"Sapienza" University of Rome
Faculty of Information Engineering,
Informatics and Statistics
Department of Computer Science

Machine Learning

Lecture notes integrated with the book TODO

Author
Alessio Bandiera

October 8, 2024

Contents

Information and Contacts

Personal notes and summaries collected as part of the *Machine Learning* course offered by the degree in Computer Science of the University of Rome "La Sapienza".

Further information and notes can be found at the following link:
https://github.com/aflaag-notes. Anyone can feel free to report inaccuracies, improvements or requests through the Issue system provided by GitHub itself or by contacting the author privately:

- Email: **alessio.bandiera02@gmail.com**
- LinkedIn: **Alessio Bandiera**

The notes are constantly being updated, so please check if the changes have already been made in the most recent version.

Suggested prerequisites:

TODO

Licence:

1

TODO

1.1 Learning problems

1.1.1 Learning

A **learning problem** is defined by the following *three components*.

Definition 1.1: Learning

Learning is defined as *improving*, through *experience* E, at some *task* T, with respect to a *performance measure* P.

Example 1.1 (Learning problems)**.** Consider the problem of learning how to play Checkers; in this example, the *task* T is to be able to play the game itself, the *performance measure* P could be the percentage of games won in a tournament, but *experience* E is more complex.

In general, *experience* can be acquired in several ways:

- in this example, a human expert may suggest optimal moves for each configuration of the board; however, this approach may not generalize for any problem, as human experts may not exists for certain tasks;
- alternatively, the computer may play against a human, and automatically detect win, draw and loss configurations;
- lastly, the computer may play against itself, learning from its own successes and failures.

For this particular game, a possible **target function** (the function that would be useful to learn in order to solve the learning problem) could be the following

$$\mathrm{ChooseMove} : \mathrm{Board} \to \mathrm{Move}$$

which, given a board state, returns the best move to perform, but also

$$V : \text{Board} \to \mathbb{R}$$

which assigns a *score* to a given board.

For instance, consider the following target function:

$$V(b) = w_0 + w_1 \cdot bp(b) + w_2 \cdot rp(b) + w_3 \cdot bk(b) + w_4 \cdot rk(b) + w_5 \cdot bt(b) + w_6 \cdot rt(b)$$

where b is a given *board state*, and

- $bp(b)$ is the number of *black pieces*
- $rp(b)$ is the number of *red pieces*
- $bk(b)$ is the number of *black kings*
- $rk(b)$ is the number of *red kings*
- $bt(b)$ is the number of *red pieces threatened by black pieces*
- $rt(b)$ is the number of *black pieces threatened by red pieces*

In this formulation, V is a *linear compbination* of multiple coefficients w_i, which are unknown. Therefore, in this example **goal** of the *learning problem* is to **learn** V, or equivalently, to **estimate each coefficient** w_i. Note that this function *can be computed*.

1.1.2 Machine Learning problems

Definition 1.2: Dataset

Let $V(b)$ be the *true target function* (always *unknown*), $\hat{V}(b)$ be the *learned function* — an approximation of $V(b)$ computed by the *learning algorithm* — and $V_t(b)$ the *training value* of b in the *training data*. Lastly, let X be an input domain.

Given a set of n inputs

$$X_D := \{b_i \mid i \in [1, n]\} \subset X$$

a **dataset** is a set of *samples*, and it is denoted as

$$D = \{(b_i, V_t(b_i)) \mid b_i \in X_D\}$$

In the previous example, $\hat{V}(b)$ would have the following form

$$\hat{V}(b) = \hat{w}_0 + \hat{w}_1 \cdot bp(b) + \hat{w}_2 \cdot rp(b) + \hat{w}_3 \cdot bk(b) + \hat{w}_4 \cdot rk(b) + \hat{w}_5 \cdot bt(b) + \hat{w}_6 \cdot rt(b)$$

Definition 1.3: Machine Learning problem

A **machine learning problem** is the *task* of *learning a function* $f : X \to Y$, given a *dataset* D.

To **learn a function** f means *computing an approximation function* $\hat{f}$ that returns values as close as possible to f, especialy for values *outside* D

$$\forall x \in X - X_D \quad \hat{f}(x) \approx f(x)$$

Note that $|X_D| << |X|$, which makes the task of learning f quite challenging.

There are multiple types of Machine Learning (ML) problems, such as *dataset type* and *target function type*, some of which include:

- **classification**, which involves the *classification* of inputs into predetermined categories;
- **regression**, which involves the *approximation* of functions defined over $\mathbb{R}$;
- **unsupervised learing**, a type of ML problem where the model learns patterns from *unlabeled data*;
- **reinforcement learning**, in which an agent learns to make decisions by interacting with an *environment* and receiving *rewards* or *penalties* based on its actions.

Some of the various ML problems will be discussed in later sections.

1.1.3 Hypothesis spaces

Definition 1.4: Hypothesis space

Given an ML problem, an **hypothesis** h for the problem is an approximation of its target function, and its **hypothesis speace** H is the set of all possible hypothesis, i.e. the set of all functions that can be learned, which correspond to all the approximations of the target function of the ML problem.

Given this definition, **learning** can be defined as *searching in the hypothesis space*, using the dataset D and some performance function P of the given ML problem

$$h^* \in \arg\max_{h \in H} P(h, D)$$

A **performance measure** is a metric that evaluates the correctness of a given hypothesis, by comparing $h(x)$ and $f(x)$ for all $x \in X_D$, where f is the target function of the ML problem.

Example 1.2 (Hypothesis)**.** Consider the ML problem of *classifying natural numbers into primes and composite numbers.* The *target function* would be the following

$$f : \mathbb{N} \to \{\mathbb{P}, \mathbb{N} - \mathbb{P}\}$$

A dataset D for this ML problem would look like the following example

$$D = \{(1, \mathbb{P}), (3, \mathbb{P}), (5, \mathbb{P}), (6, \mathbb{N} - \mathbb{P}), (8, \mathbb{N} - \mathbb{P}), (10, \mathbb{N} - \mathbb{P})\}$$

The hypothesis space is the set of all possible *classification functions* of the form

$$h_A : \mathbb{N} \to \{A, \mathbb{N} - A\}$$

Definition 1.5: Inductive learning hypothesis

The **inductive learning hypothesis** states the following:

> Given an ML problem, any hypothesis that approximates the target function well over a sufficiently large set of training examples, will also approximate the target function well over other unobserved examples.

1.1.4 Version spaces

Definition 1.6: Hypothesis consistency

Given an ML problem defined by a target function $c : X \to Y$ — for some sets X and Y — and a training dataset $D = \{(x, c(x))\}$, a hypothesis $h \in H$ is said to be **consistent with** D if and only if

$$\forall x \in D \quad h(x) = c(x)$$

Note that this definition is important, because $h(x)$ can be evaluated for any $x \in X$, but only inputs that appear in the dataset can be verified, for which $c(x)$ is known. Therefore, *consistency* should be desirable for a hypothesis, since the real goal of an ML system is to find *the best* h that predicts correct values of $h(x')$, for instances $x' \notin X_D$, with respect to the unknown values $c(x')$.

Definition 1.7: Version space

The **version space** $VS_{H,D}$ of an ML problem, is the subset of hypotheses of H consistent with all training examples in D. Using symbols

$$VS_{H,D} := \{h \in H \mid \forall x \in X_D \quad h(x) = c(x)\} \subset H$$

Compute version spaces is not a straightforward task. For instance, consider the following algorithm.

Algorithm 1.1: List-Then-Eliminate

Given an ML problem, the algorithm returns $VS_{H,D}$.

```
1: function listThenEliminate($X_D$, $D$)
2:     $VS_{H,D} := H$                                  ▷ initially it contains any hypothesis
3:     for $(x, c(x)) \in D$ do
4:         $H' := \{h \in H \mid h(x) \neq c(x)\}$       ▷ set of inconsistent hypotheses for $x$
5:         $VS_{H,D} = VS_{H,D} - H'$
6:     end for
7:     return $VS_{H,D}$
8: end function
```

This algorithm can theoretically find the version space for any ML problem, but *it is not computable*, as it requires to **enumerate all the possible hypotheses**.

1.1.5 Representation issues

Consider a *binary classification* ML problem — commonly referred to as **Concept Learning** (CL) — , and its hypotheses space H; usually, every hypothesis is associated to the set of the instances that are classified as 1 by such hypothesis

$$\phi : H \to \mathcal{P}(S) : h \mapsto \{x \in X \mid h(x) = 1\}$$

note that it is not always true that, for any set $S \subseteq X$, there exists an h such that for each $x \in S$, $h(x) = 1$. Assume that, for the considered CL problem, there exists a hypothesis space H' such that

$$\forall S \subseteq \mathcal{P}(X) \quad \exists h \in H' \mid S = \{x \in X \mid h(x) = 1\}$$

therefore H' can represent *any subset* of X. Now, consider the following:

- $\forall x' \notin X_D \quad \exists h', h'' \in VS_{H',D} : \begin{cases} h'(x) = 1 \\ h''(x) = 0 \end{cases}$ because H' can represent any subset $S \subseteq X$ — therefore, for *all* inputs outside X_D, a system using H' would not be able to perform a prediction;
- $\exists x' \notin X_D \mid \exists h', h'' \in VS_{H,D} : \begin{cases} h'(x) = 1 \\ h''(x) = 0 \end{cases}$ because H represents some subsets $S \subseteq X$ — therefore, for *some* inputs outside X_D, a system using H would not be able to perform a prediction;
- $h^* \in \arg\max_{h \in H} P(h, D)$ is such that for all $x' \notin X_D$, $h^*(x)$ is either 1 or 0 — therefore, for *all* inputs outside X_D a system using h^* would be able to perform a prediction.

Note that H' is the most powerful hypothesis space, because it can represent *any subset* of X, H is less powerful than X because it can represent *some subsets* of X, and h^* will only represent *one* subset of X, meaning that it is the least powerful representation.

However, the more information the hypothesis space encapsulates about the values in X_D, the harder it becomes to **generalize** and **predict** values for samples *outside* X_D. In other words, a more expressive hypothesis space can **overfit** to the data, making it more difficult to make accurate predictions on unseen data.

The process of reducing the **representation power**, in favor of **generalization power** — i.e., reducing the hypothesis space from H' to H — is called **language bias**, because it is a *restriction*, a *bias*, on the language, used to represent the hypothesis. Moreover, the process of *selecting* one particular hypothesis among the set of possible ones — i.e., choosing $h^* \in H$ — is called **search bias**. Note that, in some contexts, it is also possible to choose a hypothesis h^* within H' directly.

Example 1.3 (Representation issues)**.** Consider the CL problem of enclosing all integer points on a 2D plane *labeled* with a $+$, thus

- X is the set of integer points on a 2D plane *labeled* with $+$ or $-$;
- Y is $\{+, -\}$;
- D is a set of pairs (p, y) where p is an integer point, and y is its label.

Consider the hypothesis space H that is composed of all the rectangles in the 2D plane with edges parallel to the axes; depending on the configuration of the points in X_D, H may not be able to enclose all the points with a $+$ in X_D. Now, consider the hypothesis space H' such that each element in H' is the *union* of the region enclosed by multiple rectangles with edges parallel to the axes; this secon hypothesis space is clearly more powerful, because it can represent *any* possible configuration of the input points with a $+$ in X_D.

However, for any given point x' that *is not* in X_D:

- it is *always* possible to find two elements in $VS_{H',D}$ wich will disagree whether x' has a $+$ label or not;
- there *may* be two elements in $VS_{H,D}$ which will disagree whether x' has a $+$ label or not;
- given h^*, it is *always* possible to *predict* whether x' has a $+$ label or not.

In machine learning, the concept of **learning bias** is crucial for improving a model's ability to generalize. A good learning bias helps guide the learning algorithm towards patterns in the data that are useful for predicting unseen samples, increasing the system's generalization power. This bias allows the model to make accurate predictions on new data that wasn't part of the training set. Without such a bias, a system would simply **memorize** the dataset, failing to predict values for samples outside the training set, rendering it *ineffective* in real-world applications. Systems lacking generalization capabilities would be of little use, as they wouldn't be able to provide meaningful predictions beyond the data they were trained on.

1.1.6 Data noise issues

In real-world applications, datasets often contain **noise**, which refers to *irrelevant or erroneous information* that can distort the true underlying patterns in the data. Noise can come from a variety of sources, including measurement errors, data entry mistakes, incomplete data, or random fluctuations in the system being studied. This noise complicates the *learning process*, as machine learning models may *struggle* to distinguish between true *signal* and *noise*, potentially leading to **overfitting**.

A noisy data-point in a dataset D can be formulated as a pair $(x_i, y_i) \in D$, where $y_i \neq c(x_i)$. This means that there may be *no consistent hypothesis* with noisy data, i.e. $VS_{H,D} = \varnothing$. In these scenarios, statisticale methods must be employed to implement robust algorithms, in order to *reduce* the noise in the data.

1.2 Performance evaluation

1.2.1 Classification problems

Consider the following typical ML classification problem:

- $f : X \to Y$ is some target function;
- $\mathcal{D}$ is the *probability distribution* over X (note that $\mathcal{D}$ $\underline{\text{may not be equal to}}$ D);
- S is a sample of n instances drawn from X, according to the distribution $\mathcal{D}$, for which $f(x)$ is known.

Additionally, consider a hypothesis h, solution of a learning problem obtained from S. What is the *best estimate* of the accuracy of h, over future instances drawn from $\mathcal{D}$? What is the *probable error* in this accuracy estimate?

Definition 1.8: True error

Given a classification problem, the **true error** of a hypothesis h — with respect to the target function f of the problem, and to the distribution $\mathcal{D}$ — is the probability that h will *misclassify* an instance drawn at random according to $\mathcal{D}$:

$$\text{error}_{\mathcal{D}}(h) := \Pr_{x \in \mathcal{D}} (f(x) \neq h(x))$$

Note that the **true error** of a classifcation problem *does not* refer to samples in the dataset, but to *any sample* in the input space, extracted accordin to $\mathcal{D}$. Moreover, note that this definition is *not operational*, and this term **cannot be computed**, because $f(x)$ *is not known.* In fact, this is a **target error**. Instead, consider the following type of error.

Definition 1.9: Sample error

Given a classification problem, the **sample error** of a hypothesis h — with respect to the target function f of the problem, and to the data sample S — is the proportion of examples that h misclassifies:

$$\text{error}_S(h) := \frac{1}{n}\sum_{x \in S} \delta(f(x), h(x))$$

where δ is an indicator variable such that

$$\delta(f(x), h(x)) := \begin{cases} 1 & f(x) \neq h(x) \\ 0 & f(x) = h(x) \end{cases}$$

In other words, the **sample error** is defined by a *counting function*, which counts how many examples h has misclassified, normalized by $|S|$.

The **goal** of a learning system is to be **accurate** in $h(x)$, for each $x \notin S$, and **accuracy** is defined as follows.

Definition 1.10: Accuracy

Given a classification problem, the **accuracy** of a hypothesis h is defined as follows:

$$\text{accuracy}(h) := 1 - \text{error}(h)$$

Note that, if $\text{accuracy}_S(h)$ is very high, but $\text{accuracy}_{\mathcal{D}}(h)$ is poor, the system considered is of little use.

In summary, the *true error* cannot be computed, but the *sample error* can be computed only on a small data sample, and it is crucial to ensure that the *sample error* closely approximates the *true error*, in order to achieve high accuracy.

Therefore, how well does $\text{error}_S(h)$ estimate $\text{error}_{\mathcal{D}}(h)$?

Definition 1.11: Estimation bias

The **estimation bias** is defined as follows:

$$\text{bias} := E[\text{error}_S(h)] - \text{error}_{\mathcal{D}}(h)$$

where $E[\text{error}_S(h)]$ is the expected value of the *sample error*, i.e. the weighted average over all the possible samples S — in this definition, S is treated as a random variable.

Note that, although it is impossible that $\text{error}_S(h) = \text{error}_{\mathcal{D}}(h)$ we can say that if $\text{bias} = 0$, then the *sample error* is a **good estimator** for the *true error*. This can be achieved through the following approaches.

- **Compute h and S independently**: if h is computed over the training set S, the $\text{error}_S(h)$ is said to be **optimistically biased**, meaning that the error measured on the training set is likely to be **lower** than the *true error* on unseen data. This happens because the hypothesis h has been directly fitted to the training set, so it tend to perform well on it. However, it does not guarantee that the model will perform equally well on data outside the dataset. Therefore, h should be computed on a dataset D, such that $D \cap S = \varnothing$.
- **Employ a large enough sample S**: even with an **unbiased** training set S, $\text{error}_S(h)$ may still differ from the *true error*; this is because the training set S is only a *sample* of the *full data distribution.* The smaller the training set, the more it is prone to variance, meaning that the performance of the hypothesis h on the training set may fluctuate more, compared to the full distribution. A **smaller sample** provides *less information* about the overall data distribution, leading to *greater potential differences* between training error and true error.

How can **confidence** of the *sample error*? What is the probability that $\text{error}_S(h)$ is close to the *true error*? For instance, consider the following approach.

Example 1.4 (Confidence intervals)**.** If S is a sample, containing $n \geq 30$ examples, drawn independently of a given hypothesis h and of each other, then with approximately $N\%$ probability, $\text{error}_\mathcal{D}(h)$ lies in the following interval

$$\text{error}_S(h) \pm z_N \sqrt{\frac{\text{error}_S(h) \cdot \text{accuracy}_S(h)}{n}}$$

where

$N\%$	50%	68%	80%	90%	95%	98%	99%
z_N	0.67	1.00	1.28	1.64	1.96	2.33	2.58

Note that *increasing n* **reduces the variance**, therefore it will me more probable that the *true error* will fall within an interval.

In machine learning, the following algorithm is usually employed, which **guarantees** an **unbiased estimator** of the *true error.*

Algorithm 1.2: Unbiased estimator

Given a classification problem, the algorithm returns an unbiased estimator for the *true error.*

```
1: function UNBIASEDESTIMATOR($D$)
2:     Partition $D$ into $T$ and $S$, such that $|T| = \frac{2}{3}|D|$
3:     Compute a hypothesis $h$ based on $T$
4:     return $\text{error}_S(h)$
5: end function
```

The split performed in the first line of the algorithm *partitions* the input dataset D into two sets T and S — therefore, $T \cup S = D$ and $T \cap S = \varnothing$. The set T is typically referred to as **training set**, and since h is computed based on T, the algorithm *guarantees* that h is **independent** of S, the set used to assess the *sample error*. Note that $\text{error}_S(h) := \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$ can be computed because S comes from D, therefore $f(x)$ is known for each $x \in S$. Note that it is *crucial* that D is partitioned into T and S **randomly**, as any split based on specific criteria would introduce *bias* into the partition.

Moreover, note that the $\frac{2}{3}$ split is a *good trade-off* between $|T|$ and $|S|$, because:

- having more samples from *training* and less for *testing* improves the performance of the model, but $\text{error}_S(h)$ does not approximate well $\text{error}_\mathcal{D}(h)$;
- having more sample from *testing* and less for *training* reduces the variance of estimation, and $\text{error}_S(h)$ approximates $\text{error}_\mathcal{D}(h)$ well, but its value mey be unsatisfactory.

Additionally, in general computing $\text{error}_S(h)$ is *insufficient*, as it represents just one realization of the random variable. To estimate $E[\text{error}_S(h)]$, it can be approximated by *averaging* the computed values of $\text{error}S_i(h)$ over several subsets S_i.

To compare two hypothesis, the following definition can be utilized.

Definition 1.12: True comparison

Given two hypotheses h and h' of some classification problem, the **true comparison** is defined as

$$d := \text{error}_\mathcal{D}(h) - \text{error}_\mathcal{D}(h')$$

and its *estimator* is defined as

$$\hat{d} := \text{error}_S(h) - \text{error}_{S'}(h')$$

Note that, in this definition, $\hat{d}$ is an **unbiased estimator** for d it and only if h, h', S and S' are independent of each other, but it is *still valid* if $S = S'$. Lastly, note that $d = E[\hat{d}]$.

Definition 1.13: Overfitting

Consider an hypothesis $h \in H$ of a classification problem computed over some training data; h **overfits** the training data if there exists an alternative hypothesis $h' \in H$ such that

$$\text{error}_S(h) < \text{error}_S(h')$$

but

$$\text{error}_\mathcal{D}(h) > \text{error}_\mathcal{D}(h')$$

When training a learning algorithm L, different splits of D may occur, resulting in different training sets, e.g. T and T', and consequently, different hypotheses, e.g. $h = L(T)$ and $h' = L(T')$. To estimate the performance of L the following algorithm can be employed.

Algorithm 1.3: k-fold Cross Validation (hypotheses)

Given a classification problem, the algorithm returns an *unbiased estimator* of the *expected value* of the the *sample error*.

```
 1: function kFoldCrossValidationHyps($D$, $L$, $k$, $\alpha$)
 2:     Partition $D$ into $k$ disjoint sets $S_1, \dots, S_k$, such that $\forall i \in [i, k] \quad |S_i| > \alpha$
 3:     for $i \in [1, k]$ do
 4:         $T_i := D - S_i$                                  ▷ note that $T_i \cap S_i = \varnothing$
 5:         $h_i := L(T_i)$
 6:         $\delta_i := \mathrm{error}_{S_i}(h_i)$
 7:     end for
 8:     $\mathrm{error}_{L,D} := \frac{1}{k} \sum_{i=1}^{k} \delta_i$
 9:     return $\mathrm{error}_{L,D}$
10: end function
```

This algorithm simply partitions the dataset D into multiple subsets, evaluates the performance of L on each split S_i and $T_i = D - S_i$, and returns the average sample error across all splits. Moreover, $\alpha = 30$ usually, and it is necessary to fix a minimum size for the partitions otherwise each single partition would not contribute enough to the average. Lastly, note that

$$\mathrm{accuracy}_{L,D}(h) := 1 - \mathrm{error}_{L,D}(h)$$

How can two learning algorithms L_A and L_B be compared instead? It would be useful to estimate

$$\mathrm{error}_{\mathcal{D}}(L_A(T)) - \mathrm{error}_{\mathcal{D}}(L_B(T))$$

where $L(T)$ is the hypothesis output of L (computed using T), by employing

$$E[\mathrm{error}_S(L_A(T)) - \mathrm{error}_S(L_B(T))]$$

This can be performed through k-fold cross validation as well, as shown below.

Algorithm 1.4: k-fold Cross Validation (learning algorithms)

Given two classification algorithms, the algorithm estimates which one is better.

```
function KFOLDCROSSVALIDATIONALGS($D$, $L_A$, $L_B$, $k$, $\alpha$)
    Partition $D$ into $k$ disjoint sets $S_1, \dots, S_k$, such that $\forall i \in [i, k] \quad |S_i| > \alpha$
    for $i \in [1, k]$ do
        $T_i := D - S_i$                                    ▷ note that $T_i \cap S_i = \varnothing$
        $h_A := L_A(T_i)$
        $h_B := L_B(T_i)$
        $\delta_i := \text{error}_{S_i}(h_A) - \text{error}_{S_i}(h_B)$
    end for
    $\bar{\delta} := \frac{1}{k} \sum_{i=1}^{k} \delta_i$
    return $\bar{\delta}$
end function
```

Note that if $\bar{\delta} < 0$, then L_A is estimated to be *better than* L_B.

1.2.2 Classification accuracy

In *classification tasks*, the terms **true positive**, **false negative**, and **false positive** and **true negative** refer to the various outcomes of a binary classification model. The effectiveness of a model is assessed based on how *accurately* it classifies instances into these categories, which ultimately reflects its overall performance. The following table illustrates the various terms in detail.

	predicted yes	*predicted* no
true yes	true positive (TP)	false negative (FN)
true no	false positive (FP)	true negative (TN)

Therefore, the following definitions hold.

Definition 1.14: Error rate and accuracy

The **error rate** of classification problems is defined as follows:

$$\text{error rate} := \frac{|\text{errors}|}{|\text{instances}|} = \frac{\text{FN} + \text{FP}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Additionally, **accuracy** is defined as

$$\text{accuracy} := 1 - \text{error rate} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Accuracy can be used as a performance metric for classification problems, but is it always *suitable*? For instance, consider the following CL problem, where

- the target function has the form $f : X \to \{+, -\}$
- the dataset D contains 90% of *negative samples*

and consider two hypotheses h_1 and h_2 such that h_1 has 90% of *accuracy* and h_2 has 85% of *accuracy* — assuming that both hypotheses were computed as accurately as possible, without bias. This may suggest that h_1 is better than h_2, but consider the following hypothesis:

$$h_1 \in H \mid \forall x \in X \quad h(x) = -$$

clearly, this hypothesis has 90% accuracy because, when tested on D, which consists of 90% *negative samples*, it will be correct exactly 90% of the time. However, this hypothesis is not a *suitable solution* for a CL problem, as it fails to model any realistic scenarios.

The issue arises from having **too many** *negative samples* in the dataset. This poses a *significant challenge* in ML because an *overabundance* of samples of the same type makes it difficult to develop models that can classify accurately.

This example clearly shows that, in general, **accuracy** alone is *not enough* to assess the performance of the solution of a classification problem, as **unbalanced datases** are *very common* in real-world scenarios (e.g. anomaly detection).

Now, consider the following metrics.

Definition 1.15: Recall and precision

Recall and **precision** are defined as follows:

$$\mathrm{recall} := \frac{|\text{true positives}|}{|\text{real positives}|} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

$$\mathrm{precision} := \frac{|\text{true positives}|}{|\text{predicted prositives}|} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

These metrics are generally **better suited** for classification problems and are widely adopted across various branches of *statistics*. In particular, **recall** measures how good is the system at *avoiding false negatives* — indeed, recall $= 0$ when FN $= 0$ — while the **precision** measures the system's ability to *avoid false positives* — in fact, precision $= 0$ when FP $= 0$.

The significance of *false positives* and *false negatives* varies depending on the system in question. For example, consider an autonomous car that must learn to identify whether a pedestrian is present in front of it. In this scenario, *false positives* — instances where the car mistakenly detects a pedestrian — are far less critical than *false negatives*, where the car fails to recognize an actual pedestrian. This means that, in assessing the performance of such a classification system, **recall** is much more relevant.

Definition 1.16: $F1$-score

The $F1$**-score** is defined as follows:

$$F1\text{−score} := 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

This score measures the trade-off between *precision* and *recall*. Note that there are are types of *performance measures*, depending on the context, such as

- **true positive rate**:
$$\text{TPR} := \frac{\text{TP}}{\text{TP} + \text{FN}}$$
- **true negative rate**:
$$\text{TNR} := \frac{\text{TN}}{\text{TN} + \text{FP}}$$
- **false positive rate**:
$$\text{FPR} := \frac{\text{FP}}{\text{TN} + \text{FP}}$$
- **false negative rate**:
$$\text{FNR} := \frac{\text{FN}}{\text{TP} + \text{FN}}$$

Note that Section 1.2.2 can be extended with other type of **classes** too — in the table shown, there are two classes, namely the *true class* and the *predicted class*. For instance, if there are n type of classes, the following table can be created

T/P	C_1	$\dots$	C_n
C_1			
$\vdots$			
C_n			

This matrix is called **multi-class confusion matrix**. Note that a cell $c_{i,j}$ represents how many times an instance of class C_i is classified in class C_j. Moreover, the **diagonal** of this matrix contains the *accuracy* of each class, and the values that lie outside the diagonal can be used to determine which classes are **confused** the most. Clearly, the sum $\sum_{j=1}^{n} c_{i,j}$ of the cells in the i-th row add up to the total number of saples of class C_i of the dataset. Usually, confusion matrices are represented as colored tables.

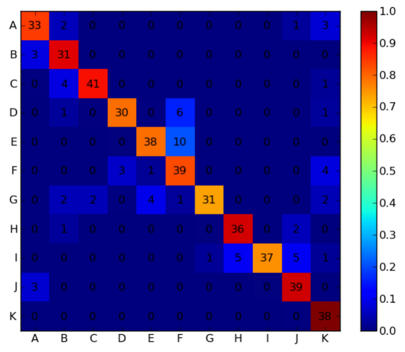

Figure 1.1: A multi-class confusion matrix.

1.2.3 Regression performance

Consider a regression problem defined by a target function $f : X \to \mathbb{R}^d$ and a testing set $S = \{(x_i, t_i) \mid i \in [1, n], x_i \in X, t_i \in \mathbb{R}^d\}$. The **performance** of regression problems can be assessed using multiple methods, which are all expressed in terms of

$$|\hat{f}(x_i) - t_i|$$

for some $(x_i, t_i) \in S$, which measures how much the estimated value $\hat{f}(x_i)$ of x_i differs from the known value t_i of x_i.

Definition 1.17: Mean Absolute Error

Given a regression problem, the **mean absolute error** (MAE) is defined as follows

$$\text{MAE} := \frac{1}{n} \sum_{i=1}^{n} |\hat{f}(x_i) - t_i|$$

Definition 1.18: Mean Squared Error

Given a regression problem, the **mean squared error** (MSE) is defined as follows

$$\text{MSE} := \frac{1}{n} \sum_{i=1}^{n} \left(\hat{f}(x_i) - t_i\right)^2$$

Additionally, the **root mean squared error** (RMSE) is defined as $\sqrt{\text{MSE}}$.

Definition 1.19: Mean Absolute Percentage Error

Given a regression problem, the **mean absolute percentage error** (MAPE) is defined as follows

$$\text{MAPE} := \frac{1}{n} \sum_{i=1}^{n} \frac{|\hat{f}(x_i) - t_i|}{t_i}$$

Definition 1.20: Mean Squared Percentage Error

Given a regression problem, the **mean absolute percentage error** (MAPE) is defined as follows

$$\text{MAPE} := \frac{100}{n} \sum_{i=1}^{n} \frac{|\hat{f}(x_i) - t_i|}{t_i}$$

Definition 1.21: Mean Squared Percentage Error

Given a regression problem, the **mean absolute percentage error** (MAPE) is defined as follows

$$\text{MAPE} := \frac{100}{n} \sum_{i=1}^{n} \left(\frac{\hat{f}(x_i) - t_i}{t_i} \right)^2$$

Analogously, the **root mean squared percentage error** (RMSPE) is defined as $\sqrt{MSPE}$.

Lastly, note that the k-fold cross validation procedure (described in Algorithm 1.3) can be modified for regression problems, using appropriate metrics.

Algorithm 1.5: k-fold Cross Validation (regression)

Given a regression problem, the algorithm returns an *unbiased estimator* of the *expected value* of the the *sample error*.

```
 1: function kFoldCrossValidationRegr($D$, $L$, $k$, $\alpha$)
 2:     Partition $D$ into $k$ disjoint sets $S_1, \dots, S_k$, such that $\forall i \in [i, k] \quad |S_i| > \alpha$
 3:     for $i \in [1, k]$ do
 4:         $T_i := D - S_i$                                ▷ note that $T_i \cap S_i = \varnothing$
 5:         $h_i := L(T_i)$
 6:         $\delta_i := \text{MAE}_{S_i}(h_i)$
 7:     end for
 8:     $\text{MAE}_{L,D} := \frac{1}{k} \sum_{i=1}^{k} \delta_i$
 9:     return $\text{MAE}_{L,D}$
10: end function
```