Tufts

Class #01: Types of Learning; Linear Methods

Machine Learning (COMP 135)

1

Defining a Learning Problem

Cover image: McGraw Hill, 1997 [link]

- Suppose we have three basic components:
  1. Set of tasks, *T*
  2. A performance measure, *P*
  3. Data describing some experience, *E*

> A computer program ***learns*** if its performance at tasks in *T*, as measured by *P*, improves based on *E*.
>
> From: Tom M. Mitchell, *Machine Learning* (1997)

2

An Example Problem

- Suppose we want to build a system, like Siri or Alexa, that responds to voice commands
- What are our components?
  1. Tasks, *T*
  2. Performance measure, *P*
  3. Experience, *E*

Task:

Take system actions, based upon speech

Performance:

How often correct action is taken during testing

Experience?

This is the tricky part!

For many domains, deriving the ***experience*** used by the system is the biggest real challenge:

- The ***evidence*** it uses.
- ***How*** it uses that evidence.

3

The Expert Systems Approach

- One (older) approach used ***expert-generated rules***:
  1. Find someone with advanced knowledge of linguistics
  2. Get them to devise the structural rules of language's grammar and semantics
  3. Encode those rules in program for parsing written language
  4. Build another program to translate speech into written language, and tie that to ***another*** program for taking actions based upon the parsing

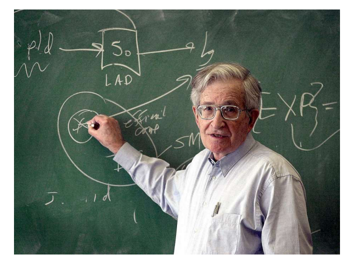


Photo of Noam Chomsky: Rich Beauchesne, AP (2001)

4

Another Approach: Supervised Learning

- In supervised learning, we:
1. Provide a set of ***correct answers*** to a problem
2. Use algorithms to find (mostly) correct answers to ***similar problems***
- We can still use experts, but their job is different:
  - ***Don't need*** to devise complex rules for understanding speech
  - ***Instead***, they just have to be able to tell what the ***correct results*** of understanding look like

Another Approach: Supervised Learning

- Collect a large set of sample things a set of test users say to our system
- For each, map it to a correct outcome action the system should take

"Call my wife"	`call(555-123-4567)`
"Set an alarm for 4:00 AM"	`alarm_set(04:00)`
"Play *Pod Save America*"	`podcast_play("Pod Save America")`
…	…

- A large set of such (*speech, action*) pairs can be created
- This can then form the experience, *E*, the system needs

Inductive Learning

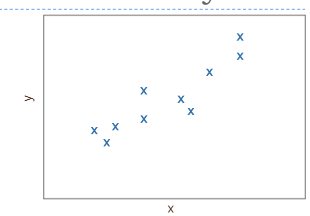
- In its simplest form, induction is the task of learning a ***function*** on some inputs from ***examples*** of its outputs
- For a function, f, that we want to learn, each of these training examples is a pair

$$(x, f(x))$$

  - We assume that we do not yet know the actual form of the function f (if we did, we don't need to learn)
- ***Learning problem***: find a hypothesis function, h, such that $h(x) = f(x)$ (at least ***most*** of the time), based on a training set of example input-output pairs

Example: 1-Dimensional Data Analysis

- What are our components?
1. Tasks, *T*
2. Performance measure, *P*
3. Experience, *E*



Task:

Predict the output, $f(x) = y$, for points we ***haven't seen yet***

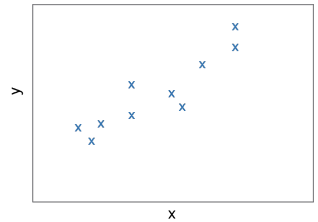
Performance:

Seek to reduce overall error of the predictions for known points

Experience?

Here, this is the ***easy*** part: we are provided with an existing data-set of (input, output) points (x, y) [i.e., $f(x) = y$]

Linear Regression



- In general, we want to learn a hypothesis function h that minimizes our error relative to the actual output function f
- Often, we will ***assume*** that this function h is linear, so the problem becomes finding a set of weights that ***minimize the error*** between f and our function:

$$h(x_1, x_2, \ldots, x_n) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

An Error Function: Least Squared Error

- For a chosen set of weights, $\mathbf{w}$, we can define an error function as the ***squared residual*** between what the hypothesis function predicts and the actual output, summed over all N test-cases:

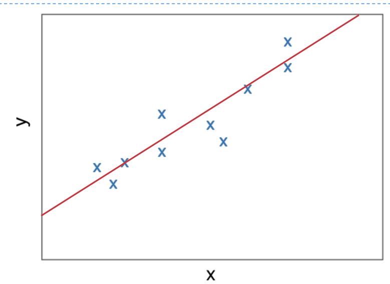
$$Loss(\mathbf{w}) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(\mathbf{x}_j))^2$$

- Learning is then the process of finding a weight-sequence that ***minimizes*** this loss:

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} Loss(\mathbf{w})$$

- ***Note***: Other loss-functions are commonly used (but the basic learning problem remains the same)

An Example



- For the data given, the best fit for a simple linear function of x is as follows:

$$h(x) \longleftarrow y = 1.05 + 1.60x$$

Finding Minimal-Error Weights (Analytically)

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} Loss(\mathbf{w})$$

- We can ***in principle*** solve for the weight with least error ***analytically***

1. Create data matrix with one training input example per row, one feature per column, and output vector of all training outputs

$$\mathbf{X} = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1n} \\ f_{21} & f_{22} & \cdots & f_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N1} & f_{N2} & \cdots & f_{Nn} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

2. ***Solve*** for the minimal weights using linear algebra (for large data, requires optimized routines for finding matrix inverses, doing multiplications, etc., as well as for certain matrix properties to hold, which are not universal):

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

Finding Minimal-Error Weights (Iteratively)

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} Loss(\mathbf{w})$$

- Weights that minimize error can instead be found (or at least approximated) using gradient descent:

1. ***Loop repeatedly*** over all weights w_i, updating them based on their "contribution" to the overall error:

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i}\,(y_j - h_{\mathbf{w}}(\mathbf{x}_j))$$

Learning rate: multiplying parameter for weight adjustments

Feature: normalized value of feature i of training input j

Overall Error: difference between current and correct outputs for case j

2. ***Stop on convergence***, when maximum update on any weight (Δ) drops below some threshold (Θ); alternatively, stop when change in error/loss grows small enough

Updating Weights

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i}\,(y_j - h_{\mathbf{w}}(\mathbf{x}_j))$$

- For each value i, the update equation takes into account:
  1. The ***current*** weight-value, w_i
  2. The ***difference*** (positive or negative) between the current hypothesis for input j and the known output: $(y_j - h_{\mathbf{w}}(\mathbf{x}_j))$
  3. The i-th feature of the data, $x_{j,i}$
- When doing this update, we must remember that for n data features, we have $(n + 1)$ weights, including the bias, w_0

$$h(x_1, x_2, \ldots, x_n) = \boxed{w_0} + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

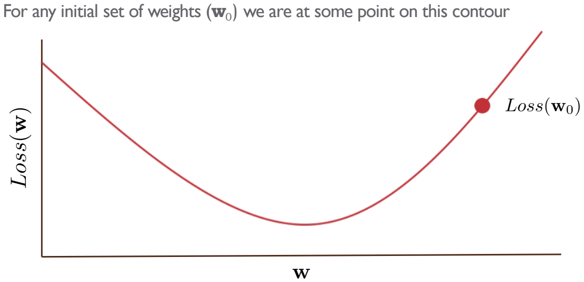
- It is presumed that the related "feature" $x_{j,0} = 1$ in every case, and so the update for the bias weight becomes:

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(\mathbf{x}_j))$$

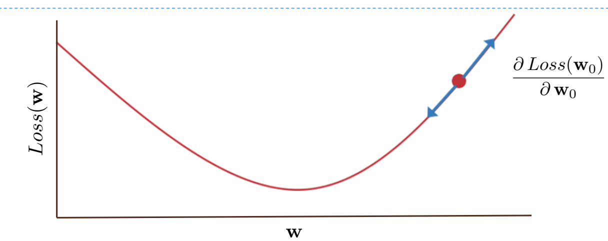
Gradient Descent

$$Loss(\mathbf{w}) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(\mathbf{x}_j))^2$$

- The loss function forms a ***contour*** (here shown for one-dimensional data)
  - For any initial set of weights ($\mathbf{w}_0$) we are at some point on this contour



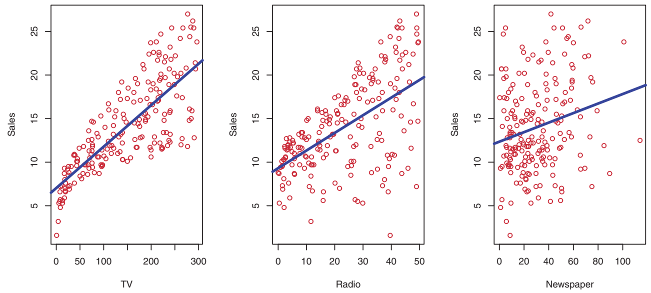
Gradient Descent



- The derivate of the loss function at the given weight settings "points uphill" along the slope of the function (note: this is true for ***this*** point, ***not every*** point)
- The gradient descent update moves along the function in the ***opposite*** direction toward the direction that ***decreases*** loss most significantly

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i}\,(y_j - h_{\mathbf{w}}(\mathbf{x}_j))$$

Practical Use of Linear Regression



Ad sales vs. media expenditure (1000's of units). From: James et al., *Intro. to Statistical Learning* (Springer, 2017)

- A linear model can often radically simplify a data-set, isolating a relatively straightforward relationship between data-features and outcomes