Part II: (Automatic) Feature Selection

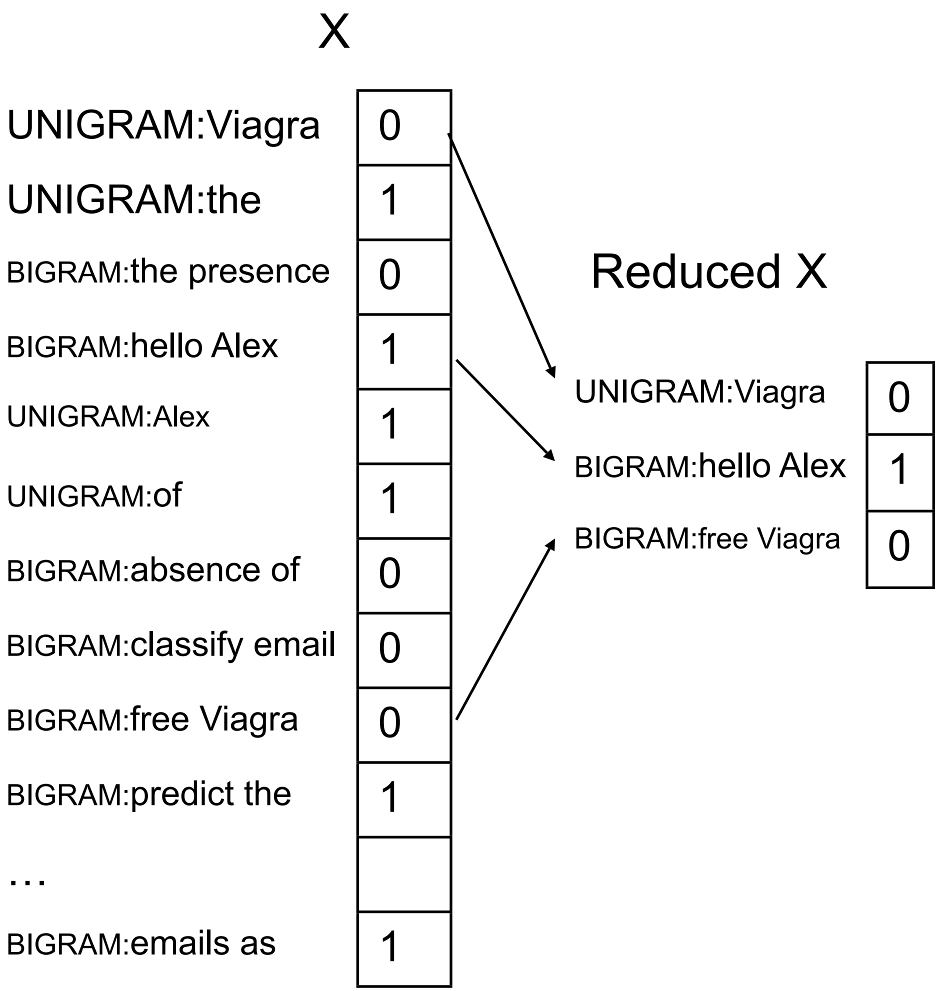
What is feature selection?

- Reducing the feature space by throwing out some of the features
- Motivating idea: try to find a simple, "parsimonious" model
  - Occam's razor: simplest explanation that accounts for the data is best

What is feature selection?

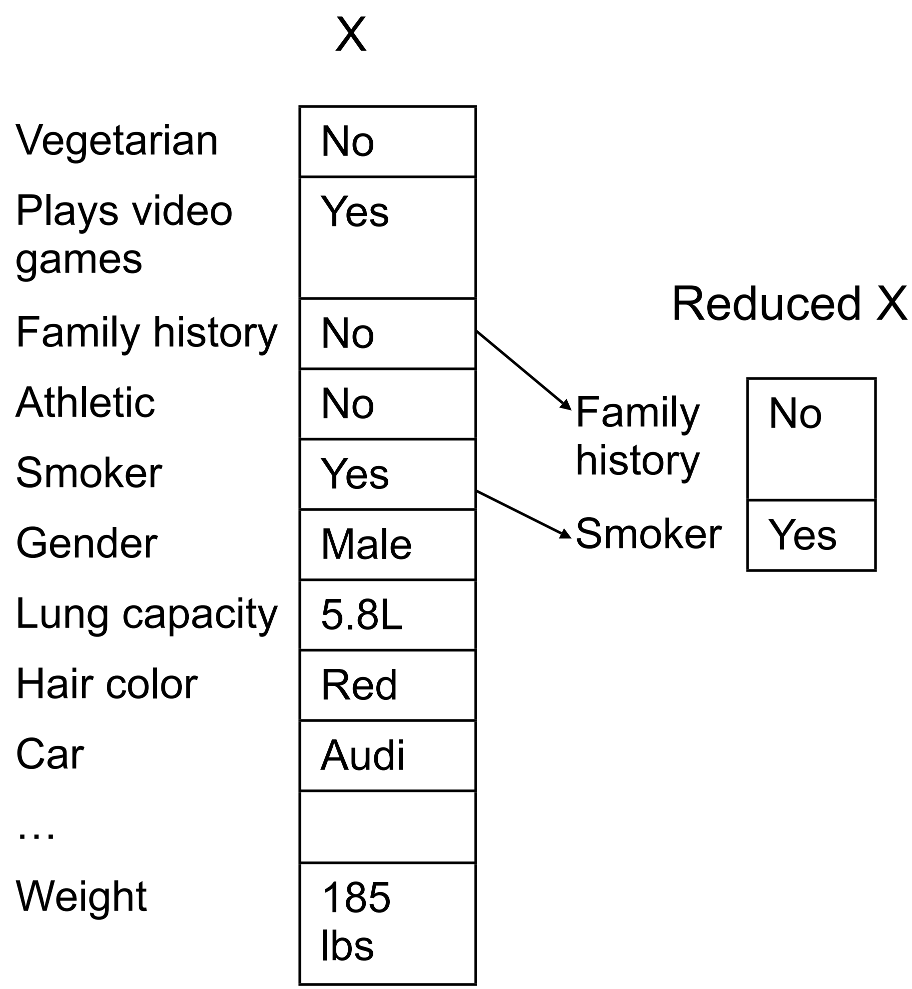
Task: classify emails as spam, work, ...

Data: presence/absence of words



Task: predict chances of lung disease

Data: medical history survey



Outline

- Review/introduction
  - What is feature selection? Why do it?
- Filtering
- Model selection
  - Model evaluation
  - Model search
- Regularization
- Summary recommendations

Why do it?

- Case 1: We're interested in *features*—we want to know which are relevant. If we fit a model, it should be *interpretable.*

- Case 2: We're interested in *prediction;* features are not interesting in themselves, we just want to build a good classifier (or other kind of predictor).

Why do it? Case 1.

We want to know which features are relevant; we don't necessarily want to do prediction.

- What causes lung cancer?
  - Features are aspects of a patient's medical history
  - Binary response variable: did the patient develop lung cancer?
  - Which features best predict whether lung cancer will develop? Might want to legislate against these features.
- What causes a program to crash? [Alice Zheng '03, '04, '05]
  - Features are aspects of a single program execution
    - Which branches were taken?
    - What values did functions return?
  - Binary response variable: did the program crash?
  - Features that predict crashes well are probably bugs

Why do it? Case 2.

We want to build a good predictor.

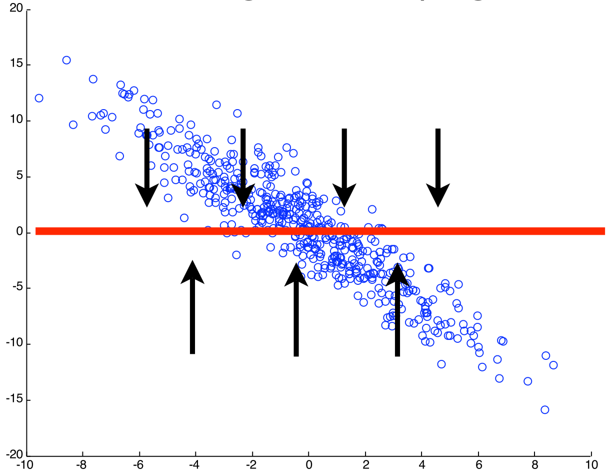
- Common practice: coming up with as many features as possible (e.g. > 10^6 not unusual)
  - Training might be too expensive with all features
  - The presence of irrelevant features hurts generalization.
- Classification of leukemia tumors from microarray gene expression data [Xing, Jordan, Karp '01]
  - 72 patients (data points)
  - 7130 features (expression levels of different genes)
- Embedded systems with limited resources
  - Classifier must be compact
  - Voice recognition on a cell phone
  - Branch prediction in a CPU
- Web-scale systems with zillions of features
  - user-specific n-grams from gmail/yahoo spam filters

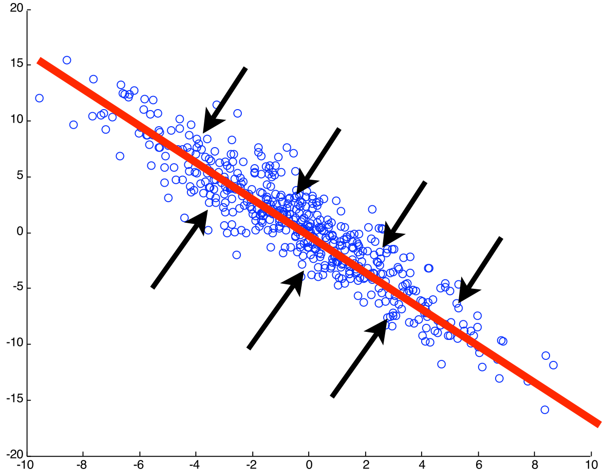
Get at Case 1 through Case 2

- Even if we just want to identify features, it can be useful to *pretend* we want to do prediction.
- Relevant features are (typically) exactly those that most aid prediction.
- But not always. Highly correlated features may be redundant but both interesting as "causes".
  - e.g. smoking in the morning, smoking at night

Feature selection vs. Dimensionality reduction

- Removing features:
  - Equivalent to projecting data onto lower-dimensional linear subspace perpendicular to the feature removed
- Percy's lecture: dimensionality reduction
  - allow other kinds of projection.
- The machinery involved is very different
  - Feature selection can can be faster at test time
  - Also, we will assume we have labeled data. Some dimensionality reduction algorithm (e.g. PCA) do not exploit this information





Outline

- Review/introduction
  - What is feature selection? Why do it?
- Filtering
- Model selection
  - Model evaluation
  - Model search
- Regularization
- Summary

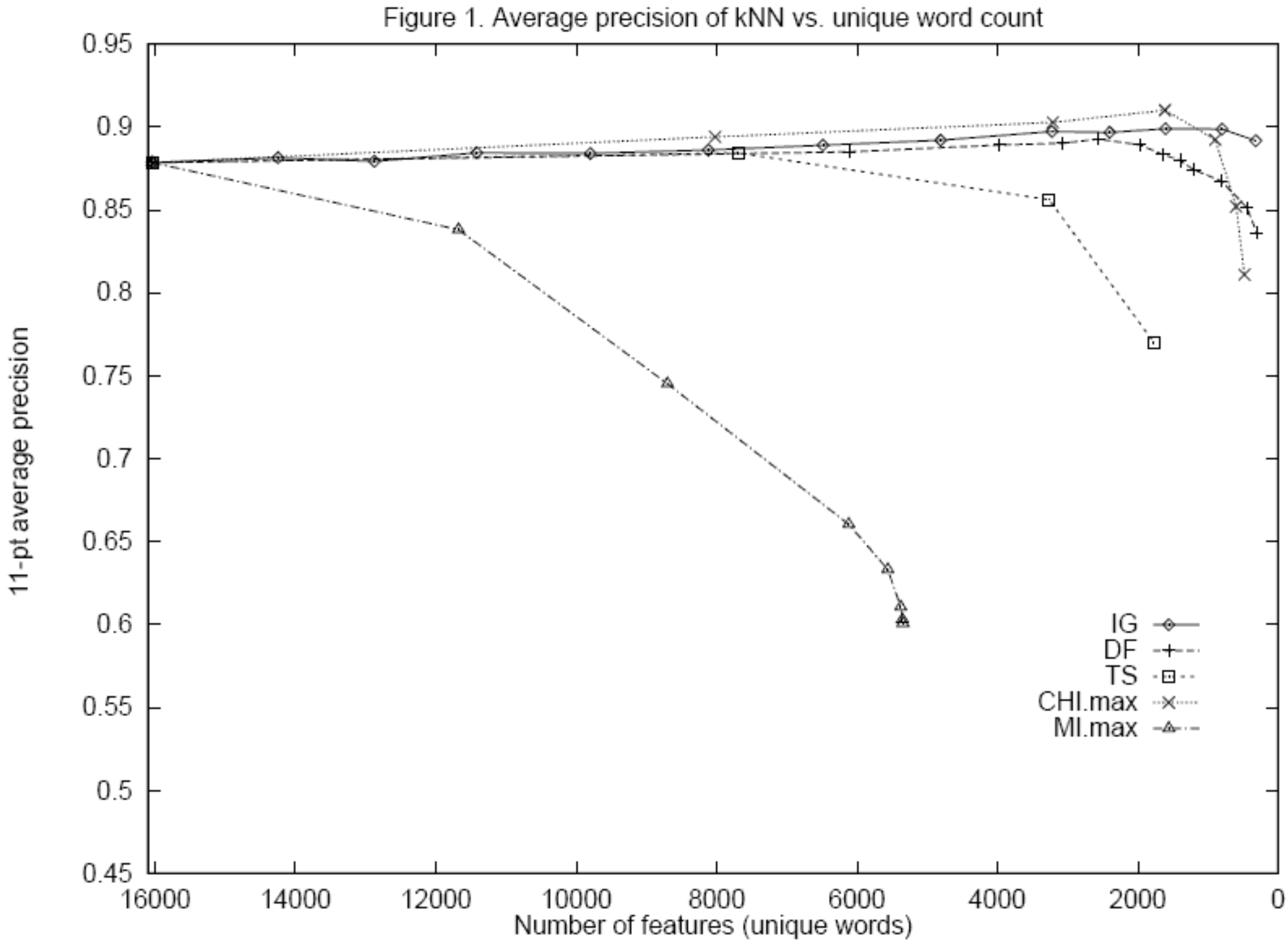
Filtering

Simple techniques for weeding out irrelevant features without fitting model

Filtering

- Basic idea: assign heuristic score to each feature f to filter out the "obviously" useless ones.
  - Does the individual feature seems to help prediction?
  - Do we have enough data to use it reliably?
  - Many popular scores [see Yang and Pederson '97]
    - Classification with categorical data: Chi-squared, information gain, document frequency
    - Regression: correlation, mutual information
    - They all depend on one feature at the time (and the data)
- Then somehow pick how many of the highest scoring features to keep

Comparison of filtering methods for text categorization [Yang and Pederson ’97]



Filtering

- Advantages:
  - Very fast
  - Simple to apply
- Disadvantages:
  - Doesn’t take into account interactions between features: Apparently useless features can be useful when grouped with others
- Suggestion: use light filtering as an efficient initial step if running time of your fancy learning algorithm is an issue

Outline

- Review/introduction
  - What is feature selection? Why do it?
- Filtering
- Model selection
  - Model evaluation
  - Model search
- Regularization
- Summary

Model Selection

- Choosing between possible models of varying complexity
  - In our case, a “model” means a set of features
- Running example: linear regression model

Linear Regression Model

Input : $\boldsymbol{x} \in \mathbb{R}^d$

Parameters: $\boldsymbol{w} \in \mathbb{R}^{d+1}$

Response : $y \in \mathbb{R}$

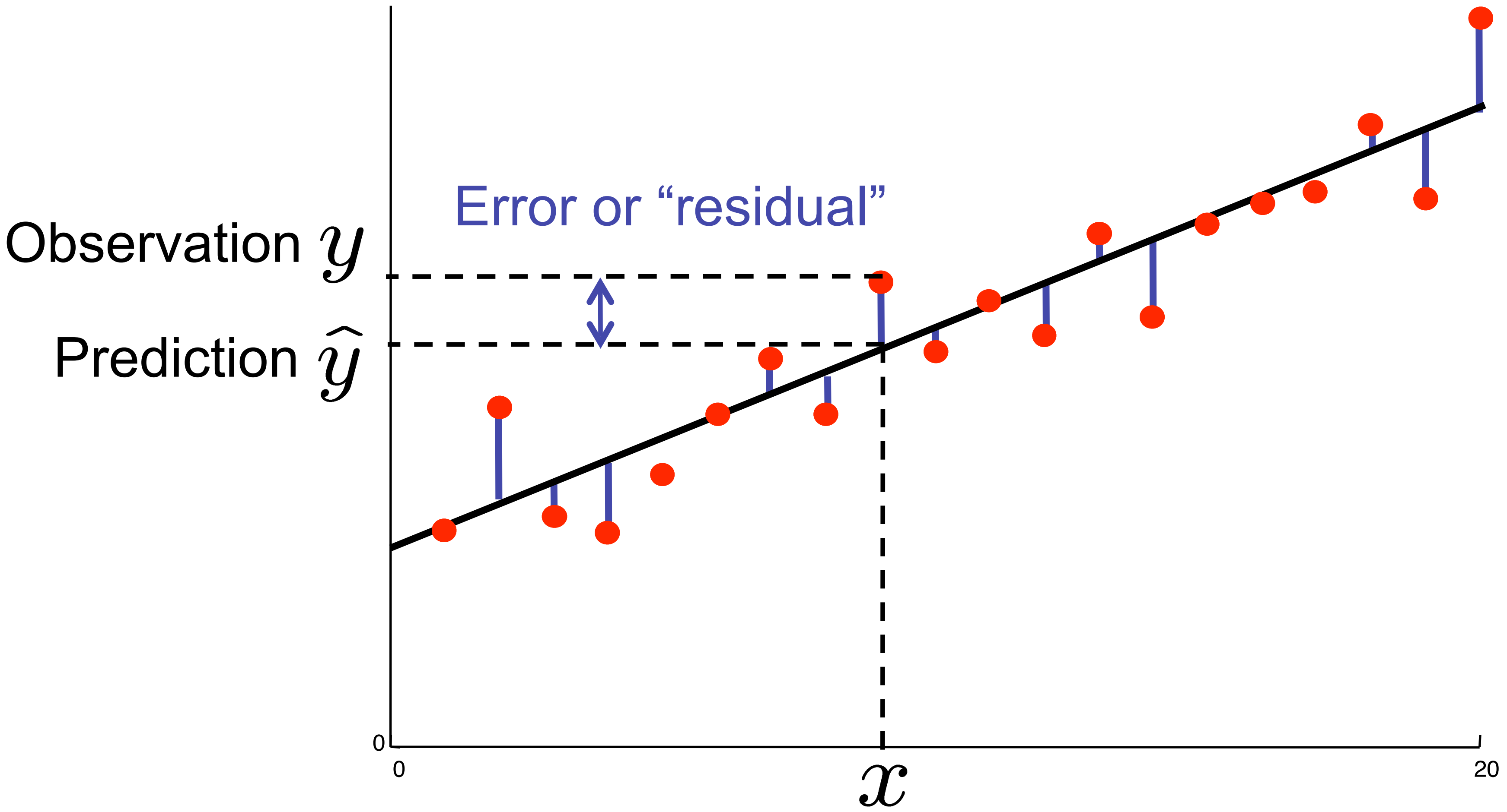
Prediction : $y = \boldsymbol{w}^\top \boldsymbol{x}$

- Recall that we can fit (learn) the model by minimizing the squared error:

$$\hat{\boldsymbol{w}} = \mathrm{argmin}_{\boldsymbol{w}} \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2$$

Least Squares Fitting

(Fabian's slide from 3 weeks ago)



Sum squared error: $L(w) = \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2$

Naïve training error is misleading

Input : $\boldsymbol{x} \in \mathbb{R}^d$

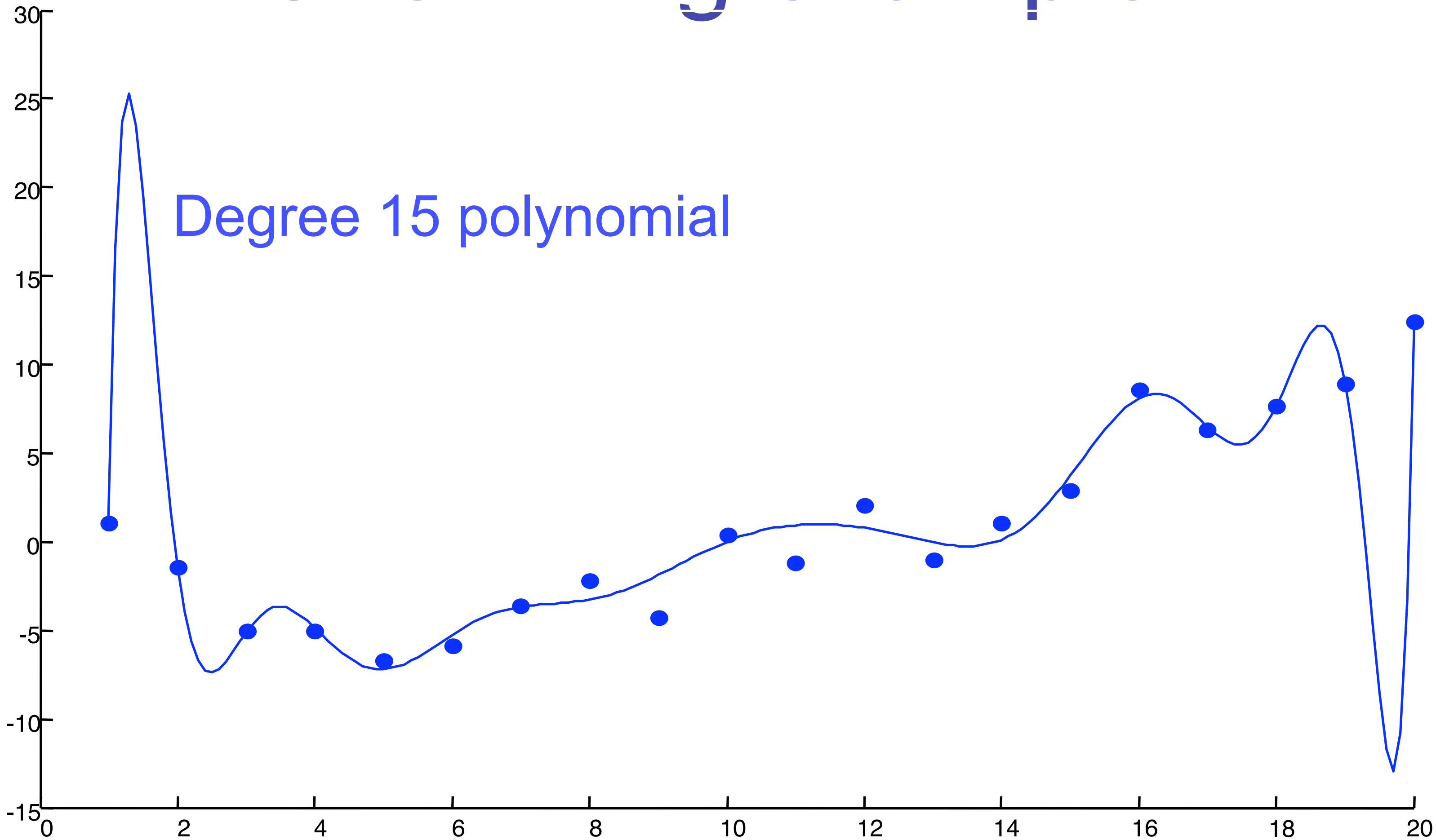
Parameters: $\boldsymbol{w} \in \mathbb{R}^{d+1}$

Response : $y \in \mathbb{R}$

Prediction : $y = \boldsymbol{w}^\top \boldsymbol{x}$

- Consider a reduced model with only those features $\boldsymbol{x}_f$ for $f \in s \subseteq \{1, 2, \ldots, d\}$
  - Squared error is now $L_s(\boldsymbol{w}_s) = \sum_{i=1}^{n} (y_i - \boldsymbol{w}_s^\top \boldsymbol{x}_{i,s})^2$
- Is this new model better? Maybe we should compare the training errors to find out?
- Note $\min_{\boldsymbol{w}_s} L_s(\boldsymbol{w}_s) \geq \min_{\boldsymbol{w}} L(\boldsymbol{w})$
  - Just zero out terms in $\boldsymbol{w}$ to match $\boldsymbol{w}_s$.
- Generally speaking, training error will only go up in a simpler model. So why should we use one?

Overfitting example 1

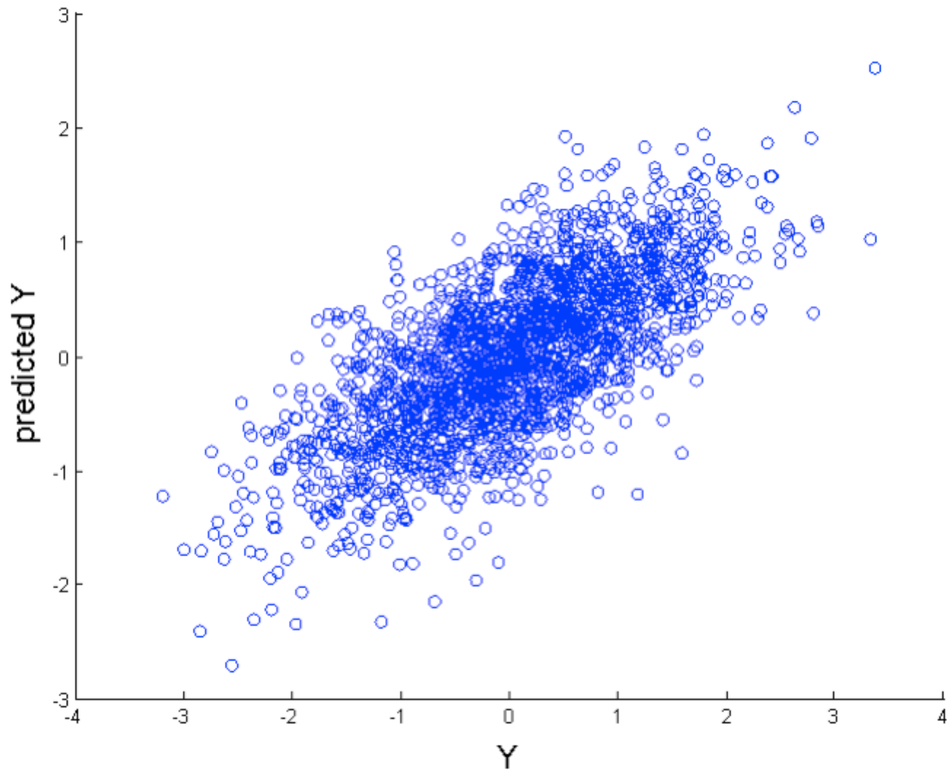


- This model is too rich for the data
- Fits training data well, but doesn't generalize.

(From Fabian's lecture)

Overfitting example 2

- Generate 2000 $\boldsymbol{x}_i \in \mathbb{R}^{1000}$, $\boldsymbol{x}_i \sim \mathcal{N}(0, I)$ i.i.d.
- Generate 2000 $y_i \in \mathbb{R}$, $y_i \sim \mathcal{N}(0, 1)$ i.i.d. *completely independent of the* $\boldsymbol{x}_i$*'s*
  - We shouldn't be able to predict y at *all* from $\boldsymbol{x}$
- Find $\hat{\boldsymbol{w}} = \operatorname{argmin}_{\boldsymbol{w}} L(\boldsymbol{w})$
- Use this to predict y_i for each $\boldsymbol{x}_i$ by $\hat{y}_i = \hat{\boldsymbol{w}}^\top \boldsymbol{x}_i$



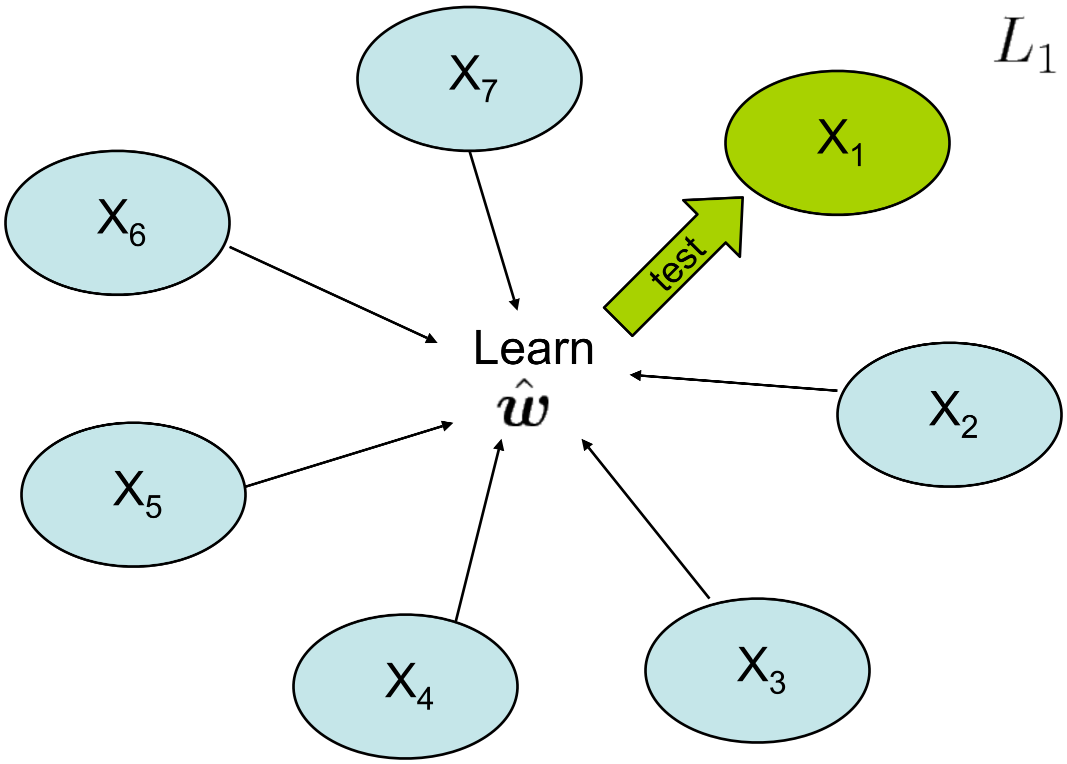
It really looks like we've found a relationship between $\boldsymbol{x}$ and y! But no such relationship exists, so $\hat{\boldsymbol{w}}$ will do no better than random on new data.

Model evaluation

- Moral 1: In the presence of many irrelevant features, we might just fit noise.
- Moral 2: Training error can lead us astray.
- To evaluate a feature set s, we need a better scoring function $K(s)$
- We're not ultimately interested in *training* error; we're interested in *test* error (error on new data).
- We can estimate test error by pretending we haven't seen some of our data.
  - Keep some data aside as a *validation set*. If we don't use it in training, then it's a better test of our model.

K-fold cross validation

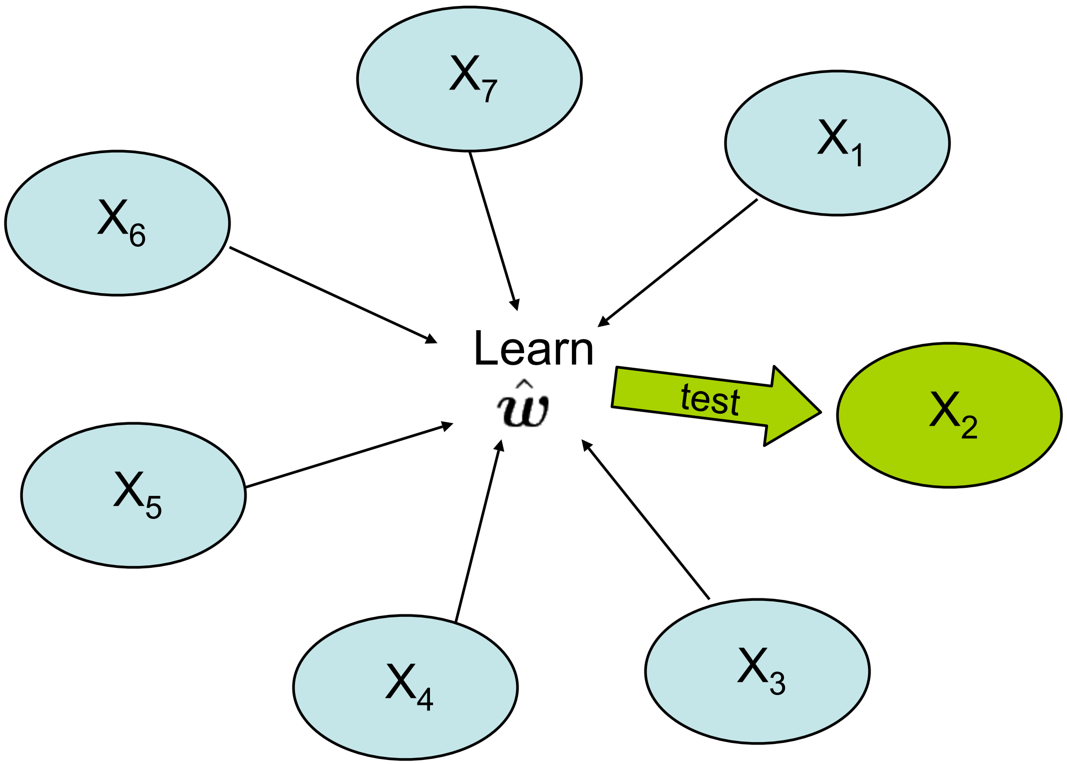
- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors



$$L_1 = \sum_{(\boldsymbol{x},y)\in X_1} (y - \hat{\boldsymbol{w}}^\top \boldsymbol{x})$$

K-fold cross validation

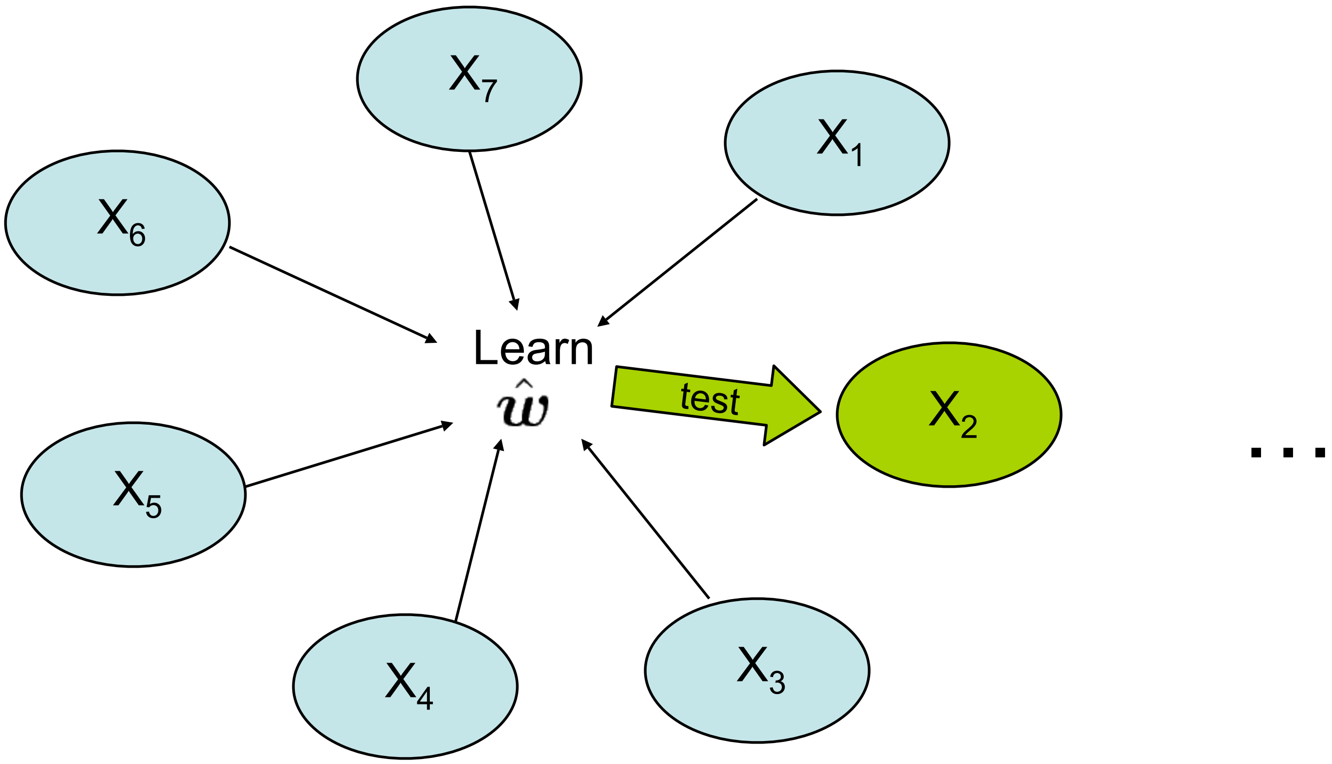
- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors



$$L_2 = \sum_{(\boldsymbol{x},y)\in X_2} (y - \hat{\boldsymbol{w}}^\top \boldsymbol{x})$$

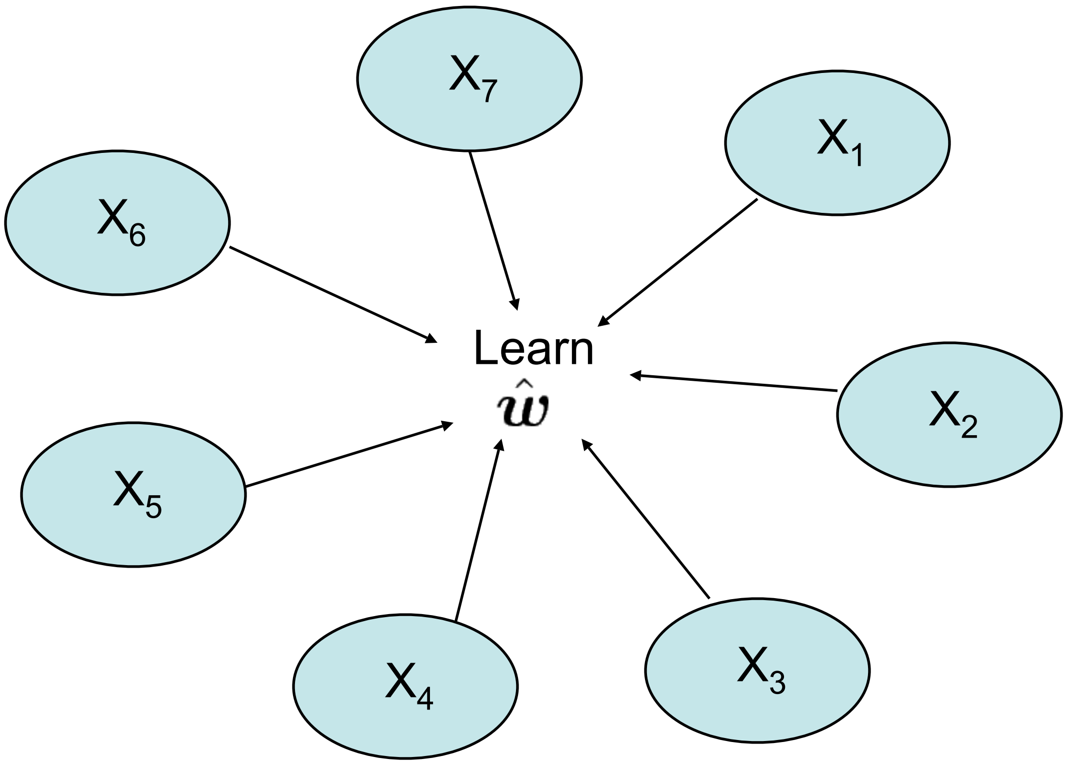
K-fold cross validation

- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors



K-fold cross validation

- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors



$$CV(s) = \frac{1}{K} \sum_{i=1}^{K} L_i$$

Model Search

- We have an objective function $\mathrm{K}(s) = \mathrm{CV}(s)$
  - Time to search for a good model.
- This is known as a "wrapper" method
  - Learning algorithm is a black box
  - Just use it to compute objective function, then do search
- Exhaustive search expensive
  - for n features, 2^n possible subsets s
- Greedy search is common and effective

Model search

Forward selection

```
Initialize s={}
Do:
       Add feature to s
       which improves K(s) most
While K(s) can be improved
```

Backward elimination

```
Initialize s={1,2,…,n}
Do:
       remove feature from s
       which improves K(s) most
While K(s) can be improved
```

- Backward elimination tends to find better models
  - Better at finding models with interacting features
  - But it is frequently too expensive to fit the large models at the beginning of search
- Both can be too greedy.

Model search

- More sophisticated search strategies exist
  - Best-first search
  - Stochastic search
  - See “Wrappers for Feature Subset Selection”, Kohavi and John 1997
- For many models, search moves can be evaluated quickly without refitting
  - E.g. linear regression model: add feature that has most covariance with current residuals
- YALE can do feature selection with cross-validation and either forward selection or backwards elimination.
- Other objective functions exist which add a model-complexity penalty to the training error
  - AIC: add penalty d to log-likelihood (number of features).
  - BIC: add penalty $d \log n$ (*n* is the number of data points)

Summary: feature engineering

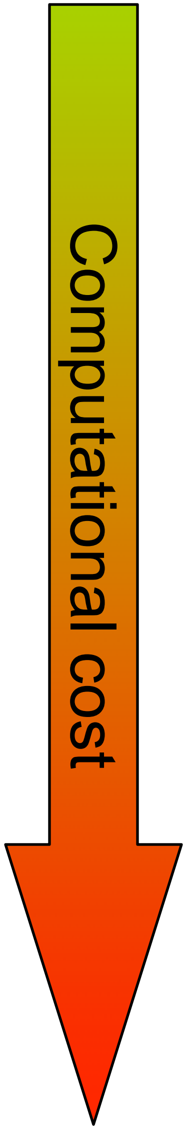
- Feature engineering is often crucial to get good results
- Strategy: overshoot and regularize
  - Come up with lots of features: better to include irrelevant features than to miss important features
  - Use regularization or feature selection to prevent overfitting
  - Evaluate your feature engineering on DEV set. Then, when the feature set is frozen, evaluate on TEST to get a final evaluation (Daniel will say more on evaluation next week)

Summary: feature selection

When should you do it?

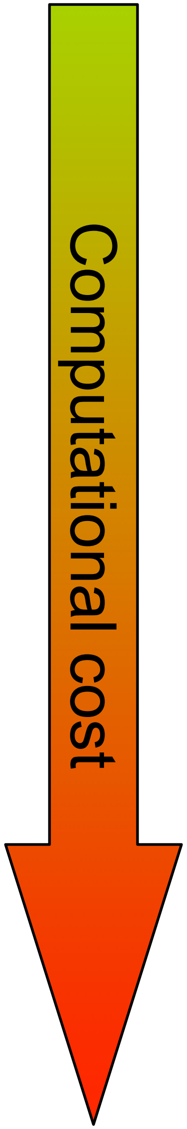
- If the only concern is accuracy, and the whole dataset can be processed, feature selection not needed (as long as there is regularization)
- If computational complexity is critical (embedded device, web-scale data, fancy learning algorithm), consider using feature selection
  - But there are alternatives: e.g. the Hash trick, a fast, non-linear dimensionality reduction technique [Weinberger et al. 2009]
- When you care about the feature themselves
  - Keep in mind the correlation/causation issues
  - See [Guyon et al., Causal feature selection, 07]

Summary: how to do feature selection



- Filtering
- L_1 regularization (embedded methods)
- Wrappers
    - Forward selection
    - Backward selection
    - Other search
    - Exhaustive

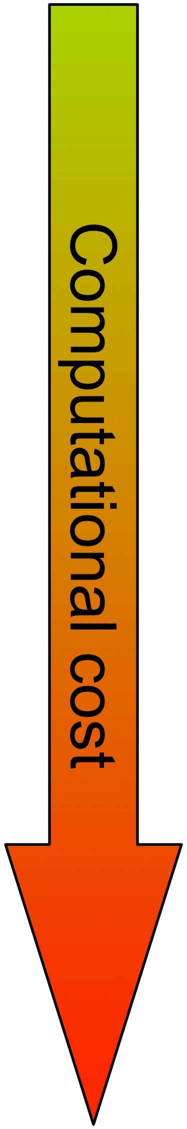
Summary: how to do feature selection



- *Filtering*
- L_1 regularization (embedded methods)
- Wrappers
  - Forward selection
  - Backward selection
  - Other search
  - Exhaustive

- Good preprocessing step
- Fails to capture relationship between features

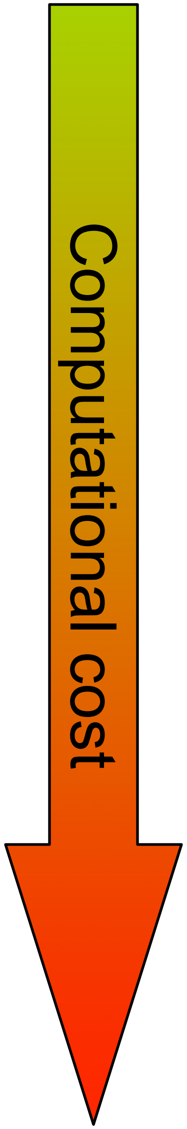
Summary: how to do feature selection



- Filtering
- *L_1 regularization (embedded methods)*
- Wrappers
  - Forward selection
  - Backward selection
  - Other search
  - Exhaustive

- Fairly efficient
  - LARS-type algorithms now exist for many linear models.

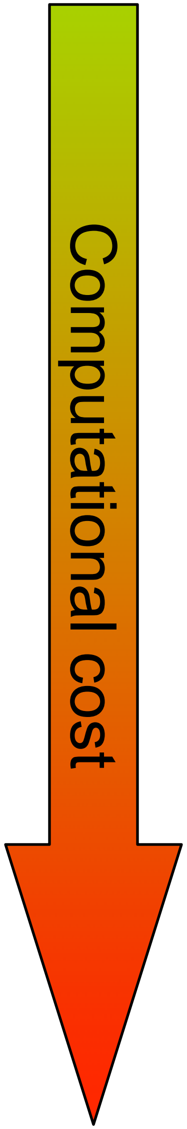
Summary: how to do feature selection



- Filtering
- L_1 regularization (embedded methods)
- *Wrappers*
    - Forward selection
    - Backward selection
    - Other search
    - Exhaustive

- Most directly optimize prediction performance
- Can be very expensive, even with greedy search methods
- Cross-validation is a good objective function to start with

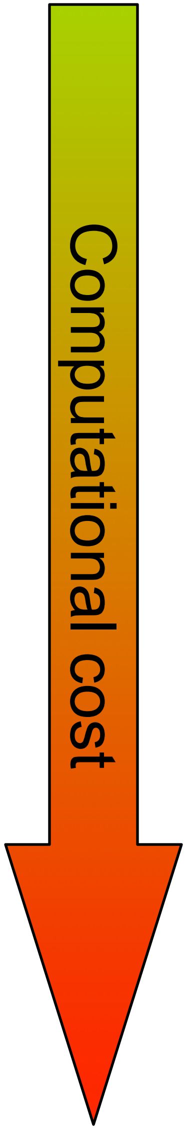
Summary: how to do feature selection



- Filtering
- L_1 regularization (embedded methods)
- Wrappers
  - *Forward selection*
  - *Backward selection*
  - Other search
  - Exhaustive

- Too greedy—ignore relationships between features
- Easy baseline
- Can be generalized in many interesting ways
  - Stagewise forward selection
  - Forward-backward search
  - Boosting

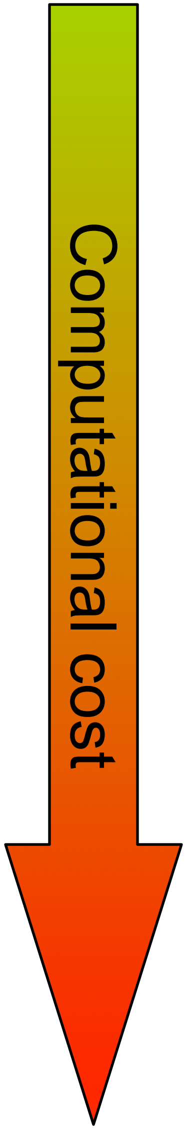
Summary: how to do feature selection



- Filtering
- L_1 regularization (embedded methods)
- Wrappers
  - Forward selection
  - Backward selection
  - *Other search*
  - Exhaustive

- Generally more effective than greedy

Summary: how to do feature selection



- Filtering
- L_1 regularization (embedded methods)
- Wrappers
  - Forward selection
  - Backward selection
  - Other search
  - *Exhaustive*

- The "ideal"
- Very seldom done in practice
- With cross-validation objective, there's a chance of over-fitting
  - *Some* subset might randomly perform quite well in cross-validation