Feature Engineering and Selection

CS 294: Practical Machine Learning
October 1st, 2009

Alexandre Bouchard-Côté

Abstract supervised setup

- Training : $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$
- $\boldsymbol{x}_i$: input vector

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,n} \end{bmatrix}, \quad x_{i,j} \in \mathbb{R}$$

- y : response variable
  - $y \in \{-1, 1\}$: binary classification
  - $y \in \mathbb{R}$: regression
  - what we want to be able to predict, having observed some new $\boldsymbol{x}$.

Concrete setup

Input | **Output**

“Danger”

Featurization

Input | **Features** | **Output**

$$\begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,n} \end{bmatrix}$$

"Danger"

$$\begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,n} \end{bmatrix}$$

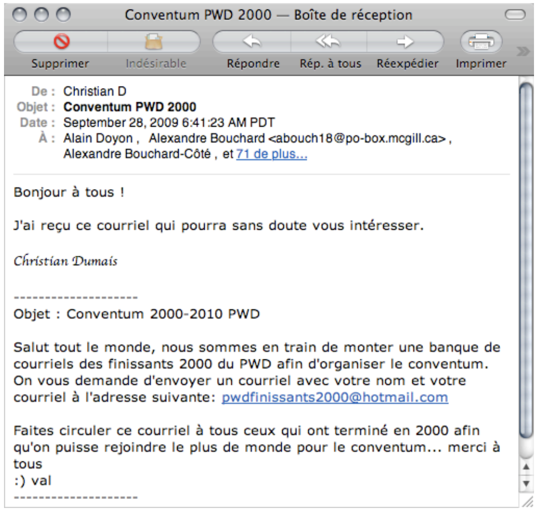
Outline

- Today: how to featurize effectively
  - Many possible featurizations
  - Choice can drastically affect performance
- Program:
  - Part I : Handcrafting features: examples, bag of tricks (feature engineering)
  - Part II: Automatic feature selection

Part I: Handcrafting Features

Machines still need us

Example 1: email classification



PERSONAL

- Input: a email message
- Output: is the email...
  - – spam,
  - – work-related,
  - – personal, ...

Basics: bag of words

- Input: $\boldsymbol{x}$ (email-valued)
- Feature vector:

$$f(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ \vdots \\ f_n(\boldsymbol{x}) \end{bmatrix}, \quad \text{e.g. } f_1(\boldsymbol{x}) = \begin{cases} 1 & \text{if the email contains ``Viagra''} \\ 0 & \text{otherwise} \end{cases}$$

Indicator or Kronecker delta function

- Learn one weight vector for each class:

$$w_y \in \mathbb{R}^n, \quad y \in \{\text{SPAM,WORK,PERS}\}$$

- Decision rule: $\hat{y} = \text{argmax}_y \langle w_y, f(\boldsymbol{x}) \rangle$

Implementation: exploit sparsity

Feature ~~vector~~ hashtable

$f(x)$

```
extractFeature(Email e) {

  result <- hashtable

  for (String word : e.getWordsInBody())
    result.put("UNIGRAM:" + word, 1.0)

  String previous = "#"
  for (String word : e.getWordsInBody()) {
    result.put("BIGRAM:"+ previous + " " + word, 1.0)
    previous = word
  }

  return result
}
```

Feature template 1:
UNIGRAM:Viagra

Feature template 2:
BIGRAM:Cheap Viagra

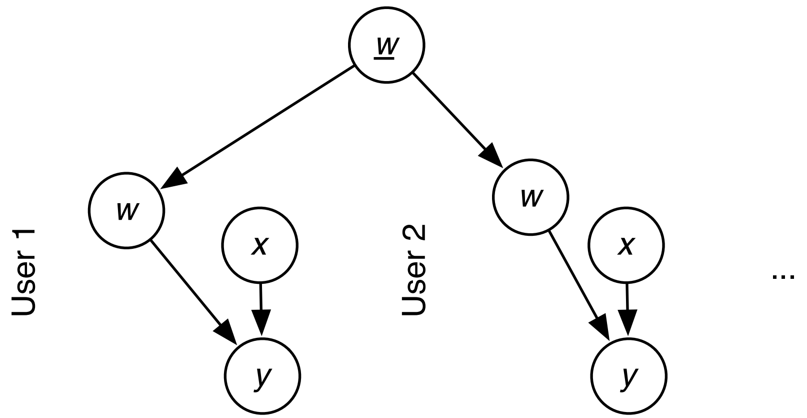
Features for multitask learning

- Each user inbox is a separate learning problem
  - E.g.: Pfizer drug designer's inbox
- Most inbox has very few training instances, but all the learning problems are clearly related

Features for multitask learning

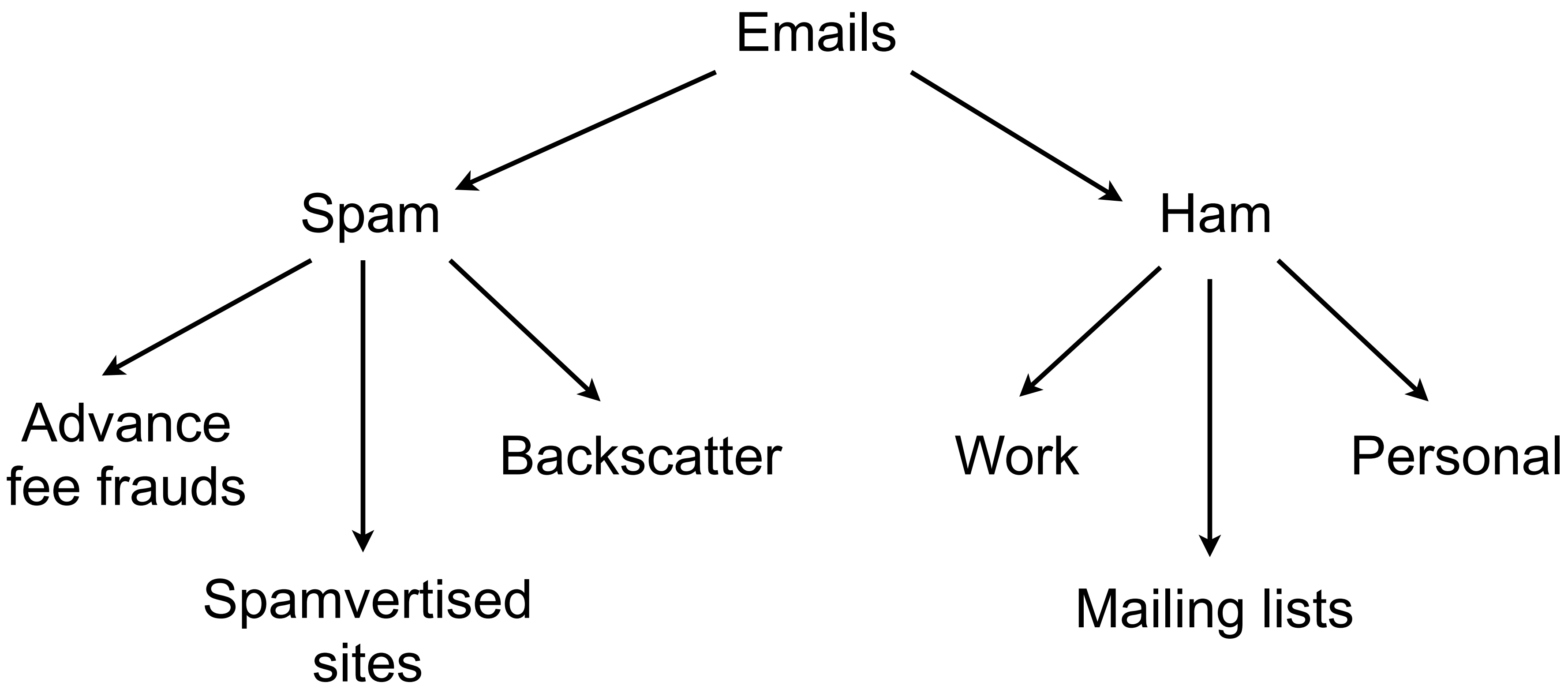
[e.g.:Daumé 06]

- Solution: include both user-specific and global versions of each feature. E.g.:
  - UNIGRAM:Viagra
  - USER_id4928-UNIGRAM:Viagra
- Equivalent to a Bayesian hierarchy under some conditions (Finkel et al. 2009)



Structure on the output space

- In multiclass classification, output space often has known structure as well
- Example: a hierarchy:



Structure on the output space

- Slight generalization of the learning/ prediction setup: allow features to depend both on the input *x* and on the class *y*

<u>Before:</u>

- One weight/class: $w_y \in \mathbb{R}^n,$
- Decision rule: $\hat{y} = \operatorname{argmax}_y \langle w_y, f(\boldsymbol{x}) \rangle$

<u>After:</u>

- Single weight: $w \in \mathbb{R}^m,$
- New rule: $\hat{y} = \operatorname{argmax}_y \langle w, f(\boldsymbol{x}, y) \rangle$

Structure on the output space

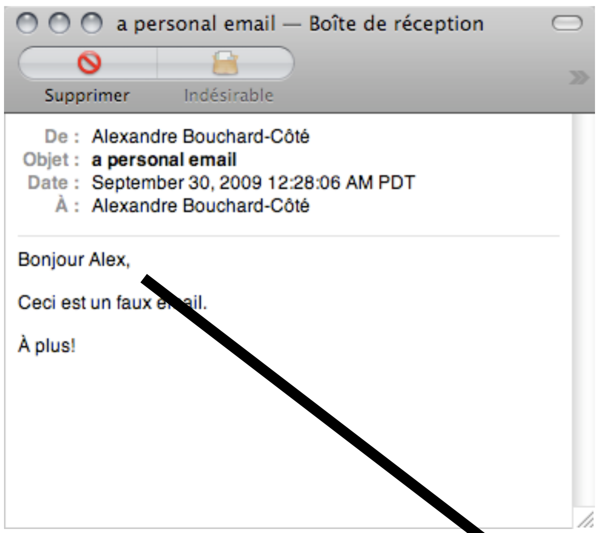
- At least as expressive: conjoin each feature with all output classes to get the same model

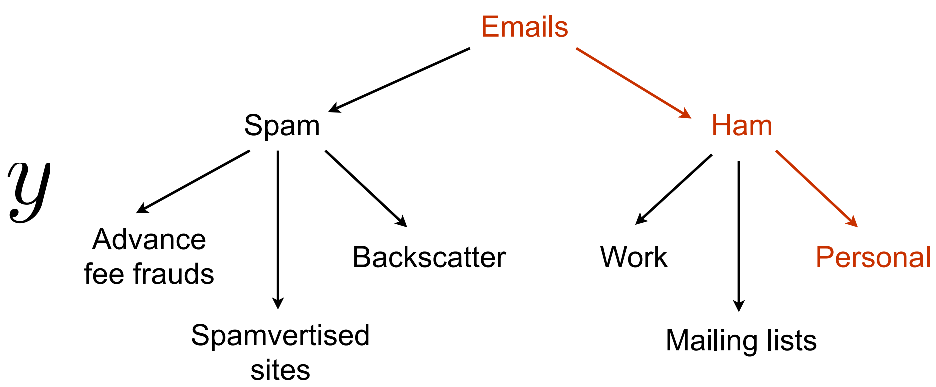
- E.g.: UNIGRAM:Viagra becomes
  - UNIGRAM:Viagra AND CLASS=FRAUD
  - UNIGRAM:Viagra AND CLASS=ADVERTISE
  - UNIGRAM:Viagra AND CLASS=WORK
  - UNIGRAM:Viagra AND CLASS=LIST
  - UNIGRAM:Viagra AND CLASS=PERSONAL

Structure on the output space

Exploit the information in the hierarchy by activating both coarse and fine versions of the features on a given input:

x





...

UNIGRAM:Alex AND CLASS=PERSONAL

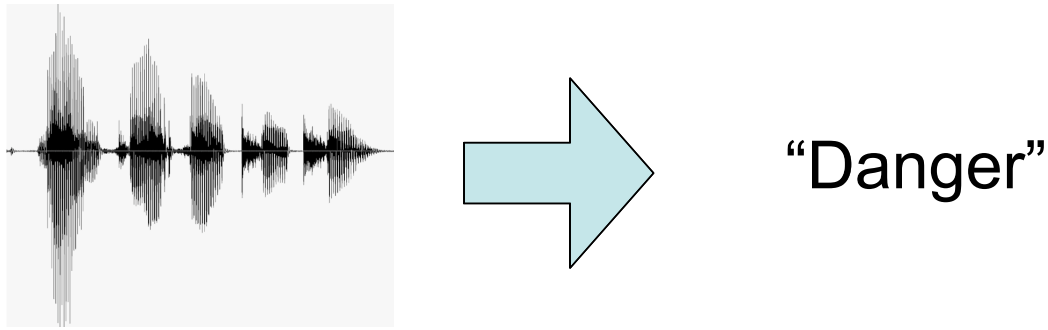
UNIGRAM:Alex AND CLASS=HAM

...

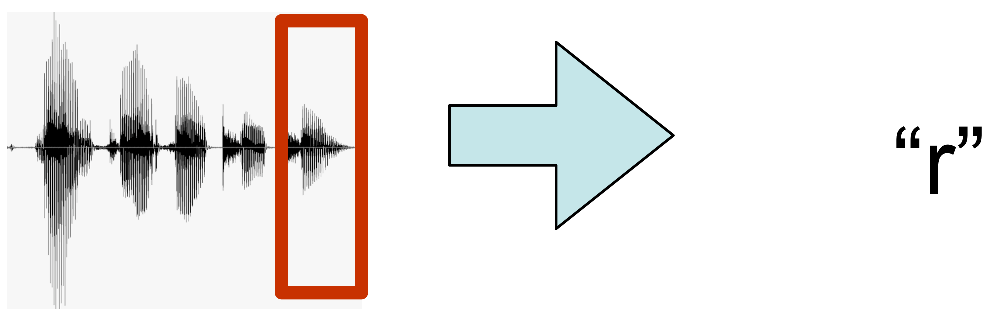
Structure on the output space

- Not limited to hierarchies
  - multiple hierarchies
  - in general, arbitrary featurization of the output
- Another use:
  - want to model that if no words in the email were seen in training, it's probably spam
  - add a *bias* feature that is activated only in SPAM subclass (ignores the input): CLASS=SPAM

Dealing with continuous data



- Full solution needs HMMs (a sequence of correlated classification problems): Alex Simma will talk about that on Oct. 15
- Simpler problem: identify a single sound unit (phoneme)

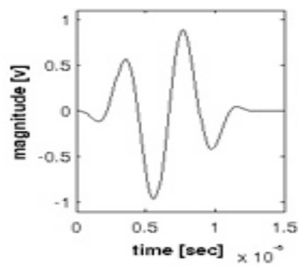


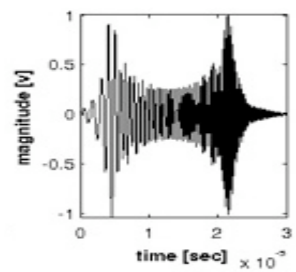
Dealing with continuous data

- Step 1: Find a coordinate system where similar input have similar coordinates
  - Use Fourier transforms and knowledge about the human ear

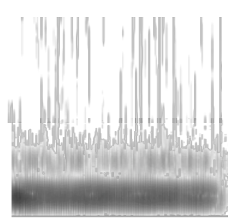
Sound 1 | Sound 2

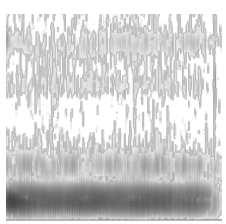
Time domain:





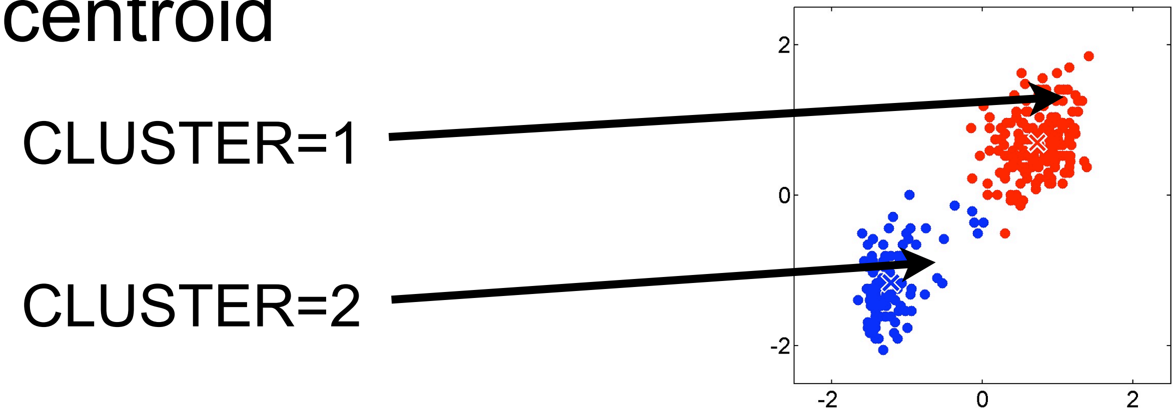
Frequency domain:





Dealing with continuous data

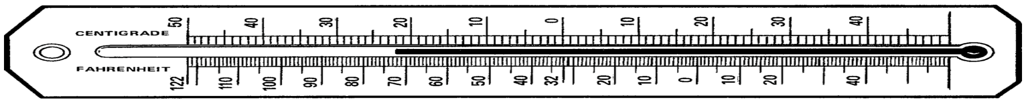
- Step 2 (optional): Transform the continuous data into discrete data
  - Bad idea: COORDINATE=(9.54,8.34)
  - Better: Vector quantization (VQ)
    - Run k-mean on the training data as a preprocessing step
    - Feature is the index of the nearest centroid



Dealing with continuous data

Important special case: integration of the output of a black box

- Back to the email classifier: assume we have an executable that returns, given a email *e*, its belief B(*e*) that the email is spam
- We want to model monotonicity
- Solution: thermometer feature



...	B(e) > 0.4 AND CLASS=SPAM	B(e) > 0.6 AND CLASS=SPAM	B(e) > 0.8 AND CLASS=SPAM	...

Dealing with continuous data

Another way of integrating a qualibrated black box as a feature:

$$f_i(\boldsymbol{x}, y) = \begin{cases} \log B(e) & \text{if } y = \text{SPAM} \\ 0 & \text{otherwise} \end{cases}$$

Recall: votes are combined additively

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

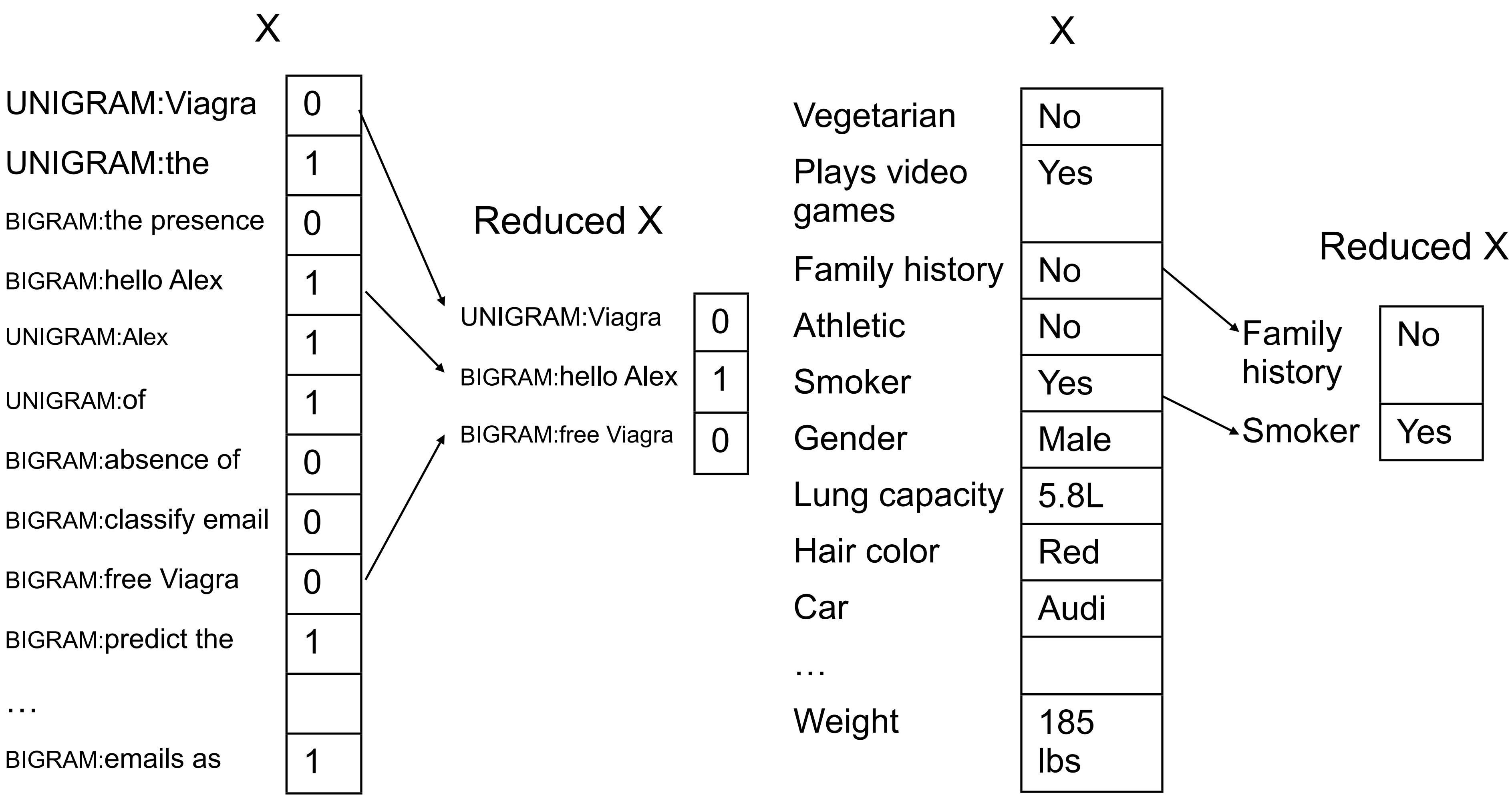
What is feature selection?

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.

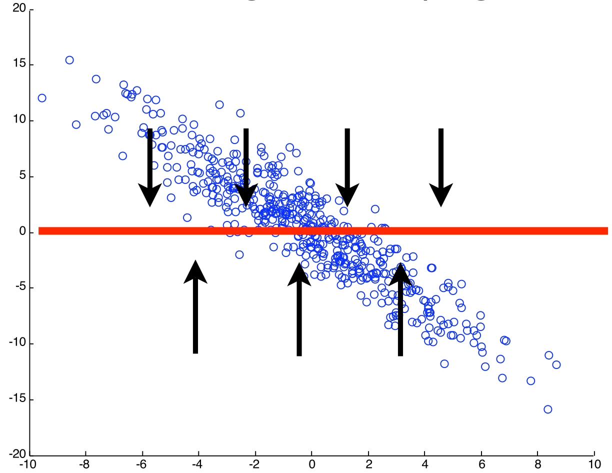
- Common practice: coming up with as many features as possible (e.g. > 10^6 not unusual)
  - Training might be too expensive with all features
  - The presence of irrelevant features hurts generalization.
- Classification of leukemia tumors from microarray gene expression data [Xing, Jordan, Karp '01]
  - 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

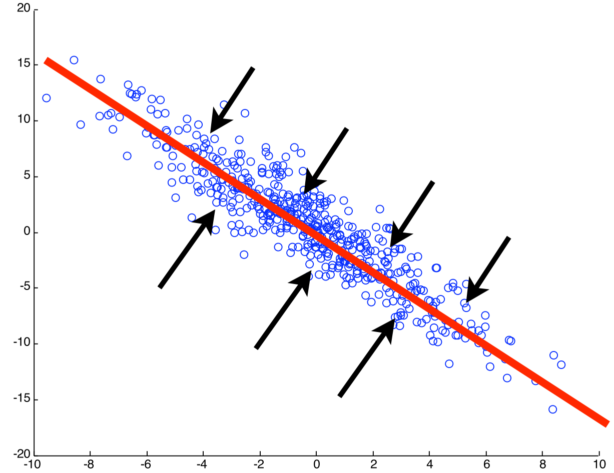
Get at Case 1 through Case 2

- 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

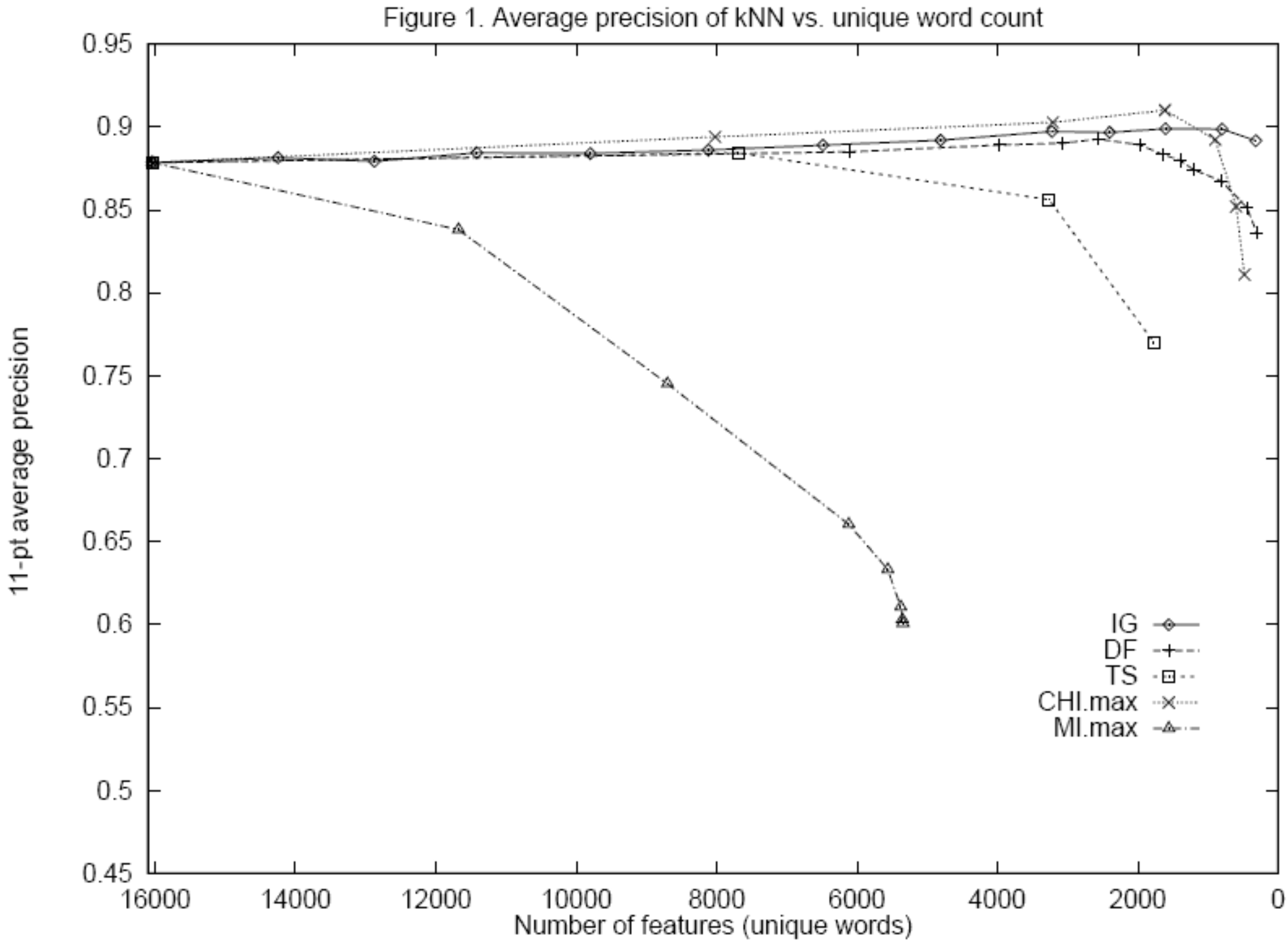
Filtering

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 : $y \in \mathbb{R}$

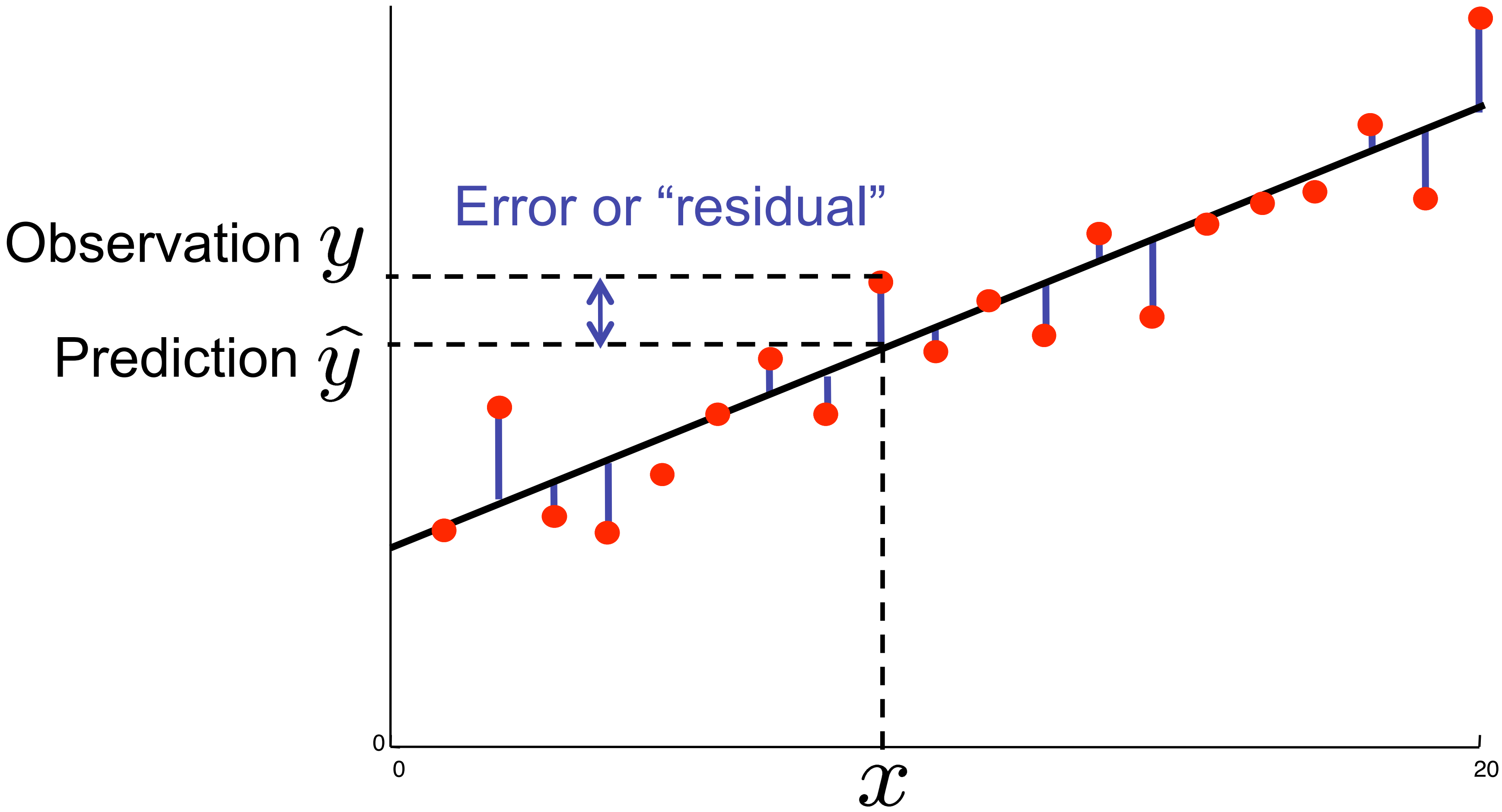
Prediction : $y = \boldsymbol{w}^\top \boldsymbol{x}$

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

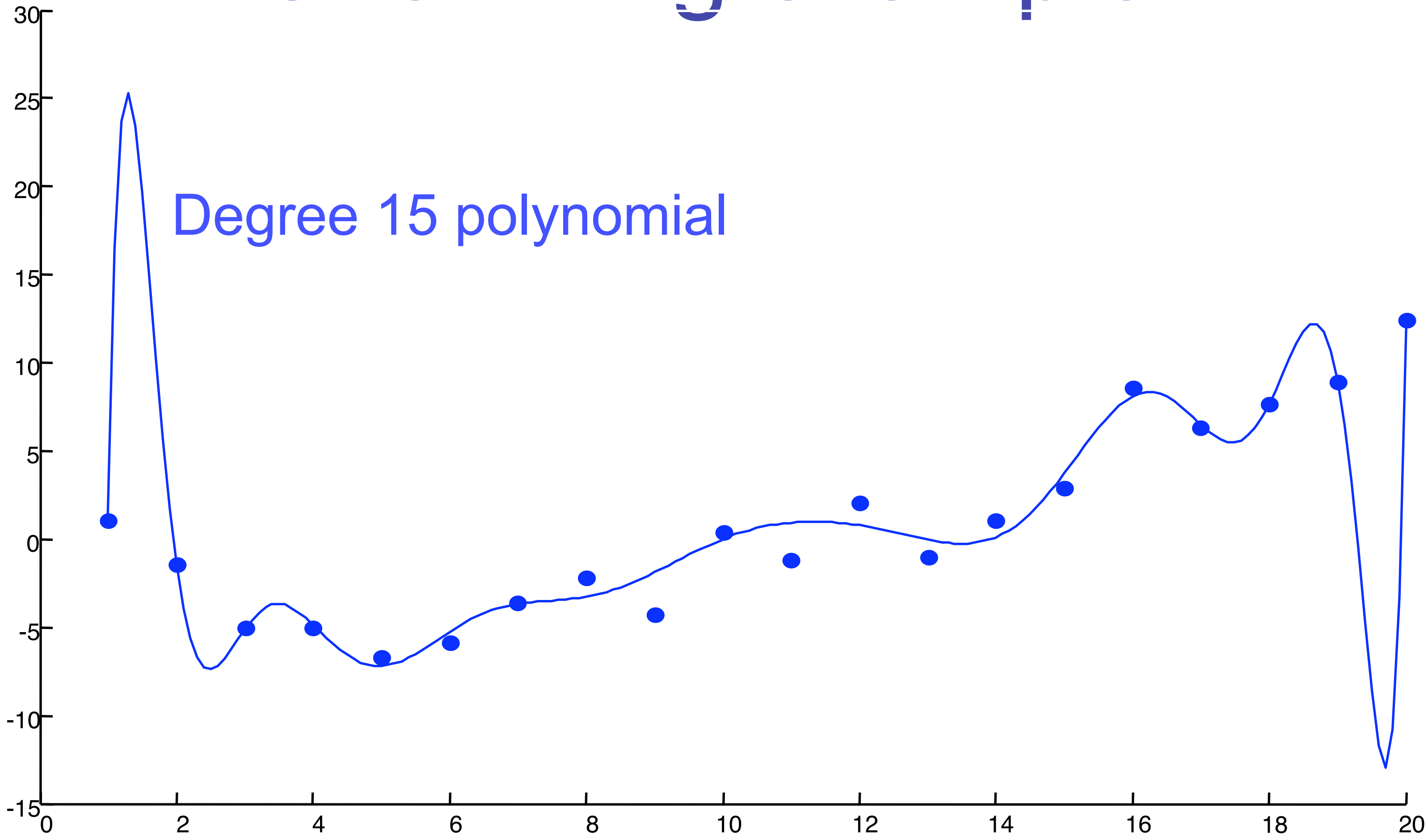
Parameters: $\boldsymbol{w} \in \mathbb{R}^{d+1}$

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

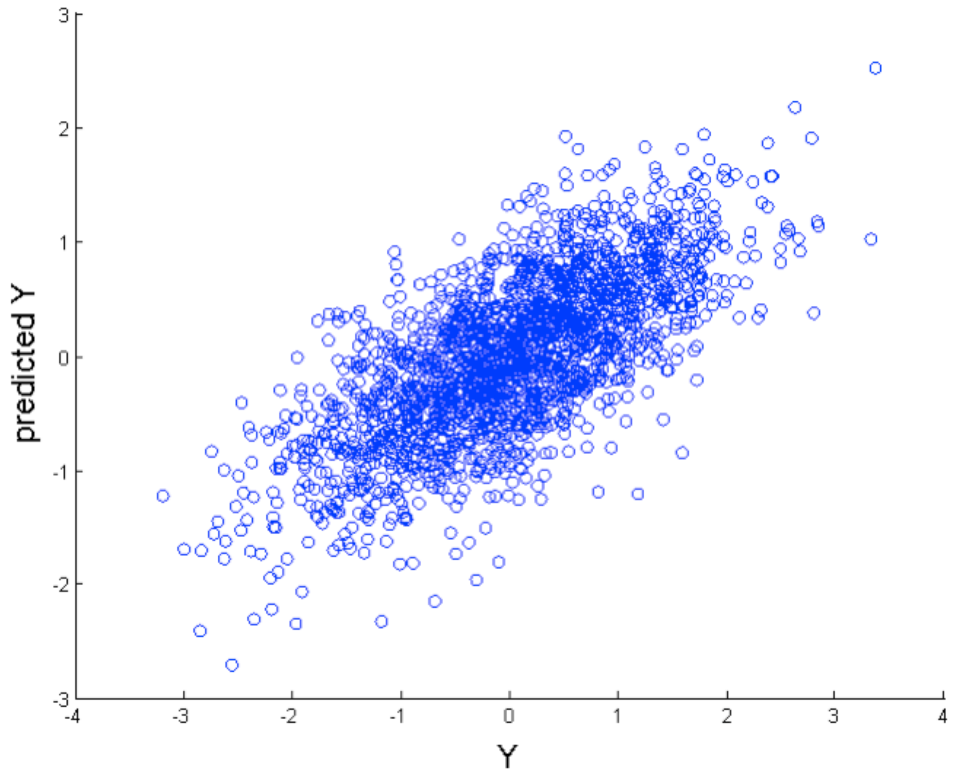


- This model is too rich for the data
- Fits training data well, but doesn't generalize.

(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 $\hat{y}_i = \hat{\boldsymbol{w}}^\top \boldsymbol{x}_i$



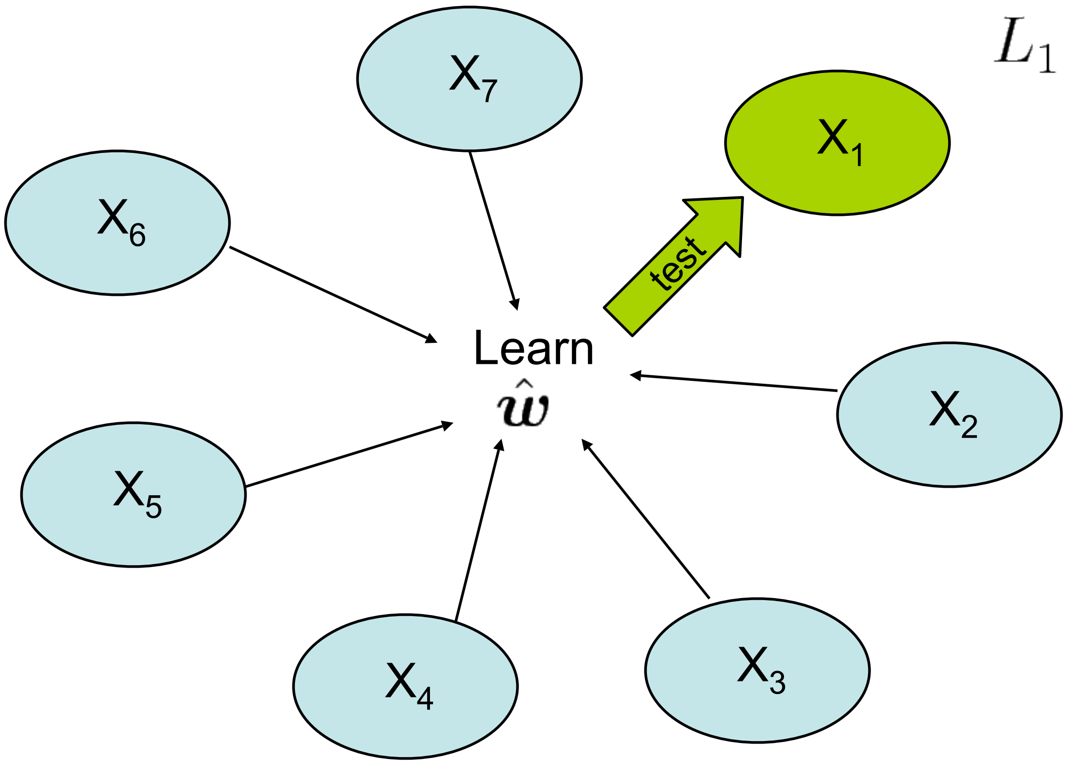
It really looks like we've found a relationship between $\boldsymbol{x}$ and y ! But no such relationship exists, so $\hat{\boldsymbol{w}}$ will do no better than random on new data.

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

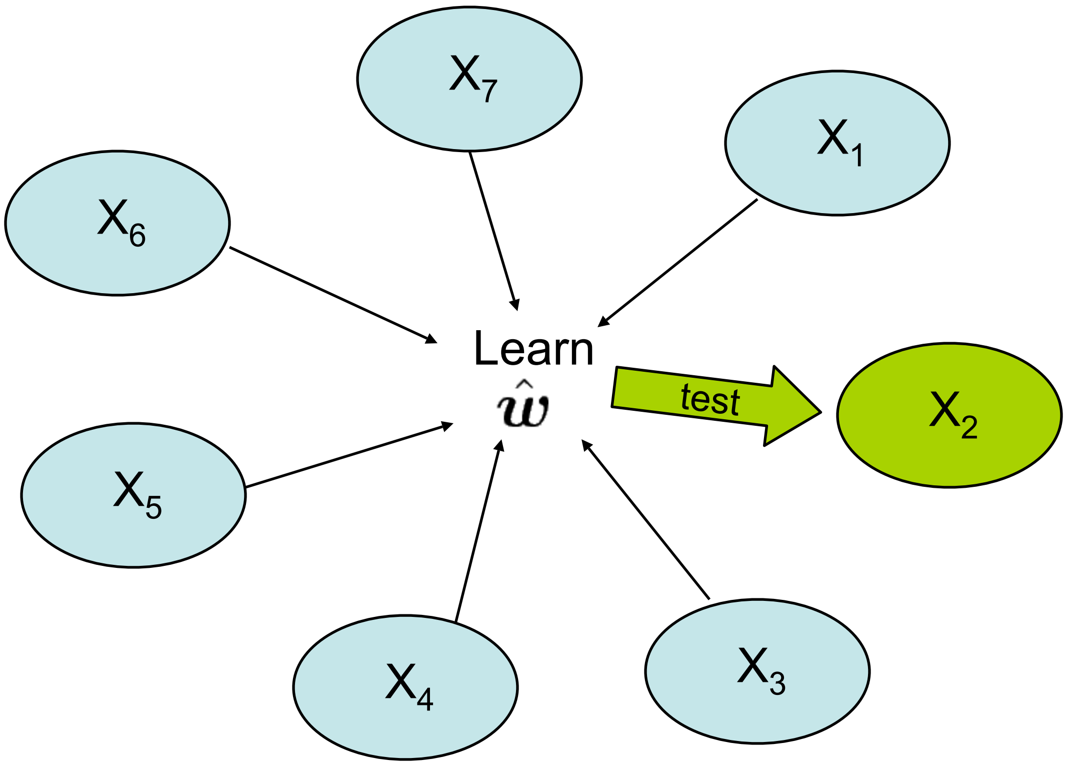
- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors



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

K-fold cross validation

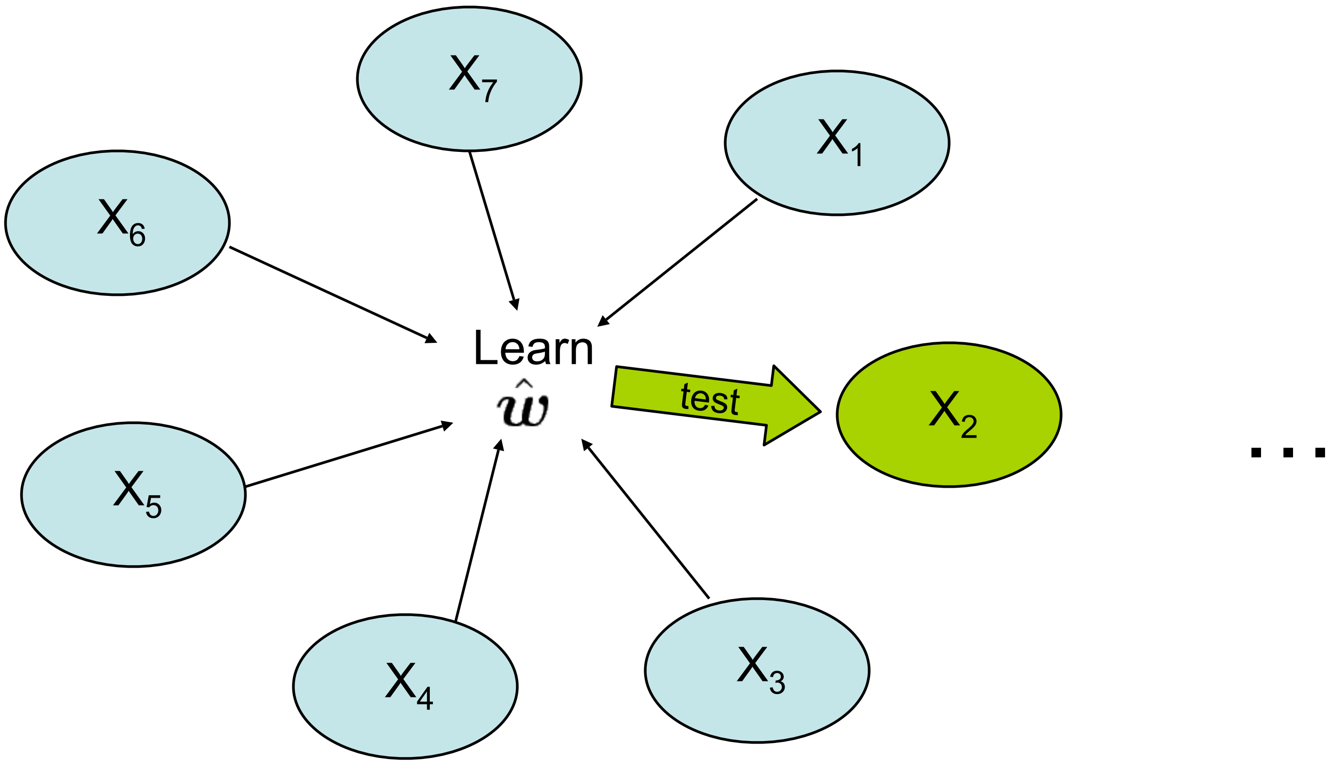
- A technique for estimating test error
- Uses *all* of the data to validate
- Divide data into K groups $\{X_1, X_2, \ldots, X_K\}$.
- Use each group as a validation set, then average all validation errors



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

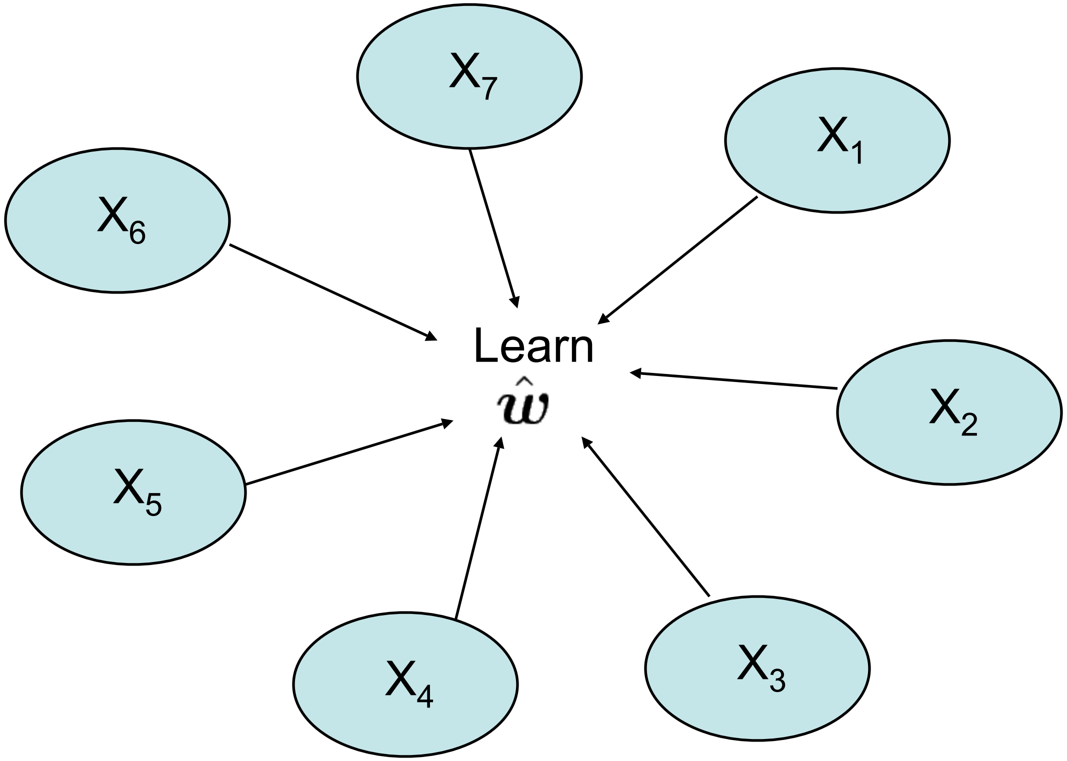
K-fold cross validation

- 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)

Outline

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

Regularization

- In certain cases, we can move model selection *into* the induction algorithm

- This is sometimes called an *embedded* feature selection algorithm

Regularization

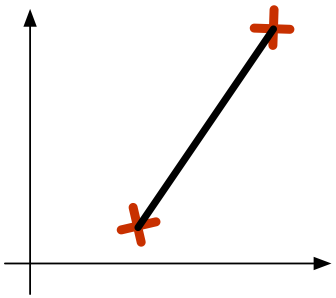
- Regularization: add model complexity penalty to training error.
- $$J(\boldsymbol{w}) = L(\boldsymbol{w}) + C\|\boldsymbol{w}\|_p = \sum_{i=1}^{n}(y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + C\|\boldsymbol{w}\|_p$$

  for some constant C
- Find $\hat{\boldsymbol{w}} = \operatorname{argmin}_w J(w)$
- Regularization forces weights to be small, but does it force weights to be exactly *zero*?
  - $w_f = 0$ is equivalent to removing feature f from the model
- Depends on the value of *p* …

p metrics and norms

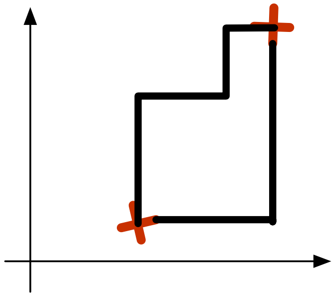
- p = 2: Euclidean

$$||\vec{w}||_2 = \sqrt{w_1^2 + \cdots + w_n^2}$$



- p = 1: Taxicab or Manhattan

$$||\vec{w}||_1 = |w_1| + \cdots + |w_n|$$

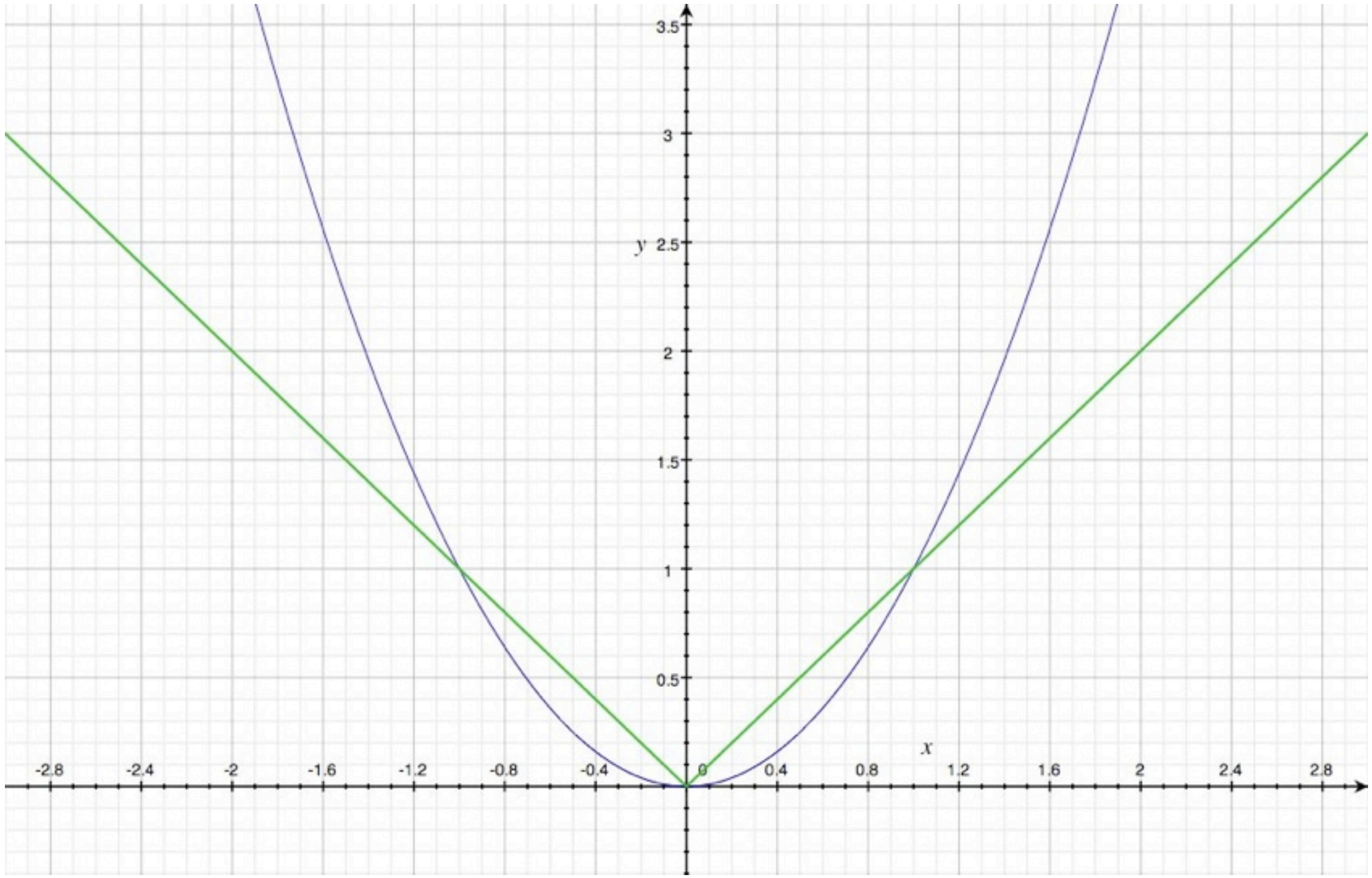


- General case: $0 < p \leq \infty$

$$||\vec{w}||_p = \sqrt[p]{|w_1|^p + \cdots + |w_n|^p}$$

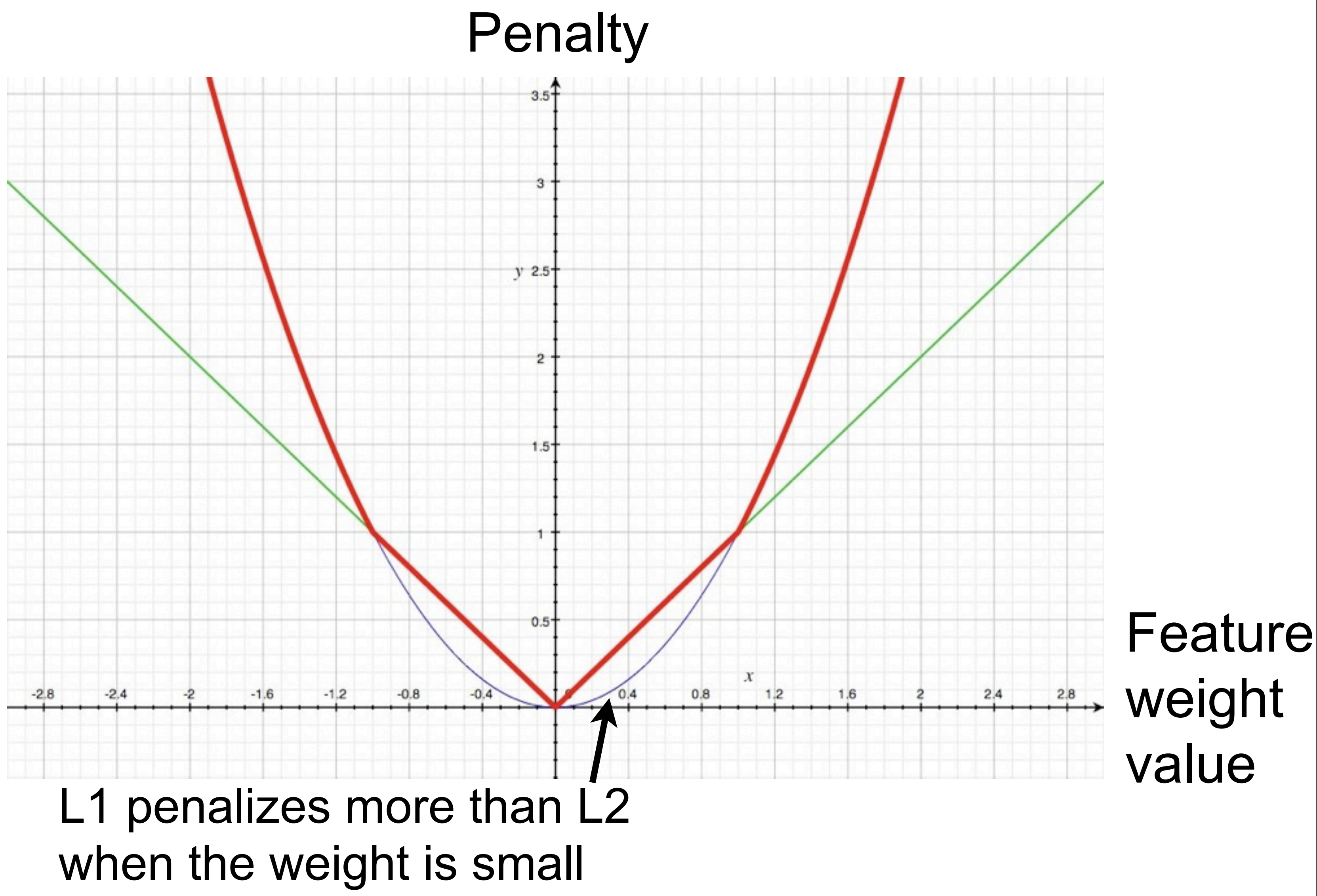
Univariate case: intuition

Penalty



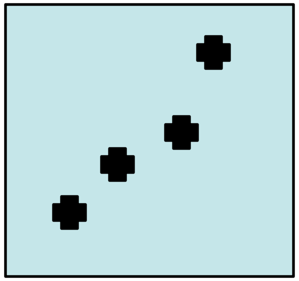
Feature weight value

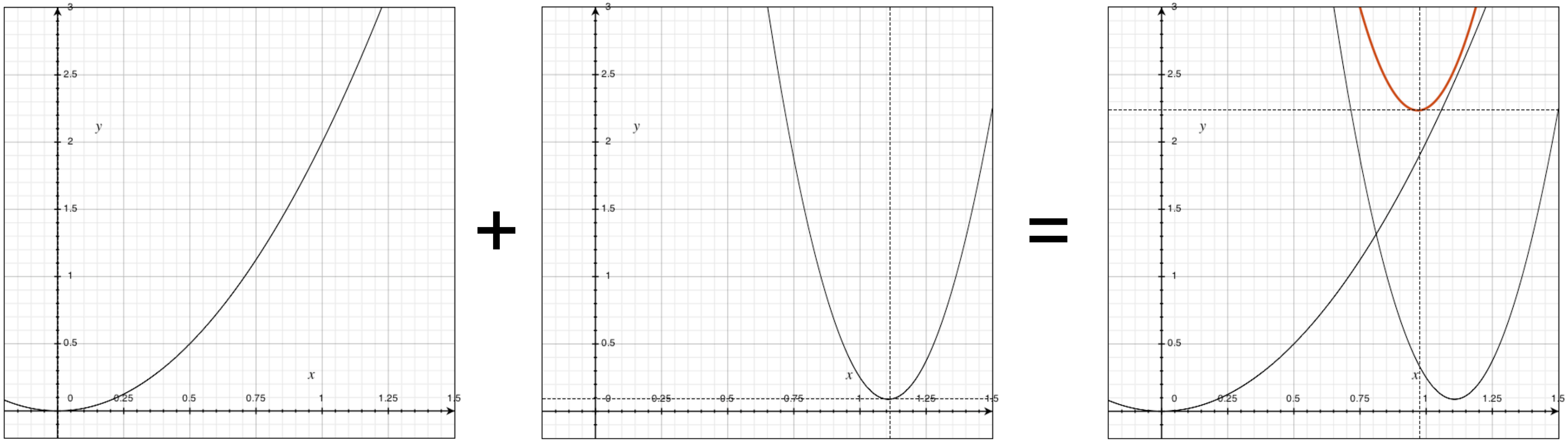
Univariate case: intuition



Univariate example: L_2

- Case 1: there is a lot of data supporting our hypothesis





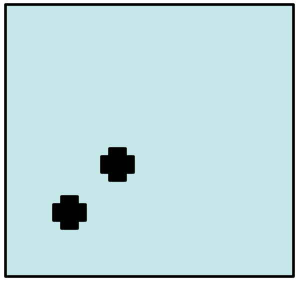
Regularization term

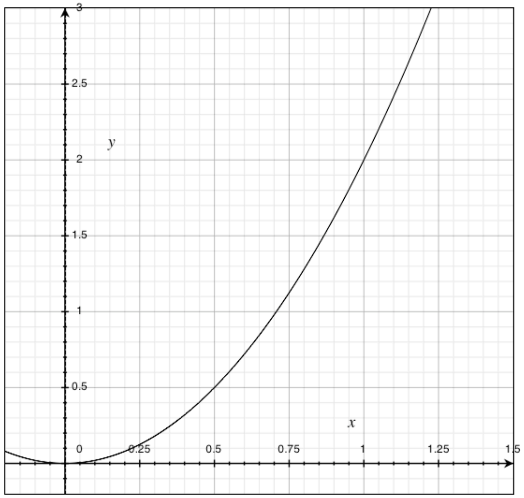
Data likelihood
By itself, minimized by w=1.1

Objective function
Minimized by w=0.95

Univariate example: L_2

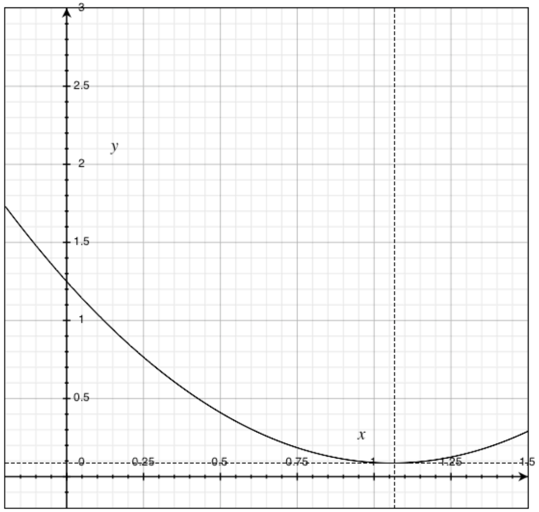
- Case 2: there is NOT a lot of data supporting our hypothesis





Regularization term

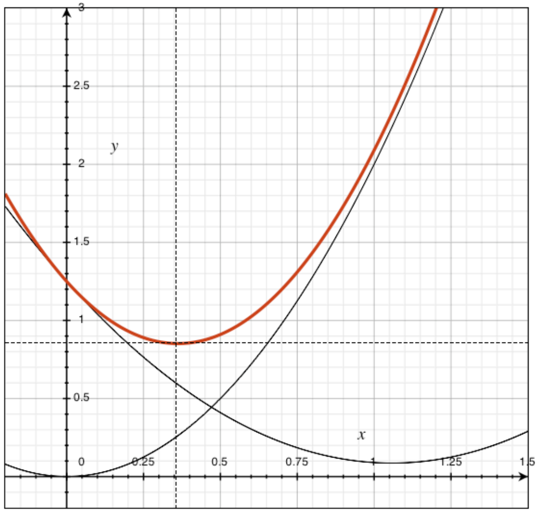
+



Data likelihood

By itself, minimized by w=1.1

=

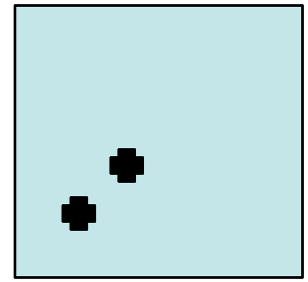


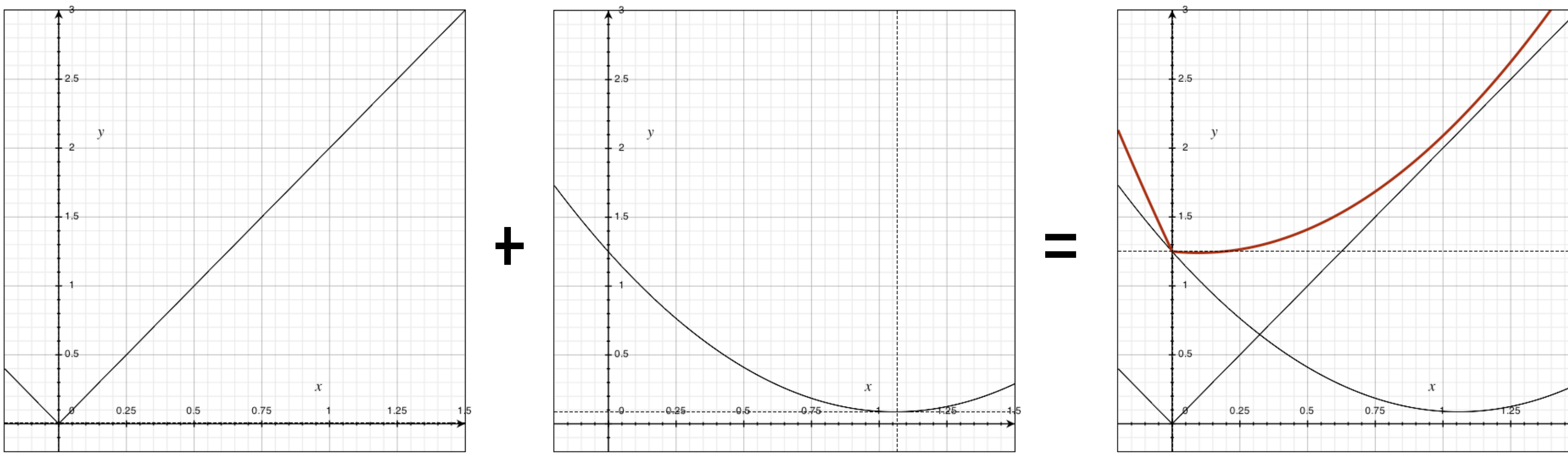
Objective function

Minimized by w=0.36

Univariate example: L_1

- Case 1, when there is a lot of data supporting our hypothesis:
  - Almost the same resulting w as L2
- Case 2, when there is NOT a lot of data supporting our hypothesis
- Get w = exactly zero





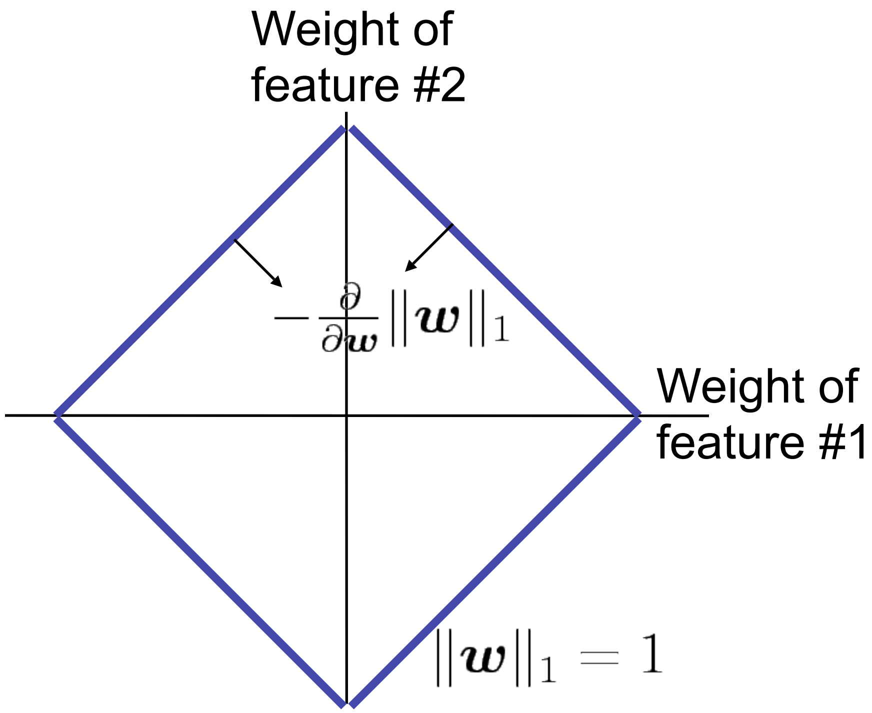
Regularization term

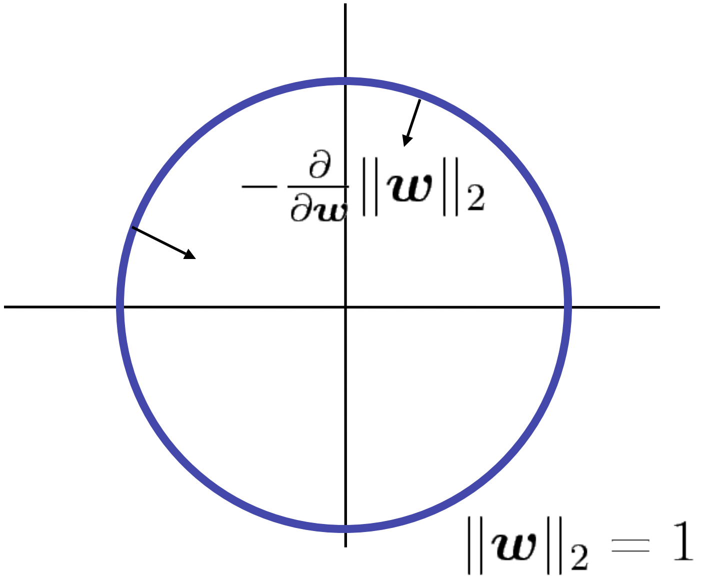
Data likelihood
By itself, minimized by w=1.1

Objective function
Minimized by w=0.0

Level sets of L_1 vs L_2 (in 2D)

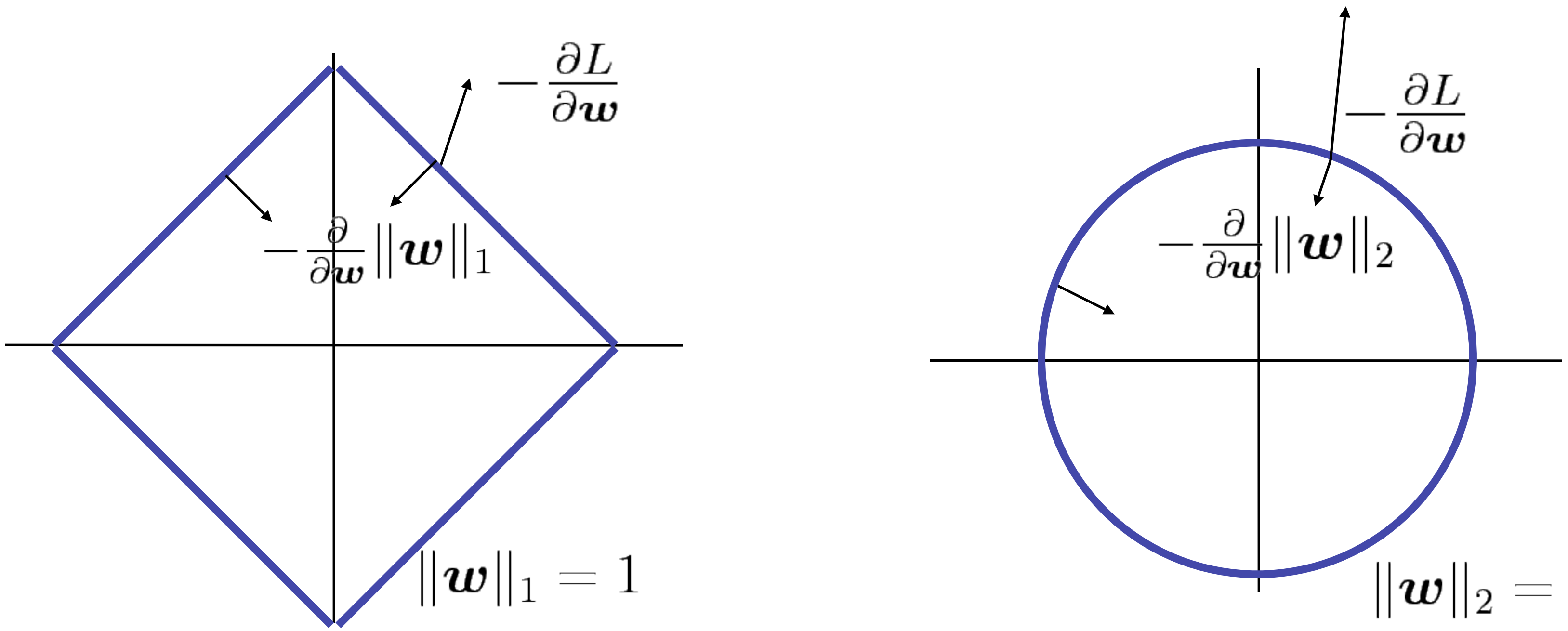
$$||\boldsymbol{w}||_1 = \sum_{f=0}^{d} |w_f| \qquad ||\boldsymbol{w}||_2 = \sqrt{\sum_{f=0}^{d} w_f^2}$$





Multivariate case: w gets cornered

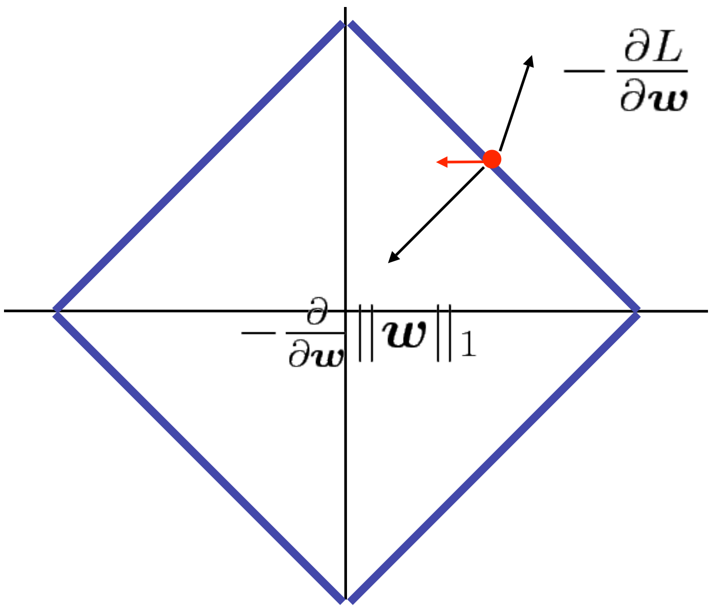
- To minimize $J(\boldsymbol{w}) = L(\boldsymbol{w}) + \|\boldsymbol{w}\|_p$, we can solve $\frac{\partial J}{\partial \boldsymbol{w}} = 0$ by (e.g.) gradient descent.



- Minimization is a tug-of-war between the two terms

Multivariate case: w gets cornered

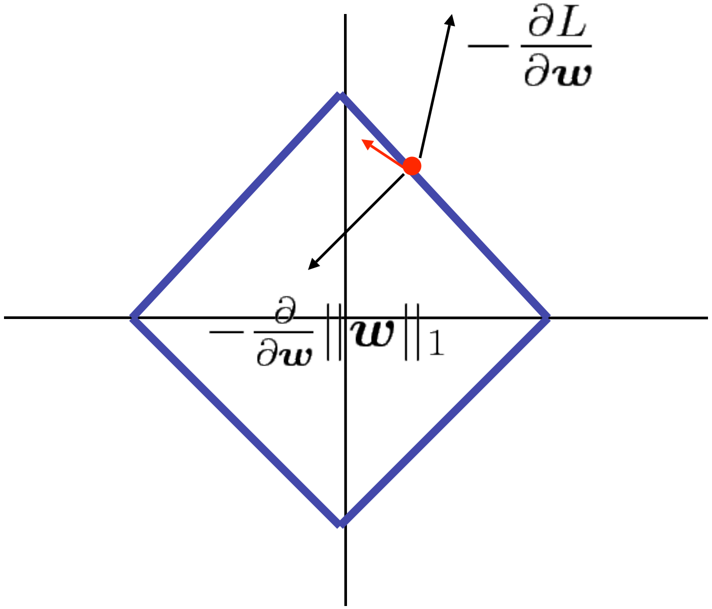
- To minimize $J(\boldsymbol{w}) = L(\boldsymbol{w}) + \|\boldsymbol{w}\|_p$, we can solve $\frac{\partial J}{\partial \boldsymbol{w}} = 0$ by (e.g.) gradient descent.



- Minimization is a tug-of-war between the two terms

Multivariate case: w gets cornered

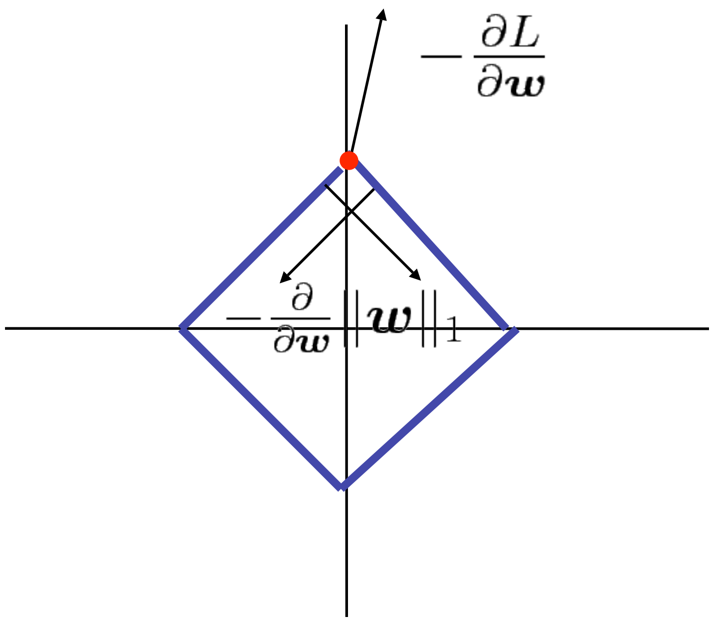
- To minimize $J(\boldsymbol{w}) = L(\boldsymbol{w}) + \|\boldsymbol{w}\|_p$, we can solve $\frac{\partial J}{\partial \boldsymbol{w}} = 0$ by (e.g.) gradient descent.



- Minimization is a tug-of-war between the two terms

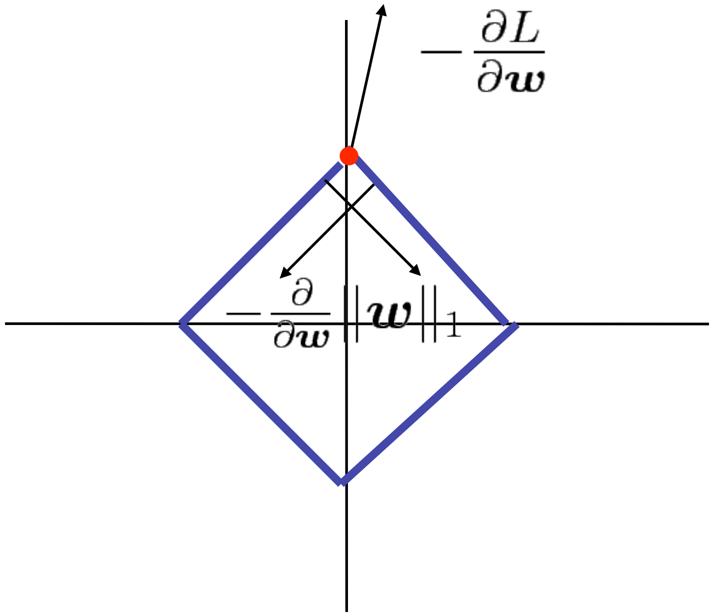
Multivariate case: w gets cornered

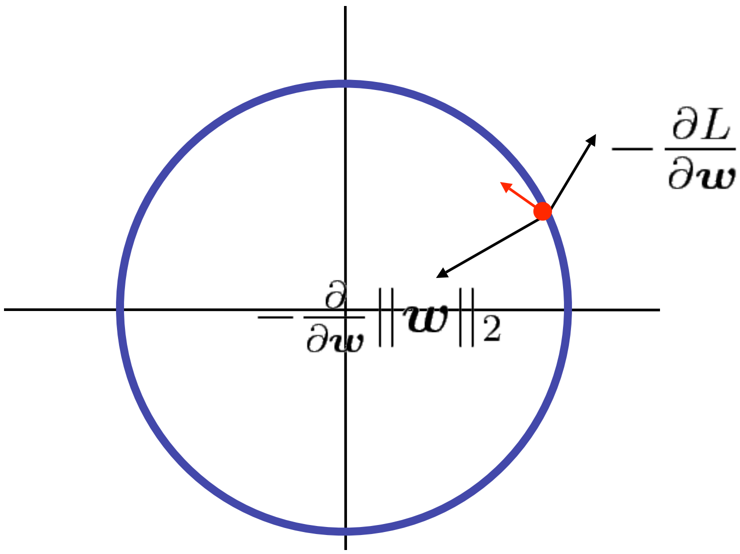
- To minimize $J(\boldsymbol{w}) = L(\boldsymbol{w}) + \|\boldsymbol{w}\|_p$, we can solve $\frac{\partial J}{\partial \boldsymbol{w}} = 0$ by (e.g.) gradient descent.



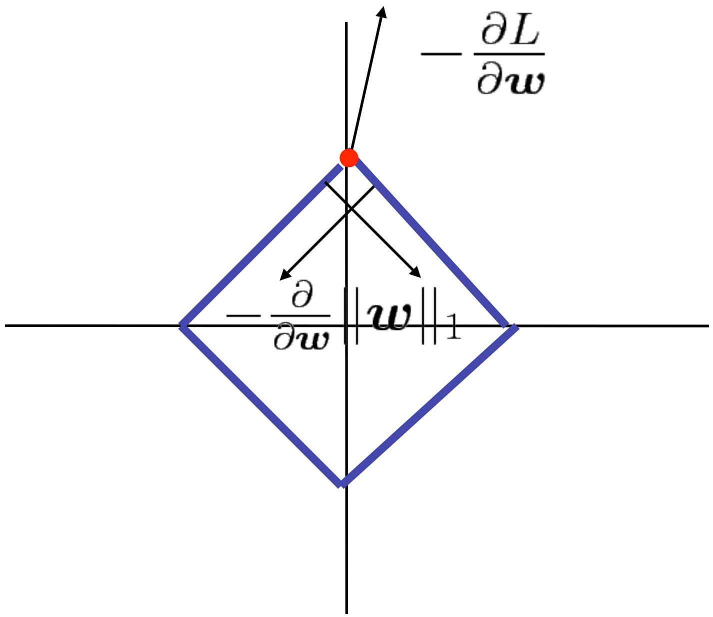
- Minimization is a tug-of-war between the two terms
- w is forced into the corners—components are zeroed
  - Solution is often *sparse*

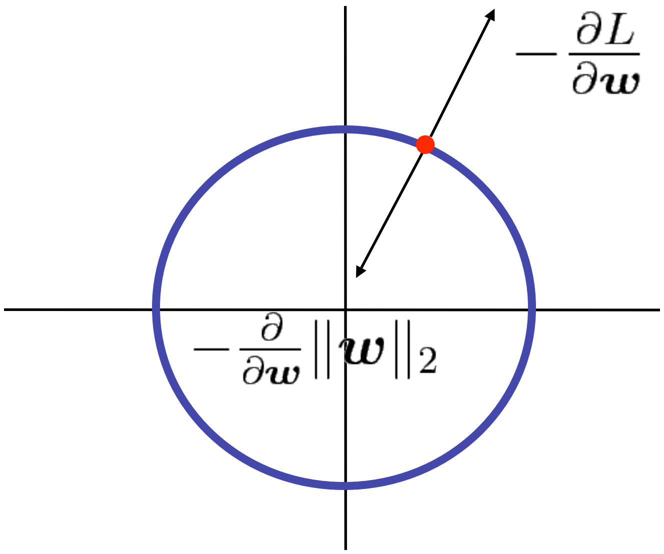
L_2 does not zero components





L_2 does not zero components





- L_2 regularization does not promote sparsity
- Even without sparsity, regularization promotes generalization—limits expressiveness of model

Lasso Regression [Tibshirani '94]

- Simply linear regression with an L_1 penalty for sparsity.

$$\hat{\boldsymbol{w}} = \mathrm{argmin}_w \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + C||\boldsymbol{w}||_1$$

- Compare with ridge regression (introduced by Fabian 3 weeks ago):

$$\hat{\boldsymbol{w}} = \mathrm{argmin}_w \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + C||\boldsymbol{w}||_2^2$$

Lasso Regression [Tibshirani '94]

- Simply linear regression with an L_1 penalty for sparsity.

$$\hat{\boldsymbol{w}} = \text{argmin}_w \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + C||\boldsymbol{w}||_1$$

- Two questions:
  – 1. How do we perform this minimization?
    - Difficulty: not differentiable everywhere
  – 2. How do we choose C?
    - Determines how much sparsity will be obtained
    - C is called an hyperparameter

Question 1: Optimization/learning

- Set of discontinuity has Lebesgue measure zero, but optimizer WILL hit them
- Several approaches, including:
  - Projected gradient, stochastic projected subgradient, coordinate descent, interior point, orthan-wise L-BFGS [Friedman 07, Andrew et. al. 07, Koh et al. 07, Kim et al. 07, Duchi 08]
  - More on that on the John's lecture on optimization
  - Open source implementation: edu.berkeley.nlp.math.OW_LBFGSMinimizer in http://code.google.com/p/berkeleyparser/

Question 2: Choosing C

- Up until a few years ago this was not trivial
  - Fitting model: optimization problem, harder than least-squares
  - Cross validation to choose *C*: must fit model for every candidate *C* value
- Not with LARS! (Least Angle Regression, Hastie et al, 2004)
  - Find trajectory of w for all possible C values simultaneously, as efficiently as least-squares
  - Can choose exactly how many features are wanted

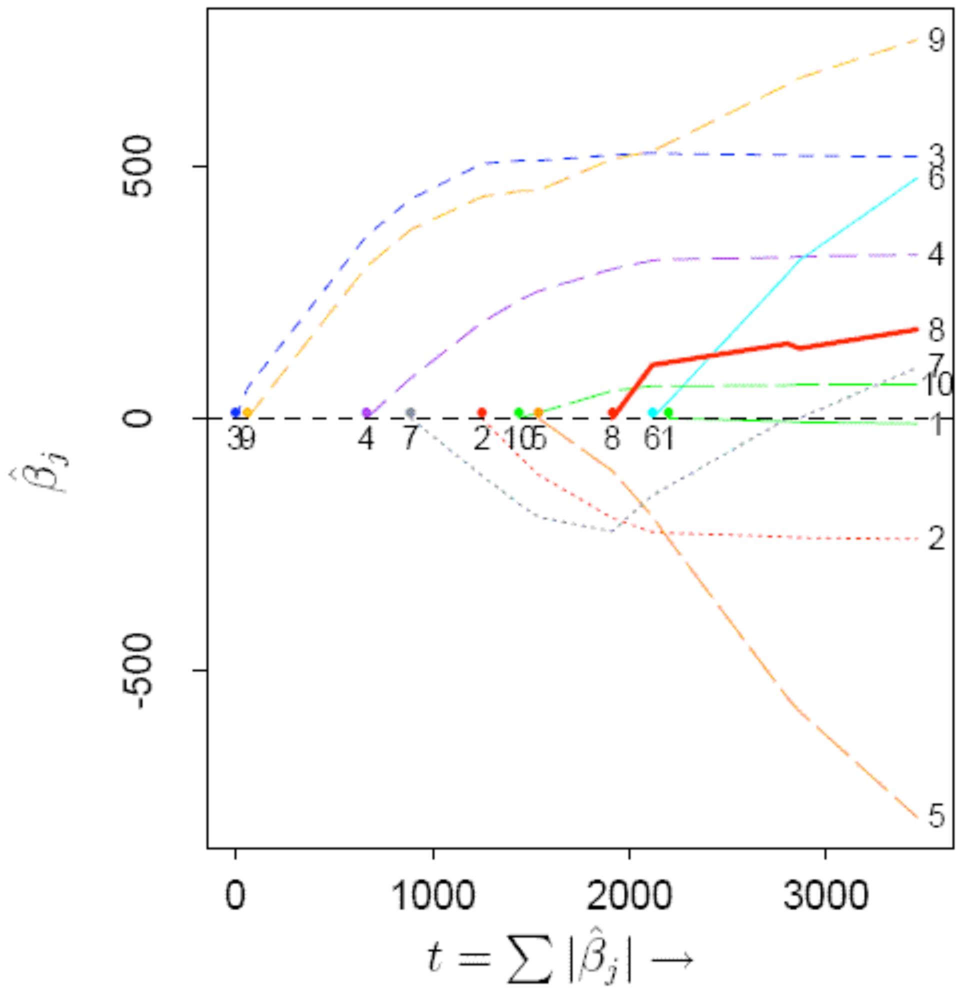


Figure taken from Hastie et al (2004)

Remarks

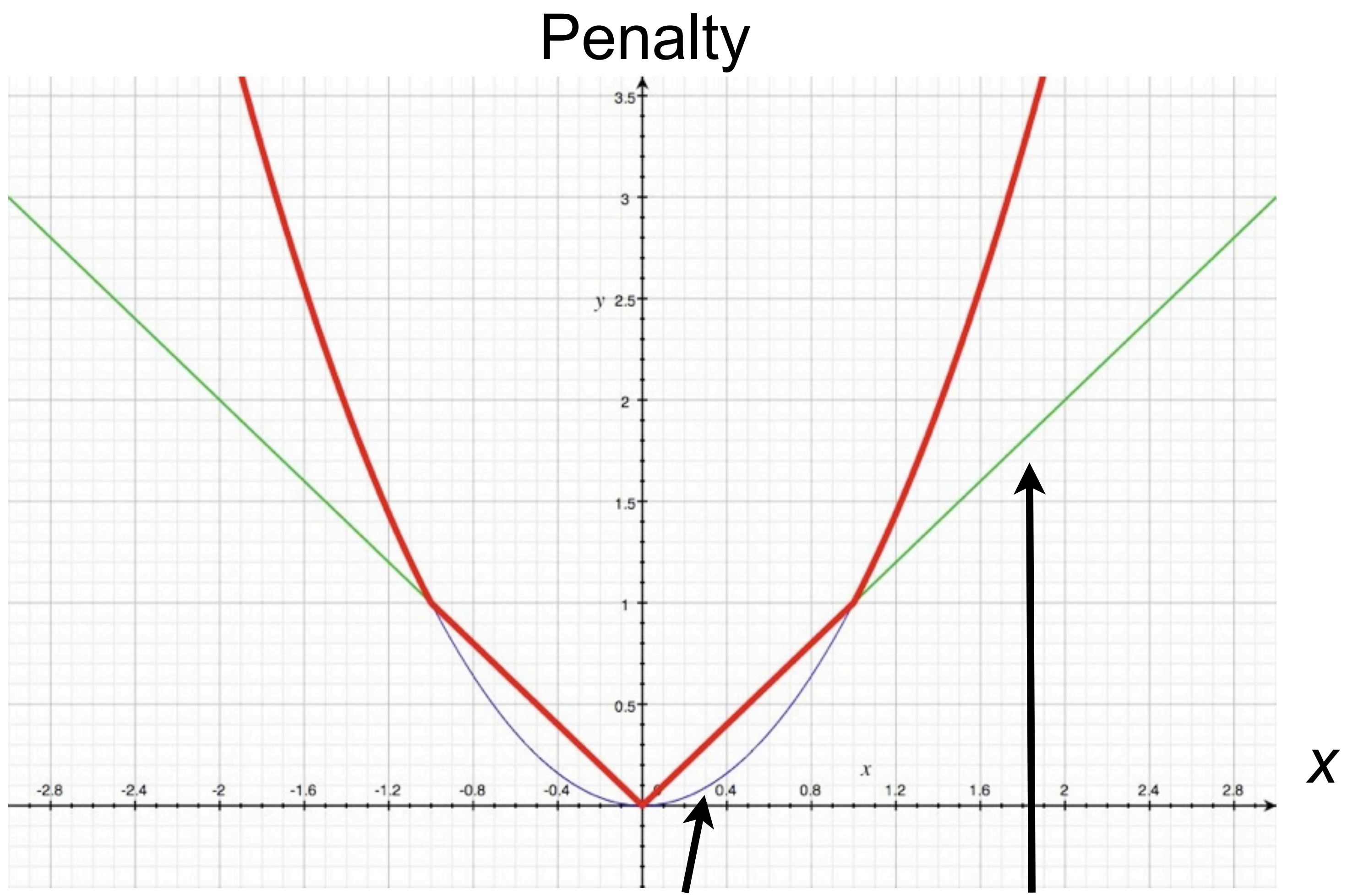
- Not to be confused: two othogonal uses of L1 for regression:
  - lasso for sparsity: what we just described

$$\hat{\boldsymbol{w}} = \text{argmin}_{\boldsymbol{w}} \sum_{i=1}^{n} (y_i - \boldsymbol{w}^\top \boldsymbol{x}_i)^2 + \boxed{C \sum_{f=1}^{d} |\boldsymbol{w}_f|}$$

  - L1 loss: for robustness (Fabian's lecture).

$$\hat{w} = \text{argmin}_w \boxed{\sum_{i=1}^{n} \left| y_i - w^T x_i \right|} + C||w||_p$$

Intuition



L1 penalizes more than L2 when *x* is small (use this for sparsity)

L1 penalizes less than L2 when *x* is big (use this for robustness)

Remarks

- L1 penalty can be viewed as a laplace prior on the weights, just as L2 penalty can viewed as a normal prior
  - Side note: also possible to learn *C* efficiently when the penalty is L2 (Foo, Do, Ng, ICML 09, NIPS 07)
- Not limited to regression: can be applied to classification, for example

L_1 Vs L_2 [Gao et al '07]

- For large scale problems, performance of L1 and L2 is very similar (at least in NLP)
  - A slight advantage of L2 over L1 in accuracy
  - But solution is 2 orders of magnitudes sparser!
  - Parsing reranking task:

(Higher F1 is better)

	F-Score	# features	time (min)	# train iter
Baseline	0.8986			
ME/L2	0.9176	1,211,026	62	129
ME/L1	0.9165	19,121	37	174
AP	0.9164	939,248	2	8
Boosting	0.9131	6,714	495	92,600
BLasso	0.9133	8,085	239	56,500

When can feature selection hurt?

- NLP example: back to the email classification task
- Zipf law: frequency of a word is inversely proportional to its frequency rank.
  - Fat tail: many n-grams are seen only once in the training
  - Yet they can be very useful predictors
  - E.g. 8-gram “today I give a lecture on feature selection” occurs only once in my mailbox, but it’s a good predictor that the email is WORK

Outline

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

Summary: feature engineering

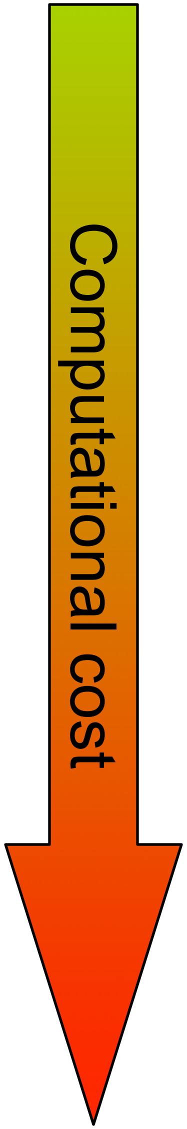
- Feature engineering is often crucial to get good results
- Strategy: overshoot and regularize
  - Come up with lots of features: better to include irrelevant features than to miss important features
  - Use regularization or feature selection to prevent overfitting
  - Evaluate your feature engineering 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?

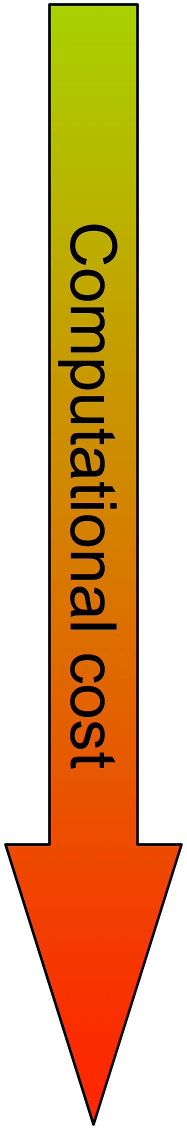
- If the only concern is accuracy, and the whole dataset can be processed, feature selection not needed (as long as there is regularization)
- If computational complexity is critical (embedded device, web-scale data, fancy learning algorithm), consider using feature selection
  - But there are alternatives: 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

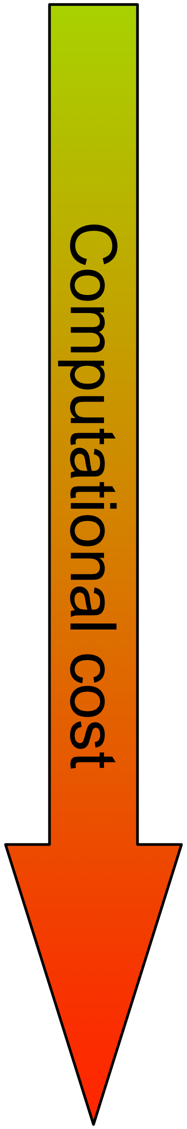
Summary: how to do feature selection



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

- Good preprocessing step
- Fails to capture relationship between features

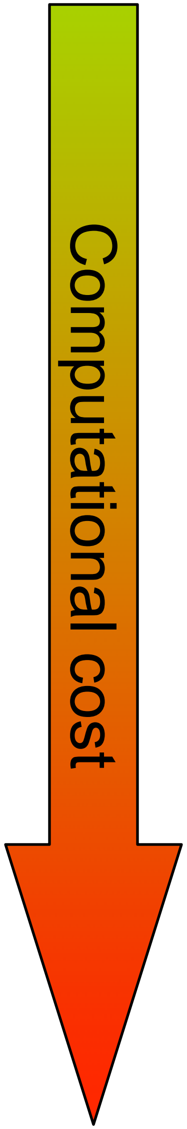
Summary: how to do feature selection



- 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.

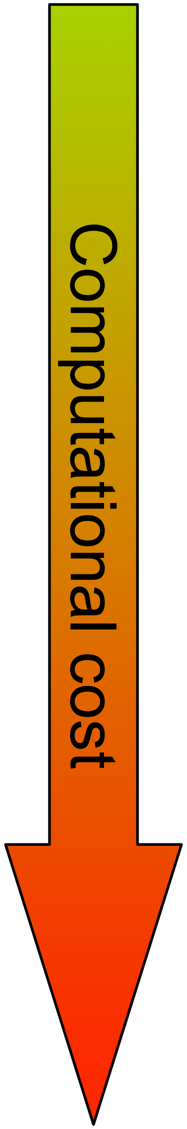
Summary: how to do feature selection



- 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

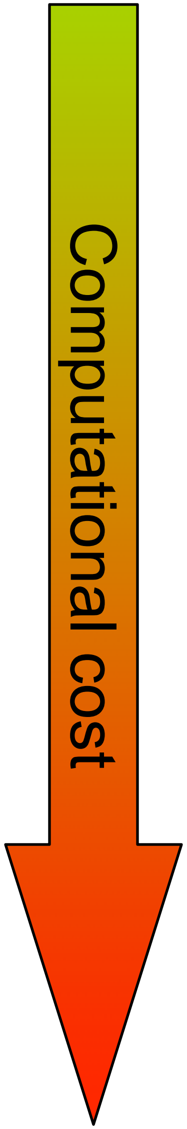
Summary: how to do feature selection



- 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

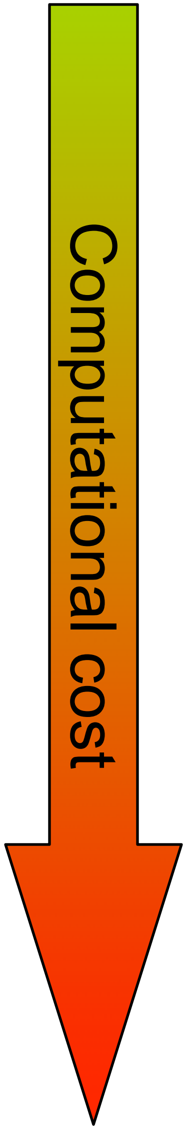
Summary: how to do feature selection



- 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

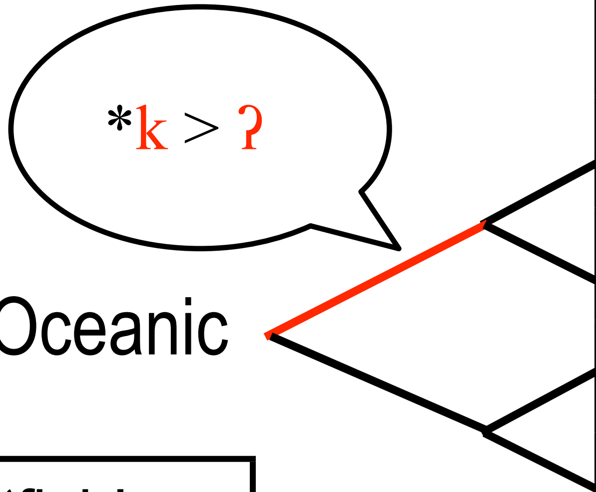
Extra slides

Feature engineering case study: Modeling language change [Bouchard et al. 07,09]

	'fish'	'fear'
Hawaiian	iʔa	makaʔu
Samoan	iʔa	mataʔu
Tongan	ika	
Maori	ika	mataku

Feature engineering case study: Modeling language change [Bouchard et al. 07,09]

Proto-Oceanic



	'fish'	'fear'
Hawaiian	iʔa	makaʔu
Samoan	iʔa	mataʔu
Tongan	ika	
Maori	ika	mataku

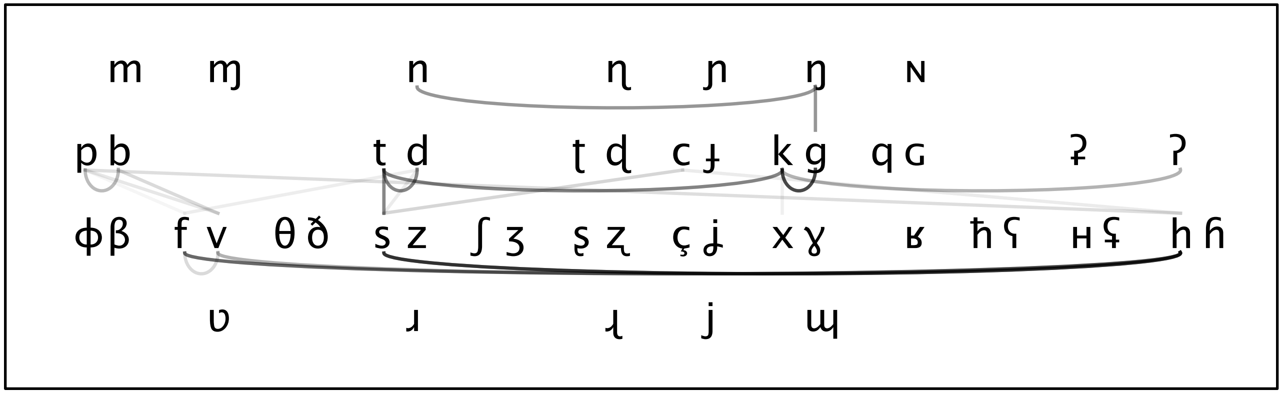
	'fish'
POc	*ika

Tasks:

- Proto-word reconstruction
- Infer sound changes

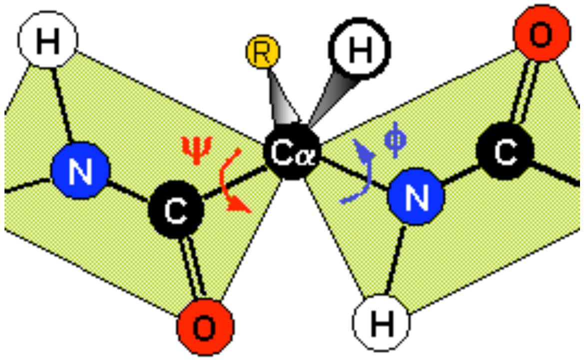
Feature engineering case study: Modeling language change [Bouchard et al. 07,09]

- Featurize sound changes
  - E.g.: substitution are generally more frequent than insertions, deletions, changes are branch specific, but there are cross-linguistic universal, etc.
- Particularity: unsupervised learning setup
  - We covered feature engineering for supervised setups for pedagogical reasons; most of what we have seen applies to the unsupervised setup



Feature selection case study: Protein Energy Prediction [Blum et al '07]

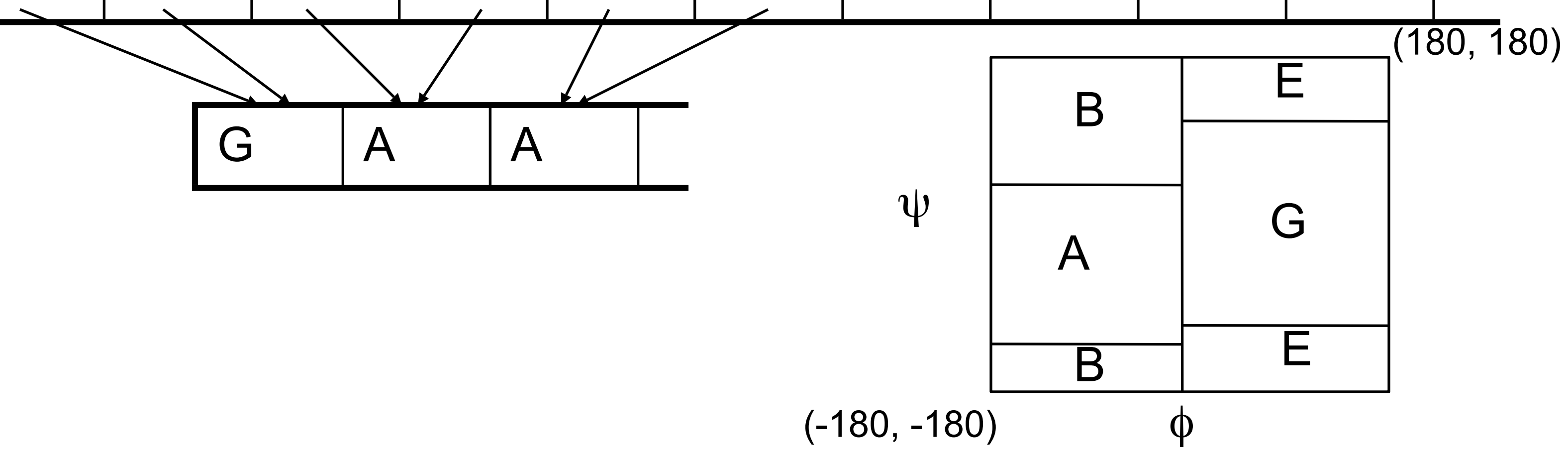
- What is a protein?
  - A protein is a chain of amino acids.
- Proteins fold into a 3D conformation by minimizing energy
  - "Native" conformation (the one found in nature) is the lowest energy state
  - We would like to find it using only computer search.
  - Very hard, need to try several initialization in parallel
- Regression problem:
  - Input: many different conformation of the same sequence
  - Output: energy
- Features derived from: ϕ and ψ *torsion angles*.
- Restrict next wave of search to agree with features that predicted high energy



Featurization

- Torsion angle features can be binned

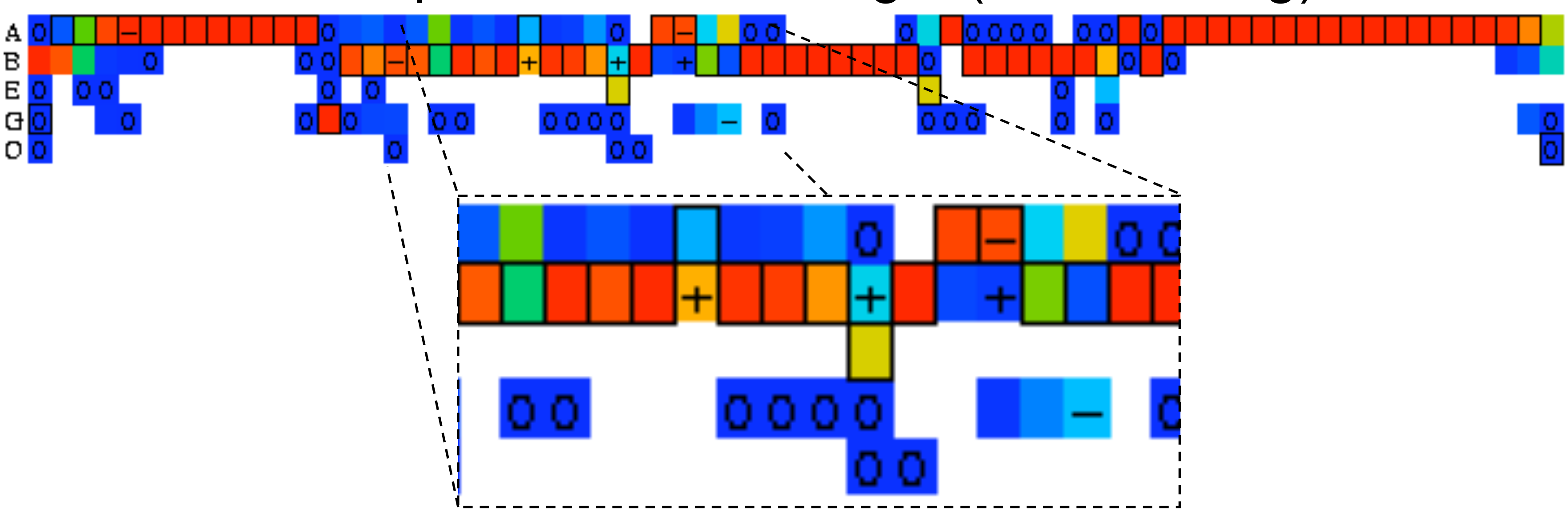
ϕ_1	ψ_1	ϕ_2	ψ_2	ϕ_3	ψ_4	ϕ_5	ψ_5	ϕ_6	ψ_6
0	75.3	-61.6	-24.8	-68.6	-51.9	-63.3	-37.6	-62.8	-42.3



- Bins in the *Ramachandran* plot correspond to common structural elements
  - Secondary structure: alpha helices and beta sheets

Results of LARS for predicting protein energy

- One column for each torsion angle feature
- Colors indicate frequencies in data set
  - Red is high, blue is low, 0 is very low, white is never
  - Framed boxes are the correct native features
  - "-" indicates negative LARS weight (stabilizing), "+" indicates positive LARS weight (destabilizing)



Other things to check out

- Bayesian methods
  - David MacKay: Automatic Relevance Determination
    - originally for neural networks
  - Mike Tipping: Relevance Vector Machines
    - http://research.microsoft.com/mlp/rvm/
- Miscellaneous feature selection algorithms
  - Winnow
    - Linear classification, provably converges in the presence of exponentially many irrelevant features
  - Optimal Brain Damage
    - Simplifying neural network structure
- Case studies
  - See papers linked on course webpage.

Acknowledgments

- Useful comments by Mike Jordan, Percy Liang
- A first version of these slides was created by Ben Blum