COMP 543: Tools & Models for Data Science

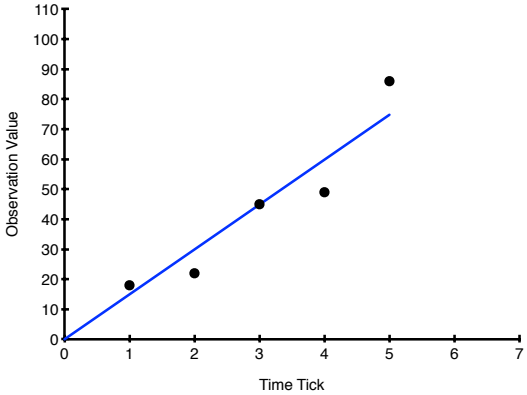
Linear Regression

Chris Jermaine & Risa Myers

Rice University

Linear Regression

- Most common model in data science! (Logistic Regression is very common as well)
  - Have a set of training data
  - Bunch of (x_i, y_i) pairs
  - x_i is a vector of real-valued "regressors"
  - y_i is a real-valued "response"
  - Want to learn a model that, given a new x, can predict y



Linear Regression

- Model is exceedingly simple
  - Predict y as $x \cdot r$
  - r is a vector of "regression coefficients"
  - $x \cdot r$ is the dot product op: $x \cdot r = \sum_j x_j \times r_j$

Regression Coefficient Example

- Specify the weight/importance and direction of each feature
- Weight is indicated with magnitude
- Direction is indicated by the sign

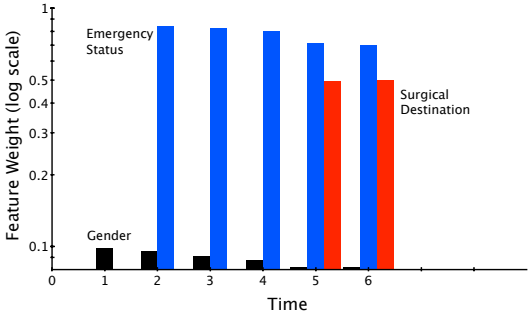
Predicting 30-day Mortality

Feature	**Regression Coefficient value**
MaleGender	-0.0662
Race	0.042
Age	-0.005
Charlson Comorbidity Index	-0.0075
ASA score	-0.323
Presurgical LOS	-0.0127
Start Location	0.2058
Enterectomy[a]	0.1233
Intercept	3.4723

[a] procedure to remove part of the intestine

Intuition behind Regression Coefficients

- Specify the weight/importance and direction of each feature
- Weight is indicated with magnitude
- Direction is indicated by the sign



How to Learn?

- Turns out there is a closed-form
  - Let the matrix **X** store the training data
  - ith row in **X** is ith training point, x_i
  - **y** is column vector storing responses
  - Then closed form least-squares estimate for r is:

$$\hat{r} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

  - This minimizes loss:

$$\sum_i (y_i - \sum_j x_{i,j} \times r_j)^2$$

- But this can be problematic for “Big Data”... why?

Problematic for “Big Data”

- Matrix may be too big to fit in memory
- e.g. 1 Billion point data set

More Reasonable Big Data Formulation

- Compute $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ as:

$$\left(\sum_i x_i^{\mathrm{T}} x_i\right)^{-1}$$

  - Note: assumes x_i is a row vector
  - $x_i^{\mathrm{T}} x_i$ is outer product of x_i with itself
  - Recall from lab that $x_i^{\mathrm{T}} x_i$ is the sum of the outer products of the matrix rows

? What's great about this formulation?

More Reasonable Big Data Formulation

- Compute $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ as:

$$\left(\sum_i x_i^{\mathrm{T}} x_i\right)^{-1}$$

  - Note: assumes x_i is a row vector
  - $x_i^{\mathrm{T}} x_i$ is outer product of x_i with itself
  - Recall from lab that $x_i^{\mathrm{T}} x_i$ is the sum of the outer products of the matrix rows
- What's great about this formulation?
- It can be parallelized!
- Distribute blocks (say 100) of rows at a time
- Compute the products
- Collect, reassemble, sum, then invert

More Reasonable Big Data Formulation

- Goal: Compute

$$\hat{r} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

1. Compute $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ as:

$$\left(\sum_i x_i^{\mathrm{T}} x_i\right)^{-1}$$

2. Compute $\mathbf{X}^{\mathrm{T}}\mathbf{y}$ as:

$$\left(\sum_i (x_i \times y_i)\right)^{\mathrm{T}}$$

- #2 can also be parallelized

Still, Bad for Very High-D Data

? Why?

Problematic for Very High-D Data

- Inverting **X** can be expensive, if the number of dimensions is high
  - $\approx$ 100K $\times$ 100K is an upper limit for a single machine
  - Easily get to this size with 4-grams in text

? What's the solution?

Still, Bad for Very High-D Data

- What's the solution?
- Gradient descent!
  - Very simple!
  - Take the partial derivative of the loss function wrt r_j is:

$$\frac{\partial}{\partial r_j} \sum_i (y_i - \sum_{j'} x_{i,j'} \times r_{j'})^2 = \sum_i -2(y_i - \hat{y}_i) x_{i,j}$$

  - Where $\hat{y}_i$ is prediction for y_i given current model
  - Again, the last term can be parallelized

Gradient Descent Algorithm

$r^0 \leftarrow$ non-stupid guess for r;
$iter \leftarrow 1$;
`repeat {`
  `for each` j
    $\Delta_j \leftarrow \sum_i -2(y_i - \hat{y}_i)x_{i,j}$
  $r^{iter+1} \leftarrow r^{iter} - \lambda\Delta$;
  $iter \leftarrow iter + 1$;
`} while (change` **`in`** `loss` $> \varepsilon$`)`

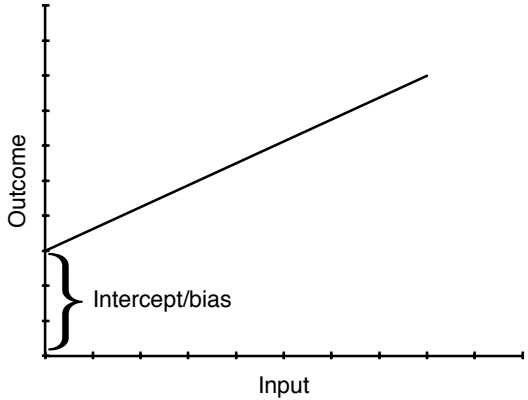
- Intuitively satisfying
  - Say our estimate ($\hat{y}_i$) for point i is too large
  - That is, $y_i - \hat{y}_i$ is negative (example, -0.05)
  - If $x_{i,j}$ is positive (ex: 2.0), point i will try to pull Δ_j so it is positive: contribution to Δ_j is $-2(-0.05)2.0 = 0.2$
  - Since $r^{iter+1} \leftarrow r^{iter} - \lambda\Delta$, point i will try to decrease r_j: will contribute a decrease of $\lambda 0.2$
  - So point i will want the regression coefficient to decrease, which makes sense

Why Is This a Good Big Data Algorithm?

- It's linear in number of data points
- Also linear in number of regressors (features)
- In particular, nice for sparse data (common in really high dimensions)
  - If $x_{i,j}$ is zero, no contribution to Δ_j

How To Add an Intercept?

- Add an extra column to each data point
- Always has a “1” value
- ? Why will this work?



How To Add an Intercept?

- Add an extra column to each data point
- Always has a “1” value
- Why will this work?
  - The model can learn a regression coefficient for that dimension
  - This is the intercept

How To Handle Categorical Data?

- Easiest: during training, treat “yes” as $+1$, “no” as -1
  - When applying model: > 0 becomes “yes”
  - When applying model: < 0 becomes “no”
- But generally understood to leave accuracy on the table. Why?
  - Every “yes”/“no” treated same way
  - Tries to map all “yes” to $+1$
  - Tries to map all “no” to -1
  - Even though not all “yes” (and all “no”) cases are the same

Why is this a Problem?

- Example: given (parents' years in school, salary) is someone a college graduate?
  - One person's parents have a PhD, make \$500K, point is $(21, 500)$
  - Another person's parents have some college, make \$60K, point is $(14, 50)$
  - LR for categorical data tries map both to $+1$
    - Rather than letting first map to arbitrarily large value (like $+10$), a really solid "yes"
    - And letting the second map to a smaller value (like $+0.5$) since a less solid "yes"
- Answer: logistic regression... will consider next time
  - Under topic of "generalized linear models"
  - Are a general class of probabilistic models based on LR
  - Logistic regression will allow more obvious "yes" cases to fall far above decision boundary
  - While obvious "no" cases fall far below

Questions?