Machine learning

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

compbio.fmph.uniba.sk/vyuka/ml
watch out for annoucements on Google Classroom (link on the website above)

Schedule

Wednesday: 13:10 C (occasional tutorials in H6)
Thursday: 9:50 C

Grading

Teoretical homework (2)	14%
Practical homework (8, points from best 4)	16%
Project (3 days before exam, latest 31.1.2021)	30%
Tutorials: Each one for up to 5 points	
Exam	40%

Need 40% from exam to pass.

Projekt

Pick some application of ML (web page will have some ideas later).

Implement, evaluate

Several options for your output:

1. Summary in form of conference paper (5 - 15 pages).
2. Small web application demonstrating your project, which includes short summary (methods, results)
3. Get at least 10th place in serious (money involved) Kaggle competion + 1 page summary of your methods.

In all cases, you should publish your source codes (Github prefered).

Part of exam would be about methods in your project and will ask you to do small update in the project.

Prereqs

Linear algebra (basics).
Derivatives
Programming (Python)
Being a honeybadger

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

Python

```
def swap(a, b):
  return b, a

c, d = swap(a, b)

for i in range(0, 10):
  if i % 2 == 0:
    print "parne"
```

Numpy

```
>>> import numpy as np
>>> x = np.array([[1,2],[3,4]], float)
>>> y = np.array([[0,1],[1,0]], float)
>>> x
array([[ 1.,  2.],
       [ 3.,  4.]])
>>> x.shape
(2, 2)
>>> x + y
array([[ 1.,  3.],
       [ 4.,  4.]])
>>> x * y
array([[ 0.,  2.],
       [ 3.,  0.]])
```

Numpy

```
>>> import numpy as np
>>> x = np.array([[1,2],[3,4]], float)
>>> y = np.array([[0,1],[1,0]], float)
>>> x.dot(y)
array([[ 2.,  1.],
       [ 4.,  3.]])
>>> np.exp(x)
array([[  2.71828183,   7.3890561 ],
       [ 20.08553692,  54.59815003]])
>>> x * 2
array([[ 2.,  4.],
       [ 6.,  8.]])
```

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

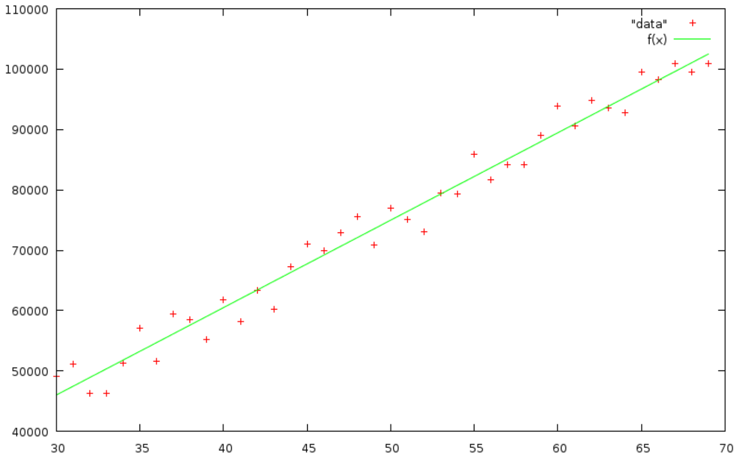
$H \subset \mathbb{R}^m \to \mathbb{R}$

Error function

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

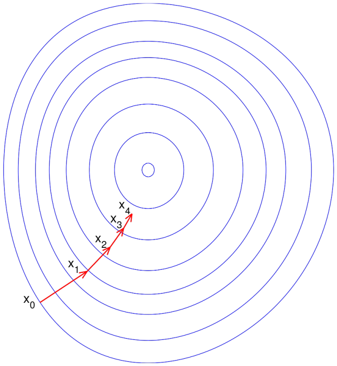
$$\theta_0 = \frac{1}{n} \left(\sum_{i=1}^{n} y_i - \theta_1 \sum_{i=1}^{n} x_i \right)$$

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

- $(\theta_0, \theta_1) =$ `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`
  - $\frac{\partial f}{\partial x} \approx \frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x}$