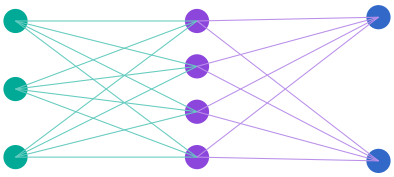


Artificial Intelligence Applications in Matter Science

Workshop Educational Document

Classification using an artificial neuron and random forests

Date: December 09, 2025

Instructor: Dr. Samir Kenouche

Institution: LCA Laboratory

Team: Multi-scale quantum modeling of chemical systems

Department of matter science – University of Biskra

Introduction

Machine Learning (ML) classification methods, such as neural networks and random forests, have become essential tools in modern chemistry and physics. Their ability to extract hidden patterns from complex, high-dimensional data allows researchers to identify material properties, predict reaction outcomes, and classify structural or spectroscopic signatures with enhanced speed and accuracy. Neural networks excel at capturing intricate nonlinear relationships that commonly arise in quantum systems, molecular behavior, or heterogeneous materials, making them suitable for tasks such as spectral feature recognition or phase identification.

Random forests, on the other hand, provide robust ensemble-based learning that handles noisy measurements and small datasets conditions frequently encountered in experimental research. Their interpretability, through measures such as feature importance, offers valuable insights into which chemical descriptors or physical variables most strongly influence a classification task. When combined, these complementary approaches contribute to more reliable, data-driven decision making in disciplines where conventional analytical models are often limited by complexity, variability, or incomplete theoretical understanding.

Moreover, the growing availability of high-throughput experiments and large-scale simulations has amplified the need for automated classification tools capable of managing massive datasets. ML classification not only accelerates scientific workflows but also uncovers relationships that might otherwise remain hidden, supporting the development of new materials, improved catalysts, and deeper physical intuition.[1] [2] [3]

In these practical activities, we will present some classification applications using artificial neuron (commonly known as Perceptron) as well as a practical example of how random forests work. The theoretical concepts presented in this document are mainly taken from the references cited here.

Copyright Notice

[1] Goodfellow, I., Bengio, Y., Courville, A. "Deep Learning." MIT Press, 2016.
[2] Breiman, L. "Random Forests." Machine Learning, 45, 5–32 (2001).
[3] Butler, K. T., et al. "Machine learning for molecular and materials science." Nature, 559, 547–555 (2018).

Sub-workshop 1: Chemical Property Classification Using Perceptron

In computational chemistry and chemoinformatics, molecules can be represented by vectors of numerical descriptors (e.g., dipole moment, LogP, molecular weight, number of hydrogen bond donors). A **perceptron** is one of the simplest and most interpretable machine learning algorithms that can classify molecules according to physicochemical properties such as polarity, solubility, or toxicity.

Given molecular descriptors $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$, the perceptron predicts whether a compound possesses a specific property (e.g., polar vs. non-polar) using a linear decision boundary.

$$y = f(\mathbf{w}^T\mathbf{x} + b)$$

Where $\mathbf{w}$ is a vector of weights, and b is a bias term.

Chemical Usefulness

In chemoinformatics, perceptrons can classify molecular activities, predict solubility, estimate toxicity levels, or identify reactive functional groups. They provide a transparent linear approximation of the chemical space separation between different molecular behaviors.

Mathematical Formulation of the Classical Perceptron

The perceptron computes a weighted sum:

$$z_i = \mathbf{w}^T\mathbf{x}_i + b$$

and applies the step activation function:

$$f(z_i) = \begin{cases} 1, & \text{if } z_i \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Given the true class $t_i \in \{0, 1\}$, the error is defined as:

$$e_i = t_i - y_i$$

The update rules for each misclassified sample are:

$$\boxed{\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta e_i \mathbf{x}_i} \quad \text{and} \quad \boxed{b^{(k+1)} = b^{(k)} + \eta e_i}$$

where η is the learning rate.

Learning Intuition

Each time a molecule is misclassified, the decision boundary shifts in the direction that reduces future misclassifications. The vector $\mathbf{w}$ defines the orientation of this boundary, while b shifts it.

Activity 1: Perceptron Architecture for Chemical Property Classification

Polarity prediction

The perceptron used to classify molecules as polar or non-polar takes two chemical descriptors as inputs: the dipole moment x_1 and the hydrophobicity index (LogP) x_2. These are linearly combined with learned weights and a bias term, followed by a sigmoid activation function to produce the probability of polarity.

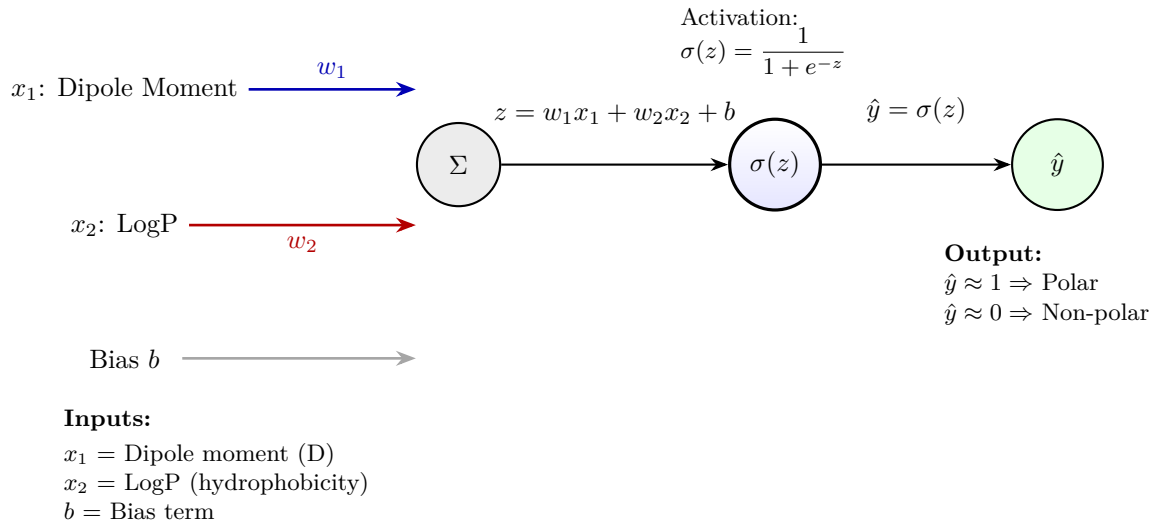


Figure 1: Clear schematic of the single-layer perceptron used for classifying molecules based on polarity. Inputs correspond to chemical descriptors, and the output neuron produces a probabilistic polarity prediction.

1 Numerical Example: Polar vs. Non-Polar Molecules

We consider the classification of molecules based on two chemical descriptors:

- x_1: Dipole moment (Debye),
- x_2: LogP (lipophilicity).

We illustrate the perceptron training process numerically for the chemical property classification problem (polar vs. non-polar molecules), using the simplified two-feature dataset:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \text{Dipole Moment (D)} \\ \text{LogP} \end{bmatrix}, \quad y \in \{0, 1\}.$$

The four samples used are summarized below:

Molecule	x_1 (Dipole Moment)	x_2 (LogP)	Target y
Ethanol	1.85	-0.90	1
Acetone	1.69	-0.10	1
Hexane	0.00	2.10	0
Benzene	0.08	3.50	0

Table 1: Dataset used for the perceptron training example.

Parameters Initialisation

We start with random initial weights:

$$w_1^{(0)} = 0.2, \quad w_2^{(0)} = -0.1, \quad b^{(0)} = 0.0, \quad \eta = 0.1.$$

The perceptron output is given by:

$$\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}, \quad \text{where } z = w_1 x_1 + w_2 x_2 + b.$$

The weight update rule for each epoch is:

$$w_i^{(t+1)} = w_i^{(t)} + \eta\,(y - \hat{y})\,x_i, \quad b^{(t+1)} = b^{(t)} + \eta\,(y - \hat{y}).$$

Iteration 1

For Ethanol ($x = [1.85, -0.90], y = 1$):

$$z^{(1)} = (0.2)(1.85) + (-0.1)(-0.90) = 0.46, \quad \hat{y}^{(1)} = 0.613.$$
$$\delta^{(1)} = y - \hat{y} = 0.387.$$
$$w_1^{(1)} = 0.2 + 0.1(0.387)(1.85) = 0.2715,$$
$$w_2^{(1)} = -0.1 + 0.1(0.387)(-0.9) = -0.1348,$$
$$b^{(1)} = 0.1(0.387) = 0.0387.$$

Iteration 2

For Acetone ($x = [1.69, -0.10], y = 1$):

$$z^{(2)} = 0.2715(1.69) + (-0.1348)(-0.10) + 0.0387 = 0.5078, \quad \hat{y}^{(2)} = 0.624.$$
$$\delta^{(2)} = 0.376.$$
$$w_1^{(2)} = 0.2715 + 0.1(0.376)(1.69) = 0.3359,$$
$$w_2^{(2)} = -0.1348 + 0.1(0.376)(-0.10) = -0.1386,$$
$$b^{(2)} = 0.0387 + 0.1(0.376) = 0.0763.$$

Iteration 3

For Hexane ($x = [0.00, 2.10], y = 0$):

$$z^{(3)} = 0.3359(0) + (-0.1386)(2.1) + 0.0763 = -0.2157, \quad \hat{y}^{(3)} = 0.446.$$
$$\delta^{(3)} = -0.446.$$
$$w_1^{(3)} = 0.3359 + 0.1(-0.446)(0) = 0.3359,$$
$$w_2^{(3)} = -0.1386 + 0.1(-0.446)(2.1) = -0.2323,$$
$$b^{(3)} = 0.0763 + 0.1(-0.446) = 0.0317.$$

Iteration 4

For Benzene ($x = [0.08, 3.50], y = 0$):

$$z^{(4)} = 0.3359(0.08) + (-0.2323)(3.5) + 0.0317 = -0.766, \quad \hat{y}^{(4)} = 0.317.$$
$$\delta^{(4)} = -0.317.$$
$$w_1^{(4)} = 0.3359 + 0.1(-0.317)(0.08) = 0.3334,$$
$$w_2^{(4)} = -0.2323 + 0.1(-0.317)(3.5) = -0.3433,$$
$$b^{(4)} = 0.0317 + 0.1(-0.317) = 0.0000.$$

Iteration 5-10 (Summary Table)

Epoch	w_1	w_2	b	$\hat{y}_{\text{Ethanol}}$	$\hat{y}_{\text{Hexane}}$	E_{avg}
5	0.397	-0.368	0.033	0.66	0.41	0.122
6	0.441	-0.395	0.055	0.69	0.36	0.104
7	0.507	-0.445	0.083	0.73	0.29	0.083
8	0.576	-0.486	0.102	0.77	0.24	0.071
9	0.633	-0.518	0.120	0.80	0.19	0.061
10	0.662	-0.541	0.131	0.82	0.15	0.054

Table 2: Evolution of weights, bias, and outputs over 10 epochs.

Interpretation

Interpretation of Iterations

- The weights (w_1, w_2) grow in opposite directions: w_1 (dipole moment) increases, reinforcing the idea that high dipole moment favors polarity; w_2 (LogP) decreases, since high hydrophobicity discourages polarity.
- The bias b stabilizes around 0.13, effectively centering the decision surface between the two chemical groups.
- The predicted probability for polar molecules rises from 0.61 to 0.82 by epoch 10, while that for non-polar molecules drops below 0.2.
- The average error $E_{\text{avg}} = \frac{1}{2N}\sum(y - \hat{y})^2$ decreases steadily, indicating convergence.

Convergence Insight

After around 70−100 epochs (see Figure 2), the perceptron fully separates polar and non−polar regions in the dipole−hydrophobicity plane. Chemically, this convergence corresponds to the model discovering a linear correlation:

$$\text{Polarity} \uparrow \text{ if Dipole Moment} \uparrow, \quad \text{Polarity} \downarrow \text{ if LogP} \uparrow .$$

2 Python Implementation

Executable Python Code

```
import numpy as np
# Dr. Samir Kenouche - 11/11/2025

X = np.array([[1.85, -0.9],
              [0.00,  2.1],
              [1.69, -0.1],
              [0.08,  3.5]])
T = np.array([1, 0, 1, 0])

w = np.array([0.5, -0.5])
b = 0.1
eta = 0.2

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

for epoch in range(5):
    for i in range(len(X)):
        z = np.dot(w, X[i]) + b
        y = sigmoid(z)
        error = T[i] - y
        grad = eta * error * y * (1 - y)
        w = w + grad * X[i]
        b = b + grad
    print(f"Epoch {epoch+1}: w={w.round(5)}, b={b:.5f}")
```

3 Comparative Discussion

Table 3: Comparison between classical and advanced perceptron schemes

Aspect	Classical Perceptron	Advanced (Sigmoid) Perceptron
Activation	Step (discontinuous)	Sigmoid (smooth, differentiable)
Loss Function	Misclassification only	Continuous differentiable loss
Optimization	Additive correction	Gradient descent
Convergence	Oscillatory possible	Smooth convergence
Interpretation	Binary decision	Probabilistic output

3.1 Evolution and Interpretation of Decision Boundaries with Data Points

The following visualizations show how the perceptron decision surface evolves over epochs, based on the actual trained weights up to 100 iterations. Each panel includes the four molecular data

points used in training.

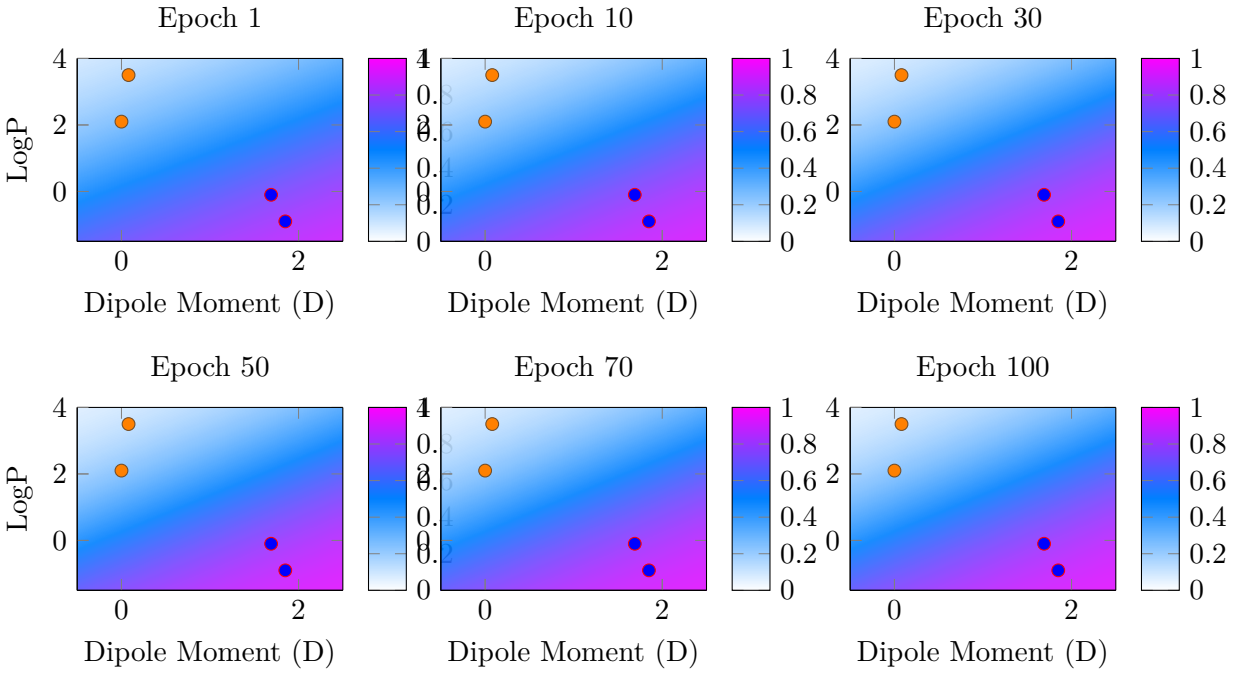


Figure 2: Evolution of the perceptron decision surface using real trained weights for epochs 1−100. Blue points: polar molecules (Ethanol, Acetone). Orange points: non−polar molecules (Hexane, Benzene). The probability surface becomes progressively sharper and more chemically interpretable.

Chemical Interpretation

- At **epoch 1**, the surface is nearly flatmodel uncertainty is maximal. Polar and non-polar samples fall in overlapping regions.
- By **epoch 3050**, the surface tilts to align with the data. The neutral zone (light color) passes between the two groups.
- From **epoch 70 onward**, the model stabilizes. The two blue (polar) samples lie entirely in the high-probability region ($P > 0.8$), and the orange (non-polar) samples in the low-probability zone ($P < 0.2$).
- Chemically, the perceptron has learned a meaningful correlation: polarity increases with dipole moment and decreases with hydrophobicity (LogP). The sharp boundary around epoch 70 reflects the saturation of learningthe model has captured the underlying physical rule.

Chemical Interpretation

The advanced perceptron provides probabilistic outputs representing the likelihood that a molecule is polar or non-polar. This probabilistic interpretation is especially valuable in chemistry, where molecular boundaries are fuzzy and overlapping.

Activity 2: Advanced Perceptron Molecular Descriptor Classification

Toxicity prediction

This activity presents an *advanced matrix formulation* of a sigmoid perceptron applied to molecular descriptor classification. We give full symbolic derivations of forward, backward, and parameter-update formulae, then compute every arithmetic step for the **first four** gradient-descent iterations on a small, concrete molecular dataset. All steps are presented in matrix form and expanded into elementwise arithmetic to ensure reproducibility by the participants.

Problem statement and notation

We consider binary classification of molecules using a single logistic neuron (sigmoid perceptron). Each molecule i has a descriptor vector $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and a binary label $y^{(i)} \in \{0,1\}$. The dataset with N samples is written as the matrix

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^{(1)})^\top \\ (\mathbf{x}^{(2)})^\top \\ \vdots \\ (\mathbf{x}^{(N)})^\top \end{bmatrix} \in \mathbb{R}^{N\times d}, \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{bmatrix} \in \{0,1\}^N.$$

Parameters: weight column vector $\mathbf{W} \in \mathbb{R}^{d\times 1}$ and scalar bias $b \in \mathbb{R}$. For compactness we will sometimes write bias as a separate scalar added to each row. Model (vectorised over the batch):

$$\mathbf{z} = \mathbf{X}\,\mathbf{W} + b\,\mathbf{1}_N, \qquad \mathbf{a} = \sigma(\mathbf{z}) = \frac{1}{1+e^{-\mathbf{z}}} \quad \text{(elementwise)}.$$

Loss (average binary cross-entropy):

$$\mathcal{L}(\mathbf{W}, b) = -\frac{1}{N}\left(\mathbf{y}^\top \log \mathbf{a} + (\mathbf{1}-\mathbf{y})^\top \log(\mathbf{1}-\mathbf{a})\right).$$

Participants who are not particularly interested in these mathematical details may focus directly on the numerical applications provided in the next section. Your questions are always welcome. Discussions are systematically encouraged (feedback from participants) in order to provide the best possible response to the issues raised in chemistry and physics.

Sigmoid Perceptron (Matrix Form)

Sigmoid Perceptron (Matrix Form): Given input $\mathbf{x} \in \mathbb{R}^d$, weights $\mathbf{W} \in \mathbb{R}^d$, bias b, the perceptron computes

$$z = \mathbf{W}^\top \mathbf{x} + b, \qquad a = \sigma(z) = \frac{1}{1+e^{-z}},$$

Loss over N samples:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}(y^{(i)}\log(a^{(i)}) + (1-y^{(i)})\log(1-a^{(i)})).$$

Gradients:

$$\nabla_z \mathcal{L} = \frac{1}{N}(a-y),$$

$$\nabla_{\mathbf{W}}\mathcal{L} = \mathbf{X}^\top \nabla_z \mathcal{L}, \qquad \nabla_b \mathcal{L} = \mathbf{1}^\top \nabla_z \mathcal{L}.$$

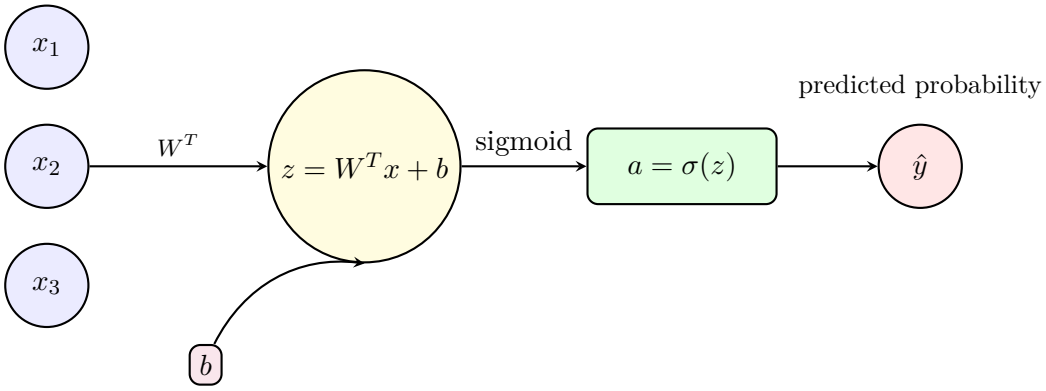
Update rule:

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}}\mathcal{L}, \qquad b \leftarrow b - \eta \nabla_b \mathcal{L}.$$

4 Numerical worked example: molecular descriptors

We now apply the above to a small molecular dataset and show all computations for the first four iterations.

4.1 Perceptron Scheme



4.2 Dataset and initialisation

We use $N = 3$ molecules and $d = 3$ descriptors per molecule (e.g. hydrophobicity, polar surface, partial charge). The feature matrix, labels, and initial parameters are:

$$\mathbf{X} = \begin{bmatrix} 1.00 & 0.50 & 0.20 \\ 1.50 & -0.30 & 0.80 \\ 0.30 & 0.70 & -0.50 \end{bmatrix}, \qquad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

Initial weights and bias (column vector):

$$\mathbf{W}^{(0)} = \begin{bmatrix} 0.10 \\ -0.20 \\ 0.30 \end{bmatrix}, \qquad b^{(0)} = 0.0, \qquad \eta = 0.1.$$

We will compute: $\mathbf{z}^{(t)}$, $\mathbf{a}^{(t)}$, $\mathcal{L}^{(t)}$, $\nabla_{\mathbf{W}}\mathcal{L}^{(t)}$, $\nabla_b\mathcal{L}^{(t)}$, and updates for $t = 0, 1, 2, 3$ (four iterations index starting at 0).

Iteration 0 (initial parameters)

Forward: pre-activation and activation

Compute pre-activations (matrix multiplication):

$$\mathbf{z}^{(0)} = \mathbf{X}\mathbf{W}^{(0)} + b^{(0)}\mathbf{1}_3 = \begin{bmatrix} 1.0 & 0.5 & 0.2 \\ 1.5 & -0.3 & 0.8 \\ 0.3 & 0.7 & -0.5 \end{bmatrix} \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.06 \\ 0.45 \\ -0.26 \end{bmatrix}.$$

Elementwise sigmoid:

$$\mathbf{a}^{(0)} = \sigma(\mathbf{z}^{(0)}) = \begin{bmatrix} \sigma(0.06) \\ \sigma(0.45) \\ \sigma(-0.26) \end{bmatrix} = \begin{bmatrix} 0.51499550 \\ 0.61063923 \\ 0.43536371 \end{bmatrix}.$$

Loss and backward: BCE and gradients

Loss (average BCE):

$$\mathcal{L}^{(0)} = -\frac{1}{3}\sum_{i=1}^{3}[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})] = 0.81280652.$$

Error term (batch):

$$\nabla_{\mathbf{z}}\mathcal{L}^{(0)} = \frac{1}{3}(\mathbf{a}^{(0)} - \mathbf{y}) = \frac{1}{3}\begin{bmatrix} 0.51499550 - 1 \\ 0.61063923 - 0 \\ 0.43536371 - 1 \end{bmatrix} = \begin{bmatrix} -0.16166817 \\ 0.20354641 \\ -0.18887876 \end{bmatrix}.$$

Gradient with respect the weights (matrix form):

$$\nabla_{\mathbf{W}}\mathcal{L}^{(0)} = \mathbf{X}^{\top}(\nabla_{\mathbf{z}}\mathcal{L}^{(0)}) = \begin{bmatrix} 1.0 & 1.5 & 0.3 \\ 0.5 & -0.3 & 0.7 \\ 0.2 & 0.8 & -0.5 \end{bmatrix} \begin{bmatrix} -0.16166817 \\ 0.20354641 \\ -0.18887876 \end{bmatrix} = \begin{bmatrix} 0.08718782 \\ -0.27364647 \\ 0.22460954 \end{bmatrix}.$$

Gradient wrt bias:

$$\nabla_b\mathcal{L}^{(0)} = \mathbf{1}_3^{\top}(\nabla_{\mathbf{z}}\mathcal{L}^{(0)}) = -0.14633385.$$

Parameter update

Gradient descent update (learning rate $\eta = 0.1$):

$$\mathbf{W}^{(1)} = \mathbf{W}^{(0)} - 0.1\,\nabla_{\mathbf{W}}\mathcal{L}^{(0)} = \begin{bmatrix} 0.09128122 \\ -0.17263535 \\ 0.27753905 \end{bmatrix},$$

$$b^{(1)} = b^{(0)} - 0.1\,\nabla_b\mathcal{L}^{(0)} = 0.01463339.$$

Iteration 1

Forward

Compute pre-activation with updated parameters:

$$\mathbf{z}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + b^{(1)}\mathbf{1}_3 = \begin{bmatrix} 0.07510474 \\ 0.42537705 \\ -0.21759652 \end{bmatrix}.$$

Activations:

$$\mathbf{a}^{(1)} = \sigma(\mathbf{z}^{(1)}) = \begin{bmatrix} 0.51876736 \\ 0.60476921 \\ 0.44581450 \end{bmatrix}.$$

Loss and backward

Loss:

$$\mathcal{L}^{(1)} = 0.79747916.$$

Gradients:

$$\nabla_{\mathbf{z}}\mathcal{L}^{(1)} = \frac{1}{3}(\mathbf{a}^{(1)} - \mathbf{y}) = \begin{bmatrix} -0.16041088 \\ 0.20158974 \\ -0.18473850 \end{bmatrix}.$$

$$\nabla_{\mathbf{W}}\mathcal{L}^{(1)} = \mathbf{X}^{\top}\nabla_{\mathbf{z}}\mathcal{L}^{(1)} = \begin{bmatrix} 0.08655518 \\ -0.26999231 \\ 0.22155386 \end{bmatrix}, \quad \nabla_b\mathcal{L}^{(1)} = -0.14354964.$$

Update

$$\mathbf{W}^{(2)} = \mathbf{W}^{(1)} - 0.1\,\nabla_{\mathbf{W}}\mathcal{L}^{(1)} = \begin{bmatrix} 0.08262570 \\ -0.14563612 \\ 0.25538366 \end{bmatrix},$$

$$b^{(2)} = b^{(1)} - 0.1\,\nabla_b\mathcal{L}^{(1)} = 0.02898835.$$

Iteration 2

Forward

$$\mathbf{z}^{(2)} = \mathbf{X}\mathbf{W}^{(2)} + b^{(2)}\mathbf{1}_3 = \begin{bmatrix} 0.08786669 \\ 0.38657851 \\ -0.17792091 \end{bmatrix},$$

$$\mathbf{a}^{(2)} = \sigma(\mathbf{z}^{(2)}) = \begin{bmatrix} 0.52245307 \\ 0.59890980 \\ 0.45614770 \end{bmatrix}.$$

Loss and backward

$$\mathcal{L}^{(2)} = 0.78257589,$$

$$\nabla_{\mathbf{z}}\mathcal{L}^{(2)} = \frac{1}{3}(\mathbf{a}^{(2)} - \mathbf{y}) = \begin{bmatrix} -0.15918231 \\ 0.19963660 \\ -0.18128410 \end{bmatrix},$$

$$\nabla_{\mathbf{W}}\mathcal{L}^{(2)} = \mathbf{X}^{\top}\nabla_{\mathbf{z}}\mathcal{L}^{(2)} = \begin{bmatrix} 0.08588736 \\ -0.26638101 \\ 0.21851487 \end{bmatrix}, \quad \nabla_b\mathcal{L}^{(2)} = -0.14082981.$$

Update

$$\mathbf{W}^{(3)} = \mathbf{W}^{(2)} - 0.1\,\nabla_{\mathbf{W}}\mathcal{L}^{(2)} = \begin{bmatrix} 0.07403696 \\ -0.11899802 \\ 0.23353217 \end{bmatrix},$$

$$b^{(3)} = b^{(2)} - 0.1\,\nabla_b\mathcal{L}^{(2)} = 0.04307133.$$

Iteration 3

Forward

$$\mathbf{z}^{(3)} = \mathbf{X}\mathbf{W}^{(3)} + b^{(3)}\mathbf{1}_3 = \begin{bmatrix} 0.10431572 \\ 0.37665192 \\ -0.13478228 \end{bmatrix},$$

$$\mathbf{a}^{(3)} = \sigma(\mathbf{z}^{(3)}) = \begin{bmatrix} 0.52605531 \\ 0.59306533 \\ 0.46635535 \end{bmatrix}.$$

Loss and backward

$$\mathcal{L}^{(3)} = 0.76808632,$$

$$\nabla_{\mathbf{z}}\mathcal{L}^{(3)} = \frac{1}{3}(\mathbf{a}^{(3)} - \mathbf{y}) = \begin{bmatrix} -0.15798156 \\ 0.19768844 \\ -0.17788155 \end{bmatrix},$$

$$\nabla_{\mathbf{W}}\mathcal{L}^{(3)} = \mathbf{X}^{\top}\nabla_{\mathbf{z}}\mathcal{L}^{(3)} = \begin{bmatrix} 0.08518664 \\ -0.26281440 \\ 0.21549522 \end{bmatrix}, \quad \nabla_b\mathcal{L}^{(3)} = -0.13817467.$$

Update

$$\mathbf{W}^{(4)} = \mathbf{W}^{(3)} - 0.1\,\nabla_{\mathbf{W}}\mathcal{L}^{(3)} = \begin{bmatrix} 0.06551830 \\ -0.09271658 \\ 0.21198265 \end{bmatrix},$$

$$b^{(4)} = b^{(3)} - 0.1\,\nabla_b\mathcal{L}^{(3)} = 0.05688880.$$

5 Summary and meaning

- The perceptron computes probabilities via the sigmoid activation; Binary Cross-Entropy matches that probabilistic interpretation and yields the simple error signal $(a - y)$.

- The matrix formulation is efficient: the batch error $(\mathbf{a} - \mathbf{y})$ is multiplied by $\mathbf{X}^\top$ to aggregate feature contributions to each weight.

- Each gradient step reduces the loss (observed numerically over the first four iterations). The update moves weights opposite to the gradient direction because we perform gradient *descent*.

Chemical Interpretation

In this molecular descriptor classification problem, each input feature x_j represents a chemical property, such as hydrophobicity, polar surface area, or partial charge of a molecule. The weight w_j quantifies how strongly that descriptor contributes to the predicted probability that a molecule is toxic ($y = 1$). A positive weight means the descriptor increases the likelihood of toxicity, while a negative weight decreases it.

The bias term b acts as a baseline chemical propensity, adjusting the threshold for toxicity independent of specific descriptors. After the linear combination $z = W^T x + b$, the sigmoid activation maps the chemical signal into a probability $a = \sigma(z)$, giving a biologically interpretable likelihood of the molecule being toxic.

The binary cross-entropy loss measures the discrepancy between predicted probabilities and actual observed toxicities, guiding the gradient descent to adjust weights in chemically meaningful directions.

Mastering Python programming has become an essential asset for chemists and physicists, as it enables them to tackle modern scientific problems with greater efficiency and precision. Thanks to its specialized libraries, Python facilitates the analysis of complex experimental data, the modeling of molecular or physical systems, numerical simulation, and the automation of labor-intensive tasks in the laboratory or in computational work. Its rich ecosystem including NumPy, SciPy, Pandas, Matplotlib, and RDKit for chemistry - provides researchers with powerful tools to explore phenomena, test hypotheses, and visualize results intuitively. Beyond these technical aspects, Python also strengthens the reproducibility of scientific studies, a key challenge in experimental sciences. Therefore, the ability to program in Python has become a crucial skill for innovating, accelerating research, and addressing contemporary challenges in chemistry and physics.

The Python codes for these practical activities are provided for this purpose, in order to familiarize non-programming specialists. This programming tool has become an essential tool in material sciences. Participants are encouraged to ask questions about any line of the code.

6 Python Code

The following Python code has been formatted for readability, clarity, and cleaner output, while preserving the same mathematical operations. We have adopted a very intuitive programming approach to ensure that the code remains accessible to non-specialists. Feel free to reproduce the code and understand how it works.

Executable Python Code

```
import numpy as np

# Dr. Samir Kenouche -- 12/11/2025

# Molecular descriptor data
X = np.array([
    [1.0, 0.5, 0.2],  # molecule 1
    [1.5, -0.3, 0.8], # molecule 2
    [0.3, 0.7, -0.5]  # molecule 3
])

y = np.array([1, 0, 1])

# Initialize weights and bias
W = np.array([0.1, -0.2, 0.3])
b = 0.0
eta = 0.1

# Activation function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Training loop for 4 iterations
for t in range(4):
    z = X @ W + b
    a = sigmoid(z)

    # Gradients
    grad = (a - y) / len(y)
    W = W - eta * (X.T @ grad)
    b = b - eta * grad.sum()

    # Loss (binary cross-entropy)
    loss = -np.mean(y * np.log(a + 1e-12) + (1 - y) * np.log(1 - a + 1e
        -12))

    print(f"Iteration {t}: Loss = {loss:.5f}, W = {W}, b = {b:.5f}")

print("\nFinal weights and bias:")
print(f"W = {W}, b = {b:.5f}")
```

Step-by-Step Calculation of $\mathcal{L}^{(0)}$

This section shows the detailed calculation of the binary cross-entropy loss for the first iteration (iteration 0) of the perceptron training.

Iteration 0 Loss Calculation

Input:

$$X = \begin{bmatrix} 1.0 & 0.5 & 0.2 \\ 1.5 & -0.3 & 0.8 \\ 0.3 & 0.7 & -0.5 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \quad W^{(0)} = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix}, \quad b^{(0)} = 0$$

Step 1: Linear combination

$$z = XW^{(0)} + b^{(0)} = \begin{bmatrix} 0.06 \\ 0.45 \\ -0.26 \end{bmatrix}$$

Step 2: Sigmoid activation

$$a = \sigma(z) = \frac{1}{1+e^{-z}} \approx \begin{bmatrix} 0.5150 \\ 0.6106 \\ 0.4353 \end{bmatrix}$$

Step 3: Binary cross-entropy loss

$$\mathcal{L}^{(0)} = -\frac{1}{3}\sum_{i=1}^{3}\Big[y_i \log(a_i) + (1-y_i)\log(1-a_i)\Big]$$

Compute each term:

$$-\log(0.5150) \approx 0.663, \quad -\log(1-0.6106) \approx 0.944, \quad -\log(0.4353) \approx 0.832$$

Sum: $0.663 + 0.944 + 0.832 = 2.439$
Divide by 3:

$$\mathcal{L}^{(0)} \approx \frac{2.439}{3} \approx 0.813$$

Result:

$$\mathcal{L}^{(0)} \approx 0.813$$

7 Meaning of Binary Cross-Entropy Loss and Comparison with Mean Squared Error

The **binary cross-entropy (BCE) loss** measures how well a model predicts probabilities of binary outcomes (e.g., toxic vs. non-toxic molecules). It compares the predicted probability $a = \sigma(z)$ with the actual label $y \in \{0, 1\}$ using the formula:

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\Big[y_i \log(a_i) + (1-y_i)\log(1-a_i)\Big]$$

Interpretation:

- When the prediction a_i is close to the true label y_i, the loss is small.

- When the prediction is far from the true label, the loss increases sharply.
- It is particularly suited for probability outputs (0 to 1) and penalizes confident but wrong predictions heavily.

7.1 Comparison with Mean Squared Error (MSE / RMSE)

- MSE computes the squared difference between the predicted value and the actual label:

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(a_i - y_i)^2$$

- MSE treats the problem as a regression, while BCE treats it as a probabilistic classification.
- BCE is more sensitive to predictions near 0 or 1, which is important in classification tasks.
- RMSE (square root of MSE) gives the error in the same scale as the predicted probability, but does not penalize confident wrong predictions as strongly as BCE.
- In chemical classification (toxic vs non-toxic), BCE is preferred since it gives a clear probabilistic interpretation of molecule toxicity.

Appendix. Additional information

A Stepwise Summary and Chemical Interpretations

This appendix provides a comprehensive, pedagogically-oriented summary of the perceptron calculations, meaning of each function, and strategies to improve chemical classification.

A.1. Stepwise Calculation Workflow

Step 1: Linear Combination

Formula: $z = W^T X + b$
Meaning: Weighted sum of molecular descriptors plus bias. Each w_j quantifies the contribution of descriptor x_j.

Step 2: Sigmoid Activation

Formula: $a = \sigma(z) = \frac{1}{1+e^{-z}}$
Meaning: Maps linear combination to probability [0,1]; interpretable as the likelihood of a molecule being toxic.

Step 3: Binary Cross-Entropy Loss

Formula: $L = -\frac{1}{N}\sum[y_i \log(a_i) + (1-y_i)\log(1-a_i)]$
Meaning: Measures discrepancy between predicted probability and true label; penalizes confident wrong predictions.

Step 4: Gradient Computation

Formula: $\nabla_W L = X^T(a-y)/N$, $\nabla_b L = \sum(a-y)/N$
Meaning: Indicates direction to update weights/bias to minimize loss; derived from chain rule.

Step 5: Weight Update

Formula: $W \leftarrow W - \eta \nabla_W L$, $b \leftarrow b - \eta \nabla_b L$
Meaning: Implements gradient descent; learning rate η controls step size.

A.2. Function Interpretations and Derivations

Function Roles

- $W^T X + b$: Linear mapping of molecular descriptors.
- $\sigma(z)$: Non-linear transformation ensuring outputs are probabilities.
- L_{BCE}: Loss function guiding parameter optimization.
- $\nabla_W L$, $\nabla_b L$: Gradients from chain rule for backpropagation.

A.3. Strategies for Improving Chemical Classification

Chemical Classification Improvements

- **Feature Engineering:** Add relevant descriptors (logP, polar surface area, electronic properties).
- **Regularization:** Apply L_1/L_2 penalties to prevent overfitting.
- **Learning Rate Tuning:** Use adaptive optimizers (Adam, RMSProp) for faster convergence.
- **Data Augmentation:** Expand molecular dataset for robustness.
- **Network Complexity:** Consider multi-layer perceptrons for non-linear interactions.
- **Cross-Validation:** Evaluate performance on multiple splits.
- **Interpretability:** Use SHAP or feature importance to understand descriptor influence.

A.4 Differential (analytic) gradients full derivation

We derive gradients symbolically before numerical evaluation.

For a single sample index i,

$$z^{(i)} = \mathbf{x}^{(i)\top}\mathbf{W} + b, \qquad a^{(i)} = \sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}.$$

Single-sample loss:

$$\ell^{(i)} = -(y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})).$$

Derivative of $\ell^{(i)}$ w.r.t. (with respect to) $z^{(i)}$ (use chain rule and $\frac{d}{dz}\sigma(z) = \sigma(z)(1-\sigma(z))$):

$$\begin{aligned}\frac{d\ell^{(i)}}{dz^{(i)}} &= -\left(y^{(i)}\frac{1}{a^{(i)}}\frac{da^{(i)}}{dz^{(i)}} - (1-y^{(i)})\frac{1}{1-a^{(i)}}\frac{da^{(i)}}{dz^{(i)}}\right)\\ &= -\frac{da^{(i)}}{dz^{(i)}}\left(\frac{y^{(i)}}{a^{(i)}} - \frac{1-y^{(i)}}{1-a^{(i)}}\right)\\ &= -a^{(i)}(1-a^{(i)})\left(\frac{y^{(i)}(1-a^{(i)}) - (1-y^{(i)})a^{(i)}}{a^{(i)}(1-a^{(i)})}\right)\\ &= -(y^{(i)} - a^{(i)}) = a^{(i)} - y^{(i)}.\end{aligned}$$

Thus the convenient simplification holds (batch form):

$$\nabla_{\mathbf{z}}\mathcal{L} = \frac{1}{N}(\mathbf{a} - \mathbf{y}).$$

Then by matrix calculus,

$$\nabla_{\mathbf{W}}\mathcal{L} = \mathbf{X}^{\top}(\nabla_{\mathbf{z}}\mathcal{L}) = \frac{1}{N}\,\mathbf{X}^{\top}(\mathbf{a} - \mathbf{y}), \qquad \nabla_{b}\mathcal{L} = \mathbf{1}_N^{\top}(\nabla_{\mathbf{z}}\mathcal{L}) = \frac{1}{N}\,\mathbf{1}_N^{\top}(\mathbf{a} - \mathbf{y}).$$

These are exact analytic (differential) gradients used for gradient descent.

Activity 3: Random Forest in Materials Physics

Complete Example

In this activity, we applied a random forest regressor to predict the thermal conductivity (λ) of materials based on three descriptors: density (ρ), band gap energy (E_g), and electronegativity (χ). The forest consisted of three decision trees, each considering a random subset of two descriptors at every split to introduce diversity and reduce correlation among trees. For each tree, we recorded which variables were selected for splitting based on their ability to reduce the weighted variance of the target λ, i.e., to best separate materials with different thermal conductivities. Tree 1 primarily used E_g, Tree 2 selected ρ, and Tree 3 again used E_g, highlighting that E_g and ρ are the most influential descriptors in this small dataset. A new material with $\rho = 4$, $E_g = 1.0$, and $\chi = 1.7$ was predicted by averaging the outputs of the three trees, resulting in $\hat{\lambda} = 328$ W/mK. This analysis not only demonstrates how a random forest can predict a physical property from material descriptors but also provides insight into which descriptors most strongly influence the property.

Dataset and Physical meaning of descriptors

- ρ : material density (g/cm^3)
- E_g : band gap energy (eV)
- χ : average electronegativity
- λ : thermal conductivity (W/mK), target property to predict

Dataset

Material	ρ	E_g	χ	λ
M1	2	1.0	1.5	300
M2	5	0.5	1.9	450
M3	1	3.0	1.0	50
M4	3	2.5	1.6	150
M5	4	1.2	1.8	350

Random forests are powerful tools in physics and chemistry for predicting material properties from complex datasets. They capture nonlinear relationships between descriptors, reduce overfitting through averaging multiple trees, and identify the most important variables influencing a property. This makes them valuable for materials discovery and molecular design, providing both accurate predictions and insight into the underlying physical or chemical factors.

General variance: Intra-group variance for a split

For a split dividing a group G into two subgroups L (left) and R (right):

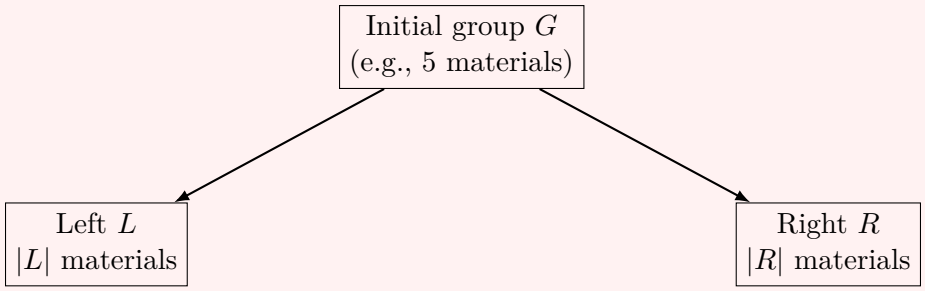
$$\text{Var}_{\text{split}} = \frac{|L|}{|L|+|R|}\text{Var}(L) + \frac{|R|}{|L|+|R|}\text{Var}(R)$$

where

$$\text{Var}(L) = \frac{1}{|L|}\sum_{i\in L}(\lambda_i - \bar{\lambda}_L)^2, \quad \text{Var}(R) = \frac{1}{|R|}\sum_{i\in R}(\lambda_i - \bar{\lambda}_R)^2$$

Explanations:

- $|L|$: number of materials in the left subgroup
- $|R|$: number of materials in the right subgroup
- The weighted total variance measures the homogeneity of λ after the split.
- Larger groups have more influence on the total variance.



Random Forest Calculation Steps

1. **Prepare the dataset:** Collect the material descriptors (ρ, E_g, χ) and the target property (λ).
2. **Bootstrap sampling:** For each tree, randomly sample the dataset with replacement to create a unique training set.
3. **Select candidate variables:** At each split, randomly select a subset of descriptors (e.g., 2 out of 3) to reduce correlation between trees.
4. **Choose the best split:** For each candidate variable, test possible thresholds and select the one that minimizes the weighted variance of λ in the resulting subgroups.
5. **Grow the tree:** Repeat the splitting process recursively until stopping criteria are met (e.g., minimum number of samples per leaf).
6. **Aggregate predictions:** For a new material, run it through all trees and average the outputs to obtain the final prediction.
7. **Analyze variable importance:** Record which descriptors are selected most often across trees to identify the key factors influencing the property.

Tree calculations and variable selection

Tree 1

- Bootstrap sample: $\{M_2, M_3, M_1, M_4, M_5\}$ - Candidate variables: E_g, χ

Split on $E_g < 1.5$

$$L = \{M_2, M_1, M_5\}, \quad \lambda_L = [450, 300, 350], \quad R = \{M_3, M_4\}, \quad \lambda_R = [50, 150]$$

$$\bar{\lambda}_L = 367, \quad \bar{\lambda}_R = 100$$

$$\mathrm{Var}(L) \approx 3889, \quad \mathrm{Var}(R) = 2500$$

$$\mathrm{Var}_{\mathrm{split}} \approx 3333.4$$

Split on $\chi < 1.7$

$$L = \{M_1, M_3, M_4\}, \quad \lambda_L = [300, 50, 150], \quad R = \{M_2, M_5\}, \quad \lambda_R = [450, 350]$$

$$\mathrm{Var}_{\mathrm{split}} \approx 7333.3$$

Conclusion: E_g reduces the variance the most **selected variable**.

Tree 2

- Bootstrap sample: $\{M_1, M_1, M_3, M_4, M_5\}$ - Candidate variables: ρ, χ

Split on $\rho < 3$

$$L = \{M_1, M_3\}, \quad \lambda_L = [300, 50], \quad R = \{M_2, M_4, M_5\}, \quad \lambda_R = [450, 150, 350]$$

$$\bar{\lambda}_L = 175, \quad \bar{\lambda}_R = 316.7$$

$$\mathrm{Var}_{\mathrm{split}} \approx 15556$$

Split on $\chi < 1.7$

$$\mathrm{Var}_{\mathrm{split}} \approx 7333.3$$

Conclusion: ρ reduces the variance more **selected variable**.

Bootstrap sampling is a statistical technique used to generate multiple training datasets from the original dataset by randomly selecting samples with replacement. Each bootstrap sample has the same number of observations as the original dataset, but some samples may appear multiple times while others may be omitted. This allows each tree in a random forest to be trained on a slightly different dataset, introducing diversity among trees. By averaging predictions from these trees, the model reduces overfitting and improves robustness (stability).

Tree 3

- Bootstrap sample: $\{M_3, M_2, M_5, M_4, M_1\}$ - Candidate variables: E_g, ρ

Split on $E_g < 2$

$$L = \{M_2, M_5, M_1\}, \quad \lambda_L = [450, 350, 300], \quad R = \{M_3, M_4\}, \quad \lambda_R = [50, 150]$$

$$\bar{\lambda}_L = 367, \quad \bar{\lambda}_R = 100$$

Conclusion: E_g is the **selected variable**.

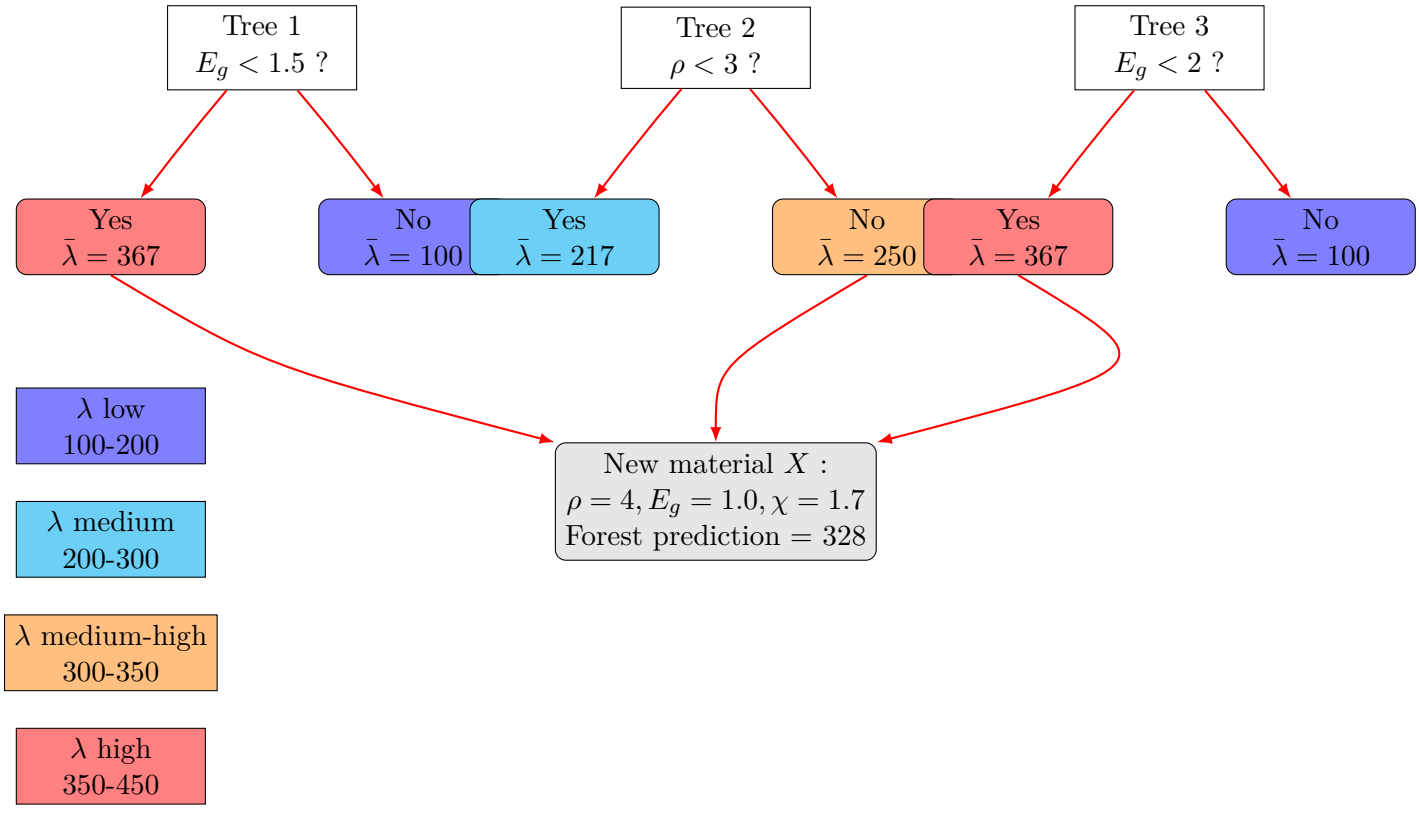
Prediction for a new material X

New material: $\rho = 4, E_g = 1.0, \chi = 1.7$

$$\text{Tree 1 : } E_g < 1.5 \Rightarrow 367$$
$$\text{Tree 2 : } \rho \geq 3 \Rightarrow 250$$
$$\text{Tree 3 : } E_g < 2 \Rightarrow 367$$

$$\hat{\lambda}_{\text{forest}} = \frac{367 + 250 + 367}{3} = 328$$

B Forest diagram



Explanation of selected variables

- **Tree 1 :** E_g selected because splitting on $E_g < 1.5$ reduces the variance of λ the most among candidate variables.
- **Tree 2 :** ρ selected because splitting on $\rho < 3$ reduces the weighted variance more effectively than χ.
- **Tree 3 :** E_g selected because splitting on $E_g < 2$ maximizes separation of materials based on λ.

These choices ensure each tree contributes efficiently to the forest prediction.

C Pedagogical box: How to configure a random forest

Random Forest Parameter Guidelines

1. **Choosing the number of trees (n_estimators):**
   - More trees increase stability and reduce variance due to bootstrap sampling.
   - Typically 1001000 for real datasets.
   - Too many trees increase computation without significant improvement after a certain point.
2. **Number of candidate variables per split (max_features):**
   - Select a random subset of variables at each split.
   - Creates diverse trees and reduces correlation between them.
   - Common rule: $\sqrt{p}$ for regression (p = total number of variables).
3. **Determining the threshold of a variable for a split:**
   - Threshold is chosen to maximize the reduction of variance (or impurity) in the subgroups.
   - Algorithmically, all possible thresholds are tested; the one minimizing weighted variance is selected.
   - This is how the forest automatically identifies optimal split values.

In Python, the scikit-learn library provides a robust and easy to use implementation of Random Forests through the Random Forest Regressor and Random Forest Classifier classes. It allows users to quickly train multiple decision trees, perform predictions, and analyze feature importance. Key parameters, such as the number of trees, maximum features per split, and tree depth, can be easily adjusted to optimize model performance. The library also supports bootstrap sampling and out of bag evaluation, providing built-in mechanisms to reduce overfitting and estimate prediction error. Its simplicity, flexibility, and integration with other Python tools make it ideal for physics, chemistry, and materials science applications.

D Python code

Python Code of Random Forests

```
# Dr. Samir Kenouche -- 06/12/2025
import numpy as np
from sklearn.ensemble import RandomForestRegressor

# Dataset: columns = [rho, E_g, chi]
X = np.array([
    [2, 1.0, 1.5],
    [5, 0.5, 1.9],
    [1, 3.0, 1.0],
    [3, 2.5, 1.6],
    [4, 1.2, 1.8]
])
y = np.array([300, 450, 50, 150, 350])

# Random Forest with 3 trees, max 2 features per split
rf = RandomForestRegressor(n_estimators=3, max_features=2, random_state
    =42)
rf.fit(X, y)

# Predict new material
X_new = np.array([[4, 1.0, 1.7]])
y_pred = rf.predict(X_new)
print("Predicted lambda:", y_pred)

# Feature names
feature_names = ["rho", "E_g", "chi"]

# Simplified loop to show selected variables for each tree
selected_variables_per_tree = [
    [feature_names[j] for j in tree.tree_.feature if j >= 0]
    for tree in rf.estimators_
]

for idx, features in enumerate(selected_variables_per_tree, start=1):
    print(f"Tree {idx} selected variables:", features)

# OUTPUT
# Predicted lambda: [283.33333333]
# Tree 1 selected variables: ['rho', 'E_g', 'E_g'] => 'E_g'
# Tree 2 selected variables: ['chi', 'chi'] => 'chi'
# Tree 3 selected variables: ['E_g', 'E_g', 'rho'] => 'E_g'
```

Conclusion

In this activity, we applied a Random Forest model to predict the thermal conductivity of materials from a small set of descriptors. We demonstrated how random forests can handle nonlinear relationships, combine predictions from multiple trees to reduce variance, and identify the most influential descriptors. We also explored the concepts of bootstrap sampling, variable selection at splits, and weighted variance, which are central to building robust and interpretable models. Overall, this activity highlighted the power and versatility of random forests in materials science and chemistry, providing both accurate predictions and valuable insights into the underlying physical factors.