ieeta instituto de engenharia electrónica e telemática de aveiro

universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

LECTURE 5:
SUPPORT VECTOR MACHINE (SVM)

Petia Georgieva
(petia@ua.pt)

LECTURE outline

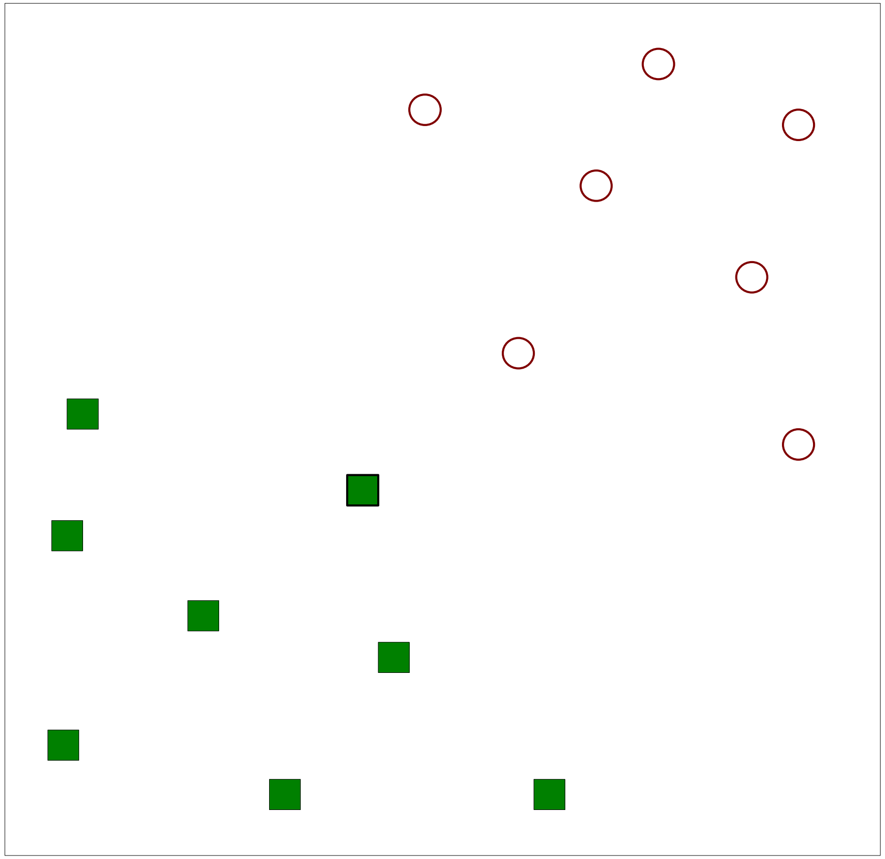
1. **Linear Support Vector Machine (SVM)**

2. **Nonlinear SVM - Gaussian RBF Kernel**

3. **Performance evaluation – confusion matrix**

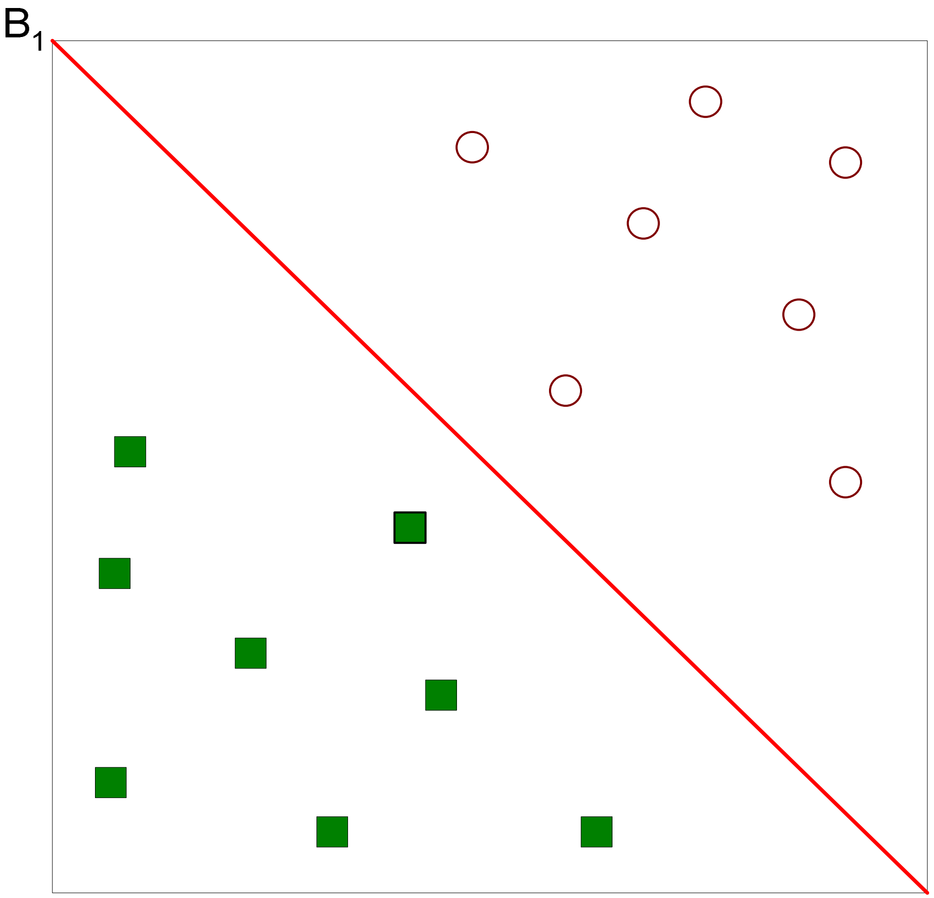
4. **Class imbalance problem**

Linearly separable classes



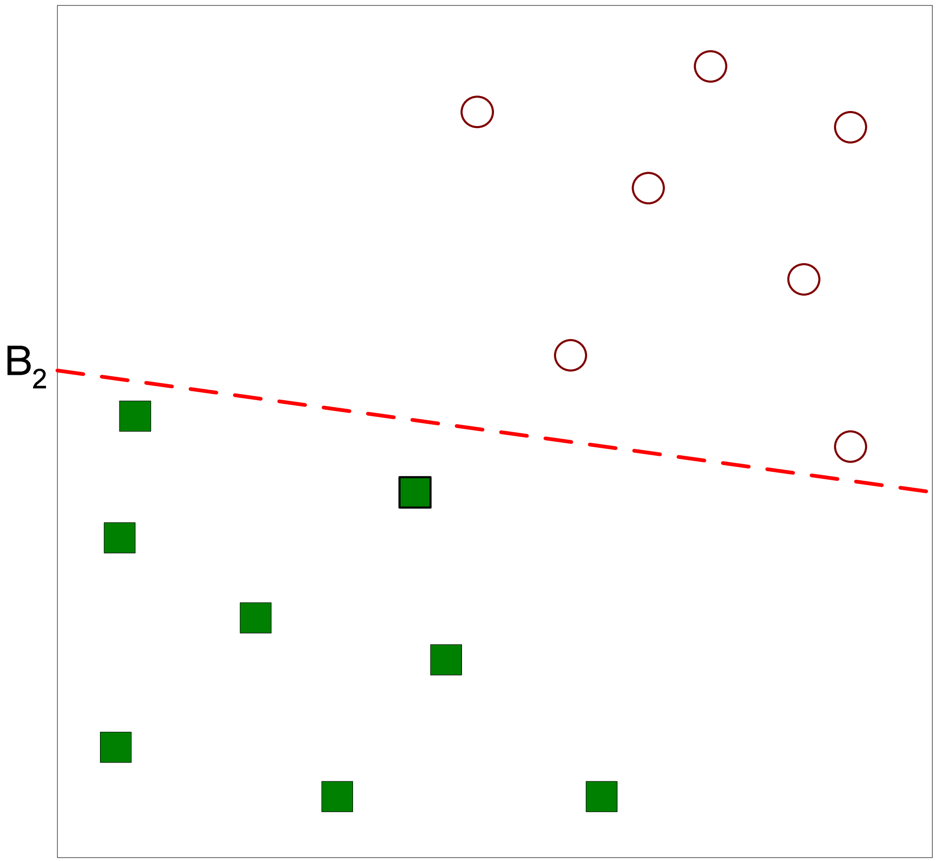
Find a decision boundary to separate data

Linearly separable classes



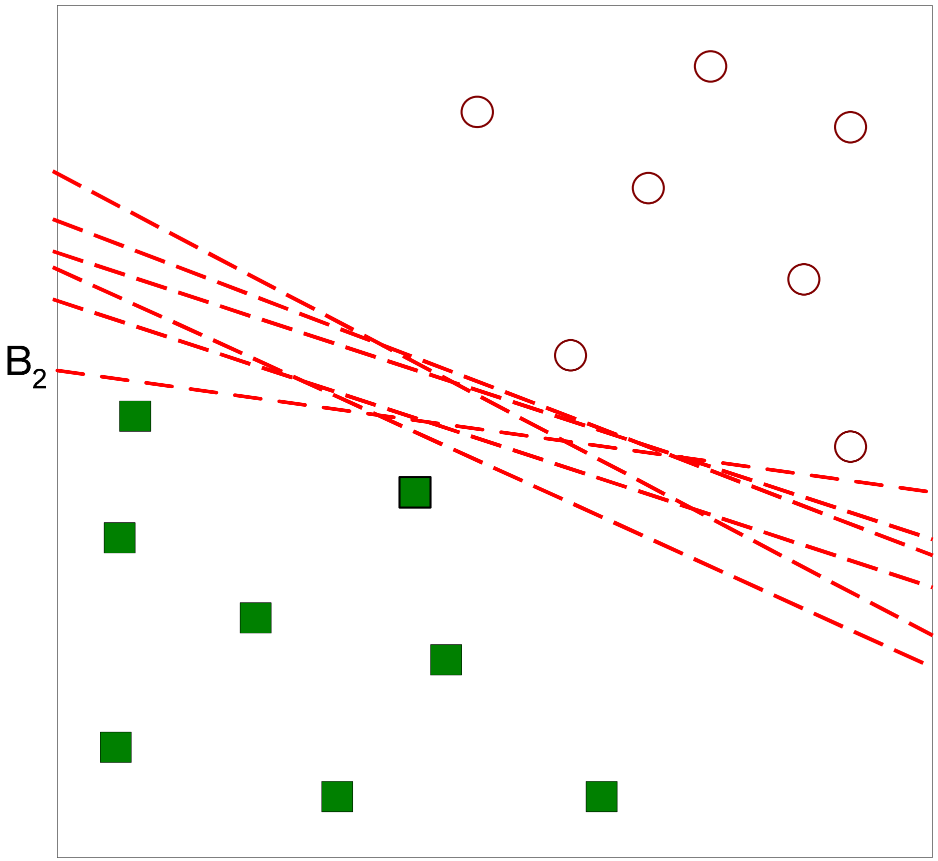
One Possible Solution

Linearly separable classes



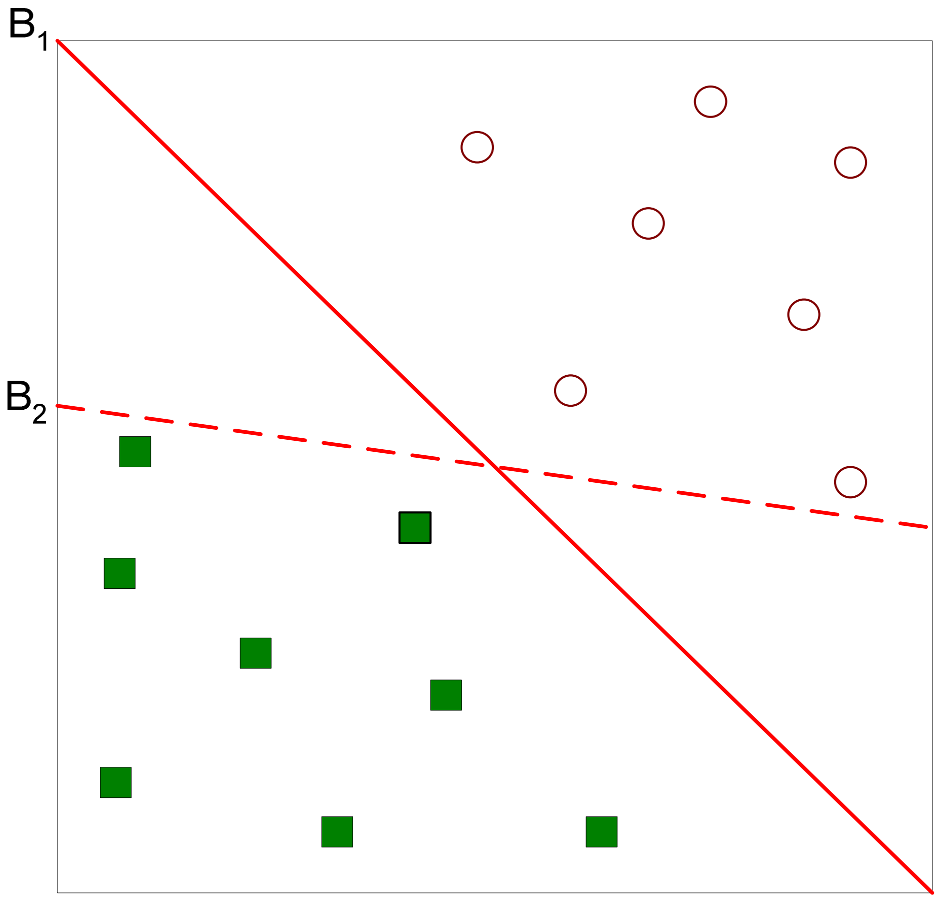
Another possible solution

Linearly separable classes



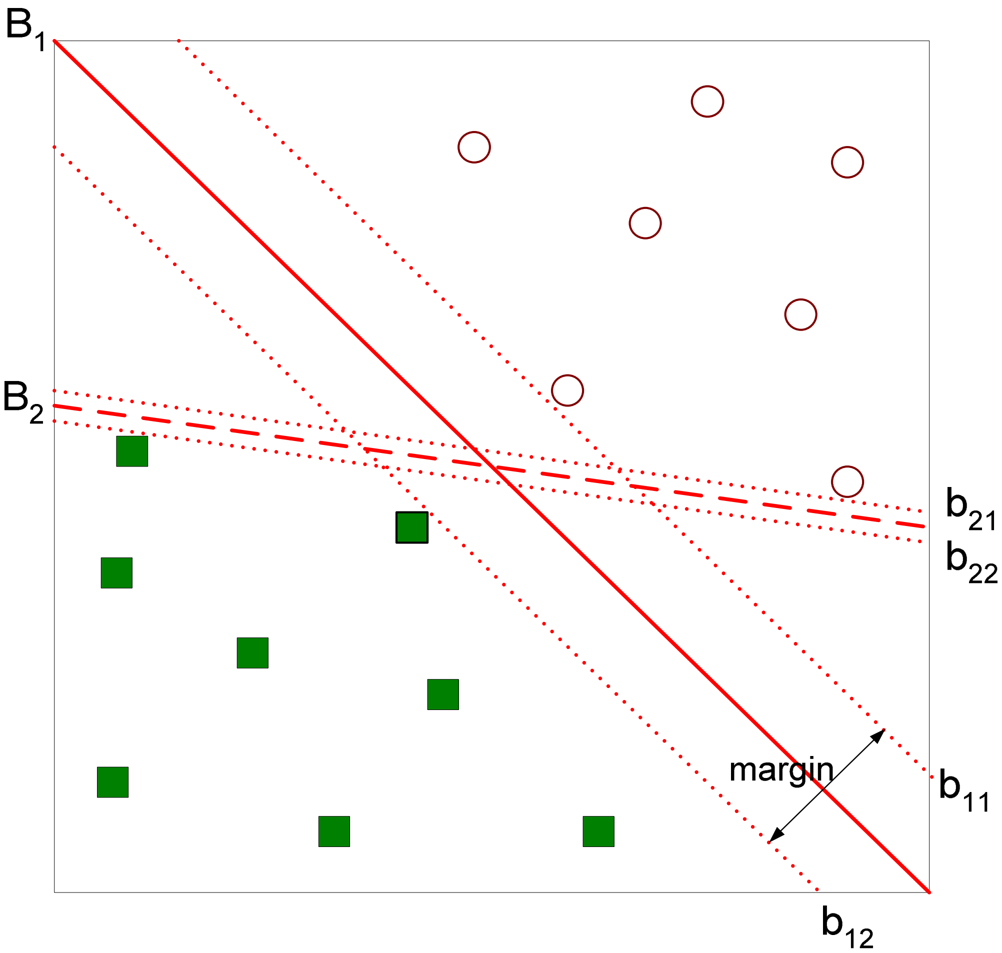
Many possible solutions

Linearly separable classes



Which one is better? B1 or B2?

SVM - Large margin classifier

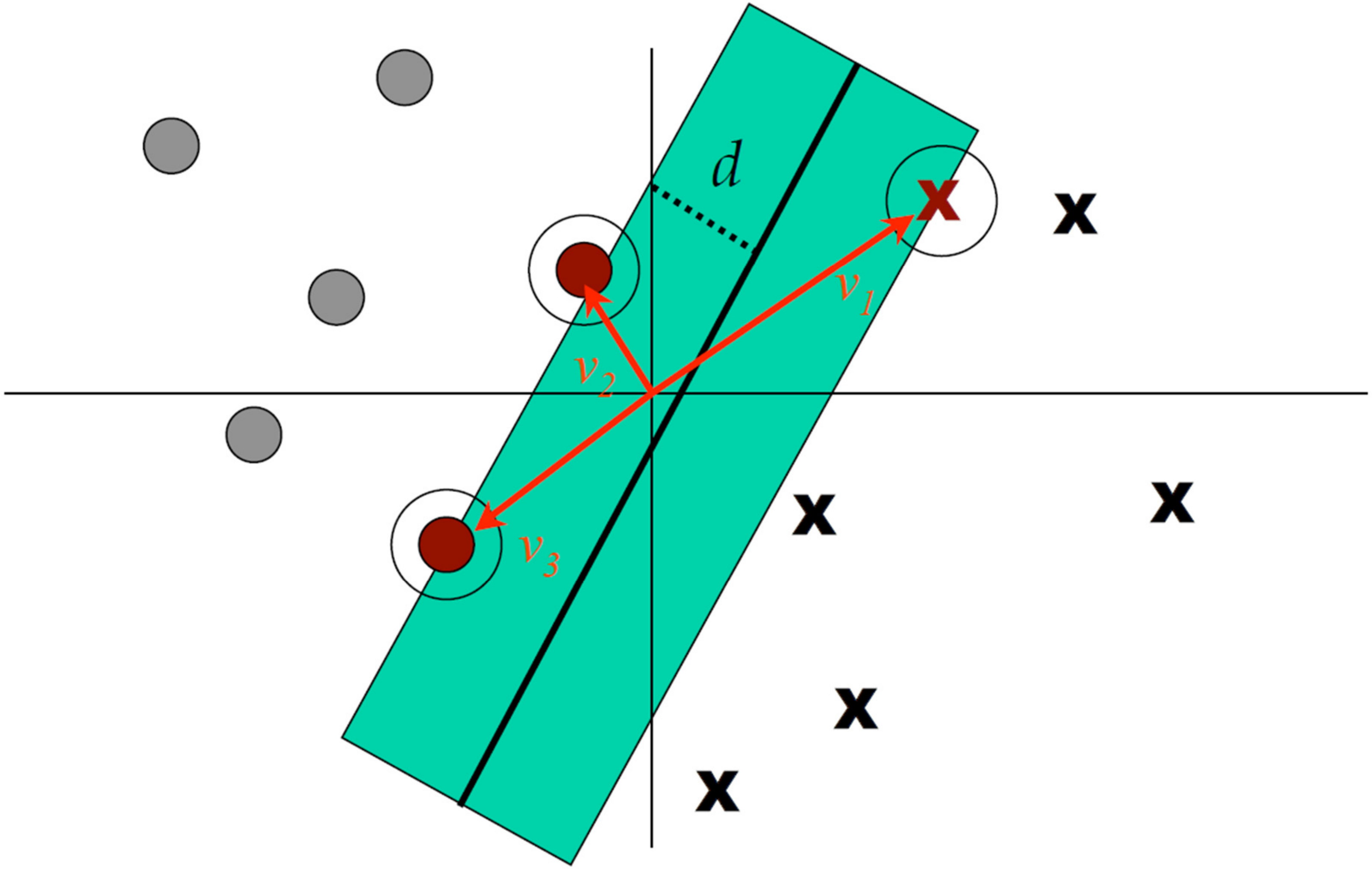


Find a boundary that maximizes the margin => B1 is better than B2

Proposed by Vladimir N. Vapnik and Alexey Chervonenkis, 1963

SUPPORT VECTORS (v1,v2,v3)

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

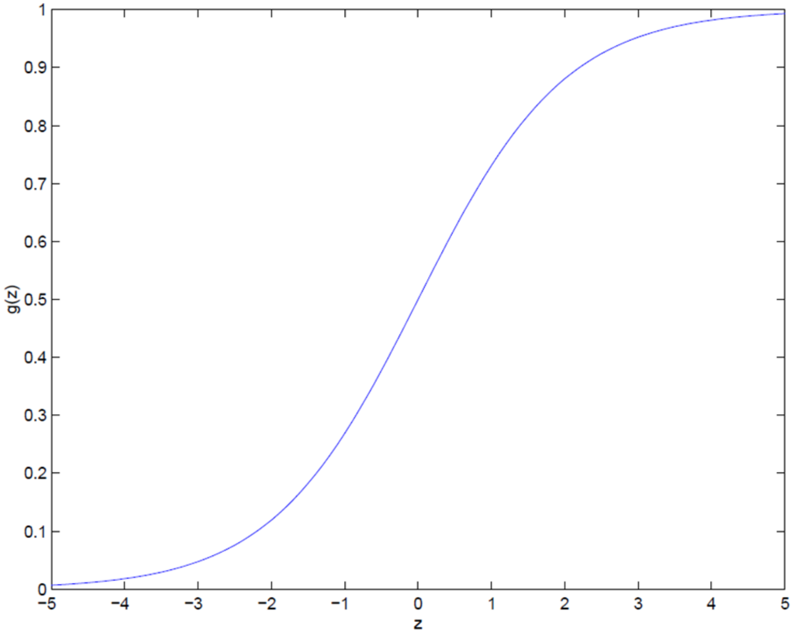


Logistic Regression (LogReg) -revised

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

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

Logistic (sigmoid) function



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

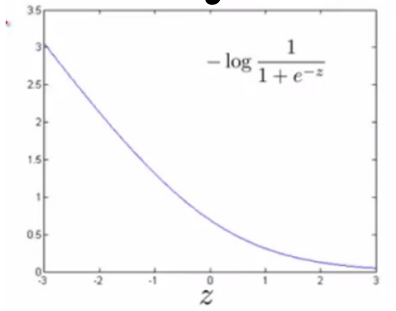
$$\text{if } \; y = 0, \text{we want } h_\theta(x) \approx 0, \quad \theta^T x << 0$$

SVM cost function

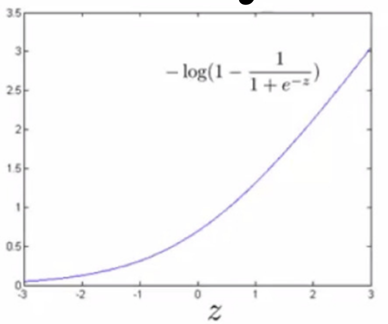
Regularized LogReg cost function:

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

for y=1

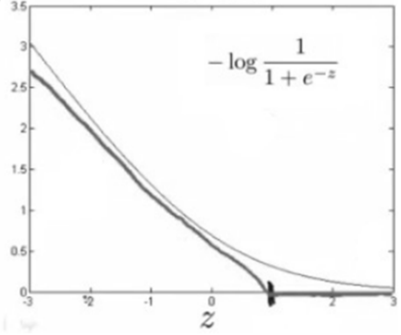


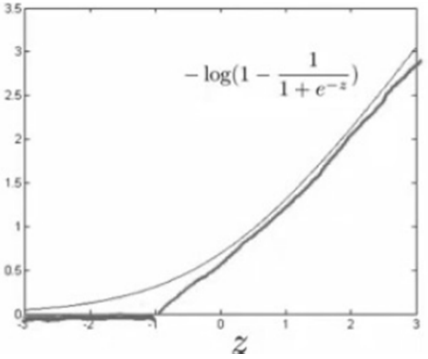
for y=0



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

$$\min_{\theta} C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$





$$z = \theta^T x$$

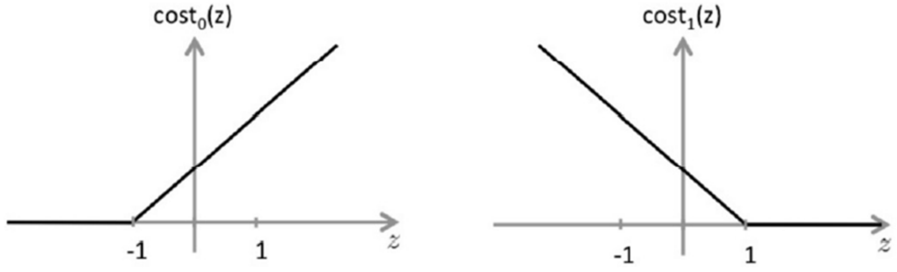
SVM cost function

Regularized LogReg cost function:

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

Regularized SVM cost function

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$



$$z = \theta^T x$$

Different way of parameterization: C is equivalent to 1/λ.

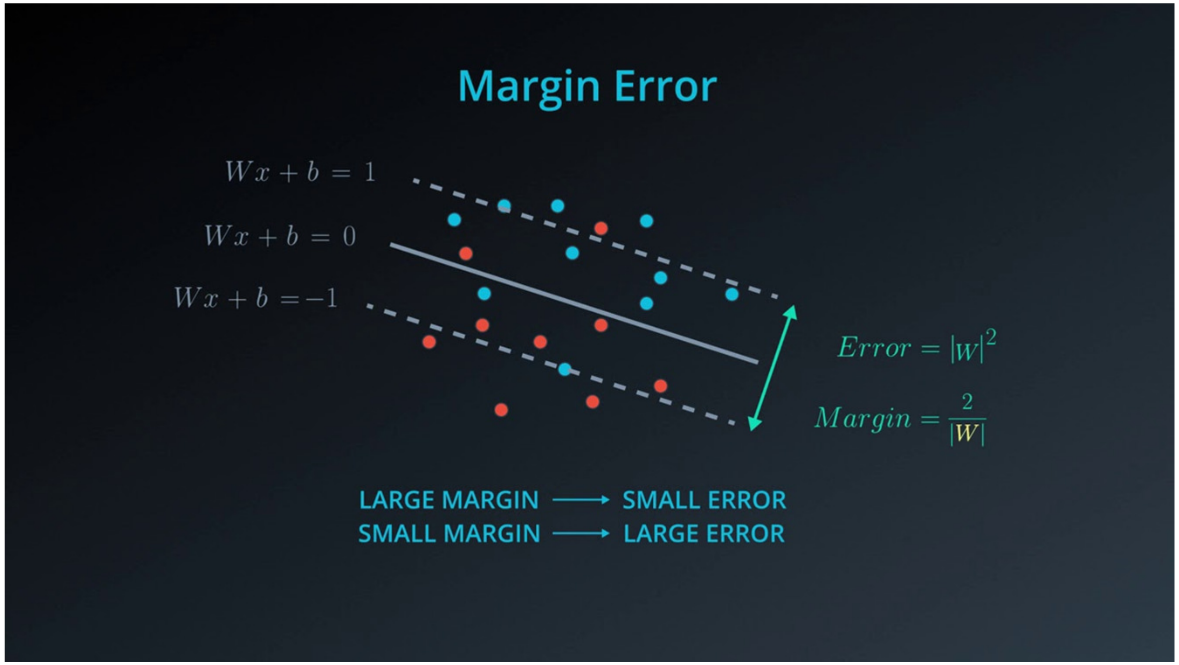
C >0 - parameter that controls the penalty for misclassified training examples.
Increase C more importance to training data fitting.
Decrease C – more importance to generalization properties (combat overfitting).

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”.



$$\theta^T x => Wx + b$$

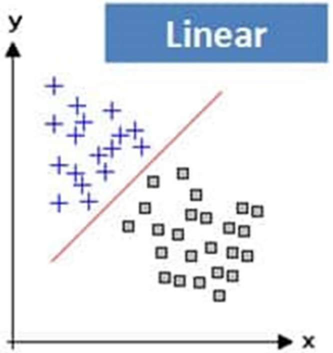
$$\min_W \sum_{j=1}^{n} W_j^2 = |W|^2$$

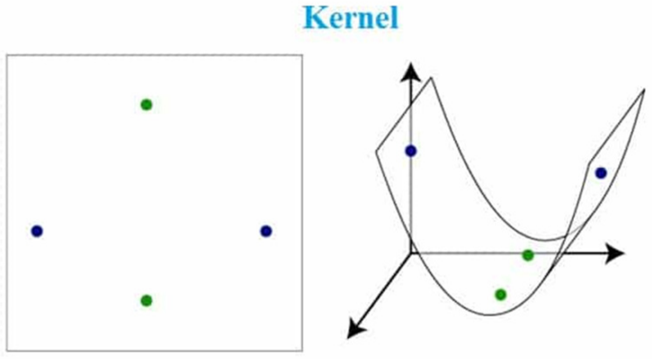
$(L_2$ norm) such that

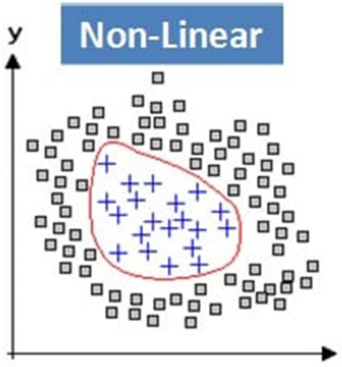
$$Wx^{(i)} + b \geq 1, \quad \text{if } y = 1$$

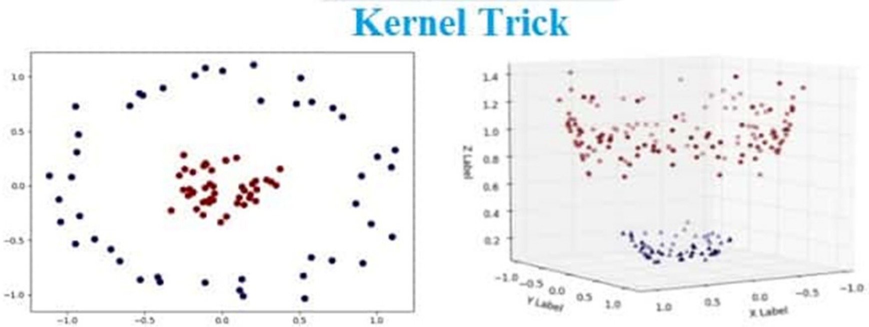
$$Wx^{(i)} + b \leq -1 \quad \text{if } y = 0$$

Nonlinearly separable data – kernel SVM









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

Tipical Kernels:

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

Nonlinear SVM – Gaussian RBF Kernel

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

$$\sigma - \text{stand. deviation}$$

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).

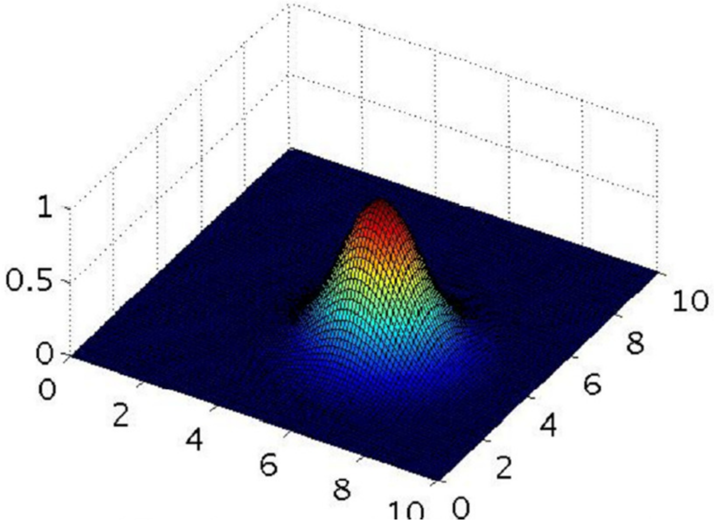
Note: the original (n+1 dimensional) feature vector is substituted by the new (m+1 dimensional) similarity feature vector.

m –number of examples, **m>>n !!!**

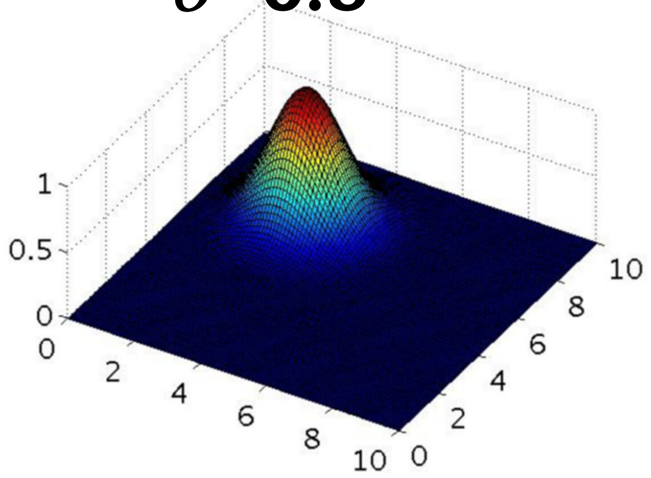
Gaussian RBF Kernel – Parameter σ

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

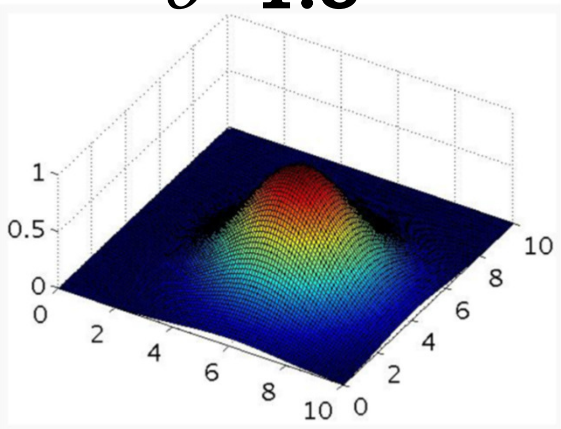
σ**=1**



σ**=0.5**



σ**=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/σ.

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(

Performance Evaluation – Confusion Matrix

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	Class=Yes	a (TP)	b (FN)
	Class=No	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

	PREDICTED CLASS		
ACTUAL CLASS		Class=Yes	Class=No
	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)

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 metrics 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

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

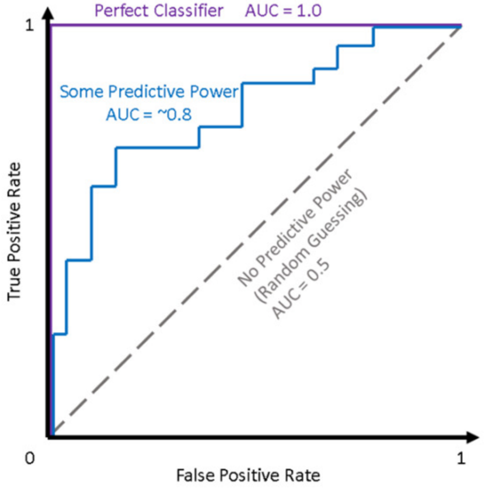
Combined performance metrics

F1 Score - weighted average of Precision and Recall

F1=2*(Recall * Precision) / (Recall + Precision)

Balanced Accuracy= (Recall+Specificity)/2

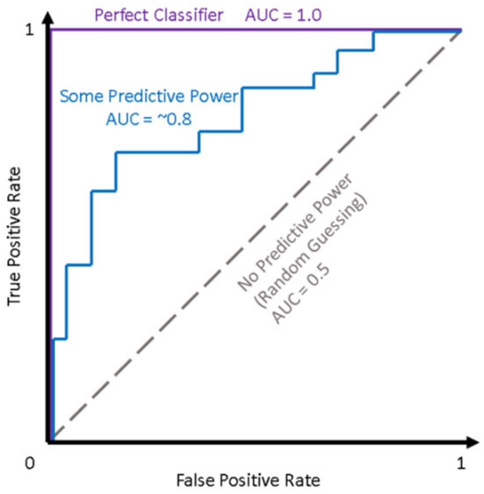
Receiver Operating Characteristic (ROC) curve



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.

Python: from sklearn.metrics import roc_curve

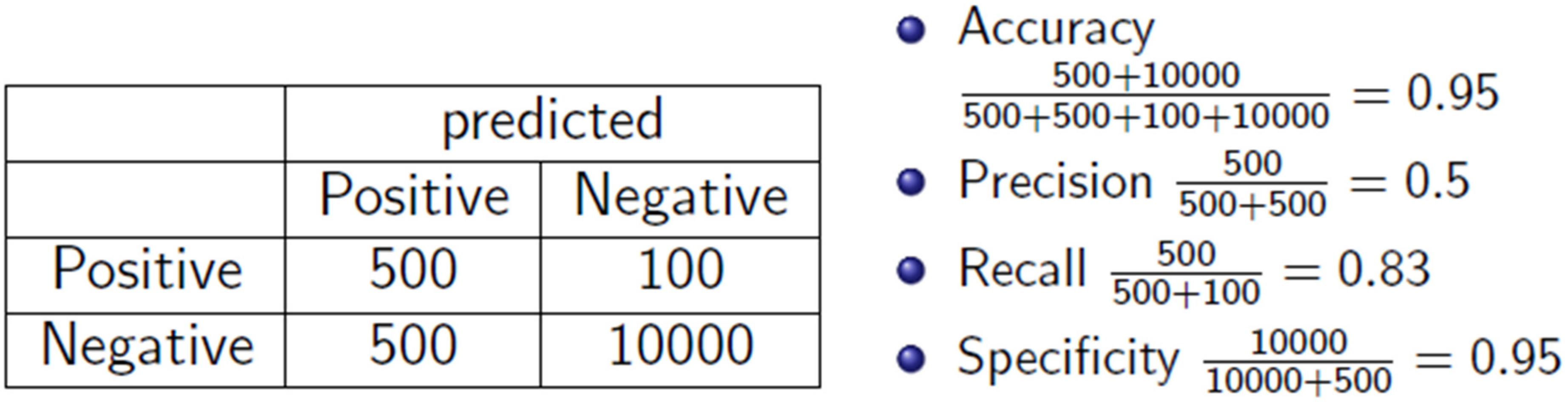
Area Under the (ROC) Curve - AUC



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



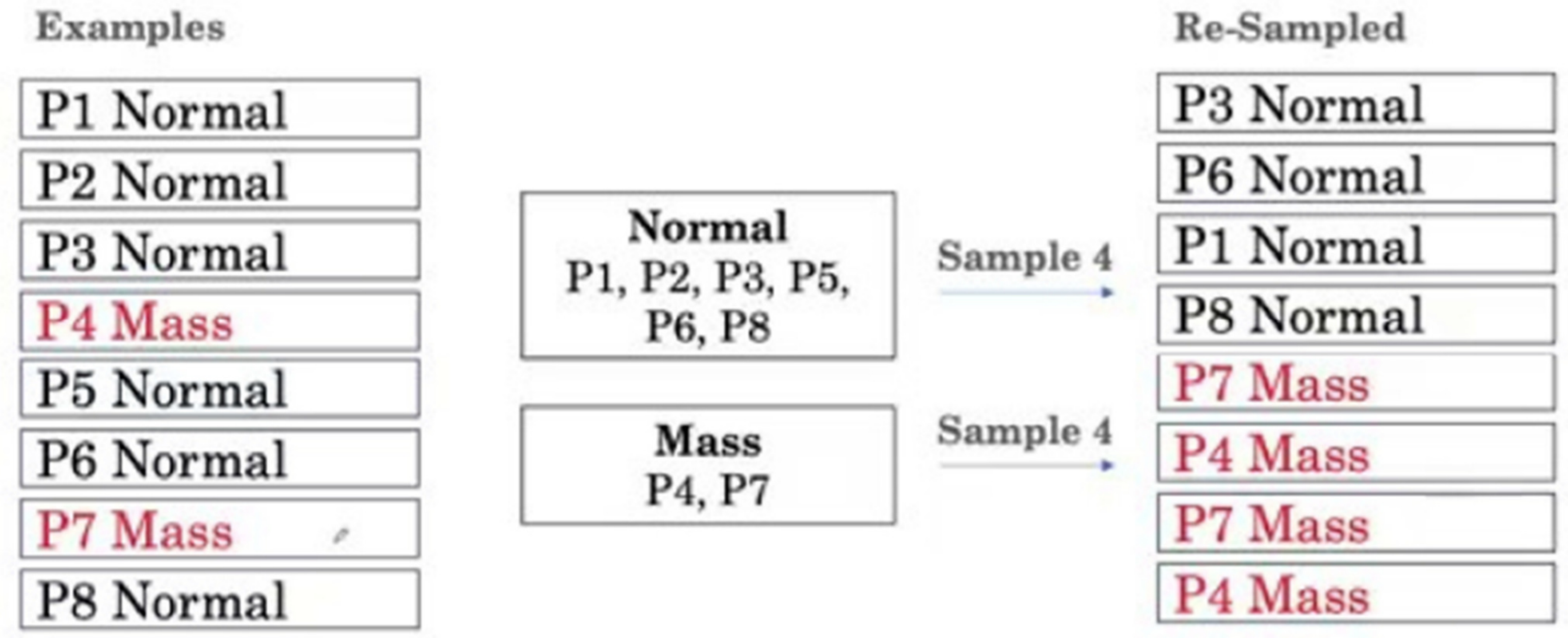
	predicted	
	Positive	Negative
Positive	500	100
Negative	500	10000

- 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

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



Examples

- P1 Normal
- P2 Normal
- P3 Normal
- P4 Mass
- P5 Normal
- P6 Normal
- P7 Mass
- P8 Normal

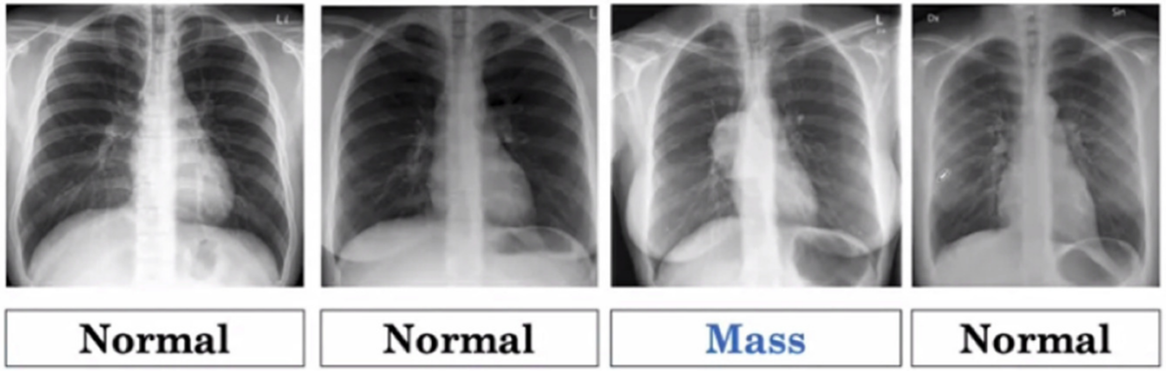
Normal
P1, P2, P3, P5,
P6, P8

Mass
P4, P7

Re-Sampled

- P3 Normal
- P6 Normal
- P1 Normal
- P8 Normal
- P7 Mass
- P4 Mass
- P7 Mass
- P4 Mass

Class Imbalance problem



Solution 2: Weighted Binary Cross Entropy Loss

Weights:

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

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

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 examples present in a single batch.

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

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

Training run/step - is one update of the model parameters.
We update the parameters after one batch or after one epoch.