https://www.cs.ubc.ca/~fwood/CS340/

Lecture IV

January 13th, 2020
https://www.cs.ubc.ca/~fwood/CS340/lectures/L4.pdf

Supervised Learning: Determine Home city

- We are given data from 248 homes
- For each home/example we have these features:
  - Elevation
  - Year
  - Bathrooms
  - Bedrooms
  - Price
  - Square feet
- Goal is to build a program that predicts **SF** of **NY**

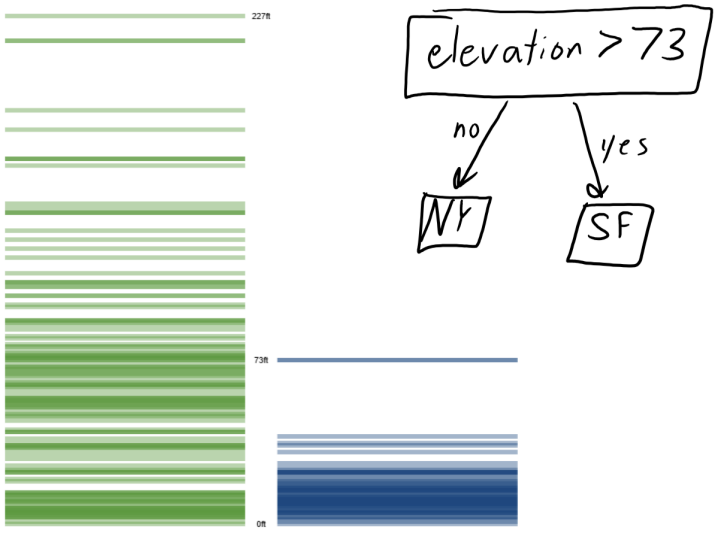


Figure 1: Plotting Elevation, and consequent simple decision stump

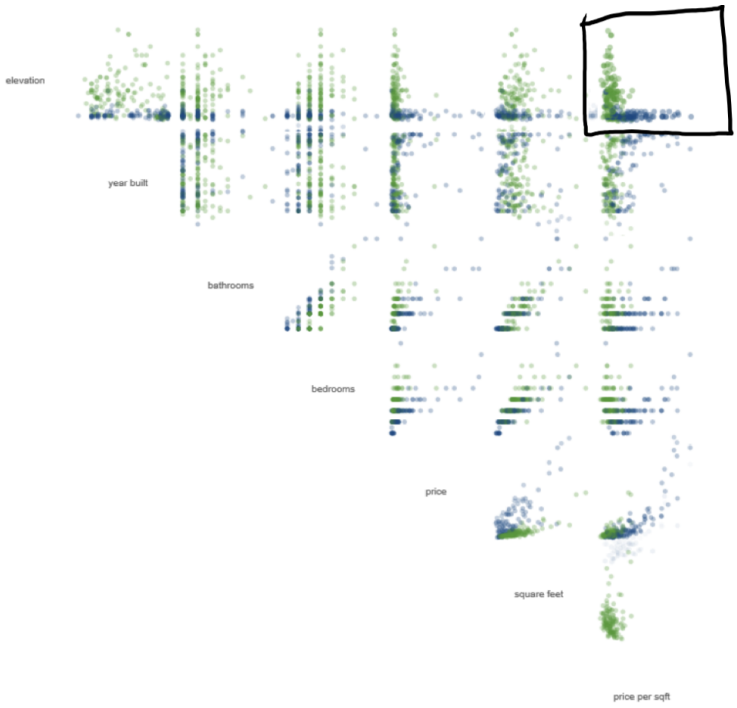


Figure 2: Scatterplot array, and best visible correlation

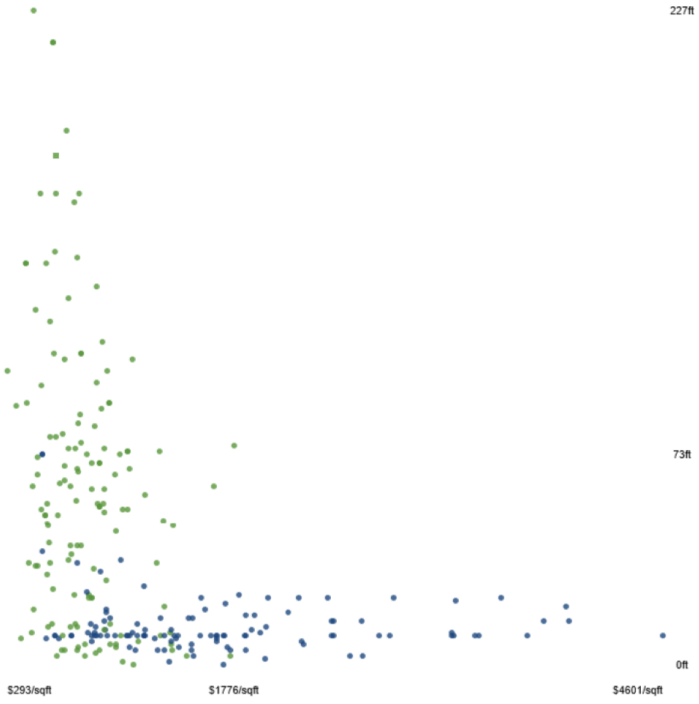


Figure 3: Plotting elevation vs. Price/SqFt

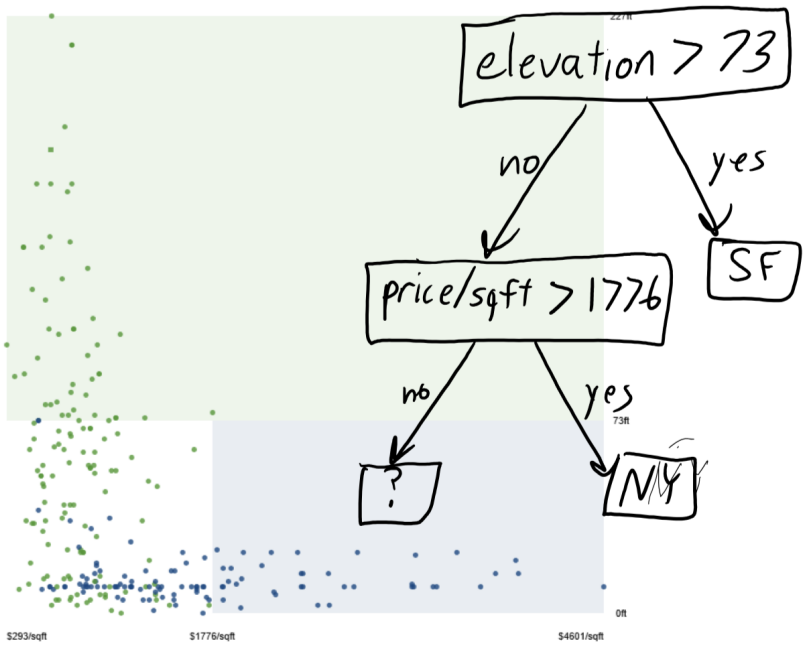


Figure 4: Simple decision tree classification

How does Depth affect accuracy?

We can add depth to the decision tree by splitting the data recursively. Accuracy keeps increasing as we add depth, and eventually, we can perfectly classify all of our data:

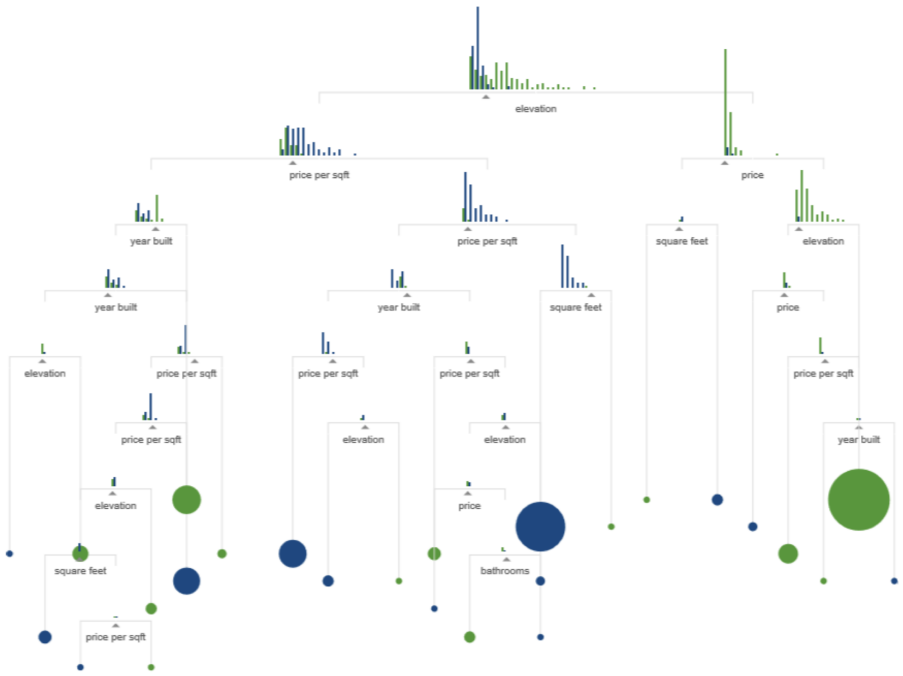


Figure 5: Deep decision tree with perfect accuracy

With this decision tree, the *training accuracy* is 1. It perfectly labels the data we used to make the tree. If we are now given 217 new homes, what is the 'testing accuracy' on the new data?

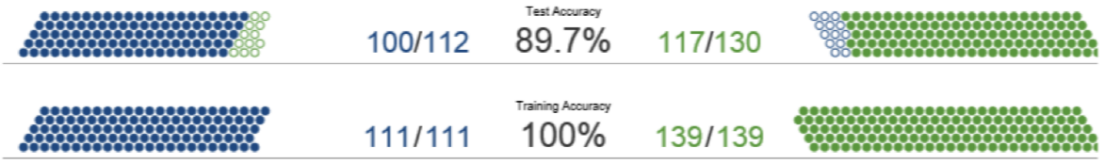


Figure 6: Testing accuracy on data not used to train

Supervised learning notation

- We are given **training data** of which we know the label:

$X =$

Egg	Milk	Fish	Wheat	Shellfish	Peanuts	...
0	0.7	0	0.3	0	0	
0.3	0.7	0	0.6	0	0.01	
0	0	0	0.8	0	0	
0.3	0.7	1.2	0	0.10	0.01	
0.3	0	1.2	0.3	0.10	0.01	

$y =$

Sick?
1
1
0
1
1

- We are given **testing data** of which we wish to know the label

$\tilde{X} =$

Egg	Milk	Fish	Wheat	Shellfish	Peanuts	...
0.5	0	1	0.6	2	1	
0	0.7	0	1	0	0	
3	1	0	0.5	0	0	

$\tilde{y} =$

Sick?
?
?
?

The typical steps are:

1. Build model based on training data X and y (training phase).
2. Model makes predictions $\hat{y}$ on test data $\tilde{X}$ (testing phase).

Instead of training error, we consider test error. Are the prediction labels $\hat{y}$ similar to the true labels $\tilde{y}$

The goal of Machine Learning

All we care about is the testing error!
Midterm analogy:

- The training error is the practice midterm
- The test error is the actual midterm
- Goal: do well on the actual midterm, not the practice one

Golden rule of Machine learning

Even though we care about the test data, ***THE TEST DATA CANNOT INFLUENCE THE TRAINING PHASE IN ANY WAY***
We're measuring test error to see how well we do on new data. If it is used in training, this isn't measured. You can start to overfit if you use it during training.

Digression: Golden rule and hypothesis testing

Note that the golden rule applies to hypothesis testing in scientific studies: the data you collect cannot influence the hypothesis that you test.
This is extremely common, and a major problem coming in many forms:

- Collect huge amounts of data until you coincidentally get the significance level you want
- Try different ways to measure performance, and use the one that *looks* best
- Choose a different type of model/hypothesis after looking at the data

In general: if you want to modify your hypotheses, you need to try on new data, or at least be aware and honest about the issue when reporting the findings.

Is learning possible?

In general, training error doesn't give us any information about testing error. The test data might have absolutely nothing to do with the training data. Thus, in order to learn, we need *assumptions*:

- The training and test data must be related in some way
- Most common assumption: ***Independent and Identically Distributed (IID)***

IID Assumption

Training/test data is *independent and identically distributed (IID)* if:

- All examples come from the same distribution (identically distributed)
- The examples are sampled independently (order doesn't matter)

Ex: IID in the food allergy example

Is the food allergy example IID:

- Do all the examples come from the same distribution?
- Does the order of the examples matter?

NO!

- Being sick might depend on what you ate yesterday! (Not independent)
- Your eating habits might've changed over time (Not identically distributed)

Learning Theory

The IID assumption makes learning possible because pattern in the training examples are likely to be similar to the ones in the testing examples. But the IID assumption is *rarely true*:

- It is a good approximation
- There are other possible assumptions

Also, we're assuming IID across examples, and not the features.
Learning Theory explores how training error is related to test error. We'll look at a few simple examples, using this notation:

- E_{train} is the error in the training data
- E_{test} is the error in the testing data

Fundamental Trade-Off

$$E_{\text{test}} = E_{\text{test}}$$
$$E_{\text{test}} = (E_{\text{test}} - E_{\text{train}}) + E_{\text{train}}$$
$$E_{\text{test}} = E_{\text{approx}} + E_{\text{train}}$$

where:

$$E_{\text{approx}} = E_{\text{test}} - E_{\text{train}}$$

If E_{approx} is small, the E_{train} is a good approximation to E_{test}, and E_{approx} is the "amount of over fitting":

- It gets smaller as n get larger
- It tends to grow, as the model gets more complicated

This leads to a **fundamental trade off**:

- E_{train}: How small can you make training error
- E_{approx}: How well the training error approximates the test error

In ***simple models*** for instance (like decision stumps):

- E_{approx} is low (since it is not very sensitive to the training set), but E_{train} might be high

However in ***complex models*** (like deep decision trees):

- E_{train} may be low, but E_{approx} may be high (since it is very sensitive to the training set)

In general:

- Training error is high for low depth (*underfitting*)
- Training error gets better with depth
- Test error initially goes down, but eventually increases (*overfitting*)

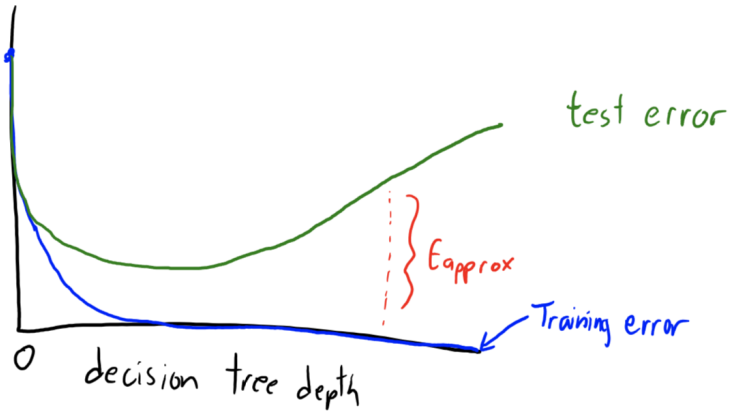


Figure 7: Effect of tree depth on approximation error

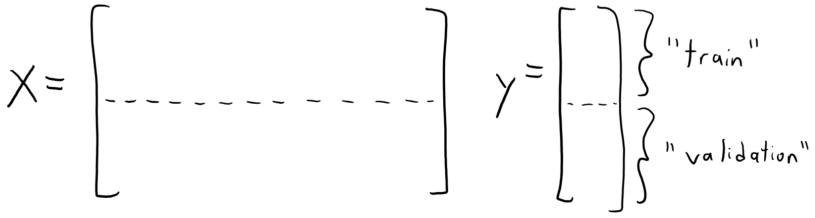
Validation Error

Questions at this point may be:

- How do we decide decision tree depth?
- We care about the test error, but we can't look at the test data!?
- So what do we do???

One answer: We can look at the testing data to approximate test error. This consists of splitting the training examples into a *training set* and a *validation set*:

- Train model based on training data
- Test model based on validation data



Step 1 is training: model = train(X_{train}, y_{train})
Step 2 is predicting: $\hat{y}$ = predict(model, $X_{validate}$)
Step 3 is validating: error = sum($\hat{y} \neq y_{validate}$)

For IID data: validation error is the unbiased approximation of test error:

$$\mathbb{E}(E_{\text{valid}}) = \mathbb{E}(E_{\text{test}})$$

Midterm analogy

- You have 2 practice midterms

- You hide one midterm, and spend a lot of time working through the other
- You then do the other practice term, to see how well you'll do on the test

The validation error is then chosen to choose the "*Hyper-parameters*"

Notation: Parameters and Hyper-parameters

1. The decision tree **rule** values are called the **parameters**. They control how well we fit the data set. We train the model to find the best parameters on training data.
2. The decision tree **depth** is called a **hyper-parameter**. It controls how *complex* our model is. We *can't train* a hyper parameter, however we *validate* a hyper parameter using a validation score

Choosing hyper-parameters with a validation set

To choose a good value of depth (hyper parameter), we could try decision trees of different depth (say 1 to 20), and compute the validation error for each. We return the depth with the lowest validation error.
After this hyper-parameter is chosen, we re-train on the whole training set, with the chosen hyper-parameter.

Optimization Bias

Another name for overfitting is **optimization bias**. How biased is the error that we optimized over many possibilities?

Optimization bias of parameter learning:

- During learning we can search over tons of decision trees.
- This means that we can get lucky and find one with a low training error by chance (overfitting of the training error)

Optimization bias of hyper-parameter tuning:

- Here, we might optimize the validation error over 20 values of "depth"
- One of the 20 trees might have low validation error by chance (Overfitting of the validation error).

Example of Optimization Bias

Consider a multiple choice (a,b,c,d) test with 10 questions.

- If you choose answers randomly, your expected grade is 25% (no bias).
- If you fill out two tests randomly and pick the best, expected grade is 33% (and there is an optimization bias of 8%).
- If you take the best among 10 tests, expected grade is ≈47%
- If you take the best among 100 tests, expected grade is ≈62%
- If you take the best among 1000 tests, expected grade is ≈73%
- If you take the best among 10000 tests, expected grade is ≈82%

Factors affecting optimization bias:
If we chose a 100 question test instead, then:

- Expected grade from best over 1 randomly-filled test is 25%.
- Expected grade from best over 2 randomly-filled test is ≈27%.

- Expected grade from best over 10 randomly-filled test is ≈32%.
- Expected grade from best over 100 randomly-filled test is ≈36%.
- Expected grade from best over 1000 randomly-filled test is ≈40%.
- Expected grade from best over 10000 randomly-filled test is ≈47%.

i.e.: The optimization bias grows with the number of things that we try, but the optimization bias shrinks drastically with the number of examples (But is still non-zero and growing, if validation set is overused).

Lecture V

January 17th, 2020
`https://www.cs.ubc.ca/~fwood/CS340/lectures/L5.pdf`

Overfitting to the Validation Set?

- Validation error usually has lower optimization bias than training error (Might optimize over 20 values of "depth", instead of millions+ of possible trees)
- But we can still overfit to the validation error (common in practice):
  - Validation error is only an unbiased approximation if you use it once.
  - Once you start optimizing it, you start to overfit to the validation set.
- This is most important when the validation set is small
  - The optimization bias decreases as the number of validation examples increases
- The goal is to do well on the testing data (new set), no the training one where we already know the labels.

Validation Error and Optimization Bias

- Optimization bias is small if you only compare a few models:
  - Best decision tree on the training set among depths 1, 2, 3,..., 10. – Risk of
  - overfitting to validation set is low if we try 10 things.
- Optimization bias is large if you only compare a lot of models:
  - All possible decision trees of depth 10 or less
  - Here we're using the validation set to pick between a billion+ models (Risk of overfitting to validation set is high: could have low validation error by chance)
  - If you did this, you might want a second validation set to detect overfitting
- Optimization bias shrinks as you grow size of the validation set

Cross-Validation (CV)

This is used to avoid "wastefulness" related to using only part of the data.
Example: 5-fold cross validation:

- Train on 80% of the data, validate on the other 20%
- Repeat this 5 more times with different splits, and average the score

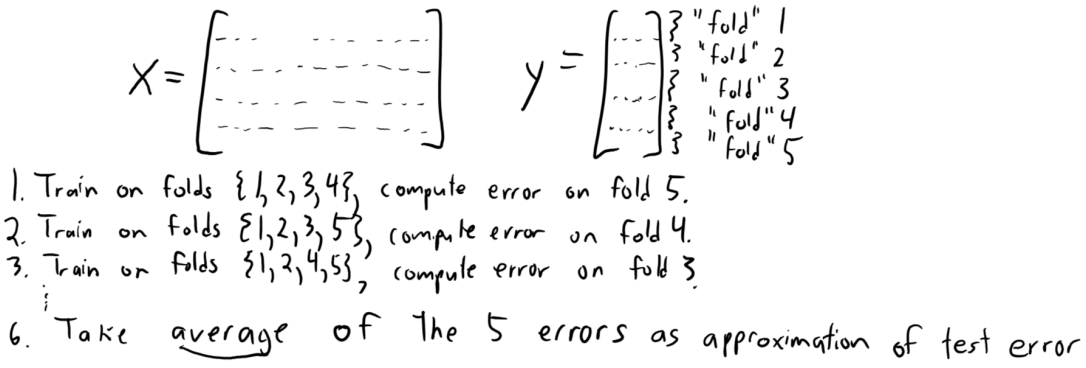


Figure 8: Cross-validation example

Cross-validation Pseudocode

To choose depth:

```
for depth in 1:20:
   compute cross validation score
return depth with highest score
```

To compute 5-fold cross validation score:

```
for fold in 1:5:
   train 80% that doesn't include fold
   test on fold
return average test error
```

You can also take this idea further: **k-fold cross validation**

- 10 fold cross validation: train on 90% of data, test on the remaining 10% (repeat 10 times, for each fold, and average)
- Leave one out cross-validation: train on all but one training example, repeat n times and average.

This gets more accurate but more expensive with each fold

The "Best" Machine Learning Model

As decision trees are not always the most accurate on the test error, we ask what is the best machine learning model

An alternative measure of performance is the generalization error

- Average error over all x_i vectors that are not seen in the training set
- "How well we expect to do for a completely unseen feature vector"

No free lunch theorem:

- There is no "best" model for achieving the best generalization error for every problem
- If model A generalizes better to new data than model B on one data set, there is another data set where model B works better.

This implies that we need to try out multiple models. In CPSC 340, we focus on models that have been effective in many applications.

Application: E-mail Spam Filtering

Obj: We want to build a system that detects spam emails. Can this be formulated in the form of supervised learning?

Procedure:

1. Collect a large number of emails, and get users to label them.
2. We can use $(y_i = 1)$ if email i is spam, and $(y_i = 0)$ otherwise
3. Extract features in each email (like bag of words)
    - $(x_{ij} = 1)$ if email i has word/phrase j, $(x_{ij} = 0)$ otherwise

Feature representation for Spam

Are there better features than bag of words?

- We add bigrams (sets of two words)
  - "CPSC 340", "wait list", "special deal".
- Or tri-grams (sets of three words)
  - "Limited time offer", "course registration deadline", "you're a winner".
- We might include the sender domain
  - <sender domain == "mail.com">
- We might include regular expressions
  - <your first and last name>

Probabilistic Classifiers

For years, the best spam filtering used naïve Baye's, which is a probabilistic classifier based on Baye's rule. It tends to work well with bag of words.

Probabilistic classifiers model the conditional probability, $P(y_i|x_i)$

- If a message has words x_i what is the probability that the message is spam?

Classify it as follows (classify it as spam if probability of spam is higher than not spam):

```
if (P(y[i] = "spam" | x[i]) > P(y[i] = "not spam" | x[i]))
    return "spam"
else:
    return "not spam"
```

To model conditional probability, Naïve Baye's uses Baye's rule:

$$P(y_i = \text{"spam"}|x_i) = \frac{P(x_i|y_i = \text{"spam"})P(\text{"spam"})}{P(x_i)}$$

So we need to figure out 3 types of terms:

1. Marginal probability $P(y_i)$ that and e-mail is spam
2. Marginal probability $P(x_i)$ that an e-mail has the set of words x_i
3. The conditional probability $P(x_i|y_i)$ that a spam email has the words x_i

Definitions of the terms:

- $P(y_i = \text{"spam"})$ is the probability that a random email is spam. This is very easy to approximate from the data (use proportion of emails that are spam):

$$P(y_i = \text{"spam"}) = \frac{\#\text{ of spam messages}}{\#\text{ of total messages}}$$

- $P(x_i)$ is the probability that a random email has features x_i. This is hard to approximate: for 'd' words we need to collect 2^d "coupons":

$$P(x_i) = \frac{\#\text{ e-mails with features } x_i}{\#\text{ of total messages}}$$

However, we can ignore it, since naïve Baye's returns "spam" if:

$$P(y_i = \text{"spam"}|x_i) > P(y_i = \text{"not spam"}|x_i)$$

This means that:

$$\frac{P(x_i|y_i = \text{"spam"})P(\text{"spam"})}{P(x_i)} > \frac{P(x_i|y_i = \text{"not spam"})P(\text{"not spam"})}{P(x_i)}$$

Multiplying by P(x˙i) on both sides yields:

$$P(x_i|y_i = \text{"spam"})P(\text{"spam"}) > P(x_i|y_i = \text{"not spam"})P(\text{"not spam"})$$

- $P(x_i|y_i = \text{"spam"})$ is the probability that spam has features x_i. This is also hard to approximate.

$$P(x_i|y_i = \text{"spam"}) = \frac{\#\text{ of spam messages with features } x_i}{\#\text{ of spam messages}}$$

Naïve Baye's

Naïve Baye's makes a big assumption to make things easier. Namely:

$$P(\text{hello} = 1, \text{vicodin} = 0, 340 = 1|\text{spam}) \approx P(\text{hello} = 1|\text{spam})P(\text{vicodin} = 0|\text{spam})P(340 = 1|\text{spam})$$

This means that we assume that all features x_i are independent given label y_i. Here:

- Once you know it's spam, the probability of "vicodin" doesn't depend on "340".
- This is definitely not true, but a good approximation.

Now only simple quantities are needed, like $P(\text{"vicodin"} = 0|y_i\text{"spam"})$.:

$$P(\text{"vicodin"} = 0|y_i\text{"spam"}) = \frac{\#\text{ of messages with "vicodin"}}{\text{total }\#\text{ of messages}}$$