CPSC 340: Machine Learning and Data Mining

More Regularization

Fall 2018
Admin

• **Midterm** is tomorrow.
  – October 18th at 6:30pm.
  – Last names starting with A-L: BUCH A102.
  – Last names starting with M-Z: BUCH A104.
  – 80 minutes.
  – Closed-book.
  – One doubled-sided ‘cheat sheet’ for midterm.
  – Auditors do not take the midterm.
  – Mike will go over midterm questions/solutions in his lecture Friday.

• There will be two types of questions on the midterm:
  – ‘Technical’ questions requiring things like pseudo-code or derivations.
    • Similar to assignment questions, and will only be on topics related to those in assignments.
  – ‘Conceptual’ questions testing understanding of key concepts.
    • All lecture slide material except “bonus slides” is fair game here.
Last Time: L2-Regularization

• We discussed regularization:
  – Adding a continuous penalty on the model complexity:
    \[ f(w) = \frac{1}{2} \| Xw - y \|^2 + \frac{\lambda}{2} \| w \|^2 \]
  – Best parameter \( \lambda \) almost always leads to improved test error.
    • L2-regularized least squares is also known as “ridge regression”.
    • Can be solved as a linear system like least squares.

– Numerous other benefits:
  • Solution is unique, less sensitive to data, gradient descent converges faster.
Parametric vs. Non-Parametric Transforms

• We’ve been using linear models with polynomial bases:

\[ y_i = w_0 + w_1 + w_2 + w_3 + w_4 \]

• But polynomials are not the only possible bases:
  – Exponentials, logarithms, trigonometric functions, etc.
  – The right basis will vastly improve performance.
  – If we use the wrong basis, our accuracy is limited even with lots of data.
  – But the right basis may not be obvious.
Parametric vs. Non-Parametric Transforms

• We’ve been using linear models with polynomial bases:

\[ y_i = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 \]

• Alternative is non-parametric bases:
  – Size of basis (number of features) grows with ‘n’.
  – Model gets more complicated as you get more data.
  – Can model complicated functions where you don’t know the right basis.
    • With enough data.
  – Classic example is “Gaussian RBFs”.
Gaussian RBFs are universal approximators (compact subsets of $\mathbb{R}^d$)

- Enough bumps can approximate any continuous function to arbitrary precision.
- Achieve optimal test error as ‘n’ goes to infinity.
Gaussian RBFs: A Sum of “Bumps”

• Polynomial fit:

• Constructing a function from bumps (“smooth histogram”):

• Bonus slides: challenges of “far from data” (and future) predictions.
Gaussian RBF Parameters

• Some obvious questions:
  1. How many bumps should we use?
  2. Where should the bumps be centered?
  3. How high should the bumps go?
  4. How wide should the bumps be?

• The usual answers:
  1. We use ‘n’ bumps (non-parametric basis).
  2. Each bump is centered on one training example \( x_i \).
  3. Fitting regression weights ‘w’ gives us the heights (and signs).
  4. The width is a hyper-parameter (narrow bumps == complicated model).
Gaussian RBFs: Formal Details

• What is a radial basis functions (RBFs)?
  – A set of non-parametric bases that depend on distances to training points.

\[
R_{\text{place}} \quad X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \text{by} \quad Z = \begin{bmatrix} g(\|x_1-x_1\|) & g(\|x_1-x_2\|) & \cdots & g(\|x_1-x_n\|) \\ g(\|x_2-x_1\|) & g(\|x_2-x_2\|) & \cdots & g(\|x_2-x_n\|) \\ \vdots & \vdots & \ddots & \vdots \\ g(\|x_n-x_1\|) & g(\|x_n-x_2\|) & \cdots & g(\|x_n-x_n\|) \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}
\]

– Most common ‘g’ is Gaussian RBF:
  \[
g(\varepsilon) = e^{\frac{-\varepsilon^2}{2\sigma^2}}
\]

• Variance \( \sigma^2 \) is a hyper-parameter controlling “width”.
  – This affects fundamental trade-off (set it using a validation set).
What is a radial basis functions (RBFs)?

- A set of non-parametric bases that depend on distances to training points.

\[ X = \begin{bmatrix} \vdots \\ \end{bmatrix} \rightarrow \begin{bmatrix} \vdots \\ \end{bmatrix} \]

Replace \( X \) by \( Z \):

\[ Z = \begin{bmatrix} g(||x_1 - x_1||) & g(||x_1 - x_2||) & \cdots & g(||x_1 - x_n||) \\ g(||x_2 - x_1||) & g(||x_2 - x_2||) & \cdots & g(||x_2 - x_n||) \\ \vdots & \vdots & \ddots & \vdots \\ g(||x_n - x_1||) & g(||x_n - x_2||) & \cdots & g(||x_n - x_n||) \end{bmatrix} \]

To make predictions on \( \tilde{X} = \begin{bmatrix} \vdots \\ \end{bmatrix} \) use \( \tilde{Z} = \begin{bmatrix} \vdots \\ \end{bmatrix} \):
Gaussian RBFs: Pseudo-Code

Constructing Gaussian RBFs given data 'X' and hyper-parameter $\sigma$:

$$Z = \text{zeros}(n, n)$$

for $i1$ in 1:n
  for $i2$ in 1:n
    $$Z[i1, i2] = \exp(-\text{norm}(X[i1, :] - X[i2, :])^2 / 2\sigma^2)$$

With test data $\tilde{X}$: Form $\tilde{Z}$ based on distances to training examples.
Non-Parametric Basis: RBFs

• Least squares with Gaussian RBFs for different $\sigma$ values:

\[
Z = \begin{pmatrix}
    \mathbf{x}_1 \\
    \mathbf{x}_2 \\
    \vdots \\
    \mathbf{x}_n
\end{pmatrix}
\begin{pmatrix}
g(\|\mathbf{x}_i - \mathbf{x}_1\|) \\
g(\|\mathbf{x}_i - \mathbf{x}_2\|) \\
\vdots \\
g(\|\mathbf{x}_i - \mathbf{x}_n\|)
\end{pmatrix}
\]

This reverts to linear regression instead of 0 away from data.
RBFs and Regularization

• **Gaussian Radial basis functions (RBFs) predictions:**
  \[
  \hat{y}_i = w_1 \exp\left(-\frac{\|x_i - x_1\|^2}{2\sigma^2}\right) + w_2 \exp\left(-\frac{\|x_i - x_2\|^2}{2\sigma^2}\right) + \ldots + w_n \exp\left(-\frac{\|x_i - x_n\|^2}{2\sigma^2}\right)
  \]
  \[
  = \sum_{j=1}^{n} w_j \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)
  \]
  
  — Flexible bases that can **model any continuous function**.
  — But with ‘n’ data points RBFs have ‘n’ basis functions.

• How do we avoid overfitting with this _huge number of features_?
  — We **regularize ‘w’** and use **validation error** to choose \(\sigma\) and \(\lambda\).
RBFs, Regularization, and Validation

• A model that is hard to beat:
  – RBF basis with L2-regularization and cross-validation to choose $\sigma$ and $\lambda$.
  – Flexible non-parametric basis, magic of regularization, and tuning for test error.

For each value of $\lambda$ and $\sigma$:
- Compute $Z$ on training data (and $\sigma$)
- Compute best $\nu$: $\nu = (Z^\top Z + \lambda I)^{-1} Z^\top y$
- Compute $\hat{Z}$ on validation data (using train data distances)
- Make predictions $\hat{y} = \hat{Z} \nu$
- Compute validation error $\|\hat{y} - \hat{y}\|^2$
RBFs, Regularization, and Validation

• A model that is hard to beat:
  – RBF basis with L2-regularization and cross-validation to choose $\sigma$ and $\lambda$.
  – Flexible non-parametric basis, magic of regularization, and tuning for test error!

  – Expensive at test time: needs distance to all training examples.
Hyper-Parameter Optimization

• In this setting we have 2 hyper-parameters ($\sigma$ and $\lambda$).
• More complicated models have even more hyper-parameters.
  – This makes searching all values expensive (increases over-fitting risk).

• Leads to the problem of hyper-parameter optimization.
  – Try to efficiently find “best” hyper-parameters.

• Simplest approaches:
  – Exhaustive search: try all combinations among a fixed set of $\sigma$ and $\lambda$ values.
  – Random search: try random values.
Hyper-Parameter Optimization

• Other common hyper-parameter optimization methods:
  – Exhaustive search with pruning:
    • If it “looks” like test error is getting worse as you decrease $\lambda$, stop decreasing it.
  – Coordinate search:
    • Optimize one hyper-parameter at a time, keeping the others fixed.
    • Repeatedly go through the hyper-parameters
  – Stochastic local search:
    • Generic global optimization methods (simulated annealing, genetic algorithms, etc.).
  – Bayesian optimization (Mike’s PhD research topic):
    • Use RBF regression to build model of how hyper-parameters affect validation error.
    • Try the best guess based on the model.
(pause)
Previously: Search and Score

• We talked about search and score for feature selection:
  – Define a “score” and “search” for features with the best score.
• Usual scores count the number of non-zeroes (“L0-norm”):

\[ f(w) = \frac{1}{2} \| \mathbf{X}w - \mathbf{y} \|^2 + \alpha \text{L0-norm}(w) \]

• But it’s hard to find the ‘w’ minimizing this objective.
• We discussed forward selection, but requires fitting O(d^2) models.
  – For robust regression, need to run gradient descent O(d^2) times.
  – With regularization, need to search for lambda O(d^2) times.
L1-Regularization

• Consider regularizing by the L1-norm:

\[ f(w) = \frac{1}{2} \| Xw - y \|^2 + \lambda \| w \|_1 \]

• Like L2-norm, it’s **convex** and improves our test error.

• Like L0-norm, it **encourages** elements of ‘w’ to be exactly zero.

• L1-regularization **simultaneously regularizes and selects features**.
  – Very fast alternative to search and score.
  – Sometimes called “LASSO” regularization.
L2-regularization vs. L1-regularization

- Regularization path of $w_i$ values as ‘$\lambda$’ varies:
Regularizers and Sparsity

• L1-regularization give sparsity but L2-regularization doesn’t.
  – But don’t they both shrink variables to zero?
• Consider problem where 3 vectors can get minimum training error:

\[
\mathbf{w}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0.02 & 0 & 0 \end{bmatrix} \quad \mathbf{w}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.02 & 0 \end{bmatrix} \quad \mathbf{w}_3 = \begin{bmatrix} 99.99 & 0 & 0 \\ 0.02 & 0 & 0 \end{bmatrix}
\]

• Without regularization, we could choose any of these 3.
  – They all have same error, so regularization will “break tie”.
• With L0-regularization, we would choose \( \mathbf{w}_2 \):

\[
\|\mathbf{w}_1\|_0 = 2 \quad \|\mathbf{w}_2\|_0 = 1 \quad \|\mathbf{w}_3\|_0 = 2
\]
Regularizers and Sparsity

- L1-regularization give sparsity but L2-regularization doesn’t.
  - But don’t they both shrink variables to zero?
- Consider problem where 3 vectors can get minimum training error:

\[ w^1 = \begin{bmatrix} 100 \\ 0.02 \end{bmatrix} \quad w^2 = \begin{bmatrix} 100 \\ 0 \end{bmatrix} \quad w^3 = \begin{bmatrix} 99.99 \\ 0.02 \end{bmatrix} \]

- With L2-regularization, we would choose \( w^3 \):

\[
\|w^1\|^2 = 100^2 + 0.02^2 = 10000.0004 \\
\|w^2\|^2 = 100^2 + 0^2 = 10000 \\
\|w^3\|^2 = 99.99^2 + 0.02^2 = 9998.0005
\]

- L2-regularization focuses on decreasing largest (makes \( w_j \) similar).
Regularizers and Sparsity

• L1-regularization give sparsity but L2-regularization doesn’t.
  – But don’t they both shrink variables to zero?

• Consider problem where 3 vectors can get minimum training error:

\[
\begin{align*}
\mathbf{w}^1 &= \begin{bmatrix} 100 \\ 0.02 \end{bmatrix} \\
\mathbf{w}^2 &= \begin{bmatrix} 100 \\ 0 \end{bmatrix} \\
\mathbf{w}^3 &= \begin{bmatrix} 99.99 \\ 0.02 \end{bmatrix}
\end{align*}
\]

• With L1-regularization, we would choose \( \mathbf{w}^2 \):

\[
\begin{align*}
\|\mathbf{w}^1\|_1 &= 100 + 0.02 = 100.02 \\
\|\mathbf{w}^2\|_1 &= 100 + 0 = 100 \\
\|\mathbf{w}^3\|_1 &= 99.99 + 0.02 = 100.01
\end{align*}
\]

• L1-regularization focuses on decreasing all \( w_j \) until they are 0.
Why doesn’t L2-Regularization set variables to 0?

• Consider an L2-regularized least squares problem with 1 feature:

\[
f(w) = \sum_{i=1}^{n} (w x_i - y_i)^2 + \frac{\lambda}{2} w^2
\]

• Let’s solve for the optimal ‘w’:

\[
f'(w) = \sum_{i=1}^{n} x_i (w x_i - y_i) + \lambda w
\]

Set equal to 0:

\[
\sum_{i=1}^{n} x_i^2 w - \sum_{i=1}^{n} x_i y_i + \lambda w = 0
\]

or

\[
w = \frac{y^T x}{\|x\|^2 + \lambda}
\]

• So as \( \lambda \) gets bigger, ‘w’ converges to 0.

• However, for all finite \( \lambda \) ‘w’ will be non-zero unless \( y^T x = 0 \) exactly.
  – But it’s very unlikely that \( y^T x \) will be exactly zero.
Why doesn’t L2-Regularization set variables to 0?

- Small $\lambda$
  - Solution further from zero

- Big $\lambda$
  - Solution closer to zero (but not exactly 0)
Why does L1-Regularization set things to 0?

- **Small $\lambda$**
  - Solution nonzero
  - (minimum of left parabola is past origin, but right parabola is not)

- **Big $\lambda$**
  - Solution exactly zero
  - (minimum of both parabola are past the origin)
Why does L1-Regularization set things to 0?

• Consider an L1-regularized least squares problem with 1 feature:
  \[ f(w) = \frac{1}{2} \sum_{i=1}^{n} (w x_i - y_i)^2 + \lambda |w| \]

• If \((w = 0)\), then “left” limit and “right“ limit are given by:
  \[
  f^-(0) = \sum_{i=1}^{n} x_i (0 x_i - y_i) - \lambda = \sum_{i=1}^{n} x_i y_i - \lambda \\
  f^+(0) = \sum_{i=1}^{n} x_i (0 x_i - y_i) + \lambda = \sum_{i=1}^{n} x_i y_i + \lambda
  \]

• So which direction should “gradient descent” go in?
  \[
  -f^-(0) = -y^T x + \lambda \quad \text{If these are positive } (-y^T x > \lambda), \quad \text{we can improve by increasing } w. \\
  -f^+(0) = -y^T x - \lambda \quad \text{If these are negative } (y^T x > \lambda), \quad \text{we can improve by decreasing } w.
  \]
  \[
  \begin{cases}
  \text{But if left and right "gradient descent" directions point in opposite directions (}\left|y^T x\right| \leq \lambda, \text{ minimum is 0}.
  \end{cases}
  \]
L2-regularization vs. L1-regularization

• So with 1 feature:
  – L2-regularization only sets ‘w’ to 0 if \( y^T x = 0 \).
    • There is only a single possible \( y^T x \) value where the variable gets set to zero.
    • And \( \lambda \) has nothing to do with the sparsity.

  – L1-regularization sets ‘w’ to 0 if \( |y^T x| \leq \lambda \).
    • There is a range of possible \( y^T x \) values where the variable gets set to zero.
    • And increasing \( \lambda \) increases the sparsity since the range of \( y^T x \) grows.

• Note that it’s important that the function is non-differentiable:
  – Differentiable regularizers penalizing size would need \( y^T x = 0 \) for sparsity.
L2-Regularization vs. L1-Regularization

• L2-Regularization:
  – Insensitive to changes in data.
  – Decreased variance:
    • Lower test error.
  – Closed-form solution.
  – Solution is unique.
  – All ‘w’ tend to be non-zero.
  – Can learn with *linear* number of irrelevant features.
    • E.g., only $O(d)$ relevant features.

• L1-Regularization:
  – Insensitive to changes in data.
  – Decreased variance:
    • Lower test error.
  – Requires iterative solver.
  – Solution is not unique.
  – Many ‘w’ tend to be zero.
  – Can learn with *exponential* number of irrelevant features.
    • E.g., only $O(\log(d))$ relevant features.

*Paper on this result by Andrew Ng*
L1-loss vs. L1-regularization

• Don’t confuse the L1 loss with L1-regularization!
  – L1-loss is robust to outlier data points.
    • You can use this instead of removing outliers.
  – L1-regularization is robust to irrelevant features.
    • You can use this instead of removing features.

• And note that you can be robust to outliers and select features:

\[
f(w) = \| Xw - y \|_1 + \gamma \| w \|_1
\]

• Why aren’t we smoothing and using “Huber regularization”?
  – Huber regularizer is still robust to irrelevant features.
  – But it’s the non-smoothness that sets weights to exactly 0.
Summary

- **Radial basis functions:**
  - Non-parametric bases that can model any function.

- **L1-regularization:**
  - Simultaneous regularization and feature selection.
  - Robust to having lots of irrelevant features.

- Next time: are we really going to use regression for classification?
Sparsity and Least Squares

• Consider 1D least squares objective:
  \[ f(w) = \frac{1}{2} \sum_{i=1}^{n} (w x_i - y_i)^2 \]

• This is a convex 1D quadratic function of ‘w’ (i.e., a parabola):

• This variable does not look relevant (minimum is close to 0).
  – But for finite ‘n’ the minimum is unlikely to be exactly zero.
Sparsity and L0-Regularization

• Consider 1D **L0-regularized** least squares objective:

\[
f(w) = \frac{1}{2} \sum_{i=1}^{n} (w x_i - y_i)^2 + \lambda \|w\|_0
\]

• This is a convex 1D quadratic function but with a discontinuity at 0:

• L0-regularized minimum is often exactly at the ‘discontinuity’ at 0:
  – Sets the feature to exactly 0 (does feature selection), but is **non-convex**.
Sparsity and L2-Regularization

• Consider 1D L2-regularized least squares objective:

\[ f(w) = \frac{1}{2} \sum_{i=1}^{n} (w x_i - y_i)^2 + \frac{\lambda}{2} w^2 \]

• This is a convex 1D quadratic function of \( w \) (i.e., a parabola):

• L2-regularization moves it closer to zero, but not all the way to zero.
  – It doesn’t do feature selection (“penalty goes to 0 as slope goes to 0”).

\[ f'(0) = 0 \quad \text{only if} \sum_{i=1}^{n} x_i y_i = 0 \]
Sparsity and L1-Regularization

• Consider 1D L1-regularized least squares objective:

\[ f(w) = \frac{1}{2} \sum_{i=1}^{n} (w x_i - y_i)^2 + \lambda |w| \]

• This is a convex piecewise-quadratic function of ‘w’ with ‘kink’ at 0:

• L1-regularization tends to set variables to exactly 0 (feature selection).
  – Penalty on slope is \( \lambda \) even if you are close to zero.
  – Big \( \lambda \) selects few features, small \( \lambda \) allows many features.
L1-Loss vs. Huber Loss

• The same reasoning tells us the difference between the L1 *loss* and the Huber loss. They are very similar in that they both grow linearly far away from 0. So both are both robust but...
  – With the L1 loss the model often passes exactly through some points.
  – With Huber the model doesn’t necessarily pass through any points.

• Why? With L1-regularization we were causing the elements of ’w’ to be exactly 0. Analogously, with the L1-loss we cause the elements of ‘r’ (the residual) to be exactly zero. But zero residual for an example means you pass through that example exactly.
Non-Uniqueness of L1-Regularized Solution

• How can L1-regularized least squares solution not be unique?
  – Isn’t it convex?

• Convexity implies that minimum value of $f(w)$ is unique (if exists), but there may be multiple ‘w’ values that achieve the minimum.

• Consider L1-regularized least squares with $d=2$, where feature 2 is a copy of a feature 1. For a solution $(w_1, w_2)$ we have:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} = w_1 x_{i1} + w_2 x_{i1} = (w_1 + w_2) x_{i1}$$

• So we can get the same squared error with different $w_1$ and $w_2$ values that have the same sum. Further, if neither $w_1$ or $w_2$ changes sign, then $|w_1| + |w_2|$ will be the same so the new $w_1$ and $w_2$ will be a solution.
Splines in 1D

• For 1D interpolation, alternative to polynomials/RBFs are splines:
  – Use a polynomial in the region between each data point.
  – Constrain some derivatives of the polynomials to yield a unique solution.

• Most common example is cubic spline:
  – Use a degree-3 polynomial between each pair of points.
  – Enforce that \( f'(x) \) and \( f''(x) \) of polynomials agree at all point.
  – “Natural” spline also enforces \( f''(x) = 0 \) for smallest and largest \( x \).

• Non-trivial fact: natural cubic splines are sum of:
  – Y-intercept.
  – Linear basis.
  – RBFs with \( g(\epsilon) = \epsilon^3 \).
    • Different than Gaussian RBF because it increases with distance.

http://www.physics.arizona.edu/~restrepo/475A/Notes/sourcea/node35.html
Splines in Higher Dimensions

• Splines generalize to higher dimensions if data lies on a grid.
  – Many methods exist for grid-structured data (linear, cubic, splines, etc.).
  – For more general (“scattered”) data, there isn’t a natural generalization.

• Common 2D “scattered” data interpolation is thin-plate splines:
  – Based on curve made when bending sheets of metal.
  – Corresponds to RBFs with \( g(\varepsilon) = \varepsilon^2 \log(\varepsilon) \).

• Natural splines and thin-plate splines: special cases of “polyharmonic” splines:
  – Less sensitive to parameters than Gaussian RBF.

http://step.polymtl.ca/~rv101/thinplates/
L2-Regularization vs. L1-Regularization

- L2-regularization conceptually restricts ‘w’ to a ball.

Minimizing $\frac{1}{2} \|Xw - y\|^2 + \frac{\lambda}{2} \|w\|^2$ is equivalent to minimizing $\frac{1}{2} \|Xw - y\|^2$ subject to the constraint that $\|w\| \leq \gamma$, for some value $\gamma$. 
L2-regularization vs. L1-regularization

• L2-regularization conceptually restricts ‘w’ to a ball.

• L1-regularization restricts to the L1 “ball”:
  – Solutions tend to be at corners where $w_j$ are zero.

Related Infinite Series video