Performance

		Actual: Pos	Actual: Neg
Predicted	Pos	True Pos (TP)	False Pos (FP)
	Neg	False Neg (FN)	True Neg (TN)

Accuracy: fraction of correct prediction, $\frac{\text{TP+TN}}{\text{TP+FP+TN+FN}}$.
Precision: accuracy of the positive predictions: $\frac{\text{TP}}{\text{TP+FP}}$.
recall/sensitivity is the accuracy for the pos class: $\frac{\text{TP}}{\text{TP+FN}}$.

Linear Regression

Univariate linear function: $\hat{y} = \hat{f}_{\theta_0,\theta_1}(x) = \theta_0 + \theta_1 x$
n features and m samples: $\hat{Y} = \hat{f}_\Theta(\mathbf{X}) = \mathbf{X}\Theta$

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \cdots & x_n^{(m)} \end{pmatrix}, \quad \Theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}$$

aim: find the optimal Θ s.t. close the actual f
Residual Sum of Squares (RSS):

$$J(\Theta) = \sum_{i=1}^{m} (\hat{f}_\Theta(X^{(i)}) - y^{(i)})^2 = \|\hat{f}_\Theta(\mathbf{X}) - Y\|_2^2.$$

goal: minimize RSS $\rightarrow$ ordinary least squares

$$\begin{cases} \theta_1 &= \frac{\sum_{i=1}^m (x^{(i)} - \overline{x})(y^{(i)} - \overline{y})}{\sum_{i=1}^m (x^{(i)} - \overline{x})^2}, \\ \theta_0 &= \overline{y} - \theta_1 \overline{x}. \end{cases}$$

OR (for all): $\hat{\Theta} = \Theta^\star = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T Y.$

$$\begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix}^T = \begin{pmatrix} a & c & e \\ b & d & f \end{pmatrix}, \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Shrinkage Method: $y = f(X) + \epsilon$, $\mathbb{E}(\epsilon) = 0, \text{Var}(\epsilon) = \sigma^2$
expected prediction error:
$\text{EPE}(X) = \mathbb{E}((y - \hat{f}(X))^2) = \text{Bias}(\hat{f}(X))^2 + \text{Var}(\hat{f}(X)) + \sigma^2$

$$\text{Bias}(\hat{f}) = \mathbb{E}(\hat{f}) - f, \text{Var}(\hat{f}) = \mathbb{E}((\hat{f} - \mathbb{E}[\hat{f}])^2)$$

low variance causes high bias, vice versa. over-fitting: low bias, high var; under-fitting: high bias, low var
Ridge Regression:

$$\hat{\Theta}^{ridge} = \arg\min_\Theta \|\mathbf{X}\Theta + \theta_0 - Y\|_2^2 + \lambda\|\Theta\|_2^2, \|\Theta\|_2^2 = \sum_{i=1}^n \theta_i^2$$

Lasso Regression

$$\hat{\Theta}^{lasso} = \arg\min_\Theta \|\mathbf{X}\Theta + \theta_0 - Y\|_2^2 + \lambda\|\Theta\|_1, \|\Theta\|_1 = \sum_{i=1}^n |\theta_i|$$

Logistic Regression

$\text{logit}(p) = \ln \text{odds}(p) = \ln\left(\frac{p}{1-p}\right), \text{logit}(p) \in (-\infty, +\infty)$
logistic sigmoid function: $\mathbb{P}(\hat{y} = 1 \mid X) = \sigma(X^T\Theta + \theta_0) = \text{logit}^{-1}(X^T\Theta + \theta_0) = \frac{1}{1+e^{-(X^T\Theta+\theta_0)}}$
$X^T\Theta \geq 0 \Rightarrow \hat{y} = 1 \quad \text{and} \quad X^T\Theta < 0 \Rightarrow \hat{y} = 0$
$p_i \overset{\text{def}}{=} \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)^{y^{(i)}} \left(1 - \mathbb{P}\left(\hat{y}^{(i)} = 1 \mid X^{(i)}\right)\right)^{1-y^{(i)}}$
$\nabla f(\Theta) = \frac{\partial f(\Theta)}{\partial \Theta} = \left(\frac{\partial f(\Theta)}{\partial \theta_1}, \cdots, \frac{\partial f(\Theta)}{\partial \theta_n}\right)$

Ensure: $\alpha > 0$
  Initialize $\Theta \leftarrow \Theta_0$ randomly
  while not converged **do**
    $\Theta \leftarrow \Theta - \alpha \nabla f(\Theta)$
  end while

Ensure: $\alpha > 0$
  Initialize Θ randomly
  while not converged **do**
    $\Theta \leftarrow \Theta - \alpha \sum_{i=1}^m \left(\frac{1}{1+e^{-X^{(i)T}\Theta}} - y^{(i)}\right) X^{(i)}$
  end while

SVM

hyperplane: $\mathbf{w}^T\mathbf{x} + b = 0$, distance: $\gamma = \frac{|\mathbf{w}^T\mathbf{x}_i + b|}{\|\mathbf{w}\|}$

$$y_i = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x}_i + b > 0; \\ -1, & \text{if } \mathbf{w}^T\mathbf{x}_i + b < 0; \end{cases} \Rightarrow |\mathbf{w}^T\mathbf{x}_i + b| = y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

distance: $\gamma = \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|}$; $y_i = \begin{cases} 1, & \text{if } \mathbf{w}^T\mathbf{x}_i + b \geq 1; \\ -1, & \text{if } \mathbf{w}^T\mathbf{x}_i + b \leq -1; \end{cases}$
at support vector $(\mathbf{x}_s, y_s)$, we have $y_s(\mathbf{w}^T\mathbf{x}_s + b) = 1$
margin: $\gamma = \frac{2}{\|\mathbf{w}\|}$
$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^m \alpha_i(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$

objective function: $\max_\alpha \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i\alpha_j y_i y_j \mathbf{x_i^T x_j}$

s.t. $\sum_{i=1}^m \alpha_i y_i = 0, \alpha_i \geq 0, i = 1, 2, \ldots, m$
optimal hyperplane

$$\begin{cases} \mathbf{w}^\star = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \\ b^\star = \frac{1}{|S|}\sum_{s\in S}(\frac{1}{y_s} - \mathbf{w^T x_s}) \end{cases} \Rightarrow \quad f(x) = \mathbf{w^T x} + b$$

KKT: if $y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 > 0 \rightarrow \alpha_i = 0$, not SV
if $y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 = 0 \rightarrow \alpha_i > 0$, SV

Kernel SVM

$f(\mathbf{x}) = \mathbf{w^T}\phi(\mathbf{x}) + b$, kernel function: $\kappa(\mathbf{x_i}, \mathbf{x_j}) = \phi(\mathbf{x_i})^T\phi(\mathbf{x_j})$

Clustering

Euclidean distance: $\text{dist}(a,b) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2}$
Manhattan distance: $\text{dist}(a,b) = \sum_{i=1}^n |a_i - b_i|$
Chebyshev distance: $\text{dist}(a,b) = \max_i(|a_i - b_i|)$
Minkowski distance: $\text{dist}(a,b) = (\sum_{i=1}^n |a_i - b_i|^p)^{\frac{1}{p}}$
Cosine similarity: $\text{dist}(a,b) = \cos(\theta) = \frac{a \cdot b}{\|a\| \cdot \|b\|}$
Intra-cluster cohesion (compactness): measures how near the data points in a cluster are to the cluster centroid
Inter-cluster speration (isolation): different cluster centroids should be far away from one another.

K-Means: Select K points as the initial centroids.
repeat: Form K clusters by assigning all points to the closest centroid. Recompute the centroid of each cluster
until fulfill the stopping condition
convergence: no (or minimum) reassignment of data points to different clusters, or no (or minimum) change of centroids, or minimum decrease in the sum of squared error (SSE)

$$\text{SSE} = \sum_{i=1}^K \sum_{x \in C_i} \text{dist}(x, \mu_i)^2$$

C_i: cluster μ_i: centroid, dist is Euclidean dist
Scale to large data sets, Guarantee convergence
needs to manually set hyper-parameter, empty cluster due to improper initialization, sensitive to outliers, non-globular shapes

Density-based clustering: Label data points into core, border and noise. Eliminate noise points. For every core point p that has not been assigned to a cluster, create a new cluster with the point p and all the points that are density-connected to p. Assign border points to the cluster of the closest core point.
ϵ: max radius, ϵ-neighbor: data pts within radius from a data pt
An object q is directly density-reachable from object p if p is a core object and q is in p's ϵ-neighborhood.

Distance matrix

min dist = dist(Cluster1, Cluster2) = min(dist(A, C), dist(A, D), dist(B, C), dist(B, D))
max dist = dist(Cluster 1, Cluster 2) = max(dist(A,C), dist(A,D), dist(B,C), dist(B,D))
Group average distance: dist(Cluster 1, Cluster 2) = (dist(A,C) + dist(A,D) + dist(B,C) + dist(B,D))/4
Distance between centroids = dist($\frac{A+B}{2}, \frac{C+D}{2}$)
closest pair: single linkage, farthest pair: complete linkage

CNN

Output size: $\frac{N-F}{\text{stride}} + 1$, N - input dimension, F - filter size

commom zero padding: $\frac{F-1}{2}$
no. of params: e.g. 8×8, stride 2, pad, $3 \to 8\times8\times3+1 = 193$

CNN training

Data augmentation for images at the input layer: load image and label

While training, dropout is implemented by only keeping a neuron active with some probability p

Transformers e.g. search for videos on Youtube, the search engine map your query (text in the search bar) against a set of keys (video title, description, etc.) associated with candidate videos in their database, then present you the best matched videos (values).

input sentence first converted into a sequence of tokens, each token is an 1D vector

Self-attention involves tokens in the input sequence attending to other tokens within the same input.

Give each token a KEY, QUERY, and VALUE representation

Uninformed Search

Single-state: complete world state and action knowledge

Multiple-state: incomplete world state / action knowledge

Contingency problem: not possible to define a complete sequence of actions because information about the intermediary states is unknown

Exploration problem: state space and effects of actions are unknown

search problem: state space, successor function, start state, a goal test

Completeness: strategy guaranteed to find a solution when there is one Optimality: the strategy find the optimal solution when there are several solutions

BFS: $\mathcal{O}(b^d), \mathcal{O}(b^d)$

Uniform-cost search: $\mathcal{O}(b^{\frac{C^\star}{\epsilon}}), \mathcal{O}(b^{\frac{C^\star}{\epsilon}})$

Instead of expanding the shallowest node, uninform-cost search expands the node with the lowest path cost

DFS: $\mathcal{O}(b^m), \mathcal{O}(bm)$

Depth-limited search: $\mathcal{O}(b^l), \mathcal{O}(bl)$

Iterative deepening depth-first search: $\mathcal{O}(b^d), \mathcal{O}(bd)$

Informed Search: Greedy best-first search

Informed search uses heuristics or extra knowledge to guide the search, while uninformed search has no such knowledge. Informed search is more efficient because it expands fewer nodes and reaches the goal faster, often yielding optimal solutions.

Heuristic function $h(n)$: cost estimated from current to goal evaluation function $f(n)$: desirability of expand

Greedy best-first search: $\mathcal{O}(b^m), \mathcal{O}(b^m)$

Informed Search: A*

$g(n)$: cost from start to reach n (Uniform-cost search)

$h(n)$: cheapest cost to get from n to goal (Greedy search)

$f(n)$: est. cost of cheapest sol through $n = g(n) + h(n)$

$\mathcal{O}(b^m), \mathcal{O}(b^m)$

Extra

A.

CNNs achieve higher accuracy than traditional methods

Automatically learn edges, textures, parts, objects end-to-end, instead of fixed hand-crafted features (e.g., SIFT/HOG).

Strong inductive biases: Local receptive fields and weight sharing capture spatial locality and translation equivariance, using far fewer parameters → better generalization.

Built-in invariances: Stride/pooling and data augmentation provide robustness to translation/scale/illumination.

End-to-end optimization at scale: Backprop on large datasets with GPUs allows very deep models and powerful regularization (BN, dropout), improving accuracy.

Residual (skip) connection / ResNet block. Form: y = ReLU(F(x) + x) where F(x) is the output of stacked weight layers (e.g., Conv–BN–ReLU–Conv–BN).

B.

Because you used ReLU instead of Sigmoid at the final layer, the output is not a probability:

ReLU outputs any value ≥ 0, with no upper bound.

Thresholding at 0.5 becomes meaningless because many non-dog images may still produce values ¿ 0.5 just due to positive activations.

You lose the probabilistic interpretation, making training unstable and misclassifications common.

Also, ReLU has zero gradient for z ¡ 0, causing dead neurons and preventing learning for negative samples.

Problem encountered: You cannot correctly perform binary classification because the final output is not a bounded probability, leading to incorrect thresholding and poor training behavior.

C.

The significant error decreases at epochs 25 and 60 are due to scheduled LR drops (multi-step LR schedule, e.g., gamma=0.1). Learning rate step decay. The LR is reduced sharply (e.g., multiplied by 0.1) at epoch 25 and again at epoch 60. This causes the characteristic step-wise drop in both training and validation error as optimization transitions to a finer step size and converges to a better minimum.

D.

$$\phi(x) = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x \\ (x-1)^2 \end{bmatrix}, \qquad t_i \in \{+1, -1\} \text{ are class labels.}$$

1. **Mapped coordinates and SVM decision boundary.**

$$x_1 = 0,\ t_1 = +1:\ \phi(x_1) = (0, 1)$$
$$x_2 = 1,\ t_2 = -1:\ \phi(x_2) = (1, 0)$$
$$x_3 = 3,\ t_3 = +1:\ \phi(x_3) = (3, 4)$$
$$x_4 = 4,\ t_4 = +1:\ \phi(x_4) = (4, 9)$$

These are linearly separable in the (u, v)-plane. A maximum–margin separator is

$$w = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \quad b = 0, \qquad \text{so } w^\top \phi(x) + b = 0 \Leftrightarrow v - u = 0 \Leftrightarrow \boxed{v = u}$$

In terms of the original variable,

$$v = u \Leftrightarrow (x-1)^2 - x = 0 \Leftrightarrow x^2 - 3x + 1 = 0.$$

2. **Support vectors.**

With the above (w, b):

$$w^\top \phi(x_1) + b = 1, \qquad t_1 = +1,$$
$$w^\top \phi(x_3) + b = 1, \qquad t_3 = +1,$$
$$w^\top \phi(x_2) + b = -1, \qquad t_2 = -1,$$
$$w^\top \phi(x_4) + b = 5.$$

Hence the support vectors are

$$\boxed{\text{Positive: } x_1, x_3 \quad \text{Negative: } x_2.}$$

(The margin width is $2/\|w\| = 2/\sqrt{2} = \sqrt{2}$; distance from each support vector to the boundary is $1/\sqrt{2}$.)

3. **Effect of adding a new negative point** $x_5 = 1.5$**.**

$$\phi(x_5) = (1.5, 0.25), w^\top \phi(x_5) + b = 0.25 - 1.5 = -1.25$$
$$t_5 = -1 \Rightarrow t_5\big(w^\top \phi(x_5) + b\big) = 1.25 > 1$$

Thus x_5 lies outside the margin and is not a support vector; it does *not* change the boundary or the margin.

E.

Use Lagrangian optimization with KKT conditions to convert the SVM primal to the dual form. Kernel SVM solves linear inseparability by using a kernel function to implicitly map data into a higher-dimensional space where a linear separator exists. Typical kernels include the RBF kernel and polynomial kernel. Kernel SVM is a supervised learning method.