Module A: Machine Learning Basics

Abdulmalek Al-Gahmi, PhD

August 24, 2023

Introduction

What is machine learning?

Typically:

- There is a pattern;
- We can see it in the available data; but
- We cannot pin it down mathematically or programmatically.

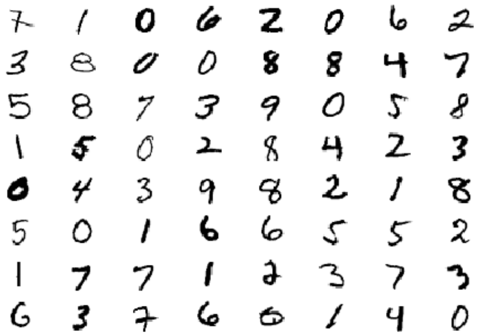
What is machine learning?

- “Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed.”
  – Arthur Samuel, 1959
- “A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*. ”
  – Tom Mitchell, 1997

What is learning?

"'A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*. "
– Tom Mitchell, 1997

An example: Recognizing handwritten digits



What is T? What is P? What is E?

What is learning? Another example

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. ”

– Tom Mitchell, 1997

Another Example: Credit approval

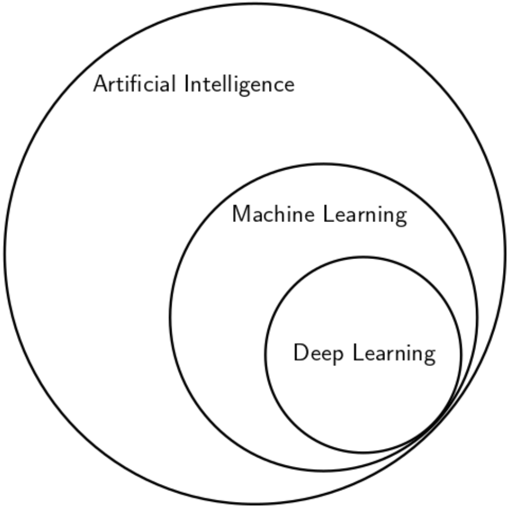
Input (Application information):

- Age
- Gender
- Employed?
- Salary
- Years in current residence
- Years in current job
- Debt

Output: Whether application is approved or not?

What is T? What is P? What is E?

Machine learning in perspective

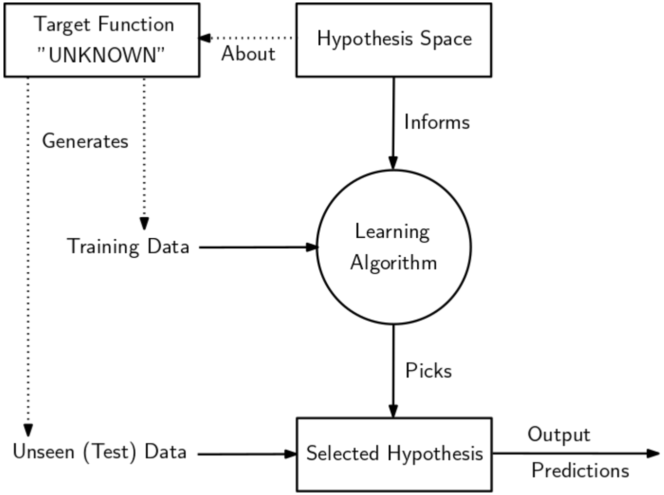


The learning problem

- We have training data generated by an unknown target function $f(x)$.
- The goal is to find a function $\hat{h}(x)$ (also called a **hypothesis**) that best approximates $f(x)$.
- Each hypothesis $h(x)$ comes from a hypothesis space $\mathcal{H}$.
- How do we choose the appropriate hypothesis?
  - Prior knowledge
  - Exploratory data analysis
  - Data visualization
  - Evaluating multiple $h(x)$ functions to see which one fits best
- **The true measure of fitness of a given hypothesis $h(x)$ is not how well it does on the training data but rather how it handles data it has not seen before.**

The learning problem

- Target function
- Data
- Hypothesis space
- Learning algorithm
- Selected hypothesis
- Cost function (error)
- Input (***x***)
- Output (***y***)



Anatomy of a learning algorithm

- An optimization criterion involving an objective or loss function L and trainable parameters $\boldsymbol{\theta}$.
- An optimization algorithm leveraging training data to find a solution to the optimization criterion.
- However, learning is different from pure optimization.
  - The goal of ML is to reduce the generalization error.
  - Fitting a ML model/algorithm to the training data is called **Empirical Risk Minimization** and expressed as:

$$\boldsymbol{\theta} = \underset{\theta}{\arg\min} \frac{1}{N} \sum_{i=1}^{N} L(y^{(i)}, h(x^{(i)}; \theta))$$

Types of learning

Depending on the feedback that can accompany the input data, there are three main kinds of learning:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Depending on whether models can learn incrementally on the fly:

- Batch learning
- Online learning

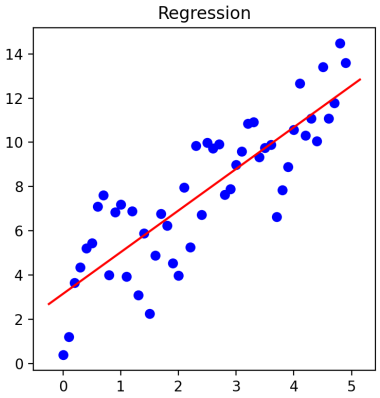
Depending on whether whether new examples are compared against existing examples:

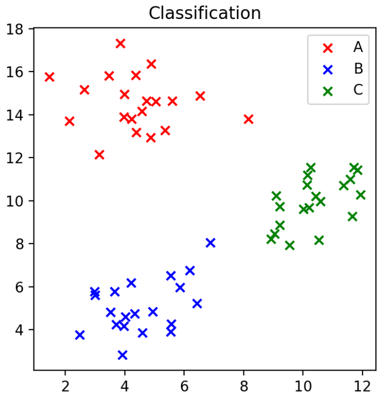
- Instance-based learing
- Model-based learning

Supervised learning

The input data is paired with some output (aka “labels”) that acts as a “teacher” guiding the learning algorithm.

- Classification: The output data is discrete and finite.
- Regression: The output is numeric and continuous.



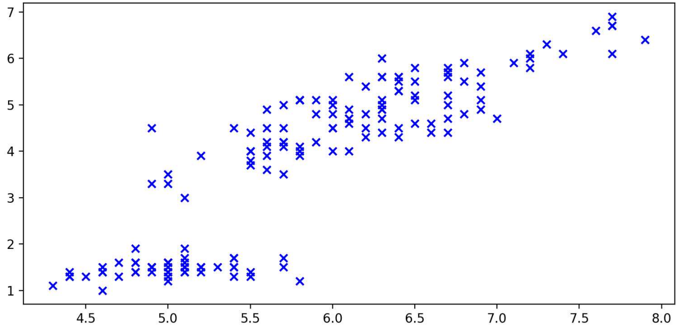


Different kinds of supervised-learning problems

- Regression
- Binary classification
- Multi-class classification
- Multi-label classification

Unsupervised learning

- No labels (or feedback) to accompany the input data
- Example unsupervised learning tasks include:
  - Clustering
  - Dimensionality reduction
  - Anomaly detection



And it’s not always a clear division

- Semi-supervised learning
- Self-supervised learning

Reinforcement learning

“the AI revolution will not be supervised”
–Yann LeCun and Alyosha Efros

No correct answers, but some feedback on how the agent is doing (good or bad) as it navigates the environment is provided.

- Learning from a series of reinforcements (rewards and punishments)
- Deciding which actions were mostly responsible for the rewards
- Altering actions to maximize the rewards

Example ML applications

- Image classification
- Detecting tumors in brain scans
- Flagging offensive comments in discussion forums
- Forecasting stock prices
- Market segmentation
- Recommending products
- Playing games

Main ML challenges

- Insufficient quantity of training data
- Nonrepresentative training data
- Poor-quality data
- Irrelevent features
- Underfitting and overfitting the training data

A bit of history

- 1943: McCulloch and Pitts neuron
- 1949: Hebbs' update rule for modifying the connection strengths between neurons
- 1957: The Perceptron algorithm
- 1959: Arthur Samuel wrote a checkers program that could defeat him
- 1969: Minsky and Papert's book "Perceptrons"
- 1980s:
  - PAC learning formalized by Leslie Valiant
  - Reinvention of back-propagation learning algorithm
  - Bayesian networks

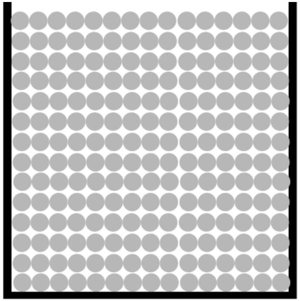
A bit of history

- 1990s:
  - Markov Chain Monte Carlo (MCMC)
  - Support Vector Machines(SVM’s)
  - Kernels
  - Boosting
  - Convolutional networks
- 2000s:
  - Wide adoption of ML in AI fields such as computer vision and NLP.
  - Big data
- 2010s: Deep learning:
  - 2012: ImageNet competition where a deep learning system shows a dramatic improvement over previous systems
  - 2016: AlphaGo defeated the human Go champion
  - 2017: “Attention Is All You Need”

More about learning

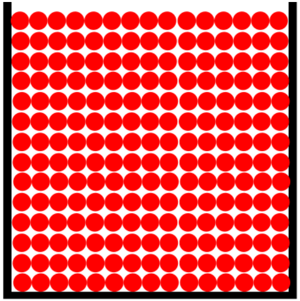
Is learning feasible?

Say that we have a bin with red and blue marbles.



We are trying to learn μ where: μ = the probability of red marbles.

Here is one hypothesis: $\mu = 1$ (all marbles are red).



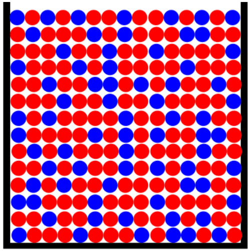
Obviously this is a bad hypothesis. How much (or long) does it take to rule it out?

Is learning feasible?

It takes a single sample with at least a single blue marble such as this

to rule the hypothesis that $\mu = 1$ out.

Now consider the hypothesis that: $\mu = .66$.



Would a sample such as:

be enough to rule this hypothesis out?

PAC learning: Probably Approximately Correct

- How many examples are needed for learning?
  - If a hypotheses *h* is seriously wrong, it will be ”found out” with high probability after a few examples.
  - Any hypothesis that is consistent with a large set of training examples is unlikely to be seriously wrong. In other words, it must be **Probably Approximately Correct** (PAC).
- PAC learning algorithm: Any learning algorithm that returns a hypothesis *h* that is probably approximately correct.
- It turns out that for PAC-learning we have a bound for *N* (the number of training examples)

$$N \geq \frac{1}{\epsilon}\left(log\frac{1}{\delta} + log|\mathcal{H}|\right)$$

  That is with a probability of $1-\delta$ after seeing at least *N* examples, the learning algorithm will return a hypothesis with an error that is at most ϵ.

Working with data

In machine learning, data is organized into tables called data sets. A typical data set consists of n examples (or rows) each of which contains m features (or columns) or dimensions. In supervised learning, a data set can mathematically be represented as a $n \times m$ matrix $\boldsymbol{X}$ where:

$$\boldsymbol{D} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_m^{(1)} & y^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_m^{(2)} & y^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & \cdots & x_m^{(3)} & y^{(3)} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \cdots & x_m^{(n)} & y^{(n)} \end{bmatrix}$$

Each example or row constist of two parts: An input data vector

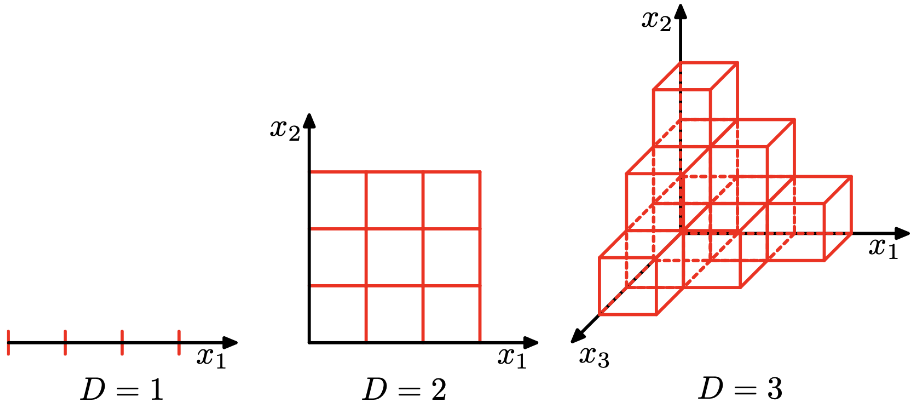
$$\boldsymbol{x} = [x_1, x_2, x_3, \cdots, x_m]$$

and a target value (or label) y.

The curse of dimensionality

- The term was coined by Richard Bellman (1961) while studying dynamic programming.
- As the number of dimensions increases, the volume of the space increases so fast that the existing data becomes sparse.
- The amount of data needed grows exponentially with number of dimensions. In other words, as the number of input features increases, more data will be needed to allow the algorithm to generalize well.

The curse of dimensionality [1]



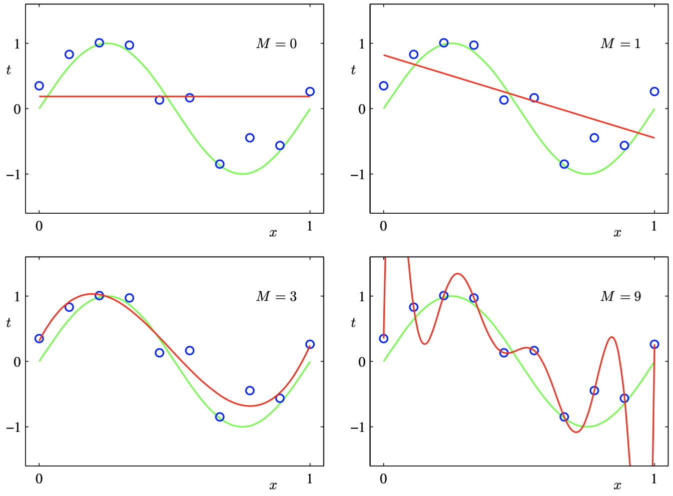
Two competing principles

- Optimization vs generalization
- Two kinds of errors to consider:
  - Training error E_{train}: results from fitting the model to the training data.
  - Testing error E_{val}: results from evaluating the model to the validation data.
- The two questions of learning:
  - Can we make sure that E_{val} is close enough to E_{train}?
  - Can we make E_{train} small enough?
- Any learning model (or algorithm), should be:
  - expressive enough so that E_{train} is low; and
  - reliable enough that a low E_{train} implies a low E_{val}. In other words, it generalizes well.

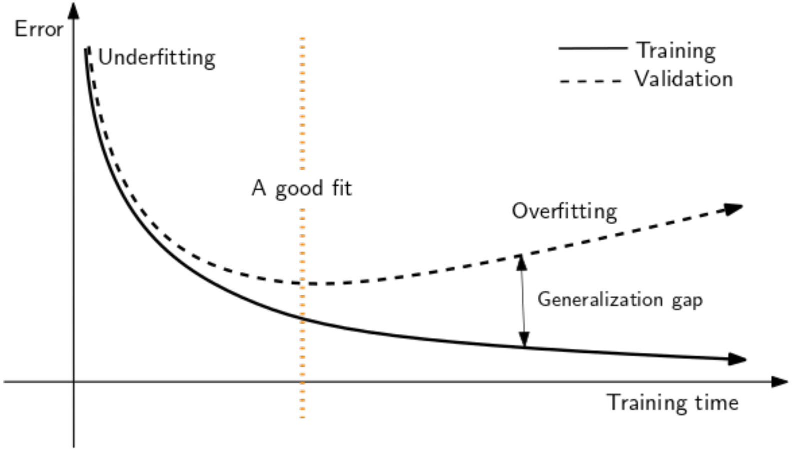
Overfitting vs underfitting

- **Underfitting**: Happens when the learning model or algorithms is not expressive (not flexible or sophisticated) enough to capture the underlying trend(s) of the data.
- **Overfitting**: Happens when the model starts to capture not only the trends of the data, but also the noise as well. In other words, the model is too close to the training data, thus causing it to perform poorly on unseen data.
- Often there is a **bias-variance-tradeoff**.

Overfitting vs underfitting: An example [1]



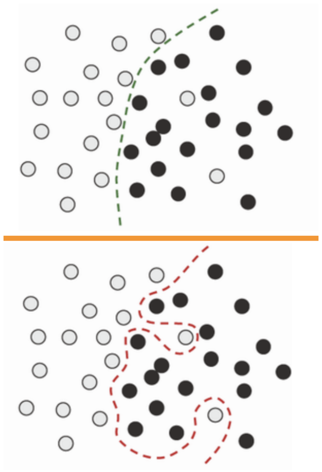
Overfitting vs underfitting



Reasons for overfitting

Overfitting is a central problem in machine learning. Many reasons can cause it:

- Noisy training data
- Ambiguous features
- Rare features
- Spurious correlations



The bias-variance tradeoff

- **Bias** refers to the inability of a model to capture a pattern in the data. It results from the model making erroneous assumptions about the data, and represents restrictions imposed by the hypothesis space $\mathcal{H}$.
- **Variance** refers to the model being overly sensitive to fluctuations in training data. High-variance means the model is not only fitting the data but also the random noise that it inherently has.
- Often there is a **bias-variance-tradeoff**, in which a choice has to be made between a more complex, low-bias, high-variance model that fits the training data well and a simpler high-bias, low-variance model that generalizes better.
- Typically:
  - low-bias, high-variance model $\longrightarrow$ Overfitting
  - high-bias, low-variance model $\longrightarrow$ Underfitting

The bias-variance decomposition

Let f refer to the target deterministic function $f(x)$, h to the hypotheses $h(x)$ being evaluated. The expected test squared error is $\mathsf{E}[(y-h)^2]$, where y is the target value (supervised learning). We will assume that $y = f + \varepsilon$, where ε is an error with $\mathsf{E}[\varepsilon] = 0$ and $\mathsf{Var}(\varepsilon) = \mathsf{E}[\varepsilon^2] - (\mathsf{E}[\varepsilon])^2 = \sigma^2$.

$$\begin{aligned}
\mathsf{E}[(y-h)^2] &= \mathsf{E}[(f+\varepsilon-h)^2] \\
&= \mathsf{E}[(f+\varepsilon-h+\mathsf{E}[h]-\mathsf{E}[h])^2] \\
&= \mathsf{E}[(B+V+\varepsilon)^2]
\end{aligned}$$

where $B = f - \mathsf{E}[h]$ and $V = \mathsf{E}[h] - h$.

$$\begin{aligned}
\mathsf{E}[(y-h)^2] &= \mathsf{E}[B^2 + 2BV + 2\varepsilon B + V^2 + 2\varepsilon V + \varepsilon^2] \\
&= \mathsf{E}[B^2] + \mathsf{E}[V^2] + \mathsf{E}[\varepsilon^2] + \mathsf{E}[2BV + 2\varepsilon B + 2\varepsilon V] \\
&= B^2 + \mathsf{E}[V^2] + \mathsf{E}[\varepsilon^2] + \mathsf{E}[2BV + 2\varepsilon B + 2\varepsilon V] \\
&= (f - \mathsf{E}[h])^2 + \mathsf{E}[(\mathsf{E}[h]-h)^2] + \mathsf{E}[\varepsilon^2] \\
&= \mathsf{Bias}[h]^2 + \mathsf{Var}[h] + \mathsf{Var}[\varepsilon]
\end{aligned}$$

Exercise: Show that $\mathsf{E}[2BV + 2\varepsilon B + 2\varepsilon V]$ is zero.

Occam’s razor (The principle of parsimony)

- “Everything should be made as simple as possible, but not simpler.”
  - Anonymous
- “plurality should not be posited without necessity.”
  - William of Ockham (14th-century)
- “The supreme goal of all theory is to make the irreducible basic elements as simple and as few as possible without having to surrender the adequate representation of a single data of experience”
  - Albert Einstein (1933)

Simplicity is however difficult to define.

- Number of parameters is not a good measure.
- Perhaps we should think of appropriateness instead!

The No-Free-Lunch theorem

- “All models are wrong, but some models are useful.”
  – George Box
- **The no-free-lunch theorem**: While there might be a best model for a particular problem, there is no universally best model.
- Consequences of this theorem:
  - There is no superior black-box optimization strategy or model.
  - If a model performs better than random in some problems, it must perform worse than random in others.

Model Selection

The I.I.D. assumption

In machine learning we operate under the assumption that future (unseen) data examples are like past data examples. This is called the **stationarity** assumption. More specifically, we assume that:

- All examples have the same prior probability.
- All examples are independent from one another.

A dataset that satisfies these two conditions is considered **independent and identically distributed** or **i.i.d.** for short.

Selecting a learning algorithm

- Explainability
- In-memory vs out-of-memory
- Number of features and examples
- Categorical vs numerical features
- Nonlinearity of the data
- Training speed
- Prediction speed

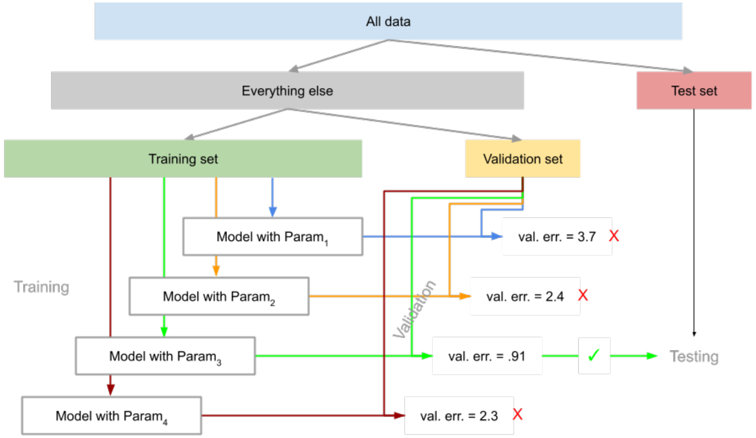
Assessing model performance

- The holdout method
- K-fold cross-validation

The holdout method: Three data sets

- Remember that the true measure of fitness of a model is not how well it does on the training dataset but rather how it handles data it has not seen before. That requires having:
  - **Training set**: for training the model.
  - **Test set**: for testing how it generalizes after it has been trained.
- For models with hyperparameters, a third set called **validation set** is held back from the **Training set** and is used to find the best hyperparameter values. This is called **hyperparameter tuning**.
- No general rule on where split points should be; depends on the dataset and the problem being solved. Generally we need to train our models with as much data as possible. Common split points are:
  - 50/25/25
  - 60/20/10
  - 80/10/10
- Notice that the test set is used only at the very end. It measures how well the model generalizes to (performs on) unseen data.
- It's critical that each one of these sets is representative of the data.

Three sets: Training, validation, and test



Question: What if we don’t have enough data for a separate validation set?

κ-fold cross-validation

- Good models require using as much data for training as possible.
- When the supply of data is limited, we use **K-fold cross-validation**.

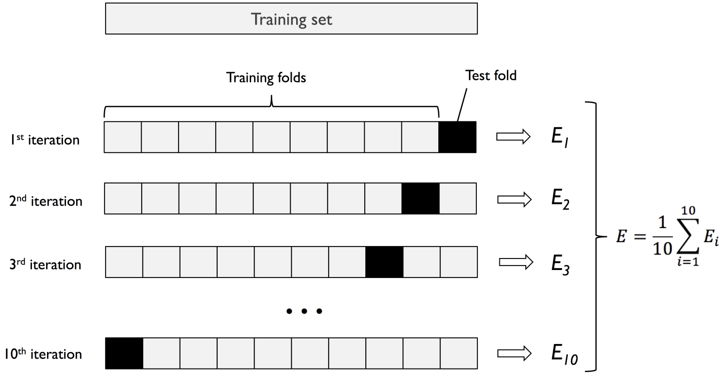


Figure: Source: [4]

K-fold cross-validation

```
function Cross-Validation($model, X\_train, y\_train, K = 10$)
    Divide $X\_train$ and $y\_train$ into $K$ equal-size folds: $X\_folds$ and $y\_folds$ respectively.
    $err \leftarrow 0$
    for $i \leftarrow 1$ to $K$ do
        $X\_cv\_valid, y\_cv\_valid \leftarrow X\_folds[i], y\_folds[i]$
        $X\_cv\_train, y\_cv\_train \leftarrow X\_train - X\_cv_valid, y\_train - y\_cv\_valid$
        $model.fit(X\_cv\_train, y\_cv\_train)$
        $err \leftarrow err + \text{Error}(y\_cv\_valid, model.predict(X\_cv\_valid))$
    end for
    return $err/K$
end function
```

Other forms of K-fold cross-validation

- When data is scarce, one can use single-example folds; this is called **leave-one-out cross-validation** (LOOCV).
- To make sure that every fold is representive of the data, the **Stratified K-fold cross-validation** is used. Here, the folds are made by preserving the percentage of samples for each class.

Hyperparameter tuning: The case of a single parameter

- Selecting the best model from a set of candidate models with differnt hyperparameter values.
- Given the original data X, y, a learning algorithm *Learner* to tune, and a list of possiple hyperparameter values *hparam_values*:

```
$errors \leftarrow$ empty array indexed by the values of the hyperparameter
$X\_train, y\_train, X\_test, y\_test \leftarrow$ split(X, y)
**for** $hparam \leftarrow hparam\_values$ **do**
    $model \leftarrow Learner(hparm, \ldots)$
    $errors[hparam] \leftarrow$ Cross-Validation($model, X\_train, y\_train$)
**end for**
$best\_hparam \leftarrow$ the index of minimum(errors)
$best\_model \leftarrow Learner(best\_hparam, \ldots)$
```

Hyperparameter tuning: The case of more than one parameter

- Grid search: All possible combinations of hyperparameter values will be evaluated. Depending on how big the hyperparameter search space is, it can be very slow and and computationally expensive.
- Randomized search: Only a fixed number of combinations is evaluated selecting a random value for each hyperparameter at every iteration.

Measuring Performance

Measuring performance

- Regression
  - Sum of squared errors (SSE): $\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$
  - Mean squared error (MSE): $\frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$
  - Mean absolute error (MAE): $\frac{1}{N}\sum_{i=1}^{N}\left|y^{(i)} - \hat{y}^{(i)}\right|$
  - R^2 (Coefficient of determination): $1 - \frac{RSS}{TSS} = 1 - \frac{\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^{N}(y^{(i)} - \bar{y})^2}$
- Classification
  - Confusion matrix
  - Accuracy
  - Cost-sensitive accuracy
  - Recall, precision, and F_1
  - Area under the ROC curve (AUC)

Confusion matrix

A squared matrix with all the predicted labels on top (as columns) and all actual target labels to the left (as rows).

		Predicted		
		C_1	C_2	C_3
Actual	C_1	6	2	2
	C_2	1	9	0
	C_3	2	1	8

- A cell $A_{c,d}$ contains the number of examples whose actual label is c but were classified by the algorithm as d.
- The accuracy of the algorithm can be calculated as:

$$Accuracy = \frac{sum\ of\ all\ leading\ diagonal\ values}{sum\ of\ all\ values}$$

Question: What is the accuracy of the above confusion matrix?

Confusion matrix (continued)

For a two-label classification problem, the confusion matrix looks like this:

		Predicted	
		Positive	*Negative*
Actual	*Positive*	#TP	#FN
	Negative	#FP	#TN

where:

- $TP = True\ Positive$
- $FN = False\ Negative$ — a miss, Type II error
- $FP = False\ Positive$ — false alarm, Type I error
- $TN = True\ Negative$

In this case the accuracy is:

$$Accuracy = \frac{\#TP + \#TN}{\#TP + \#FN + \#FP + \#TN}$$

Sensitivity (recall) and specificity

Given:

		Predicted	
		Positive	*Negative*
Actual	*Positive*	#TP	#FN
	Negative	#FP	#TN

Sensitivity (also known as **the true positive rate**) is the percentage of actual positives that are correctly classified:

$$\textit{Sensitivity} = \textit{True Positive Rate} = \frac{\#TP}{\#P} = \frac{\#TP}{\#TP + \#FN}$$

Specificity (also known as **the true negative rate**) is the percentage of negatives that are correctly classified:

$$\textit{Specificity} = \textit{True Negative Rate} = \frac{\#TN}{\#N} = \frac{\#TN}{\#TN + \#FP}$$

Question: What if we have more than two classes?

Precision and recall

Given:

		Predicted	
		Positive	*Negative*
Actual	*Positive*	#TP	#FN
	Negative	#FP	#TN

Precision: Of the examples classified as positives, how many were correctly classified?

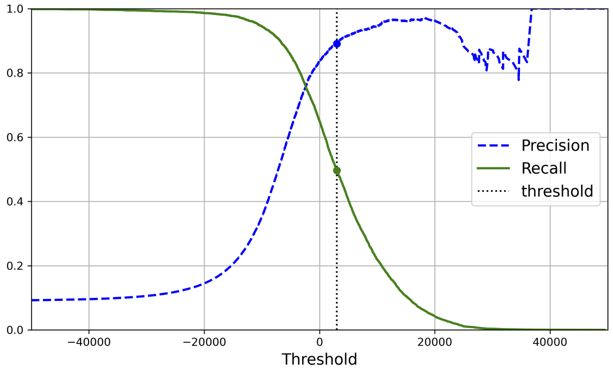
$$Precision = \frac{\#TP}{\#TP + \#FP}$$

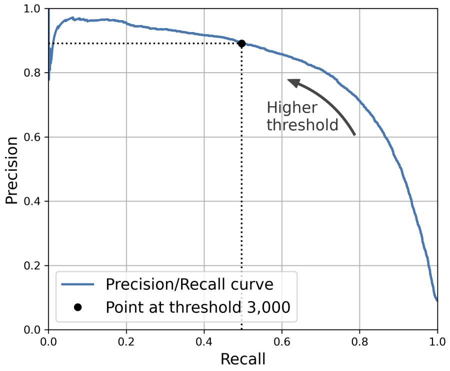
Recall: Of the actual possitive examples, how many were correctly classified?

$$Recall = \frac{\#TP}{\#TP + \#FN}$$

Question: Is recall the same as sensitivity?

The recall/precision trade-off





The F_1 score

Given:

		Predicted	
		Positive	*Negative*
Actual	*Positive*	#TP	#FN
	Negative	#FP	#TN

The F_1 score is defined as:

$$F_1 = \frac{2}{\frac{1}{\mathit{precision}} + \frac{1}{\mathit{recall}}} = 2 \times \frac{\mathit{precision} \times \mathit{recall}}{\mathit{precision} + \mathit{recall}} = \frac{2\#TP}{2\#TP + \#FP + \#FN}$$

Note that the $F1$ score favors classifiers that have similar precision and recall, which is not always what you want.

The micro and macro F_1 scores

For multiclass classification problems, there are multiple F_1 scores. We define two quantities that bring these scores together:

- **The micro F_1 score** which is defined as:

$$\textbf{micro F}_\mathbf{1} = \frac{2\#TP}{2\#TP + \#FP + \#FN}$$

  where $\#TP$, $\#FP$, and $\#FN$ are the total numbers of $\#TP$, $\#FP$ and $\#FN$ accross all classes instead of individually for each class.

- **The macro F_1 score** is the unweighted mean of the F_1 scores calculated per class. That is:

$$\textbf{macro F}_\mathbf{1} = \frac{sum(F_1 \text{ scores})}{\# \text{ classes}}$$

Question: Which is better for **unbalanced datasets**?

An example

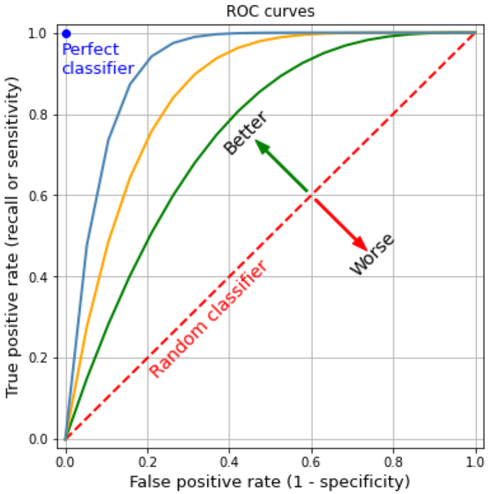
Calculate the following metrics, given the below confusion matrix that results from applying logistic regression to a fictional cancer-diagnosis dataset.

		Predicted	
		With cancer	*No cancer*
Actual	*With cancer*	239	31
	No cancer	43	287

- $\mathit{Accuracy} = \frac{\#\mathit{TP}+\#\mathit{TN}}{\#\mathit{TP}+\#\mathit{FN}+\#\mathit{FP}+\#\mathit{TN}} =$
- $\mathit{Sensitivity} = \frac{\#\mathit{TP}}{\#\mathit{TP}+\#\mathit{FN}} =$
- $\mathit{Specificity} = \frac{\#\mathit{TN}}{\#\mathit{TN}+\#\mathit{FP}} =$
- $\mathit{Precision} = \frac{\#\mathit{TP}}{\#\mathit{TP}+\#\mathit{FP}} =$
- $F_1 = \frac{2\#\mathit{TP}}{2\#\mathit{TP}+\#\mathit{FP}+\#\mathit{FN}} =$

The Receiver Operator Characteristic (ROC) Curve

- It's a plot of the false positive rate ($1 - \textit{specificity}$) on the x-axis against the true positive rate (*sensitivity*) on the y-axis.
- It is useful in comparing classifiers.
- A single run of a classifier produces a single point on the ROC plot:
  - Point (0, 1) — A perfect classifier
  - Point (1, 0) — An anti-classifier that got everything wrong



Feature engineering techniques

Feature engineering

- One-hot encoding
- Binning (Bucketing)
- Normalization

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- Standardization

$$x_{new} = \frac{x - \bar{x}}{\sigma}$$

- Log-transformation

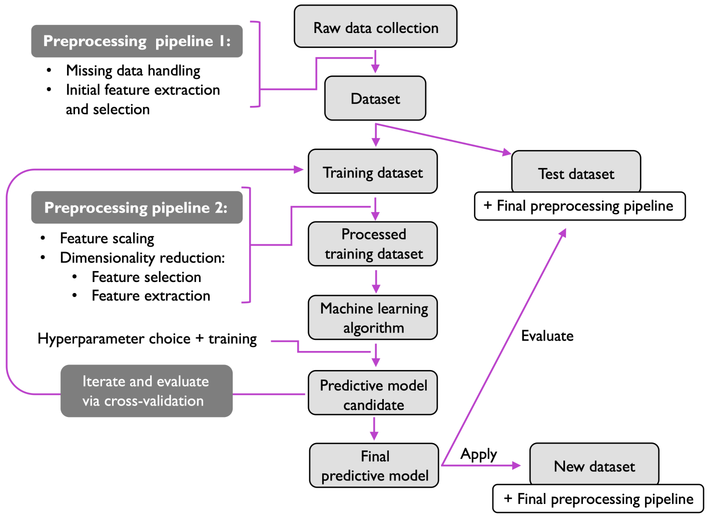
$$x_{new} = \log x$$

As we preprocess data, we must watch out for **data leaks**. What would be an example of a data leak?

Handling missing data

- Remove them
- Use a learning algorithm that can deal with them.
- Use a data imputation technique

Pipeline of an end-to-end machine learning project [4]


Preprocessing pipeline 1:
Missing data handling
Initial feature extraction and selection
Raw data collection
Dataset
Training dataset
Test dataset
+ Final preprocessing pipeline
Preprocessing pipeline 2:
Feature scaling
Dimensionality reduction:
Feature selection
Feature extraction
Processed training dataset
Machine learning algorithm
Hyperparameter choice + training
Evaluate
Iterate and evaluate via cross-validation
Predictive model candidate
Final predictive model
Apply
New dataset
+ Final preprocessing pipeline

Non-parametric learners

Supervided learning: datasets, inputs, and labels

Given a training dataset:

$$\boldsymbol{D} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_m^{(1)} & y^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_m^{(2)} & y^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & \cdots & x_m^{(3)} & y^{(3)} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \cdots & x_m^{(n)} & y^{(n)} \end{bmatrix}$$

We can break it into an input matrix $\boldsymbol{X}$ and a labels vector $\boldsymbol{y}$:

$$\boldsymbol{X} = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_m^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_m^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & \cdots & x_m^{(3)} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \cdots & x_m^{(n)} \end{bmatrix} \text{ and } \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ y^{(3)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Nearest Neighbor (NN)

- Given an unseen example $\boldsymbol{x} = [x_1 \ x_2 \ x_3 \ \cdots \ x_m]$, we need to calculate the corresponding y given the training dataset $\boldsymbol{D}$.
- One very simple algorithm to do so:

```
function Nearest-Neighbor($\boldsymbol{D}$, $\boldsymbol{x}$)
    $\boldsymbol{X}, \boldsymbol{y}$ = Separate($\boldsymbol{D}$)
    Find the example $(\boldsymbol{x}^*, y^*)$ in $\boldsymbol{D}$ that is nearest to $\boldsymbol{x}$. That is:
        $$(\boldsymbol{x}^*, y^*) = \underset{\boldsymbol{x}^{(i)} \in \boldsymbol{X}}{\text{argmin}} \ distance(\boldsymbol{x}^{(i)}, \boldsymbol{x})$$
    return $y^*$
end function
```

- We can formalize "nearest" in terms of the Euclidean distance:

$$distance(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) = \|\boldsymbol{x}^{(a)} - \boldsymbol{x}^{(b)}\|_2 = \sqrt{\sum_{i=1}^{m} (x_i^{(a)} - x_i^{(b)})^2}$$

- **Question**: Can we do without calculating the square root? Why?

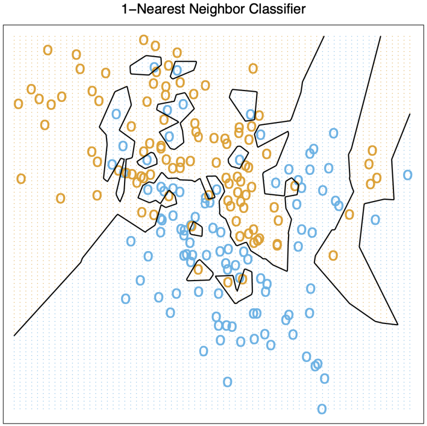
K-Nearest Neighbors (KNN)

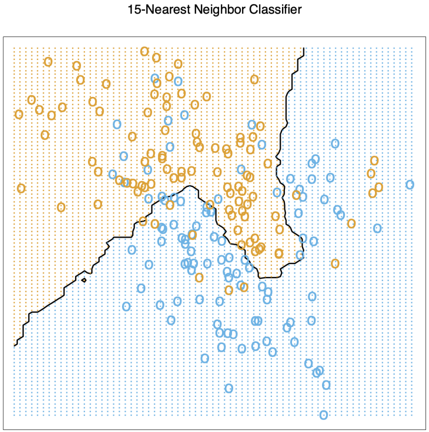
- Given the training dataset $\boldsymbol{D}$, an unseen example $\boldsymbol{x}$, and a hyperparameter K:

```
function K-Nearest-Neighbors($\boldsymbol{D}$, $\boldsymbol{x}$, $K$)
    $\boldsymbol{X}, \boldsymbol{y}$ = Separate($\boldsymbol{D}$)
    Find the $K$ examples $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}$ in $\boldsymbol{D}$ that are nearest to $\boldsymbol{x}$.
    return the majority class of the nearest $K$ neighbors $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}$.
end function
```

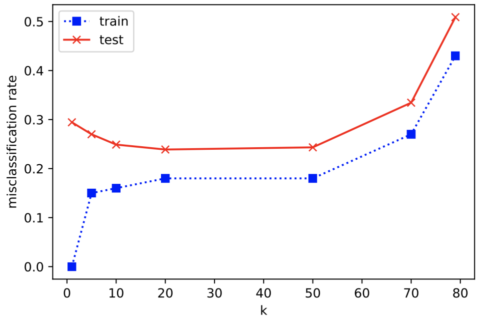
- Majority vote vs. plurality vote?
- How do we choose K?
  - K is a hyperparameter: a property of the leaner that is independent from the data.
  - Use cross-validation to select the best value for K. Typically $k < \sqrt{N}$ where N is the size of the training dataset.
  - Small K — Likely to overfit; unstable; captures fine-grained patterns.
  - Large k — Likely to underfit; stable; fails to capture all patterns.

KNN decision boundries: An example [2]





KNN training and test errors [3]



KNN: Different measures of distance

- For Typically we use The **Minkowski distance**:

$$distance_p(\boldsymbol{x}^{(a)}, \boldsymbol{x}^{(b)}) = \|\boldsymbol{x}^{(a)} - \boldsymbol{x}^{(b)}\|_p = \sqrt[p]{\sum_{i=1}^{m} |x_i^{(a)} - x_i^{(b)}|^p}$$

  When $p = 2$ this is the **Euclidean distance**, and when $p = 1$ this is the **Manhattan distance**.

- Use the Euclidean distance for measuring similar features such as width, length, and depth. Use the Manhattan distance for dissimilar features such as age, weight, and height
- For boolean values, use the **Hamming distance**: the number of features on which the two examples are different.
- A more complex distance metric is the **Mahalanobis distance** which takes into account the covariance between features.

Bayes classification

- Given an unseen example $\boldsymbol{x}$, we want to predict its corresponding y from a set of possible classes $y_1, y_2, \ldots$. We can do so by picking the y such that:

$$y = \operatorname*{argmax}_i \; P(y_i|\boldsymbol{x})$$

- Combining $P(y_i, \boldsymbol{x}) = P(y_i)P(\boldsymbol{x}|y_i)$ and $P(y_i, \boldsymbol{x}) = P(\boldsymbol{x})P(y_i|\boldsymbol{x})$ gives us the Bayes rule:

$$P(y_i|\boldsymbol{x}) = \frac{P(y_i)P(\boldsymbol{x}|y_i)}{P(\boldsymbol{x})}$$

- The denominator is a normalization factor making sure that

$$\sum_i P(y_i|\boldsymbol{x}) = 1$$

- That is:

$$P(y_i|\boldsymbol{x}) = \alpha \, P(y_i)P(\boldsymbol{x}|y_i)$$

Naive Bayes classifier

- Given that: $P(y_i|\boldsymbol{x}) = \alpha\, P(y_i)P(\boldsymbol{x}|y_i)$
- We can estimate $P(y_i)$ from the data.
- For $P(\boldsymbol{x}|y_i)$, we have:

$$P(\boldsymbol{x}|y_i) = P(x_1, x_2, x_3, \cdots, x_m|y_i)$$

  How do we calculate this?

- Naive Bayes makes the strong "naive" assumption that the features $x_1, x_2, x_3, \cdots, x_m$ are independent from one another.

$$P(\boldsymbol{x}|y_i) = P(x_1, x_2, x_3, \cdots, x_m|y_i) = P(x_1|y_i)P(x_2|y_i)P(x_3|y_i)\cdots P(x_m|y_i)$$

- The conditional probability $P(x_j|y_i)$ is sometimes called the likelihood of x_j given the class y_i. If feature x_j is discrete, we can estimate it from the data.
- What should be done if x_j is continuous?

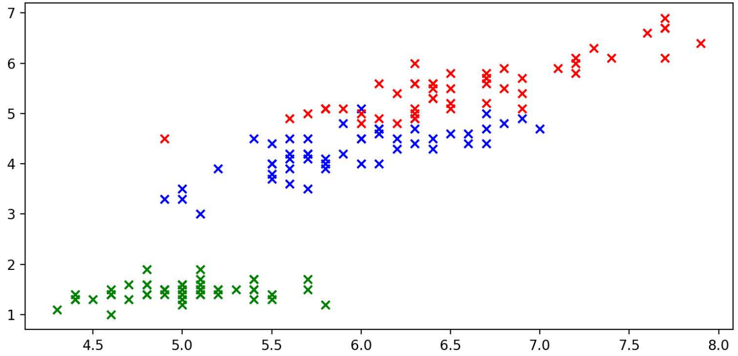
Naive Bayes classifier: An example (from the textbook)

Deadline?	Party?	Lazy?	Activity
Urgent	Yes	Yes	Party
Urgent	No	Yes	Study
Near	Yes	Yes	Party
None	Yes	No	Party
None	No	Yes	Pub
None	Yes	No	Party
Near	No	No	Study
Near	No	Yes	TV
Near	Yes	Yes	Party
Urgent	No	No	Study

What activity would a naive Bayes classifier predict for $(Near, No, Yes)$ given the above data?

Gaussian naive Bayes classifier

- Use the Gaussian (normal) distribution $N(\mu, \sigma)$ to estimate the conditional probability $P(x_j|y_i)$.
- Both parameters μ and σ are estimated using the training dataset.
- The prior probabilities $P(y_i)$ are also estimated using the training dataset.



Naive Bayes classifiers in practice

- Naive Bayes classifiers surprisingly work well in practice, despite the strong “naive” assumption they make about features being independent from one another.
- They are known to be better classifiers and poor predictors.
- Commonly used in text classification/ spam filtering/ sentiment analysis problems.

Questions?

References I

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. isbn: 0387310738.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning, 2nd edition*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2009.

[3] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. url: `probml.ai`.

[4] Sebastian Raschka, Yuxi Liu, and Vahid Mirjalili. *Machine Learning with PyTorch and Scikit-Learn Learning with Python*. Packt, 2022. isbn: 9781801819312.