Regularization, part 2

Thanks to Professor Dale Schuurmans
University of Alberta and Google Brain

Instructor: Farzaneh Mirzazadeh
Department of Computer Science, UCSC, Winter 2017

Recall

Recall: Linear Hypotheses

$$h(\mathbf{x}) = \mathbf{x}^\top \mathbf{w}$$

Recall: Regularization concept

- Goal : Give a preference to models with smaller weight.
- How? By adding a penalty term, **regularizer**, with value proportional to the size of $\mathbf{w}$ to error term:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{t} err(X_{i:}\mathbf{w}, \mathbf{y}_i) + \beta\|\mathbf{w}\|$$

- $\beta \geq 0$ **regularization parameter**
- Trade-off between minimizing **error** vs **size of** $\mathbf{w}$
- Two parts:
  - Error function (aka loss function) tries to fit the model to data.
  - Regularizer tries to shrink the weights.

Recall: Regularization

Important types of regularizers

- How to measure size of w? i.e which norm to use?
  1. L_2 norm squared regularization
  2. L_1 norm regularization

Math background. Recall: Definition of L_P norm of a vector

$$\|\mathbf{a}\|_p = \|[a_1, \ldots, a_n]^\top\|_p = (|a_1|^p + |a_2|^p + \cdots + |a_n|^p)^{1/p}, p \geq 1$$

Recall: L_2 regularized L_2 error minimization

Fancy name: Ridge Regression

Training Problem

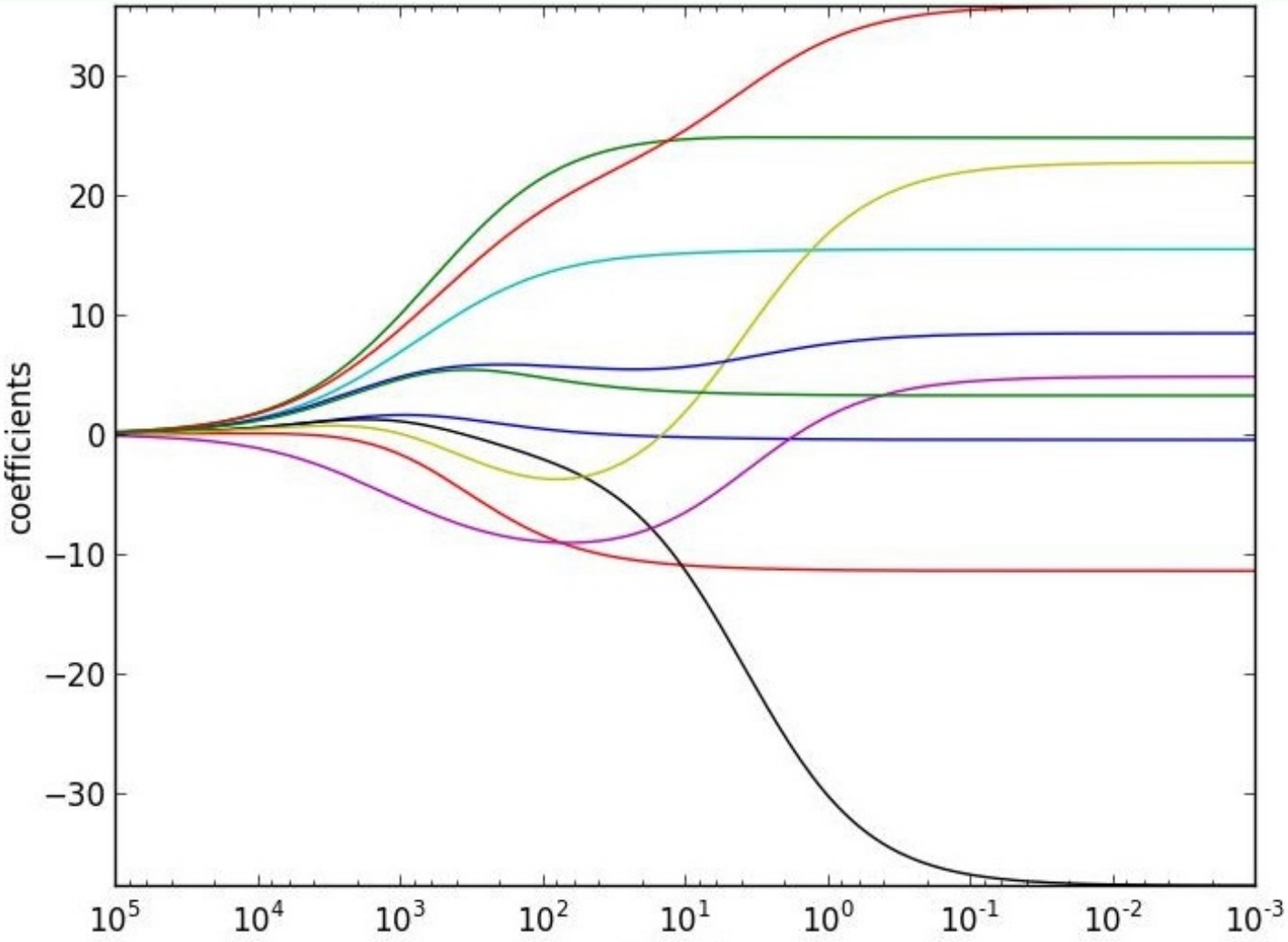
$$\min_{\mathbf{w}} \sum_{i=1}^{t} \|X_{i:}\mathbf{w} - \mathbf{y}_i\|^2 + \beta\|\mathbf{w}\|_2^2$$
$$= \min_{\mathbf{w}} (X\mathbf{w} - \mathbf{y})^\top (X\mathbf{w} - \mathbf{y}) + \beta\, \mathbf{w}^\top \mathbf{w}$$

How to solve?

It has closed-form solution

- Approach similar to least squares. Solve a system of equations.

In practice: Increasing β pushes weights w of ridge regression to zero.



L_2 regularization

- Increasing the regularization parameter leads weights $\mathbf{w}$ getting close to zero, but not exacly zero.
- The contribution of features to the final prediction decreases.
- The learned function would be smoother.

Question

- Is it possible to completely drop some of features?
- Is there a regularization method in which some of $\mathbf{w}$ components become exactly zero?
- Answer: YES! L_1 regularization

L_2 regularization

- Increasing the regularization parameter leads weights $\mathbf{w}$ getting close to zero, but not exacly zero.
- The contribution of features to the final prediction decreases.
- The learned function would be smoother.

Question

- Is it possible to completely drop some of features?
- Is there a regularization method in which some of $\mathbf{w}$ components become exactly zero?
- Answer: YES! L_1 regularization

Regularization is not needed on the intercept.

See page 64 of the course book.

Standardizing data, penalizing bias term

- L_2 regularized solutions are not equivalent under scaling of inputs $\Longrightarrow$ It is a good idea to standardize inputs first.
- Intercept (bias term $\mathbf{w}_0$) does not need to be penalized: Penalizing intercept make the procedure dependent on the origin
- In other words, adding a constant c to each of the targets would not simply result in a shift of the predictions by the same amount.
- $\hat{\mathbf{w}}_0 = \bar{\mathbf{y}}$ After centering inputs, the intercept could be estimated as average target.

Recall

L_2 regularization has a side effect: Representation theorem!

Recall: Representer Theorem

A consequence of L_2 regularization that leads to kernels

Representer Theorem

For any loss function $l(\hat{y}, y) = err(\hat{y}, y)$ and for the optimal solution of training problem:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{i=1}^{\top} l(X_{i:}\mathbf{w}, \mathbf{y}_i) + \beta\|\mathbf{w}\|_2^2$$

$\mathbf{w}^*$, satisfies $\mathbf{w}^* = X^\top \boldsymbol{\alpha}^*$, for some $\boldsymbol{\alpha}^*$.

- In other words, the optimal $\mathbf{w}^*$ is actually a weighted average of training examples. Each element of α^* explains how much the corresponding training example contributes.
- In other words, The optimal weight vector must be in the row span of X.

Recall: Duality

Primal (one way of looking at the learning problem)

- Training $\min_{\mathbf{w}} \sum_{i=1}^{t} l(X_{i:}\mathbf{w}, \mathbf{y}) + \beta\|\mathbf{w}^*\|_2^2$
- Prediction: Given $\mathbf{x}_\circ$, predict $\hat{y} = \mathbf{w}^{*\top}\mathbf{x}_\circ$

Dual (another equivalent way of looking at the problem

- Training $\min_{\boldsymbol{\alpha}} \sum_{i=1}^{t} l(X_{i:}X_{i:}^\top\boldsymbol{\alpha}, \mathbf{y}_i) + \beta\boldsymbol{\alpha}^\top XX^\top\boldsymbol{\alpha}$
- Prediction: Given $\mathbf{x}_\circ$, predict $\hat{y} = \boldsymbol{\alpha}^{*\top}X\mathbf{x}_\circ$

Observation

- Note: In dual form X is never alone. Always $XX^\top$
- Can solve for example weights $\boldsymbol{\alpha}$ instead of feature weights $\mathbf{w}$
- Size of variables $\mathbf{w}$, $\boldsymbol{\alpha}$ is different in primal and dual.
  n vs t $\implies$ Difference in computational complexity of solving primal and dual.

Recall: kernelization

Kernelization

1. Reformulate training and prediction in a form that X never appears alone, but always in terms of $XX^\top$
2. Replace $XX^\top$ with any *eligible* $t \times t$ matrix K where:
   $K_{ij} =$ The similarity between ith and jth training example. Such a matrix is called kernel matrix.

Effects

- Any form of non-linear feature expansion could be used this way.
- Using very high dimensional feature expansions would become possible. (Optimize in $\boldsymbol{\alpha}$ instead of $\mathbf{w}$.)

Recall: Kernelized training and prediction formulations

Kernelized training

Kernelized Training

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}} \sum_{i=1}^{t} l(K_{i:}\boldsymbol{\alpha}, \mathbf{y}_i) + \beta \boldsymbol{\alpha}^\top K \boldsymbol{\alpha}$$

where K_{ij} is the inner product between the feature expansions of the i th and jth examples.

Kernelized prediction

Prediction: Given $\mathbf{k}_\circ$, predict $\hat{y} = \boldsymbol{\alpha}^{*\top} \mathbf{k}_\circ$.
Note: $\mathbf{k}_\circ$ holds the similarity (in terms of inner product) of the test example to all training examples. $\mathbf{k}_i = X_{i:}^\top x_\circ$.

Only needs kernel values!!! Nice!

What are valid matrices to be used as kernels?

- Technically any positive semidefinite matrix.
- $K = \Phi\Phi^\top$ for some Φ

Linear algebra background: Symmetric Positive semidefinite S

A square and symmetric matrix $S = S^\top$

- $\mathbf{u}^\top S\mathbf{u} \geq 0$, for any vector $\mathbf{u}$
- All eigenvalues are non-negative.
- Could be written as $S = A^\top A$ for some matrix A.

Kernels: Meaning

- Similarity measure on space of objects $\mathcal{X}$
- Examples
  1. Similarity between vectors, $\mathcal{X} = \mathbb{R}^n$
  2. Similarity between strings, $\mathcal{X} = \{a, \ldots, z\}^*$
  3. Similarity between graphs
  4. Similarity between sentences
  5. Similarity between parse trees

How to come up with a kernel matrix?

1. From explicit feature maps. Can use trick (kernel trick) for efficiently generating K.
2. Forming new kernels from old kernels.
3. Special hand crafted kernels. Background knowledge would be involved: How similar are two proteins P_1, P_2? A biologist might be able to tell us?

How to construct kernel operators

By explicit feature mapping

$$x \to \boldsymbol{\phi}(x)$$

E.g. let x be a string.

$$x \in \{a \ldots, z\}^*$$

Given string x, could measure features like:
number of a s
number of b s
number of ab s
Then define kernel operator

$$\begin{aligned} k(x, y) &= \boldsymbol{\phi}(x)^\top \boldsymbol{\phi}(y) \\ &= \sum_f \phi_f(x)\phi_f(y) \end{aligned}$$

By using kernel operators that preserve kernality

Methods for making kernels from simpler kernels

1. Addition of a nonnegative constant $\tilde{k}(x,y) = k(x,y) + c, c \geq 0$
2. Normalization $\tilde{k}(x,y) = \frac{k(x,y)}{\sqrt{k(x,x)k(y,y)}}$
3. Summation $\tilde{k}(x,y) = k_1(x,y) + k_2(x,y)$
4. Nonnegative scalar multiplication $\tilde{k}(x,y) = ak(x,y)$ where $a > 0$
5. Product $\tilde{k}(x,y) = k_1(x,y)k_2(x,y)$
6. Composition $\tilde{k}(x,y) = k_2(\phi_1(x), \phi_1(y))$
7. Power $\tilde{k}(x,y) = k(x,y)^d, d \in \mathbb{R}$
8. Exponential $\tilde{k}(x,y) = e^{k(x,y)}$

Example:

$$\tilde{k}(x, y) = k_1(x, y) + k_2(x, y)$$

Let $k(x, y) = \phi_1^\top(x)\phi_1(y)$, and $k(x, y) = \phi_2^\top(x)\phi_2(y)$ Define:

$$\tilde{\phi}(x) := [\phi_1^\top(x)\phi_2^\top(x)]^\top$$

Then $\tilde{k}(x, y) = \tilde{\phi}_1^\top(x)\tilde{\phi}_1(y)$
In other words, by concatenating two different feature representations and then computing their kernel, the corresponding kernels would be added together.
Exercise: Show why the rest of combination rules work.

Engineering special kernel constructions

An example string kernel

- A kernel between two strings s_1, s_2:
- Number of consequent substrings of length l in common.
- Exercise: Show that such a construct is a kernel.

Where to read about kernels?

- Shaw-Taylor and Cristianini (2004), sec 2.2.3, chap 3
- Hastie et al. 2nd ed (2009), Sec 5.8, 6.1, 6.2, 6.7
- Bishop (2006), sec 6.2

L_1 regularization

L_1 regularization

L_1 regularized training

$$\min_{\mathbf{w}} \sum_i l(X_{i:}\mathbf{w}, \mathbf{y}_i) + \beta\|\mathbf{w}\|_1$$

- Regularization parameter $\beta \geq 0$,
- Problem still convex in $\mathbf{w}$ (if the loss function $l(.)$ is convex).
- Has a global min.

Example: L1 regularized sum of squared errors

$$\min_{\mathbf{w}} \sum_{i=1}^{t} \|X_{i:}\mathbf{w} - \mathbf{y}_i\|_2^2 + \beta\|\mathbf{w}\|_1$$

- Fancy name: LASSO (Tibshirani)

Comparison of L_1 and L_2 regularization

L_2 regularizer

Recall: for any loss function

$$\begin{aligned} L_2 \text{ regularization} &\implies \text{Representer Theorem} \\ &\implies \text{Automatically get equivalent dual formulation.} \\ &\implies \text{Automatically get equivalent kernelized formulatio} \end{aligned}$$

Also convex and differentiable

L_1 regularizer

- L_1 regularization $\not\Longrightarrow$ Representer theorem
- Do NOT get kernelization automatically.

Also ctonvex, but non-differentiable

So why use L_1 regularization?

L_1 regularization

- Get Sparsity instead
- Exact zero weights instead of small weights
- $\Longrightarrow$ Automatic basis selection.

Background: Sparse vectors

- When elements of a vector have mostly zero values, the vector is said to be sparse.
- The benefit of having a sparse weight vector $\mathbf{w}$ is that many features drop.

So why use L_1 regularization?

L_1 regularization

- Get Sparsity instead $\implies$ Automatic basis selection.

Background: Sparse vectors

- When elements of a vector have mostly zero values, the vector is said to be sparse.
- The benefit of having a sparse weight vector $\mathbf{w}$ is that many features drop.

L_1 regularizer

Geometry

- Note: $|\mathbf{w}|_1 = \sum_j |\mathbf{w}_j|$
- L_1 norm of a vector is sum of absolute values of components of that vector.
- Note: Absolute value function is non-differentiable at 0.

$$\frac{\partial \|\mathbf{w}\|_1}{\partial w_j} = \begin{cases} 1, \text{if } w_j > 0 \\ -1, \text{ if } w_j < 0 \\ \text{undefined, if } w_j = 0 \end{cases}$$

- For a differentiable function any point in the domain where derivative is zero is a local min.
- For a non-differentiable function any point in the domain where 0 is one of its sub-gradients is a local min.

To be continued