous example, the x1 coordinates will “weight” more than the x2 coordinates to classify new cases!

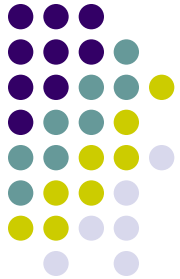
Support Vector Machine (SVM)

An example with the Iris dataset (more on the Lab class!):

we want to predict the species of Iris (3 possibilities) by using the sepal width and lenght



Support Vector Machine (SVM)



But what if we don't want (or cannot fit) a straight line to find support vectors? **Move to a curved one**!

We achieve this not by drawing curves, but by "lifting" the features we observe into higher dimensions

For example, if we can't draw a line in the space (x1, x2) then we may try adding a third dimension, (x1, x2, x1 * x2). If we project the "line" (actually called a "hyperplane") in this higher dimension down to our original dimension, it looks like a curve

This is known as a "**kernel trick**" (kernel is nothing but a similarity measure between two transformed vectors)

SVM works pretty well, but compared to RF, it usually requires a **larger training-set**

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

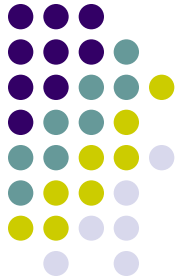
Validation

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

If a method is performing well, it will **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

Validation



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

Initial model fitting would be performed on the training-set

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

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: V-fold validation;** sometimes also **K-fold)** can be used to replicate this ideal procedure

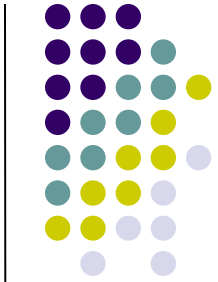
In V-fold cross-validation, the training set is randomly partitioned into some groups (say two: V1 and V2). In this case we have a V-fold cross-validation (v=2)

Validation

For each group, the first model is trained on V1, then applied to the V2 to assess performance; similarly a model is trained on the V2 and then applied to V1 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 V-fold cross-validation with V 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
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 V-fold cross-validation=3…

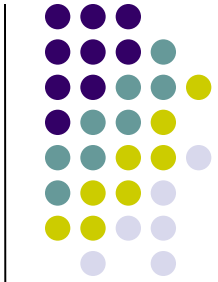
You split your training-set into 3 sets; you use V2+V3 to estimate V1; then V1+V3 to estimate V2; then V1+V2 to estimate V3. Then you take the average across the three scenarios

Validation

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

Usually V-fold cross-validation in the literature employs v=5 or v=10

Validation

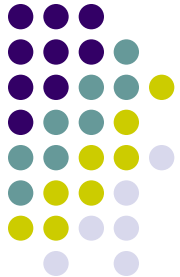


When you run a ML algorithm on the test-set, there are no available statistics to control for the goodness-of-fit of your prediction (by definition the “true” values of the test-set are unknown!)

That is why **cross-validation** is so important! This 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)!

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 should we us to assess model performance?

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

Validation

Accuracy: proportion of correctly classified documents

Validation

Precision or **Positive Predictive Value** (for a category k):
number of documents correctly classified into category k, divided by the total number of documents that the model classifies as category k (i.e., given that the machine guesses category k, what is the probability that the machine made the right guess?)

Validation

Recall or **Sensitivity** (for a category k): number of correctly classified category k documents divided by the number of human coded documents in category k (given that a human coder labels a document as belonging to category k, what is the chance the machine identifies the document?)

Validation: an example

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

Suppose this matrix (also called the «**confusion matrix**») is the result of our classification

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

Note however that in some given circumstances Accuracy is not a reliable metric for the real performance of a classifier, because it will yield misleading results if the **data set is unbalanced** (that is, when the numbers of observations in different classes vary greatly)

For example, if there were 95 cats and only 5 dogs in the data, a particular classifier **might classify all the observations as cats**

The overall accuracy would be then…how much?

True Positive over the entire dataset: 95%!

Validation

However, in this same circumstance, the classifier would have a **recall rate (sensitivity)** for the *cat class* equals to…?

Validation

How to estimate precision and recall for each single category *k* in a nutshell. Some jargon:

"*true positive*" (TP) for correctly predicted event values (the diagonal of the confusion matrix)

"*false positive*" (FP) for incorrectly predicted event values (for a certain class: the sum of values in the corresponding row, excluding the TP)

"*false negative*" (FN) for correctly predicted no-event values (for a certain class: the sum of values in the corresponding column, excluding the TP)

"*true negative*" (TN) for incorrectly predicted no-event values (for a certain class: the sum of values of all columns and rows excluding that class' column and row)

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:

The Precision for a given class *k* is given as: True Positive/Predicted Positive or: TP/(TP+FP))

The Recall for a given class *k* is given as: True Positive/Actual Positive or: TP/(TP+FN)

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

Recall is: 5/(5+3)=0.625

Validation

Back to our example

We 95 cats and only 5 dogs in the data, and a particular classifier has classified **all the observations as cats**

In this circumstance, accuracy would be equal to .95

But the **recall rate (sensitivity)** for the cat class equals to…?

….100%!

And a recall rate for the dog class equals to…?

0%!!

What about precision?

0.95 for cat class; 0 for dog class

Validation

If you find for example market differences between the **precision and recall** (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)

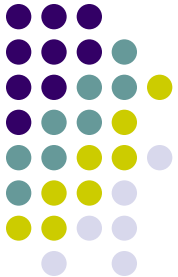
This sometimes applies when the original *k* category, compared to the other k_{n-1} categories, is the most relevant category in the training-set (more on this in next week!)

Validation

Depending on the application, scholars may conclude that the supervised method is able to sufficiently replicate human coders

Or, additional steps can be taken to improve accuracy, including: applying other methods…or switching the quantity of interest to proportions!!!!

R pakcages to install



install.packages("gridExtra", repos='http://cran.us.r-project.org')

install.packages("syuzhet", repos='http://cran.us.r-project.org')

install.packages("plotly", repos='http://cran.us.r-project.org')

install.packages("e1071", repos='http://cran.us.r-project.org')

install.packages("caTools", repos='http://cran.us.r-project.org')

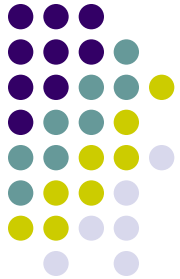
install.packages("randomForest", repos='http://cran.us.r-project.org')

install.packages ("caret", ='http://cran.us.r-project.org')

install.packages ("stringr", ='http://cran.us.r-project.org')

install.packages ("reshape2", ='http://cran.us.r-project.org')

R pakcages to install for NEXT week!



install.packages("rtweet", repos='http://cran.us.r-project.org')
install.packages("streamR", repos='http://cran.us.r-project.org')
install.packages("dplyr", repos='http://cran.us.r-project.org')
install.packages("tidytext", repos='http://cran.us.r-project.org')
install.packages("httpuv", repos='http://cran.us.r-project.org')
install.packages("maps", repos='http://cran.us.r-project.org')
install.packages ("leaflet", ='http://cran.us.r-project.org')
install.packages ("ROAuth", ='http://cran.us.r-project.org')
install.packages ("RCurl", ='http://cran.us.r-project.org')
install.packages ("grid", ='http://cran.us.r-project.org')
install.packages ("tm", ='http://cran.us.r-project.org')
install.packages ("wordcloud", ='http://cran.us.r-project.org')