Fundamentals of Machine Learning

Introduction to Machine Learning

Exercises

Instructions

- This exercise set mixes conceptual (theory) questions and small practical tasks.
- For programming exercises, you may use any language (Python with `scikit-learn` is recommended).
- When a public dataset is requested, a direct online source is provided.
- Aim for concise but well-argued answers; plots or tables are encouraged when helpful.

1 Conceptual Questions

Exercise 1 – "Is this Machine Learning?"

For each scenario below, say whether it is best described as *(a) traditional programming* or *(b) machine learning*. Briefly justify.

(a) A spam filter built from hundreds of manually written rules (e.g., "if the subject contains 'WIN MONEY NOW' then mark as spam").

(b) A system that takes thousands of labelled emails (spam / not spam) and automatically learns a classifier.

(c) A physics simulation engine that solves Newton's equations exactly for a given configuration.

(d) A recommendation system that observes the movies a user watched and then suggests new ones using a learned model.

Explain how the notion of *generalization to unseen data* appears in the ML cases.

Exercise 2 – Features and Representation Design

Consider the following two tasks:

(T1) Predict whether a user will still be active in a mobile game 7 days after installation (binary classification).

(T2) Group images of handwritten digits into clusters, without labels (unsupervised learning).

(a) For (T1), propose at least **six** candidate features that could be collected during the first **24 hours** of gameplay. For each feature, state whether it is numerical, categorical, or binary, and briefly justify why it might be informative.

(b) For (T2), list at least **four** possible feature representations for images (from very simple to more advanced). Indicate which ones are likely to be more robust to small translations or deformations of the digits.

(c) Discuss one possible drawback of using too many raw features in (T1), and one drawback of using too few features.

Exercise 3 – Distances and Similarity

Let $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$ be two feature vectors.

(a) Write the definitions of the Manhattan distance (ℓ_1) and Euclidean distance (ℓ_2) between x and y.

(b) Suppose $x = (0, 0)$ and $y = (3, 4)$ in $\mathbb{R}^2$. Compute $d_1(x, y)$ and $d_2(x, y)$.

(c) Consider a dataset with two features: *age in years* and *number of mobile games installed*. Explain why it might be problematic to compute Euclidean distances without any scaling.

(d) Describe one simple scaling or normalization strategy, and explain how it affects distance-based methods such as k-nearest neighbours.

Exercise 4 – Supervised vs. Unsupervised Learning

You are given a dataset of patients with the following information:

- Age, blood pressure, cholesterol level (numerical features),
- A binary indicator: smoker / non-smoker,
- For some patients: whether they developed a certain disease within 5 years.

(a) Formulate two different ML tasks on this dataset: one supervised and one unsupervised.

For each task, specify:

- The input features,
- The output / objective,
- A plausible real-world use case.

(b) Explain how you would evaluate the performance of the supervised model, and how you would *evaluate or interpret* the unsupervised model.

Exercise 5 – Overfitting Intuition

Consider a binary classification problem where the training accuracy is 100%, but the accuracy on a held-out test set is only 60%.

(a) Explain why this is a typical sign of *overfitting*.

(b) Give two distinct strategies to reduce overfitting, and briefly justify them (one should be related to the model complexity, the other to the data).

(c) Provide a simple real-world analogy (not related to ML) that illustrates the difference between memorization and generalization.

2 Practical Exercises

For all programming tasks, you may use any language (Python is recommended). You are encouraged to focus on:

- exploring and understanding datasets,
- designing meaningful feature representations,
- implementing very simple models such as the perceptron.

Exercise 6 – Exploring the Iris Dataset

Use the classic **Iris** dataset, a small multiclass classification dataset with 150 samples and 4 numeric features (sepal length, sepal width, petal length, petal width) and 3 species of iris. It is available at the UCI Machine Learning Repository and on Kaggle. :contentReference[oaicite:0]index=0

Dataset sources.

- UCI ML Repository (Iris Data Set): `https://archive.ics.uci.edu/ml/datasets/iris`

- Kaggle (Iris Species): https://www.kaggle.com/datasets/uciml/iris

Tasks.

(a) Load the dataset and report:

- the number of samples,
- the number of features,
- the number of samples in each class.

(b) For each feature, compute basic descriptive statistics (minimum, maximum, mean, standard deviation). Present them in a small table.

(c) Produce at least one visualization that helps you understand how well the classes are separated in feature space. Possible ideas:

- a scatter plot of two features (e.g., petal length vs. petal width),
- colored by class.

Comment briefly on which pairs of features appear most informative.

(d) Consider the task of predicting whether a flower is *setosa* or *non-setosa* (merge the two non-setosa classes). Based on your plots and statistics, do you expect there exists a straight line in the plane (for some pair of features) that can separate setosa from non-setosa? Justify your answer qualitatively.

Exercise 7 – A Simple Perceptron on a Linearly Separable Subset

In this exercise, you will implement a basic **perceptron** and apply it to a binary classification task derived from the Iris dataset. The goal is not to use an ML library implementation, but to understand the algorithm as a simple model.

Data preparation.

(a) Start from the Iris dataset again.

(b) Restrict the dataset to two classes, e.g., *Iris setosa* and *Iris versicolor*, so that you have a binary task.

(c) Select two numeric features only (for example, petal length and petal width). Standardize or rescale them if you wish (e.g., to zero mean and unit variance), but clearly state what you did.

Perceptron implementation.

(d) Implement a perceptron classifier from scratch:

- initialize the weight vector and bias (e.g., zeros or small random values),
- repeatedly scan the training set and update the weights whenever a sample is misclassified (use a fixed learning rate),
- stop after a fixed number of passes or when there are no misclassifications in one full pass.

(e) Split the data into a training set and a test set (for example, 70% train, 30% test). Train the perceptron on the training set only, but report the number of updates (weight changes) during training.

(f) Evaluate the perceptron on both the training and test sets. Report the accuracy and discuss whether the model seems to *generalize* well.

(g) Plot the training data points together with the final decision boundary learned by the perceptron (a line in the 2D feature plane). Comment briefly:

- Are there many points close to the boundary?
- Does the boundary match your intuition from Exercise 6?

References

[1] UCI Machine Learning Repository, "Iris Data Set," https://archive.ics.uci.edu/ml/datasets/iris.

[2] Kaggle, "Iris Species," https://www.kaggle.com/datasets/uciml/iris.

[3] UCI Machine Learning Repository, "Breast Cancer Wisconsin (Diagnostic) Data Set," https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic.

[4] Scikit-learn, "Toy Datasets," https://scikit-learn.org/stable/datasets/toy_dataset.html.

[5] Kaggle, "Gamelytics: Mobile Analytics Challenge," https://www.kaggle.com/datasets/debs2x/gamelytics-mobile-analytics-challenge.

[6] Wikipedia, "List of datasets for machine-learning research," https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research.

Appendix

A The Perceptron Algorithm

A.1 Motivation and Intuition

The perceptron is one of the earliest and simplest models for binary classification. It takes a feature vector $x \in \mathbb{R}^d$ as input and predicts a binary label $y \in \{-1, +1\}$.

The main idea:

- Represent the input as a point in d-dimensional space.
- Learn a *linear decision boundary*, i.e., a hyperplane that tries to separate the two classes.
- Update the hyperplane whenever a training point is misclassified.

Despite its simplicity, the perceptron already illustrates several important concepts in machine learning: feature representation, linear decision boundaries, training vs. test error, and the effect of learning rates and number of iterations.

A.2 Model Definition

We assume we are given a training set

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{n},$$

where each $x^{(i)} \in \mathbb{R}^d$ is a feature vector and $y^{(i)} \in \{-1, +1\}$ is the corresponding label.

The perceptron maintains:

- a weight vector $w \in \mathbb{R}^d$,
- a bias (or intercept) $b \in \mathbb{R}$.

Given an input x, the perceptron computes the *score*

$$s(x) = w^\top x + b,$$

and outputs the prediction

$$\hat{y} = \begin{cases} +1 & \text{if } s(x) \geq 0, \\ -1 & \text{otherwise.} \end{cases}$$

Geometrically, the set of points satisfying $w^\top x + b = 0$ forms a hyperplane. The sign of $w^\top x + b$

tells us on which side of the hyperplane x lies.

A.3 Learning Rule (Training Algorithm)

The perceptron learning algorithm is an iterative procedure that adjusts w and b whenever a mistake is made on a training example.

Basic Algorithm

1. Initialize $w = 0$ (or small random values) and $b = 0$.
2. Repeat for a fixed number of passes over the data (epochs), or until no mistakes occur:
    1. For each training example $(x^{(i)}, y^{(i)})$:
        1. Compute the prediction
        $$\hat{y}^{(i)} = \text{sign}(w^\top x^{(i)} + b).$$
        2. If $\hat{y}^{(i)} \neq y^{(i)}$ (the example is misclassified), update:
        $$w \leftarrow w + \eta\, y^{(i)} x^{(i)}, \quad b \leftarrow b + \eta\, y^{(i)},$$
        where $\eta > 0$ is the learning rate (often set to $\eta = 1$ for the basic algorithm).

Intuition of the update:

- If the true label is $y^{(i)} = +1$ but the model predicts -1, we *add* $x^{(i)}$ to w (scaled by η), so that the score $w^\top x^{(i)}$ increases.
- If the true label is $y^{(i)} = -1$ but the model predicts $+1$, we *subtract* $x^{(i)}$ from w, so that the score $w^\top x^{(i)}$ decreases.

A.4 A Simple 2D Example

Consider a toy dataset in $\mathbb{R}^2$:

Index	x_1	x_2	y
1	2	1	$+1$
2	-1	-2	-1
3	1	2	$+1$
4	-2	-1	-1

This dataset is linearly separable: points with $y = +1$ are in the upper-right region, and points

with $y = -1$ are in the lower-left region.

Visualization of the Data Points

Figure 1 shows the four points in the 2D plane.

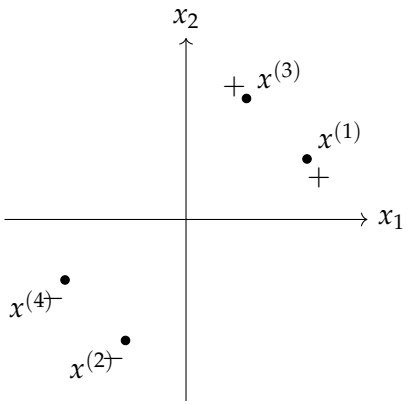

Figure 1: Toy dataset for the perceptron in 2D.

We train a perceptron with learning rate $\eta = 1$. Start with

$$w^{(0)} = (0, 0), \quad b^{(0)} = 0.$$

We go through the data in order; each pass over the data is called an *epoch*.

Epoch 1

Example 1: $(2, 1)$**,** $y = +1$**.**

$$s = w^{(0)\top} x^{(1)} + b^{(0)} = 0, \quad \hat{y} = \text{sign}(0) = +1$$

(we can define $\text{sign}(0) = +1$ for convenience). Prediction is correct, so no update.

Example 2: $(-1, -2)$**,** $y = -1$**.**

$$s = w^{(0)\top} x^{(2)} + b^{(0)} = 0, \quad \hat{y} = +1$$

Prediction is wrong. Update:

$$w^{(1)} = w^{(0)} + yx^{(2)} = (0, 0) + (-1) \cdot (-1, -2) = (1, 2),$$

$$b^{(1)} = b^{(0)} + y = 0 + (-1) = -1.$$

Example 3: $(1,2)$**,** $y=+1$**.**

$$s = w^{(1)\top} x^{(3)} + b^{(1)} = (1,2)\cdot(1,2) - 1 = (1+4) - 1 = 4 > 0,$$

$$\hat{y} = +1 \quad \text{(correct)}.$$

No update: $w^{(2)} = (1,2)$, $b^{(2)} = -1$.

Example 4: $(-2,-1)$**,** $y=-1$**.**

$$s = w^{(2)\top} x^{(4)} + b^{(2)} = (1,2)\cdot(-2,-1) - 1 = (-2-2) - 1 = -5 < 0,$$

$$\hat{y} = -1 \quad \text{(correct)}.$$

No update: $w^{(3)} = (1,2)$, $b^{(3)} = -1$.

At the end of the first epoch, all four points are already classified correctly. So the algorithm can stop with

$$w = (1,2), \quad b = -1.$$

Visualization of the Decision Boundary

The decision boundary is the line

$$w^\top x + b = 0 \quad \Longleftrightarrow \quad 1\cdot x_1 + 2\cdot x_2 - 1 = 0.$$

Figure 2 shows the same points together with the learned decision boundary.

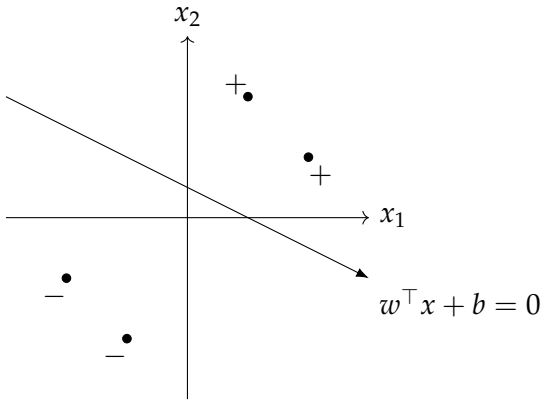

Figure 2: Perceptron decision boundary separating the two classes.

You can verify that the two positive examples lie on one side of the line and the two negative examples on the other side.

A.5 Small Hand-Calculation Exercises

The following exercises are designed to be solvable by hand on paper, to build intuition about how the perceptron updates its parameters.

Exercise A (1D Perceptron)

Consider a 1D problem ($d = 1$). Inputs are real numbers $x \in \mathbb{R}$, labels are $y \in \{-1, +1\}$. We use learning rate $\eta = 1$ and define $\text{sign}(0) = +1$.

Training set:

Index	x	y
1	-1	-1
2	2	$+1$

We start with $w^{(0)} = 0$, $b^{(0)} = 0$.

1. For each of the two examples, compute the score $s = wx + b$ and the prediction $\hat{y} = \text{sign}(s)$.
2. Whenever the prediction is wrong, update

$$w \leftarrow w + yx, \quad b \leftarrow b + y.$$

3. Perform two full passes (epochs) over the data (in the fixed order: example 1, then example 2). Write down w and b after each update.
4. At the end, what are the final values of w and b? Does the resulting decision rule $wx + b \geq 0$ correctly classify both points?

Exercise B (2D Perceptron with One Mistake)

Consider a 2D problem with inputs $x = (x_1, x_2)$ and labels $y \in \{-1, +1\}$. We again use $\eta = 1$ and start with $w^{(0)} = (0, 0)$, $b^{(0)} = 0$.

Training set:

Index	x_1	x_2	y
1	1	0	$+1$
2	0	1	$+1$
3	-1	-1	-1

1. Go through the three examples once (one epoch). For each example:
   - compute $s = w^\top x + b$ and $\hat{y} = \text{sign}(s)$,

- if $\hat{y} \neq y$, update w and b.

2. Track w and b after each update. In particular, write down w and b after finishing example 3.

3. Using the final (w, b), check whether the perceptron now classifies all three points correctly.

4. Sketch the three points in the plane and draw the resulting decision boundary $w^\top x + b = 0$. By inspection, explain why a linear classifier can separate these three points.

Exercise C (Effect of the Order of Examples)

Using the same dataset as in Exercise B, repeat the perceptron training for one epoch, but now present the examples in the order:

$$(3) \rightarrow (1) \rightarrow (2).$$

1. Perform the updates by hand and record the final w and b.

2. Compare the final parameters with those obtained in Exercise B. Are they the same?

3. Reflect on how the order of training examples can influence the final decision boundary for the perceptron.