Introdução à Aprendizagem Automática (IAA)

SUSANA BRÁS

SUSANA.BRAS@UA.PT

IAA – L6

Nonlinear SVM
- Kernel: definition, examples, application
- Gaussian RBF Kernel
- SVM hyper-parameters optimization
- SVM advantages and disadvantages
- Best practices

K-NN
- Selection of distance or similarity measure.
- Selection of K nearest neighbors
- Lazy learning algorithm
- Non-parametric learning algorithm
- Advantages, disadvantages

Classification performance evaluation
- Confusion matrix
- Accuracy, True Positive Rate, True Negative Rate, False Positive Rate, Precision, F1-score, Balanced Accuracy, ROC curve, AUC

Class Imbalance Problem
- Definition
- Implications
- Why is important
- Techniques for handling class imbalance
  - Data level: Undersampling the majority class, Oversampling the minority class
  - Metric level: Accuracy a biased metric, Consider metrics not biased
  - Classifier level: Penalize learning algorithms, Ensemble methods, One-Class classification

Epoch /Batch Size / Iterations / Train step

Data Matrix

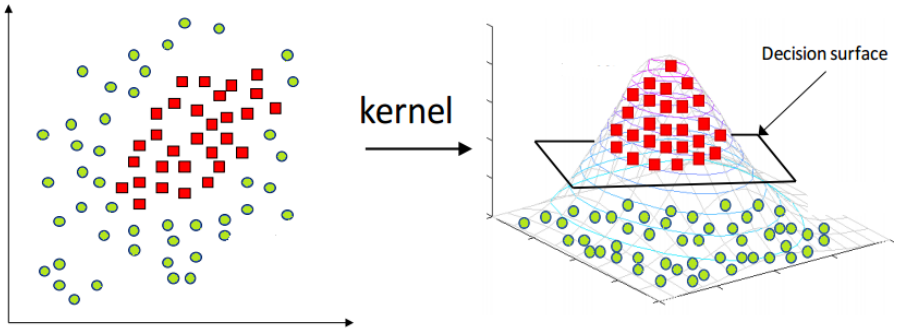
matrix X (mxn)	feature x_1	feature x_2	.....	feature x_n	Target Y
Example 1	$x_1^{(1)}$	$x_2^{(1)}$		$x_n^{(1)}$	$Y^{(1)}$
Example 2	$x_1^{(2)}$	$x_2^{(2)}$		$x_n^{(2)}$	$Y^{(2)}$
…					
Example i	$x_1^{(i)}$	$x_2^{(i)}$		$x_n^{(i)}$	$Y^{(i)}$
…					
…					
Example m	$x_1^{(m)}$	$x_2^{(m)}$		$x_n^{(m)}$	$Y^{(m)}$

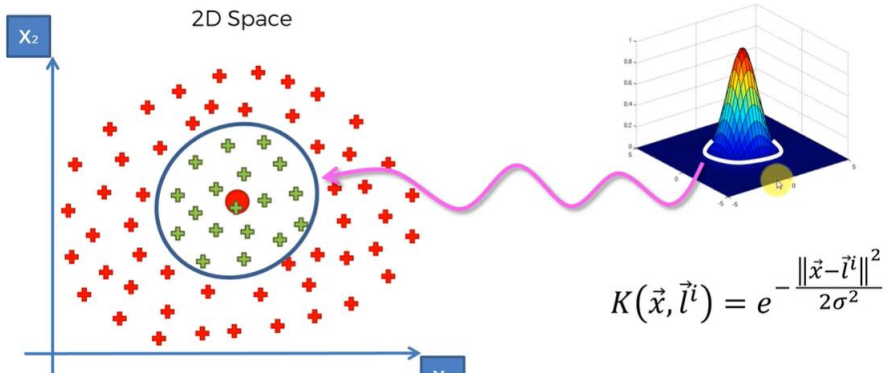
x – input vector of features, attributes
y – output vector of labels, ground truth, target
m - number of training examples
n – number of features
$h_\theta(x)$ - model (hypothesis)
θ - vector of model parameters

Training set: data matrix X (m rows, n columns)

Nonlinear SVM - Gaussian RBF Kernel

Nonlinearly separable data – kernel SVM





$$K(\vec{x}, \vec{l}^{i}) = e^{-\frac{\|\vec{x}-\vec{l}^{i}\|^2}{2\sigma^2}}$$

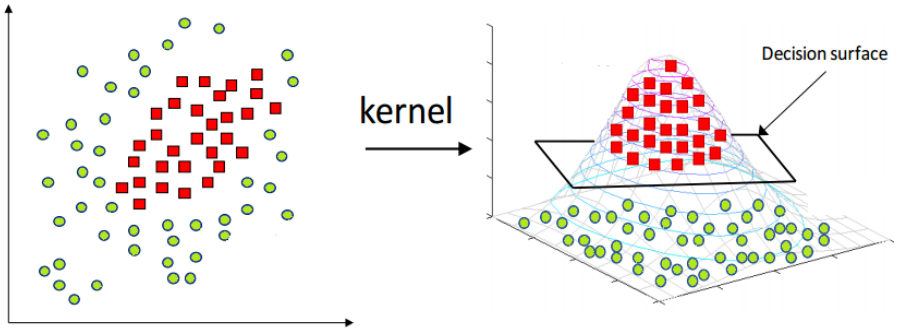
The key idea is that we don't need to explicitly compute the mapping to the higher-dimensional feature space.
The **kernel function** computes the similarity between data points in the higher-dimensional space without having to directly compute the coordinates of each point in that space. This allows SVMs to handle complex, non-linear relationships between features while maintaining computational efficiency

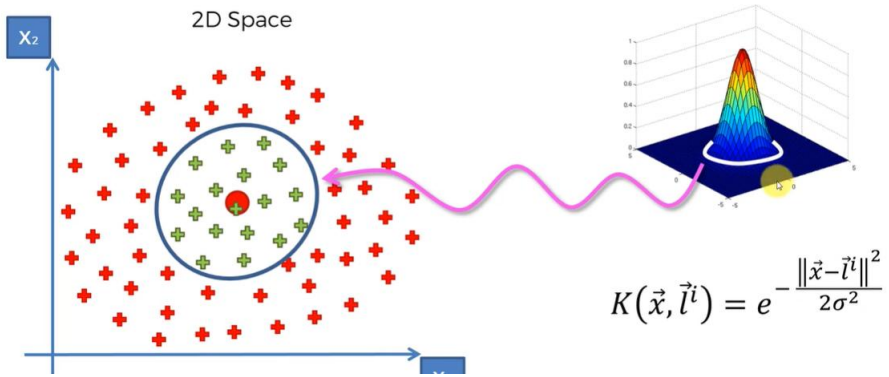
Kernel - A mathematical function that helps organize data and make it easier to classify

Typical Kernels:

- Polynomial Kernel - adding extra polynomial terms
- Gaussian Radial Basis Function (RBF) kernel – the most used kernel
- Laplace RBF kernel
- Hyperbolic tangent kernel
- Sigmoid kernel, etc.

Nonlinear SVM – Gaussian RBF Kernel





RBF kernel (proportional to Gaussian distribution) is a **metric of similarity** between examples, $x^{(i)}$ and $x^{(j)}$.

RBF kernel varies between:

- max value =1 (for $x^{(i)} = x^{(j)}$)
- tends to 0 when $x^{(i)}$ and $x^{(j)}$ go away of each other.

The goal: Substitute the original features with similarity features (kernels).

Note: the original (n+1 dimensional) feature vector is substituted by the new (n_new+1 dimensional) similarity feature vector.

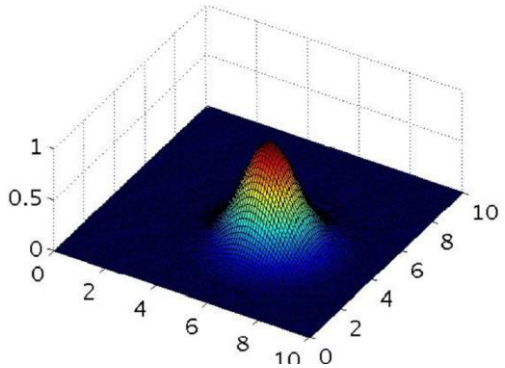
m –number of examples, m>>n !!!

Gaussian RBF Kernel – Parameter σ

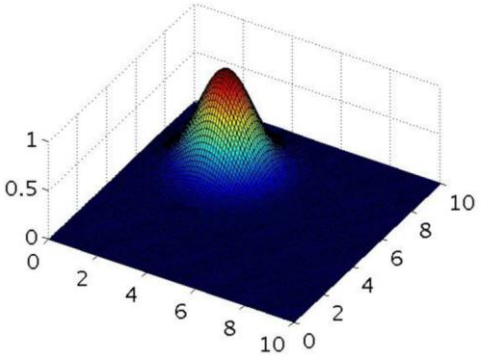
$$k(x_i, x_j) = e^{-\gamma \left\| x_i - x_j \right\|^2}, \quad \gamma \approx \frac{1}{\sigma^2} > 0$$

- σ determines how fast the similarity metric decreases to 0 as the examples go away of each other.

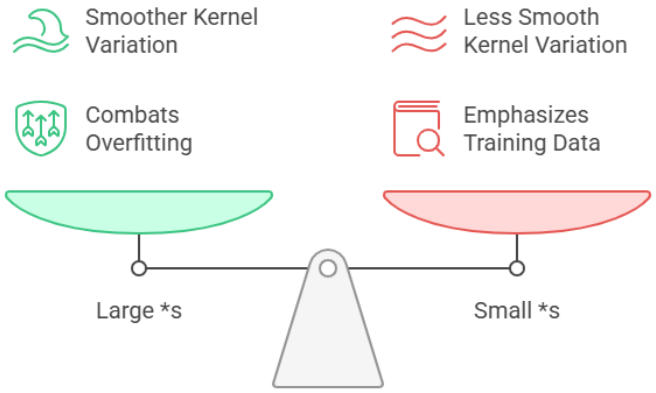
σ=1



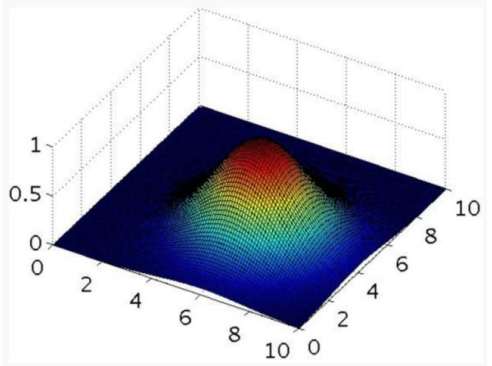
σ=0.5



Balancing Kernel Smoothness and Training Data Fitting



σ=1.5



SVM parameters

How to choose **hyper-parameter C**:

Large C: lower bias, high variance (equivalent to small regular. param. λ)

Small C: higher bias, lower variance (equivalent to large regular. param. λ)

How to choose **hyper-parameter** σ:

Large σ: features vary more smoothly. Higher bias, lower variance

Small σ: features vary less smoothly. Lower bias, higher variance

SVM Implementation

Python notes:

Use SVM software packages to solve SVM optimization.

Use Scikit-learn (**sklearn**) machine learning library

Import **SVC** (Support Vector Classification):

from sklearn.svm import SVC

*classifier = SVC(**kernel="?",gamma =?, C=?**)*

Properly define parameters:

"rbf" (Radial Basis Function) corresponds to the Gaussian kernel.

gamma = $1/\sigma$

SVM math explained: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

https://datamites.com/blog/support-vector-machine-algorithm-svm-understanding-kernel-trick/#:~:text=A%20Kernel%20Trick%20is%20a,Lagrangian%20formula%20using%20Lagrangian%20multipliers.%20

SVM

Aspect	Advantages	Disadvantages
Performance	Effective in high-dimensional spaces and with small datasets.	Computationally expensive for large datasets.
Flexibility	Handles nonlinear data using kernel functions.	Requires careful parameter tuning (e.g., kernel type, regularization).
Generalization	Robust to overfitting, especially in high-dimensional spaces.	Sensitive to noise and mislabeled data.
Applications	Versatile in tasks like text classification, image recognition, and bioinformatics.	Lack of probabilistic outputs requires additional methods for probability scores.
Scalability	Reliable performance on small to medium-sized datasets.	Poor scalability for very large datasets due to time and memory constraints.

SVM

Advantages

1.Effective in High-Dimensional Spaces
SVMs are highly effective when dealing with datasets that have many features (dimensions). This makes them suitable for text classification and gene expression data analysis.
2.Handles Nonlinear Data with Kernels
SVMs use kernel functions to transform data into higher dimensions, enabling the separation of classes that are not linearly separable in the original space. This flexibility is a key advantage.
3.Robust to Overfitting
Due to the principle of margin maximization, SVMs generalize well on unseen data, reducing the risk of overfitting, especially in high-dimensional feature spaces.
4.Works Well with Small Datasets
SVMs perform reliably even with small amounts of labeled data, as long as the data is well-separated.
5.Versatile in Applications
SVMs are widely used in applications like image classification, text categorization, bioinformatics, and even in real-time systems due to their reliable performance.

Disadvantages

1.Computationally Expensive
Training an SVM can be slow, especially for large datasets, due to the quadratic programming optimization process.
2.Requires Careful Parameter Tuning
SVMs require careful tuning of hyperparameters like the regularization parameter C and kernel type, which can be time-consuming.
3.Not Suitable for Large Datasets
SVMs are not ideal for very large datasets as they scale poorly with the size of the data, both in terms of time and memory.
4.Sensitive to Noise
SVMs can be sensitive to noisy data, especially if the data points are mislabeled, as they try to maximize the margin between classes.
5.Lack of Probabilistic Outputs
SVMs do not natively provide probability estimates, which can limit their use in certain applications. While methods like Platt scaling exist to address this, they add complexity.

SVM Best Practices

- **Choose the Right Kernel**: Select the appropriate kernel (e.g., linear, polynomial, RBF) based on the problem and data characteristics.

- **Scale Features**: Normalize or scale the features for better performance, especially with RBF kernels.

- **Handle Imbalanced Data**: Use techniques like class weighting or resampling to deal with imbalanced datasets.

- **Optimize Hyperparameters**: Use grid search or random search for tuning hyperparameters like C, gamma, and kernel type.

- **Combine with Other Models**: For large datasets, consider using ensemble methods or hybrid models for scalability.

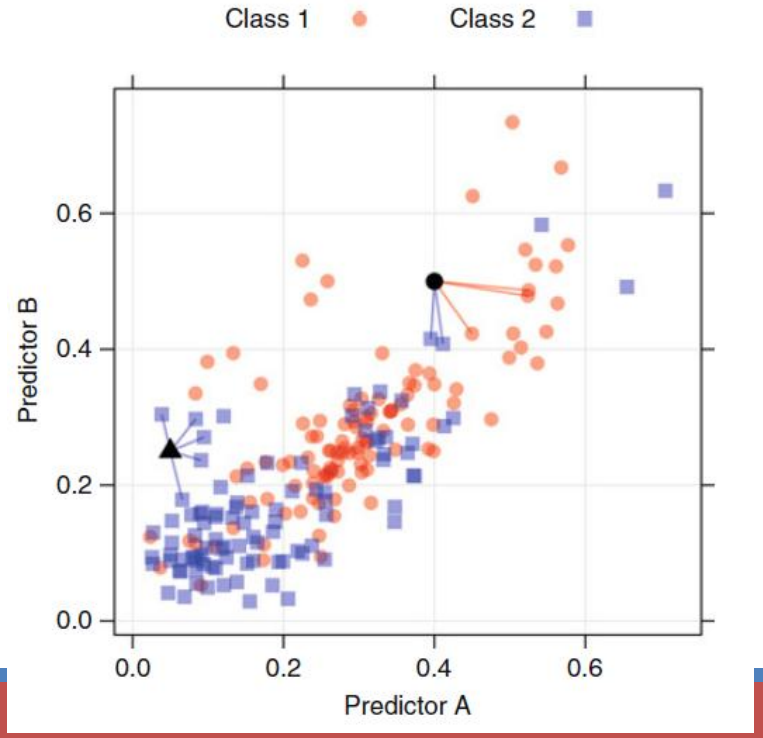
K NN

K- Nearest-Neighbor (kNN) Classifier

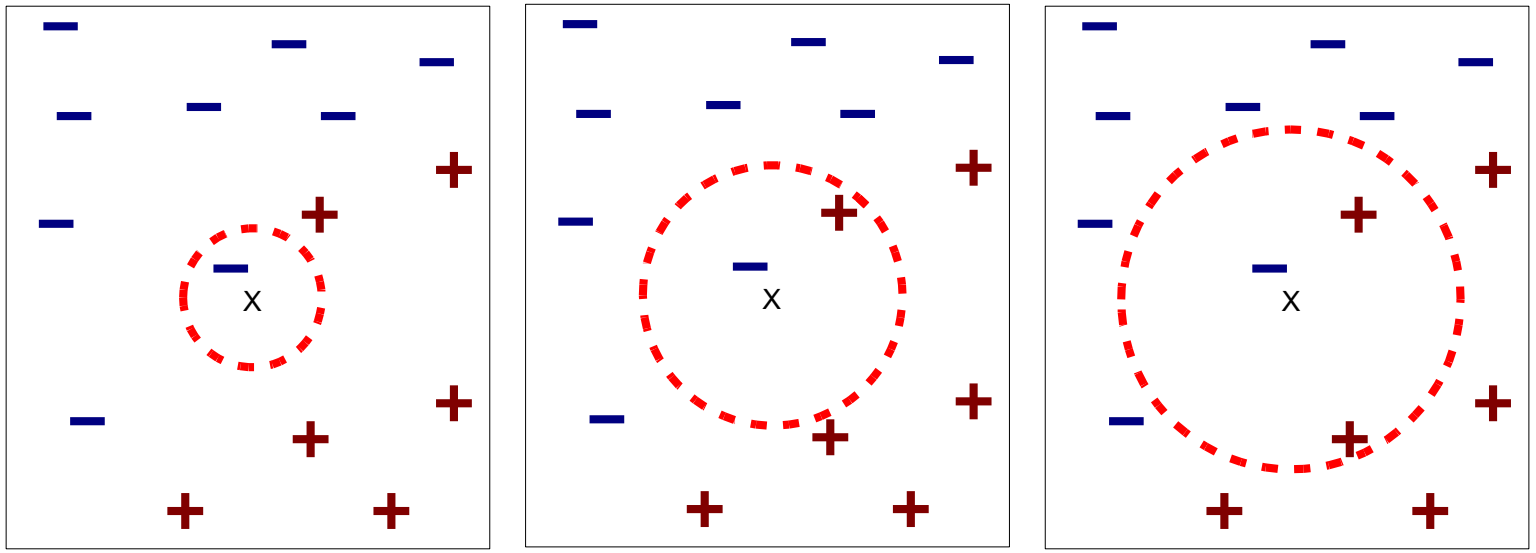
In the K-nearest neighbor classification model, a new sample is predicted based on the K-closest data points in the training set.

In the following example, two new samples (denoted by the solid dot and filled triangle) are being predicted.

- One sample (dot) is near a mixture of the two classes; three of the five neighbors indicate that the sample should be predicted as the first class.
- The other sample (triangle) has all five points, indicating the second class should be predicted.



kNN – k definition



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

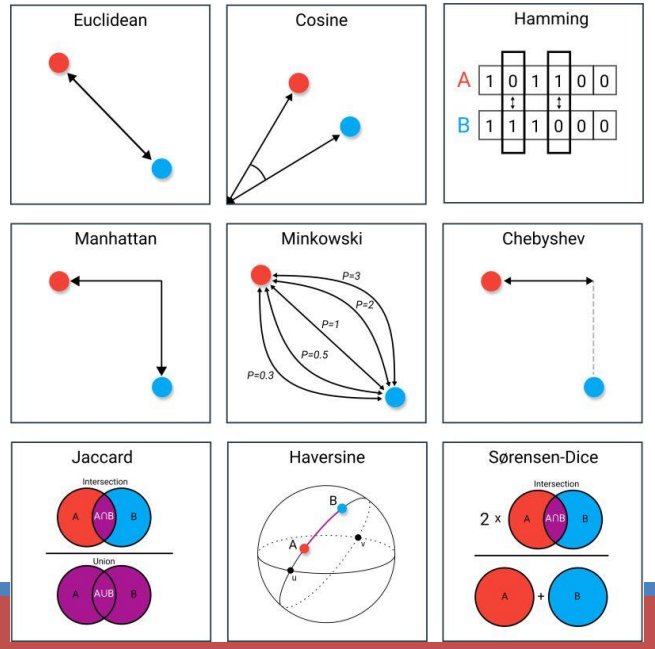
K-nearest neighbors of the new point x are the points that have the smallest distance to x

Classification – K-NN

Because the KNN method fundamentally depends on the distance between samples, the scale of the predictors can have a dramatic influence on the distances among samples.

Using distances between samples can be problematic if one or more of the predictor values for a sample is missing, since it is then not possible to compute the distance between samples.

KNN can present poor predictive performance when the local predictor structure is not relevant to the response. Irrelevant or noisy predictors are one culprit, as they can cause similar samples to be driven apart in the predictor space. Hence, removing irrelevant, noise-laden predictors is a key pre-processing step for KNN.



Classification – K-NN

•**Lazy learning algorithm**: KNN is a lazy learning algorithm since it does not have a specialized training phase and uses all the data for training during classification.

•**Non-parametric learning algorithm**: KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

•**Step 1**: We must load the training test dataset in the first step.
•**Step 2**: Next, we need to choose the value of K, i.e., the nearest data points. K can be any integer.
•**Step 3**: For each point in the test data, do the following:

- a. Calculate the distance between test data and each row of training data with the help of Euclidean, Manhattan, or Hamming distance methods. The common method to calculate distance is Euclidean.
- b. Now, sort them in ascending order based on the distance value.
- c. Next, it will choose the top K rows from the sorted array.
- d. Now, it will assign a class to the test point based on the most frequent class of these rows.

•**Step 4**: End

Classification – K-NN

Pros:

•KNN is known for its simplicity, comprehensibility, and scalability. Learning and implementation are extremely simple and intuitive.

•It is easy to interpret. The mathematical computations are easy to comprehend and understand.

•The calculation time is less.

•Predictive power is very high, which makes it effective and efficient.

•KNN is very effective for large training sets.

•It is very useful for nonlinear data because this algorithm has no assumptions about data.

•It is a versatile algorithm as we can use it for classification and regression.

•It has relatively high accuracy.

Cons:

•KNN can be expensive in determining K if the dataset is large. It requires more memory storage than an effective classifier or supervised learning algorithms.

•In KNN, the prediction phase is slow for a larger dataset. The computation of accurate distances plays a big role in determining the algorithm's accuracy.

•One of the major steps in KNN is determining the parameter K. Sometimes, it is unclear which type of distance to use and which feature will give the best result.

•It is very sensitive to the data's scale and irrelevant features. Irrelevant or correlated features have a high impact and must be eliminated.

•The computation cost is quite high as each training example's distance is calculated.

•KNN is a lazy learning algorithm as it doesn't learn from the training data; it simply memorizes it and then uses that data to classify the new input.

•Typically difficult to handle high dimensionality

Classification – When to use K-NN

- **Simple classification or regression tasks** where interpretability and ease of implementation are key, and the dataset is small.
- When you have **no prior assumption** about the data distribution.
- **Non-linear data** where simple linear models like logistic regression or SVM might struggle.
- For **small datasets** where training time is not a concern, and prediction speed is less critical.

Performance Evaluation

Performance Evaluation - Confusion Matrix

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	(TP)	(FN)
	Class=No	(FP)	(TN)

TP (true positive)

FN (false negative)

FP (false positive)

TN (true negative)

Python: from sklearn.metrics import confusion_matrix

Performance Evaluation - Confusion Matrix

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	(TP)	(FN)
	Class=No	(FP)	(TN)

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

Accuracy - fraction of examples correctly classified.

1-Accuracy: Error rate (misclassification rate)

Performance Evaluation - Accuracy

Dataset: 100 patients
- 90 healthy
- 10 diseased

Classifier always predicts:
Healthy

	Predicted Healthy	Predicted Diseased
Actual Healthy	90 ✅	0
Actual Diseased	10 ❌	0

Accuracy = 90% ✅
Disease detection = 0% ❌

👉 **High accuracy can hide critical errors in minority classes!**

Performance Evaluation - Accuracy

Limitations of Accuracy

- **Class imbalance**
  - Accuracy can be misleading when one class is much more frequent than others.
  - Example: 95% accuracy may mean the model is always predicting the majority class.

- **No insight on error types**
  - Accuracy does not distinguish between false positives and false negatives.
  - Critical in domains like healthcare or fraud detection.

👉 Use other performance metrics (e.g. Precision, Recall, F1-score, AUC)

👉 Find a way to balance the data set (e.g. oversampling, under-sampling)

Performance Evaluation – Other metrics

True Positive Rate (TPR), Sensitivity, Recall
Of all actual positives, how many are correctly detected?
(Focus: false negatives)

$$TPR = \frac{TP}{TP + FN}$$

True Negative Rate (TNR), Specificity
of all negative examples, how many are truly negative?
(Focus: false negatives)

$$TNR = \frac{TN}{TN + FP}$$

False Positive Rate (FPR)
how often an actual negative instance will be classified as positive, i.e. “false alarm”
(Focus: false positives)

$$FPR = 1 - TNR = \frac{FP}{FP + TN}$$

Precision
Of all predicted positives, how many are truly positive?
(Focus: false positives)

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

Performance Evaluation – Other metrics

F1-score

Harmonic mean of Precision and Recall, useful for imbalanced classes

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

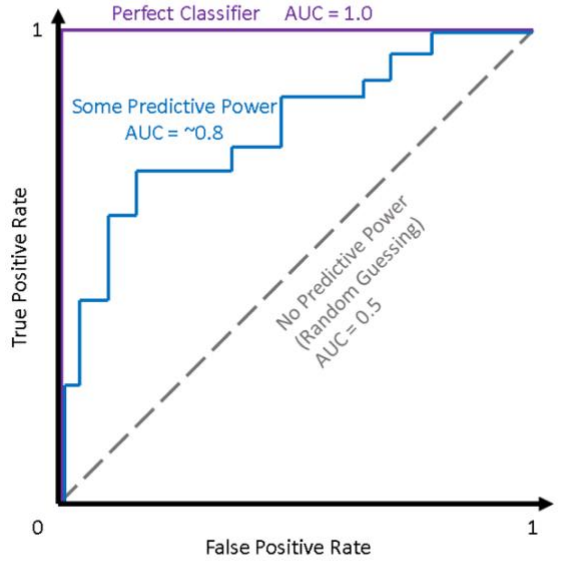
Balanced Accuracy

Average recall for each class, compensates class imbalance

$$Balanced\ Accuracy = \frac{Recall + Specificity}{2}$$

ROC curve - Curve plotting Recall (TPR) vs False Positive Rate

ROC curve is produced by calculating and plotting the TPR against the False Positive Rate for a single classifier at a variety of thresholds. For example, in logistic regression, the threshold would be the predicted probability of an observation belonging to the positive class. Usually, in logistic regression, if an observation is predicted to be positive at > 0.5 probability, it is labeled as positive. However, we could choose any threshold between 0 and 1 (0.1, 0.3, 0.6, 0.99, etc.) — and ROC curves help us visualize how these choices affect classifier performance.



Area under ROC curve (AUC) - Measures the ability to distinguish classes

The higher the AUC, the better a classifier performs for the given task.
For a random guessing classifier => AUC = 0.5.
For a perfect classifier => AUC = 1.0.
Most classifiers fall between 0.5 and 1.0

Performance Evaluation – Other metrics

Metric	Definition / Focus	When Useful
Accuracy	Overall % correct predictions	Only if classes are balanced
Precision	Of predicted positives, % correct	When false positives are costly
Recall	Of actual positives, % detected	When false negatives are costly
F1-score	Harmonic mean of Precision & Recall	Balances FP & FN trade-off
Balanced Accuracy	Average recall across classes	Imbalanced datasets
ROC & AUC	Trade-off TPR vs FPR; AUC = area under curve	Overall discrimination ability

Class Imbalance

Class Imbalance Problem

Classification predictive modeling involves predicting a class label for a given observation.

An **imbalanced classification problem** is an example of a classification problem where the **distribution of examples across the known classes is biased or skewed**. The distribution can vary from a **slight bias to a severe imbalance** where there is one example in the minority class for hundreds, thousands, or millions of examples in the majority class or classes.

Imbalanced classifications challenge predictive modeling since most machine learning algorithms assume equal examples for each class. This results in poor performance, particularly for the often more important minority class, making classification errors more significant for it compared to the majority class.

Class Imbalance Problem

Techniques for handling class imbalance:

Class Imbalance Problem

Techniques for handling class imbalance:

Data Level

- Undersampling the majority class
- Oversampling the minority class

Metric Level

- Consider metrics not biased

Classifier Level

- Penalize learning algorithms
- Ensemble methods
- One-Class classification

Class Imbalance Problem

Techniques for handling class imbalance:

Data Level

- A widely adopted technique for dealing with highly unbalanced datasets is called resampling. It consists of removing samples from the majority class (under-sampling) and/or adding more examples from the minority class (over-sampling).
  - The simplest implementation of **over-sampling** is to duplicate random records from the minority class, **which can cause overfitting**.
  - In **under-sampling**, the simplest technique involves removing random records from the majority class, **which can cause loss of information**.

- Cluster the records of the majority class, and do the under-sampling by removing records from each cluster, thus seeking to preserve information.
- In over-sampling, instead of creating exact copies of the minority class records, we can introduce small variations into those copies, creating more diverse synthetic samples.

Techniques:

Random under-sampling, random over-sampling, Tomek links, SMOTE, NearMiss

Class Imbalance Problem

Techniques for handling class imbalance:

Metric Level

- Accuracy is not the best metric to use when evaluating imbalanced datasets as it can be misleading.
- Metrics that can provide better insight are:
  - Confusion Matrix: a table showing correct predictions and types of incorrect predictions.
  - Precision: the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.
  - Recall: the number of true positives divided by the number of positive values in the test data. The recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.
  - F1: Score: the weighted average of precision and recall.
  - Area Under ROC Curve (AUROC): AUROC represents the likelihood of your model distinguishing observations from two classes.

Class Imbalance Problem

Techniques for handling class imbalance:

Classifier Level

- Penalize learning algorithms
  - This methods increase the cost of classification mistakes on the minority class, by penalizing mistakes on the minority class by an amount proportional to how underrepresented it is.
  - It may be appropriate to believe that misclassifying true events (false negatives) is X times as costly as incorrectly predicting nonevents (false positives). Incorporation of specific costs during model training may bias the model towards less frequent classes. Unlike using alternative cutoffs, unequal costs can affect the model parameters and thus have the potential to make true improvements to the classifier.
- Ensemble methods
  - It is a machine learning technique that combines several base models in order to produce one optimal predictive model.
  - As example Random Forest, and Bagging Trees, combination of decision trees.

Class Imbalance Problem

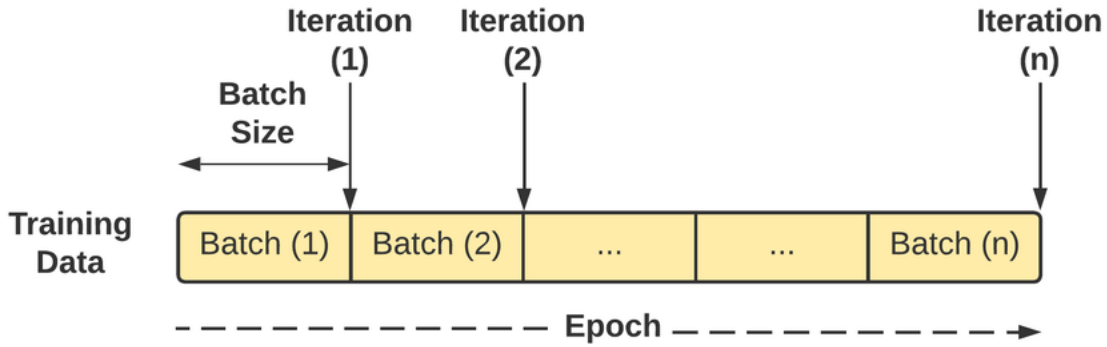
Techniques for handling class imbalance:

Classifier Level

- One-class classification
  - It is the technique of handling class imbalance by modelling the distribution of only the minority class and treating all other classes as out-of-distribution/anomaly classes.
  - Using this technique, we aim to create a classifier that can detect examples belonging to the minority class rather than discriminating between minority and majority class.
  - This is done in practice by training the model on only the instances belonging to the minority class and during test time using examples belonging to all the classes to test the ability of the classifier to correctly identify examples belonging to the minority class and at the same time flagging examples belonging to other classes.

Epoch / Batch

Epoch /Batch Size / Iterations / Train step



One Epoch is when an ENTIRE dataset is passed through the model only ONCE.

If data is too big to feed to the computer at once one epoch is divided in several smaller batches.
Batch Size: Total number of training samples present in a single batch.
Iterations is the number of batches needed to complete one epoch.

Example: Let's say we have 2000 training samples.
We can divide the dataset of 2000 samples into batches of 500 then it will take 4 iterations to complete 1 epoch.

Training run/step - one update of the model parameters (weights).
We update the parameters usually after one iteration.