Linear Regression continued

- Example online
  - new dataset – predict_wage_regression.zip in Resources > Example Problems section
  - Exercise: change the example matlab code (grad_descent.m) to work with it

Cost function

- Find **w** that gives the lowest approximation error given the training data.
  - Minimize the **sum of square errors**:

$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2$$

Same **w** for all the training instances.

Iterative Method

- Start at some $\mathbf{w}_0$; take a step along **steepest slope**.
  - What's the steepest slope?
- Gradient of E:

$$\nabla_E(\mathbf{w}) = -\frac{1}{n}\sum_{k=1}^{n}\left(y^k - \mathbf{w}'\mathbf{x}^k\right)\mathbf{x}^k$$

Same form as before

Gradient Descent Algorithm

Initialize at some $\mathbf{w}_0$

For t=0,1,2,…do

Compute the gradient $\nabla_E(\mathbf{w}_t) = -\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$

Update the weights $\mathbf{w}_{t+1} = \mathbf{w}_t - \kappa\nabla_E(\mathbf{w}_t) = \mathbf{w}_t + \kappa\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$

Iterate with the next step until **w** doesn't change too much (or for a fixed number of iterations)

Return final **w**.

Attribute Scaling

- In order for Gradient Descent to converge quickly, scale attributes, e.g.

$$f_1' = \frac{f_1 - m_1}{\max_1 - \min_1} \quad \text{or} \quad f_1' = \frac{f_1 - m_1}{s_1}$$

where m_1 is the mean (average) of f_1, and $\max_1$, $\min_1$, s_1 are the max, min, and stdev of values for f_1.

Don't scale the all 1's attribute.

Learning Rate

- For small learning rate kappa, the value of E should decrease in each iteration.
  - If this doesn't happen, kappa is too big, so decrease kappa.
  - However, don't make kappa too small as GD will be slow to converge.

- Practical advise:
  - Start with kappa=1, and decrease it if too big.

THE MATRIX WAY: CANONICAL EQUATIONS

Matrix **X** and vector **y**

x_0	GPA	YearsOfExperience	Salary
1	90	1	50
1	80	3	60
1	90	2	55
1	70	8	70

$$\mathbf{X} = \begin{bmatrix} 1 & 90 & 1 \\ 1 & 80 & 3 \\ 1 & 90 & 2 \\ 1 & 70 & 8 \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} 50 \\ 60 \\ 55 \\ 70 \end{bmatrix}$$

The Matrix Way: Canonical Equations

- E will have the smallest value when the gradient is equal to zero.

$$\nabla_E(\mathbf{w}) = -\frac{1}{n}\sum_{k=1}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}'\mathbf{x}^k\right) = \mathbf{0}$$

$$\sum_{k=0}^{n}\mathbf{x}^k\left(y^k - \mathbf{w}'\mathbf{x}^k\right) = \mathbf{0}$$

$$\sum_{k=1}^{n}\left(\mathbf{x}^k y^k - \mathbf{x}^k\mathbf{w}'\mathbf{x}^k\right) = \mathbf{0}$$

$$\sum_{k=1}^{n}\mathbf{x}^k y^k = \sum_{k=1}^{n}\mathbf{x}^k\mathbf{w}'\mathbf{x}^k$$

$$\mathbf{X}'\mathbf{y} = (\mathbf{X}'\mathbf{X})\mathbf{w}$$

$$(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} = \mathbf{w}$$

- **X** is the $n \times (m+1)$ data matrix
  - one row of m+1 elements for each data instance
    - without the y attribute
- **y** is the n-vector of class values

- **X'X** is $(m+1) \times (m+1)$ matrix
  - Good if number m of attributes is not too big.
- **w** is m-vector, i.e. $(m+1) \times 1$

Matlab/Octave

```
X=[1 1;
   1 2];
y=[1.5; 2];

w=pinv(X'*X)*X'*y

Result (as expected)
w =
    1.0000
    0.5000
```


Matlab/Octave

```
X=[1 90 1;
   1 80 3;
   1 90 2;
   1 70 8]

y=[50;
   60;
   55;
   70]

w=pinv(X'*X)*X'*y

Result
w =
   86.50000
   -0.40000
    1.50000
```

n training tuples
m attributes

Discussion: GD vs. canonical equations

GD

Needs to iterate, sometimes a lot.
Need to play with kappa.

Method of choice when m is big (>10,000).

Canonical equations

No need to iterate, adjust kappa, or scale attributes.

However, the challenge is to compute:

$$(\mathbf{X}'\mathbf{X})^{-1}$$

X'X not a problem;
result is $(m+1) \times (m+1)$
Inverting takes ~$O(m^3)$ time.
Fine for m<10,000, difficult after that.

What if X^TX is non-invertible

- Causes:
  - Some columns are linearly dependent
    - E.g. salary is given in two columns; both in CAD and USD
  - Too many attributes, few tuples
- Solution:
  - Delete some attributes
  - Use regularization (developed later in the course)

Variants of Grad Descent

- Batch
  - Use the whole dataset to compute the gradient

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \kappa \nabla_E(\mathbf{w}_t) = \mathbf{w}_t + \kappa \frac{1}{n}\sum_{k=1}^{N} \mathbf{x}^k \left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$$

- Mini-Batch
  - Use subsets of the data (e.g. 1…b, b+1...2*b, etc) to compute gradient

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \kappa \nabla_E(\mathbf{w}_t) = \mathbf{w}_t + \kappa \frac{1}{n}\sum_{k=1}^{b} \mathbf{x}^k \left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)$$

- Stochastic Gradient Descent *** Will be on AS 3
  - Calculate the gradient using one training example at a time

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \kappa \nabla_E(\mathbf{w}_t) = \mathbf{w}_t + \kappa \left(\mathbf{x}^k \left(y^k - \mathbf{w}_t'\mathbf{x}^k\right)\right)$$

Regularization

- Let's say the input data X has dimension n x p
- If there are many parameters to learn, this can lead to overfitting
  - E.g. when n < p
- We need a simpler model
  - Recall Naïve Bayes params vs the full joint probability.
    - Naïve Bayes has O(?) parameters
    - The joint probability has O(?) parameters
- How can we make our regression model simpler?
  - Is a model with q << p parameters simpler?
  - (yes)
- How can we learn a regression model with fewer parameters?

L2 Regularization

$$E(\mathbf{w}) = \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2 + \frac{\lambda}{2}\sum_{i=1}^{p}(\mathrm{w}_i)^2$$

$$= \frac{1}{2n}\sum_{k=1}^{n}(y^k - \mathbf{w}'\mathbf{x}^k)^2 + \frac{\lambda}{2}\|w\|_2^2$$

- We get to take the derivative again!!!!

Derivative of the Regularizer

$$\frac{\partial}{\partial \mathrm{w}_j}\left(\frac{\lambda}{2}\sum_{i=1}^{p}(\mathrm{w}_i)^2\right) = ?$$

Exercises:

1. What's the update for L2 regression using Gradient Descent?
2. Can you set the derivative to zero and solve for w using this version of the error function?