Feature Engineering and Selection

CS 294: Practical Machine Learning
October 1st, 2009

Alexandre Bouchard-Côté
Abstract supervised setup

• Training: \( \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \)

• \( x_i \): input vector

\[
\begin{bmatrix}
   x_{i,1} \\
   x_{i,2} \\
   \vdots \\
   x_{i,n}
\end{bmatrix}, \quad x_{i,j} \in \mathbb{R}
\]

• \( y \): 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 \( x \).
Concrete setup

Input

Output

“Danger”
Featurization

Input

Features

Output

“Danger”

Input

Output

HOT OR NOT
Over 12 Billion votes counted & 25,987,000 photos submitted.
Official Rating

8.9

Based on 4984 votes
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

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

PERSONAL
Basics: bag of words

• Input: \( \mathbf{x} \) (email-valued)
• Feature vector:

\[
\begin{bmatrix}
  f_1(\mathbf{x}) \\
  f_2(\mathbf{x}) \\
  \vdots \\
  f_n(\mathbf{x})
\end{bmatrix}, \quad \text{e.g. } f_1(\mathbf{x}) = \begin{cases} 
1 & \text{if the email contains "Viagra"} \\
0 & \text{otherwise}
\end{cases}
\]

• Learn one weight vector for each class:

\( w_y \in \mathbb{R}^n, \ y \in \{\text{SPAM, WORK, PERS}\} \)

• Decision rule:  \( \hat{y} = \arg\max_y \langle w_y, f(\mathbf{x}) \rangle \)
Implementation: exploit sparsity

**Feature vector hashtable**

```java
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)
In multiclass classification, output space often has known structure as well.

Example: a hierarchy:

- **Emails**
  - **Spam**
    - Advance fee frauds
    - Spam advertised sites
  - Backscatter
  - **Ham**
    - Work
    - **Personal**
    - Mailing lists
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 $y$

<table>
<thead>
<tr>
<th>Before:</th>
<th>• One weight/class: $w_y \in \mathbb{R}^n$,</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Decision rule: $\hat{y} = \text{argmax}_y \langle w_y, f(x) \rangle$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>After:</th>
<th>• Single weight: $w \in \mathbb{R}^m$,</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• New rule: $\hat{y} = \text{argmax}_y \langle w, f(x, y) \rangle$</td>
</tr>
</tbody>
</table>
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
Exploit the information in the hierarchy by activating both coarse and fine versions of the features on a given input:

\[
\begin{align*}
X & \rightarrow \quad & UNIGRAM: Alex \ AND \ CLASS = \text{PERSONAL} \\
X & \rightarrow \quad & UNIGRAM: Alex \ AND \ CLASS = \text{HAM}
\end{align*}
\]
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

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

   CLUSTER=1

   CLUSTER=2
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 an email \( e \), its belief \( B(e) \) that the email is spam

– We want to model monotonicity

– Solution: thermometer feature

\[
\begin{align*}
B(e) > 0.4 & \text{ AND } \quad \text{CLASS=} \text{SPAM} \\
B(e) > 0.6 & \text{ AND } \quad \text{CLASS=} \text{SPAM} \\
B(e) > 0.8 & \text{ AND } \quad \text{CLASS=} \text{SPAM} \\
\text{...} & \quad \text{...} \\
\end{align*}
\]
Dealing with continuous data

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

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

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

- UNIGRAM: Viagra
- UNIGRAM: the
- BIGRAM: the presence
- BIGRAM: hello Alex
- UNIGRAM: Alex
- UNIGRAM: of
- BIGRAM: absence of
- BIGRAM: classify email
- BIGRAM: free Viagra
- BIGRAM: predict the
- ...
- BIGRAM: emails as

Reduced X

Vegetarian
Plays video games
Family history
Athletic
Smoker
Gender
Lung capacity
Hair color
Car
...
Weight

Reduced X
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 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]

Figure 1. Average precision of kNN vs. unique word count
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 : \( \mathbf{x} \in \mathbb{R}^d \)

Parameters: \( \mathbf{w} \in \mathbb{R}^{d+1} \)

Response : \( y \in \mathbb{R} \)

Prediction : \( y = \mathbf{w}^\top \mathbf{x} \)

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

\[
\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2
\]
Least Squares Fitting
(Fabian’s slide from 3 weeks ago)

Observation $y$
Prediction $\hat{y}$

Error or “residual”

Sum squared error: $L(w) = \sum_{i=1}^{n} (y_i - w^\top x_i)^2$
Naïve training error is misleading

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

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

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

Prediction : $y = w^\top x$

• Consider a reduced model with only those features $x_f$ for $f \in s \subseteq \{1, 2, \ldots, d\}$
  – Squared error is now $L_s(w_s) = \sum_{i=1}^{n} (y_i - w_s^\top x_{i,s})^2$

• Is this new model better? Maybe we should compare the training errors to find out?

• Note $\min_{w_s} L_s(w_s) \geq \min_w L(w)$
  – Just zero out terms in $w$ to match $w_s$.

• 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 $\mathbf{x}_i \in \mathbb{R}^{1000}$, $\mathbf{x}_i \sim \mathcal{N}(0, I)$ 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 $\hat{\mathbf{w}} = \text{argmin}_{\mathbf{w}} L(\mathbf{w})$
- Use this to predict $y_i$ for each $\mathbf{x}_i$ by $\hat{y}_i = \mathbf{w}^\top \mathbf{x}_i$

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’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

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

\[
L_2 = \sum_{(x,y) \in X_2} (y - \hat{w}^\top x)
\]
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 $K(s) = CV(s)$
  – 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
  – 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 $s = \{1, 2, \ldots, n\}$

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.
  \[ J(w) = L(w) + C\|w\|_p = \sum_{i=1}^{n}(y_i - w^\top x_i)^2 + C\|w\|_p \]
  for some constant C
- Find \( \hat{w} = \arg\min_w J(w) \)
- 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
  \[ ||\mathbf{w}||_2 = \sqrt{w_1^2 + \cdots + w_n^2} \]

- $p = 1$: Taxicab or Manhattan
  \[ ||\mathbf{w}||_1 = |w_1| + \cdots + |w_n| \]

- General case: $0 < p \leq \infty$
  \[ ||\mathbf{w}||_p = p\sqrt[|w_1|^p + \cdots + |w_n|^p} \]
Univariate case: intuition

Penalty

Feature weight value
Univariate case: intuition

L1 penalizes more than L2 when the weight is small.
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 $L_2$
- 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)

$$\| \mathbf{w} \|_1 = \sum_{f=0}^{d} |w_f|$$

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

Weight of feature #1

Weight of feature #2

$$\| \mathbf{w} \|_1 = 1$$

$$\| \mathbf{w} \|_2 = 1$$
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
To minimize $J(w) = L(w) + \|w\|_p$, we can solve 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
L₂ does not zero components

• L₂ 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.

$$\hat{w} = \arg\min_w \sum_{i=1}^{n} (y_i - w^\top x_i)^2 + C||w||_1$$

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

$$\hat{w} = \arg\min_w \sum_{i=1}^{n} (y_i - w^\top x_i)^2 + C||w||_2^2$$
Lasso Regression [Tibshirani ‘94]

- Simply linear regression with an $L_1$ penalty for sparsity.

\[ \hat{w} = \arg\min_w \sum_{i=1}^{n} (y_i - w^\top x_i)^2 + C ||w||_1 \]

- 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)
• Not to be confused: two orthogonal uses of L1 for regression:
  – lasso for **sparsity**: what we just described
    \[ \hat{w} = \arg\min_w \sum_{i=1}^{n} (y_i - w^T x_i)^2 + C \sum_{f=1}^{d} |w_f| \]
  – L1 loss: for **robustness** (Fabian’s lecture).
    \[ \hat{w} = \arg\min_w \sum_{i=1}^{n} |y_i - w^T x_i| + C \|w\|_p \]
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
L₁ Vs L₂ [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)
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

Computational cost
Summary: how to do feature selection

- **Filtering**
  - L₁ regularization (embedded methods)
- **Wrappers**
  - Forward selection
  - Backward selection
  - Other search
  - Exhaustive
- Good preprocessing step
- Fails to capture relationship between features

Computational cost
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

Computational cost
• 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]

<table>
<thead>
<tr>
<th>Language</th>
<th>‘fish’</th>
<th>‘fear’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hawaiian</td>
<td>iʔa</td>
<td>makaʔu</td>
</tr>
<tr>
<td>Samoan</td>
<td>iʔa</td>
<td>mataʔu</td>
</tr>
<tr>
<td>Tongan</td>
<td>ika</td>
<td>mataku</td>
</tr>
<tr>
<td>Maori</td>
<td>ika</td>
<td>mataku</td>
</tr>
</tbody>
</table>
Feature engineering case study: Modeling language change [Bouchard et al. 07,09]

Tasks:
- Proto-word reconstruction
- Infer sound changes

<table>
<thead>
<tr>
<th></th>
<th>‘fish’</th>
<th>‘fear’</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hawaiian</td>
<td>iʔa</td>
<td>makaʔu</td>
</tr>
<tr>
<td>Samoan</td>
<td>iʔa</td>
<td>mataʔu</td>
</tr>
<tr>
<td>Tongan</td>
<td>ika</td>
<td></td>
</tr>
<tr>
<td>Maori</td>
<td>ika</td>
<td>mataku</td>
</tr>
</tbody>
</table>

Proto-Oceanic

* k > ?
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
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:
\( \phi \) and \( \psi \) torsion angles.

Restrict next wave of search to agree with features that predicted high energy
Featurization

- Torsion angle features can be binned

<table>
<thead>
<tr>
<th>φ₁</th>
<th>ψ₁</th>
<th>φ₂</th>
<th>ψ₂</th>
<th>φ₃</th>
<th>ψ₄</th>
<th>φ₅</th>
<th>ψ₅</th>
<th>φ₆</th>
<th>ψ₆</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>75.3</td>
<td>-61.6</td>
<td>-24.8</td>
<td>-68.6</td>
<td>-51.9</td>
<td>-63.3</td>
<td>-37.6</td>
<td>-62.8</td>
<td>-42.3</td>
</tr>
</tbody>
</table>

- 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/](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