أكاديمية كاوست
KAUST ACADEMY

Day 2: Machine Learning Algorithms Summary

Note: Do Not depend entirely on this and study from the official slides

Contributed by: Hassan Mohammed Nasr

1 - Linear Models (Supervised Learning)

- The foundation of Ml. Assume the output (y) is a linear combination of inputs (sum of products)
- Fast and efficient, but sensitive to outliers, underfit, and needs scaling

Linear regression (regression)

- Model
  - Predicts continuous output

$$\hat{y} = w_0 + w_1x_1 + w_2x_2 \ldots = w^T x$$

- loss
  - Mean Squared Error (MSE)
    (Heavily penalize outliers)

$$J = \frac{1}{N}\sum(y - \hat{y})^2$$

- Optimization
  - Normal Equation (closed form/ Analytical)

$$\omega = (x^T x)^{-1}x^T y$$

  - Gradient Descent(Iterative)

$$\omega_{new} = \omega_{old} - a\nabla J$$

Logistic regression (Classification)

- Model
  - Predicts Discrete output
  - Sigmoid function squashes linear output into probability [0,1]

$$z = w_0 + w_1x_1 + w_2x_2 \ldots = w^T x$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

- loss
  - Cross Entropy Loss (log loss)
    (Penalizes confident wrong predictions)

$$J = -\frac{1}{N}\sum[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

- Optimization
  - Gradient Descent(Iterative)

$$\omega_{new} = \omega_{old} - a\nabla J$$

Note: for multiclass use softmax

Regularization (Solution for overfitting)

- add penalty term to the loss to penalize large weights (overfitting) to keep them small

$$J_{total} = J_{mse} + \lambda R(w)$$

High lambda = huge penalty = tiny weights

Small lambda = small penalty = large weights = increase overfitting

	Lasso (L1)	Ridge (L2)
Penalty	Sum of Absolute Values $R(\omega) = \sum \lvert\omega\rvert$	Sum of squares $R(\omega) = \sum \omega^2$
Effect	Sparcity: forces many weights to be 0	Weight decay: shrinks weights smoothly toward zero but rarely equal 0
Geometric Shape	Diamond shaped	Circular shaped

2 - Distance based Models (Supervised Learning / non parametric)
K-Nearest Neighbor (K-NN)

- Uses Distance instead of learnable parameters
- Lazy Learner: memorizes data and make predictions based on local similarity

    1 - calculate distance between new point and all data.
    2 - pick the (K) closest neighbors
    3 - Classification = majority vote, Regression = Average

- Simple and no training, but memory intensive, slow inference, need scaling, and performance degrade in high dimensions

3 - Kernel based Models (Supervised Learning)
Support Vector Machine (SVM)

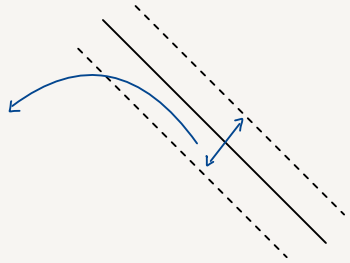
- logistic regression teams foundry that fits probabilities; however, sun chooses the boundary with the largest safety gap (margin)

Hyperplane (model)

$$\omega^T x + b = 0$$

Linear!!

Safely gap to closest points



Goal: find the hyperplane with the maximum margin (widest safety gap)

$$min \frac{1}{2} \parallel w \parallel^2$$

Subject to $y_i(w^T x + b) \geq 1$

Hart margin: problem is real data is noisy, 'hard margin' would crash if just one outlier existed.

Solution: soft margin, relax rules by allowing some violations but penalize them using a Regularization hyper parameter C

- Large C (Strict): penalize errors heavily = narrow margin = risk of overfitting
- Small C (More Violations): ignores small errors = wider margin = better generalization

Kernel Trick (Non Linear Separable)

- SVM finds linear boundary, so to fit non linear separable data we transform the data into higher dimensional space $x \rightarrow \phi(x)$

  ↓

  Computing it is computationally expensive

- Solution: use special type of functions called Kemal functions which compute the high-jim similarity directly without explicitly building them

Linear kernel

Finds straight lines
Good for linear problems

$$k(x, y) = x^T y$$

Polynomial kernel

Finds Curved Boundaries
and feature interactions

$$k(x, y) = (x^T y + c)^d$$

Radial Basis Function (RBF)

Finds local similarity
Points close in distance

$$K(x, y) = e^{-\gamma ||x-y||^2}$$

SVMs are powerfull modeling complex non-lineur relationships but not scalable, needs preprocessing and noise sensitive

4 - Tree based Models

1 - Decision Trees

- Split based on sequence questions Try every possible split for every feature (no gradient descent)

  ↳ Split on feature that maximizes information gain by reducing impurity.

  ⇓

  Impurity metrics
  - 1-Gini impurity. $G = 1 - \sum p_i^2$
  - 2 - Entropy. $H = -\sum \rho_i log_2(\rho_i)$

- Simple, easy to understand, interpretable, and does not need preprocessing. However, high variance and overfitting

  ↳ Solution (Bagging): train many trees on random subsets and average them (Random Forests)

Random forests are robust, high performance, und tells feature importance, however, slow inference, And memory intensive.
Moreover, doesn't learn from mistakes.

↳ Solution: Train trees sequentially where each tree fixes the errors of the previous one. → Gradient Boosting

- SOTA performance, Flexible performance and does not need preprocessing . However, sensitive to tuning, serial making it slow, easy to overfit especially when data is small

No model works best for every problem

- we must make assumptions about our data
- We must experiment with multiple models