1) Logistic Regression (binary)

Model & linear algebra setup

- Data matrix $X \in R^{n \times d}$ (rows $x_i^\top$), labels $y \in \{0,1\}^n$.
- Parameters $w \in R^d, b \in R$. Let $z = X w + b 1$.
- Sigmoid $\sigma(t) = \frac{1}{1+e^{-t}}$. Predicted probs $p = \sigma(z)$ elementwise: $p_i = P(y_i = 1 \vee x_i)$.

Likelihood → loss

- Bernoulli log-likelihood:

$$\ell(w,b) = \sum_{i=1}^{n} \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right].$$

- Minimize negative log-likelihood (cross-entropy):

$$L(w,b) = -\ell(w,b) = -\sum_{i=1}^{n} \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right].$$

Gradients (calculus)

Key derivatives: $\frac{d}{dt} \sigma(t) = \sigma(t)(1 - \sigma(t))$. For a single example, $z_i = w^\top x_i + b, p_i = \sigma(z_i)$:

$$\frac{\partial L}{\partial w} = \sum_{i=1}^{n} (p_i - y_i) x_i, \frac{\partial L}{\partial b} = \sum_{i=1}^{n} (p_i - y_i).$$

Matrix form:

$$\nabla_w L = X^\top (p - y), \frac{\partial L}{\partial b} = 1^\top (p - y).$$

Hessian & convexity (linear algebra)

Let $S = diag(p_i(1 - p_i))$. Then

$$\nabla_w^2 L = X^\top S X \succcurlyeq 0, \nabla_{wb}^2 L = X^\top S 1, \nabla_{bb}^2 L = 1^\top S 1.$$

Since $S \succcurlyeq 0$, the loss is convex → unique global minimizer (modulo separability issues).

Optimization

- Gradient descent: $w \leftarrow w - \eta X^\top (p - y)$, $b \leftarrow b - \eta 1^\top (p - y)$.
- Newton/IRLS: use Hessian $X^\top S X$ for second-order updates; fast convergence.

Regularization

- L2 (ridge): $L_\lambda = L + \frac{\lambda}{2} \| w \|_2^2$ → gradient adds λw; keeps convexity, improves generalization.
- L1 (lasso): $L_\lambda = L + \lambda \| w \|_1$ → subgradient $\lambda\, sign(w)$; promotes sparsity.
- Elastic net: combination.

Decision boundary & interpretation

- Boundary: $w^\top x + b = 0$.
- Log-odds are linear: $\log \frac{p}{1-p} = w^\top x + b$. Each weight is a change in log-odds per unit of its feature.

Multiclass (softmax)

- Scores: $s_k(x) = w_k^\top x + b_k$, $k = 1..K$.
- $P(y = k \vee x) = \frac{e^{s_k}}{\sum_j e^{s_j}}$.
- Loss: $-\sum_i \log P(y_i \vee x_i)$. Gradient for class k: $\nabla_{w_k} L = \sum_i (p_{ik} - 1\{y_i = k\}) x_i$. Convex in the parameters $\{w_k\}$.

---

2) Linear Regression *as a classifier* (why it's problematic but instructive)

Setup

- Treat $y \in \{0, 1\}$ and fit least squares: minimize $\| X w - y \|_2^2$.
- Normal equations (assuming full rank): $w = (X^\top X)^{-1} X^\top y$.

Math & issues

- Predicted "probabilities" $ \hat{y}=Xw$ are unconstrained → can be <0 or >1.
- Loss is quadratic, solution is closed-form, but misaligned with Bernoulli likelihood; decision boundary still linear by thresholding (e.g., 0.5), yet calibration is poor and class imbalance can hurt. This motivates logistic loss (proper for Bernoulli).

3) Support Vector Machine (SVM) Classifier — Math

We'll start with the **hard margin** case, then relax to **soft margin**.

Step 1 – Problem Setup

- Data: $X \in R^{n \times d}$ (rows $x_i^\top$), labels $y_i \in \{-1, +1\}$.
- Goal: Find a hyperplane $w^\top x + b = 0$ that maximizes the margin between classes.

Step 2 – Margin definition

For a given hyperplane, the **geometric margin** for (x_i, y_i) is:

$$\gamma_i = y_i \frac{w^\top x_i + b}{\| w \|}.$$

We want the smallest margin (over all i) to be **as large as possible**.

Step 3 – Optimization formulation (Hard Margin)

We scale w, b so that the **support vectors** satisfy:

$$y_i (w^\top x_i + b) = 1.$$

Then the optimization becomes:

$$\max_{\gamma, w, b} \gamma \text{ s.t. } y_i (w^\top x_i + b) \geq 1 \, \forall i.$$

Maximizing γ is equivalent to minimizing $\frac{1}{2} \| w \|^2$:

$$\min_{w,b} \frac{1}{2} \| w \|^2 \text{s.t. } y_i (w^\top x_i + b) \geq 1.$$

Step 4 – Lagrangian (Linear Algebra + Calculus)

Introduce Lagrange multipliers $\alpha_i \geq 0$ for constraints:

$$L(w, b, \alpha) = \frac{1}{2} \| w \|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i (w^\top x_i + b) - 1 \right].$$

From KKT conditions:

1. **Gradient w.r.t.** w:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

1. **Gradient w.r.t. b**:

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Step 5 – Dual Problem

Substitute w back into the Lagrangian to eliminate w:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(x_i^{\top} x_j \right),$$

subject to:

$$\alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0.$$

This is a **quadratic programming** problem in α.

Step 6 – Prediction

Given $\alpha^{\text{¿}}$ and $w^{\text{¿}}$, the decision function is:

$$f(x) = \operatorname{sign}\left(\sum_{i \in \mathrm{SV}} \alpha_i^{\text{¿}} y_i \left(x_i^{\top} x \right) + b^{\text{¿}} \right).$$

Only **support vectors** ($\alpha_i > 0$) contribute to w.

Step 7 – Soft Margin (Hinge Loss)

Allow some misclassification with slack variables $\xi_i \geq 0$:

$$\min_{w,b,\xi} \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} \xi_i \text{ s.t. } y_i \left(w^{\top} x_i + b \right) \geq 1 - \xi_i.$$

Dual form just bounds α_i:

$$0 \leq \alpha_i \leq C.$$

Step 8 – Kernel Trick

Replace $x_i^{\top} x_j$ in the dual with a kernel $K\left(x_i, x_j\right) = \phi\left(x_i\right)^{\top} \phi\left(x_j\right)$ without computing ϕ explicitly. Common kernels:

- Linear: $K(x,z)=x^\top z$
- Polynomial: $K(x,z)=(x^\top z+c)^p$
- RBF: $K(x,z)=\exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right)$

Step 9 – Geometric intuition

- The margin width = $\frac{2}{\|w\|}$.
- Support vectors lie exactly at distance $\frac{1}{\|w\|}$ from the boundary.
- C balances margin maximization and misclassification tolerance.

 Summary Math Links:

- Linear algebra: w is a weighted sum of support vectors.
- Calculus: Lagrangian derivatives give primal-dual connection.
- Optimization: Quadratic program, convex → unique global optimum.

3) Support Vector Machine (SVM) Classifier — Math

We'll start with the **hard margin** case, then relax to **soft margin**.

Step 1 – Problem Setup

- Data: $X \in R^{n\times d}$ (rows $x_i^\top$), labels $y_i \in \{-1,+1\}$.
- Goal: Find a hyperplane $w^\top x+b=0$ that maximizes the margin between classes.

Step 2 – Margin definition

For a given hyperplane, the **geometric margin** for (x_i, y_i) is:

$$\gamma_i=y_i\frac{w^\top x_i+b}{\|w\|}.$$

We want the smallest margin (over all i) to be **as large as possible**.

Step 3 – Optimization formulation (Hard Margin)

We scale w, b so that the **support vectors** satisfy:

$$y_i\left(w^\top x_i + b\right) = 1.$$

Then the optimization becomes:

$$\max_{\gamma, w, b} \gamma \text{ s.t. } y_i\left(w^\top x_i + b\right) \geq 1 \; \forall i.$$

Maximizing γ is equivalent to minimizing $\frac{1}{2} \| w \|^2$:

$$\min_{w,b} \frac{1}{2} \| w \|^2 \text{ s.t. } y_i\left(w^\top x_i + b\right) \geq 1.$$

Step 4 – Lagrangian (Linear Algebra + Calculus)

Introduce Lagrange multipliers $\alpha_i \geq 0$ for constraints:

$$L(w, b, \alpha) = \frac{1}{2} \| w \|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i\left(w^\top x_i + b\right) - 1 \right].$$

From KKT conditions:

1. **Gradient w.r.t.** w:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

1. **Gradient w.r.t.** b:

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Step 5 – Dual Problem

Substitute w back into the Lagrangian to eliminate w:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(x_i^\top x_j\right),$$

subject to:

$$\alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0.$$

This is a **quadratic programming** problem in α.

Step 6 – Prediction

Given $\alpha^{¿}$ and $w^{¿}$, the decision function is:

$$f(x) = \text{sign}\left(\sum_{i \in \text{SV}} \alpha_i^{¿} y_i (x_i^\top x) + b^{¿}\right).$$

Only **support vectors** ($\alpha_i > 0$) contribute to w.

Step 7 – Soft Margin (Hinge Loss)

Allow some misclassification with slack variables $\xi_i \geq 0$:

$$\min_{w,b,\xi} \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} \xi_i \text{ s.t. } y_i (w^\top x_i + b) \geq 1 - \xi_i.$$

Dual form just bounds α_i:

$$0 \leq \alpha_i \leq C.$$

Step 8 – Kernel Trick

Replace $x_i^\top x_j$ in the dual with a kernel $K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$ without computing ϕ explicitly. Common kernels:

- Linear: $K(x, z) = x^\top z$
- Polynomial: $K(x, z) = (x^\top z + c)^p$
- RBF: $K(x, z) = \exp\left(-\frac{\| x - z \|^2}{2\sigma^2}\right)$

Step 9 – Geometric intuition

- The margin width = $\frac{2}{\| w \|}$.
- Support vectors lie exactly at distance $\frac{1}{\| w \|}$ from the boundary.
- C balances margin maximization and misclassification tolerance.

Summary Math Links:

- Linear algebra: w is a weighted sum of support vectors.
- Calculus: Lagrangian derivatives give primal-dual connection.

- Optimization: Quadratic program, convex → unique global optimum.

3) Support Vector Machine (SVM) Classifier — Math

We'll start with the **hard margin** case, then relax to **soft margin**.

Step 1 – Problem Setup

- Data: $X \in R^{n \times d}$ (rows $x_i^\top$), labels $y_i \in \{-1,+1\}$.
- Goal: Find a hyperplane $w^\top x+b=0$ that maximizes the margin between classes.

Step 2 – Margin definition

For a given hyperplane, the **geometric margin** for (x_i, y_i) is:

$$\gamma_i = y_i \frac{w^\top x_i + b}{\| w \|}.$$

We want the smallest margin (over all i) to be **as large as possible**.

Step 3 – Optimization formulation (Hard Margin)

We scale w, b so that the **support vectors** satisfy:

$$y_i \left(w^\top x_i + b \right) = 1.$$

Then the optimization becomes:

$$\max_{\gamma, w, b} \gamma \text{ s.t. } y_i \left(w^\top x_i + b \right) \geq 1 \, \forall i.$$

Maximizing γ is equivalent to minimizing $\frac{1}{2} \| w \|^2$:

$$\min_{w,b} \frac{1}{2} \| w \|^2 \text{ s.t. } y_i \left(w^\top x_i + b \right) \geq 1.$$

Step 4 – Lagrangian (Linear Algebra + Calculus)

Introduce Lagrange multipliers $\alpha_i \geq 0$ for constraints:

$$L(w, b, \alpha) = \frac{1}{2} \| w \|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i \left(w^\top x_i + b \right) - 1 \right].$$

From KKT conditions:

1. **Gradient w.r.t.** w:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i .$$

1. **Gradient w.r.t.** b:

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0 .$$

Step 5 – Dual Problem

Substitute w back into the Lagrangian to eliminate w:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(x_i^{\top} x_j \right),$$

subject to:

$$\alpha_i \geq 0 , \sum_{i=1}^{n} \alpha_i y_i = 0 .$$

This is a **quadratic programming** problem in α.

Step 6 – Prediction

Given $\alpha^{¿}$ and $w^{¿}$, the decision function is:

$$f(x) = \operatorname{sign}\left(\sum_{i \in \mathrm{SV}} \alpha_i^{¿} y_i \left(x_i^{\top} x \right) + b^{¿} \right).$$

Only **support vectors** ($\alpha_i > 0$) contribute to w.

Step 7 – Soft Margin (Hinge Loss)

Allow some misclassification with slack variables $\xi_i \geq 0$:

$$\min_{w,b,\xi} \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} \xi_i \text{ s.t. } y_i \left(w^{\top} x_i + b \right) \geq 1 - \xi_i .$$

Dual form just bounds α_i:

$$0 \leq \alpha_i \leq C .$$

Step 8 – Kernel Trick

Replace $x_i^\top x_j$ in the dual with a kernel $K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$ without computing ϕ explicitly. Common kernels:

- Linear: $K(x, z) = x^\top z$
- Polynomial: $K(x, z) = (x^\top z + c)^p$
- RBF: $K(x, z) = \exp\left(-\frac{\| x - z \|^2}{2\sigma^2}\right)$

Step 9 – Geometric intuition

- The margin width = $\frac{2}{\| w \|}$.
- Support vectors lie exactly at distance $\frac{1}{\| w \|}$ from the boundary.
- C balances margin maximization and misclassification tolerance.

□ **Summary Math Links**:

- Linear algebra: w is a weighted sum of support vectors.
- Calculus: Lagrangian derivatives give primal-dual connection.
- Optimization: Quadratic program, convex → unique global optimum.

6) Naive Bayes Classifier — Math

Step 1 – Bayes' Theorem refresher

For class C_k and input x:

$$P(C_k \vee x) = \frac{P(x \vee C_k) \cdot P(C_k)}{P(x)}$$

- $P(C_k)$ = prior probability of class C_k
- $P(x \vee C_k)$ = likelihood of data given the class
- $P(x)$ = evidence (same for all classes, so often ignored in argmax)

Classification rule:

$$\hat{y} = \arg\max_k P(C_k) \cdot P(x \vee C_k)$$

Step 2 – “Naive” assumption

Features $x=(x_1,x_2,\ldots,x_d)$ are **conditionally independent given the class**:

$$P(x \vee C_k)=\prod_{j=1}^{d} P(x_j \vee C_k)$$

This is rarely true in reality, but works surprisingly well.

Step 3 – Model types

The math for $P(x_j \vee C_k)$ changes depending on feature type:

a) **Gaussian Naive Bayes** (continuous features):

If $x_j \vee C_k \sim N(\mu_{jk},\sigma_{jk}^2)$:

$$P(x_j \vee C_k)=\frac{1}{\sqrt{2\pi\sigma_{jk}^2}}\exp\left(-\frac{(x_j-\mu_{jk})^2}{2\sigma_{jk}^2}\right)$$

b) **Multinomial Naive Bayes** (counts, e.g., text):

If feature x_j is a count of word j:

$$P(x \vee C_k)=\frac{\left(\sum_j x_j\right)!}{\prod_j x_j!}\prod_{j=1}^{d} p_{jk}^{x_j}$$

where p_{jk} is probability of word j in class k.

c) **Bernoulli Naive Bayes** (binary features):

$$P(x_j \vee C_k)=p_{jk}^{x_j}(1-p_{jk})^{1-x_j}$$

Step 4 – Log space trick

Multiplying many probabilities can cause **underflow**. We use logs:

$$\log P(C_k \vee x) \propto \log P(C_k)+\sum_{j=1}^{d}\log P(x_j \vee C_k)$$

Since $\log$ is monotonic, argmax is preserved.

Step 5 – Training math

To train:

1. Estimate $P\left(C_k\right)=\frac{\text{count of class}\,k}{N}$
2. Estimate $P\left(x_j \vee C_k\right)$ based on chosen distribution (Gaussian mean/variance, word frequency, etc.)
3. Store these parameters.

No gradient descent — it's all closed-form from counts & averages.

Step 6 – Decision rule

Final prediction:

$$\hat{y}=\underset{k}{arg\,max}\left[\log P\left(C_k\right)+\sum_{j=1}^{d}\log P\left(x_j \vee C_k\right)\right)$$

Math recap:

- **Bayes' theorem** + independence assumption
- **Product of likelihoods** → log-sum
- **Closed-form parameter estimation** from data
- Works even with small datasets

6) Naive Bayes Classifier — Math

Step 1 – Bayes' Theorem refresher

For class C_k and input x:

$$P\left(C_k \vee x\right)=\frac{P\left(x \vee C_k\right)\cdot P\left(C_k\right)}{P\left(x\right)}$$

- $P\left(C_k\right)$ = prior probability of class C_k
- $P\left(x \vee C_k\right)$ = likelihood of data given the class
- $P\left(x\right)$ = evidence (same for all classes, so often ignored in argmax)

Classification rule:

$$\hat{y} = arg\,\underset{k}{max}\,P\left(C_k\right) \cdot P\left(x \vee C_k\right)$$

Step 2 – "Naive" assumption

Features $x = \left(x_1, x_2, \ldots, x_d\right)$ are **conditionally independent given the class**:

$$P\left(x \vee C_k\right) = \prod_{j=1}^{d} P\left(x_j \vee C_k\right)$$

This is rarely true in reality, but works surprisingly well.

Step 3 – Model types

The math for $P\left(x_j \vee C_k\right)$ changes depending on feature type:

a) **Gaussian Naive Bayes** (continuous features):

If $x_j \vee C_k \sim N\left(\mu_{jk}, \sigma_{jk}^2\right)$:

$$P\left(x_j \vee C_k\right) = \frac{1}{\sqrt{2\pi\sigma_{jk}^2}} \exp\left(-\frac{\left(x_j - \mu_{jk}\right)^2}{2\sigma_{jk}^2}\right)$$

b) **Multinomial Naive Bayes** (counts, e.g., text):

If feature x_j is a count of word j:

$$P\left(x \vee C_k\right) = \frac{\left(\sum_j x_j\right)!}{\prod_j x_j!} \prod_{j=1}^{d} p_{jk}^{x_j}$$

where p_{jk} is probability of word j in class k.

c) **Bernoulli Naive Bayes** (binary features):

$$P\left(x_j \vee C_k\right) = p_{jk}^{x_j}\left(1 - p_{jk}\right)^{1 - x_j}$$

Step 4 – Log space trick

Multiplying many probabilities can cause **underflow**. We use logs:

$$\log P\left(C_k \vee x\right) \propto \log P\left(C_k\right) + \sum_{j=1}^{d} \log P\left(x_j \vee C_k\right)$$

Since $\log$ is monotonic, argmax is preserved.

Step 5 – Training math

To train:

1. Estimate $P(C_k) = \frac{\text{count of class } k}{N}$
2. Estimate $P(x_j \vee C_k)$ based on chosen distribution (Gaussian mean/variance, word frequency, etc.)
3. Store these parameters.

No gradient descent — it's all closed-form from counts & averages.

Step 6 – Decision rule

Final prediction:

$$\hat{y} = \arg\max_k \left[\log P(C_k) + \sum_{j=1}^{d} \log P(x_j \vee C_k) \right)$$

 Math recap:

- **Bayes' theorem** + independence assumption
- **Product of likelihoods** → log-sum
- **Closed-form parameter estimation** from data
- Works even with small datasets

9) Confusion Matrix — Structure

For **binary classification** (positive vs. negative):

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

10) Metrics — Formulas

1. **Accuracy** Measures the proportion of correct predictions:

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

2. **Precision** (Positive Predictive Value) Out of all predicted positives, how many are correct?

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

3. **Recall** (Sensitivity, True Positive Rate) Out of all actual positives, how many did we catch?

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

4. **F1 Score** Harmonic mean of Precision & Recall:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

 Key intuition:

- Precision cares about *quality* of positives predicted.
- Recall cares about *quantity* of positives caught.
- F1 balances both — good when classes are imbalanced.