SAPIENZA
UNIVERSITÀ DI ROMA

METODI DI INTELLIGENZA ARTIFICIALE E MACHINE LEARNING IN FISICA

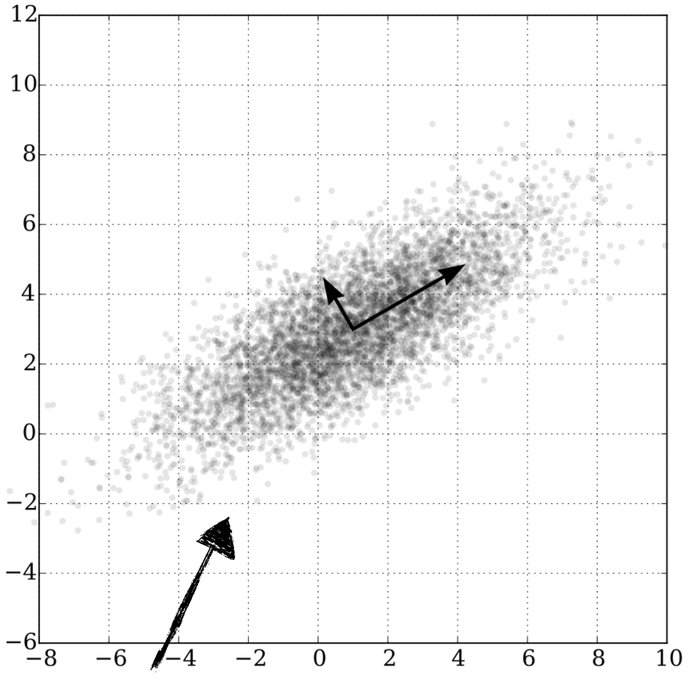
S. Giagu - AA 2019/2020

Lezione 8: 19.3.2020

ANALISI A COMPONENTI PRINCIPALI E FDL DI FISHER

- l'analisi a componenti principali (PCA) è un noto metodo di riduzione dimensionale per determinare la rappresentazione più accurata di più campioni D-dimensionali in uno spazio a dimensione ridotta d<D

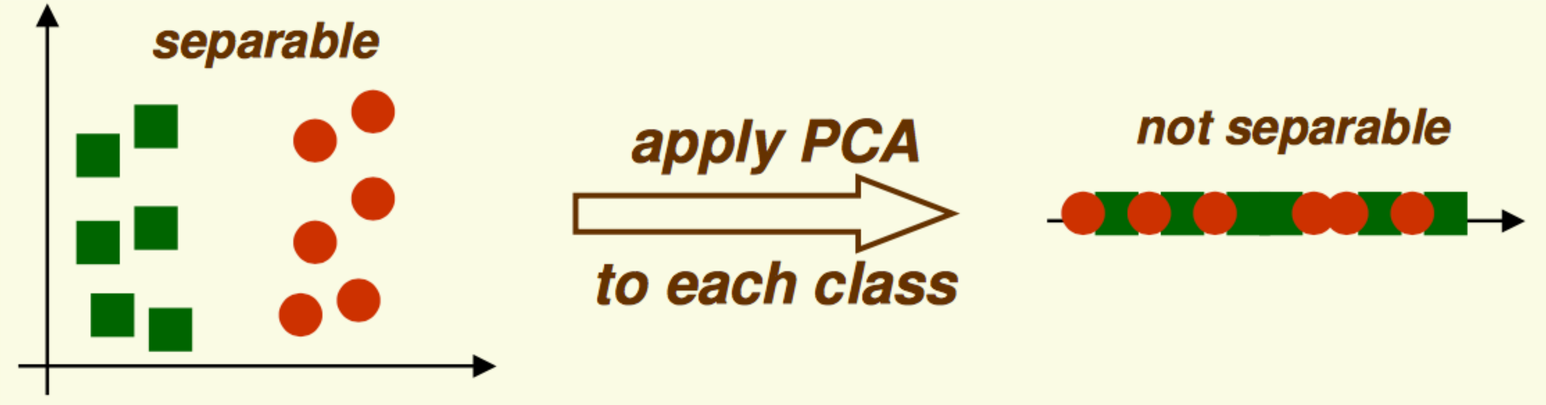
- l'idea è quella di combinare le feature $x_1 \ldots x_D$ in un un numero inferiore di feature latenti limitando il più possibile la perdita di informazioni
- ad esempio potremmo scegliere m tale che la somma delle distanze tra m e le varie x_K sia la più piccola possibile:
  - $m = \langle x \rangle = 1/N \, \Sigma x_i \rightarrow$ rappresentazione 0-dimensionale
  - la media è sicuramente un parametro utile ma chiaramente si sta scartando troppa informazione



- per mantenere informazione anche sullo spread dei campioni si proiettano i dati su un nuovo sistema cartesiano centrato nel punto m e in cui gli assi rappresentano le direzioni di massima varianza, seconda massima varianza, etc… del campione e si sceglie di rappresentare il campione con le prime d di tali variabili

ANALISI A COMPONENTI PRINCIPALI E FDL DI FISHER

- ci possiamo chiedere se rappresentare i dati nello spazio latente PCA possa essere una buona opzione in problemi di classificazione
- risposta: NO, PCA risulta inutile e spesso dannosa in un problema di classificazione:

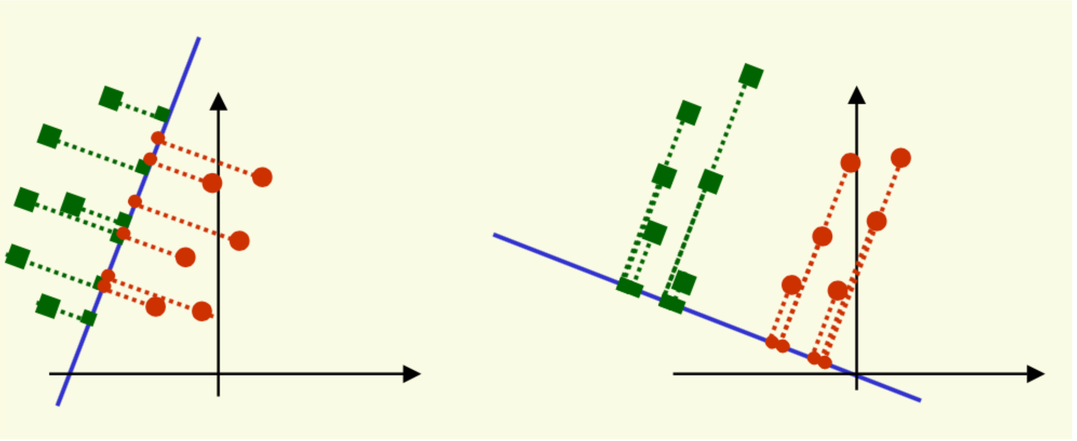


Il discriminate lineare di Fisher rappresenta una estensione della tecnica PCA in cui le direzioni di proiezione vengono scelte in modo tale da massimizzare il potere di classificazione tra le due classi

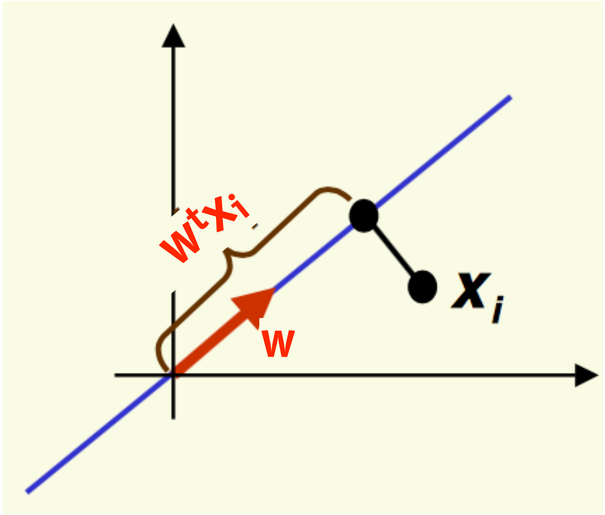
FDL DI FISHER

- idea principale: trovare la direzione **w** per cui i campioni proiettati su tale direzione sono ben separati

- supponiamo di avere 2 classi d-dimensionali e $n=n_1$ (classe 1) + n_2 (classe 2) eventi $x_1 \ldots x_n$
- consideriamo la proiezione sulla retta definita dal versore **w**



mal separati ben separati



- il prodotto scalare $y_i = w^t x_i$ rappresenta la proiezione di x_i dall'origine

- $w^t x_i$ rappresenta quindi la proiezione di x_i in un sotto-spazio a dimensione ridotta (d-1)

- siano m_1 e m_2 i valori medi delle distribuzioni delle due classi nello spazio originale → i valori proiettati saranno:

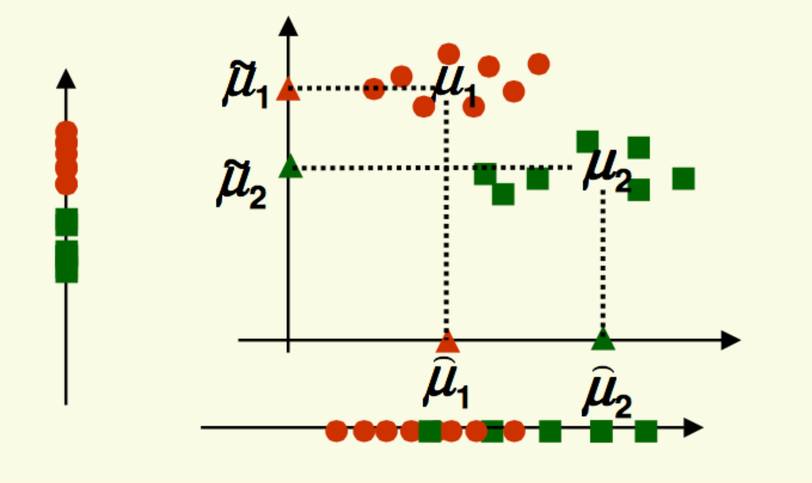
  - $\mu_i = 1/n_i \Sigma\, y_i = 1/n_i \Sigma\, w^t x_i = w^t\, (1/n_i \Sigma\, x_i) = w^t m_i$

  - μ_i è quindi semplicemente dato dalla proiezione di m_i

- strategia di Fisher: usiamo $|\mu_i - \mu_i|$ come misura della separazione tar i campioni per trovare la direzione **w**

FDL DI FISHER

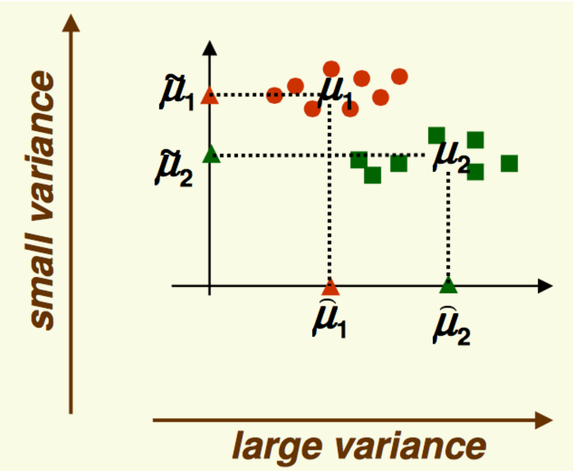
- quanto è buono $|\mu_1 - \mu_2|$ come criterio per la stima della separazione?
- a prima vista ci aspetteremmo che più grande sia $|\mu_1 - \mu_2|$ maggiore è la separazione → OK



tuttavia nell'esempio:

- l'asse verticale separa meglio di quello orizzontale
- ma $|\tilde{\mu}_1 - \tilde{\mu}_2| < |\hat{\mu}_1 - \hat{\mu}_2|$

- il problema stà nel fatto che $|\mu_1 - \mu_2|$ non tiene in conto la varianza delle due classi nelle due direzioni



- si risolve il problema normalizzando $|\mu_1 - \mu_2|$ a qualche cosa che sia proporzionale alle varianze di ciascuna classe (misura di quanto i dati sono raggruppati intorno alla media)
- $s_i^2 = \Sigma (x_i - m_i)^2$ → nello spazio trasformato: $\sigma_i^2 = \Sigma (y_i - \mu_i)^2$
- definiamo la varianza per classe totale del campione come: $\sigma^2 = \sigma_1^2 + \sigma_2^2$

FDL DI FISHER

- Criterio di Fisher: nella procedura di ottimizzazione viene massimizzata la distanza tra le medie delle due classi e minimizzata contemporaneamente la varianza all'interno di ciascuna classe

$$J(\mathbf{w}) = \frac{|\mu_1 - \mu_2|^2}{\sigma^2} = \frac{|\mathbf{w}^t(\mathbf{m}_1 - \mathbf{m}_2|^2}{\mathbf{w}^t\mathbf{S}_w\mathbf{w}}$$

$$\mathbf{m}_i = \frac{1}{n_i}\sum_{x \in T_i} \mathbf{x}$$

$$\mathbf{S}_w = \sum_{i=1}^{2}\sum_{x \in T_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^t$$

Sw somma delle matrici di covarianza in ciascuna classe

- **w** che massimizza J garantisce prestazioni ottimali in separazione tra le due classi per variabili gaussiane con correlazioni lineari
- NOTA: se una variabile ha stesso valore medio nelle due classi → nessuna separazione → spesso conviene usare |var|

Calcolando il gradiente di J rispetto a **w** per trovare il massimo si ottiene:

$$\mathbf{w} = \frac{(\mathbf{m}_1 - \mathbf{m}_2)}{\mathbf{S}_w}$$

FDL di Fisher

ESEMPI CLASIFICATORI LINEARI IN SCIKIT-LEARN

https://www.dropbox.com/s/hy79h33vvhd7tkd/Perceptron_and_FDL_classification.ipynb?dl=0

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import Perceptron
from sklearn import datasets
from sklearn.model_selection import train_test_split


# IRIS dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # take only the first two features
y = iris.target

X_train, X_test, y_train, y_test = \
        train_test_split(X, y, test_size=.5, random_state=1111, shuffle=True)
```

IRIS dataset

```
usePerc = False

if usePerc == True:
    # preceptron
    # stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol)
    logreg = Perceptron(tol=1e-7, random_state=0)
else:
    #Fsher LDA
    logreg = LinearDiscriminantAnalysis()


# Create an instance of Logistic Regression Classifier and fit the data.
logreg.fit(X_train, y_train)
```

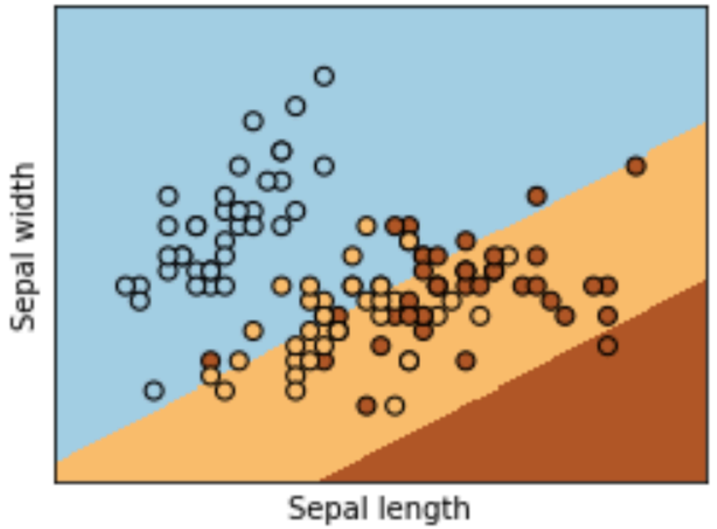
perceptron

LDA (Fisher)

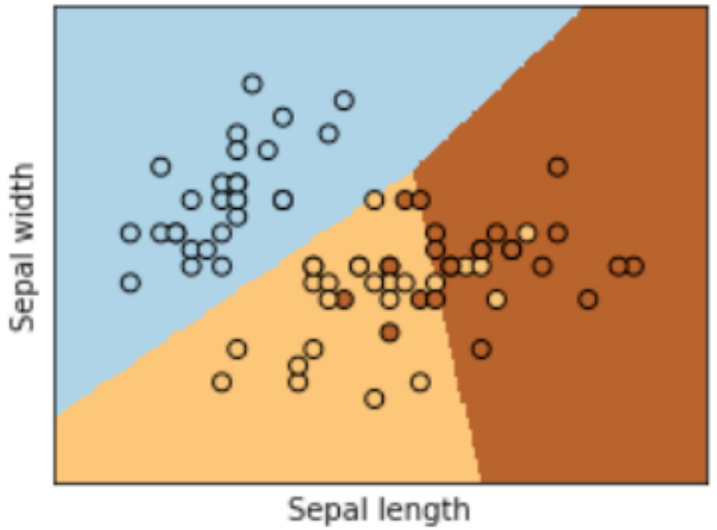
ESEMPI CLASIFICATORI LINEARI IN SCIKIT-LEARN

```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X_train[:, 0].min() - .5, X_train[:, 0].max() + .5
y_min, y_max = X_train[:, 1].min() - .5, X_train[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```

perceptron



LDA



FISHER E CRITERIO MSE

- Si dimostra che Fisher è un caso speciale del criterio MSE (Duda et Al.):

2 classi, n_1 eventi T_1 e n_2 eventi T_2 $n=n_1+n_2$, si definiscono:

$$\mathbf{Z} = \begin{bmatrix} \mathbf{1}_1 & \mathbf{X}_1 \\ -\mathbf{1}_2 & -\mathbf{X}_2 \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} \frac{n}{n_1}\mathbf{1}_1 \\ \frac{n}{n_2}\mathbf{1}_2 \end{bmatrix} \qquad \mathbf{m}_i = \frac{1}{n_i}\sum_{x \in T_i} \mathbf{x} \qquad \mathbf{S}_w = \sum_{i=1}^{2}\sum_{x \in T_i}(\mathbf{x}-\mathbf{m}_i)(\mathbf{x}-\mathbf{m}_i)^t$$

$$\mathbf{Zw} = \mathbf{b} \rightarrow \mathbf{Z}^t\mathbf{Zw} = \mathbf{Z}^t\mathbf{b} \longrightarrow \begin{bmatrix} \mathbf{1}_1^t & -\mathbf{1}_2^t \\ \mathbf{X}_1^t & -\mathbf{X}_2^t \end{bmatrix} \begin{bmatrix} \mathbf{1}_1 & \mathbf{X}_1 \\ -\mathbf{1}_2 & -\mathbf{X}_2 \end{bmatrix} \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} \mathbf{1}_1^t & -\mathbf{1}_2^t \\ \mathbf{X}_1^t & -\mathbf{X}_2^t \end{bmatrix} \begin{bmatrix} \frac{n}{n_1}\mathbf{1}_1 \\ \frac{n}{n_2}\mathbf{1}_2 \end{bmatrix}$$

$$\longrightarrow \begin{bmatrix} n & (n_1\mathbf{m}_1 + n_2\mathbf{m}_2)^t \\ (n_1\mathbf{m}_1 + n_2\mathbf{m}_2) & \mathbf{S}_w + n_1\mathbf{m}_1\mathbf{m}_1^t + n_2\mathbf{m}_2\mathbf{m}_2^t \end{bmatrix} \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} 0 \\ n(\mathbf{m}_1 - \mathbf{m}_2) \end{bmatrix}$$

sono due equazioni:

prima riga: (**m** = media di tutti i campioni) $\longrightarrow$ $w_0 = -\mathbf{m}^t\mathbf{w}$ $\longrightarrow$ sostituendo nella seconda riga + algebra:

$$\left[\frac{1}{n}\mathbf{S}_w + \frac{n_1 n_2}{n^2}(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^t\right]\mathbf{w} = (\mathbf{m}_1 - \mathbf{m}_2)$$

osservando che $(\mathbf{m}_1\text{-}\mathbf{m}_2)(\mathbf{m}_1\text{-}\mathbf{m}_2)^t\mathbf{w}$ è diretto come $(\mathbf{m}_1\text{-}\mathbf{m}_2)$ $\forall\mathbf{w}$ $\longrightarrow$

$$\frac{n_1 n_2}{n^2}(\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^t\mathbf{w} = (1-\alpha)(\mathbf{m}_1 - \mathbf{m}_2) \longrightarrow \mathbf{w} = \alpha n\frac{(\mathbf{m}_1 - \mathbf{m}_2)}{\mathbf{S}_w}$$

con α opportuno scalare

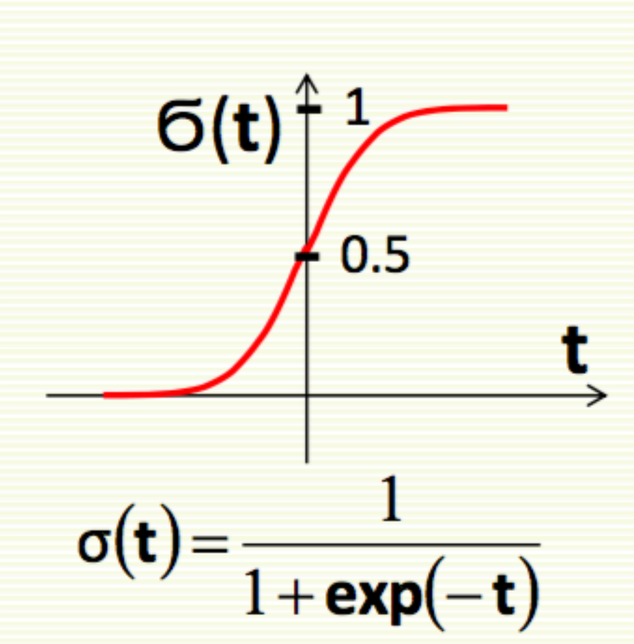
LOGISTIC REGRESSION

- L'algoritmo MSE prova a trovare un vettore di pesi ottimale minimizzano la distanza quadratica tra il target atteso e il vettore di input:

$$L_{MSE} = ||\mathbf{xw} - \mathbf{y}||^2$$

- Il problema principale con questa loss function e che risulta estremamente sensibile agli outliers del training set (eventi rari sulle code delle distribuzioni) e di conseguenza tende rapidamente ad overfittare i dati stessi

- Idea alternativa per evitare questo problema: invece di fare in modo che $\mathbf{w}^t\mathbf{x}$ sia il più vicino possibile a **y**, usiamo una funzione continua $\sigma(\mathbf{w}^t\mathbf{x})$ che sia in grado di "comprimere" la dinamica di $\mathbf{w}^t\mathbf{x}$, e cerchiamo di fare in modo che sua $\sigma(\mathbf{w}^t\mathbf{x})$ il più vicino possibile a **y**
- Una delle funzioni che funzionano meglio con questo approccio è la funzione logistica (o sigmoide):



- funzione convessa che può essere ottimizzata efficientemente con algoritmi di discesa lungo il gradiente
- $\sigma(-t) = 1-\sigma(t)$
- $t = \ln(\sigma/(1-\sigma)$ ← logit function (inversa della sigmoide)

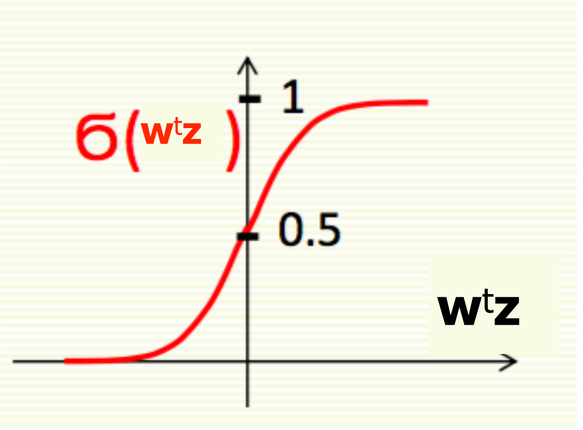
LOGISTIC REGRESSION

- Nella regressione logistica la classificazione è di tipo soft: fornisce una probabilità di appartenenza alla classe e non un indice di classe come nelle classificazioni di tipo hard (ex. preceptron)

- Si parte dall'assunzione di base che:

$$p(\omega_2 \mid \mathbf{x}) = \frac{1}{1 + e^{-(w_0 + \mathbf{w}^t \mathbf{x})}} = \sigma(w_0 + \mathbf{w}^t \mathbf{x}) = \sigma(\mathbf{w}^t \mathbf{z})$$

$$p(\omega_1 \mid \mathbf{x}) = \frac{e^{-(w_0 + \mathbf{w}^t \mathbf{x})}}{1 + e^{-(w_0 + \mathbf{w}^t \mathbf{x})}} = 1 - p(\omega_2 \mid \mathbf{x})$$



- la classificazione tra le due classi usano il likelihood ratio di bayes: assegna x a: $\begin{cases} \omega_1 \text{ se } \log \frac{p(\omega_1 \mid \mathbf{x})}{p(\omega_2 \mid \mathbf{x})} > 1 \\ \omega_2 \text{ se } \log \frac{p(\omega_1 \mid \mathbf{x})}{p(\omega_2 \mid \mathbf{x})} < 1 \end{cases}$

- corrisponde a:

assegna x a: $\begin{cases} \omega_1 \text{ se } w_0 + \mathbf{w}^t \mathbf{x} > 0 \\ \omega_2 \text{ se } w_0 + \mathbf{w}^t \mathbf{x} < 0 \end{cases}$

stessa regola di classificazione trovata nel caso dei discriminanti lineari

LOGISTIC REGRESSION

- NOTA:
  - consideriamo una collezione di sistemi a due stati accoppiati ad un bagno termico (ad esempio una collezione di atomi che possono trovarsi in due stati possibili)
  - sia $y_i = \{0,1\}$ l'indice di stato (una variabile binaria)
  - dalla meccanica statistica elementare sappiamo che se i due stati hanno energia E_0 e E_1 allora la probabilità di trovare il sistema nello stato y_i è:

$$p(y_i = 1) = \frac{e^{-\beta E_0}}{Z} = \frac{e^{-\beta E_0}}{e^{-\beta E_0} + e^{-\beta E_1}} = \frac{1}{1 + e^{-\beta \Delta E}}$$

$$p(y_i = 0) = 1 - p(y_i = 1)$$

- che corrispondono alle espressioni trovate per la regressione logistica se la differenza in energia tra i due stati $\Delta E = w_0 + \mathbf{w}^t\mathbf{x}$ ← relazione stretta tra meccanica statistica e classificazione

LOGISTIC REGRESSION

- dobbiamo ancora definire la funzione di loss: usiamo il principio di massima verosimiglianza
- consideriamo il dataset $T=\{x_i, y_i\}$ con label binarie $y_i \in \{0,1\}$, la likelihood è data da:

$$P(T|\mathbf{w}) = \prod_{i=1}^{n} [\sigma(\mathbf{w}^t \mathbf{z}_i)]^{y_i} [1 - \sigma(\mathbf{w}^t \mathbf{z}_i)]^{1-y_i}$$

- con log-likelihood:

$$l(\mathbf{w}) = \sum_{i=1}^{n} y_i \log \sigma(\mathbf{w}^t \mathbf{z}_i) + (1 - y_i) \log[1 - \sigma(\mathbf{w}^t \mathbf{z}_i)]$$

- considerando come loss function L = -l(w):

$$L(\mathbf{w}) = -\sum_{i=1}^{n} y_i \log \sigma(\mathbf{w}^t \mathbf{z}_i) + (1 - y_i) \log[1 - \sigma(\mathbf{w}^t \mathbf{z}_i)]$$

Cross-Entropia

- per il principio di massima verosimiglianza:

$$\tilde{\mathbf{w}} = \underset{\mathbf{w}}{argmax}\, l(\mathbf{w}) \Rightarrow = \underset{\mathbf{w}}{argmin}\, L(\mathbf{w})$$

LOGISTIC REGRESSION

- calcolando il gradiente e sfruttando la proprietà della funzione logistica:

$$\frac{d\sigma(t)}{dt} = \sigma(t)(1 - \sigma(t))$$

NB: il contributo dell'evento i-esimo è dato dall "errore" ($\sigma(\mathbf{w}^t\mathbf{z}_i)$-$y_i$) tra il valore target e la predizione del modello

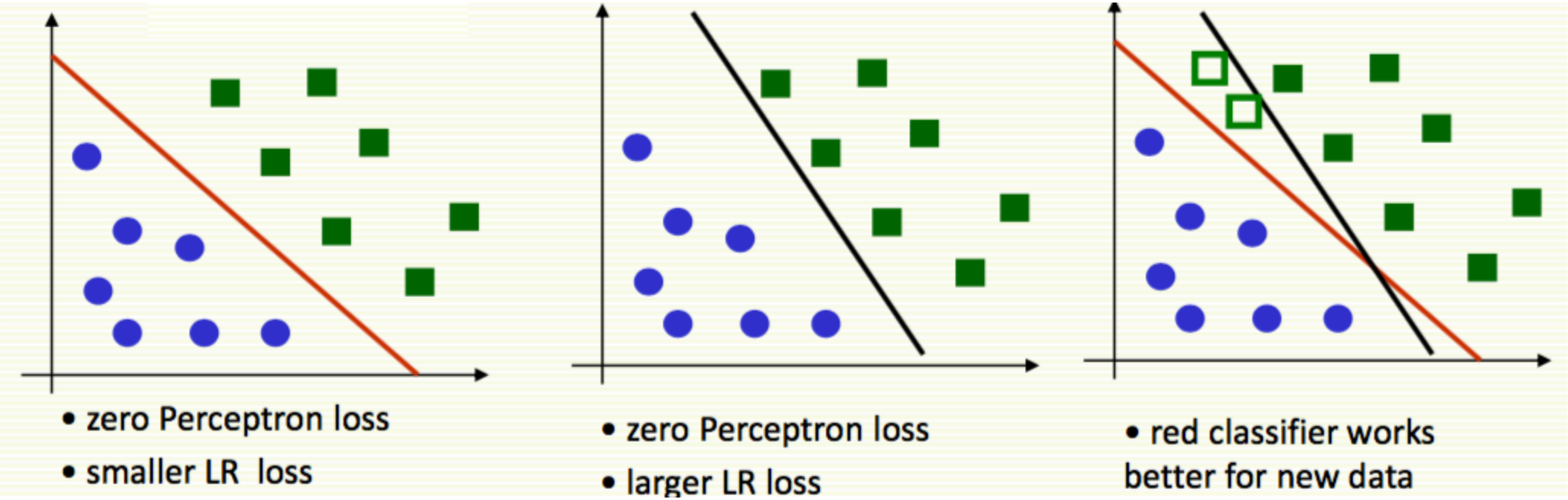
- si ottiene:

$$0 = \nabla L(\mathbf{w}) = -\sum_{i=1}^{n} [\sigma(\mathbf{w}^t\mathbf{z}_i) - y_i]z_i$$

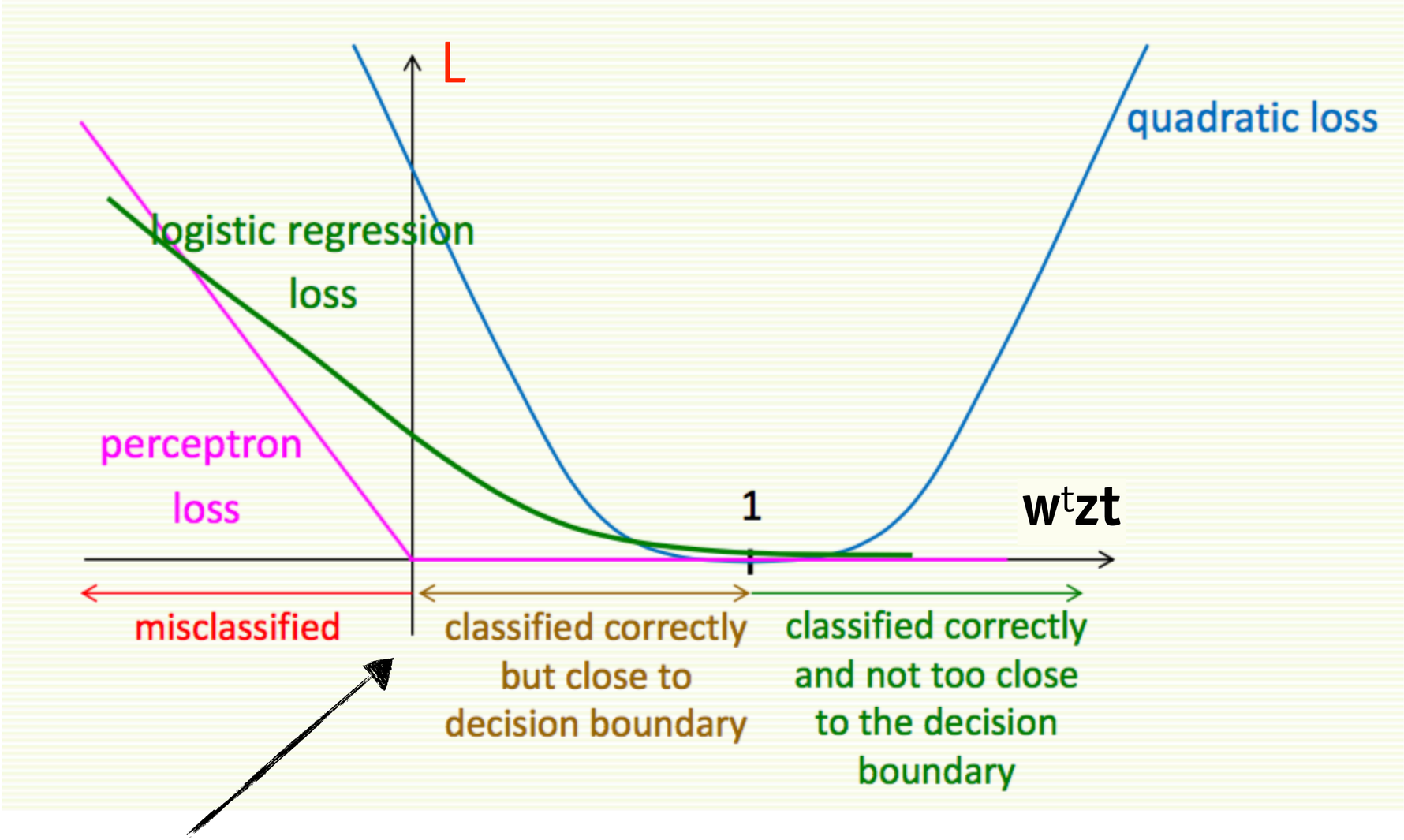
- Una forma semplificata del gradiente che può essere risolto con metodi numerici …

LOGISTIC REGRESSION VS PERCEPTRON

- immaginiamo che uno dei training vector sia classificato correttamente ma si trovi vicino al inerpicano di separazione
- ad esempio: assumiamo $\mathbf{w}^t\mathbf{x} = 0.8$ per tale evento
  - poiché l'eventi è classificato correttamente $L_{perc} = 0$, invece $L_{LR} = -\log(\sigma(0.8)) = 0.37$
  - la regressione logistica incoraggia una scelta per l'iperpiano che sia più lontana dai vettori del training set → i.e. promuove scelte che generalizzano meglio!



CONFRONTO TRA GLI ALGORITMI VISTI



LINEAR CLASSIFICATOR EXAMPLES IN SCIKIT-LEARN

https://www.dropbox.com/s/wwzx8ofy4i2iyue/Perceptron_and_logistic_FDL_classification.ipynb?dl=0

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn import datasets
from sklearn.model_selection import train_test_split


# IRIS dataset
iris = datasets.load_iris()
X = iris.data[:, :2]  # take only the first two features
y = iris.target

X_train, X_test, y_train, y_test = \
        train_test_split(X, y, test_size=.5, random_state=21, shuffle=True)
```

IRIS dataset

```
usePerc = True

if usePerc == True:
    # preceptron
    # stopping criterion. If it is not None, the iterations will stop when (loss > previous_loss - tol)
    logreg = Perceptron(tol=1e-5, random_state=0)
else:
    #logistic regression classifier
    # C: Inverse of regularization strength (smaller values means stronger regularisation)
    # solver: algorithm to use in the optimization problem (sag: stochastic average gradient)
    logreg = LogisticRegression(C=1e5, solver='sag')


# Create an instance of Logistic Regression Classifier and fit the data.
logreg.fit(X_train, y_train)
```

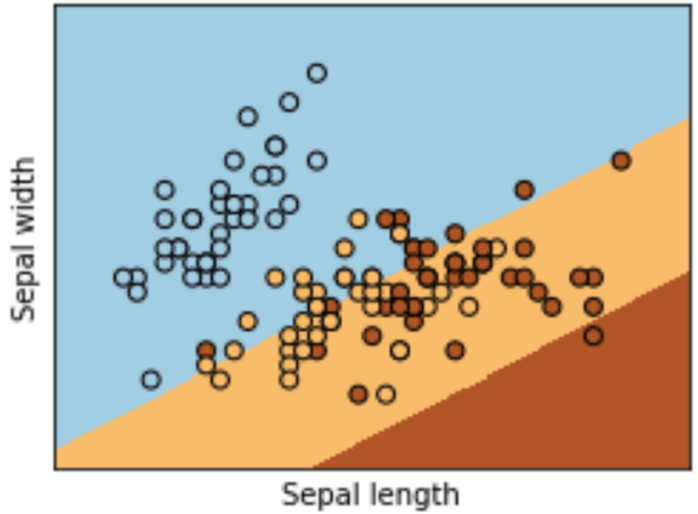
perceptron

logistic

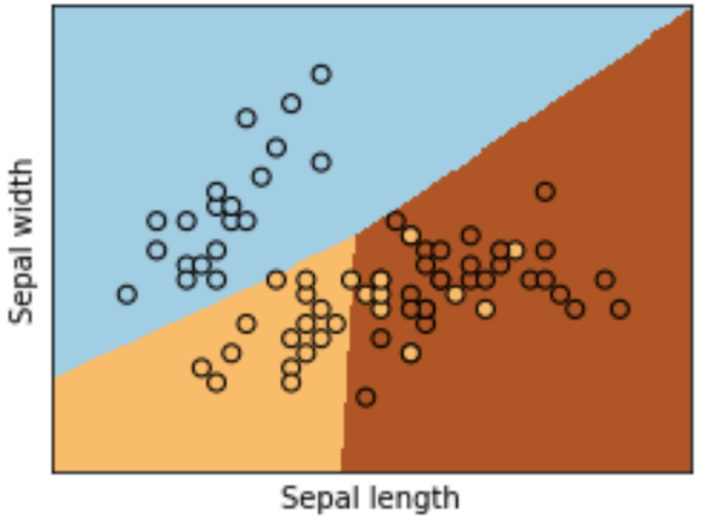
LINEAR CLASSIFICATOR EXAMPLES IN SCIKIT-LEARN

```
# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X_train[:, 0].min() - .5, X_train[:, 0].max() + .5
y_min, y_max = X_train[:, 1].min() - .5, X_train[:, 1].max() + .5
h = .02  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```

perceptron



logistic



SVM

- the methods we have seen so far present all the same issue: they are too flexible (i.e. overtrain easily)

- a way to to fix this issue is to impose external constraints on the choice of weights (parameters) in order to preserve the generalisation capacity of the classifier
  - two mmethods:
    - regularization techniques (we'll discuss it later in the course)
    - Support Vector Machines

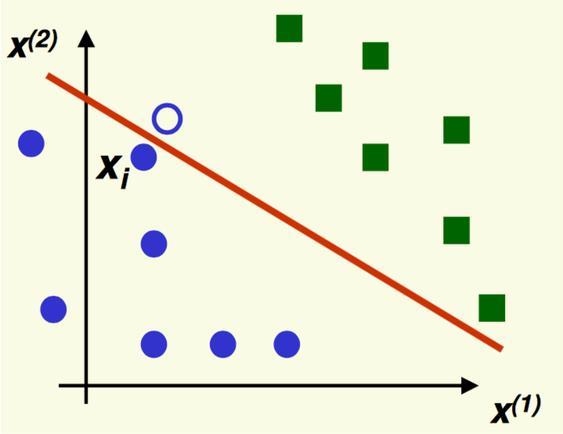
- SVMs were born in the late 70s thanks to Vapnik's studies
  - great theoretical / formal development throughout the 90s
  - elegant theory → among the most important developments of the pattern recognition of the last 20 years (together with the deep learning that has largely supplanted them in the last ~ 5-10 years)
- Pro:
  - excellent generalization properties
  - allows defining linear and non-linear discriminants
- Disadvantage:
  - hard optimisation problem, and slow compared to other classifiers (NN, DNN, BDT)

SVM AND LINEARLY SEPARABLE PROBLEMS

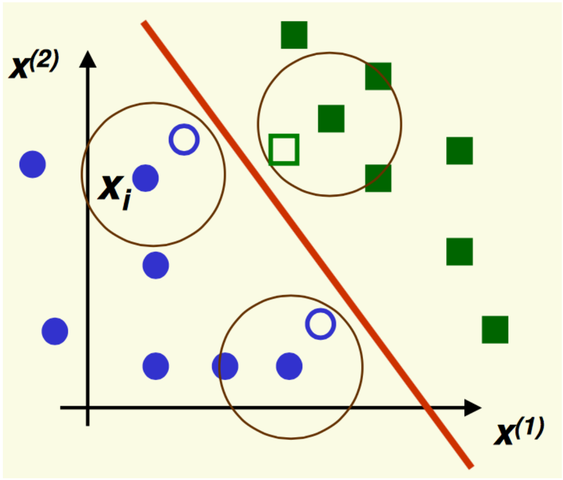
- basic idea: makes better generalisation capacity of a linear discriminant by imposing constraints in the choice of the separation plane

$$g(x) = w^t x + w_0$$

$$g(x) > 0 \Rightarrow x \in \text{class } 1$$
$$g(x) < 0 \Rightarrow x \in \text{class } 2$$



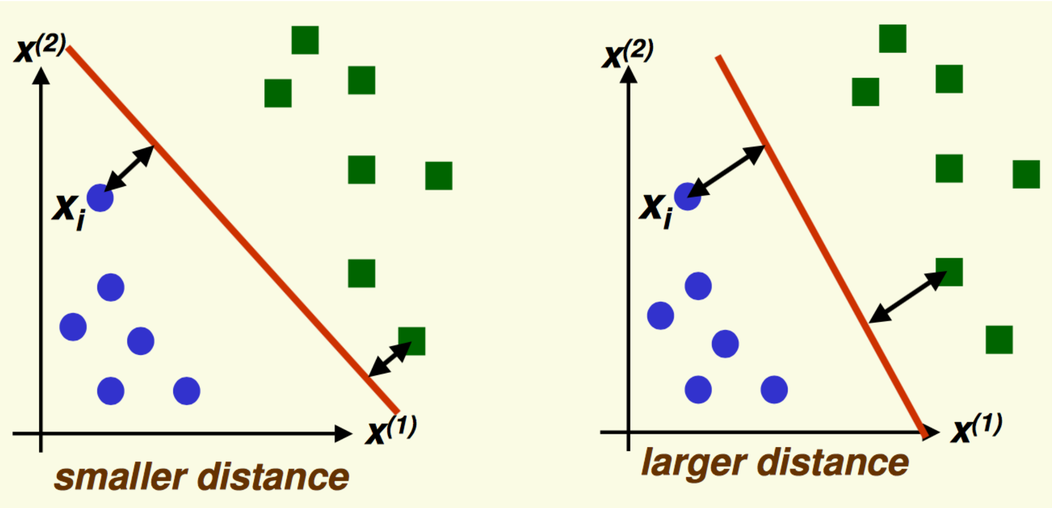
closer is the hyperplane to the training vectors, higher is the probability to have a new vector to fall in the wrong part of the plane due to statistical fluctuation and to be misclassified ⇒ low level of generalisation

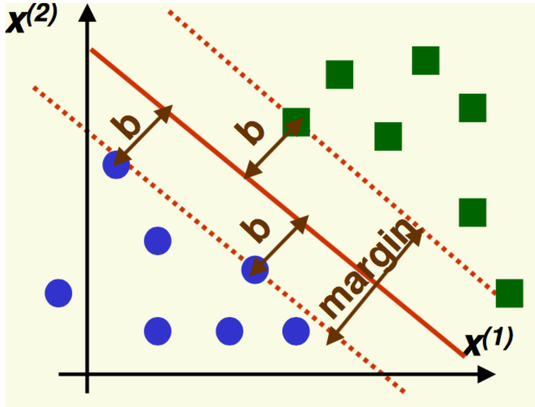


let's then chose the hyperplane as the farthest possible from all the training set vectors: i.e. SVM = linear discriminant with a tolerance gap ⇒ new events close to the events of the training set will be more likely well classified

SVM AND LINEARLY SEPARABLE PROBLEMS

- procedure: find the hyperplane that maximise the distance to the closest events of the train set (called support vectors)
- the optimal hyperplane is the one for which the smallest distance for a vector in one class is equal to the smallest distance in the second one





- in practice SVM maximize the margin defined as 2x(minimal distance from the support vectors)
  - the optimal hyperplane is completely defined by the support vectors
  - to find the plane is a problem of quadratic optimisation

- Can be demonstrated that SVM is the algorithm with the best generalisation properties

NON LINEARLY SEPARABLE CASES

- brilliant trick (kernel trick) to solve non linear problems:

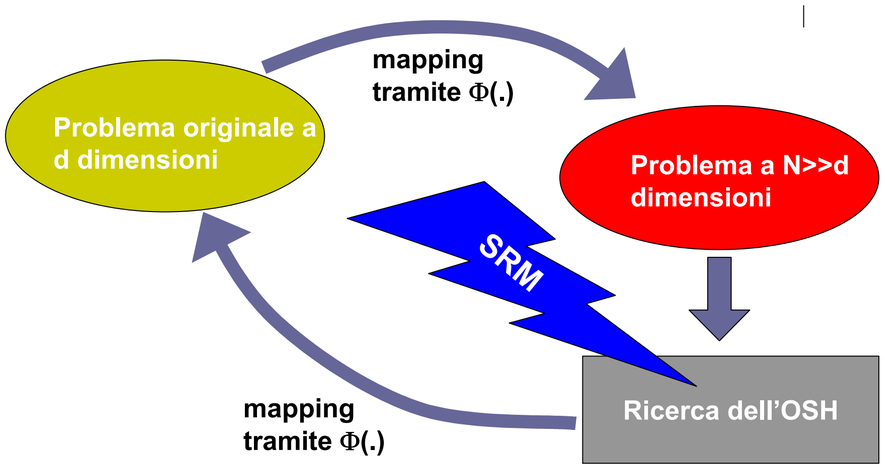
  - is introduced a mapping $\Phi(x)$ into higher-dimensional feature spaces where the events are linearly separable

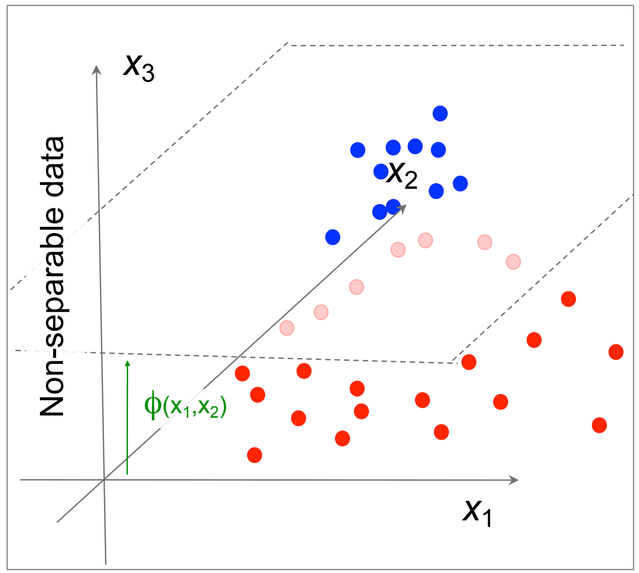
$$\Phi: R^d \rightarrow R^\infty$$

$$g(\mathbf{x}) = \mathbf{w}^t \Phi(\mathbf{x}) + w_0$$

Cover Theorem:
given a set of training data that is not linearly separable, one can with high probability transform it into a training set that is linearly separable by projecting it into a higher-dimensional space via some non-linear transformation

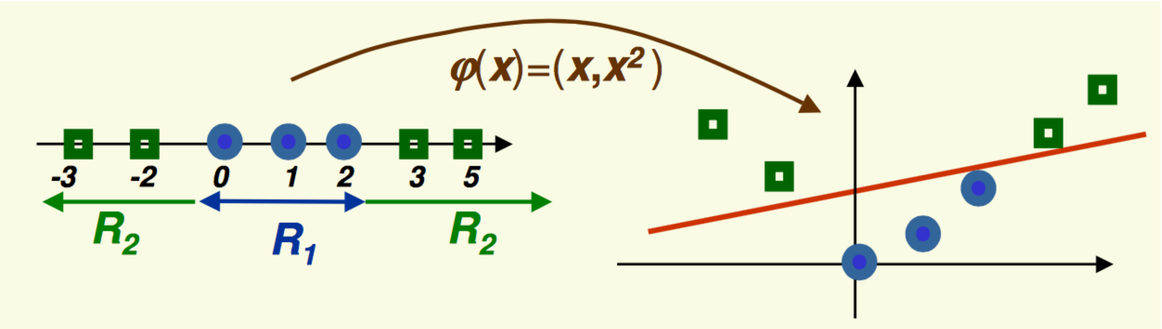
- instead of increase the complexity of the classifier (that remains an hyperplane), we increase the complexity of the feature space





TOY EXAMPLE

- Steps to solve the non-non-linear problem with a linear model:
  - project the data x over a higher dimensional space with a non linear transformation Φ
  - find a linear discriminant function for the transformed data $\Phi(x)$
  - Get back in the original feature space via $g(x) = w^t \Phi(x)+w_0$



1D non linearly separable

mapping: $\Phi(x) = (x,x^2)$

2D linearly separable

- in 2D: the discriminant function is linear:

$$g\left(\begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix}\right) = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} x^{(1)} \\ x^{(2)} \end{bmatrix} + w_0$$

- in 1D: the discriminant function is quadratic:

$$g(x) = w_1 x + w_2 x^2 + w_0$$

SVM: KERNELS TRICK

- It can be demonstrated that the SVM discriminant function is written as:

$$f(x) = \sum_i \alpha_i y_i \Phi(x_i)\Phi(x) + b$$

with α_i coefficients to be determined (iper-parameters of the model)

- with classification rule: ω_1 if f(x)>0, ω_2 otherwise

- For high dimensional data can be very complex to evaluate → curse of dimensionality
- actually what we really need is just the scalar product: $\Phi(x)\cdot\Phi(y)$
- it can be demonstrated under general conditions (Mercer's theorem) that this scalar product can be approximated via simple kernel functions: $K(x,y)= \Phi(x)\cdot\Phi(y)$
- The discriminant function becomes:

$$f(x) = \sum_i \alpha_i y_i K(x_i, x) + b$$

type of SVM	K(x,y)
Polynomial learning machine	$(\mathbf{x}^T\mathbf{y} + 1)^p$
Radial basis function	$\exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}-\mathbf{y}\|^2\right)$
Two-layer perceptron	$\tanh(\beta_o\mathbf{x}^T\mathbf{y} + \beta_1)$

kernel examples

- The use of the kernel functions in practice "hides" the mapping into the N-dimensional space
- Powerful also because allow to use convex optimisation techniques that are guaranteed to converge efficiently

COSTRUZIONE DELL'OSH (OPTIMAL SEPARATING HYPERPLANE)

- per costruire l'iperpiano ottimo bisogna quindi massimizzare M=2/||w|| con i vincoli:
  - $(w \cdot x_i + w_0) \geq 1$ se x_i è un esempio positivo
  - $(w \cdot x_i + w_0) \leq -1$ se x_i è un esempio negativo
- che può essere convertito nel minimizzare la funzione: L(w) = $\frac{1}{2}||\mathbf{w}||^2$
- con vincolo: $(w \cdot x_i + w_0)\, y_i \geq 1 \ \forall i$
- poiché L(w) è una funzione quadratica ⇒ esiste un minimo globale $\min_{w,b} \frac{1}{2}||\mathbf{w}||^2$

- introducendo i moltipicatori di lagrange (α_i, uno per ogni evento) il problema si riscrive come la richiesta di minimizzare il lagrangiano:

$$L_P \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i y_i \left(\mathbf{w} \cdot \mathbf{x}_i + w_0\right) + \sum_{i=1}^{n} \alpha_i, \qquad \alpha_i \geq 0. \ \forall i$$

con vincoli:

- il problema risulta essere un problema convesso di programmazione quadratica, per il quale la soluzione è:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

  - nell'espressione che fornisce w solo alcuni moltiplicatori di lagrange α_i saranno non nulli (quelli che sono vettori di supporto) di conseguenza, solo i corrispondenti punti del training set entreranno come vettori di supporto nella definizione dell'iperpiano

COSTRUZIONE DELL'OSH

- risolto il problema, la funzione discriminante sarà data da:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + \mathbf{b}$$

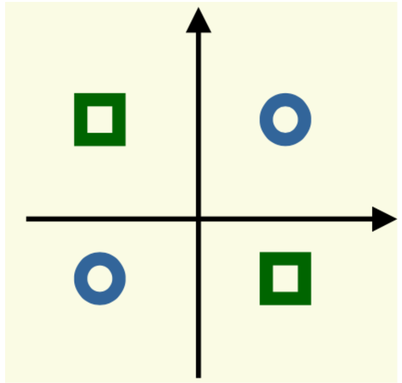
$$\Rightarrow f(\mathbf{x}) = \left(\sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}} \alpha_{\mathbf{i}} \mathbf{y}_{\mathbf{i}} \mathbf{x}_{\mathbf{i}} \right) \cdot \mathbf{x} + \mathbf{b}$$

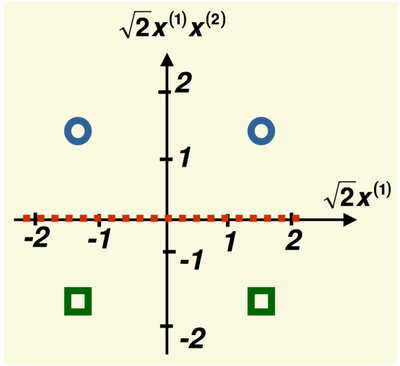
$$\Rightarrow f(\mathbf{x}) = \sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{n}} \alpha_{\mathbf{i}} \mathbf{y}_{\mathbf{i}} (\mathbf{x}_{\mathbf{i}} \cdot \mathbf{x}) + \mathbf{b}$$

SVM XOR

- Exclusive OR: simplest problem not solvable with a linear discriminant

$(x_1=1,x_2=1) \in \omega_1$
$(-1,-1) \in \omega_1$
$(1,-1) \in \omega_2$
$(-1,1) \in \omega_2$





mapping:

$$y = \Phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1 \\ \sqrt{2}x_1x_2 \\ \sqrt{2}x_2 \\ x_2^2 \\ 1 \end{pmatrix}$$

maps x_k from a space with h=2D in a p=6D space

$$y_1 = \Phi(x_1) = \begin{pmatrix} 1 \\ \sqrt{2} \\ \sqrt{2} \\ \sqrt{2} \\ 1 \\ 1 \end{pmatrix} \quad y_3 = \Phi(x_3) = \begin{pmatrix} 1 \\ -\sqrt{2} \\ \sqrt{2} \\ -\sqrt{2} \\ 1 \\ 1 \end{pmatrix}$$

$$y_2 = \Phi(x_2) = \begin{pmatrix} 1 \\ \sqrt{2} \\ -\sqrt{2} \\ -\sqrt{2} \\ 1 \\ 1 \end{pmatrix} \quad y_4 = \Phi(x_4) = \begin{pmatrix} 1 \\ -\sqrt{2} \\ -\sqrt{2} \\ \sqrt{2} \\ 1 \\ 1 \end{pmatrix}$$

in the transformed space the hyperplane is simply:

$g(x_1,x_2) = x_1x_2 = 0$

In the original space the retransformed planes are hyperboles:

$x_1x_2 = \pm 1$

Can be described with a polynomial kernel: $K(x,x_i) = (1+x^tx_i)^2$

$$g(x) = w\varphi(x) = \sum_{i=1}^{6} w_i\varphi_i(x) = -\sqrt{2}\left(\sqrt{2}x^{(1)}x^{(2)}\right) = -2x^{(1)}x^{(2)}$$

SVM: XOR

lagrangiano da minimizzare

$$L_D(\alpha)=\sum_{i=1}^{4}\alpha_i-\frac{1}{2}\sum_{i=1}^{4}\sum_{j=1}^{4}\alpha_i\alpha_j z_i z_j \underbrace{\left(x_i^t x_j+1\right)^2}_{y_i y_j}$$

vincoli

$$0\le\alpha_i \quad \forall i$$

$$\alpha_1+\alpha_2-\alpha_3-\alpha_4=0$$

può essere riscritto come:

$$L_D(\alpha)=\sum_{i=1}^{4}\alpha_i-\frac{1}{2}\alpha^t H\alpha$$

con:

$$\alpha=[\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4]^t$$

$$H=\begin{bmatrix}9 & 1 & -1 & -1\\ 1 & 9 & -1 & -1\\ -1 & -1 & 9 & 1\\ -1 & -1 & 1 & 9\end{bmatrix}$$

deriviamo rispetto a α ed uguagliando a zero:

$$\frac{d}{da}L_D(\alpha)=\begin{bmatrix}1\\1\\1\\1\end{bmatrix}-\begin{bmatrix}9 & 1 & -1 & -1\\ 1 & 9 & -1 & -1\\ -1 & -1 & 9 & 1\\ -1 & -1 & 1 & 9\end{bmatrix}\alpha=0$$

$$\alpha_1=\alpha_2=\alpha_3=\alpha_4=0.25$$

- soddisfa i vincoli
- tutti i campioni sono vettori di supporto

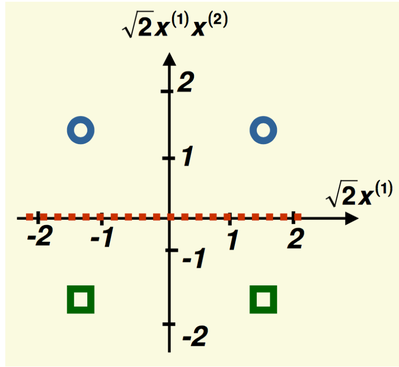
vettore dei pesi:

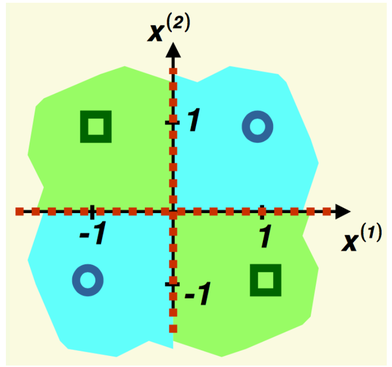
$$w=\sum_{i=1}^{4}\alpha_i z_i\varphi(x_i) \quad y_i: \; 0.25(\varphi(x_1)+\varphi(x_2)-\varphi(x_3)-\varphi(x_4))$$

$$=\left[0 \quad 0 \quad 0 \quad -\sqrt{2} \quad 0 \quad 0\right]$$

funzione discriminante SVM finale:

$$g(x)=w\varphi(x)=\sum_{i=1}^{6}w_i\varphi_i(x)=-\sqrt{2}\left(\sqrt{2}x^{(1)}x^{(2)}\right)=-2x^{(1)}x^{(2)}$$





iperpiano

regione di decisione non lineare

SVM CLASSIFICATION EXAMPLE IN SCIKIT-LEARN

https://www.dropbox.com/s/d4mtt9d9kewyuia/SVM_3class_classification.ipynb?dl=0

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split

# iris dataset (3 iris types with 4 features)
iris = datasets.load_iris()
X = iris.data[:, :2]  # use only the first two features.
y = iris.target

print(X.shape)
print(y.shape)

X_train, X_test, y_train, y_test = \
        train_test_split(X, y, test_size=.2, random_state=21, shuffle=True)

print (X_train.shape)
print (X_test.shape)
```

IRIS dataset

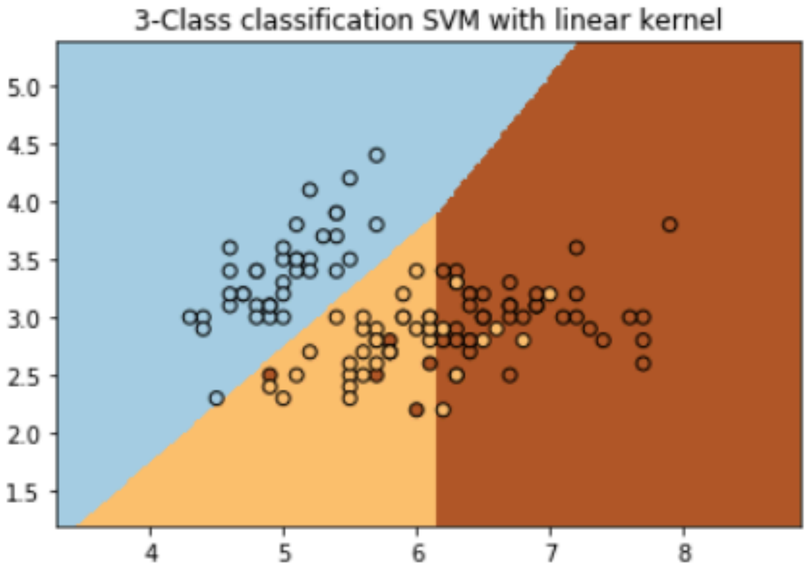
```
(150, 2)
(150,)
```

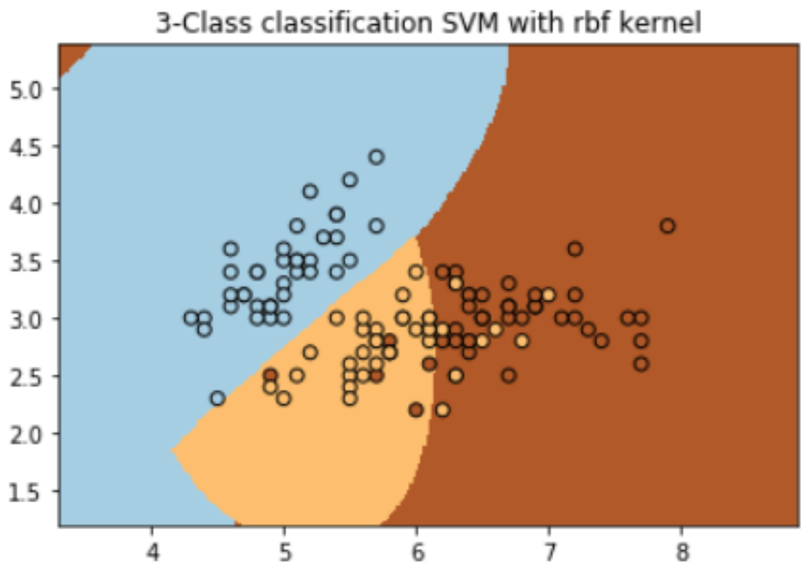
```
(120, 2)
(30, 2)
```

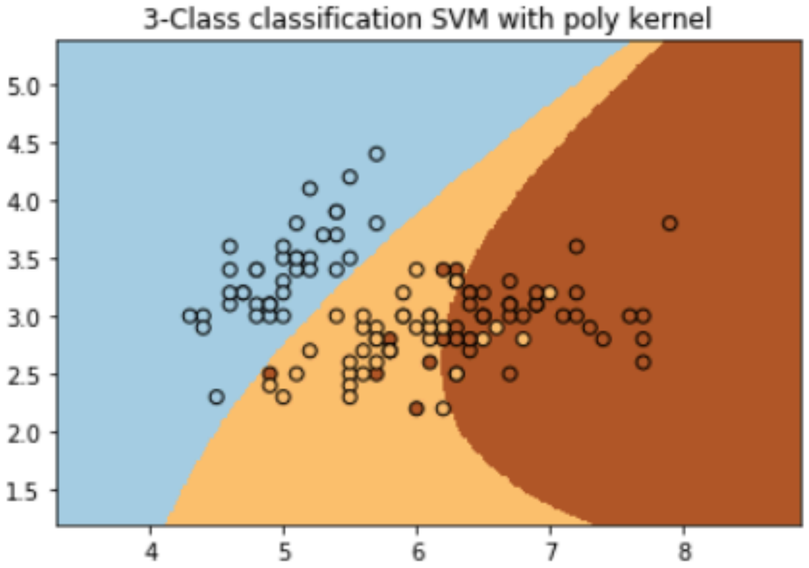
SVM CLASSIFICATION EXAMPLE IN SCIKIT-LEARN

```
#kernel = 'linear'
kernel = 'poly'
#kernel = 'rbf'
h = .02  # step size in the mesh

for kernel in ('linear', 'poly', 'rbf'):
    # create an instance of SVM and fit out data.
    clf = svm.SVC(kernel=kernel)
    clf.fit(X_train, y_train)

    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
    y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

    # Plot also the training points
    plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=plt.cm.Paired, edgecolors='k')
    plt.title('3-Class classification SVM with %s kernel' % kernel)
    plt.axis('tight')
    plt.show()
```

SVM CLASSIFICATION EXAMPLE IN SCIKIT-LEARN







type of SVM	K(x,y)
Polynomial learning machine	$(\mathbf{x}^T\mathbf{y} + 1)^p$
Radial basis function	$\exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x}-\mathbf{y}\|^2\right)$

performances on test sample

```
prediction = clf.predict(X_test)
print(prediction)
print(y_test)
print(abs(prediction-y_test))
print('accuracy: ', clf.score(X_test, y_test))
```

```
[1 0 0 0 1 2 0 1 0 0 1 1 2 2 0 2 1 1 0 1 2 1 1 2 0 1 0 0 2 2]
[1 0 0 0 1 1 0 2 0 0 1 1 2 2 0 1 2 1 0 2 2 1 1 1 0 1 0 0 1 2]
[0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0]
accuracy:  0.7666666666666667
```