Coursera: Practical Machine Learning

[*Jeffrey Leek - John Hopkins Bloomberg School of Public Health*]

Course Description

One of the most common tasks performed by data scientists and data analysts are prediction and machine learning. This course will cover the basic components of building and applying prediction functions with an emphasis on practical applications. The course will provide basic grounding in concepts such as training and tests sets, overfitting, and error rates. The course will also introduce a range of model based and algorithmic machine learning methods including regression, classification trees, Naive Bayes, and random forests. The course will cover the complete process of building prediction functions including data collection, feature creation, algorithms, and evaluation.

Course Content

- Prediction study design
- In sample and out of sample errors
- Overfitting
- Receiver Operating Characteristic (ROC) curves
- The caret package in R
- Preprocessing and feature creation
- Prediction with regression
- Prediction with decision trees
- Prediction with random forests
- Boosting
- Prediction blending

Week 1: Prediction, errors and cross-validation

Index:

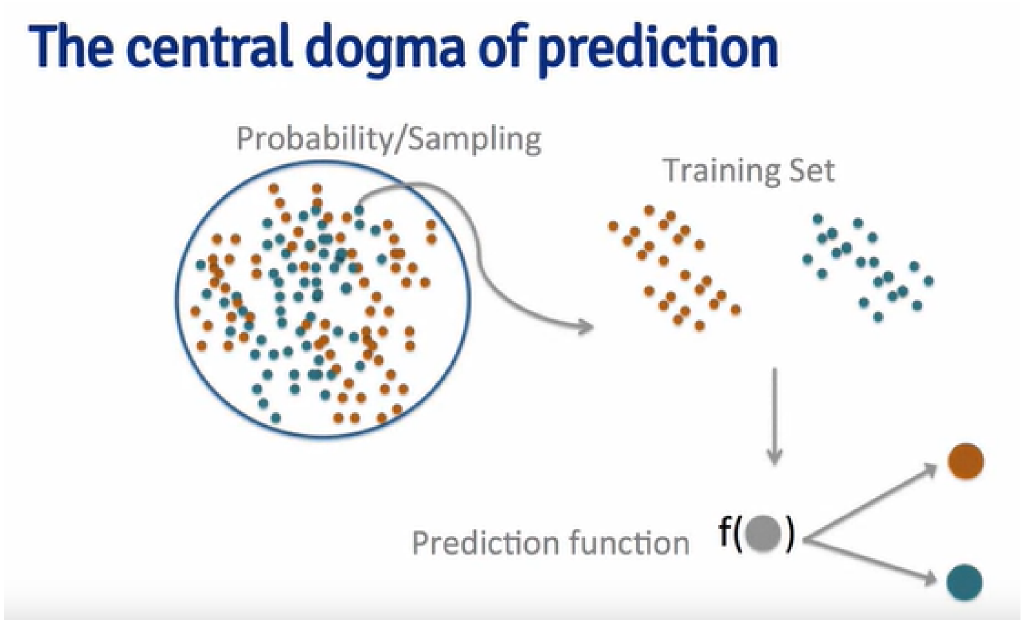
Week 1.1: Motivation and Pre-requisites

About this course:

This course covers the basic ideas behind machine learning/prediction:

- Study design: **training** vs **test** sets
- Conceptual issues: out of **sample error** and **overfitting**, **ROC curves** and other **methods for evaluating predictors**.
- Practical implementation: the **caret package** (a nice unifying framework for a lot of ML packages that exist in R)

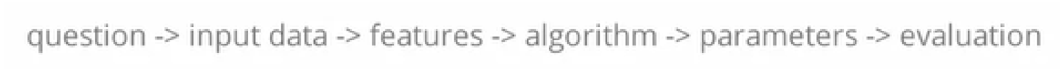
Week 1.2: What is prediction?



- You want to predict whether the dots are red or blue.
- You have a big group of dots and via probability and sampling you extract a training set.
- The training set will include some blue and some red dots and a bunch of other characteristics of those dots.
- You will use those characteristics to build a prediction function which can predict whether a new dot is either red or blue using those characteristics you have used to build the function itself.
- Then you can proceed and evaluate whether that prediction function works well or not.

The extraction of the training set using probability and sampling is often overlooked because a lot of attention in ML is focussed on the prediction function stage --> but choosing the training dataset is paramount!

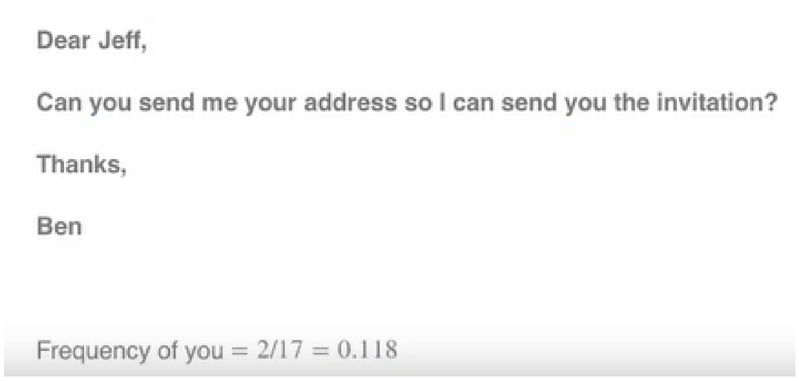
Components of a predictor


question -> input data -> features -> algorithm -> parameters -> evaluation

1) A very specific and well-defined **question** is required to start with: what are you trying to predict? what are you trying to predict it with?
2) You need then to collate the best **input data** that you can, so to enable prediction.
3) From the input data you might either use **measured characteristics** that you already have or you might use computations to build **features** that you think might be useful to predict the outcome you care about.
4) You can then start to use ML algorithm/s you want to apply.
5) You can then **estimate** the **parameters** of those algorithms.
6) You can use those parameters to apply the algorithm to a new data set and **evaluate the algorithm** on that new data.

SPAM Example

- Start with a general question: *Can I automatically detect emails that are SPAM and those that are not?*
- Make it concrete: *Can I use quantitative characteristics of the emails to classify them as SPAM/HAM?*
- Find input data (e.g. SPAM dataset in the R kernlab package)
- We then need to calculate something about features: e.g. frequency of the appearance of the word "you" in each email of the dataset.


Dear Jeff,

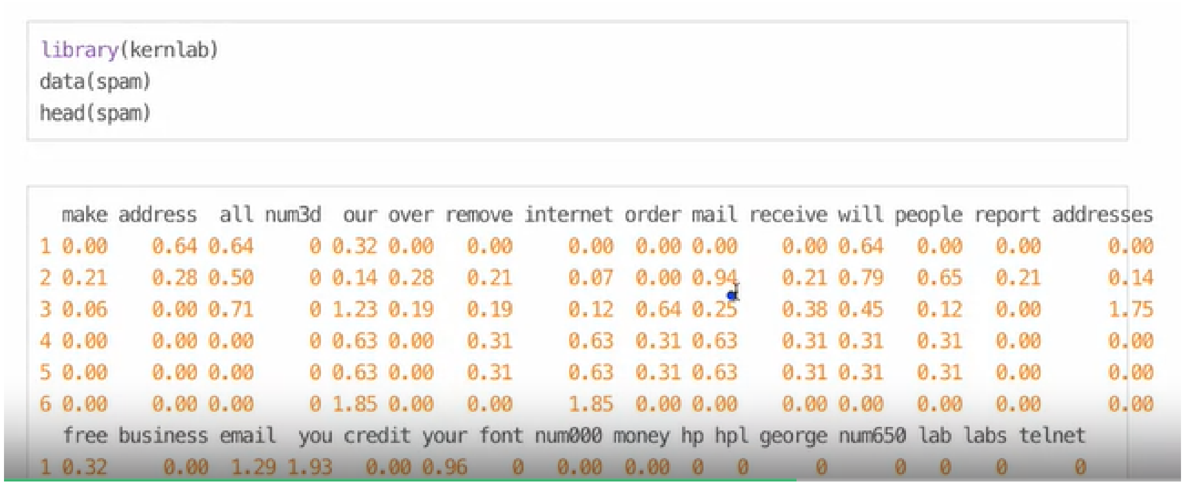
Can you send me your address so I can send you the invitation?

Thanks,

Ben

Frequency of you = 2/17 = 0.118

- In the kernlab SPAM dataset we have such kind of features: for each email the frequency that specific words appear in it:

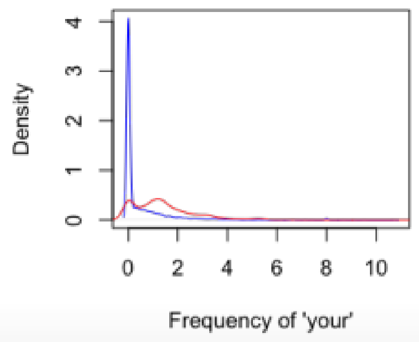


```
library(kernlab)
data(spam)
head(spam)
```

```
  make address  all num3d  our over remove internet order mail receive will people report addresses
1 0.00    0.64 0.64     0 0.32 0.00   0.00     0.00  0.00 0.00    0.00 0.64   0.00   0.00      0.00
2 0.21    0.28 0.50     0 0.14 0.28   0.21     0.07  0.00 0.94    0.21 0.79   0.65   0.21      0.14
3 0.06    0.00 0.71     0 1.23 0.19   0.19     0.12  0.64 0.25    0.38 0.45   0.12   0.00      1.75
4 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31 0.31   0.31   0.00      0.00
5 0.00    0.00 0.00     0 0.63 0.00   0.31     0.63  0.31 0.63    0.31 0.31   0.31   0.00      0.00
6 0.00    0.00 0.00     0 1.85 0.00   0.00     1.85  0.00 0.00    0.00 0.00   0.00   0.00      0.00
  free business email  you credit your font num000 money hp hpl george num650 lab labs telnet
1 0.32     0.00  1.29 1.93   0.00 0.96    0   0.00  0.00  0   0      0      0   0    0      0
```

- As an example, we have looked at the frequency of the appearance of the word 'your' in the emails of the SPAM dataset. Here is a density plot of the data (on the x-axis is the frequency the word 'your' appeared in the email, while on the y-axis is the density or number of times that frequency appears amongst email). Most of the SPAM emails are in red and tend to have more appearances of the word 'your':

```
plot(density(spam$your[spam$type=="nonspam"]),
     col="blue",main="",xlab="Frequency of 'your'")
lines(density(spam$your[spam$type=="spam"]),col="red")
```

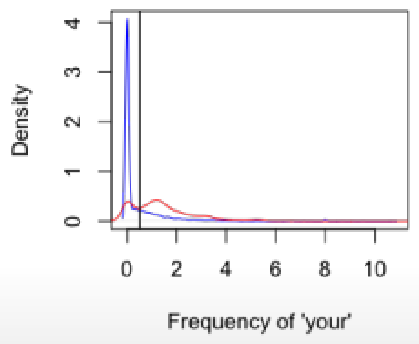


- We want to build an algorithm to find a constant cut-off:

`frequency of 'your' > cutoff --> predict 'spam', otherwise 'ham`

This is an example of cut-off you can use: if the frequency of 'your' in the email is > 0.5 then the email is classified as 'spam':

```
plot(density(spam$your[spam$type=="nonspam"]),
     col="blue",main="",xlab="Frequency of 'your'")
lines(density(spam$your[spam$type=="spam"]),col="red")
abline(v=0.5,col="black")
```



We then evaluate our classifier, evaluating the predictions for each of the different emails. We make a table of counts of those predictions and divide the counts but the total number of observations that we have:

```
prediction <- ifelse(spam$your > 0.5,"spam","nonspam")
table(prediction,spam$type)/length(spam$type)
```

```
prediction nonspam   spam
   nonspam  0.4590 0.1017
   spam     0.1469 0.2923
```

Accuracy$\approx 0.459 + 0.292 = 0.751$

Having the accuracy been calculated on the same dataset used to create the prediction function, it is an optimistic estimate of the overall error rate.

Week 1.3: Relative importance of steps

This lecture is about the trade-offs and the different components of building a ML algorithm.

Recall the different components of building a good ML algorithm:

- **Questions** --> the most important part. You need a very concrete and specific question
- **Data** --> the next most important step is to be able to collect
- **Features** --> if you don't compress the data in the right way you might loose valuable information

- **Algorithms** --> it is often the least important part, although it can still be very important, depending on the type/kind of data you use (e.g. image and voice data require specific kinds of algorithms).

An important point

"*The combination of some data and an aching desire for an answer does not ensure that a reasonable answer can be extracted from a given body of data*" (John Tukey)

An important component of knowing how to do a prediction is to know when to give up, when the data you have is just not sufficient to answer the question that you are trying to answer.

Input data: Garbage in = Garbage out

If you have bad data that you have collected, or that data is not very useful for performing predictions, no matter how good your ML algorithm is, you will often obtain very bad results out.

1) In general, the easiest thing to predict is when you have data on the exact same thing that you are trying to predict.
(may be easy: movie ratings -> new movie ratings,
may be harder: gene expression data -> disease)
2) It depends on what is a "good prediction"
4) Often: **more data > better models**!
5) The most important step is collecting the right data that is relevant for the question you are trying to answer.

Features matter!

Properties of good features:

- Lead to data compression
- Retain relevant information
- Are created based on expert application knowledge (there is a huge debate in the community whether it is better to create features automatically or using expert domanin knowledge)

The important properties of good features are that they compress data in a way that makes it possible to compute your prediction in very simple way. Features might be more interpretable than the raw data you have collected (which might be complicated)

Common mistakes:

- Trying to automate feature selection (especially in a way that does not make you understand how those features are actually being applied to make good predictions --> "black-box" predictions can be very useful/accurate, but they can also change on a dime if we are not paying attention to how those features actually do predict the outcome)
- Not paying attention to specific data quirks (there might be outliers in the dataset or weird behaviours of specific features, and not understanding those can cause problems)
- Throwing away information unnecessary

Features may be automated with care

There is a whole area of research (sort of **semi-supervised learning** or **deep learning**).

A research project tried to discover features of YouTube videos that could later be used in prediction algorithms --> for example, they were able to create a very accurate predictor of whether there was a cat in the video or not based on a bunch of features they sort of collected in an unsupervised way. They filtered through the data in a way to identify those features that might be useful for later predictive algorithms. However, they still try to identify why those features were predictive.

http://arxiv.org/pdf/1112.6209v5.pdf

Algorithms matter less than you think

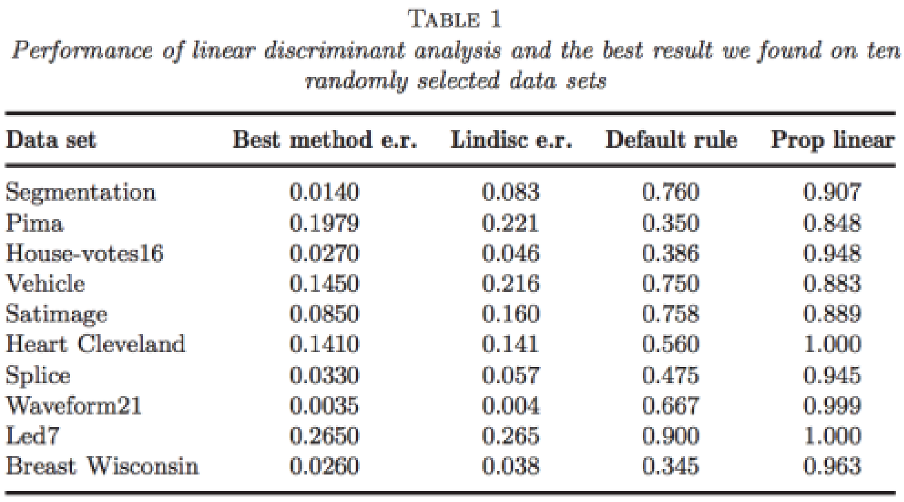


TABLE 1
Performance of linear discriminant analysis and the best result we found on ten randomly selected data sets

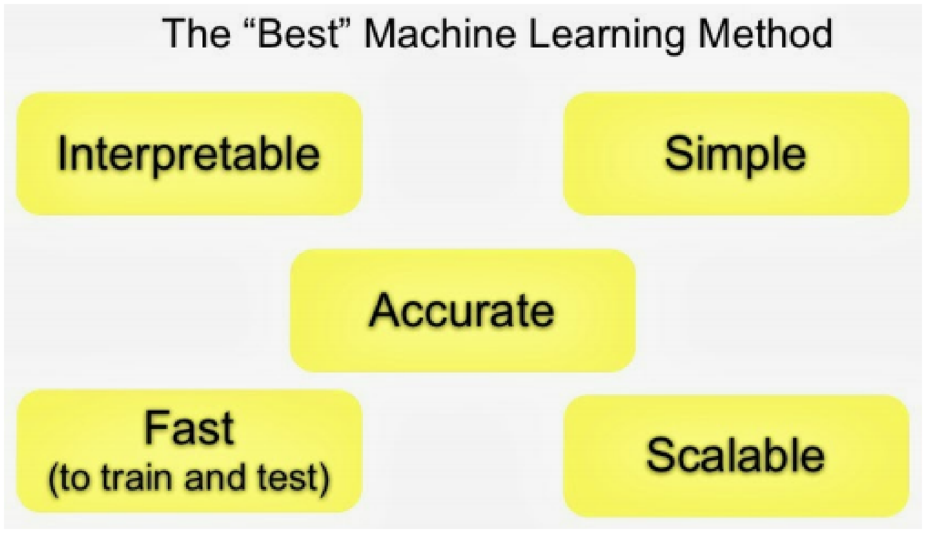
Data set	Best method e.r.	Lindisc e.r.	Default rule	Prop linear
Segmentation	0.0140	0.083	0.760	0.907
Pima	0.1979	0.221	0.350	0.848
House-votes16	0.0270	0.046	0.386	0.948
Vehicle	0.1450	0.216	0.750	0.883
Satimage	0.0850	0.160	0.758	0.889
Heart Cleveland	0.1410	0.141	0.560	1.000
Splice	0.0330	0.057	0.475	0.945
Waveform21	0.0035	0.004	0.667	0.999
Led7	0.2650	0.265	0.900	1.000
Breast Wisconsin	0.0260	0.038	0.345	0.963

In the table above, they tried to predict a variety of different outcomes for different datasets (e.g. US House of Representatives, a segmentation task, predicting waveforms, etc ...). The prediction was carried out in 2 different ways:

- Using linear discriminant analysis (a very basic predictor)
- The best prediction algorithm that could have been found for that dataset.

The table shows the prediction errors (Best method e.r. and Lindisc e.r.) for those two different approaches --> the best algorithm prediction error is always better than the linear discriminant error, but not that far off --> using the same method (one that is good and sensible enough for the specific analyses) over and over again did not make the prediction error much worse than using the best algorithm. Finding then the absolute best method does not result in a highly significant improvement in the prediction error score.

Issues to consider



http://strata.oreilly.com/2013/09/gaining-access-to-the-best-machine-learning-methods.html

- The predictor should be **interpretable** so also other (non-expert) people could understand it.
- Part of being "interpretable" means often being **simple** and easy to explain
- Often you have a trade-off with respect of **accuracy** in terms of interpretability and simplicity, as often a simpler and more interpretable algorithm is less accurate, so there is a choice to make where to set the bar.
- The model needs to be **fast** to train and test in small samples
- **Scalable** means that it is very easy to apply the model to a large dataset, either because it is very fast or because it is parallelisable across multiple samples, for example.

Prediction is about accuracy trade-off

- **Interpretability** vs accuracy --> interpretability is often important in medical studies (e.g. using decision tree algorithms allow which can be interpreted as if...else statements, i.e. prediction rules)
- **Speed** vs accuracy
- **Simplicity** vs accuracy
- **Scalability** vs accuracy --> e.g. if the algorithm is a blend of many other ML algorithms it might not scale up very well

Generally, you want to find the right balance between those other properties of the prediciton algorithm that

Week 1.4: In and out of sample errors

This is one of the most fundamental concepts we deal with in ML and prediction.

In sample versus out of sample

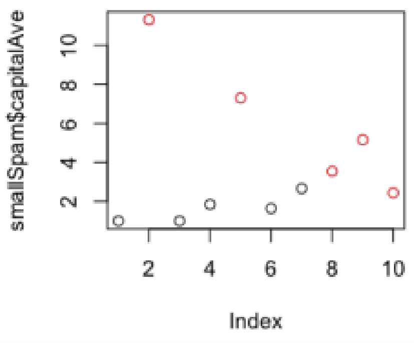
- **In Sample Error**: the error rate you get on the **same dataset you used to build your predictor**. Sometimes it is called **redistribution error**.
- **Out of Sample Error**: the error rate you get on a **new dataset**. Sometimes it is called **generalisation error**.

Key ideas:
1) **Out of sample error is what you care about**
2) **In sample error < out of sample error**
3) The reason is **overfitting** (i.e. matching your algorithm to the data you have) -> the in sample error is always a little optimistic from what is the error you would get from a new sample. This happens because in your specific sample sometimes the prediction algorithm will tune itself a bit to the noise that has been collected on a particular dataset. Thus, when you use it on a new dataset, there will be some different noise and the algorithm's accuracy will go down a little bit.

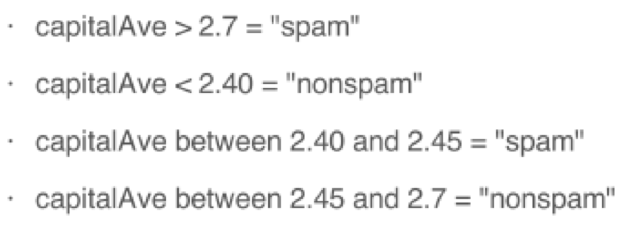
In order to avoid the overfitting issue, once we have built a model on sample data we collected, we might want to test it on a new sample, to have a more realistic expectation of how well the trained ML algorithm will perform on new data.

Sometimes you want to give to give up a little bit of accuracy in the sample you have to be able to get reasonable accuracy on new datasets, so that when the noise is a little bit different, the algorithm will be robust.

```
library(kernlab); data(spam); set.seed(333)
smallSpam <- spam[sample(dim(spam)[1],size=10),]
spamLabel <- (smallSpam$type=="spam")*1 + 1
plot(smallSpam$capitalAve,col=spamLabel)
```

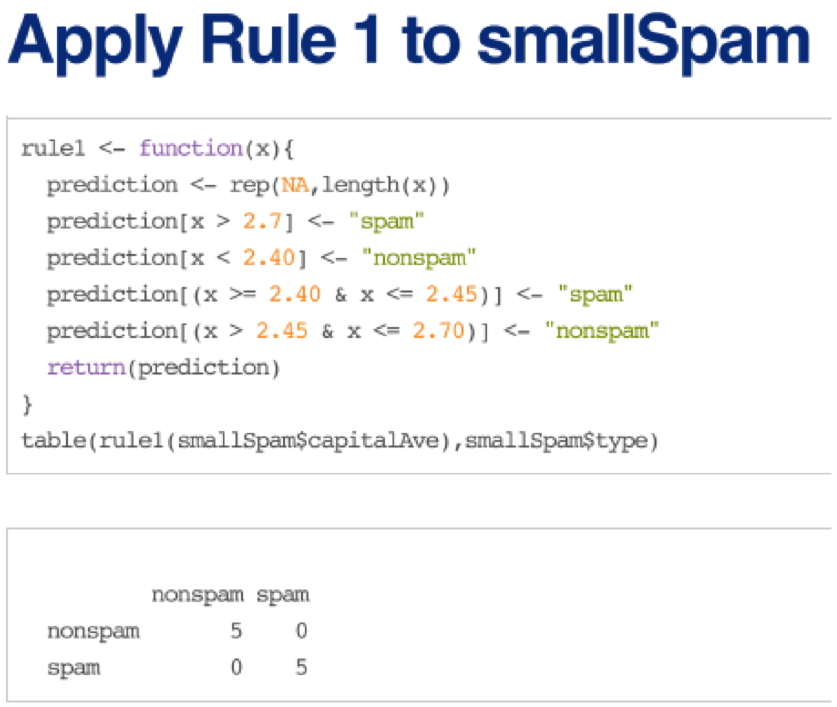


In this example, we checked the number of capital letters in 10 emails (spam vs ham) from the SPAM dataset. We want to build a predictor that classifies the emails as spam/non spam on the basis of the number of capital letters they contain (e.g. a spam email will contain a higher number of capital letters), e.g. using a rule such as:



- capitalAve > 2.7 = "spam"
- capitalAve < 2.40 = "nonspam"
- capitalAve between 2.40 and 2.45 = "spam"
- capitalAve between 2.45 and 2.7 = "nonspam"

The last two rules is to make the prediction perfect in the training set.

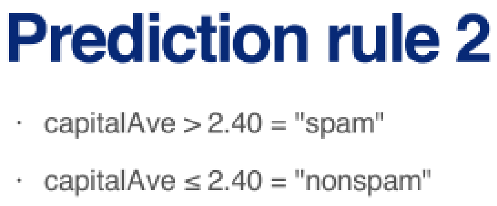


Apply Rule 1 to smallSpam

```r
rule1 <- function(x){
  prediction <- rep(NA,length(x))
  prediction[x > 2.7] <- "spam"
  prediction[x < 2.40] <- "nonspam"
  prediction[(x >= 2.40 & x <= 2.45)] <- "spam"
  prediction[(x > 2.45 & x <= 2.70)] <- "nonspam"
  return(prediction)
}
table(rule1(smallSpam$capitalAve),smallSpam$type)
```

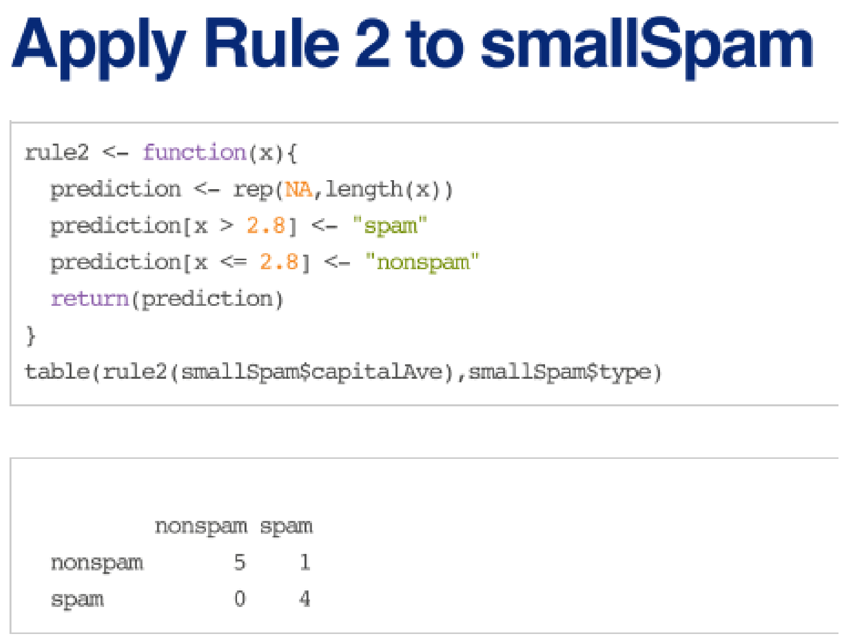
```
          nonspam spam
  nonspam       5    0
  spam          0    5
```

An alternative rule will not be tidied tightly to the training set and a slightly smaller accuracy score:



Prediction rule 2

- capitalAve > 2.40 = "spam"
- capitalAve ≤ 2.40 = "nonspam"

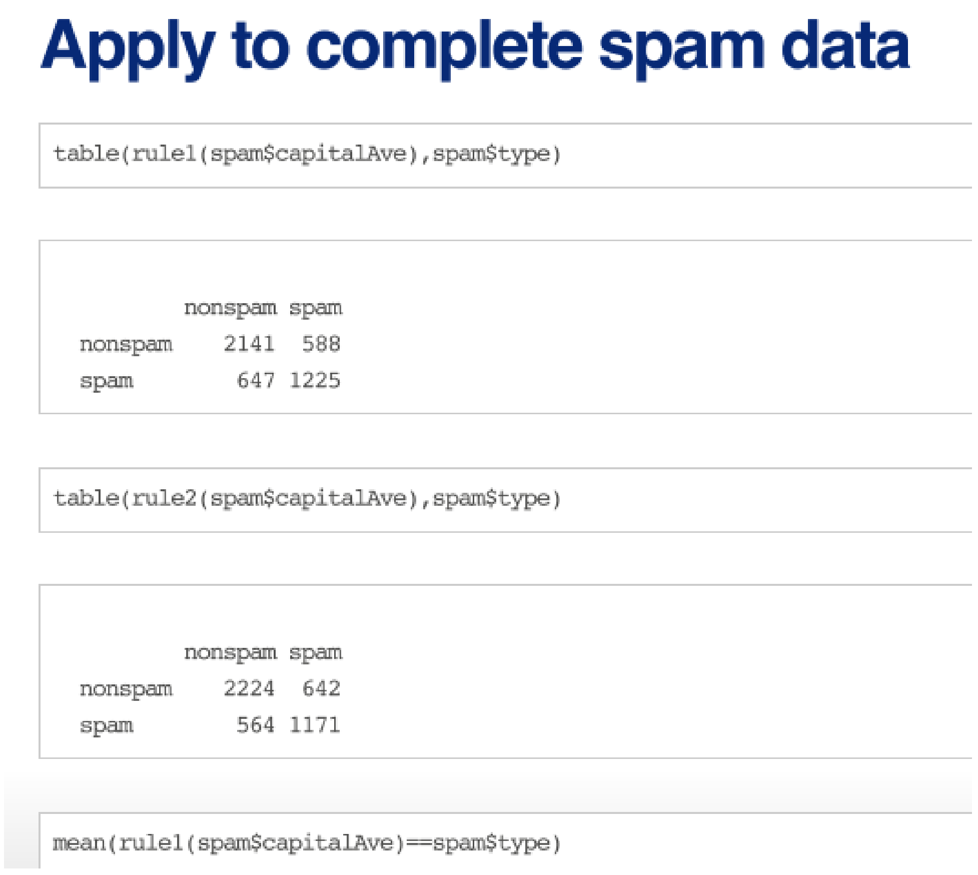


Apply Rule 2 to smallSpam

```r
rule2 <- function(x){
  prediction <- rep(NA,length(x))
  prediction[x > 2.8] <- "spam"
  prediction[x <= 2.8] <- "nonspam"
  return(prediction)
}
table(rule2(smallSpam$capitalAve),smallSpam$type)
```

```
          nonspam spam
  nonspam       5    1
  spam          0    4
```

We then try to apply Rule 1 and Rule 2 to the whole SPAM dataset:



Apply to complete spam data

```r
table(rule1(spam$capitalAve),spam$type)
```

```
          nonspam spam
  nonspam    2141  588
  spam        647 1225
```

```r
table(rule2(spam$capitalAve),spam$type)
```

```
          nonspam spam
  nonspam    2224  642
  spam        564 1171
```

```r
mean(rule1(spam$capitalAve)==spam$type)
```

Look at accuracy

```
sum(rule1(spam$capitalAve)==spam$type)
```

```
[1] 3366
```

```
sum(rule2(spam$capitalAve)==spam$type)
```

```
[1] 3395
```

What is going on? Overfitting

The data has 2 parts:

- **Signal**
- **Noise**

The goal of a predictor is to find signal (and ignore the noise). You can always design a perfect in-sample predictor (especially in a small dataset), but you capture both signal + noise when you do that --> the predictor won't perform as well on new samples!

Week 1.5: Prediction study design

We talk how to minimise the problems that can be caused by in sample vs out of sample errors.

1) Define your **error rate**
2) **Split data** into: **training** (to build your model), **testing** (to evaluate your model), **validation** (optional further model evaluation)
3) On the **training set** pick **features** (use **cross-validation**)
4) On the **training set** pick **prediction function** (use **cross-validation**)
5) If **no validation**: apply (the best model we have) **1x** to **test set**. Only one time because if we applied multiple models to our testing set and then pick the best one we are in some sense use the testing set to train the model as well and the estimation of the error rate would be optimistic.
6) If **validation**: apply to **test set** and **refine** --> apply **1x** to **validation**. In this case, you might apply your best prediction models all to your test set and refine them a little bit. In order to avoid the error rate to be too optimistic, we apply the best model to our validation set.

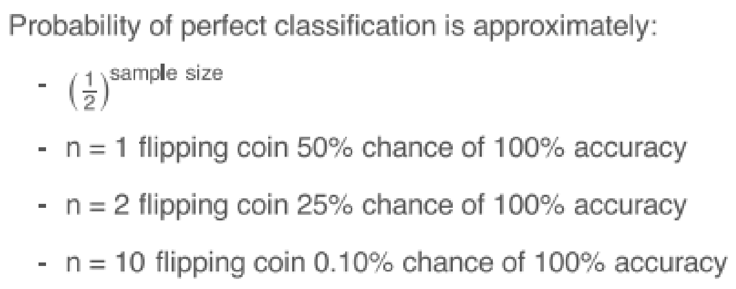
The idea is that there is **one dataset that is held out from the very start**, that you apply exactly only one model to, and you never do any training/tuning/testing to, and that will give you a good estimate of your out of sample error rates.

If you are building prediction models, you always have to hold one dataset out and leave it completely aside.

Avoid small sample sizes

When you are splitting your datasets up into training, testing and validation sets, they can get a bit small. You need to avoid small sample sizes, especially small test sample size.

Suppose you were predicting a binary outcome (e.g. diseased vs healty people). One possible classifier is "flipping a coin" to predict. With this absurd classifier, half of the time you will be right in your classification, just by chance:



Probability of perfect classification is approximately:

- $\left(\frac{1}{2}\right)^{\text{sample size}}$
- n = 1 flipping coin 50% chance of 100% accuracy
- n = 2 flipping coin 25% chance of 100% accuracy
- n = 10 flipping coin 0.10% chance of 100% accuracy

Consider 'n' be the test sample size. you can see above, the higher the test sample size, the less is probable to get 100% accuracy just by chance.

Rule of thumb for prediction study design

- If you have a **large sample size**: 60% training, 20% test, 20% validation
- If you have a **medium sample size**: 60% training, 40% testing
- If you have a **small sample size**: do **cross validation**; report **caveat of small sample size**.

Week 1.6: Types of errors

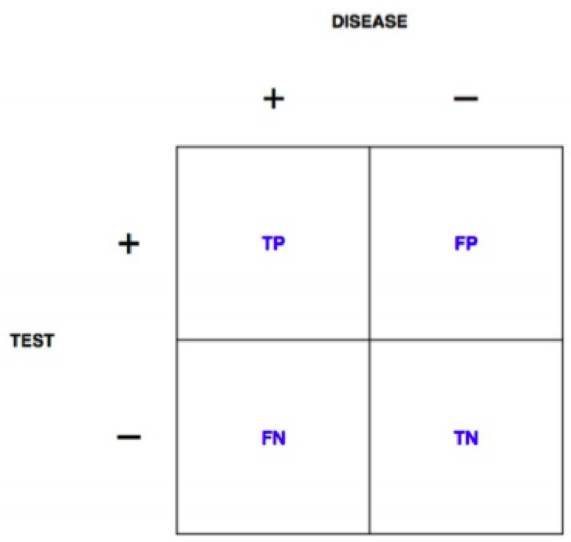
This lecture is about the types of errors and the way you will evaluate predictions functions.

Basic terms

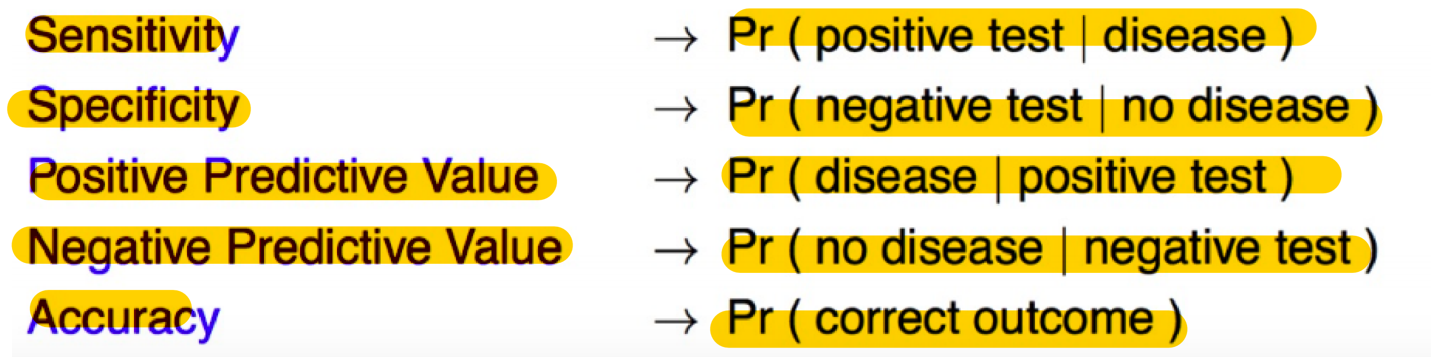
First we will concentrate on the types of errors you can encounter when you are working on a binary prediction problem.
In general, **Positive = identified**, **Negative = rejected**. Therefore:

True positive = correctly identified (e.g. sick people correctly diagnosed as sick).
False positive = incorrectly identified (healthy people incorrectly identified as sick).
True negative = correctly rejected (healthy people correctly identified as healthy).
False negative = incorrectly rejected (sick people incrrectly identified as healthy).

Key quantities.

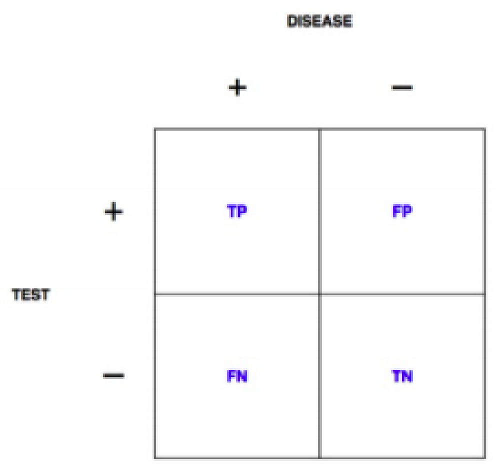


		DISEASE +	DISEASE −
TEST	+	TP	FP
TEST	−	FN	TN



Sensitivity	$\rightarrow$ Pr (positive test | disease)
Specificity	$\rightarrow$ Pr (negative test | no disease)
Positive Predictive Value	$\rightarrow$ Pr (disease | positive test)
Negative Predictive Value	$\rightarrow$ Pr (no disease | negative test)
Accuracy	$\rightarrow$ Pr (correct outcome)

Key quantities as fractions



		DISEASE +	DISEASE −
TEST	+	TP	FP
TEST	−	FN	TN

Sensitivity	$\rightarrow$ TP / (TP+FN)
Specificity	$\rightarrow$ TN / (FP+TN)
Positive Predictive Value	$\rightarrow$ TP / (TP+FP)
Negative Predictive Value	$\rightarrow$ TN / (FN+TN)
Accuracy	$\rightarrow$ (TP+TN) / (TP+FP+FN+TN)

Screening tests

Assume that some disease has a 0.1% prevalence in the population. Assume we have a test kit for that disease that works with 99% sensitivity and 99% specificity. What is the probability of a person having the disease giving the test result is positive, if we randomly select a subject from:

- (A) the general population?
- (B) a high risk sub-population with 10% disease prevalence?

p(T+|D+) = 0.99
p(T-|D-) = 0.99 --> p(T+|D-) = 1 - 0.99 = 0.01

Scenario (A)
p(D+|T+)? This is the positive predicted value
prevalence = p(D+) = 0.001
p(D-) = 0.999
--> use Bayes' theorem -->

$$p(D+|T+) = \frac{p(T+|D+) * p(D+)}{[p(T+|D+) * p(D+)] + [(p(T+|D-) * p(D-)]}$$

$$p(D+|T+) = \frac{0.99 * 0.001}{0.99 * 0.001 + 0.01 * 0.999} = \frac{0.00099}{0.00099 + 0.00999} = 0.0901$$

Scenario (B)

p(D+|T+)?

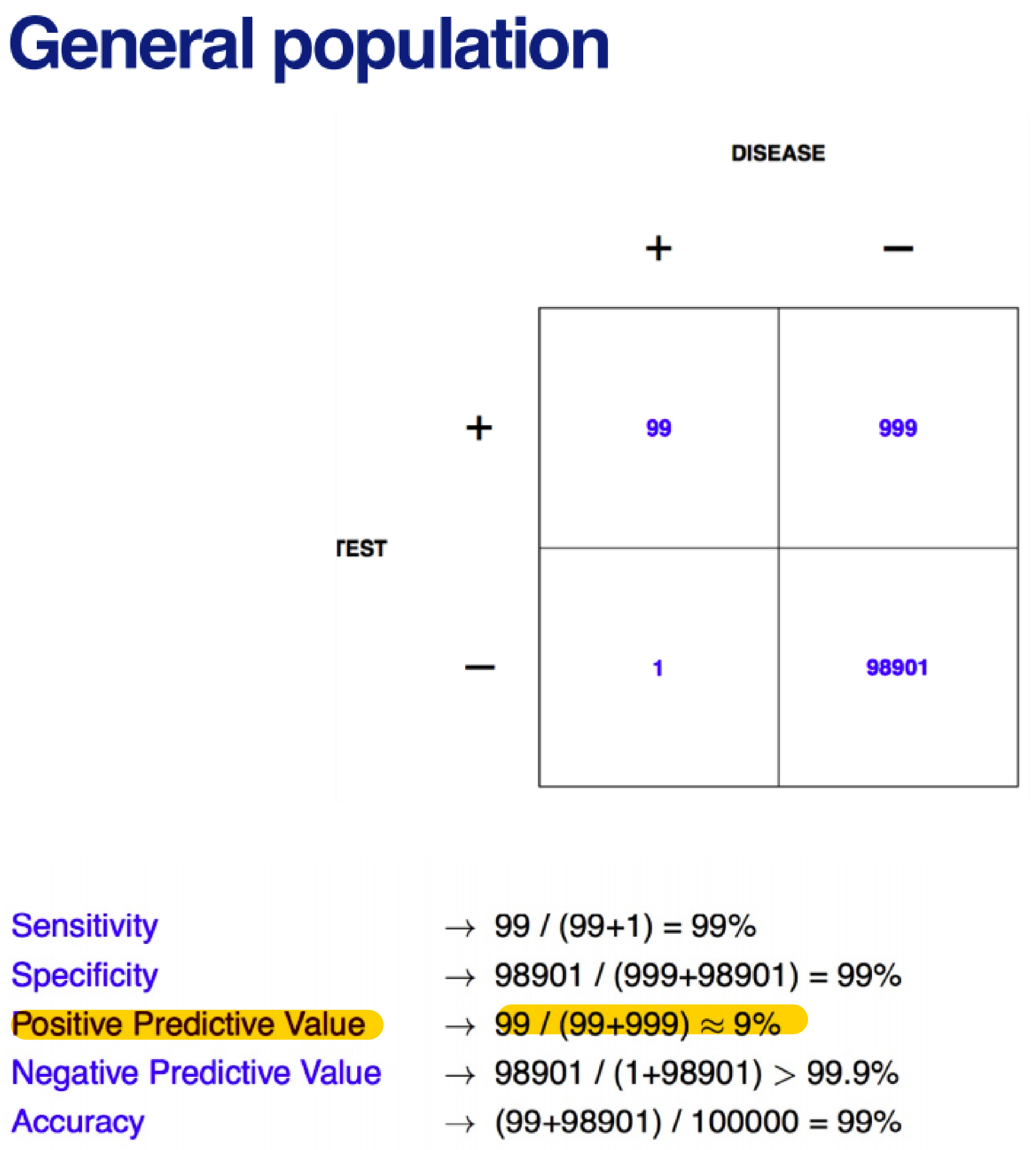
prevalence = p(D+) = 0.1

p(D-) = 0.9

--> use Bayes' theorem -->

$$p(D+|T+) = \frac{p(T+|D+) * p(D+)}{[p(T+|D+) * p(D+)] + [(p(T+|D-) * p(D-)]}$$

$$p(D+|T+) = \frac{0.99 * 0.1}{0.99 * 0.1 + 0.01 * 0.9} = \frac{0.099}{0.099 + 0.009} = 0.917$$

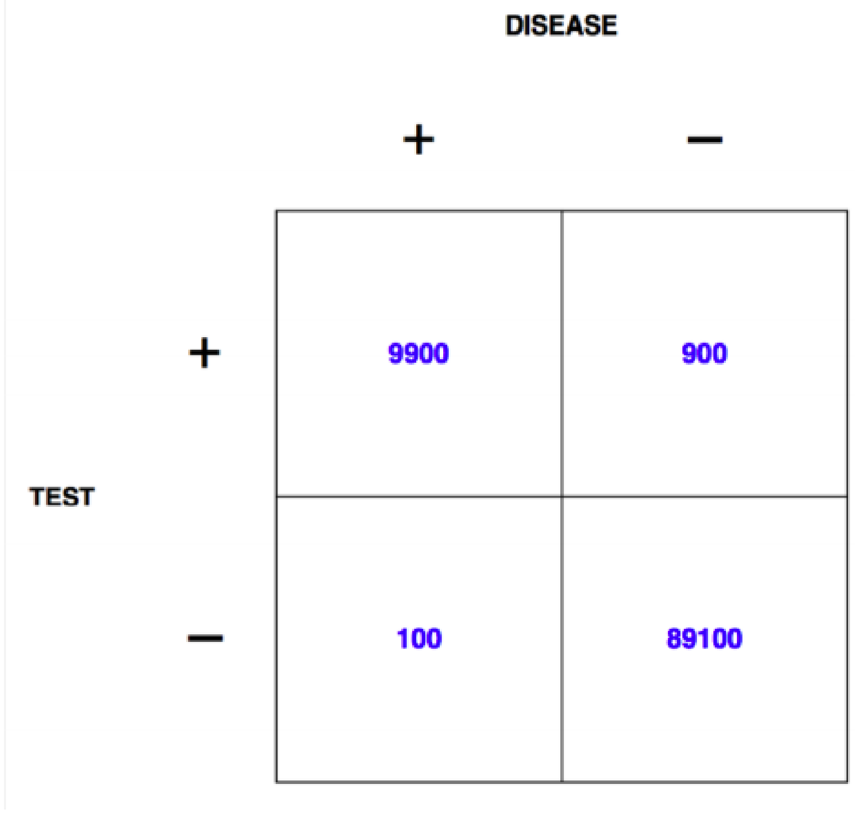


General population

		DISEASE +	DISEASE −
TEST	+	99	999
	−	1	98901

Sensitivity	$\rightarrow$ 99 / (99+1) = 99%
Specificity	$\rightarrow$ 98901 / (999+98901) = 99%
Positive Predictive Value	$\rightarrow$ 99 / (99+999) $\approx$ 9%
Negative Predictive Value	$\rightarrow$ 98901 / (1+98901) $>$ 99.9%
Accuracy	$\rightarrow$ (99+98901) / 100000 = 99%

At risk subpopulation



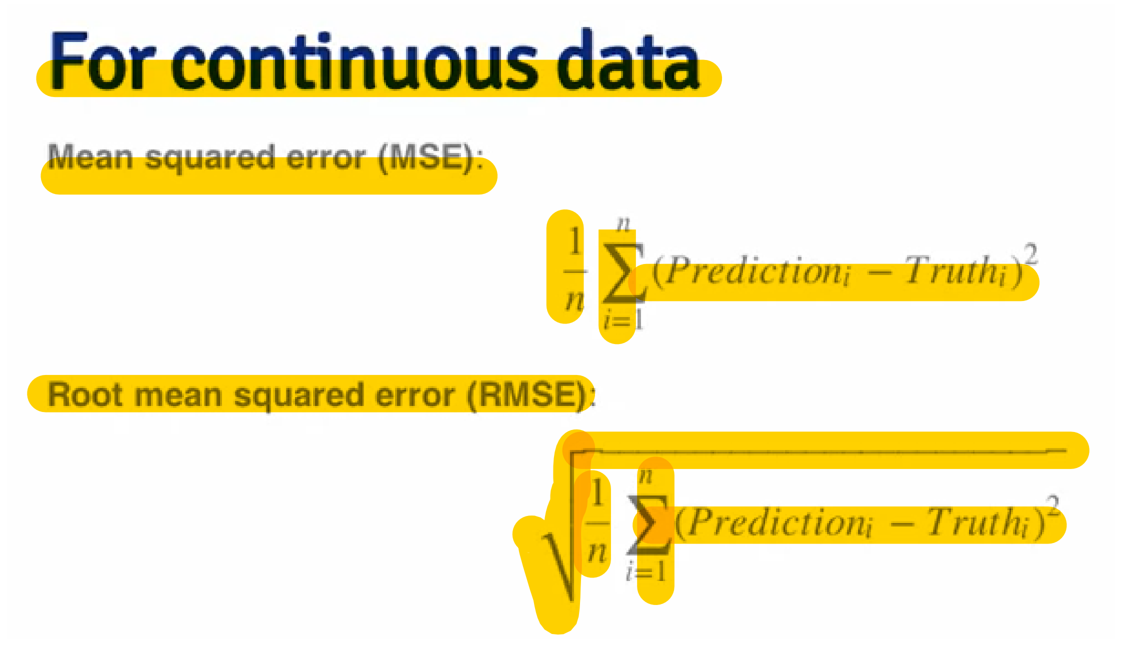
		DISEASE +	DISEASE −
TEST	+	9900	900
	−	100	89100

Sensitivity	$\rightarrow$ 9900 / (9900+100) = 99%
Specificity	$\rightarrow$ 89100 / (900+89100) = 99%
Positive Predictive Value	$\rightarrow$ 9900 / (9900+900) $\approx$ 92%
Negative Predictive Value	$\rightarrow$ 89100 / (100+89100) $\approx$ 99.9%
Accuracy	$\rightarrow$ (9900+89100) / 100000 = 99%

The take home message is that you are trying to predict a rare event you need to know how rare the event is. This goes back to the idea of knowing what population you are sampling from when you are building a predictive model.

This is actually a key public health issue: e.g. what is the value of mammograms or prostate cancer screening test as they try to detect a **fairly rare disease** and, even though the screening mechanisms are relatively good, it is very hard to know whether you are getting **many false positives** as a fraction of the total number of positives the test detects.

For continuous data you actually don't have a scenario as simple as for binary data. The goal here is to see how close you are to the truth.



For continuous data

Mean squared error (MSE):

$$\frac{1}{n}\sum_{i=1}^{n}(Prediction_i - Truth_i)^2$$

Root mean squared error (RMSE):

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(Prediction_i - Truth_i)^2}$$

With the MSE, you have a prediction for each sample you want to predict and you also know the truth of the prediction in the test set. It is a bit difficult to interpret, being a squared number, thus many people use its square, the RMSE.

Continuous error measures

1) **Mean Square Error** (**MSE**) or **Root Mean Square Error** (**RMSE**): used for continuous data, sensitive to outliers.
2) **Median absolute deviation**: used for continous data, often more robust to the presence of outliers.
3) **Sensitivity** (or **recall**): if you want few missed positives.
4) **Specificity**: if you want few negatives to be called positives.
5) **Accuracy**: it weights false positives and false negatives equally.
6) **Concordance**: one example is **kappa** (an example of distance measure). Used for multiclass data.

Week 1.7: Receiver Operating Characteristics (ROC) Curves

These are very commonly used techniques to measure the **quality** or the **goodness** of a **prediction algorithm**.

Why a curve?

In binary classification you are predicting one of two categories, e.g.:

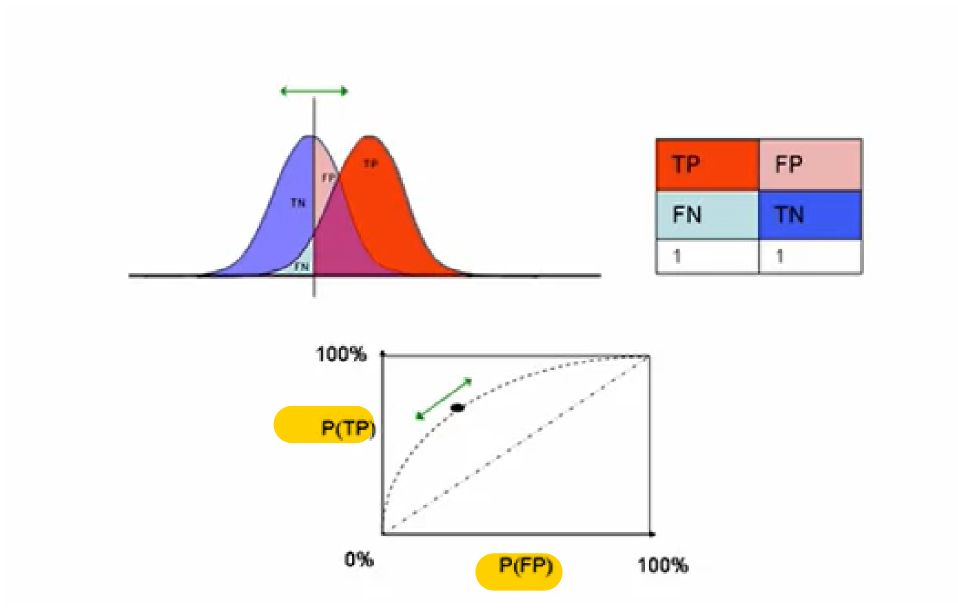
- alive/dead.
- click/don't click on advert.

However, your predictions are often quantitative:

- Probability of being alive
- Prediction on a scale from 1 to 10

The **cut-off** you choose gives different results (e.g. using a cut-off of probability of either 0.7 or 0.8 for predicting being alive, different cut-offs will be better at predicting people who are alive or who are dead).

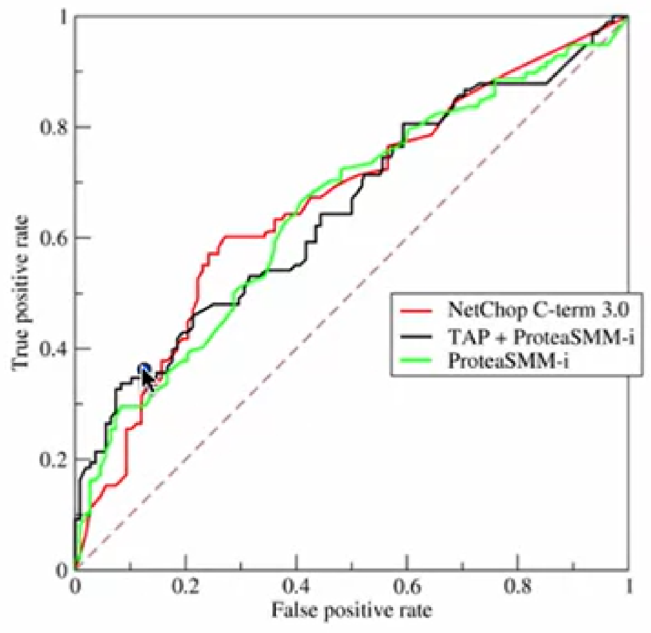
In order to choose the cut-off, ROC curves are used.



- On the **x-axis** it is plotted **1-specificity** = **prob**(**False Positive**) (**FP**)
- On the **y-axis** it is plotted **sensitivity** = **prob**(**True Positive**) (**TP**)

A ROC curve is produced where every single point along the curve corresponds to exactly one cut-off. An ROC curve can be used to define wether an algorithm it is good or bad by plotting a different point for every single cut-off you may choose and then plotting a curve through those points.

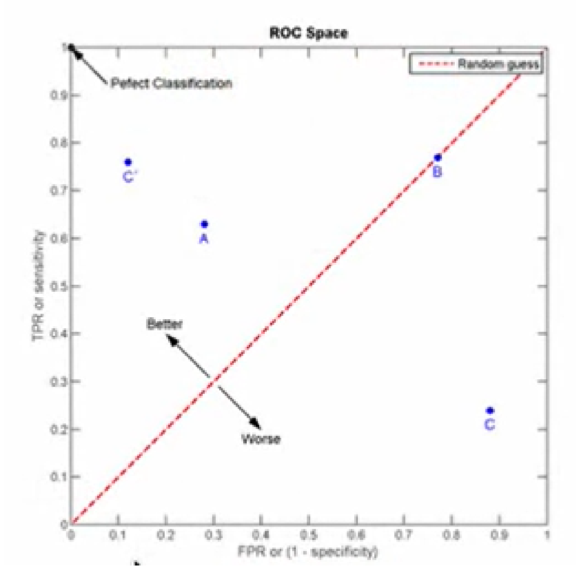
An example



To establish which prediction algorithm is better at the task you are testing them on, many people quantify the **area under the curve (AUC)** for each algorithm. Each area quantifies how good each prediction algorithm is: **the higher the area, the better the prediction algorithm is**.

There are some standard values that makes sense to pay attention to:

- **AUC = 0.5**: **random guessing** (prediction algorithm that is on the diagonal line, where sensitivity = specificity = 0.5). If AUC < 0.5, the predictor is worse than random guessing.
- **AUC = 1**: **perfect classifier** (perfect sensitivity and specificity)
- In general, **AUC > 0.8** is considered **"good"**



Week 1.8: Cross-validation

Cross-validation is one of the most widely used tools in ML for detecting relevant features and for building models and estimating their parameters.

Key idea

- Accuracy on the training set (**resubstitution accuracy**) is **optimistic**
- A better estimate comes from an **independent set** (**test set accuracy**)
- But we can't use the test set when building the model or it becomes part of the training set (if we keep on using always the same test set to use the model's accuracy, it sort of becomes part of the training set itself)
- So we estimate the test set accuracy with the training set

Cross-validation

Approach:
1) Use the training set
2) Split it into training/test sets
3) Build a model on the training set

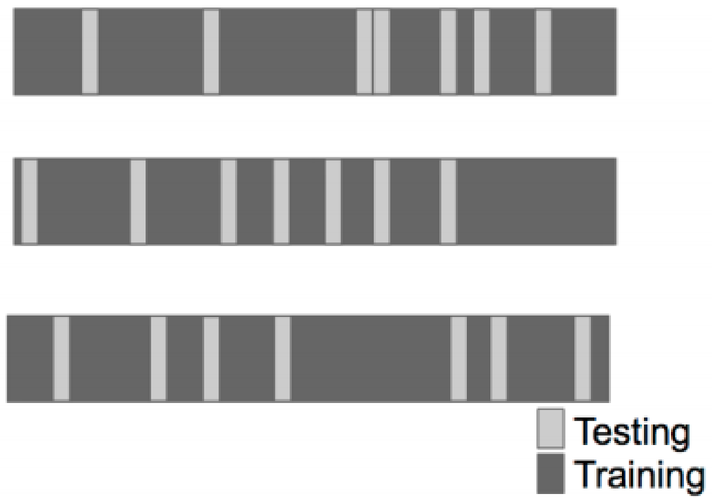
4) Evaluate the model on the test set
5) Repeat (from (1)) and average the estimated errors

Used for:

- Picking **variables** to include in the model
- Picking the **type of prediction function** to use
- Picking the **parameters** in the prediction function
- Comparing the **different predictors**

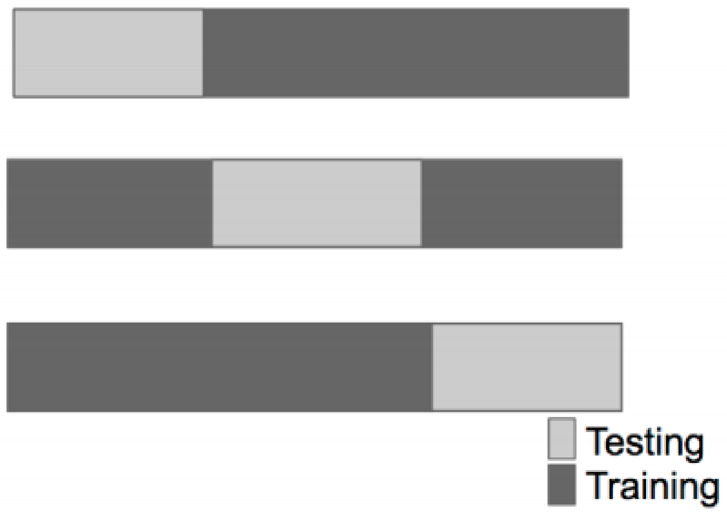
Random subsampling

The light grey samples used for testing are chosen randomly:



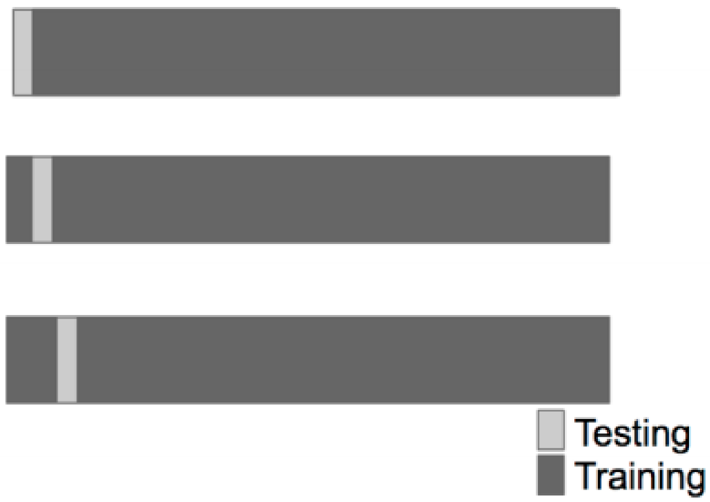
K-fold

The data set is broken into K- equal size datasets.
An example for 3-fold cross-validation:



Leave one out cross validation

We leave out exactly one sample and build the model on all the remaining samples and predict the only sample that was left out.



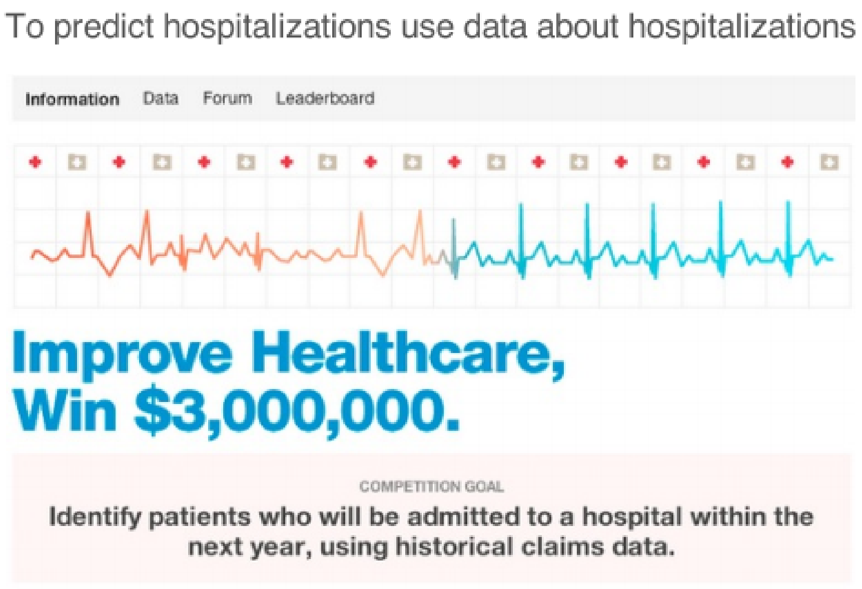
Considerations

- For **time series data**, the data must be in "**chunks**" (i.e. **blocks of times that are all contiguous** since one time point might depend on all the time points that came previously and you are ignoring a huge rich structure if you take just random samples)
- For **k-fold** cross validation:
  - **Larger k** = **less bias, more variance**
  - **Smaller k** = **more bias, less variance**
- **Random sampling** must be done **without replacement**
- Random sampling **with replacement** is the **bootstrap**:
  - Underestimation of the error (because some samples will appear both in the training and test set)
  - It can be corrected but it is complicated (0.632 bootstrap)
- If you **cross-validate** to **pick predictors** estimate, you must **estimate errors on independent data**. In other words, if you do cross validation to predict your model or to pick your model, the cross-validated error rates will not always be a good representation of what the real out-of-sample error rate is (because you always pick the best model). The best way to deal with this is applying your prediction function just one time to an independent test set.

Week 1.9: What data should you use?

After defining your question of interest, the next most important thing is to identify the dataset that will allow you to create the best possible predictions with you ML algorithms.

Key idea: **to predict X use data related to X** ("like predicting like").



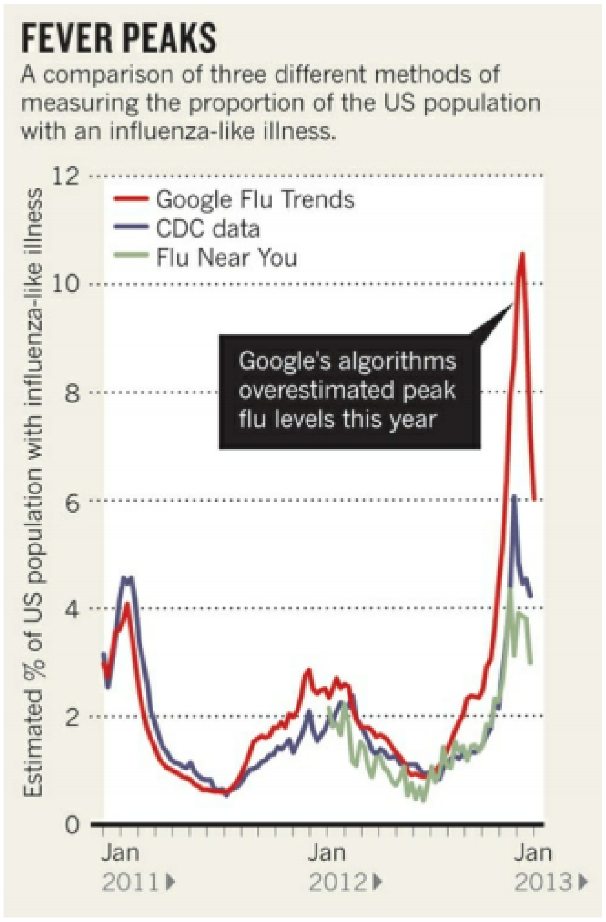
To predict hospitalizations use data about hospitalizations

Improve Healthcare, Win $3,000,000.

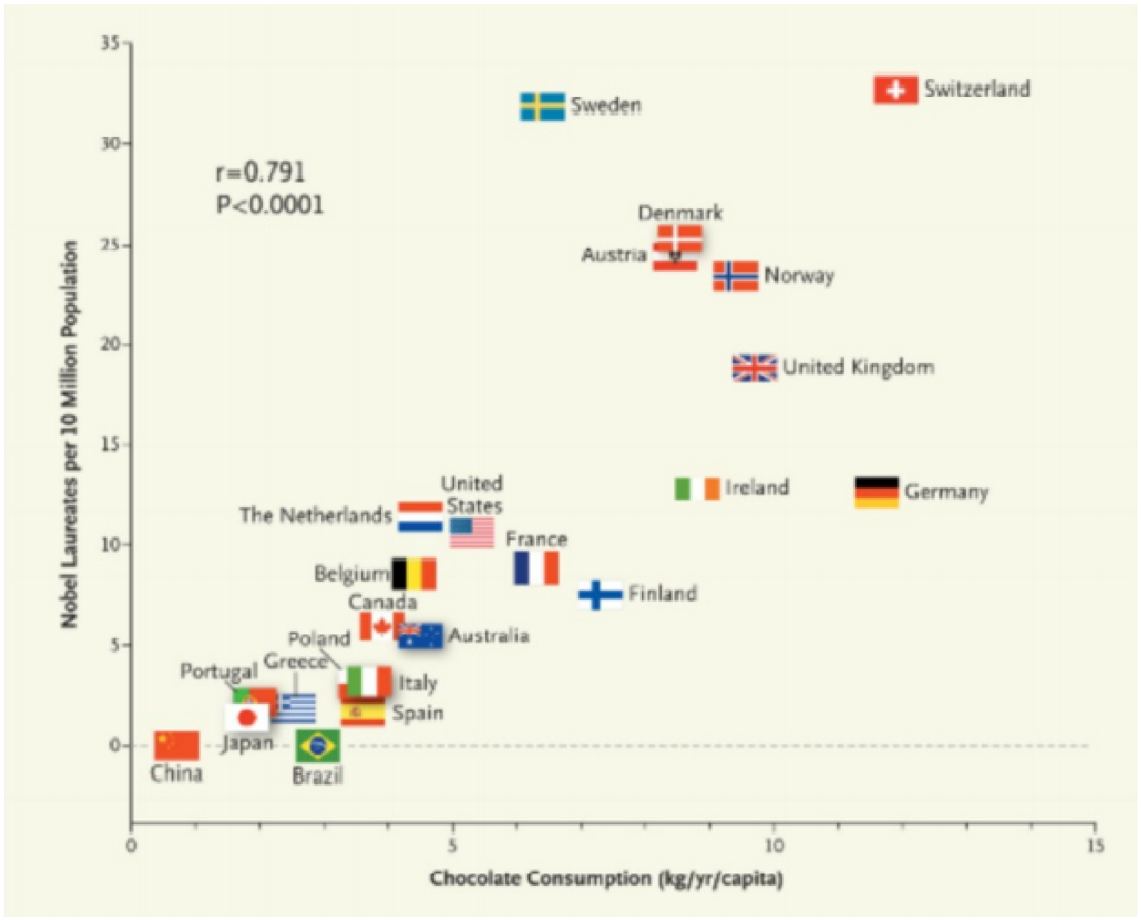
COMPETITION GOAL

Identify patients who will be admitted to a hospital within the next year, using historical claims data.

- The **looser the connection**, the **harder the prediction**.
- The data properties matter (e.g. using flu symptoms datasets to predict flu outbreaks is better than using the number of times people have Googled key-words related to flu: the Google flu algorithms had overestimation problems):



- **Unrelated data** is the most common mistake (e.g. correlation between chocolate consumption in kg per year per capita versus number of Nobel laureates per 10 million people in the population). If you have built your predictive model on these unrelated characteristics, once your population characteristics change your model will not work very well anymore...



Week 1: Quiz

1) Which of the following are components in building a ML algorithm?

- Statistical inference
- Training and test sets
- **Deciding on an algorithm**
- Machine learning
- Artificial intelligence

2) Suppose we build a prediction algorithm on a data set and it is 100% accurate on that data set. Why might the algorithm not work well if we collect a new dataset?

- We are not asking a relevant question that can be answered with machine learning
- **Our algorithm may be overfitting the training data, predicting both the signal and the noise**
- We may be using bad variables that don't explain the outcome
- We have used neural networks which has notoriously bad performance

3) What are typical sizes for the training and testing sets?

- 90% training set, 10% test set
- 20% training set, 80% test set
- **60% training set, 40% training set**
- 100% training set, 0% test set

4) What are some common error rates for predicting binary variables (i.e. variables with 2 possible values: yes/no, disease/healthy):

- Median absolute deviation
- Root mean squared error
- **Accuracy**
- R^2
- Correlation

5) Suppose that we have created a machine learning algorithm that predicts whether a link will be clicked with 99% sensitivity and 99% specificity. The rate the link is clicked is 1/1000 of visits to a website. If we predict the link will be clicked on a specific visit, what is the probability it will actually be clicked?

- 99.9%
- **9%** (see below for answer explanation)
- 99%
- 50%

sensitivity = p(test+|clicked+) = 0.99
specificity = p(test-|clicked-) = 0.99 --> p(test-|clicked-) = 1 - 0.99 = 0.01

p(clicked+|test+)?
prevalence = p(clicked+) = 0.001
p(clicked-) = 0.999
--> use Bayes' theorem -->

$$p(clicked+|test+) = \frac{p(test+|clicked+) * p(clicked+)}{[p(test+|clicked+) * p(clicked+)] + [(p(test+|clicked-) * p(clicked-)}$$

$$p(clicked+|test+) = \frac{0.99 * 0.001}{0.99 * 0.001 + 0.01 * 0.999} = \frac{0.00099}{0.00099 + 0.00999} = 0.0901$$