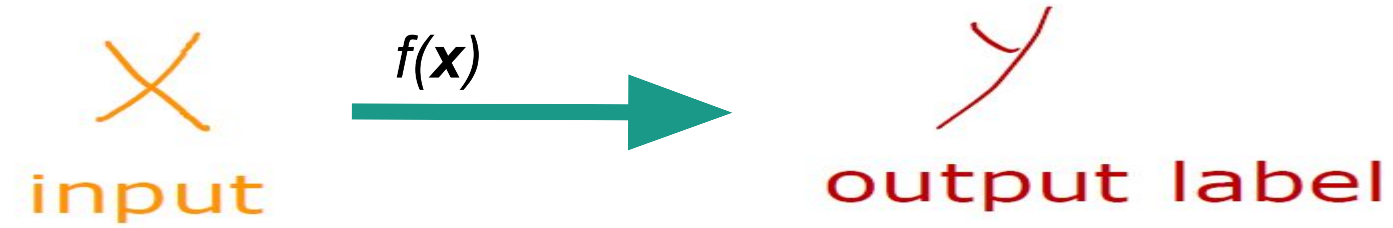
Supervised Learning

Ali Aburas, PhD

Slide credit:Andrew Ng

Supervised Learning

Learns from being given "Right Answers"



We consider systems that apply a function f( ) to input items x and return an output y=f(x).

In (supervised) machine learning, we deal with systems whose *f(x)* is learned from examples.

- Output labels $y \in Y$ are categorical:
  - Binary classification: Two possible labels
  - Multiclass classification: k possible labels
- Output labels $y \in Y$ are numerical:
  - Regression (linear/polynomial):
    - Labels are continuous-valued

Problem Formulation

- Let us formalize the supervised machine learning setup. Our training data comes in pairs of inputs (x, y) , where $X \in R^d$ 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:
  - R^d is the d-dimensional feature space
  - x_i is the input vector of the i^{th} sample
  - y_i is the label of the i^{th} sample
  - C is the label space
- **Examples of Label Spaces**
  - There are multiple scenarios for the label space C

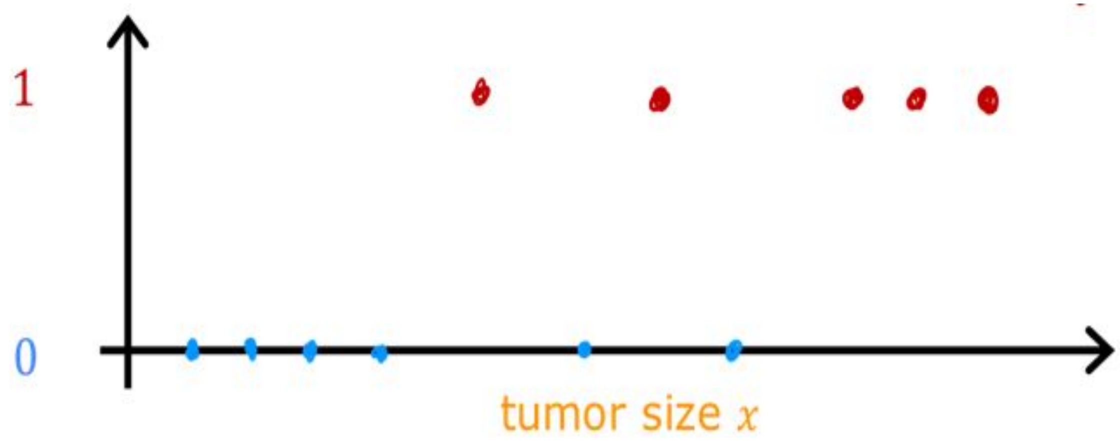
Binary classification	$\mathcal{C} = \{0, 1\}$ or $\mathcal{C} = \{-1, +1\}$.	Eg. spam filtering. An email is either spam (+1), or not (−1).
Multi-class classification	$\mathcal{C} = \{1, 2, \cdots, K\}$ $(K \geq 2)$.	Eg. face classification. A person can be exactly one of K identities (e.g., 1="Barack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

Binary classification: Breast cancer detection

A binary classifier, capable of distinguishing between just two classes

1	Malignant
0	Benign

Size	Diagnosis
2	0
5	1
7	1
1	0
..	..
..	..



Multi-class classification: Breast cancer detection

○ benign

✕ malignant type1

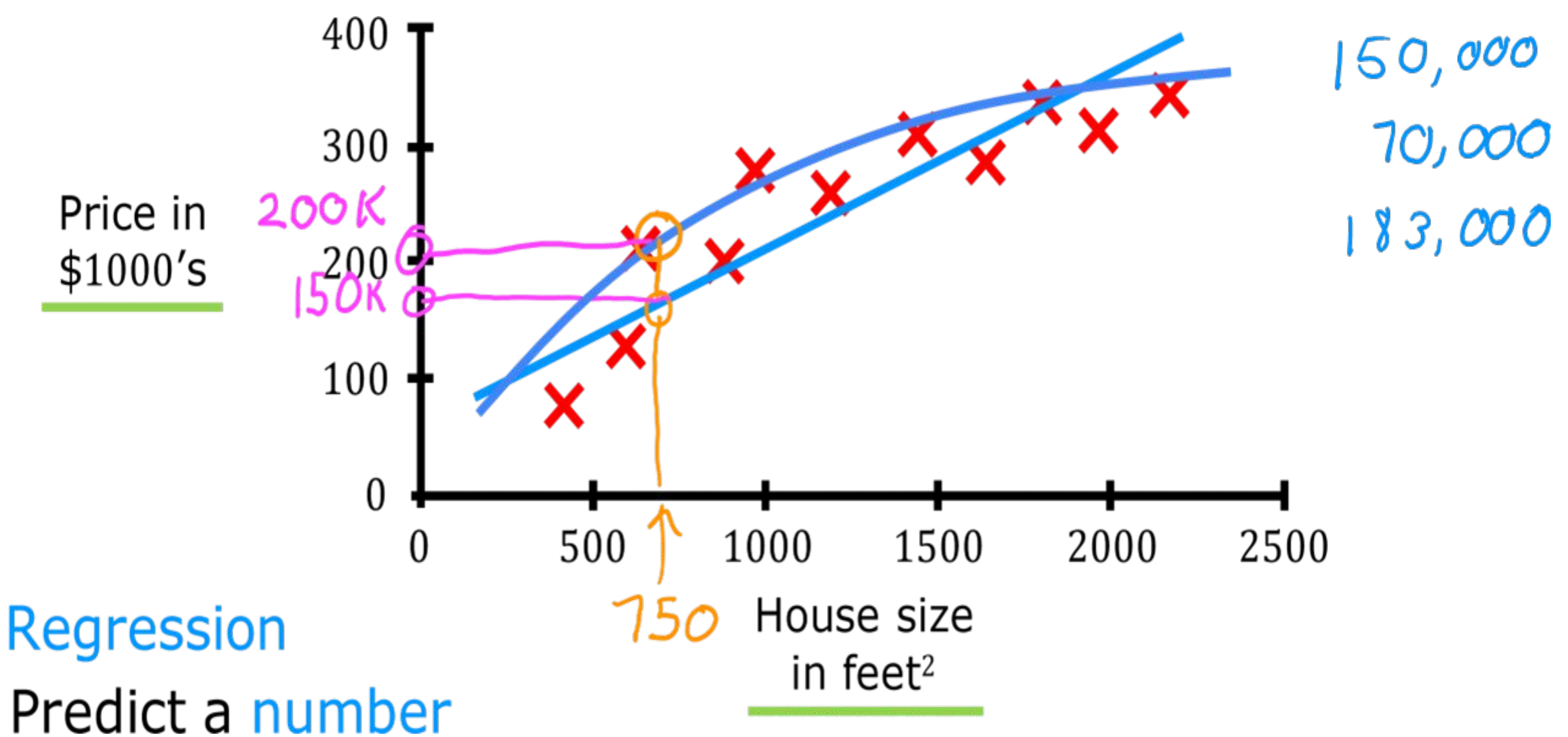
△ malignant type 2

Classification

predict categories cat dog benign malignant 0, 1, 2

small number of possible outputs

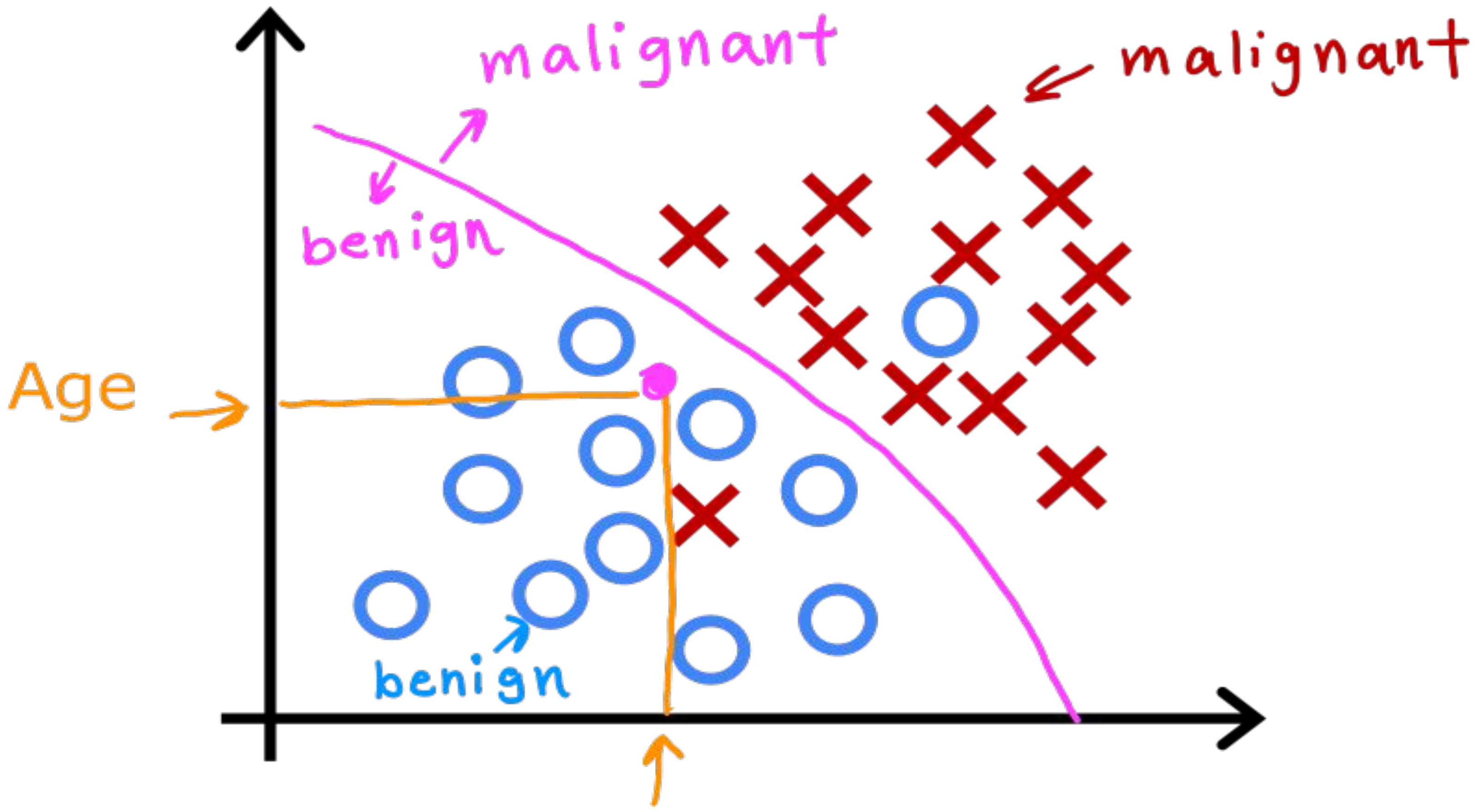
Regression: Housing price prediction



Regression

Predict a number

Two or more inputs



tumor size

Key Issues in Machine Learning

- **Modeling**
  - How to formulate application problems as machine learning problems ? How to represent the data?
  - Learning Protocols (where is the data & labels coming from?)
- **Representation**
  - What **functions** should we learn (**hypothesis spaces**) ?
  - How to map raw **input** to an instance space?
    - Any rigorous way to find these? Any general approach?
- **Algorithms**
  - What are good algorithms?
  - How do we define success?
    - Generalization vs. over fitting
  - The computational problem

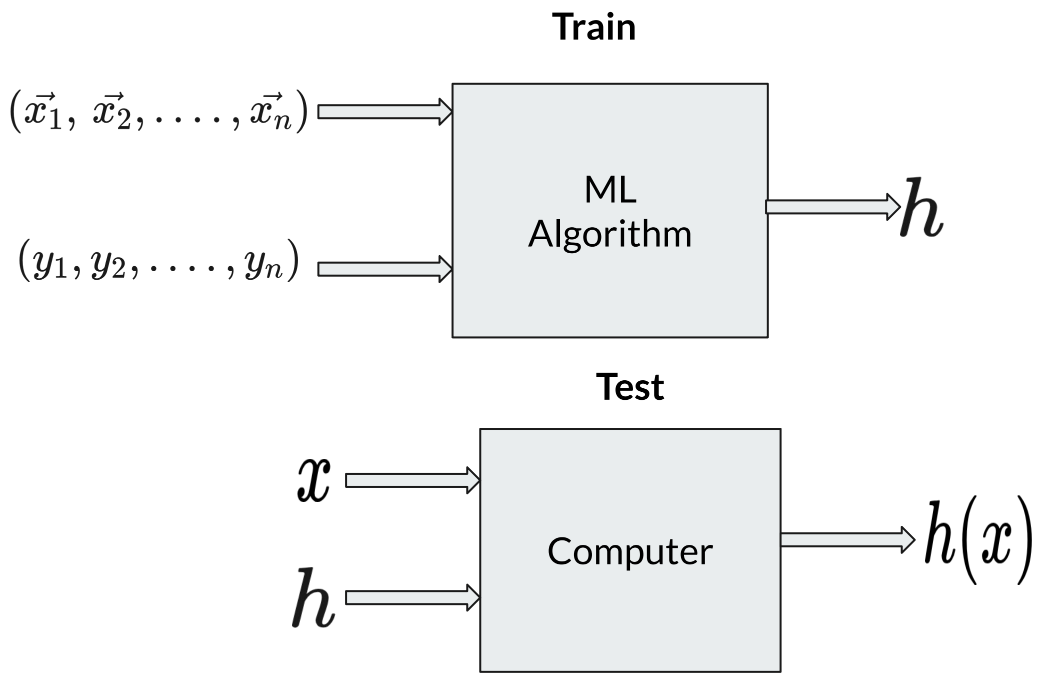
A Typical (Supervised) Machine Learning Routine

- Step 1 – training
  - Input: a **labelled training dataset**
  - Output: a classifier
- Step 2 – testing
  - Inputs: **a classifier, a test dataset**
  - Output: predictions for each test data point
- Step 3 – evaluation
  - Inputs: predictions from step 2, test dataset labels
  - Output: some measure of how good the predictions are; usually (but not always) error rate

Examples of feature vectors

We call x_i a feature vector. Each one of its $\boldsymbol{d}$ dimensions is a features describing the i–th sample. Let us look at some examples:

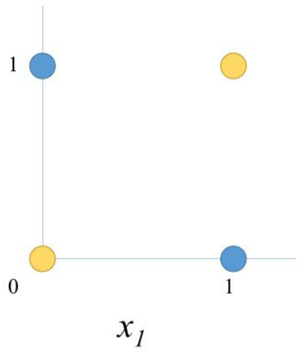
- Patient Data in a hospital. $\mathbf{x}_i = (x_i^1, x_i^2, \cdots, x_i^d)$, where $x_i^1 = 0$ or 1, may refer to the patient i's gender, x_i^2 could be the height of patient i in cm, and x_i^3 may be his/her in years, etc. In this case, $d \leq 100$ and the feature vector is dense, i.e., the number of nonzero coordinates in $\mathbf{x}_i$ is large relative to d.


Train
$(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n)$
$(y_1, y_2, \ldots, y_n)$
ML Algorithm
h
Test
x
h
Computer
$h(x)$

Model hypothesis (*h*)

- A classification hypothesis maps features to labels
  - $h\colon \mathbb{R}^d \mapsto C$
- Our goal (usually*) is to find h such that $\forall x, h(x) = \arg\max_y P(y|x)$
  - (*anomaly detection)
- Does such an h always exist?
  - No! we are limited by our model assumption
  - A linear model , for example, cannot capture a XOR distribution
  - Other models might be more flexible yet are still limited

x	y	x xor y
0	0	0
0	1	1
1	0	1
1	1	0



Hypothesis classes

- The data points $(\mathbf{x}_i, y_i)$ are drawn from some (unknown) distribution P(X,Y)
- Ultimately we would like to learn a function ***h*** such that for a new pair (x,y)~P
- There are typically two steps involved in learning a hypothesis function ***h()***
  - First, we select the type of machine learning algorithm that we think is appropriate for this particular learning problem. This defines the hypothesis class ***H*** , i.e. the set of functions we can possibly learn.
  - The second step is to find the best function within this class, $h \in \mathcal{H}$. This second step is the actual learning process and often, but not always, involves an optimization problem.
- Essentially, we try to find a function ***h*** within the hypothesis class that makes the fewest mistakes within our training data.

Loss Function

The question: How can we find the best function?

- For this we need some way to evaluate what it means for one function to be better than another. This is where the loss function $h \in \mathcal{H}$ (aka risk function) comes in.
- A loss function evaluates a hypothesis h∈H on our training data and tells us how bad it is.
- The **higher the loss**, the **worse it is -** a loss of zero means it makes perfect predictions.
- It is common practice to normalize the loss by the total number of training samples, n, so that the output can be interpreted as the average loss per sample (and is independent of n).

loss function: Zero-one loss

- The simplest **loss function** is the zero-one loss. It literally counts how many mistakes an hypothesis function h makes on the training set.
- For every single example it suffers a loss of 1 if it is mispredicted, and 0 otherwise. The normalized zero-one loss returns the fraction of misclassified training samples, also often referred to as the training error.
- The zero-one loss is often used to evaluate classifiers in multi-class/binary classification settings but rarely useful to guide optimization procedures because the function is non-differentiable and non-continuous.
- Formally, the zero-one loss can be stated has:

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

loss function: Squared loss

- The **squared loss function** is typically used in regression settings. It iterates over all training samples and suffers the loss $(h(\mathbf{x}_i) - y_i)^2$.
- The squaring has two effects: 1., the loss suffered is always nonnegative; 2., the loss suffered grows quadratically with the absolute mispredicted amount.
- The latter property encourages no predictions to be really far off (or the penalty would be so large that a different hypothesis function is likely better suited).
- On the flipside, if a prediction is very close to be correct, the square will be tiny and little attention will be given to that example to obtain zero error.
- For example, if $|h(\mathbf{x}_i) - y_i|$=0.001 the squared loss will be even smaller, 0.000001, and will likely never be fully corrected.
- Formally the squared loss is:

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

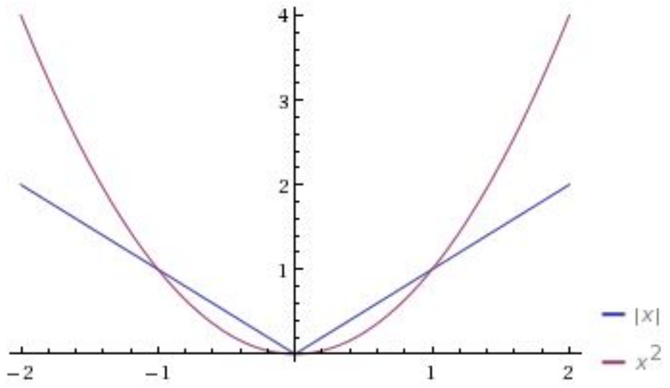
loss function: Absolute loss

- 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).
- Formally, the absolute loss can be stated as:

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

Choosing Between MSE and MAE:

- To decide between **MSE** and **MAE**, it's crucial to assess the nature of your data.
- If your dataset includes outliers — data points that don't conform to the general pattern — it's advisable to opt for ***MAE***.
- By treating all errors equally, ***MAE*** provides better resilience against the distortions introduced by outliers.
- Conversely, if your data is relatively clean and without significant outliers, ***MSE***'s faster convergence might offer an advantage.



Generalization

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

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

- A big part of machine learning focuses on the question, **how to do this minimization efficiently**.
- If you find a function ***h(·)*** with low loss on your data ***D***, how do you know whether it will still get examples right that are not in ***D***?
- **Bad Algorithm**: "memorizer" ***h(·)***

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

- For this ***h(·)*** , 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.

Main Challenges of Machine Learning

Underfitting the Training Data

- Underfitting occurs when your model is too simple to learn the underlying structure of the data.
- For example, a linear model of life satisfaction is prone to underfit; reality is just more complex than the model, so its predictions are bound to be inaccurate, even on the training examples. The main options to fix this problem are:
  - Selecting a more powerful model, with more parameters
  - Feeding better features to the learning algorithm (feature engineering)
  - Reducing the constraints on the model (e.g., reducing the regularization hyperparameter

Main Challenges of Machine Learning

- **Insufficient Quantity of Training Data:**
  - For a toddler to learn what an apple is, all it takes is for you to point to an apple and say “apple” (possibly repeating this procedure a few times).
  - Machine Learning takes a lot of data for most Machine Learning algorithms to work properly. Even for very simple problems you typically need thousands of examples, and for complex problems such as image or speech recognition you may need millions of examples

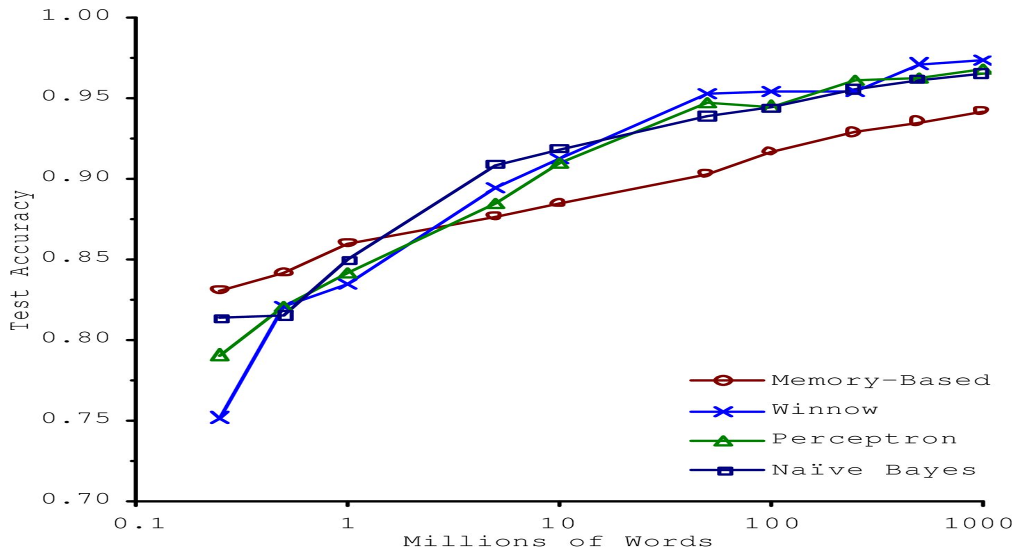
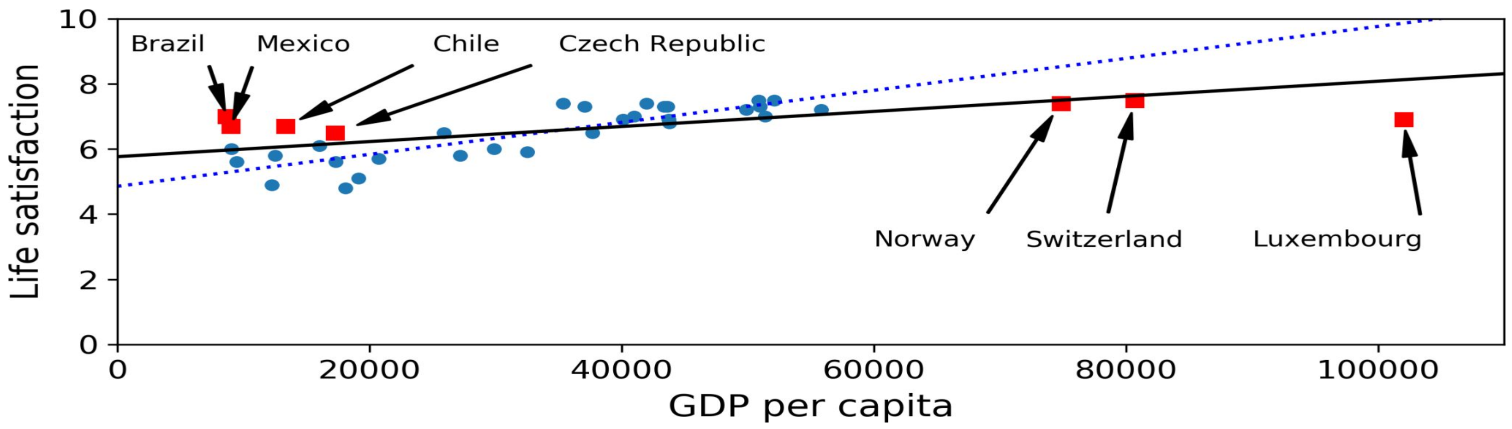


Figure reproduced with permission from Banko and Brill (2001), “Learning Curves for Confusion Set Disambiguation.”

Main Challenges of Machine Learning

Nonrepresentative Training Data:

- In order to generalize well, it is crucial that your training data be representative of the new cases you want to generalize.
- For example, if you train a linear model on data (when you add the missing countries), you get the solid line, while the old model is represented by the dotted line (a few countries were missing).



Main Challenges of Machine Learning

Poor-Quality Data:

- Obviously, if your training data is full of errors, outliers, and noise (e.g., due to poor quality measurements), it will make it harder for the system to detect the underlying patterns, so your system is less likely to perform well.
- It is often well worth the effort to spend time cleaning up your training data. The truth is, most data scientists spend a significant part of their time doing just that.
- For example:
  - If some instances are clearly outliers, it may help to simply discard them or try to fix the errors manually.
  - If some instances are missing a few features (e.g., 5% of your customers did not specify their age), you must decide whether you want to ignore this attribute altogether, ignore these instances, fill in the missing values (e.g., with the median age), or train one model with the feature and one model without it, and so on.

Main Challenges of Machine Learning

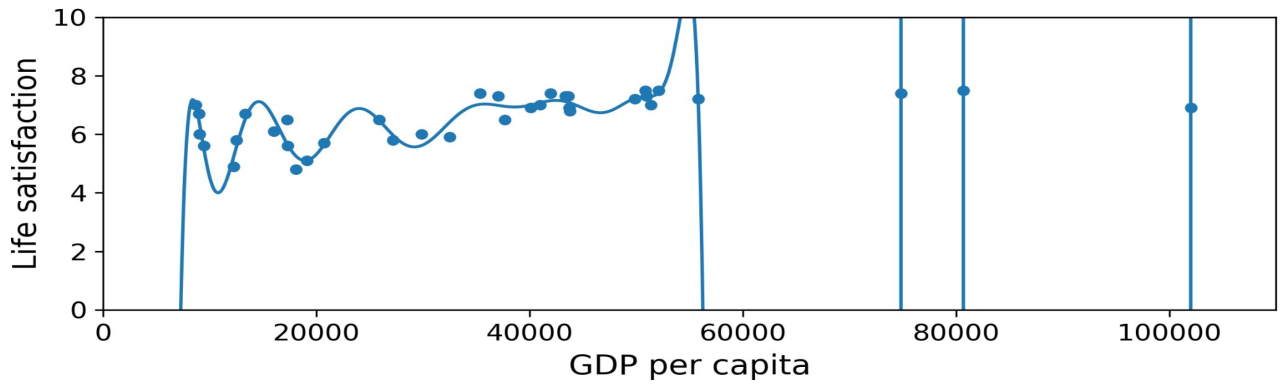
Irrelevant Features :

- Your system will only be capable of learning if the training data contains enough relevant features and not too many irrelevant ones. A critical part of the success of a Machine Learning project is coming up with a good set of features to train on. This process, called **feature engineering**, involves:
  - **Feature selection:** selecting the most useful features to train on among existing features.
  - **Feature extraction:** combining existing features to produce a more useful one**.**
  - **Creating new features by gathering new data.**

Main Challenges of Machine Learning

Overfitting the Training Data

- In Machine Learning this is called **overfitting**: it means that the model performs well on the training data, but it does not generalize well.



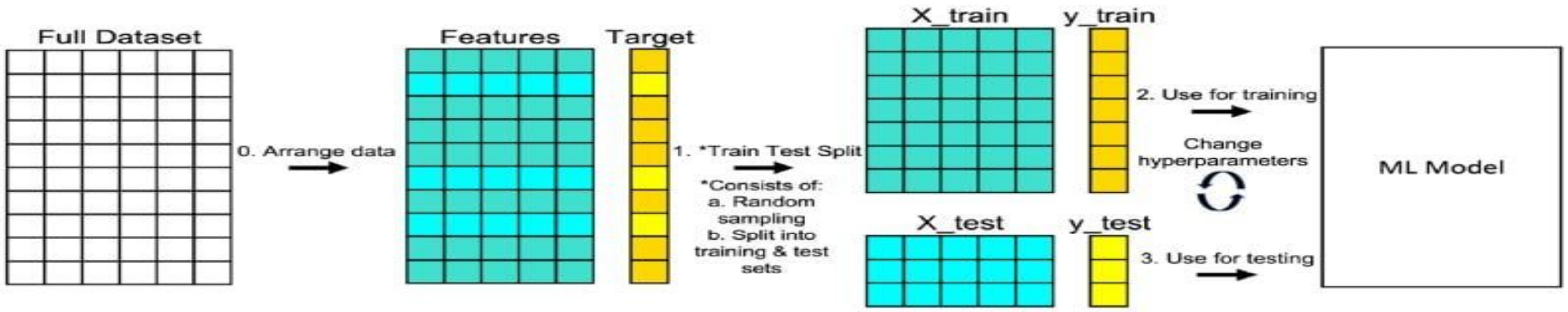
- Constraining a model to make it simpler and reduce the risk of overfitting is called **regularization**.
- The amount of regularization to apply during learning can be controlled by a **hyperparameter**. A **hyperparameter** is a parameter of a learning algorithm (not of the model).

Testing and Validating

- The only way to know how well a model will generalize to new cases is to actually try it out on new cases.
- One way to do that is to put your model in production and monitor how well it performs. This works well, but if your model is horribly bad, your users will complain—not the best idea.
- A better option is to split your data into two sets:
  - the **training set** and the **test set**.
  - As these names imply, you train your model using the training set, and you test it using the test set. The error rate on new cases is called the generalization error (or out-of-sample error), and by evaluating your model on the test set, you get an estimate of this error.
  - This value tells you how well your model will perform on instances it has never seen before.

Train / Test splits

- To resolve the overfitting issue, we usually split D into three subsets: $\mathbf{D}TR$ as the training data, $\mathbf{D}VA$, as the validation data, and $\mathbf{D}TE$, as the test data.
- Usually, they are split into a proportion of **80%** , **10%** , and **10%** . Then, we choose $h(\cdot)$ based on $\mathbf{D}TR$, and evaluate $h(\cdot)$ on $\mathbf{D}TE$.
- **Quiz: Why do we need DVA ?**
- $\mathbf{D}VA$ is used to check whether the $h(\cdot)$ obtained from $\mathbf{D}TR$ suffers from the overfitting issue.
- $h(\cdot)$ will need to be validated on $\mathbf{D}VA$, if the loss is too large, $h(\cdot)$ will get revised based on $\mathbf{D}TR$, and validated again on $\mathbf{D}VA$.
- This process will keep going back and forth until it gives low loss on $\mathbf{D}VA$. Here's a trade-off between the sizes of $\mathbf{D}TR$ and $\mathbf{D}VA$: the training results will be better for a larger $\mathbf{D}TR$, but the validation will be more reliable (less noisy) if $\mathbf{D}VA$ is larger.



How to Split the Data?

- It is common to use **80% of the data for training** and hold out **20% for testing**.
- However, this depends on the size of the dataset: if it contains 10 million instances, then holding out 1% means your test set will contain 100,000 instances: that’s probably more than enough to get a good estimate of the generalization error.
- The **error rate on new cases is called the generalization error** (or out-of-sample error), and by evaluating your model on the test set, you get an estimate of this error.
- This value tells you how well your model will perform on instances it has never seen before.
- If the **training error is low** (i.e., your model makes few mistakes on the training set) but **the generalization error is high**, it means that your model is **overfitting the training data**.