This lecture focus on Learning Machines. The ML function ($f(D_n)$ is a blackbox that takes $\mathscr{X}$ and produces output $\mathscr{Y}$. where D_n is the training data consisting of $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$
pairs (n >1).

Once the learning function is trained on the training data, we use ($f(D_n)$ to predict the output y^* for a new input x^*.

Least Square Regression

Here $\mathscr{X} \in \mathbb{R}$ and $\mathscr{Y} \in \mathbb{R}$. We can define $f(\mathscr{X}) = W^T x + b$.

w – vector of features.

b – intercept

$$\min_{w,b} = \sum_{i=1}^{n} (y_i - (w^T x_i + b))^2 \tag{1}$$

Learning is solving the above equation. The RHS of the equation is called as emperical risk.

Loss function

$l(y, u) = (y - u)^2$ u can be anything. You substitute u to find the loss.

In ML there are many degrees of freedom.

1. (X) - Input space
2. (Y) - Output space
3. Hypothesis class (function space where f lives)
4. Loss function

Using these four ingredients we can tackle any real world problems.

You want to design a machine, when given an images tells you if an object is in a image or not. (image classification)
input space - raw image (space of all images in the world, image is a grid)

$\mathscr{X}$ = { list of images }. This is not a good idea as the data is raw. Machine needs something that can be represented as a vector or something that the machine can understand.

So we vectorize $\mathscr{X}$.

$\mathscr{X}$ – { set of vectors. } $\mathscr{X} \in \mathbb{R}^d$ where d is an integer in $\mathbb{N}$ (space of all points in the plane of dimension d).

Feature representation

Representing raw data as feature vector is called as *feature representation*

Two ways to do feature representation.

1. For each field there will be experts. We rely on the experts to represent the data in a convenient mathematical form.
2. We can use ML to identify feature vector itself.

Data scientist assumes that feature vector is available. Feature representation might take decades to be perfected. The vectors might be huge and not in a easily computable form. But it is still better than the raw data.

Hypothesis class

Simplest function – *constant value funciton.*

$\forall x \in \mathscr{X}$, f(x) = k;

Affine functions $f(x) = W^T x + b$

Beyond constant value funcitons and affine functions we have polynomial functions. We will see that affine functions takes us very very far. Reason being that a good feature representation works well even with a linear learning machine.

When features are crap, we migh tneed complex hypothesis functions.

Output space

Suppose we want to identify Taylor Swift.

y = { 0, 1 } – binary classification. 0 – Not Taylor Swift. 1 – Taylor Swift.

Suppose we have k possible classifications, y = $\{0,1\}^k \rightarrow$ k class classification or k-way classification. (Vector of size k and each can be zero or one).

If we need structured output such as string, say speech recording into strings :

y = {space of all strings }

y = $\mathbb{R}^k$ (real value vector to the power of k). When we are solvingusing spatial coordinates — k = 3.

Loss function

(most important)
You assume that you are going to evaluate the performance of your machine on some data.

Performance Metrices

$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ — training data.

$$R(f) = \frac{1}{n} \sum_{1}^{n} l(y_i, f(x_i)) \tag{2}$$

You take the data, leran how to answer on one part and measure on the next part.

Over fitting

– taking your training data and fitting it so well that you dont learn much.
Say, we have n labelled training examples:
Assume training data (x_i, y_i) is drawn from $\mathbb{P}\mathrm{x,y}$ (Probability distribution).
$(x_i, y_i) \approx \mathbb{P}\mathrm{x,y}$ (Identity distributed - independent)
We care about the expected risk R(f).

$$R(f) = \mathbb{E}_{\mathbb{P}\mathrm{x,y}} = \int \mathrm{l} * \mathrm{y}, \mathrm{f(x)} \mathbb{P}\mathrm{xydx}, \mathrm{dy}$$

$$f = \{(x_1^{test}, y_1^{test}), (x_2^{test}, y_2^{test}), ..., (x_m^{test}, y_m^{test})\}$$

$$R_{test}(f) = \sum_{j=1}^{m} l(y_j, x_j)$$

Difference between statistics and ML

In stat, you assume that the model follows some physical considerations. In ML, you dont assume any model, instead you assume that there is an underlying Probability distribution. You only need PDF to get the results for new inputs.

In real world, either the model is not available or is too complicated. ML always focuses on decision making.

How to validate a model

1. Train your machine on $\mathscr{D}_n$
2. Test your machine on T_m
3. Important question here is: How to create T_m

Issues with dividing the data arbitrarily

If we are dealign with classification (say binary), we need to make sure that the sets are balanced. Holds true for k-class classificaiton too.

Simplest way to solve this problem is to brute-force and check if things are balanced. (Will be covered in the following course with scikit learn. Scikit learn has ways to divide the data).

Skewed data

If data is skewed, we need to check if the data represents the real world or if the data is not good. (Examples of real world data that might be skewed: Say we want to identify a very rare bird which can be sighted only a few times in a life time. We have methods to transfer the features from other bird samples to this bird.)

Dealing with imbalance

Case 1: Imbalance caused by data collection.

Case 2: Imbalance is inherent to the task (Spam classification).

Holdout validation

Taking your data and dividing into training and test set is called as Holdout validation.

Cross validatiaon

Generate many splits and validate across splits.

Say you have 5 splits of data. You rotate the test set from 4, 0, 1, 2, 3.

Cross validation is more roboust assesment of real life data.

Advice on dealing with data

Always normalize data. Assume $\mathscr{X} \in \mathbb{R}^{\mathrm{d}}$ and assume data can range from $[0\ to\ 255]^3$. Convert the scale to $[0\ to\ 1]^3$.

Most of the algorithms we use will assume that data is normalized.

Methods to normalize

1. Scaling
2. Centering (if data is poission distributed)

Centering

Centering means $\forall j \in [1, d]$ (d dimensions), We want to make sure that the emperical value $\mathbb{E}(\mathrm{x} : \mathrm{i}, \mathrm{j}) = 0$ We can also say that the variance is 1.

$$\frac{1}{n}\sum_{1}^{n}(x_{i,j})^2 = 1(unit\ variance)$$

Why would you do centering?

Training is equivalent to doing some form of gradient descent. For that we need to calculate the gradient. When data is not centered, the conditioning is not good. Centering helps in computing a proper conditioned data. If we dont center, we will run into numerical instabilities.

Missing data

In real world we have NA and MD(missing data). Most ML algorithms break when there is missing data. We need to take care before giving the data to algorithm. (Missing data imputation). This is beyond the scope of this course.

Two ways to deal with missing data

1. Get the median
2. Fit with the rest of the data.

Integral valued data

If you have integral value data, you should first wonder why it is integra?.

1. **Missed notes for age and binary representation**
2. When data is in histogram

When data is in histogram

Trick to amplify low counts and de-amplify high counts.

Histogram is a vecotr $(P_1, P_2, ..., P_N)$ and N bins.

$$\forall j, P_j > 0,\ do\ (counts/frequencies)$$

$$\sum_{j=1}^{N} P_j = 1(Probability\ simplex)$$

We have to convert the histogram into euclidean space. We need a transform for that. The below function does the trick.

$$u_j = \frac{\sqrt{P_j}}{\sum_{j=1}^{N} \sqrt{P_j}}$$

Once we have moved the histogram to euclidean space, we have something that makes sense for doing dot product.