Supervised learning

Vladimír Boža
boza@fmph.uniba.sk
M-25 (ask by email about office hours)

Aplications of ML

Supervised learning

Speech recognition, image recongition
Machine translation, text generation
Recommendations of movies, books, ...
House price prediction
Marketing predictions (conversion rates, ...)

Unsupervised learing

Signal decomposition
Clustering
Visualization of data
Learning embeddings

Reinforcement learning

Games (go, chess, ...)
Robotics

Supervised learning

Data

Set of n pairs x - input, y - expected output. This is called training set.

Goal

Predict output for new x.

Note

In most cases, the $\vec{x}$ is a vector with m values (**attributes**) and y is scalar value.

Example: house prices

$\vec{x}$			y
Size	# of rooms	Distance from city centre	Price
122	3	0.5	400000
39	1	6	76000
67	3	2	175000
88	2	4	???

Nearest neighbour

- Got a new input $\vec{x}_t$.
- From training examples, pick one $(\vec{x}, y)$ where $\vec{x}$ is the most similar to $\vec{x}_t$. Predict y.
- (Modification: pick k most similar, predict average.)

Good

Good accuracy, when we have a lots of data.

Bad

Slow, bulky (we need to store whole training set in fast memory). Need to define similarity. Sensitive to scaling and irelevant attributes.

Picking from set of hyphothesis

Input

Set of examples $(\vec{x_1}, y_1), \ldots, (\vec{x_n}, y_n)$.

Set of hyphoteses

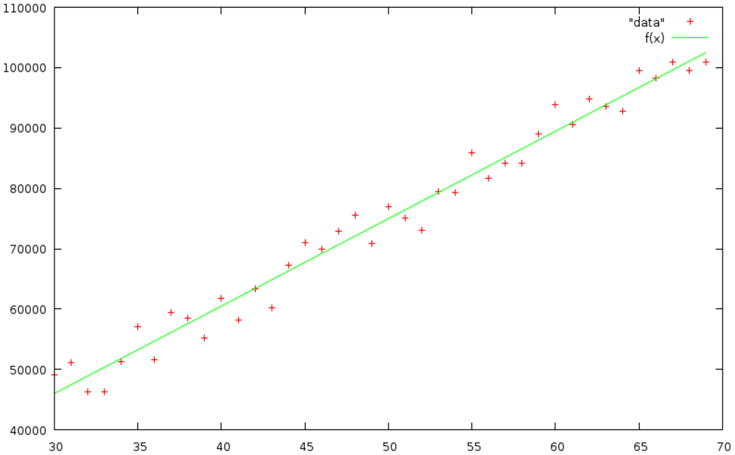
$H \subset \mathbb{R}^m \to \mathbb{R}$

Error function

Pick hyphothesis $h \in H$, which gives the lowest error.: $\sum_{i=1}^{t} \mathrm{err}(h(\vec{x_i}), y_i)$, where err is an **error function**.

Simple linear regression

One attribute (flat size).
Hyphotesis set: $H = \{h_\Theta(x) = \Theta_0 + \Theta_1 x\}$
Error function: $\mathrm{err}(y_p, y) = (y_p - y)^2$



Simple linear regression cont.

Looking for θ_0, θ_1, such that error is smallest as possible:

$$J(\theta_0, \theta_1) = \sum_{i=1}^{n} (\theta_0 + \theta_1 x_i - y_i)^2$$

Derivatives should be zero:

$$\frac{\partial J}{\partial \theta_0} = 0$$

$$\frac{\partial J}{\partial \theta_1} = 0$$

Example

Given training data:

x	y
3	6.5
4	7.9
5	9.9

Error would be:

$$J(\theta_0, \theta_1) = (\theta_0 + 3\theta_1 - 6.5)^2 + (\theta_0 + 4\theta_1 - 7.9)^2 + (\theta_0 + 5\theta_1 - 9.9)^2$$

Derivatives:

$$0 = \frac{\partial E}{\partial \theta_0} = 2(\theta_0 + 3\theta_1 - 6.5) + 2(\theta_0 + 4\theta_1 - 7.9) + 2(\theta_0 + 5\theta_1 - 9.9)$$

$$0 = \frac{\partial E}{\partial \theta_1} = 2(\theta_0 + 3\theta_1 - 6.5) \cdot 3 + 2(\theta_0 + 4\theta_1 - 7.9) \cdot 4 + 2(\theta_0 + 5\theta_1 - 9.9) \cdot 5$$

Example cont.

$$0 = \frac{\partial J}{\partial \theta_0} = 6\theta_0 + 24\theta_1 - 48.6$$

$$0 = \frac{\partial J}{\partial \theta_1} = 24\theta_0 + 100\theta_1 - 201.2$$

2 linear equations with 2 unknowns – boring and easy.

In general

$$J(\theta_0, \theta_1) = \sum_{i=1}^{n} (\theta_0 + \theta_1 x_i - y_i)^2$$

Derivatives:

$$0 = \frac{\partial J}{\partial \theta_0} = \sum_{i=1}^{n} 2(\theta_0 + \theta_1 x_i - y_i)$$

$$0 = \frac{\partial J}{\partial \theta_1} = \sum_{i=1}^{n} 2x_i(\theta_0 + \theta_1 x_i - y_i)$$

Generalization cont.

$$0 = \theta_0 n + \theta_1 \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} y_i$$

$$0 = \theta_0 \sum_{i=1}^{n} x_i + \theta_1 \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i y_i$$

$$0 = \theta_0 n \sum_{i=1}^{n} x_i + \theta_1 \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i$$

$$0 = \theta_0 n \sum_{i=1}^{n} x_i + \theta_1 n \sum_{i=1}^{n} x_i^2 - n \sum_{i=1}^{n} x_i y_i$$

...cont.

$$0 = \theta_1 \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i - \theta_1 n \sum_{i=1}^{n} x_i^2 + n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i$$

$$\theta_1 = \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i - n \sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i - n \sum_{i=1}^{n} x_i^2}$$

From first equation:

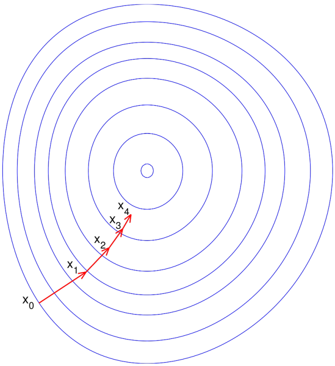
$$\theta_0 = \frac{1}{n} \left(\sum_{i=1}^{n} y_i - \theta_1 \sum_{i=1}^{n} x_i \right)$$

Other ways of minimalization

- Grid search
  - Try several grid spaced values. Zoom in.
  - Only for few variables.
- Numerical methods.

Numerical minimalization

Vector $\left(\frac{\partial J}{\partial \theta_0}, \frac{\partial J}{\partial \theta_1}\right)$ gives direction upwards (gradient).
Idea: Use gradient to move down.



Gradient descent

- $(\theta_0, \theta_1) = \texttt{Good initialization}$
- `while (error changes):`
  - $\theta_0 = \theta_0 - \alpha \frac{\partial J}{\partial \theta_0}$
  - $\theta_1 = \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}$

We need to pick α. Trial and error works well. Usual values $1, 0.1, 0.01, \ldots$. There are better ways.

Derivatives

Options:

- Manually
- Wolfram alpha
- Libraries, which do it for you (pytorch, autograd). Keyword here is autograd.
- Numerical derivative
  - `scipy.optimize.approx_fprime`
  - $\frac{\partial f}{\partial x} \approx \frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x}$

Generalized linear regression

We use column vectors for now.
We extend each input with attribute 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 attribute 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

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

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

Gradient

Gradient definition:

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

Shows direction up (i.e. if you move parameters this way, loss will increase).

Gradient of error

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

One part of the gradient:

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

Using matrices:

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

Matrix magic - conclusion

We want to have:

$$\nabla_{\vec{\theta}} J = 2X^T(X\vec{\theta} - \vec{y}) = \vec{0}$$

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

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

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:

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

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 (not converged):`
  - `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. But, you need to decrease alpha over time (this is not needed for vanilla gradient descent).

Summary

Linear regression

Inputs: rows in matrix X.
Expected outputs: vector $\vec{y}$.
We are looking for parameters $\vec{\theta}$, such that $E = (X\vec{\theta} - \vec{y})^T(X\vec{\theta} - \vec{y})$ was smallest as possible.

Training

Option 1: solve system of equations $X^T X\vec{\theta} = X^T\vec{y}$
Option 2: (stochastic) gradient descent: $\vec{\theta} = \vec{\theta} - \alpha X^T(X\vec{\theta} - \vec{y})$

E is convex function, it has at most one local minimum, which is also global and both methods will find same solution (apart from numerical errors).

Prediction from new input

$y_{new} = \vec{x}_{new}^T \cdot \vec{\theta}$

Other models

Still regression (one real number as an output).
Sometimes data are nonlinear.

- Locally weighted linear regression (in Machine learing course)
- Polynomial regression and its reduction on linear
- Neural nets (not today)

Polynomial regression

One input x, model with degree 2:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$

Polynomial regression

One input x, model with degree 2:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$

Two inputs x_1, x_2, model (up to degree 2):

$$y = \theta_0 + \theta_{10} x_1 + \theta_{01} x_2 + \theta_{11} x_1 x_2 + \theta_{20} x_1^2 + \theta_{02} x_2^2$$

We can use same procedure as last time and find values of θ. Or reduce the problem to linear regression.

Reduction of polynomial regression

For two inputs

Input: $(1, x_1, x_2)$ we change into:
$(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$
And we can solve linear regression (we do not change outputs).

Reduction of polynomial regression

For two inputs

Input: $(1, x_1, x_2)$ we change into:
$(1, x_1, x_2, x_1 x_2, x_1^2, x_2^2)$
And we can solve linear regression (we do not change outputs).

In general

We have p basis functions: $\phi_1(\vec{x}), \phi_2(\vec{x}), \ldots, \phi_p(\vec{x})$, kde $\phi_i \in \mathbb{R}^m \to \mathbb{R}$.
We preprocess input matrix X into matrix Φ:

$$\begin{pmatrix} \phi_1(\vec{x}^{(1)}) & \phi_2(\vec{x}^{(1)}) & \ldots & \phi_p(\vec{x}^{(1)}) \\ \phi_1(\vec{x}^{(2)}) & \phi_2(\vec{x}^{(2)}) & \ldots & \phi_p(\vec{x}^{(2)}) \\ & & \vdots & \\ \phi_1(\vec{x}^{(n)}) & \phi_2(\vec{x}^{(n)}) & \ldots & \phi_p(\vec{x}^{(n)}) \end{pmatrix}$$

And we solve linear regression, for example the system: $\Phi^T \Phi \vec{\theta} = \Phi^T \vec{y}$

Basis fuctions - examples

Not only polynomials.

- $\phi(\vec{x}) = x_4 x_7$, $\phi(\vec{x}) = x_2$
- 0-1 functions: $\phi(\vec{x}) = x_6 > 0$
- Some preprocessings: $\phi(\vec{x}) = \log(x_5 + 1)$
- Kernel fuctions: $\phi(\vec{x}) = e^{\frac{-\|\vec{z}-\vec{x}\|^2}{\sigma^2}}$

Linear regression with preprocessing