Overview

Fuhao Zou(邹复好)

Intelligent and Embedded Computing Lab,
Huazhong University of Science & Technology

fuhao_zou@hust.edu.cn

2019年04月09日

Table of contents

Table of Contents

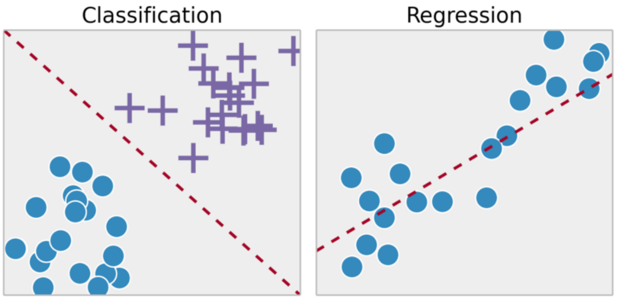
What is machine learning?

machine learning

- A branch of artificial intelligence, concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data.
- As intelligence requires knowledge, it is necessary for the computers to acquire knowledge.

- Study of algorithms that improve their performance at some task with experience
- Optimize a performance criterion using example data or past experience
- Role of Statistics: Inference from a sample
- Role of Computer science: Efficient algorithms to Solve the optimization problem Representing and evaluating the model for inference

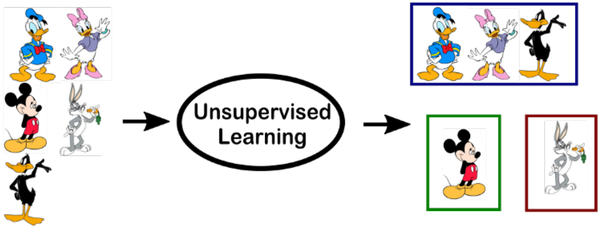
Supervised Learning



Supervised learning, an algorithm learns from a training dataset. We know the correct answers or desired output, the algorithm makes predictions using the given dataset and is corrected by the “supervisor”. The learning stops as and when the algorithm achieves a level of performance which is acceptable. There are two types of supervised learning- regression and classification.

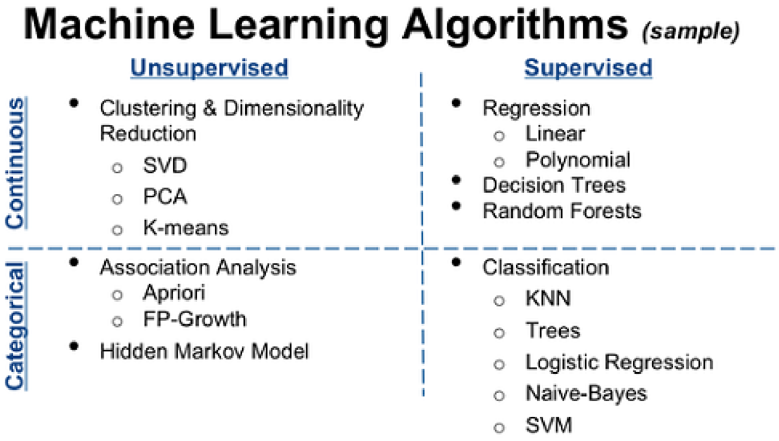
Regression models are applied to solve various problems; e.g. predicting stock price. (https://towardsdatascience.com/stock-prediction-in-python-

Unsupervised Learning



In unsupervised learning there is no trainer or "supervisor" as the name suggests. We do not show the output, or the specific input required to achieve specific output. The machine learns based on its own capability, it decides what should be the input and output- clustering is a method of unsupervised learning, where the machine learns on its own. A well known algorithm in clustering is k-means clustering.

Supervised Learning and Unsupervised Learning



Machine Learning Algorithms *(sample)*

	Unsupervised	**Supervised**
Continuous	• Clustering & Dimensionality Reduction ○ SVD ○ PCA ○ K-means	• Regression ○ Linear ○ Polynomial • Decision Trees • Random Forests
Categorical	• Association Analysis ○ Apriori ○ FP-Growth • Hidden Markov Model	• Classification ○ KNN ○ Trees ○ Logistic Regression ○ Naive-Bayes ○ SVM

Table of Contents

Examples of feature vectors

Let us formalize the supervised machine learning setup. Our training data comes in pairs of inputs $(\mathbf{x}, y)$, where $\mathbf{x} \in \mathcal{R}^d$ is the input instance and y its label. The entire training data is denoted as

$$D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\} \subseteq \mathcal{R}^d \times \mathcal{C}$$

where:

- $\mathcal{R}^d$ is the d-dimensional feature space
- $\mathbf{x}_i$ is the input vector of the i^{th} sample
- y_i is the label of the i^{th} sample
- $\mathcal{C}$ is the label space

The data points $(\mathbf{x}_i, y_i)$ are drawn from some (unknown) distribution $\mathcal{P}(X, Y)$. Ultimately we would like to learn a function h such that for a new pair $(\mathbf{x}, y) \sim \mathcal{P}$, we have $h(\mathbf{x}) = y$ with high probability (or $h(\mathbf{x}) \approx y$). We will get to this later. For now let us go through some examples of X and Y.

Examples of Label Spaces

There are multiple scenarios for the label space $\mathcal{C}$:

Binary classification	$\mathcal{C} = \{0, 1\}$ or $\mathcal{C} = \{-1, +1\}$.	Eg. spam filtering. An email is either spam $(+1)$, or not (-1).
Multi-class classification	$\mathcal{C} = \{1, 2, \cdots, K\}$ $(K \geq 2)$.	Eg. face classification. A person can be exactly one of K identities (e.g., 1="Barack Obama", 2="George W. Bush", etc.).
Regression	$\mathcal{C} = \mathbb{R}$	Eg. predict future temperature or the height of a person.

Table of Contents

What’s a Loss Function

What’s a Loss Function

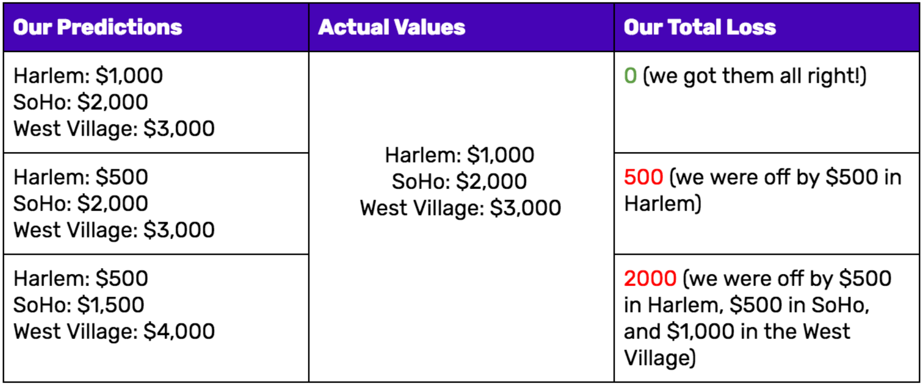
At its core, a loss function is incredibly simple: it’ s a method of evaluating how well your algorithm models your dataset. If your predictions are totally off, your loss function will output a higher number. If they’ re pretty good, it’ ll output a lower number. As you change pieces of your algorithm to try and improve your model, your loss function will tell you if you’ re getting anywhere.

What’s a Loss Function

In fact, we can design our own (very) basic loss function to further explain how it works. For each prediction that we make, our loss function will simply measure the absolute difference between our prediction and the actual value. In mathematical notation, it might look something like $abs(y_predictedy)$.

What’s a Loss Function

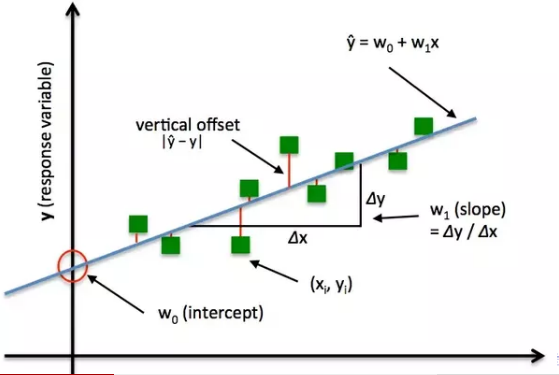
Here’ s what some situations might look like if we were trying to predict how expensive the rent is in some NYC apartments:



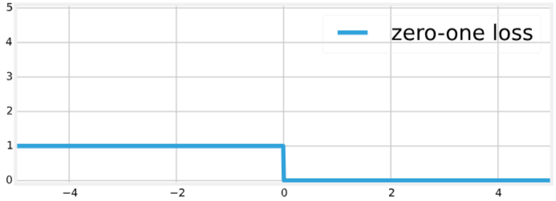
Our Predictions	Actual Values	Our Total Loss
Harlem: $1,000 SoHo: $2,000 West Village: $3,000	Harlem: $1,000 SoHo: $2,000 West Village: $3,000	0 (we got them all right!)
Harlem: $500 SoHo: $2,000 West Village: $3,000		500 (we were off by $500 in Harlem)
Harlem: $500 SoHo: $1,500 West Village: $4,000		2000 (we were off by $500 in Harlem, $500 in SoHo, and $1,000 in the West Village)

What’s a Loss Function

Notice how in the loss function we defined, it doesn’t matter if our predictions were too high or too low. All that matters is how incorrect we were, directionally agnostic. This is not a feature of all loss functions: in fact, your loss function will vary significantly based on the domain and unique context of the problem that you’re applying machine learning to. In your project, it may be much worse to guess too high than to guess too low, and the loss function you select must reflect that.



Zero-one loss



Zero-one loss

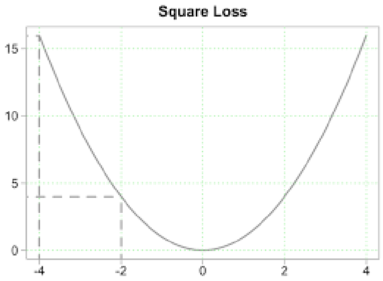
Formally, the zero-one loss can be stated has:

$$\mathcal{L}_{0/1}(h) = \frac{1}{n}\sum_{i=1}^{n} \delta_{h(\mathbf{x}_i)\neq y_i}, \text{ where } \delta_{h(\mathbf{x}_i)\neq y_i} = \begin{cases} 1, & \text{if } h(\mathbf{x}_i) \neq y_i \\ 0, & \text{o.w.} \end{cases}$$

This loss function returns the error rate on this data set D. For every example that the classifier misclassifies (i.e. gets wrong) a loss of 1 is suffered, whereas correctly classified samples lead to 0 loss.

Disadvantage: The 0-1 loss imposes the same penalty for each misclassified point, so

Squared loss



The squared loss function is typically used in regression settings. Formally the squared loss is:

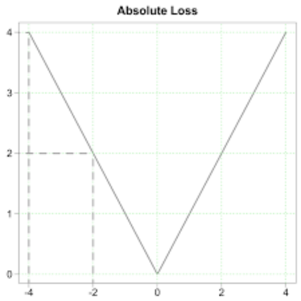
$$\mathcal{L}_{sq}(h) = \frac{1}{n}\sum_{i=1}^{n}(h(\mathbf{x}_i) - y_i)^2.$$

The squaring has two effects:

- 1., the loss suffered is always nonnegative;
- 2., the loss suffered grows quadratically with the absolute mispredicted amount.

Disadvantage: if a prediction is very close to be correct, the square will be tiny and little

Absolute loss



Similar to the squared loss, the absolute loss function is also typically used in regression settings. It suffers the penalties $|h(\mathbf{x}_i) - y_i|$. Because the suffered loss grows linearly with the mispredictions it is more suitable for noisy data (when some mispredictions are unavoidable and shouldn't dominate the loss). If, given an input $\mathbf{x}$, the label y is probabilistic according to some distribution $P(y|\mathbf{x})$ then the optimal prediction to minimize the absolute loss is to predict the median value, i.e. $h(\mathbf{x}) = \mathrm{MEDIAN}_{P(y|\mathbf{x})}[y]$. Formally, the absolute loss can be stated as:

Table of Contents

Generalization

Given a loss function, we can then attempt to find the function h that minimizes the loss:

$$h = \mathrm{argmin}_{h \in \mathcal{H}} \mathcal{L}(h)$$

A big part of machine learning focuses on the question, how to do this minimization efficiently.

If you find a function $h(\cdot)$ with low loss on your data D, how do you know whether it will still get examples right that are not in D?

Bad example: "memorizer" $h(\cdot)$

$$h(x) = \begin{cases} y_i, & \text{if } \exists(\mathbf{x}_i, y_i) \in D, \text{ s.t., } \mathbf{x} = \mathbf{x}_i, \\ 0, & \text{o.w.} \end{cases}$$

For this $h(\cdot)$, we get 0% error on the training data D, but does horribly with samples not in D, i.e., there's the overfitting issue with this function.

What is overfitting?

Whenever working on a data set to predict or classify a problem, we tend to find accuracy by implementing a design model on first train set, then on test set. If the accuracy is satisfactory, we tend to increase accuracy of data-sets prediction either by increasing or decreasing data feature or features selection or applying feature engineering in our machine learning model.

But sometime our model maybe giving poor result. The poor performance of our model maybe because, the model is too simple to describe the target, or may be model is too complex to express the target.

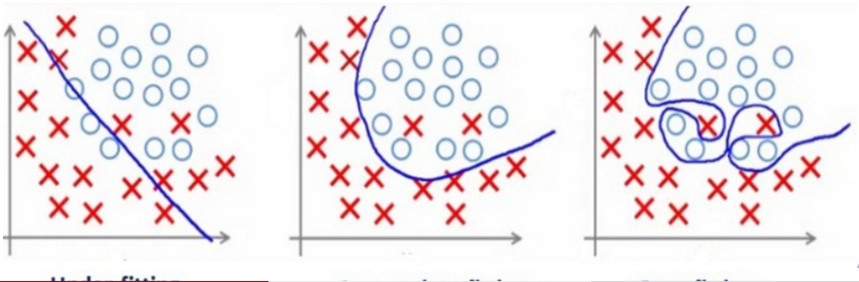
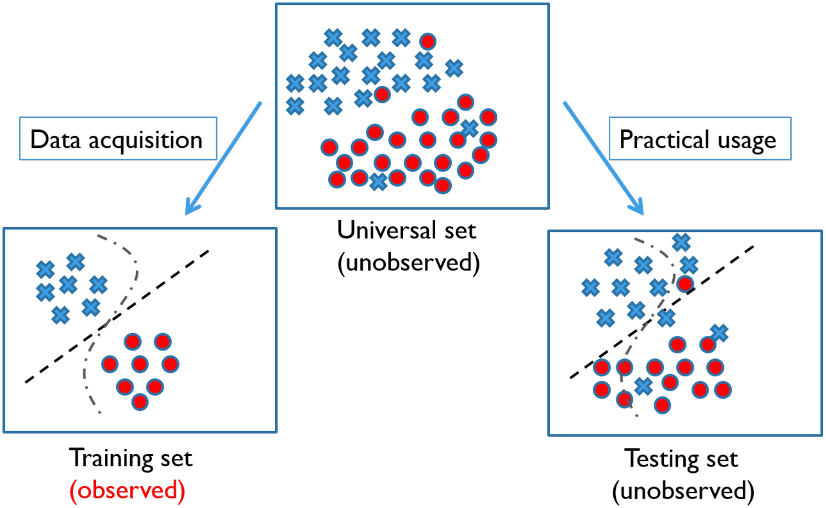


Table of Contents

Training and testing

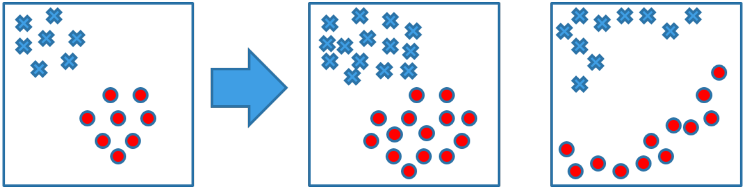


Training and testing

- Training is the process of making the model able to learn.

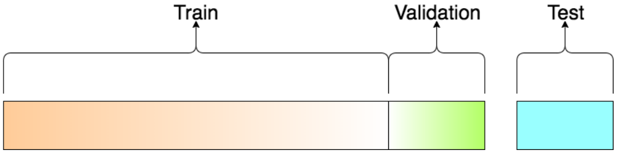
No free lunch rule:

- Training set and testing set come from the same distribution
- Need to make some assumptions or bias



Train / Test splits

To resolve the overfitting issue, we usually split D into three subsets: D_{TR} as the training data, D_{VA}, as the validation data, and D_{TE}, as the test data. Usually, they are split into a proportion of 80%, 10%, and 10%. Then, we choose $h(\cdot)$ based on D_{TR}, and evaluate $h(\cdot)$ on D_{TE}.



Quiz:Why do we need D_{VA}?

D_{VA} is used to check whether the $h(\cdot)$ obtained from D_{TR} suffers from the overfitting issue. $h(\cdot)$ will need to be validated on D_{VA}, if the loss is too large, $h(\cdot)$ will get revised based on D_{TR}, and validated again on D_{VA}. This process will keep going back and forth until it gives low loss on D_{VA}. Here's a trade-off between the sizes of D_{TR} and D_{VA}: the training results will be better for a larger D_{TR}, but the validation will be more reliable (less noisy) if D_{VA} is larger.

How to Split the Data?

You have to be very careful when you split the data in Train,Validation,Test. The test set must simulate a real test scenario, i.e. you want to simulate the setting that you will encounter in real life. For example, if you want to train an email spam filter, you train a system on past data to predict if future email is spam. Here it is important to split train / test temporally - so that you strictly predict the future from the past. If there is no such thing as a temporal component, it is often best to split uniformly at random. Definitely never split alphabetically, or by feature values. By time, if the data is temporally collected. In general, if the data has a temporal component, we must split it by time. Uniformly at random, if (and, in general, only if) the data is $i.i.d.$.

The test error (or testing loss) approximates the true generalization error/loss.

Table of Contents

Putting everything together

We train our classifier by minimizing the training loss:

$$\text{Learning: } h^*(\cdot) = \operatorname{argmin}_{h(\cdot)\in\mathcal{H}} \frac{1}{|D_{\mathrm{TR}}|} \sum_{(\mathbf{x},y)\in D_{\mathrm{TR}}} \ell(\mathbf{x}, y|h(\cdot)),$$

where $\mathcal{H}$ is the hypothetical class (i.e., the set of all possible classifiers $h(\cdot)$). In other words, we are trying to find a hypothesis h which would have performed well on the past/known data. We evaluate our classifier on the testing loss:

$$\text{Evaluation: } \epsilon_{\mathrm{TE}} = \frac{1}{|D_{TE}|} \sum_{(\mathbf{x},y)\in D_{\mathrm{TE}}} \ell(\mathbf{x}, y|h^*(\cdot)).$$

If the samples are drawn i.i.d. from the same distribution $\mathcal{P}$, then the testing loss is an unbiased estimator of the true generalization loss:

$$\text{Generalization: } \epsilon = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{P}}[\ell(\mathbf{x}, y|h^*(\cdot))].$$

Putting everything together

Quiz

Why does $\epsilon_{\mathrm{TE}} \to \epsilon$ as $|D_{\mathrm{TE}}| \to +\infty$? This is due to the weak law of large numbers, which says that the empirical average of data drawn from a distribution converges to its mean.

No free lunch

Every ML algorithm has to make assumptions on which hypothesis class $\mathcal{H}$ should you choose? This choice depends on the data, and encodes ¡u¿your assumptions¡/u¿ about the data set/distribution $\mathcal{P}$. Clearly, there's no one perfect $\mathcal{H}$ for all problems.

Example

Assume that $(\mathbf{x}_1, y_1) = (1, 1)$, $(\mathbf{x}_2, y_2) = (2, 2)$, $(\mathbf{x}_3, y_3) = (3, 3)$, $(\mathbf{x}_4, y_4) = (4, 4)$, and $(\mathbf{x}_5, y_5) = (5, 5)$.

Question

what is the value of y if $\mathbf{x} = 2.5$? Well, it is utterly impossible to know the answer without assumptions. The most common assumption of ML algorithms is that the

The End