Model Evaluation

Confusion Matrix – evaluates the performance of a classification model

		predicted: class = yes	predicted: class = no
actual	class = yes	a – true positive	b – false negative
	class = no	c – false positive	d – true negative

$$\text{accuracy} = \frac{TP + TN}{a+b+c+d}$$

→ measures the proportion of correct predictions
→ can be misleading in imbalanced datasets
  class 0: 9990 examples
  class 1: 10 examples
  if the model predicts everything as class 0
  9990 true negatives and 10 false negatives
  → will get high accuracy score because imbalanced but will fail to classify class 1.

Cost Matrix

		predicted: class = yes	predicted: class = no
actual	class = yes	C(Yes|Yes)	C(No|Yes)
	class = no	C(Yes|No)	C(No|No)

correctly diagnosed cancer → cost = 0

missed a cancer diagnosis very costly = 1000

assign how costly each prediction type is because not all mistakes are equal
→ we want to minimize total cost

cost of classification – higher accuracy does not always mean better performance especially when costs of error matter

Other metrics:

$$\text{Precision} = \frac{a}{a+c}$$

positive predictions (1st col)

of all the positive predictions, how many were actually correct?
→ high precision: few false positives
→ measures how reliable "Yes" prediction is

$$\text{Recall} = \frac{a}{a+b}$$

positive actuals (1st row)

of the actual positives, how many did we correctly predict?
→ high recall: few false negatives

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

→ harmonic mean of precision and recall
→ balances the trade-off between false positives and false negatives

Methods of estimation:

Goal: Get a reliable estimate of the performance of the model on unseen data
→ we care most about how will it performs on new, unseen data to be able to simulate real-world knowledge / generalize

1.) Holdout Method: split the dataset into training and testing
2.) Cross-validation:
  → partition the data into K disjoint subsets
  → K-fold: train on k-1 partitions in each subset, and test on the remaining
    → each column one round
  → Leave-One-Out: special case when K = n (number of data points)
    → each test set was only one observation
    → very accurate and tests on every single point
    → but slow for large datasets

Validation Set - for tuning parameters
we adjust our model based on how well it does on this set - but we don't train on it

Ensemble Methods
Suppose we have trained 17 classifiers on a dataset
→ with each having an error rate of $\epsilon = 0.20$ → since each has 0.20 → Binomial distribution
→ assume all classifiers are independent

To classify a new record, we poll all 17 classifiers and take the majority agrees on.
What is the IP that this ensemble classifier makes a wrong prediction?
→ how likely is it that more than half (≥9 classifiers) are wrong on the same example?
→ the majority needs to make a mistake

$$IP(X \geq 9) = \sum_{k=9}^{17} \binom{19}{k} (0.2)^k (1-0.2)^{17-k} = 0.00258$$

How do we generate independent classifiers?
To make an ensemble effective, we need classifiers that make different kinds of errors and are trained on different views / slices of data
→ generate diverse training sets using
1.) Bagging - Bootstrap Aggregating
→ create multiple of your training data by random sampling with replacement, then train a seperate classifier on each version
→ at prediction time, combine their outputs
2.) Boosting - focuses on mistakes to improve performance
→ start with all samples having equal probability of being selected
→ equal weight
→ identify which examples are misclassified
→ increase the weights of those samples that were misclassified
decreasing weight of those that were correctly classified
→ train a new classifier using the updated weights
→ combine all the classifiers to make final prediction
→ can overfit if not regularized well
Adaptive method that builds a strong classifier by sequentially training weak classifiers each one trying to correct the errors made by the previous ones
→ Boosting adapts the sampling distribution to give more attention to "difficult" samples

Classifiers trained on each sample are given a weight that is a function of their error rate
→ weighted vote at prediction time