


Departamento de Eletrónica, Telecomunicações e
Informática

**Naïve Bayes & Decision Tree classifiers
SUPPORT VECTOR MACHINE (SVM)
& MODEL PERFORMANCE**

Author: Petia Georgieva

Edited by: Susana Brás (Susana.bras@ua.pt)

LECTURE outline

1. Naïve Bayes (NB) classifier
2. Decision tree (DT) classifier 
3. Linear Support Vector Machine (SVM)
4. Nonlinear SVM - Gaussian RBF Kernel
5. Performance evaluation – confusion matrix
6. Class imbalance problem
7. k-Nearest Neighbor (k-NN) classifier

Naïve Bayes (NB) classifier



Bayesian Classifier

- Given labeled data with p classes (C_1, C_2, \dots, C_p), each example has n features $x = [x_1, x_2, \dots, x_n]$
- Each feature (x_j) is treated as a random variable.
- Compute the probability of a new example
- $x_{new} = (x_{new1}, x_{new2}, \dots, x_{newn})$ to belong to each one of the classes.
- The class with the highest probability is assigned to the new example.
- Use Bayes Theorem to compute the *a posteriori* probability

$$P(C_i | x) = \frac{P(x | C_i) P(C_i)}{P(x)}$$

$$x_{new} \Rightarrow C_i : P(C_i | x_{new}) > P(C_j | x_{new}), \quad \forall j \neq i$$

Nonparametric probability estimation -example

<i>Tid</i>	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Prior probability of the class:

$P(\text{Class} = \text{No}) = ?$

$P(\text{Class} = \text{Yes}) = ?$

Which are the categorical Features ?

How many values have the categorical Features ?

$P(\text{Status} = \text{'Married'} \mid \text{Class No}) = ?$

$P(\text{Status} = \text{'Married'} \mid \text{Class Yes}) = ?$

Nonparametric probability estimation -example

$$P(C_i) = \frac{\text{N}^\circ \text{of train examples that belong to } C_i}{\text{N}^\circ \text{of all train examples}}$$

$$P(x = \text{'string value'} | C_i) = \frac{\text{\# of train examples that belong to } C_i \text{ \& has this 'string value'}}{\text{Number of all train examples that belong to } C_i}$$

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

$$P(\text{Class} = \text{No}) = 7/10$$
$$P(\text{Class} = \text{Yes}) = 3/10$$

Feature *Marital Status* has 3 string values :single, married, divorced

$$P(\text{Status} = \text{'Married'} | \text{Class No}) = 4/7$$
$$P(\text{Status} = \text{'Married'} | \text{Class Yes}) = 0/3$$

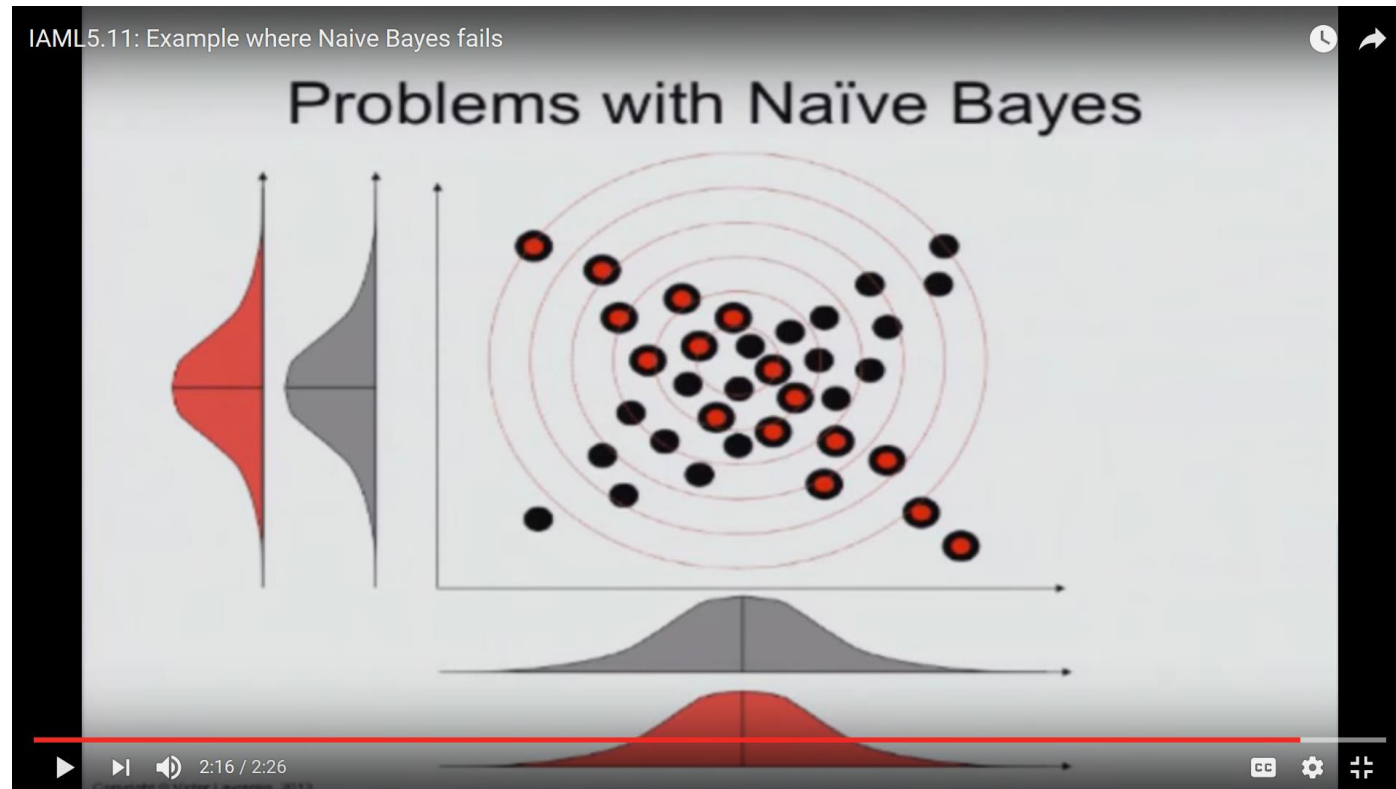
The same counting is done for each string value of the feature, and for each feature.



Naive Bayes (NB) Classifier -problems

- Main assumption of NB classifier:
the features are independent !!!
- The NB classifier will fail if the only thing that separates two classes is the co-variance of the attributes (not the mean or the variance).
- Solution: Multivariate Gaussian Distribution, compute covariance matrix, compute joint distribution.

NB will not be able to separate these classes



2 feature, 2 class problem:

The Gaussian distribution of the data for each class is the same, but for the black class we have a positive correlation between the features (if x_1 increases, x_2 also increases), and for the red class we have a negative correlation between the features (if x_1 increases, x_2 decreases).

Naive Bayes (NB) Classifier

It is based on Bayes' theorem, which is a mathematical formula that describes the probability of an event occurring given the knowledge of other events.

How Naive Bayes works

The Naive Bayes classifier works by calculating the probability of each class given the input features. The class with the highest probability is then predicted as the output.

Pros

- It is easy and fast to predict a class of test data set.
- Naive Bayes classifier performs better compared to other models assuming independence.
- It performs well in case of categorical data as compared to numeric data.
- It requires less training data.

Cons

- If an instance in test data set has a category that was not present during training then it will assign it "Zero" probability and won't be able to make prediction. This is known as **Zero frequency problem**.
- It is also known as a bad estimator.
- It solely relies on the independence predictor assumption.

Decision Trees

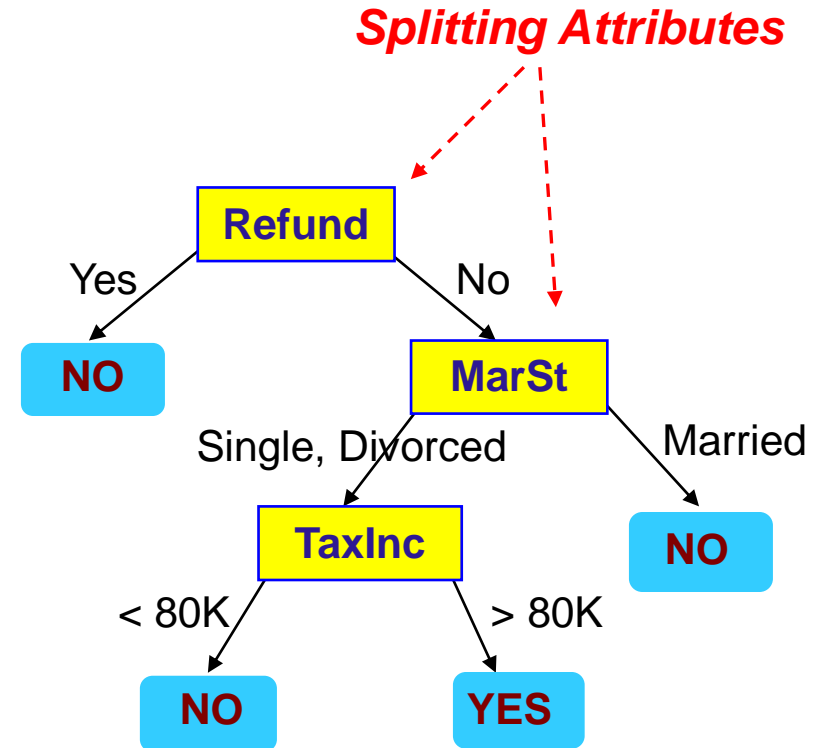


Classification by Decision Tree – model 1

categorical
categorical
continuous
class

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data

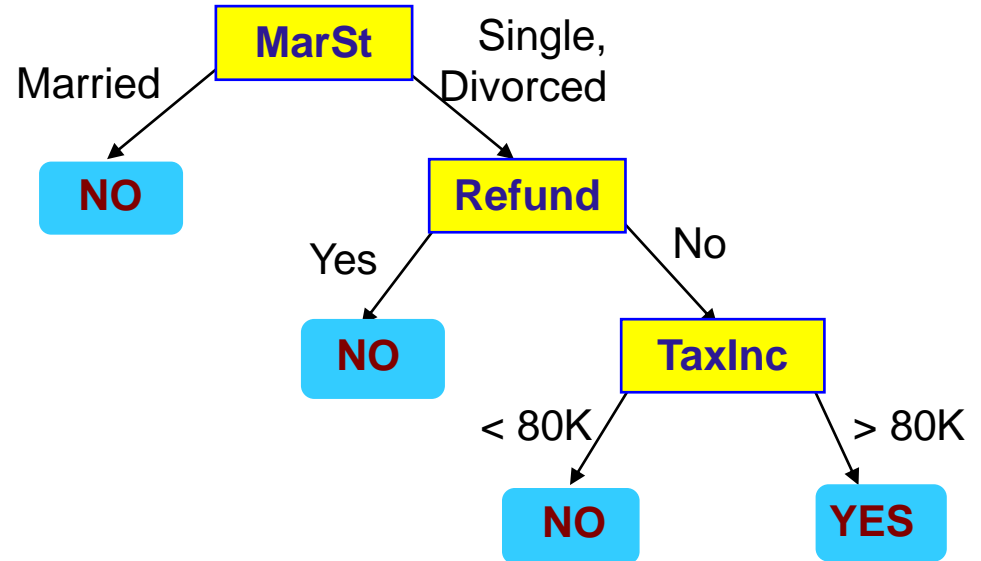


Model: Decision Tree

Classification by Decision Tree – model 2

<i>Tid</i>	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data



There could be more than one tree that fits the same data!

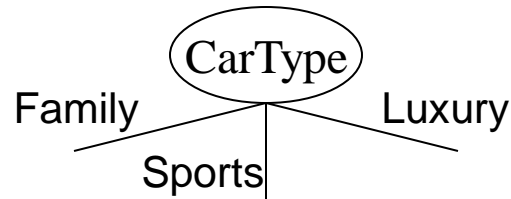
Decision Tree

Decision Tree has 2 basic type of nodes:

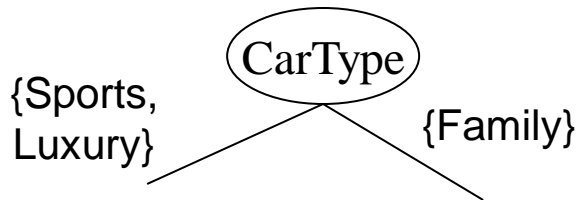
- **Leaf node** - a class label, determined by majority vote of training examples reaching that leaf.
- **Node** is a question on one feature. It branches out according to the answers.
 - **root node**: the first (top) node of the tree
 - **child node**: the next node to a current node

Splitting of Categorical Features

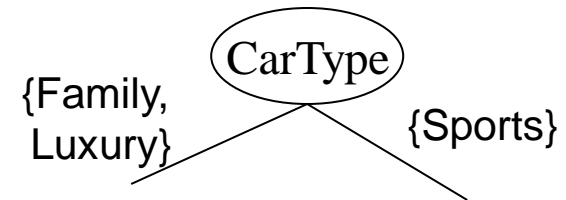
- **Multi-way split:** Use as many partitions as distinct values.



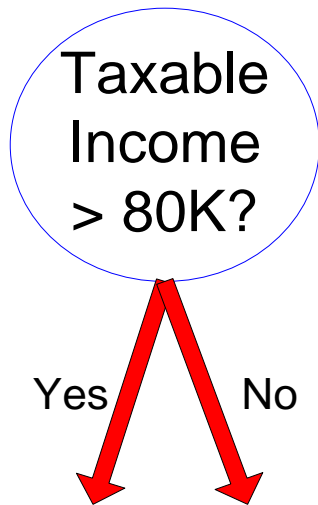
- **Binary split:** Divides values into two subsets.



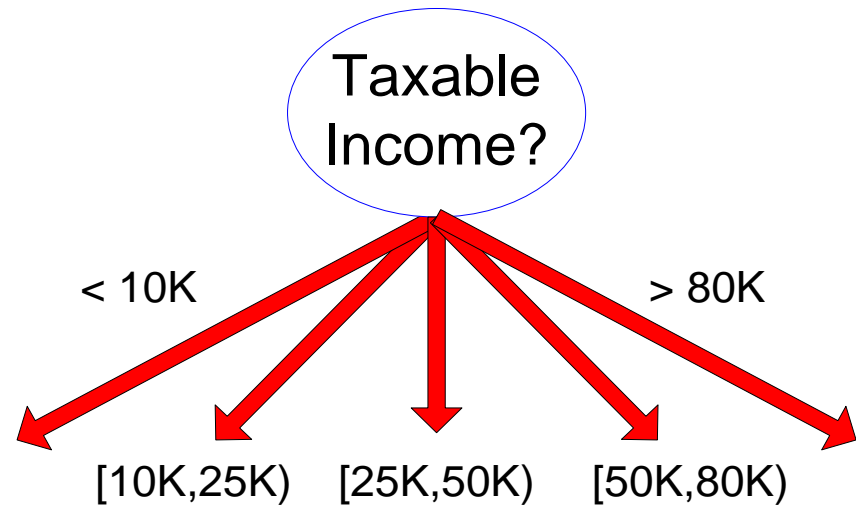
OR



Splitting of Continuous Features




(i) Binary split



(ii) Multi-way split

How to determine the best split ?

We want:

- To get the smallest tree
- Pure leaf nodes, i.e. all examples having (almost) the same class (ex. C0 or C1).
- Choose the feature that produces the “purest” (the most homogeneous) nodes
- We need a measure of node impurity  (homogeneity).

C0: 5 C1: 5

**Non-homogeneous,
High degree of impurity**

C0: 9 C1: 1

**Homogeneous,
Low degree of impurity**

Measures of Node Impurity

- **Gini Index** at a given node:

$$GINI(node) = 1 - \sum_{Class_j} [p(Class_j | node)]^2$$

- **Classification error** at a given node:

$$Error(node) = 1 - \max_{Class_j} p(Class_j | node)$$

- **Entropy (H)** at a given node:



$$H(node) = - \sum_{Class_j} p(Class_j | node) \log p(Class_j | node)$$

$p (Class_j / node)$: probability of $Class_j$ at a given $node$

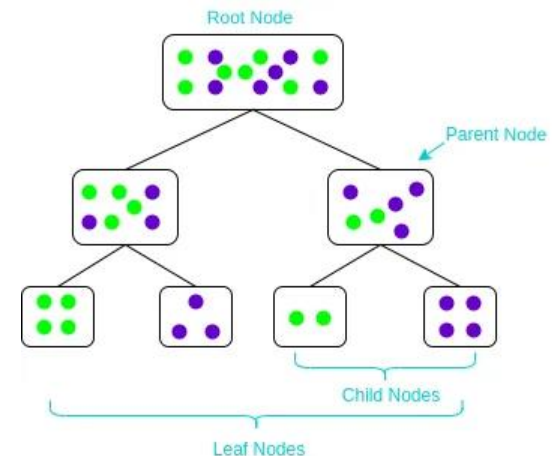
Building a Decision Tree

Choose feature that maximizes the information gain (minimizes the node impurity) !

Information Gain = Entropy (H) before split - Entropy (H) after split at one node (one feature).
How much uncertainty was reduced after splitting on a given feature.

$$H(node) = - \sum_{Class_j} p(Class_j | node) \log p(Class_j | node)$$

This procedure is repeated for each node. Splitting stops when data cannot be split any further or the class is defined by the majority of elements of a subset.



Other DT Algorithms

- **ID3 (Iterative Dichotomiser 3)** algorithm. Use Information Gain to select the best attribute.
- **Modified ID3** – use Gain ratio.
- **C4.5** (extension of ID3)- deals with numeric attributes, missing values, noisy data
- **CART (Classification And Regression Tree)** – similar approach
- Random Forest – ensemble of DT classifiers 🗨

Remark: There are many other feature selection criteria!

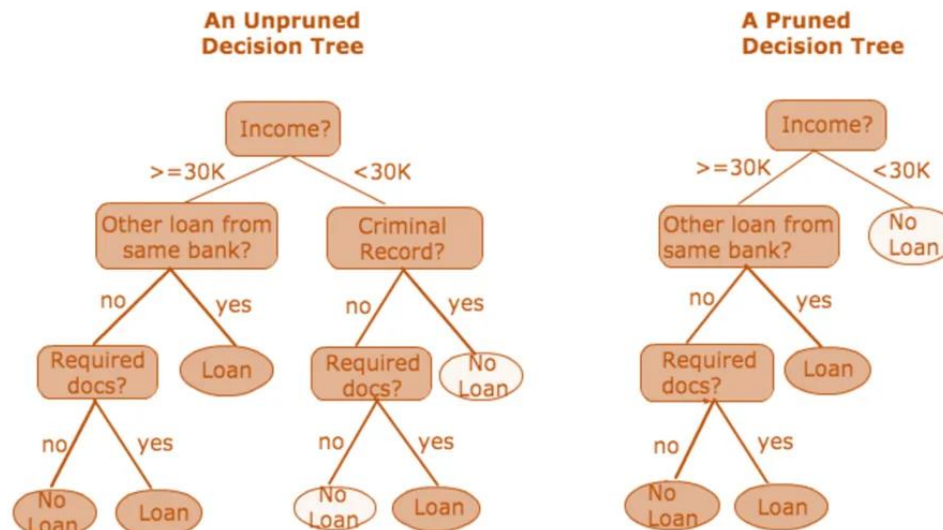
Gain ratio is a modification of the splitting criteria to deal better with overfitting. Gain ratio takes number and size of branches into account when choosing the most favorable feature to split at a node.

$$\text{Gain ratio} = \text{Information_Gain} / \text{Split Entropy}$$

DT Pruning

Two strategies to prevent DT overfitting:

- **Post-pruning** - First, build full decision tree and then discard unreliable parts
- **Pre-pruning** – stop growing a branch when information gain does not change significantly 🗨
- Post-pruning preferred in practice—pre-pruning can “stop too early” 🗨



A decision tree is a non-parametric supervised learning algorithm that can be used for both classification and regression. It uses a tree-like structure to represent decisions and their potential outcomes. Decision trees are simple to understand and interpret and can be easily visualized. However, when a decision tree model becomes too complex, it does not generalize well from the training data and results in overfitting.

The Decision Tree classifier operates by recursively splitting the data based on the most informative features.

1. Start with the entire dataset at the root node.
2. Select the best feature to split the data (based on measures like Gini impurity).
3. Create child nodes for each possible value of the selected feature.
4. Repeat steps 2–3 for each child node until a stopping criterion is met (e.g., maximum depth reached, minimum samples per leaf, or pure leaf nodes).
5. Assign the majority class to each leaf node.

Classification Step

1. Start at the root node of the trained decision tree.
2. Evaluate the feature and split condition at the current node.
3. Repeat step 2 at each subsequent node until reaching a leaf node.
4. The class label of the leaf node becomes the prediction for the new instance.

Key Parameters

Decision Trees have several important parameters that control their growth and complexity:

1. **Max Depth**: This sets the maximum depth of the tree, which can be a valuable tool in preventing overfitting.
2. **Min Samples Split**: This parameter determines the minimum number of samples needed to split an internal node.
3. **Min Samples Leaf**: This specifies the minimum number of samples required at a leaf node.
4. **Criterion**: The function used to measure the quality of a split (usually “gini” for Gini impurity or “entropy” for information gain).

Pros:

1. **Interpretability**: Easy to understand and visualize the decision-making process.
2. **No Feature Scaling**: Can handle both numerical and categorical data without normalization.
3. **Handles Non-linear Relationships**: Can capture complex patterns in the data.
4. **Feature Importance**: Provides a clear indication of which features are most important for prediction.



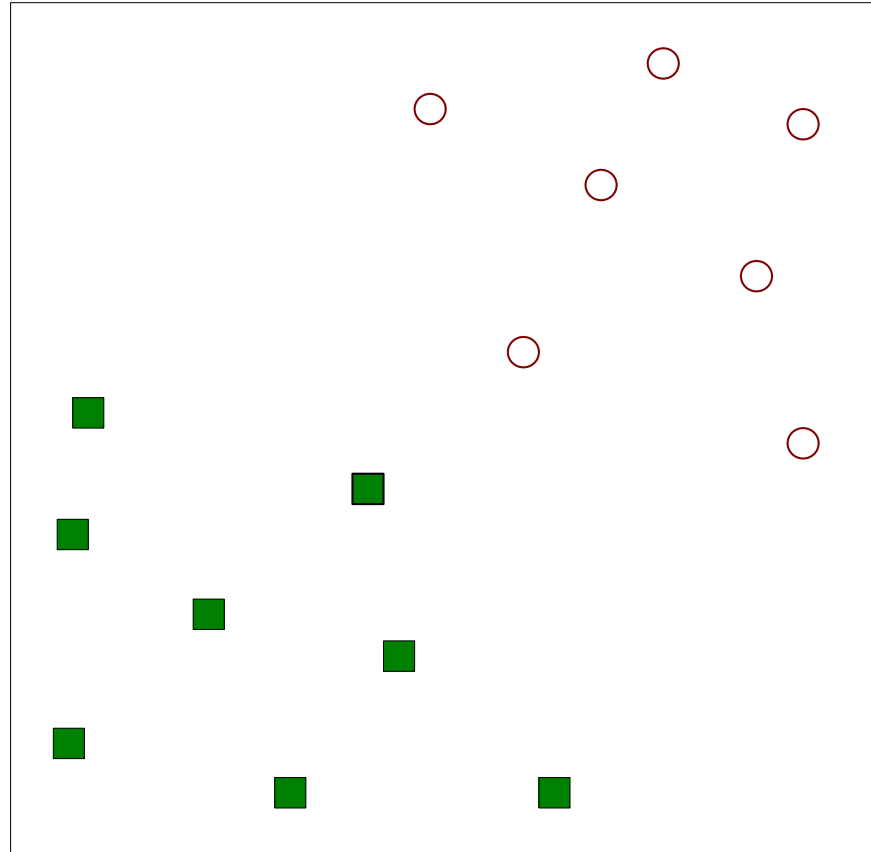
Cons:

1. **Overfitting**: Prone to creating overly complex trees that don't generalize well, especially with small datasets.
2. **Instability**: Small changes in the data can result in a completely different tree being generated.
3. **Biased with Imbalanced Datasets**: Can be biased towards dominant classes.
4. **Inability to Extrapolate**: Cannot make predictions beyond the range of the training data.

Linear Support Vector Machine (SVM)

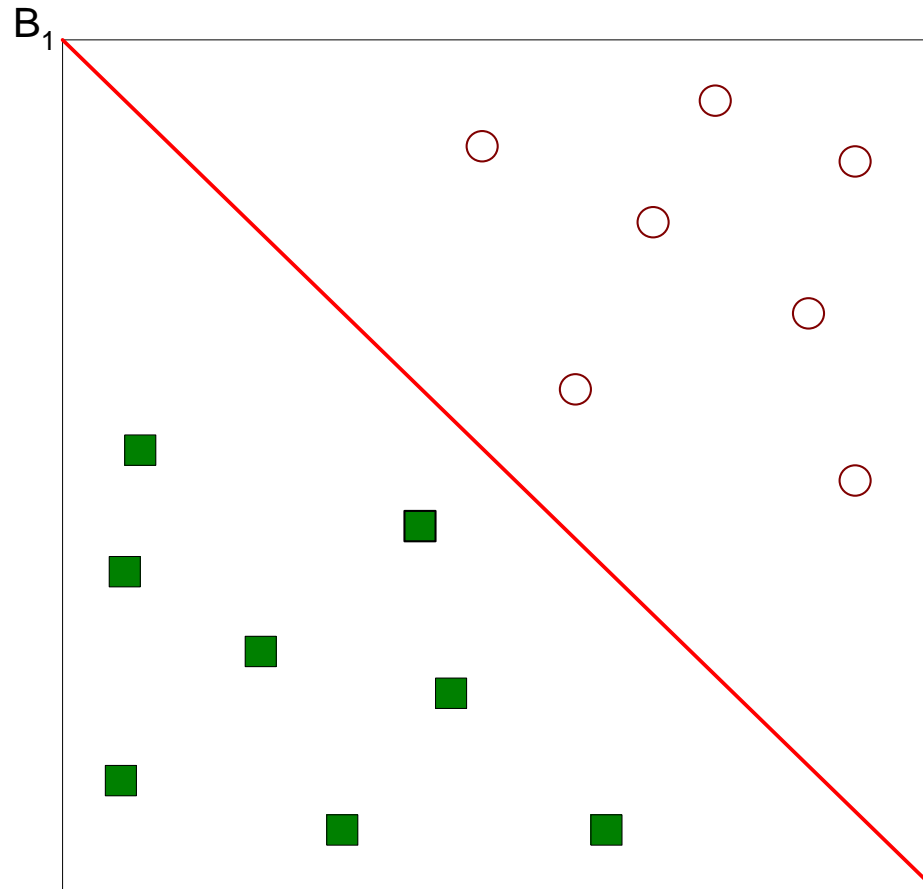


Linearly separable classes



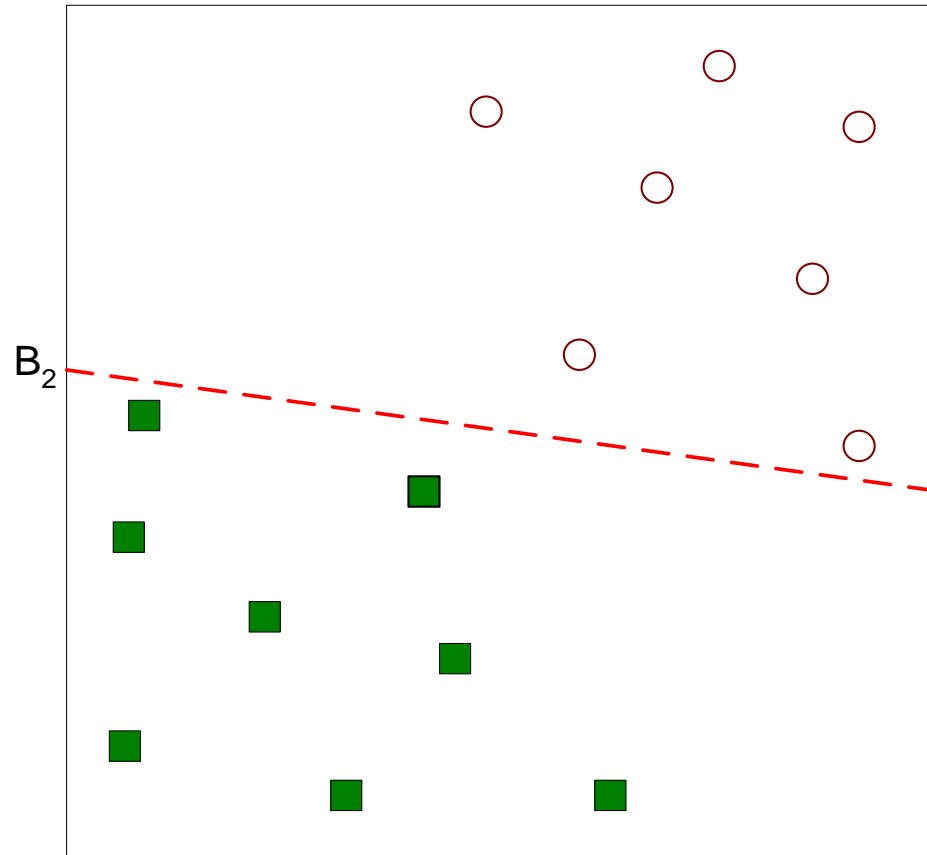
Find a decision boundary to separate data

Linearly separable classes



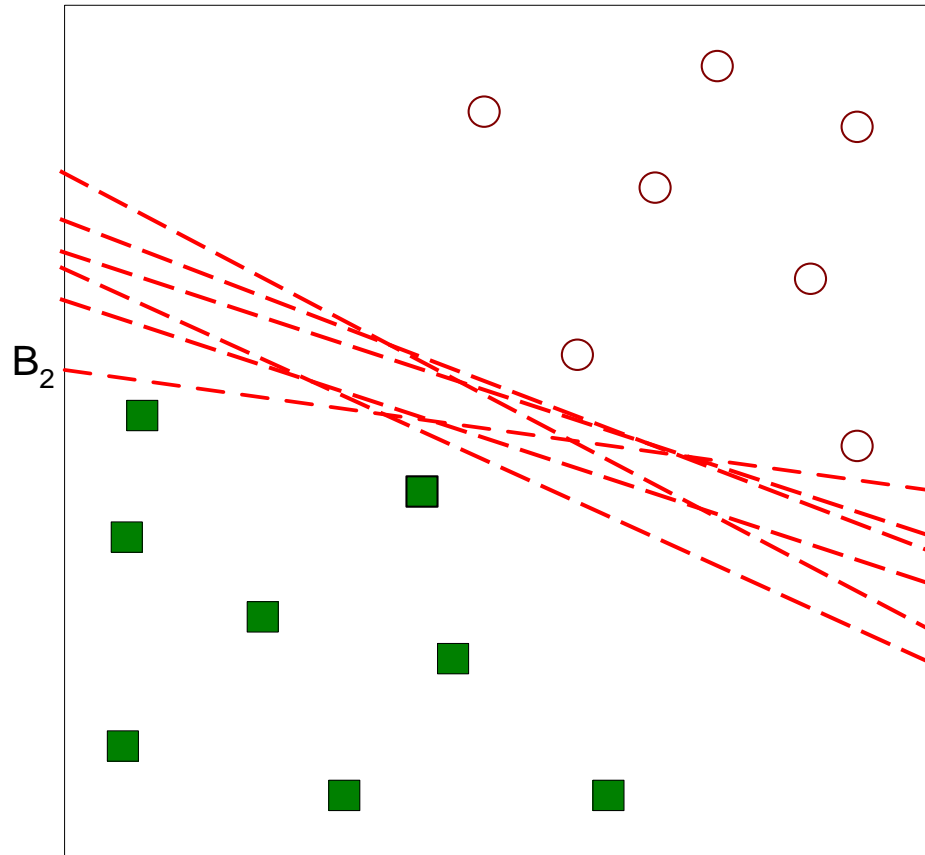
One Possible Solution

Linearly separable classes



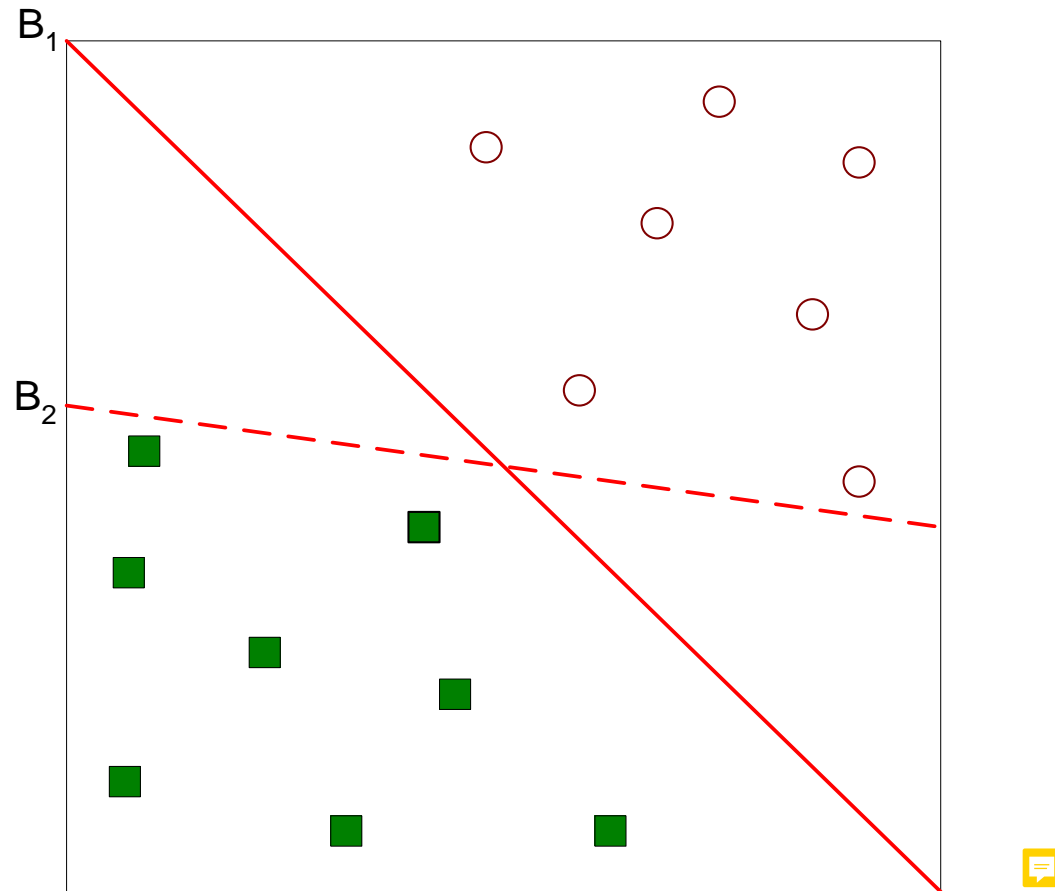
Another possible solution

Linearly separable classes



Many possible solutions

Linearly separable classes

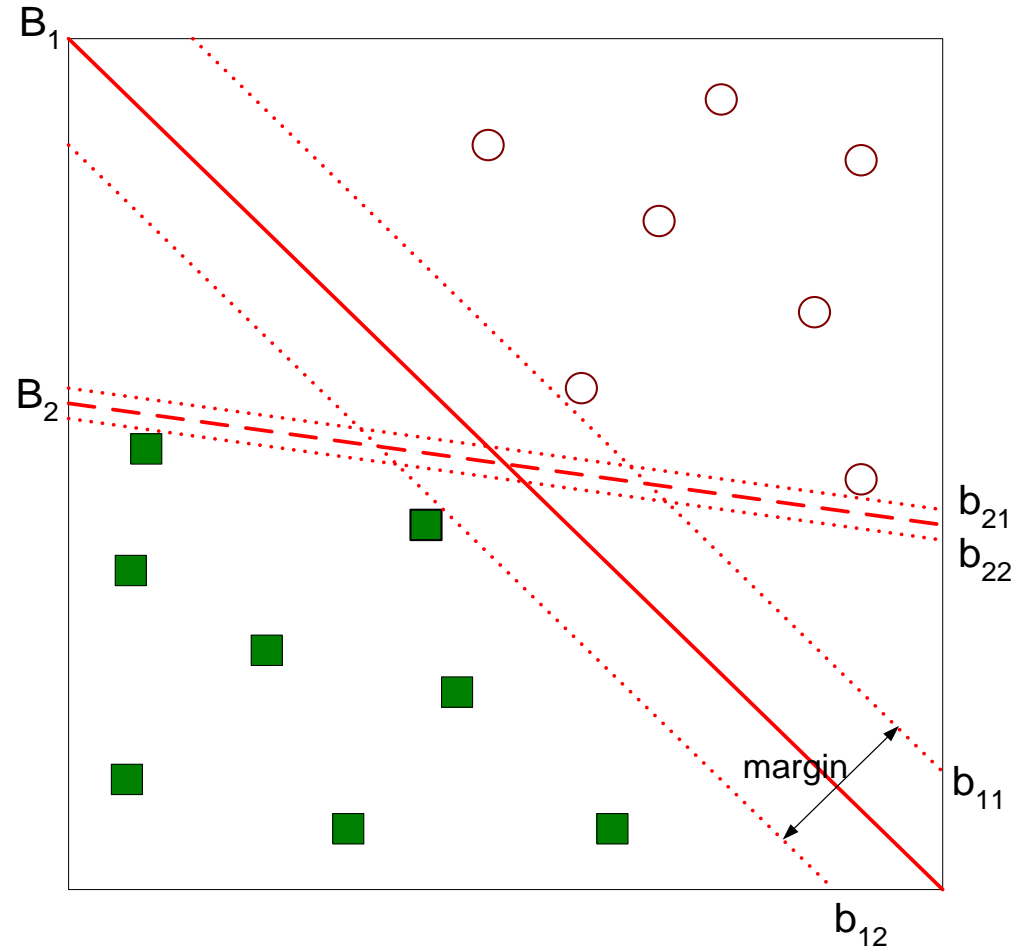


Which one is better? B1 or B2?

SVM - Large margin classifier

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N — the number of features) that distinctly classifies the data points.

Our objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.



Find a boundary that **maximizes the margin => B1 is better than B2**

Proposed by Vladimir N. Vapnik and Alexey Chervonenkis, 1963

SUPPORT VECTORS (v_1, v_2, v_3)

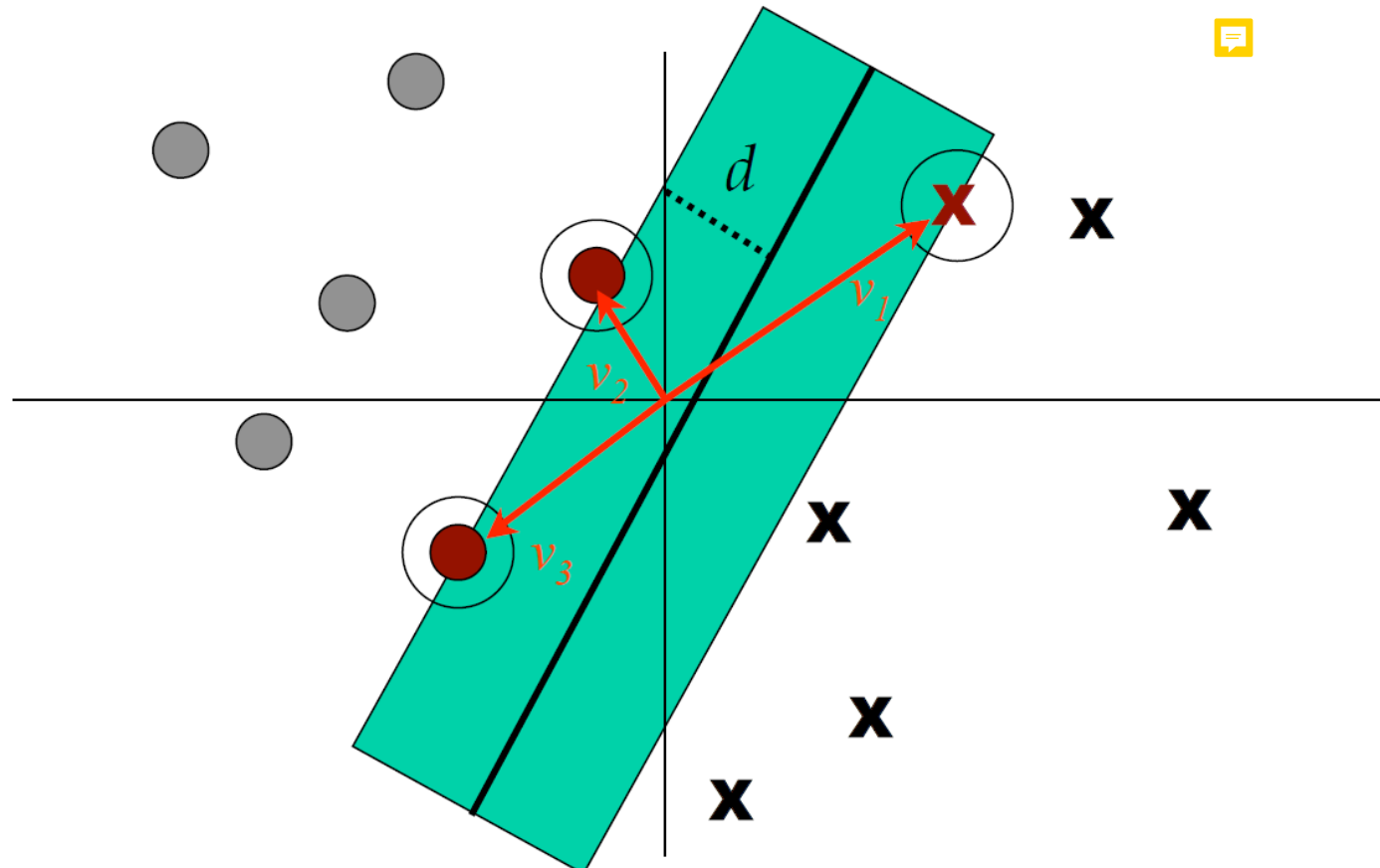
Only the closest points (support vectors) from each class are used to decide which is the optimum (the largest) margin between the classes.

The dimension of the hyperplane depends upon the number of features.

If the number of input features is 2, then the hyperplane is just a line.

If the number of input features is 3, then the hyperplane becomes a two-dimensional plane.

...

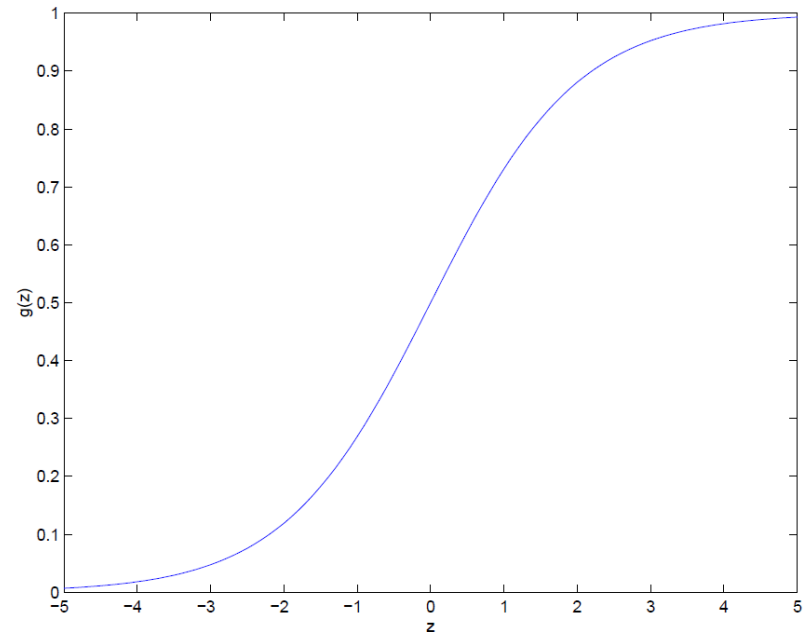


Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane.

?

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}} = \frac{1}{1 + e^{-z}} = g(\theta^T x) = g(z)$$

$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \theta_n x_n$$



Logistic Regression (LogReg) -revised

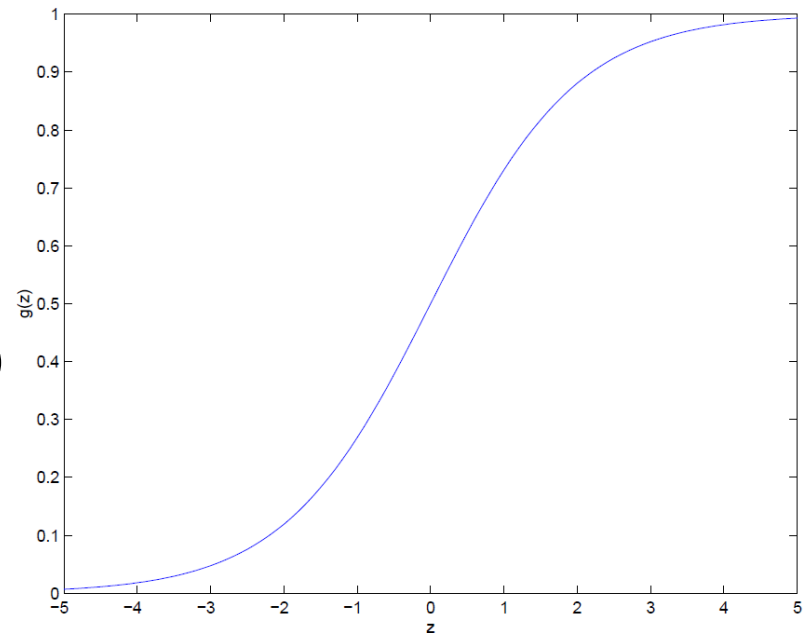
$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}} = \frac{1}{1 + e^{-z}} = g(\theta^T x) = g(z)$$

$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \theta_n x_n$$

if $y = 1$, we want $h_{\theta}(x) \approx 1$, $\theta^T x \gg 0$

if $y = 0$, we want $h_{\theta}(x) \approx 0$, $\theta^T x \ll 0$

Logistic (sigmoid) function

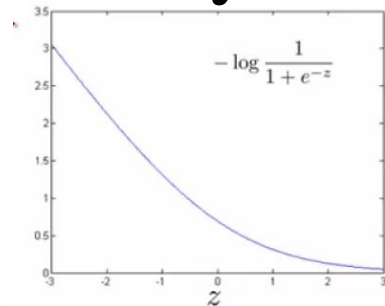


SVM cost function

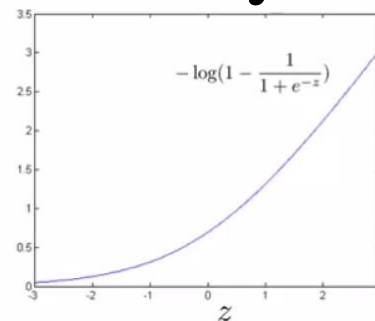
Regularized LogReg cost function:

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \left(-\log h_{\theta}(x^{(i)}) \right) + (1 - y^{(i)}) \left(-\log(1 - h_{\theta}(x^{(i)})) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

for $y=1$

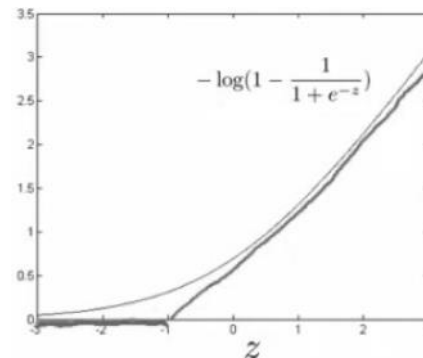
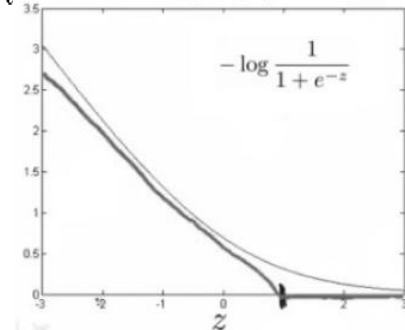


for $y=0$



Regularized SVM cost function (Modification of LogReg cost function.
cost0 & **cost1** are asymptotic safety margins with computational advantages)

$$\min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



$$z = \theta^T x$$

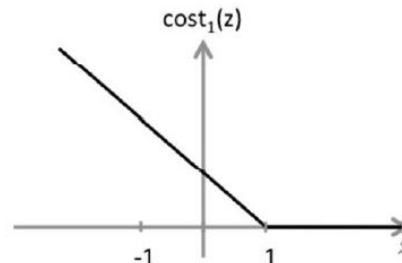
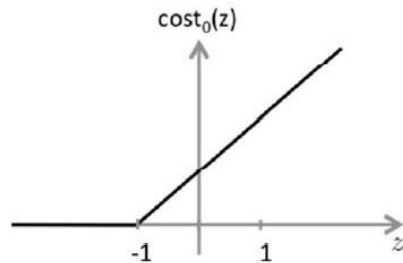
SVM cost function

Regularized LogReg cost function:

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \left(-\log h_{\theta}(x^{(i)}) \right) + (1 - y^{(i)}) \left(-\log(1 - h_{\theta}(x^{(i)})) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Regularized SVM cost function

$$\min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



$$z = \theta^T x$$

Different way of parameterization: C is equivalent to $1/\lambda$.



$C > 0$ - parameter that controls the penalty for misclassified training examples.
Increase C - more importance to training data fitting.

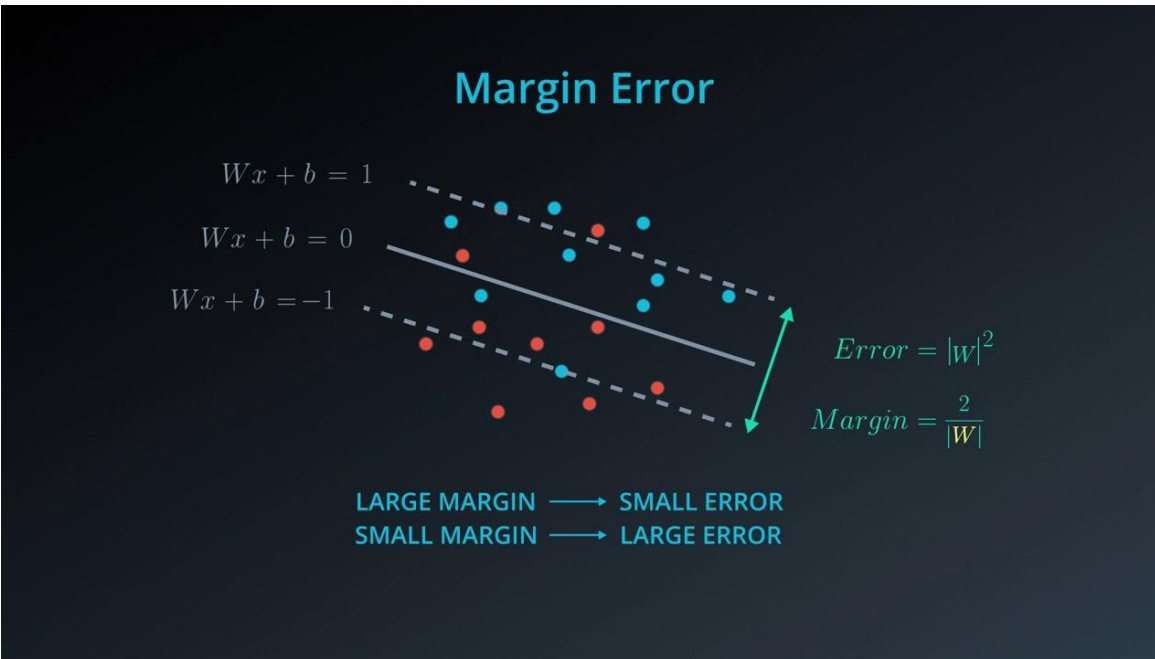
Decrease C - more importance to generalization properties (combat overfitting).

SVM Algorithm

Two quantities to optimize : classification error (how many points are wrongly classified) and “margin error” (optimize the margin between the two classes)

Search for the largest margin that minimizes the classification error.

C controls how wide is the “street”.



$$\theta^T x \Rightarrow Wx + b$$

$$\min_W \sum_{j=1}^n W_j^2 = |W|^2$$

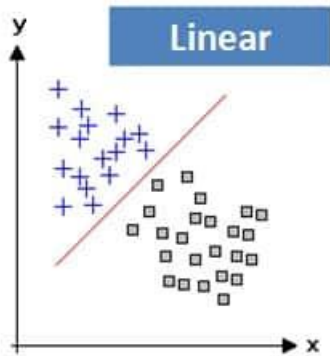
(L_2 norm) such that

$$Wx^{(i)} + b \geq 1, \quad \text{if } y = 1$$

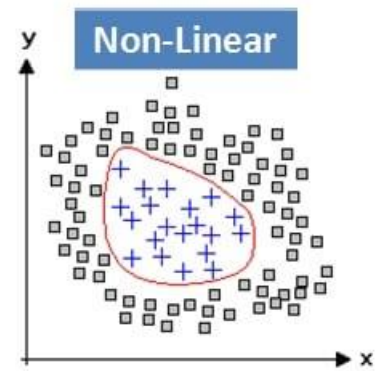
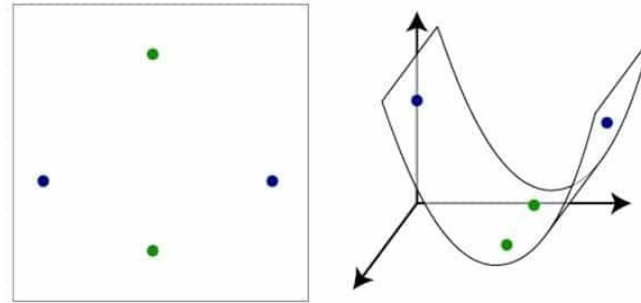
$$Wx^{(i)} + b \leq -1 \quad \text{if } y = 0$$

Nonlinear SVM - Gaussian RBF Kernel

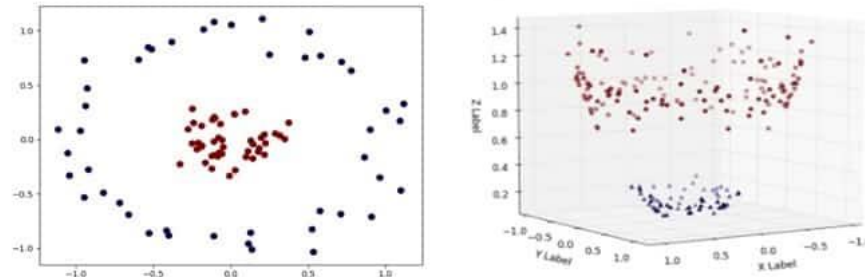
Nonlinearly separable data – kernel SVM



Kernel



Kernel Trick



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

Kernel: function which maps a lower-dimensional data into higher dimensional data.

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

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

σ – stand. deviation

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

Substitute the original features with similarity features (kernels).

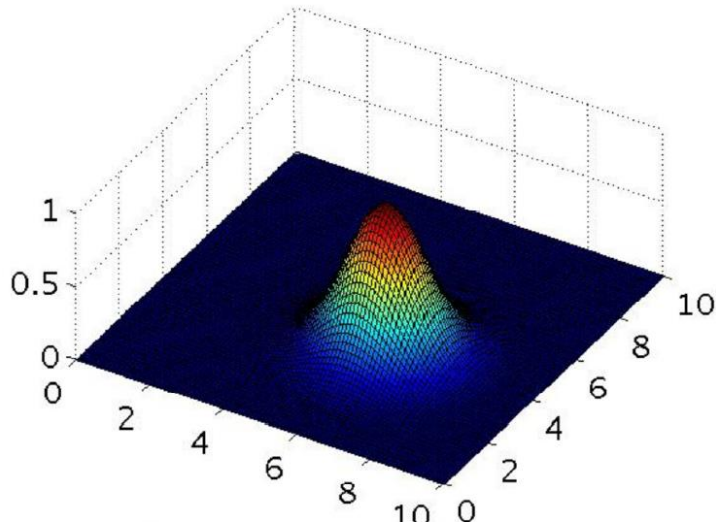
Note: the original ($n+1$ dimensional) feature vector is substituted by the new ($m+1$ dimensional) similarity feature vector.

m –number of examples, $m \gg n$!!!

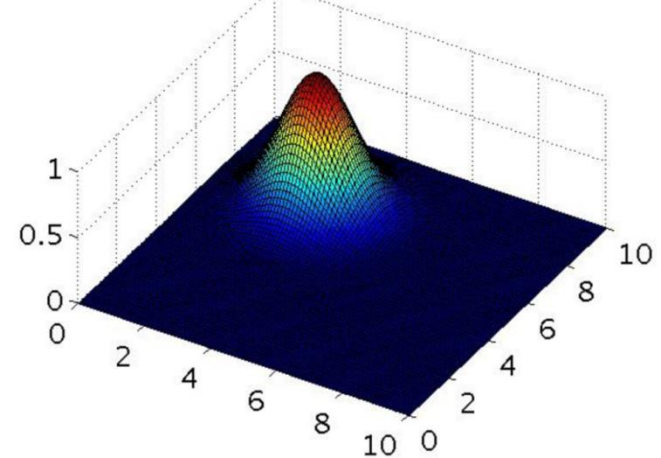
Gaussian RBF Kernel – Parameter σ

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

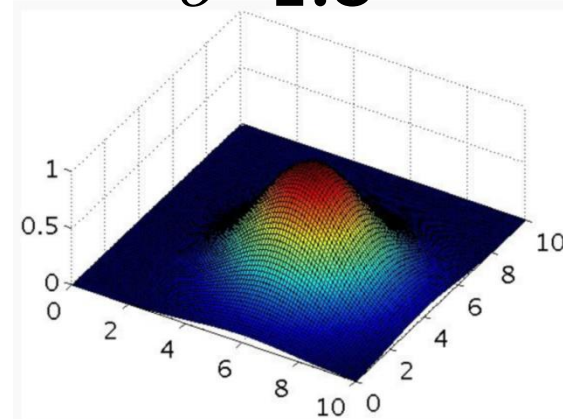
$\sigma=1$



$\sigma=0.5$



$\sigma=1.5$



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

Large σ : kernels vary more smoothly (combat overfitting)

Small σ : kernels vary less smoothly (more importance to training data fitting)

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

Use SVM software packages to solve SVM optimization !!!

In Python, use Scikit-learn (sklearn) machine learning library and

Import SVC (Support Vector Classification):

```
from sklearn.svm import SVC  
classifier = SVC(kernel="?", gamma=?, C=?)
```

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

Performance evaluation confusion matrix



Performance Evaluation – Confusion Matrix

ACTUAL CLASS	PREDICTED CLASS	
	Class=Yes	Class=No
Class=Yes	a (TP)	b (FN)
	c (FP)	d (TN)

a: TP (true positive)

b: FN (false negative)

c: FP (false positive)

d: TN (true negative)

Python: from sklearn.metrics import confusion_matrix

Performance metric - Accuracy

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

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

Accuracy - fraction of examples correctly classified.

1-Accuracy: Error rate (misclassification rate)

Limitation of Accuracy

- Consider binary classification (**Unbalanced data set**)
 - Class 0 has 9990 examples
 - Class 1 has 10 examples
- If model classify all examples as class 0, accuracy is $9990/10000 = 99.9 \%$
- Accuracy is misleading metric because model does not classify correctly any example of class 1
 - => Use other performance metrics.
 - => Find a way to balance the data set(re-sampling methods: oversampling, under-sampling)

Performance metrics from Conf Matrix

True Positive Rate (TPR), Sensitivity, Recall

of all positive examples the fraction of correctly classified
(ex. skin cancer)

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

True Negative Rate (TNR), Specificity

of all negative examples the fraction of correctly classified
(ex. spam/not spam emails)

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

False Positive Rate (FPR) - how often an actual negative instance will be classified as positive, i.e. “false alarm” (ex. cyber attack)

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

Precision - the fraction of correctly classified positive samples from all classified as positive

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

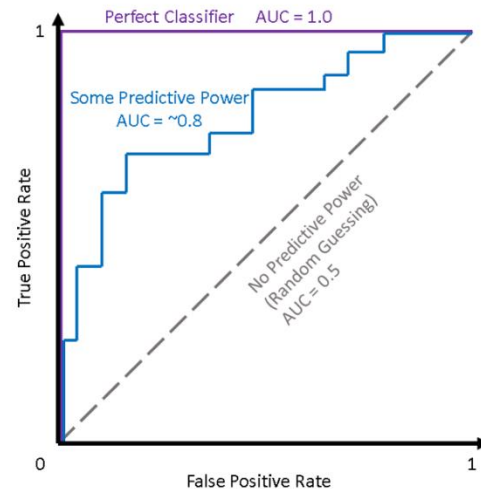
Combined performance metrics

F1 Score - weighted average of Precision and Recall

$$F1 = 2 * (\text{Recall} * \text{Precision}) / (\text{Recall} + \text{Precision})$$

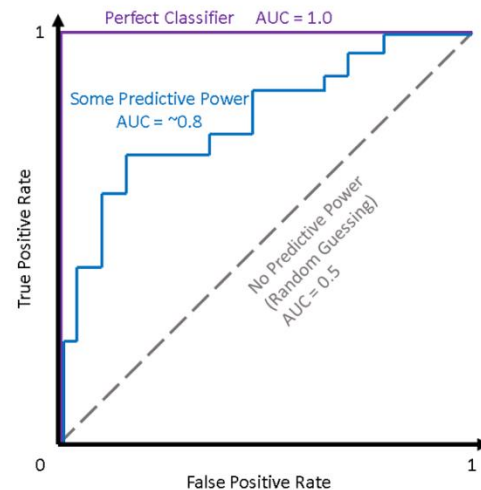
Balanced Accuracy = $(\text{Recall} + \text{Specificity}) / 2$

Receiver Operating Characteristic (ROC) curve



ROC curve is produced by calculating and plotting the **True Positive Rate (TPR)** against the **False Positive Rate (FPR)** 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. Normally in logistic regression, if an observation is predicted to be positive at > 0.5 probability, it is labeled as positive. However, we could really 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 the (ROC) Curve - AUC



ROC curve is useful for visualization, but it's good to have also a single metric => AUC. The higher the AUC score, the better a classifier performs for the given task. For a classifier with no predictive power (i.e., random guessing) => AUC = 0.5. For a perfect classifier => AUC = 1.0. Most classifiers fall between 0.5 and 1.0, with the rare exception being a classifier performs *worse* than random guessing (AUC < 0.5).

Python: from sklearn.metrics import auc

Performance metrics – example

	predicted	
	Positive	Negative
Positive	500	100
Negative	500	10000

- Accuracy $\frac{500+10000}{500+500+100+10000} = 0.95$
- Precision $\frac{500}{500+500} = 0.5$
- Recall $\frac{500}{500+100} = 0.83$
- Specificity $\frac{10000}{10000+500} = 0.95$

- Positive class is predicted poorly
- Accuracy is not a reliable measure for un-balanced datasets
- If # of examples of one class is much lower than # of examples of the other class => **F1 score and balanced accuracy are better measures.**

Class imbalance problem



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 pose a challenge for predictive modeling as most of the machine learning algorithms used for classification were designed around the assumption of an equal number of examples for each class. This results in models that have poor predictive performance, specifically for the minority class. This is a problem because typically, the minority class is more important and therefore the problem is more sensitive to classification errors for the minority class than 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

- 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:

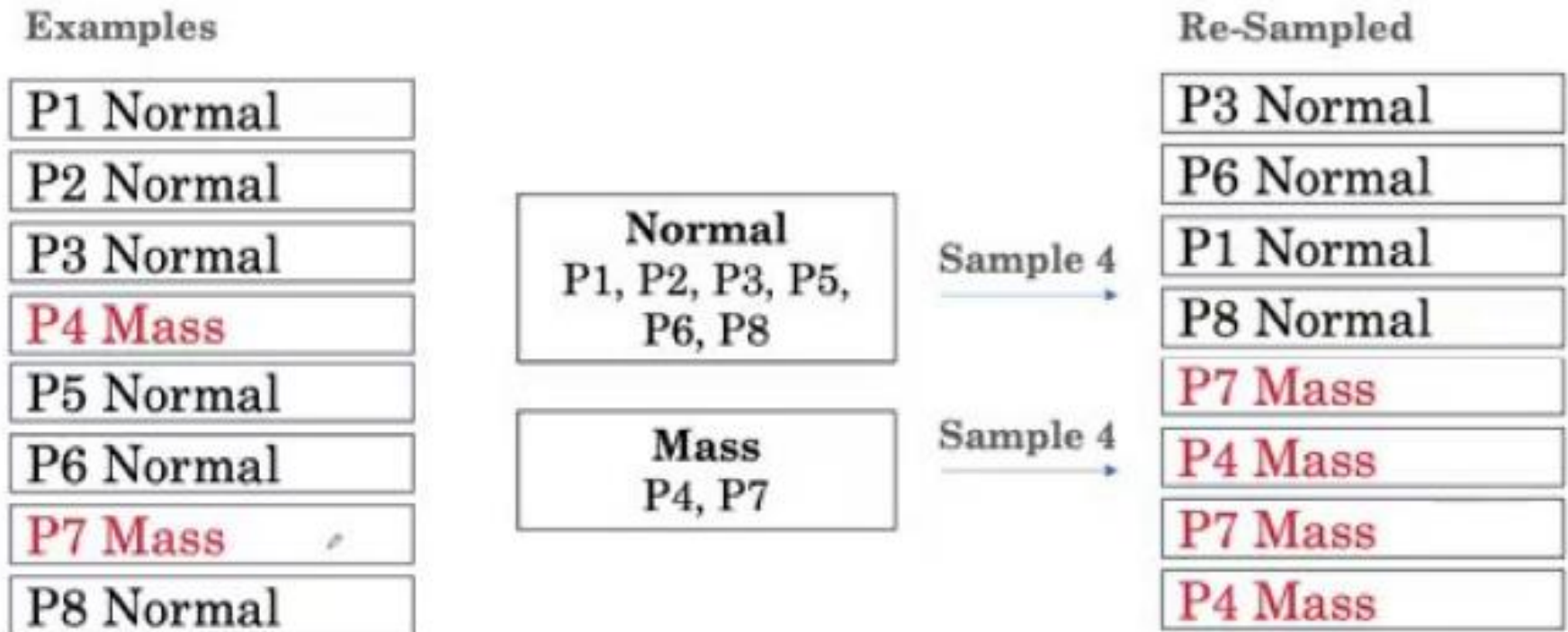
Classifier Level

- One-class classification

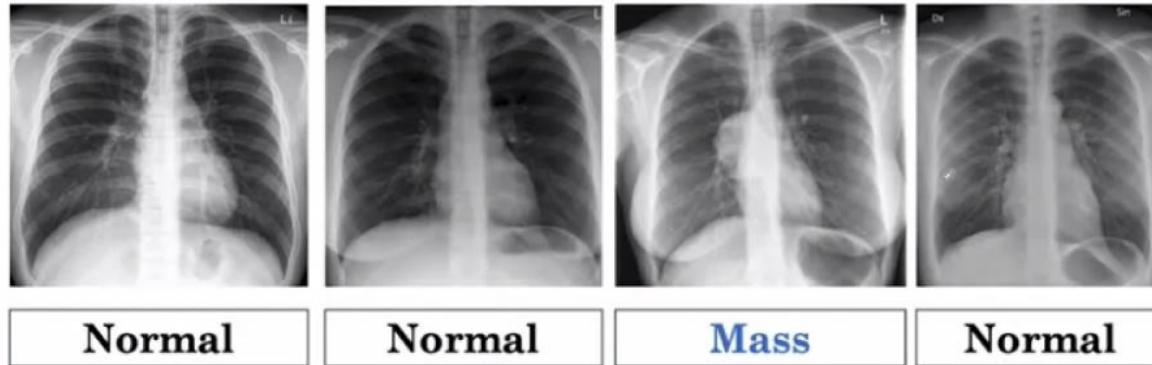
- It is the technique of handling class imbalance by modelling the distribution of only the minority class and treating all other classes as out-of-distribution/anomaly classes.
- Using this technique, we aim to create a classifier that can detect examples belonging to the minority class rather than 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.

Class Imbalance problem

Solution 1: Re-sampling methods (under-sampling, oversampling)



Class Imbalance problem



Solution 2: Weighted Binary Cross Entropy Loss

Weights:

$$w_p = \frac{\text{num negative}}{\text{num total}} \qquad w_n = \frac{\text{num positive}}{\text{num total}}$$

$$\mathcal{L}_{\text{cross-entropy}}^w(x) = -(w_p y \log(f(x)) + w_n (1 - y) \log(1 - f(x))).$$

Epoch / Batch Size / Iterations / Train step

One Epoch is when an ENTIRE dataset is passed through the model (e.g. forward and backward in a neural network) 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.

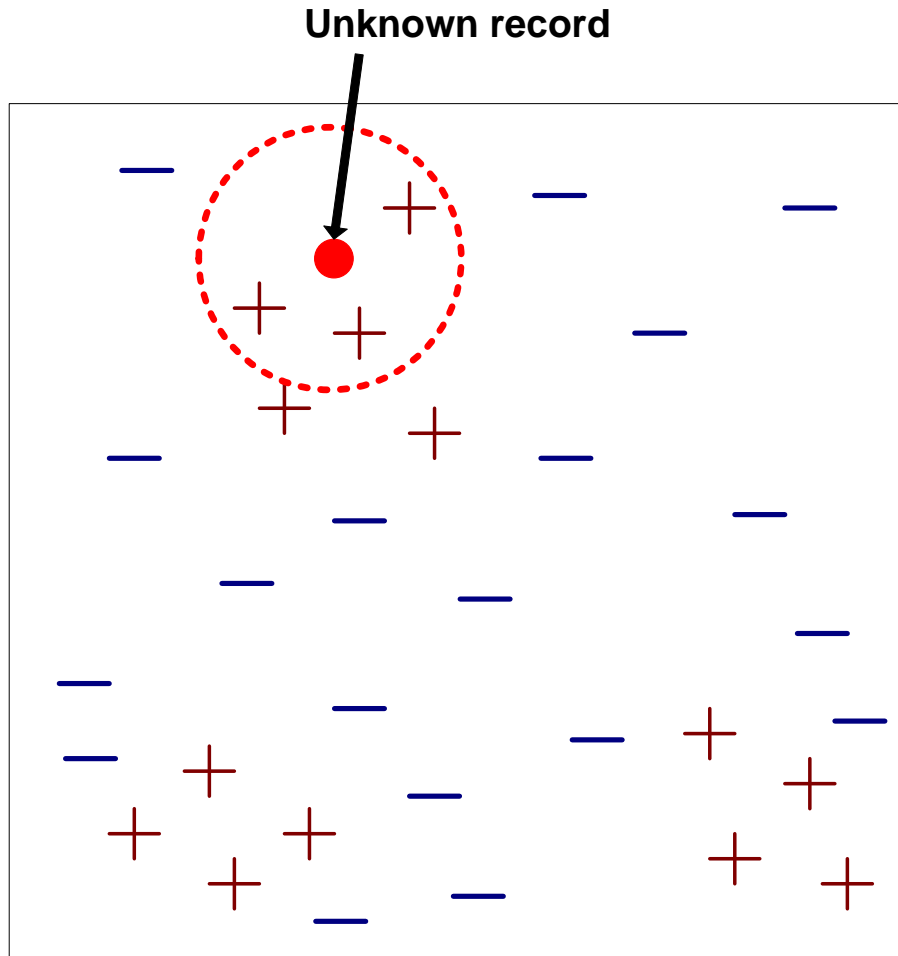
Iterations is the number of batches needed to complete one epoch.

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.

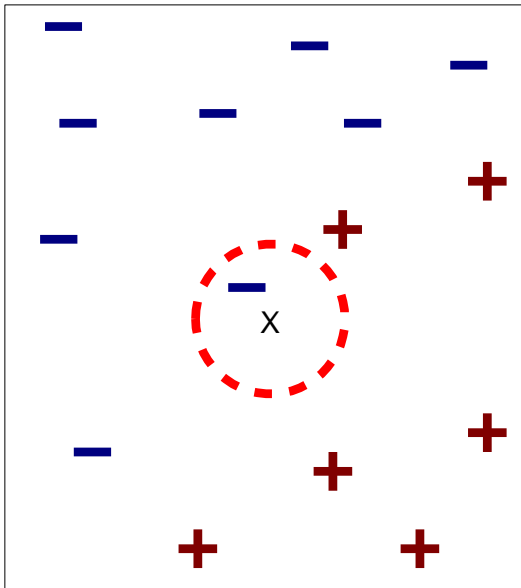
k-Nearest Neighbor (k-NN) classifier

K- Nearest-Neighbor (kNN) Classifier

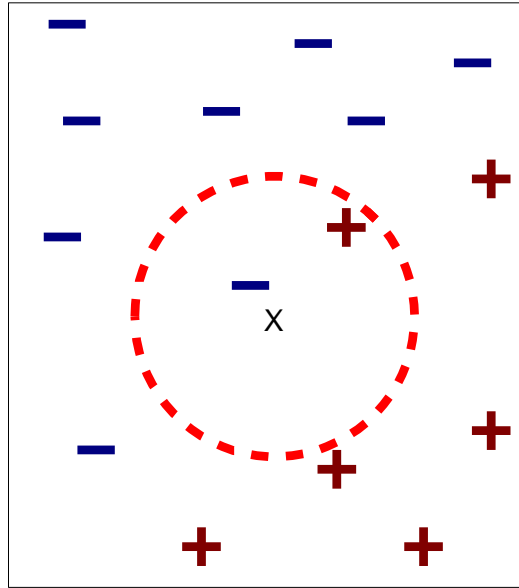


- KNN requires:
 - Set of labeled records.
 - Measure to compute distance (similarity) between records.
 - K is the number of nearest neighbors (the closest points).
- To classify a new (unlabeled) record:
 - Compute its distance to all labeled records.
 - Identify k nearest neighbors.
 - The class label of the new record is the label of the majority of the nearest neighbors.

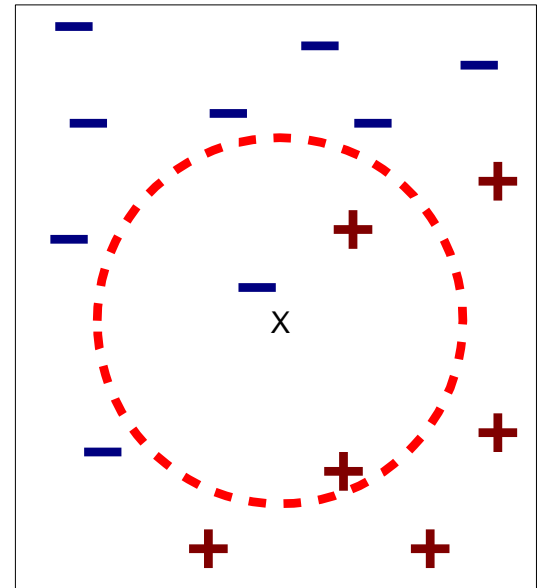
kNN- choice of k



(a) 1-nearest neighbor



(b) 2-nearest neighbor

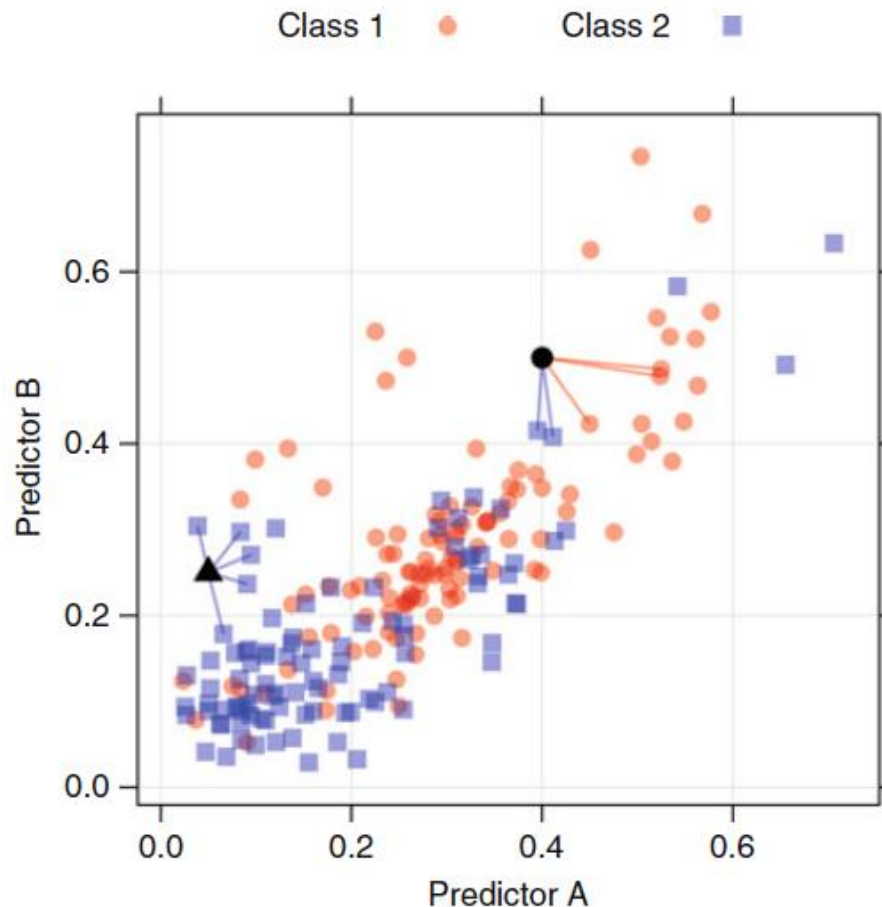


(c) 3-nearest neighbor

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

Classification – K-NN

- 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.



Classification – K-NN

- Because the KNN method fundamentally depends on 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 local predictor structure is not relevant to the response. Irrelevant or noisy predictors are one culprit, since these can cause similar samples to be driven away from each other in the predictor space. Hence, removing irrelevant, noise-laden predictors is a key pre-processing step for KNN.

