MITx 6.86x - Machine Learning with Python: from Linear Models to Deep Learning

https://www.edx.org/course/machine-learning-with-python-from-linear-models-to

Lecturers: Regina Barzilay, Tommi Jaakkola, Karene Chu

Notes assembled by: Antonello Lobianco

Student's notes (2020 run)

Disclaimer: The following notes are a mesh of my own notes, selected transcripts, some useful forum threads and various course material. I do not claim any authorship of these notes, but at the same time any error could well be arising from my own interpretation of the course material.

Contributions are really welcome. If you spot an error , want to specify something in a better way (English is not my primary language), add material or just have comments, you can clone, make your edits and make a pull request (preferred) or just open an issue.

(This PDF versions may be outdated ! Please refer to the GitHub repository source files for latest version.)

Table of contents

[MITx 6.86x Notes Index]

Unit 00 - Course Overview, Homework 0, Project 0

Recitation 1 - Brief Review of Vectors, Planes, and Optimization

Points and Vectors

Norm of a vector

Norm: Answer the question how big is a vector

- $\|X\|_l \equiv l - NORM := (\sum_{i=1}^{size(X)} x_i^l)^{(1/l)}$
- Julia: `norm(x)`
- NumPy: `numpy.linalg.norm(x)`

If l is not specified, it is assumed to be 2, i.e. the Euclidian distance. It is also known as “length”, “2-norm”, “l2 norm”, …

Dot product of vector

Aka “scalar product” or “inner product”.

It has a relationship on how vectors are arranged relative to each other

- Algebraic definition: $x \cdot y \equiv x'y := \sum_{i=1}^{n} 2x_i * y_i$
- Geometric definition: $x \cdot y := \|x\| * \|y\| * cos(\theta)$ (where θ is the angle between the two vectors and $\|x\|$ is the 2-norm)
- Julia: `dot(x,y)`
- Numpy: `np.dot(x,y)`

Note that using the two definitions and the `arccos`, the inverse function for the cosine, you can retrieve the angle between two functions as `angle_x_y = arccos(dot(x,y)/(norm(x)*norm(y)))`.

- Julia: `angle_x_y = acos(dot(x,y)/(norm(x)*norm(y)))`

Geometric interpretation of a vector

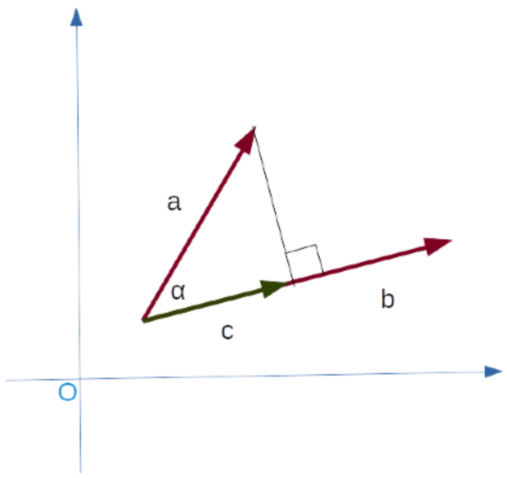
Geometrically, the elements of a vector can be seen as the coordinates of the position of arrival *compared to the position of departing*. They represent hence the *shift* from the departing point.

For example the vector [-2,0] could refer to the vector from the point (4,2) to the point (2,2) but could also represent the vector going from (6,4) to (4,4).

Vectors whose starting point is the origin are called *position vectors* and they define the coordinates in the n-space of the points where they arrive to.

Vector projections

Let’s be *a* and *b* two (not necessary unit) vectors. We want to compute the vector *c* being the projection of *a* on *b* and its l-2 norm (or length):



Let's start from the length. We know from a well-known trigonometric equation that

$\|c\| = \|a\| * cos(\alpha)$, where α is the angle between the two vectors *a* and *b*:

But we also know that the dot product $a \cdot b$ is equal to $\|a\| * \|b\| * cos(\alpha)$.

By substitution we find that $\|c\| = \frac{a \cdot b}{\|b\|}$. This quantity is also called the *component* of *a* in the direction of *b*.

To find the vector *c* we now simply multiply $\|c\|$ by the unit vector in the direction of *b*, $\frac{b}{\|b\|}$, obtaining $c = \frac{a \cdot b}{\|b\|^2} * b$.

If *b* is already a unit vector, the above equations reduce to:

$\|c\| = a \cdot b$ and $c = (a \cdot b) * b$

In Julia:

```
using LinearAlgebra
a = [4,1]
b = [2,3]
normC = dot(a,b)/norm(b)
c = (dot(a,b)/norm(b)^2) * b
```

In Python:

```
import numpy as np
a = np.array([4,1])
b = np.array([2,3])
normC = np.dot(a,b)/np.linalg.norm(b)
c = (np.dot(a,b)/np.linalg.norm(b)**2) * b
```

Planes

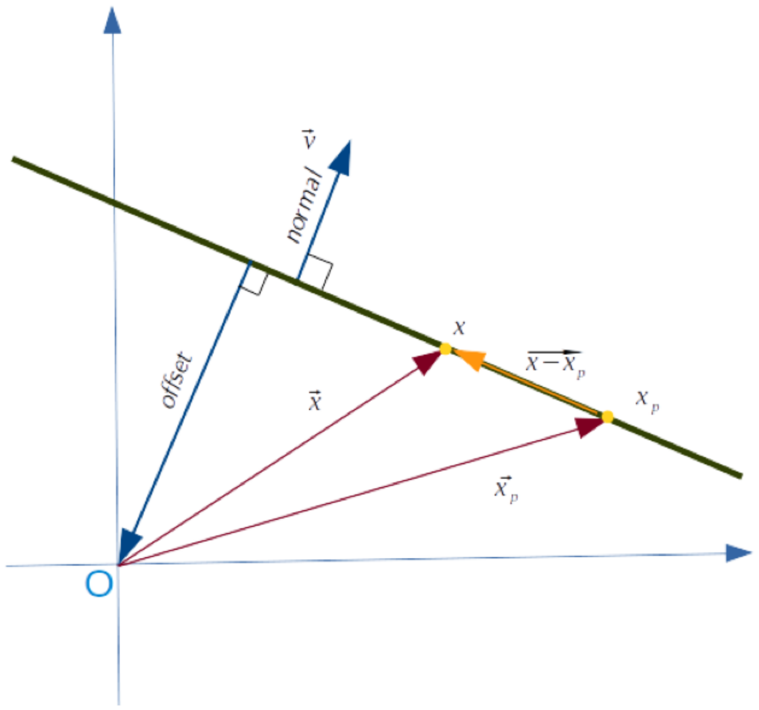
An (hyper)plane in n dimensions is any n−1 dimensional subspace defined by a linear relation. For example, in 3 dimensions, hyperplanes span 2 dimensions (and they are just called "planes") and can be defined by the vector formed by the coefficients {A,B,C,D} in the equation $Ax + By + Cz + D = 0$. Note that while the plane is unique, the vector defining it is not: in relation to the A-D coefficients, the equation is homogeneous, i.e. if we multiply all the A-D coefficients by the same number, the equation remains valid.

As hyperplanes separate the space into two sides, we can use (hyper)planes to set boundaries in classification problems, i.e. to discriminate all points on one side of the plane vs all the point on the other side.

Besides to this analytical definition, a plane can be uniquely identified also in a geometrical way starting from a point x_p on the plane and a vector $\vec{v}$ normal to the plane (not necessarily departing from the point or even from the plane):. Let's define:

- *Normal* of a plane: any n-dimensional vector perpendicular to the plane.

- *Offset of the plane with the origin*: the distance of the plan with the origin, that is the specific normal between the origin and the plane



Given a point x_p known to sit on the plane and $\overrightarrow{x_p}$ its positional vector, a generic point x and corresponding positional vector $\vec{x}$, the point x is part of the plane if and only if ("iff") the vector connecting the two points, that is $\overrightarrow{x - x_p}$, lies on the plane. In turn this is true iff such vector is orthogonal to the normal of the plane $\vec{v}$, that we can check using the dot product.

To sum up, we can define the plane as the set of all points x x such that $(\overrightarrow{x - x_p}) \cdot \vec{v} = 0$.

As from the coefficients A-D in the equation, while x_p and $\vec{v}$ unambiguously identify the plane, the converse is not true: any plane has indeed infinite points and normal vectors.

For example, let's define a plane in two dimensions passing by the point $x_p = (3, 1)$ and with norm $\vec{v} = (2, 2)$, and let's check if point $a = (1, 3)$ is on the plane. Using the above equation we find $(\vec{a} - \overrightarrow{x_p}) \cdot \vec{v} = \left(\begin{bmatrix} 1 \\ 3 \end{bmatrix} - \begin{bmatrix} 3 \\ 1 \end{bmatrix} \right) \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 0$, *a* is part of the plane.

Let's consider instead the point b = (1,4). As $(\vec{b} - \overrightarrow{x_p}) \cdot \vec{v} = \left(\begin{bmatrix} 1 \\ 4 \end{bmatrix} - \begin{bmatrix} 3 \\ 1 \end{bmatrix} \right) \cdot \begin{bmatrix} 2 \\ 2 \end{bmatrix} = 2$, *b* is not part of the plane.

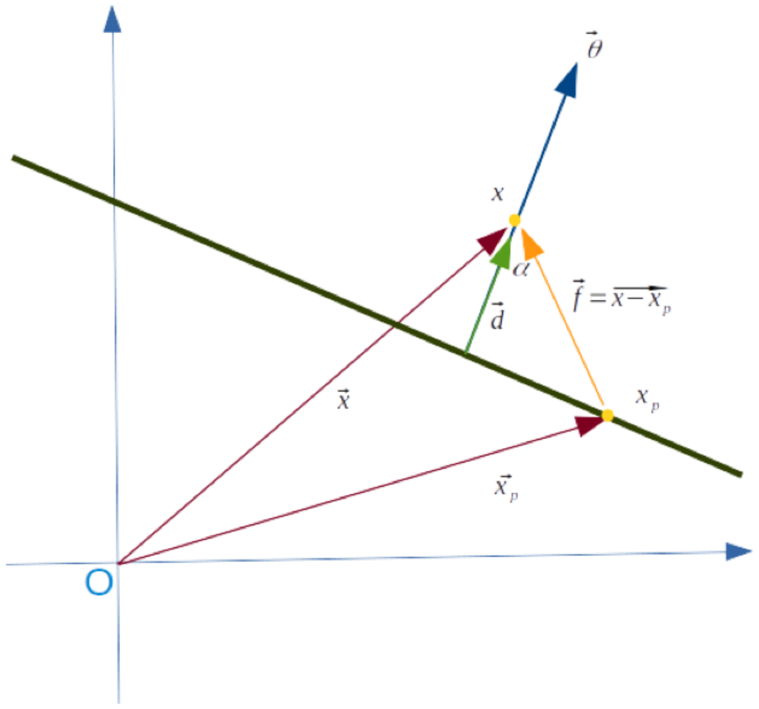
Starting from the information on the point and the normal we can retrieve the algebraic equation of the plane rewriting the equation $(\overrightarrow{x - x_p}) \cdot \vec{v} = 0$ as $\vec{x} \cdot \vec{v} - \overrightarrow{x_p} \cdot \vec{v} = 0$: the first dot product gives us the polynomial terms in the n-dimensions (often named θ), the last (negatively signed) term gives the associated offset (often named θ_0). Seen from the other side, this also means that the coefficients of the dimensions of the algebraic equation represent the normal of the plane. We can hence define our plane with just the normal and the corresponding offset (always a scalar).

Note that when θ is a unit vector (a vector whose 2-norm is equal to 1) the offset θ_0 is equal to the offset of the plane with the origin.

Using the above example ($x_p = [3, 1], \vec{v} = [2, 2]$), we find that the algebraic equation of the plane is $2x_1 + 2x_2 - (3 * 2 + 1 * 2) = 0$, or, equivalently, $\frac{x_1}{\sqrt{2}} + \frac{x_2}{\sqrt{2}} - \frac{4}{\sqrt{2}} = 0$.

Distance of a point to a plane

Given a plan defined by its norm θ and the relative offset θ_0, which is the distance of a generic point x from such plane ? Let's start by calling $\vec{f}$ the vector between any point on the plan x_p and the point x, that is $\vec{f} = \vec{x} - \overrightarrow{x_p}$. The distance $\|\vec{d}\|$ of the point with the plane is then $\|\vec{d}\| = \|\vec{f}\| * cos(\alpha)$, where α is the angle between the vector $\vec{f}$ and $\vec{d}$.



But we know also that $\vec{f} \cdot \vec{\theta} = \|\vec{f}\| * \|\vec{\theta}\| * cos(\alpha)$ from the definition of the dot product.

By substitution, we find that $\|\vec{d}\| = \|\vec{f}\| * \frac{\vec{f} \cdot \vec{\theta}}{\|\vec{f}\| * \|\vec{\theta}\|} = \frac{\vec{f} \cdot \vec{\theta}}{\|\vec{\theta}\|}$ (we could have arrived to the same conclusion by considering that $\|\vec{d}\|$ is the component of $\vec{f}$ in direction of $\vec{\theta}$).

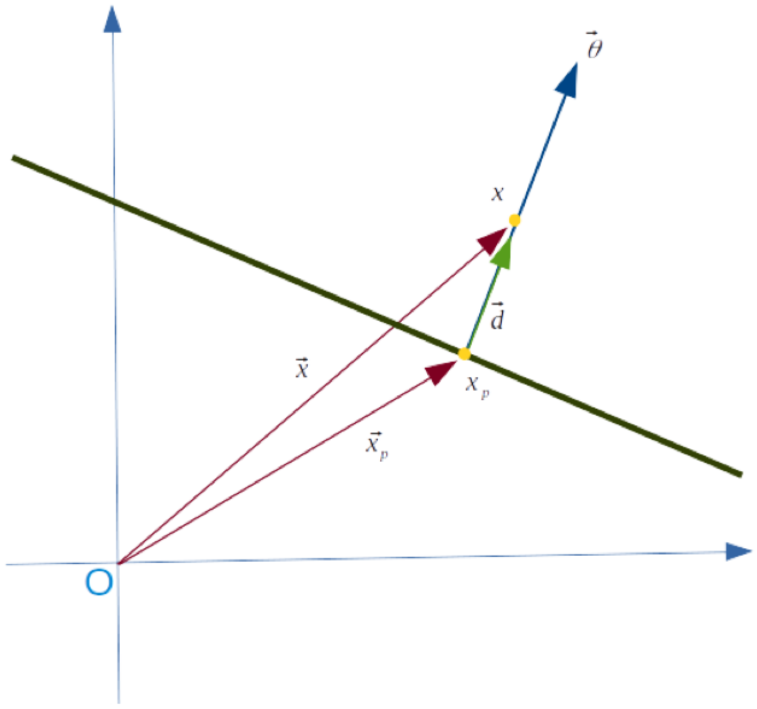
By expanding $\vec{f}$ we find that $\|\vec{d}\| = \frac{(\vec{x} - \overrightarrow{x_p}) \cdot \vec{\theta}}{\|\vec{\theta}\|} = \frac{\vec{x} \cdot \vec{\theta} + \theta_0}{\|\vec{\theta}\|}$

The distance is positive when x is on the same side of the plane as $\vec{\theta}$ points and negative when x is on the opposite side.

For example the distance between the point $x = (6, 2)$ and the plane as define earlier with $\theta = (1, 1)$ and $\theta_0 = -4$ is $\frac{\vec{x} \cdot \vec{\theta} + \theta_0}{\|\vec{\theta}\|} = \frac{\begin{bmatrix} 6 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 4}{\|\begin{bmatrix} 1 \\ 1 \end{bmatrix}\|} = \frac{8-4}{\sqrt{2}} = 2 * \sqrt{2}$

Projection of a point on a plane

We can easily find the projection of a point on a plane by summing to the positional vector of the point, the vector of the distance from the point to the plan, in turn obtained multiplying the distance (as found earlier) by the *negative* of the unit vector of the normal to the plane.



Algebraically: $\overrightarrow{x_p} = \vec{x} - \frac{\vec{x}\cdot\vec{\theta}+\theta_0}{\|\vec{\theta}\|} * \frac{\vec{\theta}}{\|\vec{\theta}\|} = \vec{x} - \vec{\theta} * \frac{\vec{x}\cdot\vec{\theta}+\theta_0}{\|\vec{\theta}\|^2}$

For the example before, the point x_p is given by $\begin{bmatrix} 6 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} * \frac{\begin{bmatrix} 6 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 4}{2} = \begin{bmatrix} 6 \\ 2 \end{bmatrix} - \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$

On this subject see also:

- https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/dot-cross-products/v/defining-a-plane-in-r3-with-a-point-and-normal-vector
- https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/dot-cross-products/v/point-distance-to-plane

Loss Function, Gradient Descent, and Chain Rule

Loss Function

This argument in details: segments 3.4 (binary linear classification), 5.4 (Linear regression)

The loss function, aka the *cost function* or the *race function*, is some way for us to value how far is our model from the data that we have.

We first define an "error" or "Loss". For example in Linear Regression the "error" is the Euclidean distance between the predicted and the observed value:

$L(x, y; \Theta) = \sum_{i=1}^{n} |\hat{y} - y| = \sum_{i=1}^{n} |\theta_1 x + \theta_2 - y|$

The objective is to minimise the loss function by changing the parameter theta. How?

Gradient Descent

This argument in details: segments 4.4 (gradient descent in binary linear classification), 4.5 (stochastic gradient descent) and 5.5 (SGD in linear regression)

The most common iterative algorithm to find the minimum of a function is the gradient descent.

We compute the loss function with a set of initial parameter(s), we compute the gradient of the function (the derivative concerning the various parameters), and we move our parameter(s) of a small delta against the direction of the gradient at each step:

$\hat{\theta}_{s+1} = \hat{\theta}_{s+1} - \gamma \nabla L(x, y; \theta)$

The γ parameter is known as the *learning rate*.

- too small learning rate: we may converge very slowly or end up trapped in small local minima;
- too high learning rate: we may diverge instead of converge to the minimum

Chain rule

How to compute the gradient for complex functions.

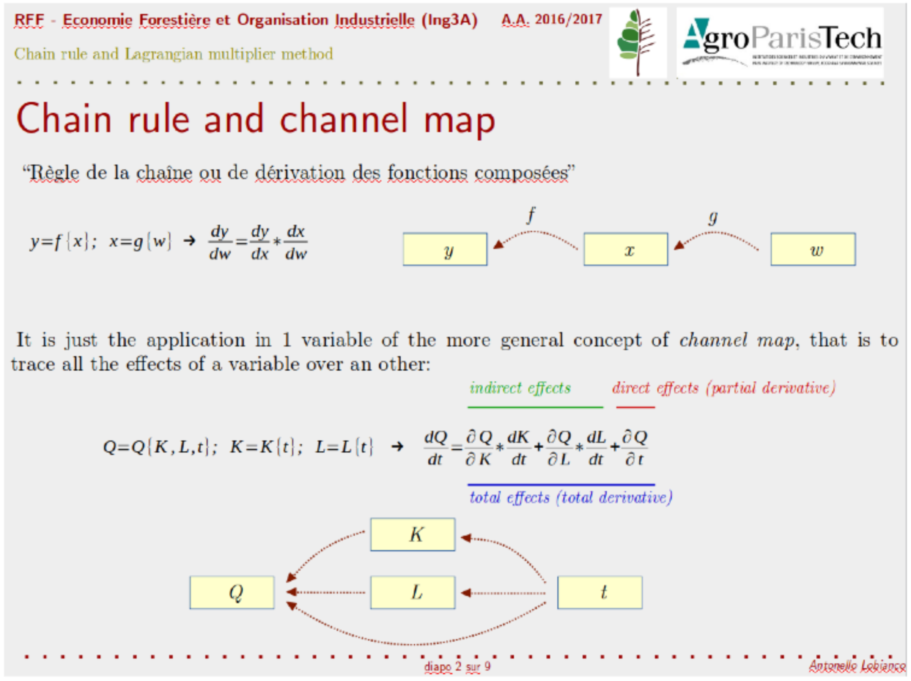
$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z} * \frac{\partial z}{\partial x}$

e.g. $\hat{y} = \frac{1}{1+e^{-(\theta_1 x+\theta_2)}} = (1+e^{-(\theta_1 x+\theta_2)})^{-1}$, $\frac{\partial \hat{y}}{\partial \theta_1} = -\frac{1}{(1+e^{-(\theta_1 x+\theta_2)})^2} * e^{-(\theta_1 x+\theta_2)} * -x$

For computing derivatives one can use SymPy, a library for symbolic computation. In this case the derivative can be computed with the following script:

```
from sympy import *
x, p1, p2 = symbols('x p1 p2')
y = 1/(1+exp( - (p1*x + p2)))
dy_dp1 = diff(y,p1)
print(dy_dp1)
```

It may be useful to recognise the chain rule as an application of the *chain map*, that is tracking all the effect from one variable to the other:



RFF - Economie Forestière et Organisation Industrielle (Ing3A)   A.A. 2016/2017

AgroParisTech

Chain rule and Lagrangian multiplier method

Chain rule and channel map

“Règle de la chaîne ou de dérivation des fonctions composées”

$y=f(x);\ x=g(w) \rightarrow \frac{dy}{dw}=\frac{dy}{dx}*\frac{dx}{dw}$

$$\begin{tikzcd} y & x \arrow[l, bend right, "f"'] & w \arrow[l, bend right, "g"'] \end{tikzcd}$$

It is just the application in 1 variable of the more general concept of *channel map*, that is to trace all the effects of a variable over an other:

$$Q=Q(K,L,t);\ K=K(t);\ L=L(t) \rightarrow \frac{dQ}{dt} = \underbrace{\overbrace{\frac{\partial Q}{\partial K}*\frac{dK}{dt}+\frac{\partial Q}{\partial L}*\frac{dL}{dt}}^{\text{indirect effects}}+\overbrace{\frac{\partial Q}{\partial t}}^{\text{direct effects (partial derivative)}}}_{\text{total effects (total derivative)}}$$

$$\begin{tikzcd} & K \arrow[dl, bend right] & \\ Q & L \arrow[l] & t \arrow[ul, bend right] \arrow[l] \arrow[ll, bend left] \end{tikzcd}$$

diapo 2 sur 9   Antonello Lobianco

Geometric progressions and series

Geometric progressions are sequence of numbers where each term after the first is found by multiplying the previous one by a fixed, non-zero number called the *common ratio*.

It results that geometric series can be wrote as $a, ar, ar^2, ar^3, ar^4, \ldots$ where r is the common ratio and the first value a is called the *scale factor*.

Geometric series are the sum of the values in a geometric progression.

Closed formula exist for both finite geometric series and, provided $|r| < 1$, for infinite ones:

- $\sum_{k=m}^{n} ar^k = \frac{a(r^m - r^{n+1}}{1-r}$ with $r \neq 1$ and $m < n$
- $\sum_{k=m}^{\infty} ar^k = \frac{ar^m}{1-r}$ with $|r| < 1$ and $m < n$.

Where m is the first element of the series that you want to consider for the summation. Typically $m = 0$, i.e. the summation considers the whole series from the scale factor onward.

For many more details on geometric progressions and series consult the relative excellent Wikipedia entry.

[MITx 6.86x Notes Index]

Unit 01 - Linear Classifiers and Generalizations

Course Introduction

There are lots and lots of applications out there, but what's so interesting about it is that, in terms of algorithms, there is a relatively small toolkit of algorithms you need to learn to understand how these different applications can work so nicely and be integrated in our daily life.

The course covers these 3 topics

Supervised Learning

- Make predictions based on a set of data for which correct examples are given
- E.g. Attach the label "dog" to an image, based on many couple (image/dog label) already provided
- The correct answers of the examples provided to the algorithm are already given
- What to produce is given explicitly as a target
- First 3 units in increasing complexity from simple linear classification methods to more complex neural network algorithms

Unsupervised Learning

- A generalisation of supervised learning
- The target is no longer given
- "Learning" to produce output (still at the end, "make predictions"), but now just observing existing outputs, like images, or molecules
- Algorithms to learn how to generate things
- We'll talk about probabilistic models and how to generate samples, mix and match (???)

Reinforced Learning

- Suggest how to act in the world given a certain goal, how to achieve an objective optimally
- It involves making predictions, aspects of SL and UL, but also it has a goal
- Learning from the failures, how to do things better
- E.g. a robot arm trying to grasp an object
- Objective is to control a system

Interesting stuff (solving real problems) happens at the mix of these 3 categories.

Lecture 1. Introduction to Machine Learning

Slides

1.1. Unit 1 Overview

Unit 1 will cover Linear classification

Exercise: predicting whether an Amazon product review, looking at only the text, will be positive or negative.

1.2. Objectives

Introduction to Machine Learning

At the end of this lecture, you will be able to:

- understand the goal of machine learning from a movie recommender example
- understand elements of supervised learning, and the difference between the training set and the test set
- understand the difference of classification and regression - two representative kinds of supervised learning

1.3. What is Machine Learning?

Many examples: Interpretation of web search queries, movie reccomendations, digital assistants, automatic translations, image analysis… playing the go game

Machine learning as a discipline aims to design, understand, and apply computer programs that learn from experience (i.e. data) for the purpose of **modelling**, **prediction**, and **control**.

There are many ways to learn or many reasons to learn:

- You can try to model, understand how things work;
- You can try to predict about, say, future outcomes;
- Or you can try to control towards a desired output configuration.

Prediction

We will start with prediction as a core machine learning task. There are many types of predictions that we can make.:

- We can predict outcomes of *events that occur in the future* such as the market, weather tomorrow, the next word a text message user will type, or anticipate pedestrian behavior in self driving vehicles, and so on.
- We can also try to predict *properties that we do not yet know*. For example, properties of materials such as whether a chemical is soluble in water, what the object is in an image, what an English sentence translates to in Hindi, whether a product review carries positive sentiment, and so on.

1.4. Introduction to Supervised Learning

Common to all these “prediction problems" mentioned previously is that they would be very hard to solve in a traditional engineering way, where we specify rules or solutions directly to the problem. It is far easier to provide examples of correct behavior. For example, how would you encode rules for translation, or image classification? It is much easier to provide large numbers of translated sentences, or examples of what the objects are on a large set of images. I don’t have to specify the solution method to be able to illustrate the task implicitly through examples. The ability to learn the solution from examples is what has made machine learning so popular and pervasive.

We will start with supervised learning in this course. In supervised learning, we are given an example (e.g. an image) along with a target (e.g. what object is in the image), and the goal of the machine learning algorithm is to find out how to produce the target from the example.

More specifically, in supervised learning, we hypothesize a collection of functions (or mappings) parametrized by a parameter (actually a large set of parameters), from the examples (e.g. the images) to the targets (e.g. the objects in the images). The machine learning algorithm then automates the process of finding the parameter of the function that fits with the example-target pairs the best.

We will automate this process of finding these parameters, and also specifying what the mappings are.

So this is what the machine learning algorithms do for you. You can then apply the same approach in different contexts.

1.5. A Concrete Example of a Supervised Learning Task

Let’s consider a movie recommender problem. I have a set of movies I’ve already seen (the **training set**), and I was to use the experience from those movies to make recommendations of whether I would like to see tens of thousands of other movies (the **test set**).

We will compile a **feature vector** (a binary list of its characteristics… genre, director, year…) for each movie in the training set as well for those I want to test (we will denote these feature vectors as x).

I also give my recommendation (again binary) if I want to see again or not the films in the training set.

Now I have the training set (feature vectors with associated labels), but I really wish to solve the task over the test set. It is this discrepancy that makes the learning task interesting. I wish to generalize what I extract from the training set and apply that information to the test set.

More specifically, I wish to learn the mapping from x to labels (plus minus one) on the basis of the training set, and hope and guarantee that that mapping, if applied now in the same way to the test examples, it would work well.

I want to predict the films in the test set based on their characteristics and my previous recommendation on the training set.

1.6. Introduction to Classifiers: Let’s bring in some geometry!

On the basis of the training set, pairs of feature vectors associated with labels, I need to learn to map each each x, each future vector, to a corresponding label, which is a plus or minus 1.

What we need is a **classifier** (here denoted with h) that maps from points to corresponding labels.

So what the classifier does is divide the space into essentially two halves – one that’s labeled 1, and the other one that’s labeled minus 1. –> a “linear classifier” is a classifier that perform this division of the full space in the two half linearly

We need to now, somehow, evaluate how good the classifier is in relation to the training examples that we have. To this end, we define something called **training error** (here denoted with ϵ) It has a subscript n that refers to the number of training examples that I have. And I apply it to a particular classifier. I evaluate a particular classifier, in relation to the training examples that I have.

$\epsilon_n(h) = \sum_{i=1}^{n} \frac{[\![h(x^i) \neq y^i]\!]}{n}$

(The double brackets denotes a function that takes a comparison expression inside and returns 1 if the expression is true, 0 otherwise. It is basically an indicator function.)

In the second example of classifier given in the lesson, the training error is 0. So, according to just the training examples, this seems like a good classifier.

The fact that the training error is zero however doesn’t mean that the classifier is perfect!

Let’s now consider a non-linear classifier.

We can build an “extreme” classifier that accept as +1 only a very narrow area around the +1 training set points.

As a result, this classifier would still have training error equal to zero, but it would classify all the points in the test set, no matter how much close to the +1 training set points they are, as -1 !

what is going on here is an issue called **generalization** --how well the classifier that we train on the training set generalizes or applies correctly, similarly to the test examples, as well? This is at the heart of machine-learning problems, the ability to generalize from the training set to the test set.

The problem here is that, in allowing these kind of classifiers that wrap themselves just around the examples (in the sense of allowing any kind of non-linear classifier), we are making the **hypothesis class**, the set of possible classifiers that we are considering, too large. We improve generalization, we generalize well, when we only have a very limited set of classifiers. And, out of those, we can find one that works well on the training set.

The more complex set of classifiers we consider, the less well we are likely to generalize (this is the same concept as overfitting in a statistical context).

We would wish to, in general, solve these problems by finding a small set of possibilities that work well on the training set, so as to generalize well on the test set.

Training data can be graphically depicted on a (hyper)plane. Classifiers are mappings that take feature vectors as input and produce labels as output. A common kind of classifier is the linear classifier, which linearly divides space (the hyperplane where training data lies) into two. Given a point x in the space, the classifier h outputs $h(x) = 1$ or $h(x) = -1$, depending on where the point x exists in among the two linearly divided spaces.

Each classifier represents a possible “hypothesis" about the data; thus, the set of possible classifiers can be seen as the space of possible hypothesis

1.7. Different Kinds of Supervised Learning: classification vs regression

A supervised learning task is one where you have specified the correct behaviour. Examples are then feature vector and what you want it to be associated with. So you give “supervision” of what the correct behaviour is (from which the term).