* 1. **Feature Selection**

In Feature Selection, we aim to identify a subset of features that are responsible for the trends we observe in the data.

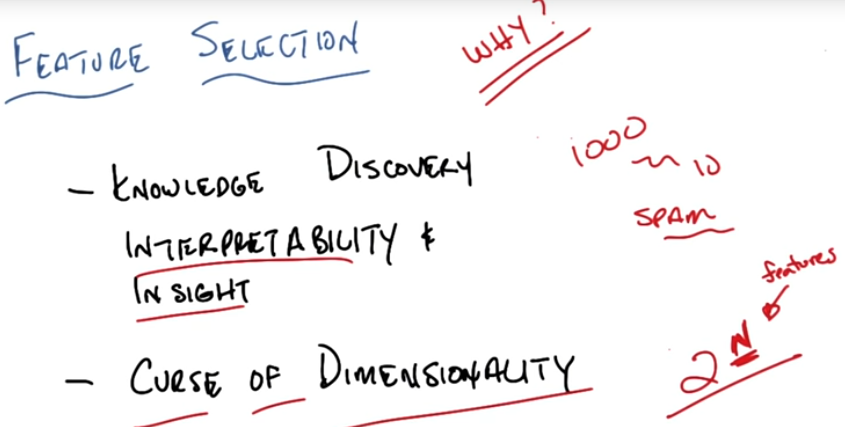
* + 1. **Introduction**

Let’s define the problem.

* + 1. **Feature Selection**

Why might we care to do this? 2 main reasons:

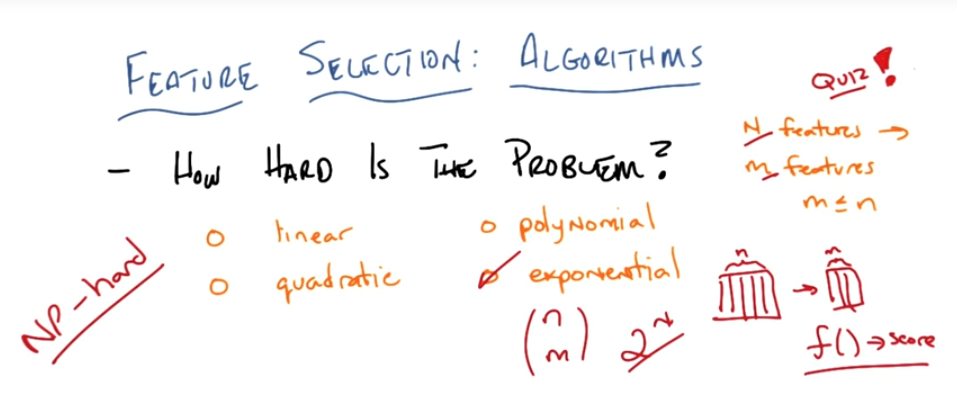
* Knowledge Discovery: to know which features matter the most
* Curse of Dimensionality: It says that the amount of data that you need grows exponentially in the number of features that you have. So it could be nice if you could have fewer features.

****

* + 1. **Quiz: Algorithms**

You’ve got a set of data with N features. And your goal is to find a subset of M features where the size of that subset is no greater than the original subset. How long does it take in terms of time complexity to do?

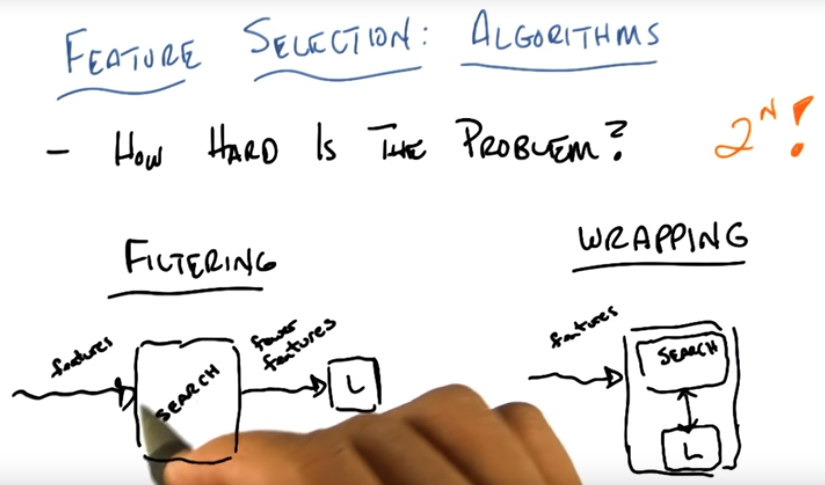
To know the best subset, only if I try all the subsets. And there’s an exponential number of subsets. What is the form of the exponential?



Of course this problem is hard, because if this problem weren’t hard then most machine learning would be pretty easy. It turns out that there are 2 ways in general that people try to approach this problem.

* + 1. **Filtering and Wrapping**

There are sort of 2 general approaches that people have used to try to attack this problem. The first is called **Filtering** and the second is called **Wrapping**.



**Filtering** works like this: you have a set of features as input. You run them through some algorithm that maximizes some kind of criteria. And then it outputs fewer features and passes those to some learning algorithm that will then use it for classification, regression. How you doing well or not is buried inside the Search algorithm.

**Wrapping,** you take your N features, and you run them into this little box that basically searches over a subset of features, asks the learning algorithm to do something with them. The learning algorithm basically reports how well it does. And it uses that to update this new subset of features that it might look for and passes to the algorithm.

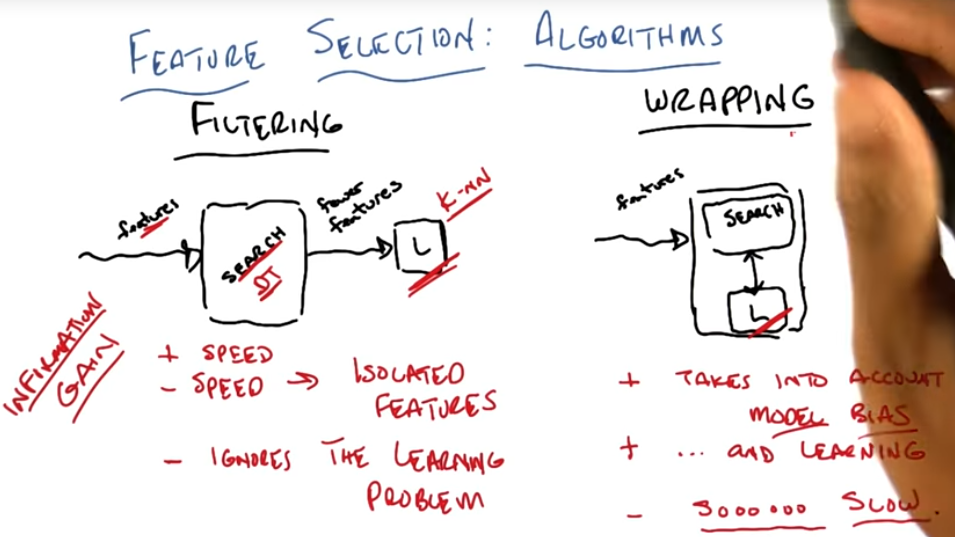
It’s called **Filtering** because this is a process that filters the features before handing it to a learning algorithm. And the other is called **Wrapping** because the search for the features is wrapped around whatever your learning algorithm is.

Which one might you pick? There are trade-offs. In the Filtering approach, the criterion or criteria is built inside the search algorithm itself without reference to the learner. By contrast, in Wrapping the criteria (or criterion if it’s a single one), is actually built inside what the learner wants. It’s built inside something like classification error or sum of squared errors, or whatever it is you measure the learner by.

* + 1. **Speed**

Filtering is faster than wrapping because you don’t have to worry about what the learner wants, you don’t have to worry about paying the cost of what the learner is going to do. You can basically just apply any fast algorithm you might imagine that takes a look at the features and does some sorting of filtering. The price you’re paying for that speed is that you tend to look at the features in isolation. You may look at a feature and say this is unimportant. But maybe that feature is important if you combined it with another feature. Kind of parity problem. Some feature might look unimportant but actually it is important.

By contrast, wrapping actually takes into account model bias, whatever the learning bias of the algorithm is, in order to determine what the best sub features are. Which means it’s actually worried about the learning problem itself. But wrapping is much slower than filtering. Because every time it tries to look at a particular subset, a candidate subset, it has to run the learning algorithm. Imagine this learning algorithm is something like neural network learning. This could take thousands and thousands of iterations, you might have to do cross validation etc. Wrapping makes a lot more sense because it takes into account the learning problem and more importantly the learning bias or the inductive bias of the learning algorithm. But it’s going to take a very long time to do it while filtering tends to go pretty well.

****

* + 1. **Wrapping**

I

* + 1. **Searching**

I

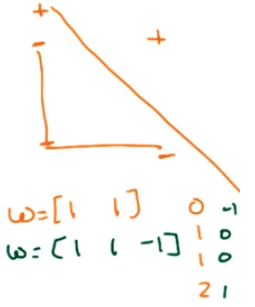
* + 1. **Quiz: Minimum features required**

I

Decision Tree case: If A is true, I can split on B and if B is true, I output pluses, otherwise minuses. This is an AND Boolean operation (if A and B are true, then it’s true)



Simple Perceptron case: We can draw A and B in a 2 dimensional plane. So it’s linearly separable. We’ll use w [1 1 -1] to A B and C, so that’s we have the result 1 when A and B are 1. So C is not useless, but less relevant.

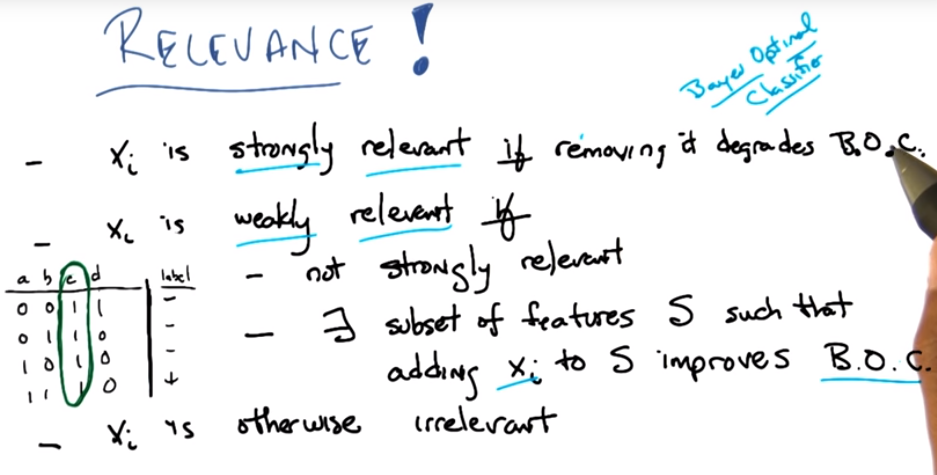


* + 1. **Relevance**

Let’s define what relevance is. So, we have a set of features and let’s just say their Xi (so there is X1, X2, X3) any particular feature Xi is going to be strongly relevant exactly in the case if removing that feature degrades the Bayes Optimal Classifier (BOC). The BOC takes the weighted average of all of the hypotheses, based on their probability of being the correct hypothesis. What we actually said is that the Bayes Optimal Classifier is the best that you could possibly do on average, if you could actually find it. Xi is strongly relevant in the case that if you didn’t have that feature, you couldn’t do as well as the Bayes Optimal Classifier that had access to all the features.

In the quiz that we did before, we know that the actual function that we’re looking for really was AND A and B. So if I remove A, I can’t actually compute A and B. Similarly, by removing B the same happens. So both are strongly relevant.

A feature is called weakly relevant if it’s the case that it’s not strongly relevant. And it happens that there exists some subset of your features, let’s call that subset S, such that if I added the feature to that subset S, it would in fact improve the BOC.



Making a little more concrete, imagine that we had another variable, let’s call it E. which had these values:



You’ll notice that E is NOT(A). That means that neither A nor E is strongly relevant, because I can remove A and still learn B AND A by basically making it B AND NOT(E). Or I could move E and still learn A AND B by simply using A and B. So B would be strongly relevant but not the other two. However, both A and E are still weakly relevant, because there exists a subset such that adding it back gives you better performance. In particular, A is weakly relevant for any subset that doesn’t include E and E is weakly relevant for any subset that doesn’t include A.

If you have a particular feature that is not strongly relevant and not weakly relevant, then we call it **irrelevant**. In this case, when you were calling C as being useless what it actually is is irrelevant. Because it provides no information to the classifier.

And yet it turned out to be helpful for the perceptron case. And that’s because there’s another notion that we could think about. Which is not relevance, but usefulness.

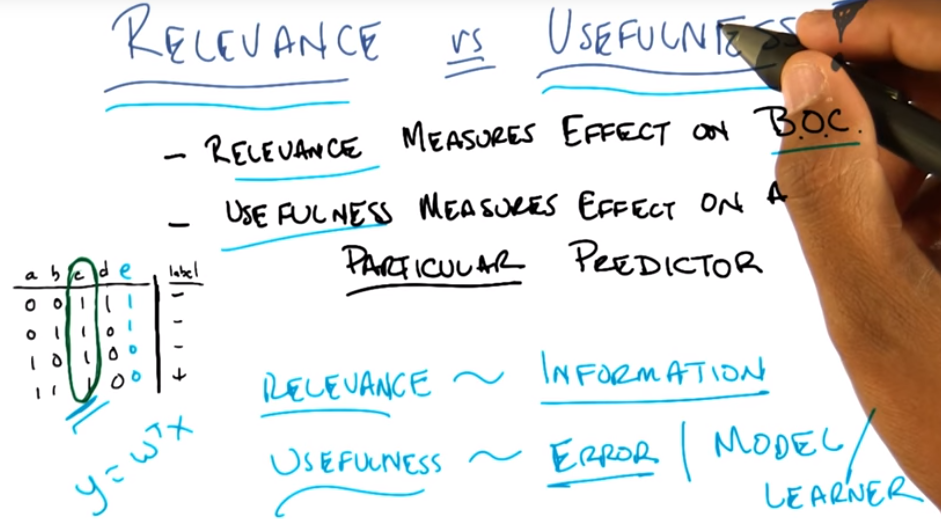
* + 1. **Relevance vs Usefulness**

A variable is relevant if it can make the BOC performance better or worse. So relevance is about information. So from the BOC point of view the only thing that matters is how much information a particular variable provides. So a variable like C here which doesn’t change, has 0 entropy, provides no information, independent of the value of the label and therefore cannot be relevant to the BOC.

Usefulness is exactly about effect on error given a particular classifier, or some specific model. Usefulness is exactly about minimizing error given some particular model or some particular learning algorithm. In this case, although C is clearly not relevant, it is in fact useful at least for something like W transpose X. Now, this is not useful for a decision tree, nor is it relevant. It is not relevant to this particular problem, but it is however useful for some algorithms.

All that clustering that we did before, like K-MEANS and E.M., you could think of those as a kind of feature transformation algorithm which is what we’ll be talking about next where you’ve taken a bunch of features and you’ve converted them into something simple like a label. And whether that is a good label or a bad label depends entirely upon whether you can then do some kind of classification or regression problem later.

There’s a lot of algorithms and if you go and you look at the material we’ve made available to everyone you’ll be able to see some of those algorithms discussed in more detail.



* + 1. **What have we learned**

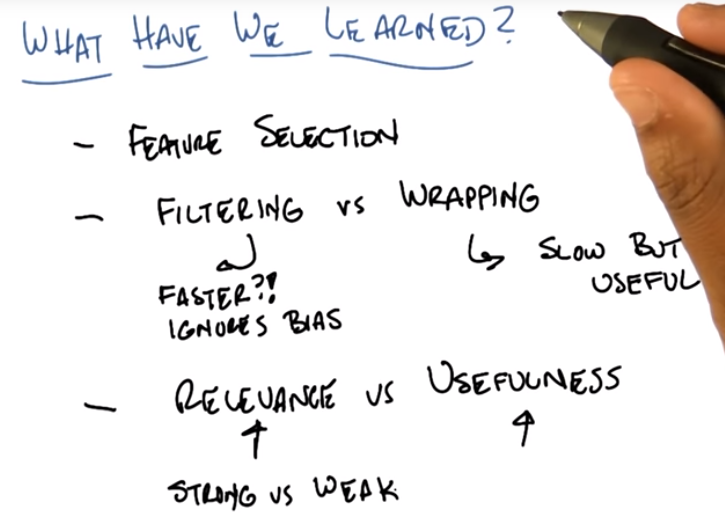
Definition for Feature Selection: which was getting a subset of the features to feed to some learning algorithms after the selection has taken place.

Wrapping and Filtering: methods for feature selection. Wrapping is slow but actually seems like it solves the problem that matters. Filtering is simpler, possibly faster, but maybe misses the point. It ignores bias.

Useful features and Relevant features: Relevant features give you information about the classification problem, or the regression problem, that you care about. Useful features help you to actually do the learning, given some specific algorithm.

Bayes Optimal Classifier: it’s sort of the gold standard, it is the ultimate that you could do if you had everything and all the time in the world.

Strong and Weak Relevance



* 1. **PCA**

PCA, principal component analysis, is a method for feature selection that turns a set of correlated variables into the underlying set of orthogonal variables.

* + 1. **Quiz: Data Dimensionality**

A

* + 1. **Quiz: Trickier Data Dimensionality**

A

* + 1. **Quiz: One-Dimensional, or Two?**

A

* + 1. **Quiz: Slightly Less Perfect Data**

A

* + 1. **Quiz: Trickiest Data Dimensionality**

A

* + 1. **PCA for Data Transformation**

A

* + 1. **Quiz: Center of a New Coordinate System**

A

* + 1. **Quiz: Principal Axis of New Coordinate System**

A

* + 1. **Quiz: Second Principal Component of New System**

A

* + 1. **Quiz: Practice Finding Centers**

A

* + 1. **Quiz: Practice Finding New Axes**

A

* + 1. **Quiz: Which Data is Ready for PCA**

A

* + 1. **Quiz: When does an Axis Dominate**

A

* + 1. **Quiz: Measurable vs. Latent Features Quiz**

A

* + 1. **Quiz: From four features to two**

A

* + 1. **Quiz: Compression while preserving Information**

A

* + 1. **Quiz: Composite Features**

A

* + 1. **Quiz: Maximal Variance**

A

* + 1. **Quiz: Advantages of Maximal Variance**

A

* + 1. **Quiz: Maximal Variance and Information Loss**

A

* + 1. **Info Loss and Principal Components**

A

* + 1. **Quiz: Neighborhood Composite Feature**

A

* + 1. **PCA for Feature Transformation**

A

* + 1. **Quiz: Maximum Number of PCs Quiz**

A

* + 1. **Review/Definition of PCA**

A

* + 1. **Applying PCA to real data**

A

* + 1. **Quiz: PCA On the Enron Finance Data**

A

* + 1. **PCA in Sklearn**

A

* + 1. **When to use PC**

A

* + 1. **Quiz: PCA For Facial Recognition**

A

* + 1. **Eigenfaces Code**

A

* 1. **PCA Mini-Project**

In this mini-project, you’ll apply principal component analysis to facial recognition.

* + 1. **PCA Mini-Project Video**
    2. **PCA Mini-Project**
    3. **Quiz: Explained Variance of Each PC**
    4. **Quiz: How many PCs to use?**
    5. **Quiz: F1 Score vs. No. Of PCs Used**
    6. **Quiz: Dimensionality Reduction and Overfitting**
    7. **Quiz: Selecting Principal Components**
  1. **Feature Transformation**

There are several methods for feature selection. In this lesson, we’ll discuss principal (PCA), independent (ICA), and relevant component analysis (RCA).

* + 1. **Introduction**
    2. **Feature Transformation**
    3. **What are our Features**
    4. **Words Like Tesla**
    5. **Independent Component Analysis**
    6. **Independent Component Analysis Two**
    7. **Cocktail Party Problem**
    8. **Matrix**
    9. **Quiz: PCA vs ICA**
    10. **PCA vs ICA Continued**
    11. **Alternatives**
    12. **Quiz: Alternatives Quiz**
    13. **Alternatives Two**
    14. **Wrap up**
  1. **Summary**

In this lesson, we’ll recap the lessons we’ve learned from unsupervised learning.

* + 1. **What have we learned**
  1. **Project: Identifying Customers by clustering them**

Now that you’ve learned a lot about unsupervised learning, it’s time to apply that to a project.

* + 1. **Overview**
    2. **Software Requirements**
    3. **Starting the Project**
    4. **Submitting the Project**
    5. **Project: Creating Customer Segments**

1. **Machine Learning Capstone**
   1. **Project: Writing up a Capstone Proposal**

Before working on a machine learning problem, write up a proposal of your project to get valuable feedback.

* + 1. **Overview**
    2. **Description**
    3. **Software and Data Requirements**
    4. **Proposal Guidelines**
    5. **Submitting the Project**
    6. **Project: Capstone Proposal**
  1. **Machine Learning Capstone Project**

Now you will put your machine learning skills to test by solving a real world problem using the algorithms you have learned in the program so far.

* + 1. **Overview**
    2. **Description**
    3. **Software and Data Requirements**
    4. **Report Guidelines**
    5. **Example Reports**
    6. **Submitting the Project**
    7. **Project: Capstone Project**