CSCI567 Machine Learning (Spring 2021)

Sirisha Rambhatla

University of Southern California

Jan 22, 2021

Outline

1. Logistics
2. Review of last lecture
3. Linear regression

Outline

1. Logistics
2. Review of last lecture
3. Linear regression

Logistics

- HW 0 is due today.
- HW 1 will be released today.
- Will be releasing the schedule of lectures.

Outline

1. Logistics
2. Review of last lecture
3. Linear regression

Multi-class classification

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{N}}, y_{\mathsf{N}})\}$
- Each $\boldsymbol{x}_n \in \mathbb{R}^{\mathsf{D}}$ is called a feature vector.
- Each $y_n \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$ is called a label/class/category.
- They are used to learn $f : \mathbb{R}^{\mathsf{D}} \to [\mathsf{C}]$ for future prediction.

Special case: binary classification

- Number of classes: $\mathsf{C} = 2$
- Conventional labels: $\{0, 1\}$ or $\{-1, +1\}$

K-NNC: predict the majority label within the K-nearest neighbor set

Datasets

Training data

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{N}}, y_{\mathsf{N}})\}$
- They are used to learn $f(\cdot)$

Test data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{M}}, y_{\mathsf{M}})\}$
- They are used to evaluate how well $f(\cdot)$ will do.

Development/Validation data

- L samples/instances: $\mathcal{D}^{\text{DEV}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{L}}, y_{\mathsf{L}})\}$
- They are used to optimize hyper-parameter(s).

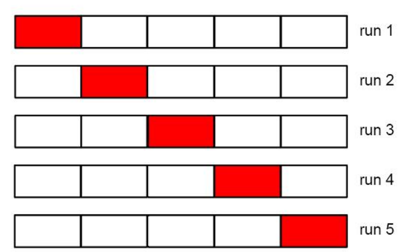
These three sets should *not* overlap!

S-fold Cross-validation

What if we do not have a development set?

- Split the training data into S equal parts.
- Use each part *in turn* as a development dataset and use the others as a training dataset.
- Choose the hyper-parameter leading to best *average* performance.

S $= 5$: 5-fold cross validation



Special case: S $=$ N, called leave-one-out.

High level picture

Typical steps of developing a machine learning system:

- Collect data, split into training, development, and test sets.
- *Train a model with a machine learning algorithm.* Most often we apply cross-validation to tune hyper-parameters.
- Evaluate using the test data and report performance.
- Use the model to predict future/make decisions.

How to do the *red part* exactly?

Outline

1. Logistics
2. Review of last lecture
3. Linear regression
   - Motivation
   - Setup and Algorithm
   - Discussions

Regression

Predicting a continuous outcome variable using past observations

- Predicting future temperature (lecture 1)
- Predicting the amount of rainfall
- Predicting the demand of a product
- Predicting the sale price of a house
- ...

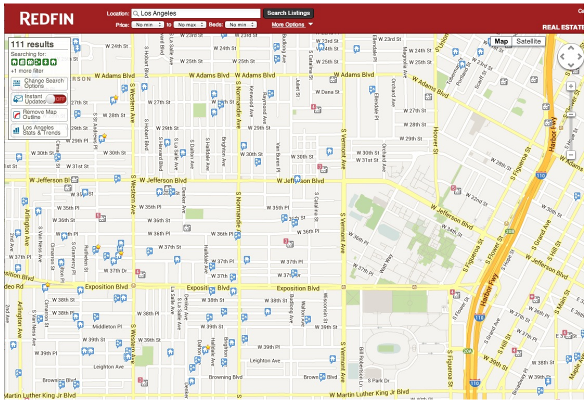
Key difference from classification

- continuous vs discrete
- measure *prediction errors* differently.
- lead to quite different learning algorithms.

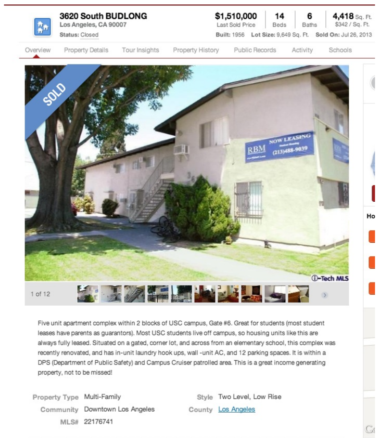
Linear Regression: regression with linear models

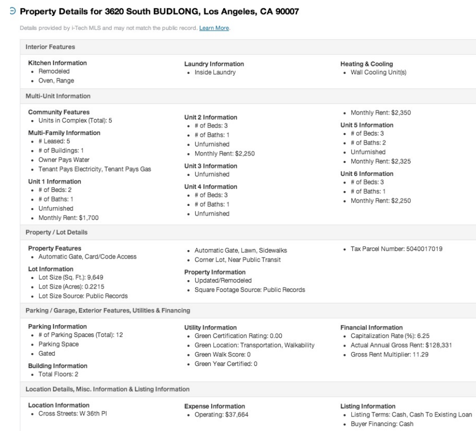
Ex: Predicting the sale price of a house

Retrieve historical sales records (training data)

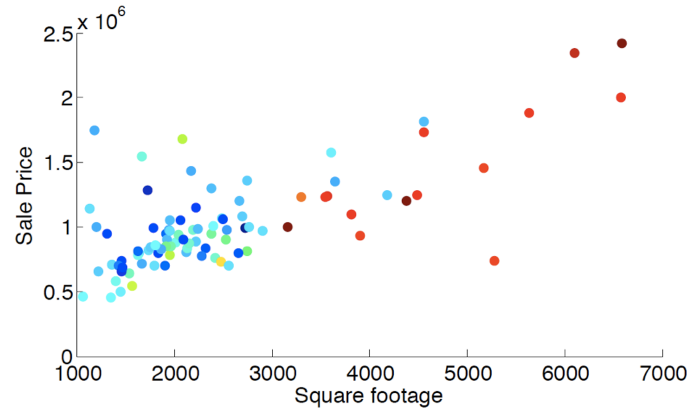


Features used to predict



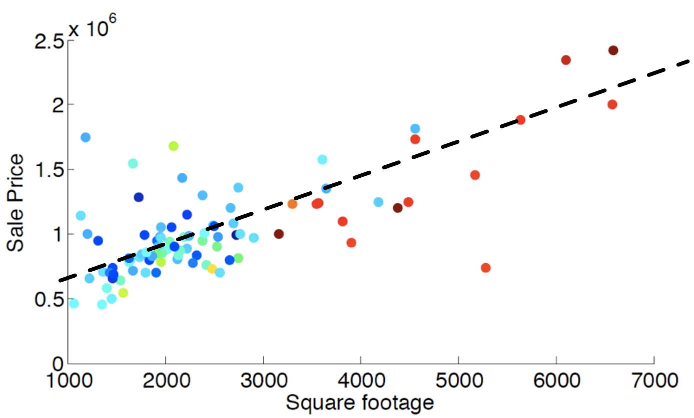


Correlation between square footage and sale price



Possibly linear relationship

Sale price ≈ **price_per_sqft** × square_footage + **fixed_expense**
(*slope*) (*intercept*)



How to learn the unknown parameters?

How to measure error for one prediction?

- The classification error (0-1 loss, i.e. *right* or *wrong*) is *inappropriate* for continuous outcomes.
- We can look at
  - *absolute* error: | prediction - sale price |
  - or *squared* error: (prediction - sale price)^2 (**most common**)

Goal: pick the model (unknown parameters) that minimizes the average/total prediction error, but *on what set*?

- test set, ideal but we *cannot use test set while training*
- training set ✓

Example

Predicted price = **price_per_sqft** × square_footage + **fixed_expense**

one model: price_per_sqft = 0.3K, fixed_expense = 210K

sqft	sale price (K)	prediction (K)	squared error
2000	810	810	0
2100	907	840	67^2
1100	312	540	228^2
5500	2,600	1,860	740^2
···	···	···	···
Total			$0 + 67^2 + 228^2 + 740^2 + \cdots$

Adjust price_per_sqft and fixed_expense such that the total squared error is minimized.

Formal setup for linear regression

Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$

Linear model: $f : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}$, with $f(\boldsymbol{x}) = w_0 + \sum_{d=1}^{D} w_d x_d = w_0 + \boldsymbol{w}^{\mathbf{T}}\boldsymbol{x}$ (superscript ^{T} stands for transpose), i.e. a *hyper-plane* parametrized by

- $\boldsymbol{w} = [w_1 \ w_2 \ \cdots \ w_{\mathsf{D}}]^{\mathrm{T}}$ (weights, weight vector, parameter vector, etc)
- bias w_0

NOTE: for notation convenience, very often we

- append 1 to each x as the first feature: $\tilde{\boldsymbol{x}} = [1 \ x_1 \ x_2 \ \ldots \ x_{\mathsf{D}}]^{\mathrm{T}}$
- let $\tilde{\boldsymbol{w}} = [w_0 \ w_1 \ w_2 \ \cdots \ w_{\mathsf{D}}]^{\mathrm{T}}$, a concise representation of all $D+1$ parameters
- the model becomes simply $f(\boldsymbol{x}) = \tilde{\boldsymbol{w}}^{\mathbf{T}}\tilde{\boldsymbol{x}}$
- sometimes just use $\boldsymbol{w}, \boldsymbol{x}, \mathsf{D}$ for $\tilde{\boldsymbol{w}}, \tilde{\boldsymbol{x}}, \mathsf{D}+1$!

Goal

Minimize total squared error (note that $\tilde{\boldsymbol{x}}_n^{\mathrm{T}}\tilde{\boldsymbol{w}} = \tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{x}}_n$)

- **Residual Sum of Squares** (RSS), a function of $\tilde{\boldsymbol{w}}$

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (f(\boldsymbol{x}_n) - y_n)^2 = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}}\tilde{\boldsymbol{w}} - y_n)^2$$

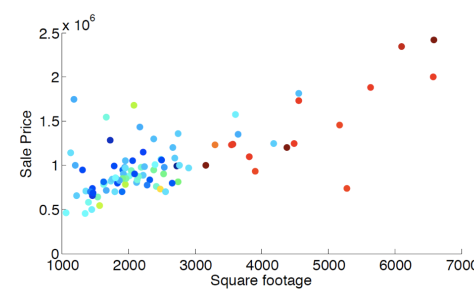
- find $\tilde{\boldsymbol{w}}^* = \underset{\tilde{\boldsymbol{w}} \in \mathbb{R}^{\mathsf{D}+1}}{\operatorname{argmin}} \mathrm{RSS}(\tilde{\boldsymbol{w}})$, i.e. **least (mean) squares solution** (more generally called **empirical risk minimizer**)

- *reduce machine learning to optimization*

- in principle can apply any optimization algorithm, but linear regression admits a *closed-form solution*

Warm-up: D = 0

Only one parameter w_0: constant prediction $f(x) = w_0$



f is a horizontal line, where should it be?

Warm-up: D = 0

Optimization objective becomes

$$\mathrm{RSS}(w_0) = \sum_n (w_0 - y_n)^2 \qquad \text{(it's a } \textit{quadratic } aw_0^2 + bw_0 + c\text{)}$$

$$= Nw_0^2 - 2\left(\sum_n y_n\right) w_0 + \text{cnt.}$$

$$= N\left(w_0 - \frac{1}{N}\sum_n y_n\right)^2 + \text{cnt.}$$

It is clear that $w_0^* = \frac{1}{N}\sum_n y_n$, i.e. the **average**

Exercise: what if we use absolute error instead of squared error?

Warm-up: D = 1

Optimization objective becomes

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (w_0 + w_1 x_n - y_n)^2$$

General approach: find *stationary points*, i.e., points with *zero gradient*

$$\begin{cases} \frac{\partial \mathrm{RSS}(\tilde{\boldsymbol{w}})}{\partial w_0} = 0 \\ \frac{\partial \mathrm{RSS}(\tilde{\boldsymbol{w}})}{\partial w_1} = 0 \end{cases} \Rightarrow \begin{array}{ll} \sum_n (w_0 + w_1 x_n - y_n) & = 0 \\ \sum_n (w_0 + w_1 x_n - y_n) x_n & = 0 \end{array}$$

$$\Rightarrow \begin{array}{ll} N w_0 + w_1 \sum_n x_n & = \sum_n y_n \\ w_0 \sum_n x_n + w_1 \sum_n x_n^2 & = \sum_n y_n x_n \end{array} \qquad \text{(}\textbf{a linear system}\text{)}$$

$$\Rightarrow \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} = \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}$$

Least square solution for D = 1

$$\Rightarrow \begin{pmatrix} w_0^* \\ w_1^* \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}$$

(assuming the matrix is invertible)

Are stationary points minimizers?

- yes for **convex** objectives (RSS is convex in $\tilde{\boldsymbol{w}}$)
- not true in general

General least square solution

Objective

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n)^2$$

Again, find stationary points (**multivariate calculus**)

$$\nabla \mathrm{RSS}(\tilde{\boldsymbol{w}}) = 2 \sum_n \tilde{\boldsymbol{x}}_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - y_n) \propto \left(\sum_n \tilde{\boldsymbol{x}}_n \tilde{\boldsymbol{x}}_n^{\mathrm{T}}\right) \tilde{\boldsymbol{w}} - \sum_n \tilde{\boldsymbol{x}}_n y_n$$
$$= (\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}}) \tilde{\boldsymbol{w}} - \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y} = \boldsymbol{0}$$

where

$$\tilde{\boldsymbol{X}} = \begin{pmatrix} \tilde{\boldsymbol{x}}_1^{\mathrm{T}} \\ \tilde{\boldsymbol{x}}_2^{\mathrm{T}} \\ \vdots \\ \tilde{\boldsymbol{x}}_{\mathrm{N}}^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{\mathrm{N} \times (D+1)}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\mathrm{N}} \end{pmatrix} \in \mathbb{R}^{\mathrm{N}}$$

General least square solution

$$(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})\tilde{\boldsymbol{w}} - \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{0} \quad \Rightarrow \quad \tilde{\boldsymbol{w}}^* = (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

assuming $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ is invertible for now.

Again by convexity $\tilde{\boldsymbol{w}}^*$ is the minimizer of RSS.

Verify the solution when $\mathsf{D} = 1$:

$$\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_{\mathsf{N}} \end{pmatrix} \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \cdots \\ 1 & x_{\mathsf{N}} \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}$$

when $\mathsf{D} = 0$: $(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1} = \frac{1}{N}$, $\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} = \sum_n y_n$

Another approach

RSS is a quadratic:

$$\mathrm{RSS}(\tilde{\boldsymbol{w}}) = \sum_n (\tilde{\boldsymbol{x}}_n^{\mathrm{T}}\tilde{\boldsymbol{w}} - y_n)^2 = \|\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\|_2^2$$
$$= \left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right)^{\mathrm{T}}\left(\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}\right)$$
$$= \tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} + \text{cnt.}$$
$$= \left(\tilde{\boldsymbol{w}} - (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\left(\tilde{\boldsymbol{w}} - (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right) + \text{cnt.}$$

Note: $\boldsymbol{u}^{\mathrm{T}}\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\boldsymbol{u} = \left(\tilde{\boldsymbol{X}}\boldsymbol{u}\right)^{\mathrm{T}}\tilde{\boldsymbol{X}}\boldsymbol{u} = \|\tilde{\boldsymbol{X}}\boldsymbol{u}\|_2^2 \geq 0$ and is 0 if $\boldsymbol{u} = 0$.
So $\tilde{\boldsymbol{w}}^* = (\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$ is the minimizer.

Computational complexity

Bottleneck of computing

$$\tilde{\boldsymbol{w}}^* = \left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

is to invert the matrix $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} \in \mathbb{R}^{(\mathsf{D}+1)\times(\mathsf{D}+1)}$

- aka *pseudo-inverse*[1] denoted by $(\cdot)^\dagger$, i.e. $\tilde{\boldsymbol{X}}^\dagger = \left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}$
- naively need $O(\mathsf{D}^3)$ time
- there are many faster approaches

see `https://en.wikipedia.org/wiki/Moore-Penrose_inverse`

What if $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ is not invertible

What does that imply?

Recall $\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)\boldsymbol{w}^* = \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$. If $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ not invertible, this equation aka *Normal Equations* has

- infinitely many solutions ($\Rightarrow$ infinitely many minimizers)
- This is because *Normal Equations* are always *consistent*[2] meaning a solution *always* exists! It may not be unique though.

See `https://sites.math.washington.edu/~burke/crs/308/LeastSquares.pdf`

What if $\tilde{X}^T\tilde{X}$ is not invertible

Why would that happen?

One situation: $N < D+1$, i.e. not enough data to estimate all parameters.

Example: $D = N = 1$

sqft	sale price
1000	500K

Any line passing through this single point is a minimizer of RSS.

How to resolve this issue?

Intuition: what does inverting $\tilde{X}^T\tilde{X}$ do?

eigendecomposition: $$\tilde{X}^T\tilde{X} = U^T \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_D & 0 \\ 0 & \cdots & 0 & \lambda_{D+1} \end{bmatrix} U$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_{D+1} \geq 0$ are **eigenvalues**.

inverse: $$(\tilde{X}^T\tilde{X})^{-1} = U^T \begin{bmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_D} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{D+1}} \end{bmatrix} U$$

i.e. just inverse the eigenvalues

How to solve this problem?

Non-invertible $\Rightarrow$ some eigenvalues are 0.

One natural fix: add something positive

$$\tilde{X}^T\tilde{X} + \lambda I = U^T \begin{bmatrix} \lambda_1+\lambda & 0 & \cdots & 0 \\ 0 & \lambda_2+\lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \lambda_D+\lambda & 0 \\ 0 & \cdots & 0 & \lambda_{D+1}+\lambda \end{bmatrix} U$$

where $\lambda > 0$ and I is the identity matrix. Now it is invertible:

$$(\tilde{X}^T\tilde{X} + \lambda I)^{-1} = U^T \begin{bmatrix} \frac{1}{\lambda_1+\lambda} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2+\lambda} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_D+\lambda} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{D+1}+\lambda} \end{bmatrix} U$$

Fix the problem

The solution becomes

$$\tilde{w}^* = \left(\tilde{X}^T\tilde{X} + \lambda I\right)^{-1} \tilde{X}^T y$$

- not a minimizer of the original RSS

This in fact comes from minimizing **regularized** RSS (covered in next lecture)!

$$\min_{\tilde{w}} \|\tilde{X}\tilde{w} - y\|_2^2 + \lambda\|\tilde{w}\|_2^2$$

λ is a *hyper-parameter*, can be tuned by cross-validation.

Comparison to NNC

Parametric versus non-parametric

- **Parametric methods**: the size of the model does *not grow* with the size of the training set N.
  - e.g. linear regression, $D + 1$ parameters, independent of N.
- **Non-parametric methods**: the size of the model *grows* with the size of the training set.
  - e.g. NNC, the training set itself needs to be kept in order to predict. Thus, the size of the model is the size of the training set.