# Designing Machine Learning Systems - Chip Huyen

## Preface

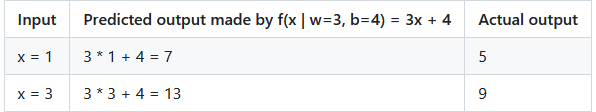
* This book assumes that readers have a basic understanding of the following topics:
* **ML models**such as **clustering, logistic regression, decision trees, collaborative filtering**, and **various neural network architectures** including **feed-forward, recurrent, convolutional, and transformer**
* **ML techniques**such as **supervised versus unsupervised, gradient descent, objective/loss function, regularization, generalization**, and **hyperparameter tuning**
* **Metrics**such as **accuracy, F1, precision, recall, ROC, mean squared error**, and **log-likelihood**
* **Statistical concepts**such as **variance, probability**, and **normal/long-tail distribution**
* **Common ML tasks**such as **language modeling, anomaly detection, object classification,** and **machine translation**
* *You don’t have to know these topics inside out but you should have a rough sense of what they mean going in*
* <https://github.com/chiphuyen/dmls-book>

## Basic ML Review

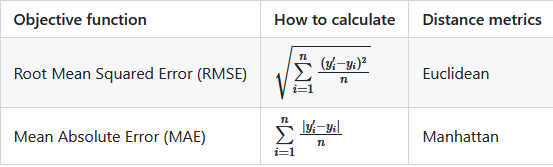
* A **model** is a **function that transforms inputs into outputs, which can then be used to make predictions**
* Ex: A binary text classification model might take sentences as inputs and output values between 0 and 1, which you can use to make predictions, such as if the value is < 0.5, output the NEGATIVE class, and if the value is >= 0.5, output the POSITIVE class.
* In *traditional programming*, functions are *given* and outputs are *calculated* from given inputs
* Ex: Your function f(x) might be given as: f(x) = 2x + 3
* Given x = 1, the output will be f(1) = 2 \* 1 + 3 = 5
* Given x = 3, the output will be f(3) = 2 \* 3 + 3 = 9.
* In *supervised ML*, the **inputs *and* outputs are given**, which are called **data**, and the **function is derived from data**
* Given x as input and y as output, **learn** **a function f** such that **applying f on x will produce y**
* However, **ML isn’t powerful enough to derive arbitrary functions from data yet**, so you still **need to specify the form that you think the function should take**
* It can be a linear function, a decision tree, a feedforward neural network with 2 hidden layers, each with 768 neurons
* Ex: Given a dataset with only 2 examples (x = 1, y = 5) and (x = 3, y = 9), you might specify that the function is a linear function, which means that it takes the form f(x) = wx + b
* Then you **learn the values of w and b to fit this dataset**
* Because w and b are **learned during the *training* process**, they are called **parameters**.
* **For each type of model, there are many possible values for the parameters**
* You need an **objective function** to **evaluate how good a given set of parameters is for a dataset**, and a **procedure to** **derive the set of parameters best suited for the given data according to that objective,** known as a **learning procedure**
* SIDEBAR: Some might wonder if the above paragraph about parameters still applies to ***non-parametric*** models such as K-means clustering and decision trees
* ***Being non-parametric doesn’t mean that models don’t use parameters***
* In a **parametric model**, the **number of parameters is fixed** with respect to the sample size
* In a **NON-parametric model**, the **effective number of parameters can grow with the sample size**
* So, complexity of the function underlying a NN remains the same, even if the amount of data grows
* *But complexity of the function underlying a decision tree grows as its # of nodes grows*
* When talking about **model selection**, *most people think about selecting a function form*
* However, **choosing the right objective function and a learning procedure is extremely important in finding a good set of parameters for your model**

### Objective Function

* The **objective function**, also known as the **loss function**, is **highly dependent on the model type and whether labels are available**
* If labels *aren’t* available (as in the case of ***unsupervised*** learning), the *objective functions depend on the data points themselves*
* Ex: For k-means clustering, the objective function = the variance within data points in the same cluster (so the objective = to put data points into clusters so that the within-cluster variance is minimized)
* *But unsupervised learning is much less commonly used in production*
* Most algorithms encountered in prod. = **supervised** or some form of **weakly or semi-supervised**
* *Given a set of parameter values*, you *calculate the outputs from the given inputs*, and *compare the given function’s predicted outputs (y') to the actual outputs (y)*
* **Objective functions evaluate how good a set of parameter values is by measuring the distance between the set of y' and the set of y**
* To make this concrete, go back to the example where we have only 2 data points (x = 1, y = 5) and (x = 3, y = 9)
* We want to find w and b such that f(x) = wx + b best suited this data
* Given the set of parameter values w = 3 and b = 4, we get the predicted outputs of 7 and 13 as shown in the table below
* The objective function measures the distance between the predicted outputs (7, 13) and the actual outputs (5, 9)



* There are many types of **distance metrics** you can use to derive objective functions
* When **outputs** are **scalars** (numbers), 2 common metrics are **Root Mean Squared Error** **(RMSE)** and **Mean Absolute Error (MAE)** as shown in the table below



* However, **many types of models don’t output just 1 number given an input, but output a** **distribution**
* For example, if your task has 3 classes: [cat, dog, chicken], your model might output an array of how likely it is that your input belongs to each class
* So, predicted output might look like [0.1, 0.5, 0.4], which means the input has 10% chance of being cat, 50% chance of being dog, 40% chance of being chicken
* The *actual* label for this example is chicken, so the output is [0, 0, 1]
* We want to measure the distance between predicted outputs that take the form [0.1, 0.5, 0.4] and actual outputs that take the form [0, 0, 1]
* In this case, the common objective function is **cross entropy and its variation.**
* You **can modify the objective function to enforce your model to learn a set of parameters with certain properties**
* Ex: Can modify the objective function to encourage a model to focus on examples of rare classes or examples that are difficult to learn
* Can also add **regularizers** such as **L1** and **L2** to your loss function to encourage a model to choose parameters of smaller values.
* **Each objective function gives you a set of possible values your parameters can take**
* This set of possible values for parameters is known as the **loss surface of a given objective function**
* *A small change to an objective function can give you a very different loss surface, which, in turn, gives you a very different function for your model*
* **Understanding the possible parameters given by different objective functions can help you choose the objective function best suited for your needs**
* *However, this understanding tends to require advanced linear algebra, so it’s common for ML engineers to use popular objective functions that are known to give decent performance for their problems without giving them much thought*
* While developing a model, if time permits, **experiment with different objective functions to see how your model’s behaviors change**, both globally on all your data or with respect to different slices of your data (You might be surprised)

### Learning Procedure

* **Learning procedures** = **the procedures that help your model find the set of parameters that minimize a given objective function for a given set of data**, and **they are diverse**
* In some cases, the parameters might be calculated exactly
* Ex: In the case of linear functions, the values of w and b can be calculated from the averages and variances of x and y
* **In most cases, however, the values of parameters can’t be calculated exactly and have to be approximated**, usually via an **iterative procedure**
* Ex: K-means clustering uses an iterative procedure called **expectation–maximization algorithm**.
* **The most popular family of iterative procedures today** is undoubtedly **gradient descent**
* The loss of a model at a given train step is given by the objective function
* The gradient of an objective function with respect to a parameter tells us how much that parameter contributes to the loss
* In other words, **the gradient = the direction that lowers the loss from a current value the most**
* The **idea** is **to subject that gradient value from that parameter, hoping that this would make the parameter contribute less to the loss, and eventually drive the loss down to 0**.
* Subtracting the raw gradient values from parameters doesn’t work extremely well
* **Transforming the gradient values first (such as multiplying the gradient value with 0.003) and *then* subtracting that transformed values from parameters helps models converge much faste**r
* The **function that determines how to update a parameter given a gradient value is called an update algorithm**, or an **optimizer**
* Common optimizers = Momentum, Adam, and RMSProp
* **Good optimizers can both speed up the model training process and help a model converge to a better set of parameters**
* *Even though optimizers help your model find the set of parameters that minimize a given objective function for a given set of data, the set of parameters that minimize the loss for your training data isn’t always the best optimizer for you, as you might want the parameters that will perform well on the data your model will encounter in production too*
* **While developing ML models, especially with gradient descent-based models, it’s often helpful to explore with different types of optimizers**