

Feature Selection and Related Topics

CS 294: Practical Machine
Learning Lecture 4

February 26th, 2006

Lecturer: Alexandre Bouchard,
Using slides from Ben Blum

Review

- Data: pairs $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$
- \mathbf{x}_i : vector of features $[x_{i,1} \ x_{i,2} \ \dots \ x_{i,d}]$
- Features can be real ($x_{i,f} \in \mathbb{R}$),
categorical
($x_{i,f} \in \{\text{red, green, blue}\}$), or more structured
- y : response (dependent) variable
 - $y \in \{-1, 1\}$: binary classification
 - $y \in \mathbb{R}$: regression
 - Typically, this is what we want to be able to predict, having observed some new \mathbf{x} .

Outline

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

Featurization

- Data is often not originally in vector form
 - Have to choose how to featurize
 - Features often encode expert knowledge of the domain
 - Can have a huge effect on performance
- Example: documents

[Cat Flees House Fire In New Mexico And Ends Up 240 Miles Away In ...](#)
CBS News, NY - Feb 18, 2008
(AP) A **cat** that fled a house fire is back home in Albuquerque, NM, after turning up some 240 miles away. The black and white **cat** named Miko disappeared in ...

 - “Bag of words” featurization: throw out order, keep count of how many times each word appears.
 - Surprisingly effective for many tasks
 - Sequence featurization: one feature for first letter in the document, one for second letter, etc.
 - Poor feature set for most purposes—similar documents are not close to one another in this representation.

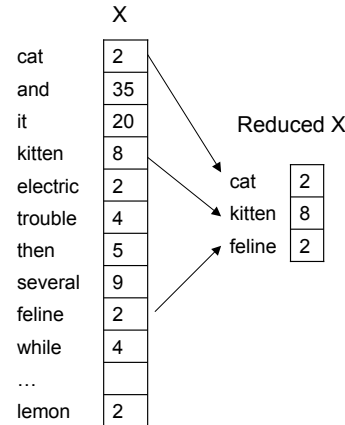
What is feature selection?

- 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?

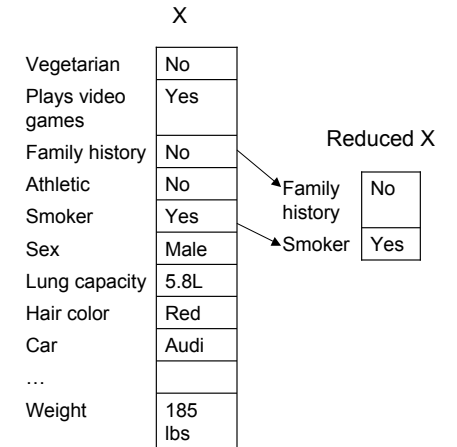
Task: classify whether a document is about cats

Data: word counts in the document



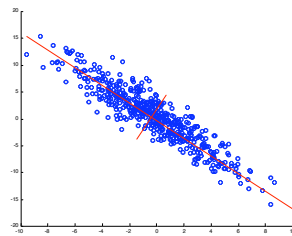
Task: predict chances of lung disease

Data: medical history survey



This vs. previous lecture

- One thing we will be doing: 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
 - Also, we will assume we have labeled data



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.

- Text classification
 - Features for all 10^5 English words, and maybe all word pairs
 - Common practice: throw in every feature you can think of, let feature selection get rid of useless ones
 - Training 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)
- Disease diagnosis
 - Features are outcomes of expensive medical tests
 - Which tests should we perform on patient?
- Embedded systems with limited resources
 - Classifier must be compact
 - Voice recognition on a cell phone
 - Branch prediction in a CPU (4K code limit)

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

Outline

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

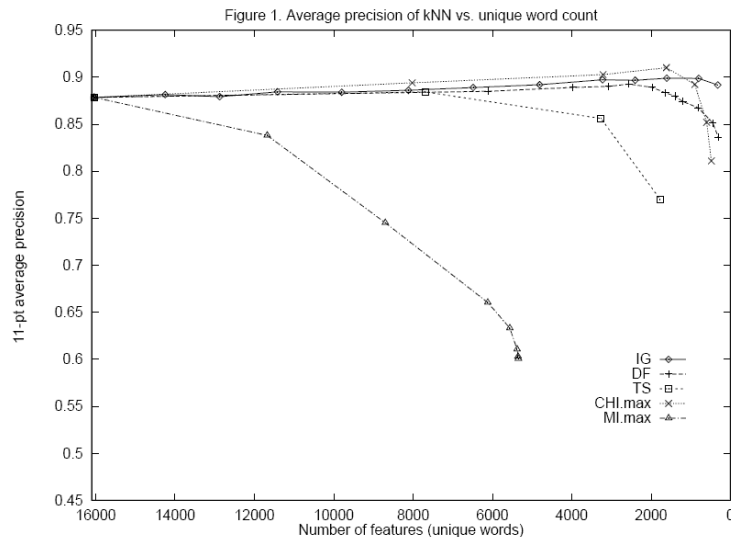
Filtering

- 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 which learning algorithm will be used.
 - Doesn't take into account interactions between features
- Suggestion: use light filtering as an efficient initial step if there are many obviously irrelevant features
 - Caveat : apparently useless features can be useful when grouped with others

Outline

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

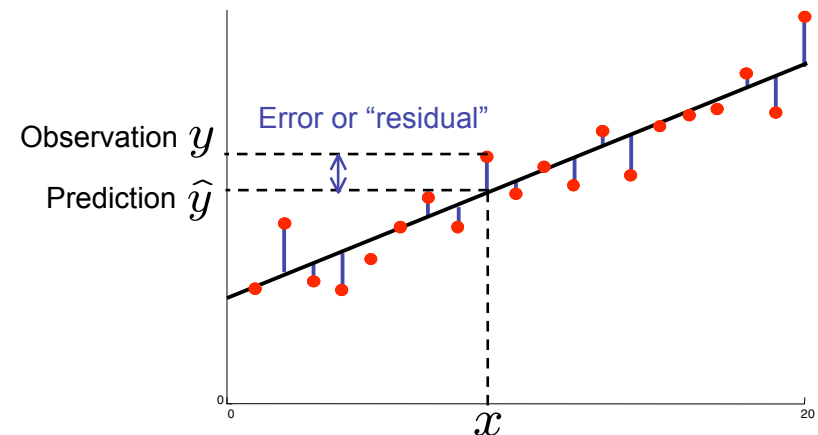
Model	Data	Parameters
Prediction rule: $y = \mathbf{w}^\top \mathbf{x}$	$\mathbf{x} \in \mathbb{R}^d$ Response: $y \in \mathbb{R}$	$\mathbf{w} \in \mathbb{R}^{d+1}$
	$\mathbf{w}^\top \mathbf{x} = w_0 + \sum_{i=1}^d x_i \cdot w_i$	

- Recall that we can fit it by minimizing the squared error:

$$\begin{aligned}\hat{\mathbf{w}} &= \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \\ &= (X^\top X)^{-1} X^\top Y\end{aligned}$$

Least Squares Fitting

(Romain's slide from 3 weeks ago)



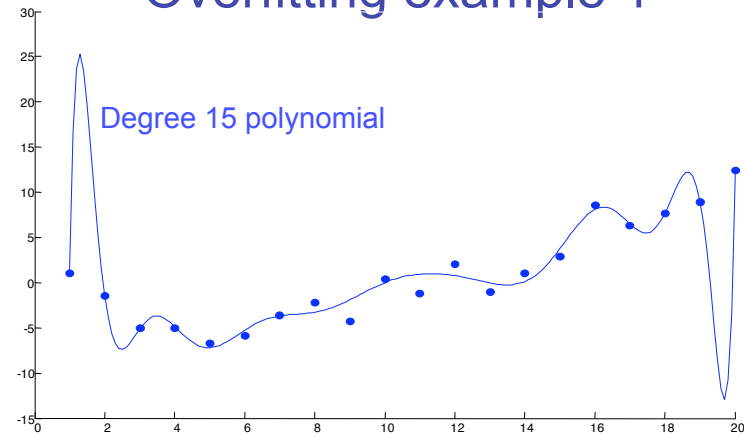
$$\text{Sum squared error: } L(\mathbf{w}) = \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

Model Selection

Model	Data	Parameters
Prediction rule:	$\mathbf{x}_s \in \mathbb{R}^{ s }$	$\mathbf{w} \in \mathbb{R}^{ s +1}$
$y = \mathbf{w}_s^\top \mathbf{x}_s$	Response: $y \in \mathbb{R}$	

- Consider a reduced model with only those features \mathbf{x}_f for $f \in s \subseteq \{1, 2, \dots, d\}$
- Squared error is now $L_s(\mathbf{w}_s) = \sum_{i=1}^n (y_i - \mathbf{w}_s^\top \mathbf{x}_{i,s})^2$
- We want to pick out the “best” s . Maybe this means the one with the lowest training error $\min_{\mathbf{w}_s} L_s(\mathbf{w}_s)$?
- Note $\min_{\mathbf{w}_s} L_s(\mathbf{w}_s) \geq \min_{\mathbf{w}} L(\mathbf{w})$
 - Just zero out terms in \mathbf{w} to match \mathbf{w}_s .
- Generally speaking, training error will only go up in a simpler model. So why should we use one?

Overfitting example 1

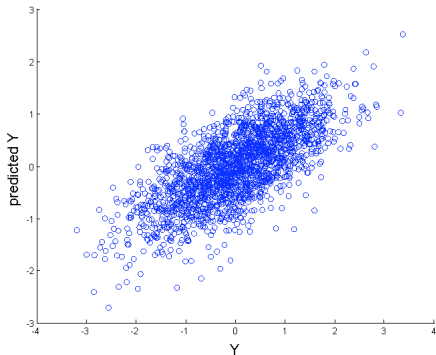


- This model is too rich for the data
- Fits training data well, but doesn't generalize.

(From Romain's lecture)

Overfitting example 2

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



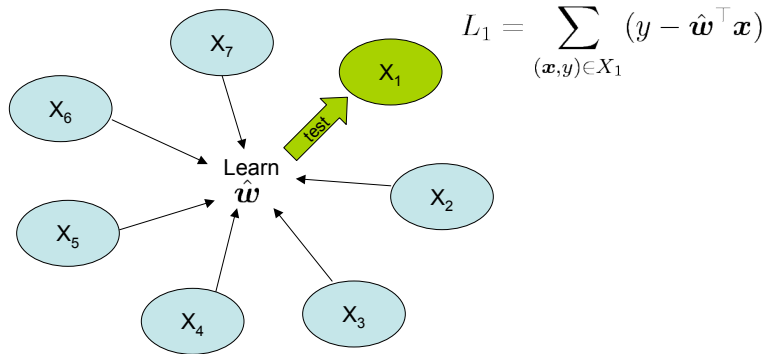
It really looks like we've found a relationship between \mathbf{x} and y ! But no such relationship exists, so $\hat{\mathbf{w}}$ will do no better than random on new data.

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've seen that $\min_{\mathbf{w}_s} L_s(\mathbf{w}_s)$ is not appropriate.
- 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.

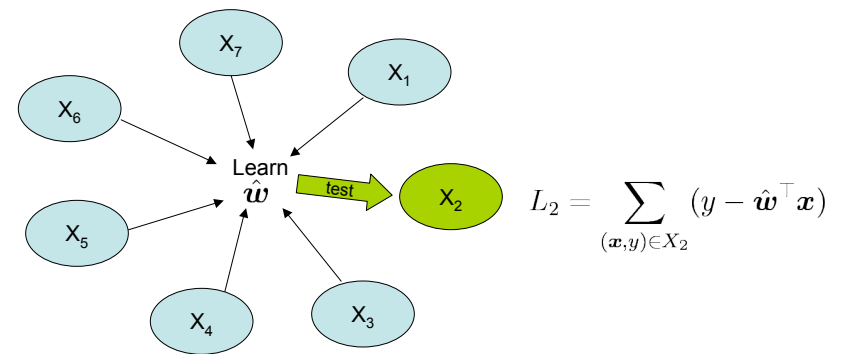
K-fold cross validation

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



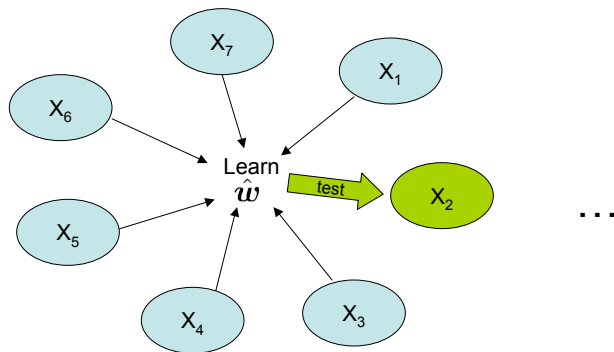
K-fold cross validation

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



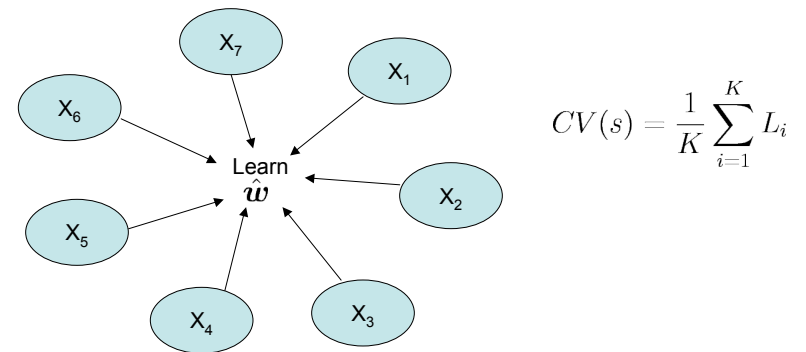
K-fold cross validation

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



Model Search

- We have an objective function $K(s) = CV(s)$
 - 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
 - 2^n possible subsets s
- Greedy search is common and effective

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.
 - BIC: add penalty $d \log n$

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.

Outline

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

Regularization

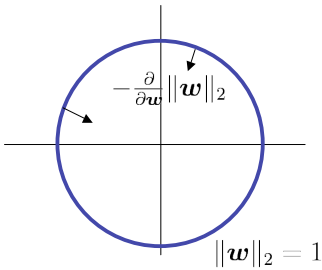
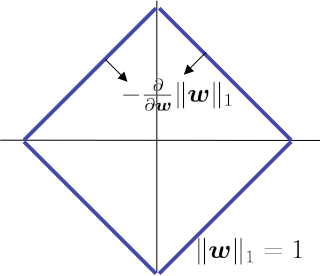
Regularization

- In certain cases, we can move model selection *into* the induction algorithm
- This is sometimes called an *embedded* feature selection algorithm

- Recall Regularization: add model complexity penalty to training error.
- $J(\mathbf{w}) = L(\mathbf{w}) + C\|\mathbf{w}\|_p = \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + C\|\mathbf{w}\|_p$ for some constant C
- Now $\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} J(\mathbf{w})$
- 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

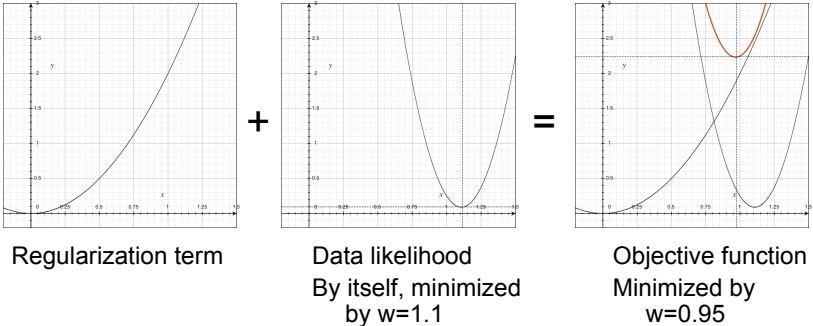
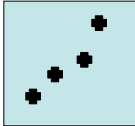
Level sets of L₁ vs L₂

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



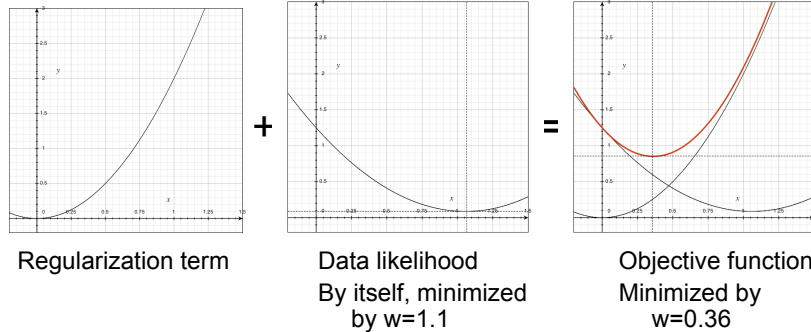
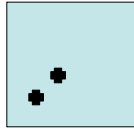
Univariate example: L₂

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



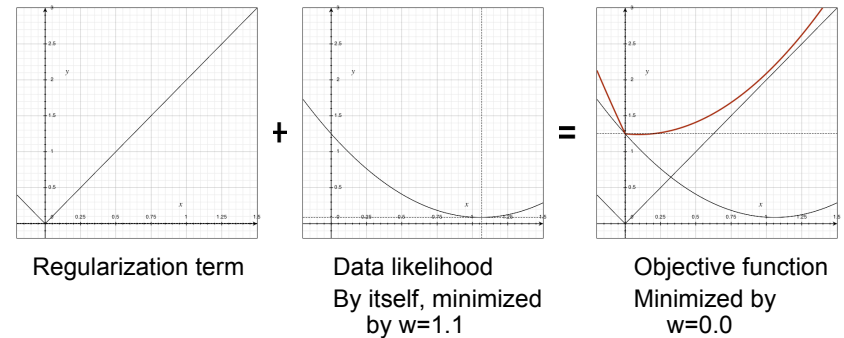
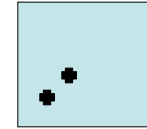
Univariate example: L_2

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



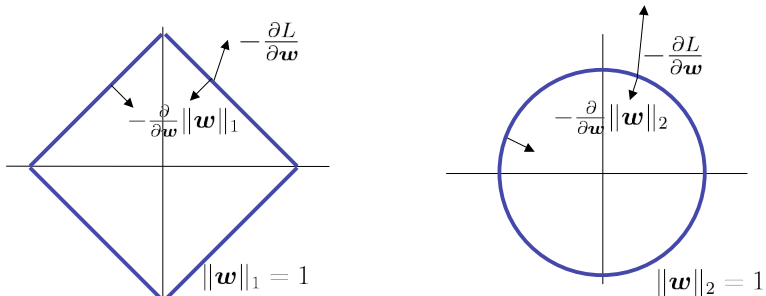
Univariate example: L_1

- Case 1, when there is a lot of data supporting our hypothesis:
 - Almost the same resulting w as L_2
- Case 2, when there is NOT a lot of data supporting our hypothesis
- Get $w = \text{exactly zero}$



Multivariate case: w gets cornered

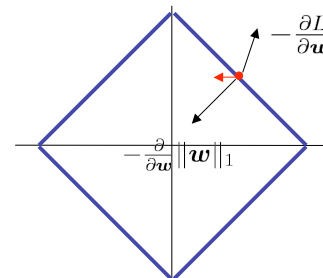
- To minimize $J(w) = L(w) + \|w\|_p$, we can solve $\frac{\partial J}{\partial w} = 0$ by (e.g.) gradient descent.



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

Multivariate case: w gets cornered

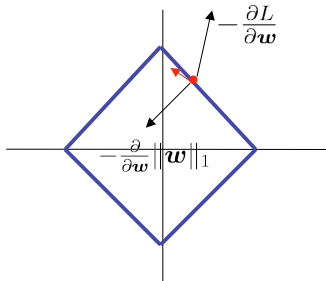
- To minimize $J(w) = L(w) + \|w\|_p$, we can solve $\frac{\partial J}{\partial w} = 0$ by (e.g.) gradient descent.



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

Multivariate case: w gets cornered

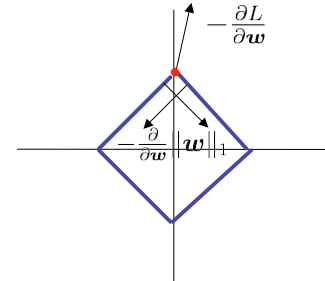
- To minimize $J(w) = L(w) + \|w\|_p$, we can solve $\frac{\partial J}{\partial w} = 0$ by (e.g.) gradient descent.



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

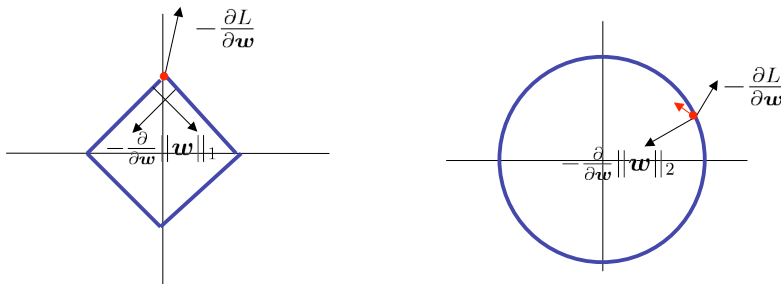
Multivariate case: w gets cornered

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

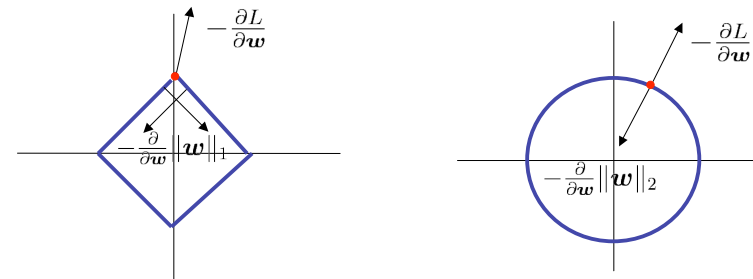


- Minimization is a tug-of-war between the two terms
- w is forced into the corners—components are zeroed
 - Solution is often *sparse*

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.

$$\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + C \sum_{f=1}^d |\mathbf{w}_f|$$
- Two big questions:
 - 1. How do we perform this minimization?
 - With L_2 penalty it's easy—saw this in a previous lecture
 - With L_1 it's not a least-squares problem anymore
 - 2. How do we choose C?

Remark

- Not to be confused: two uses of L1 for regression:
 - lasso for sparsity---what we just saw
$$\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 + C \sum_{f=1}^d |\mathbf{w}_f|$$
 - L1 loss---for robustness, another topic.

$$\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n |y_i - \mathbf{w}^\top \mathbf{x}_i| + C \|\mathbf{w}\|_p$$

Least-Angle Regression

- 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 \mathbf{w} for all possible C values simultaneously, as efficiently as least-squares
 - Can choose exactly how many features are wanted

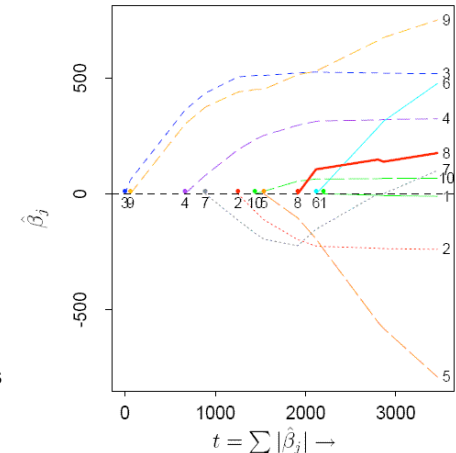


Figure taken from Hastie et al (2004)

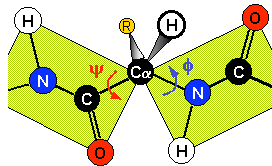
L_1 Vs L_2 [Gao et al '07]

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

	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

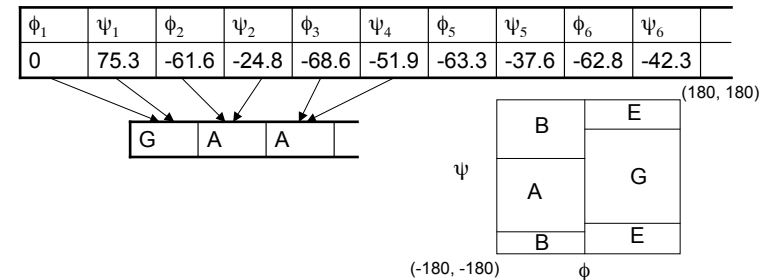
Case Study: Protein Energy Prediction [Blum et al '07]

- What is a protein?
 - A protein is a chain of amino acids.
 - The sequence of amino acids (there are 20 different kinds) is called the “primary structure.”
 - E.g. protein 1di2 (double stranded RNA binding protein A):
MPVGS LQELAVQKGWRLPEYTV AQESGPPHKREFTITCRVETF
VETGSGTSKQVAKRVAAEKLTKFKT
 - Certain amino acids like to bond to certain others
- Proteins fold into a 3D conformation by minimizing energy
 - “Native” conformation (the one found in nature) is the lowest energy state.
- Data: many different conformations of the same amino acid sequence
- Response variable: energy
- Natural structure representation: ϕ and ψ *torsion angles*.



Featurization

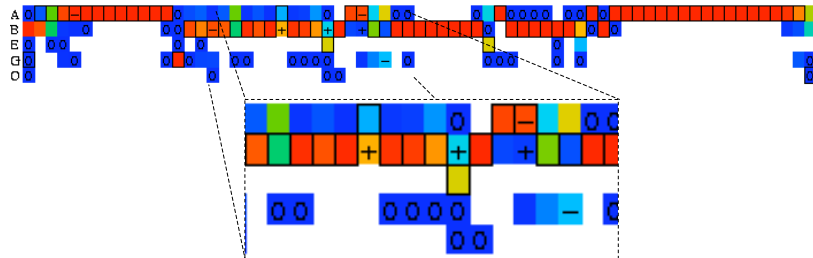
- Torsion angle features can be continuous or discrete



- Bins in the *Ramachandran* plot correspond to common structural elements
 - Secondary structure: alpha helices and beta sheets
 - Here, domain knowledge used in featurization.

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)



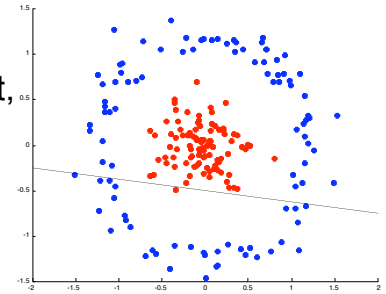
Outline

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

Kernel Methods

- Expanding feature space gives us new potentially useful features.
- Kernel methods let us work implicitly in a high-dimensional feature space.
 - All calculations performed quickly in low-dimensional space.

Recall...



- Linear models: convenient, fairly broad, but limited
- We can increase the expressiveness of linear models by expanding the feature space.
 - E.g. $\Phi([x_1 \ x_2]) = [1 \ \sqrt{2}x_1 \ \sqrt{2}x_2 \ \sqrt{2}x_1x_2 \ x_1^2 \ x_2^2]$
 - Now feature space is \mathbb{R}^6 rather than \mathbb{R}^2
 - Example *linear* predictor in these features:

$$y = [1 \ 0 \ 0 \ 0 \ -1 \ -1] \cdot \Phi(\mathbf{x}) = 1 - x_1^2 - x_2^2$$

The kernel trick

- Can still fit by old methods, but it's more expensive
 - If \mathbf{x} is itself d -dimensional, $\Phi(\mathbf{x})$ is $\mathcal{O}(d^2)$ -dimensional
- Many algorithms we've looked at only see data through inner products (or can be rephrased to do so)
 - Perceptron, logistic regression, etc.
- But notice:

$$\begin{aligned} \Phi(\mathbf{x})^\top \Phi(\mathbf{z}) &= [1 \ \sqrt{2}x_1 \ \sqrt{2}x_2 \ \sqrt{2}x_1x_2 \ x_1^2 \ x_2^2] \\ &\quad \cdot [1 \ \sqrt{2}z_1 \ \sqrt{2}z_2 \ \sqrt{2}z_1z_2 \ z_1^2 \ z_2^2] \\ &= 1 + 2x_1z_1 + 2x_2z_2 + 2x_1x_2z_1z_2 + x_1^2z_1^2 + x_2^2z_2^2 \\ &= (1 + x_1z_1 + x_2z_2)^2 \\ &= (1 + \mathbf{x}^\top \mathbf{z})^2 \end{aligned}$$
- We can just compute inner product in original space.
- This is called the *kernel trick*:
 - Working in high-dimensional feature space implicitly through an efficiently-computable inner product *kernel*.

Kernel methods

- Representation theorem: for many kinds of models with linear parameters \mathbf{w} , we can write for some α .

$$\hat{\mathbf{w}} = \sum_{i=1}^n \alpha_i \mathbf{x}_i$$
 - For linear regression, our predictor can be written

$$y = \mathbf{w}^\top \mathbf{x} = \sum_{i=1}^n \alpha_i \mathbf{x}_i^\top \mathbf{x}$$
 - Never need to deal with \mathbf{w} explicitly; just need a kernel $k(\mathbf{x}_i, \mathbf{x})$ to take the place of $\mathbf{x}_i^\top \mathbf{x}$ in comparing data points to *each other*.
- Mercer theorem: every “qualifying” inner product kernel has an associated (possibly infinite-dimensional) feature space.
- Polynomial kernels:
 - $k(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^\top \mathbf{z})^r$: feature space = all monomials in \mathbf{x} and \mathbf{z} of degree $\leq r$
- RBF kernel:
 - $k(\mathbf{x}, \mathbf{z}) = e^{-\frac{1}{\sigma^2} \|\mathbf{x} - \mathbf{z}\|^2}$: feature space is infinite dimensional

Dynamic programming string kernel

[Lodhi et al, 2002]

- Feature space: all possible substrings of k letters, not necessarily contiguous.
 - E.g. a-p-l-s in “apples are tasty”
 - Value for each feature is $\exp\{-(\text{full length of substring in text})\}$

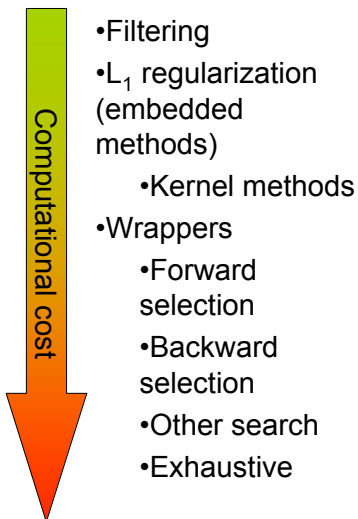
...	
a-p-l-s	0.12
p-l-e-t	0.01
s-a-r-e	1.0
...	

- Very high dimensional!
- Surprisingly, kernel can be computed efficiently using dynamic programming.
 - Runs in time $O(\text{product of lengths of 2 documents})$
- Text classification results: superior to using bag-of-words feature space.
 - No way we could use this feature space without kernel methods.

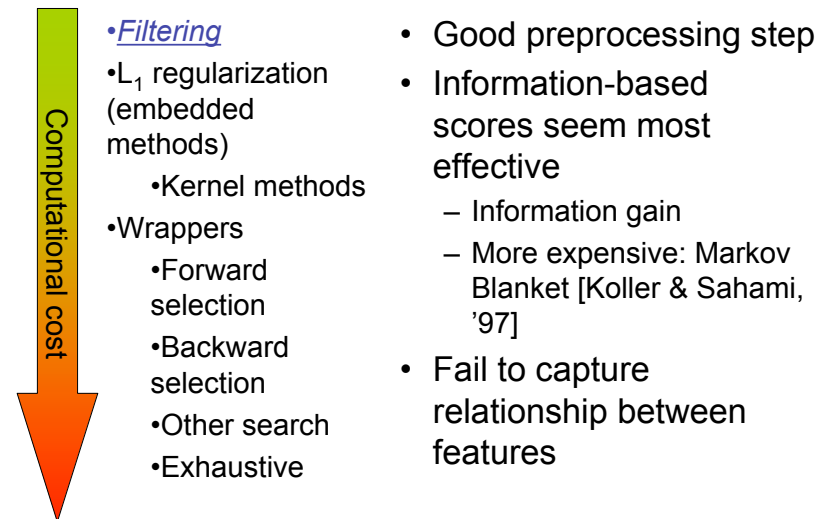
Outline

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

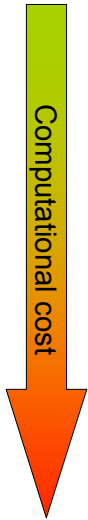
Summary



Summary



Summary



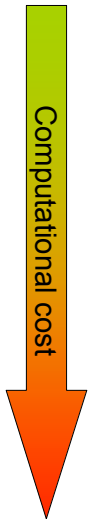
- Filtering
- L_1 regularization (embedded methods)
 - Kernel methods
- Wrappers
 - Forward selection
 - Backward selection
 - Other search
 - Exhaustive
- Fairly efficient
 - LARS-type algorithms now exist for many linear models
 - Ideally, use cross-validation to determine regularization coeff.
- Not applicable for all models
- Linear methods can be limited
 - Common: fit a linear model initially to select features, then fit a nonlinear model with new feature set

Summary



- Filtering
- L_1 regularization (embedded methods)
 - Kernel methods
- Wrappers
 - Forward selection
 - Backward selection
 - Other search
 - Exhaustive
- Expand the expressiveness of linear models
- Very effective in practice
- Useful when a similarity kernel is natural to define
- *Not* as interpretable
 - They don't really perform feature selection as such
- Achieve parsimony through a different route
 - Sparsity in *data*

Summary



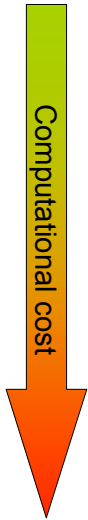
- Filtering
- L_1 regularization (embedded methods)
 - Kernel methods
- Wrappers
 - Forward selection
 - Backward selection
 - Other search
 - Exhaustive
- Most directly optimize prediction performance
- Can be very expensive, even with greedy search methods
- Cross-validation is a good objective function to start with

Summary



- Filtering
- L_1 regularization (embedded methods)
 - Kernel 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



- Filtering
 - L_1 regularization (embedded methods)
 - Kernel methods
 - Wrappers
 - Forward selection
 - Backward selection
 - Other search
 - Exhaustive
- Generally more effective than greedy

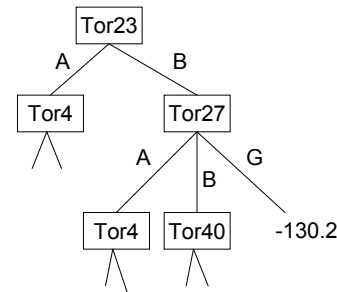
Summary



- Filtering
 - L_1 regularization (embedded methods)
 - Kernel 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

Additional slides

Decision Trees



- Effectively a stepwise filtering method
- In each subtree, only a subset of the data is considered
- Split on top feature according to filtering criterion
- Stop according to some stopping criterion
 - Depth, homogeneity, etc
- In final tree, only a subset of features are used
- Very useful with boosting
 - Connection between Adaboost and forward selection

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.