Generalized linear regression

We use column vectors for now.
We extend each input with attribut with value 1 (to simplify a lot of things).
Our model is:

$$y = \vec{x}^T \cdot \vec{\theta}$$

Generalized linear regression

We use column vectors for now.
We extend each input with attribut with value 1 (to simplify a lot of things).
Our model is:

$$y = \vec{x}^T \cdot \vec{\theta}$$

Each input will make one row in matrix and expected outputs will be a column vector:

$$X = \begin{pmatrix} (\vec{x}^{(1)})^T \\ (\vec{x}^{(2)})^T \\ \dots \\ (\vec{x}^{(n)})^T \end{pmatrix} \qquad \vec{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(n)} \end{pmatrix}$$

Matrix magic

$$X\vec{\theta} - \vec{y} = \begin{pmatrix} (\vec{x}^{(1)})^T\vec{\theta} - y^{(1)} \\ (\vec{x}^{(2)})^T\vec{\theta} - y^{(2)} \\ \cdots \\ (\vec{x}^{(n)})^T\vec{\theta} - y^{(n)} \end{pmatrix}$$

$$(X\vec{\theta} - \vec{y})^T(X\vec{\theta} - \vec{y}) = \sum_{i=1}^{n}((\vec{x}^{(i)})^T\vec{\theta} - y^{(i)})^2 = E(\vec{\theta})$$

Gradient

Gradient definition:

$$\nabla_{\vec{\theta}} E = \left(\frac{\partial E}{\partial \theta_1}, \frac{\partial E}{\partial \theta_2}, \ldots, \frac{\partial E}{\partial \theta_n} \right)$$

Show direction up.

Gradient of error

$$E(\vec{\theta}) = \sum_{i=1}^{n}((\vec{x}^{(i)})^T\vec{\theta} - y^{(i)})^2$$

One part of the gradient:

$$\frac{\partial E}{\partial \theta_j} = \sum_{i=0}^{n} 2((\vec{x}^{(i)})^T\vec{\theta} - y^{(i)})x_j{}^{(i)}$$

Using matrices:

$$\nabla_{\vec{\theta}} E = 2X^T(X\vec{\theta} - \vec{y})$$

Matrix magic - conclusion

We want to have:

$$\nabla_{\vec{\theta}} E = 2X^T(X\vec{\theta} - \vec{y}) = \vec{O}$$

$$X^T X\vec{\theta} = X^T\vec{y}$$

$$\vec{\theta} = (X^T X)^{-1}X^T\vec{y}$$

These are called normal equations for linear regression.

Source code

```
import numpy as np

X = [[122, 3], [39, 1], [67, 3]]
y = [400000, 76000, 175000]

X = np.hstack([np.array(X, float),
               np.ones(shape=(len(y),1))])
y = np.array(y, float)

XXi = np.linalg.inv(X.T.dot(X))
theta = XXi.dot(X.T).dot(y)
print theta

print np.linalg.solve(X.T.dot(X), X.T.dot(y))
```

Time complexity

- $X^T X$ – $O(m^2 n)$
- Inversion of matrix / solving system of linear equations – $O(m^3)$.

Numerical methods - gradient descent

We iterate:

$$\vec{\theta} = \vec{\theta} - \alpha \nabla_{\vec{\theta}} E$$

After substiting for our gradient (factor 2 is hidden in α):

$$\vec{\theta} = \vec{\theta} - \alpha X^T (X\vec{\theta} - \vec{y})$$

Stochastic gradient descent

Instead of calculation error and gradient from all training examples, we do update after each example (we calcuate gradient from one example):

- `while (mení sa chyba):`
  - `for i in range(n):`
    - $\theta = \theta - \alpha \vec{x}^{(i)}((\vec{x}^{(i)})^T\theta - y^{(i)})$

It usually converges faster than vanilla gradient descent.