Tools & Models for Data Science

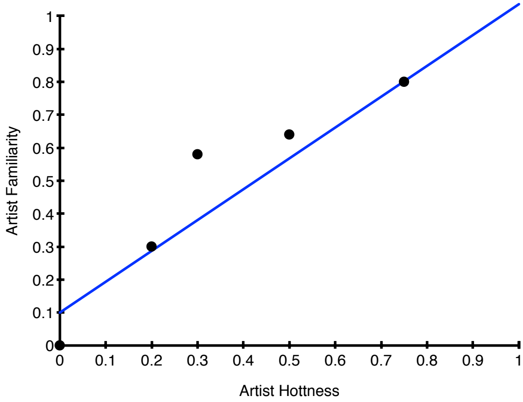
Linear Regression

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" / features / dimension
  - 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 $\hat{y}$ as $x \cdot r$
  - r is a vector of "regression coefficients"
  - $x \cdot r$ is the dot product of: $x \cdot r = \sum_j x_j \times r_j = \hat{y}_i$
  - Can be used with loss functions:
    - Least Squares (L2 norm)

$$Loss = \|y - f(x)\|_2^2$$

    - Mean Squared Error (MSE), where n is the number of training points

$$Loss = \frac{\|y - f(x)\|_2^2}{n}$$

    - Others...

Regression Coefficient Example

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

Predicting Song Tempo

Feature	**Regression Coefficient value**
Duration	-0.0061
Latitude	-0.1197
Loudness	1.1527
Year	0.0013
Intercept	139.72

Our Data

- Let the matrix **X** store the training data
- ith row in **X** is ith training point, x_i
- **y** is a column vector storing responses

$$\mathbf{X} = \begin{bmatrix} \text{———} & x_1 & \text{———} \\ \text{———} & x_2 & \text{———} \\ & \vdots & \\ \text{———} & x_i & \text{———} \\ & \vdots & \\ \text{———} & x_n & \text{———} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix}$$

How to Learn?

- Turns out there is a closed-form solution to this minimization problem we are solving for regression
  - Then closed form least-squares estimate for r is (you can look this up):

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

  - This minimizes loss:

$$\sum_i (y_i - x_i \times r)^2$$

Problematic for “Big Data”

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

- 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 dataset of 1 Billion observations / data points

- So, how can we perform linear regression on big data?

More Reasonable Big Data Formulation

- Recall the closed form least squares estimator for $\hat{r}$:

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

- We can 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 the outer product of x_i with itself, resulting in an $n \times n$ matrix
  - 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 (continued)

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

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

- What's great about this formulation?
  - It can be parallelized!
  - Distribute blocks of rows (say 100) at a time
  - Compute the products
  - Collect, reassemble, sum, then invert

More Reasonable Big Data Formulation (continued)

- Goal: Compute the closed form least squares estimate for $\hat{r}$

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

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

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

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
- since it is a sum of products

Still, Bad for Very High-D Data

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

? 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
  - Laptop maxes out at 4-8K $\times$ 4-8K
  - The matrix doesn't fit in memory!

? What's the solution?

Problematic for Very High-D Data

- Closed form LR takes too much memory for High-D data
- So, don't use it!
- Instead, use Gradient Descent on the Mean Squared Error Loss function:

$$\frac{\sum_i (y_i - x_i \times r)^2}{n}$$

- where r is the vector of regression coefficients
- and n is the number of data points

Gradient Descent on MSE

- The partial derivative of the loss function wrt r_j is:

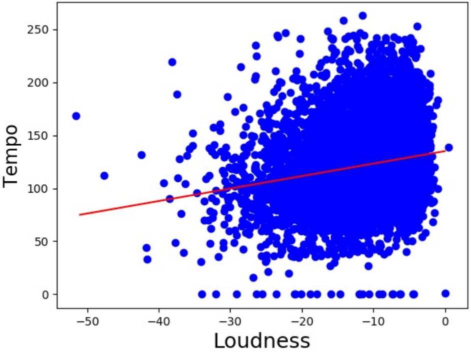
$$\frac{\partial}{\partial r_j} \frac{\sum_i (y_i - \sum_{j'} x_{i,j'} \times r_{j'})^2}{n} = \frac{\sum_i -2(y_i - \hat{y}_i)x_{i,j}}{n}$$

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

Gradient Descent Algorithm

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

- Where r is the vector of regression coefficients
- and n is the number of training data points
- 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 $-\frac{2}{n}(-0.05)2.0 = 0.2$
- Since $r^{iter+1} \leftarrow r^{iter} - \lambda\Delta$, point i will try to decrease r_j: will contribute a decrease of $\lambda 0.2$



Tempo
Loudness
250
200
150
100
50
0
−50
−40
−30
−20
−10
0

Why is this a Good Big Data Algorithm?

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

- It's linear in number of data points
- Also linear in number of regressors (features / dimensions)

Why is this a Good Big Data Algorithm?

- In particular, nice for sparse data (common in really high dimensions)
  - If $x_{i,j}$ is zero, no contribution to Δ_j
  - Note: You must use a sparse matrix representation to benefit

$$\text{Dense: } \begin{bmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -5 & 3 \end{bmatrix} \text{ Sparse: } \begin{bmatrix} 0: & (1,2) \\ 1: & (2,7) \\ 2: & \\ 3: & (0,4) \\ 4: & \\ 5: & (3,1) \\ 6: & (2,-5),(3,3) \end{bmatrix}$$

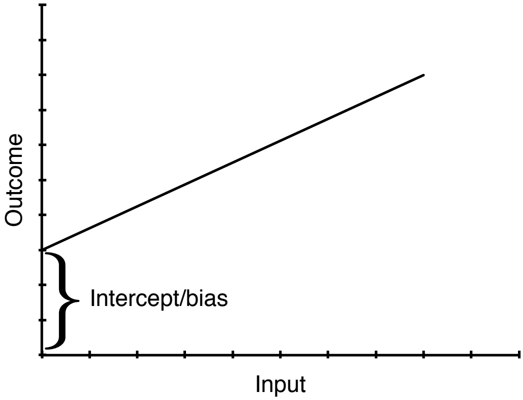
Why is this a Bad Big Data Algorithm?

- It's linear in number of data points
- Also linear in number of regressors (features / dimensions)
- Alternatives
  - Mini-batch Gradient Descent - use a small number of randomly sampled data points at each iteration
  - Stochastic Gradient Descent - use a single randomly sampled data point at each iteration

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?

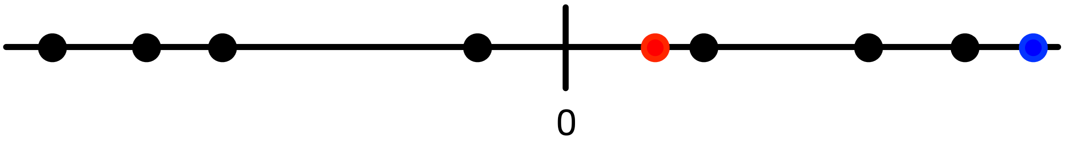
- Add an extra column to each data point
- Always has a “1” value
- Why will this work?
  - The model can learn a regression coefficient for that dimension
  - This is the intercept

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 this mapping is understood to leave accuracy on the table. Why?

How To Handle Categorical Data?

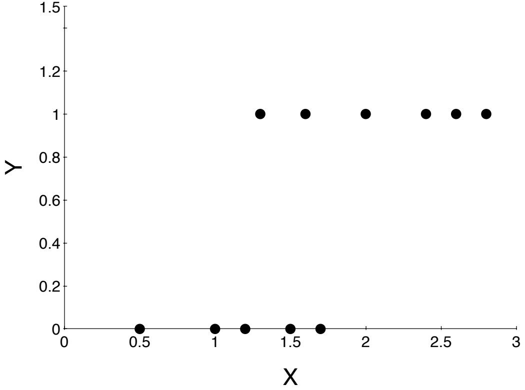
- Easiest: during training, treat “yes” as $+1$, “no” as -1
- But generally this mapping is understood to leave accuracy on the table. Because
  - Every “yes”/“no” treated same way
  - Tries to map all “yes” cases to $+1$
  - Tries to map all “no” cases to -1
  - Even though not all “yes” (and all “no”) cases are the same
  - The blue point is a strong “yes” than the red point

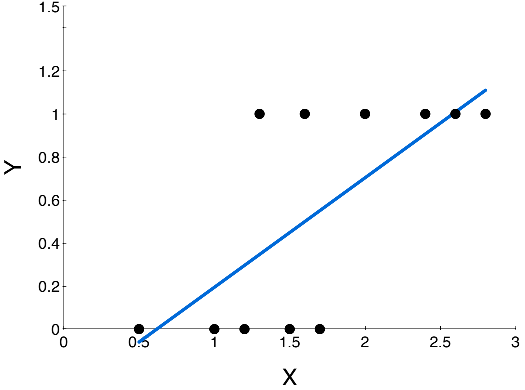


Why is this a Problem?

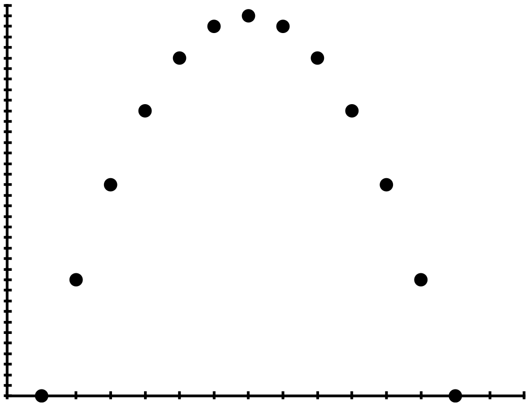
- Example: A song's duration and loudness, can we predict if the tempo will be ≥ 100?
  - One song might have a really short duration, be really quiet
  - Another song might be of average duration, and be a little quiet
  - Linear Regression for **categorical** data tries map both to -1
    - Rather 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

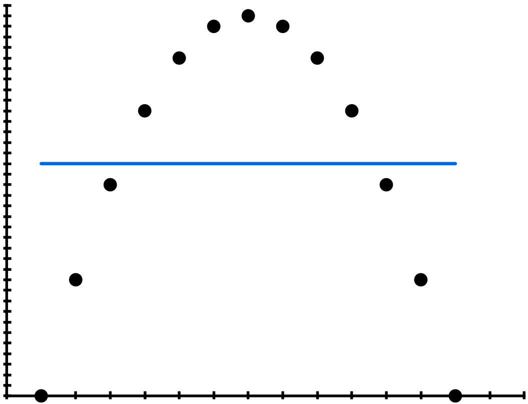
Data not Handled Well by Linear Regression: Categorical





Data not Handled Well by Linear Regression: Other Non-Linear





Questions?

- What do we know now that we didn't know before?
- How can we use what we learned today?