**EBook Gratuito**

# APPRENDIMENTO machine-learning

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#machine-learning

# Sommario

# Di

# Capitolo 1: Iniziare con l'apprendimento automatico

## Osservazioni

Machine Learning è la scienza (e l'arte) della programmazione dei computer in modo che possano apprendere dai dati.

Una definizione più formale:

È il campo di studio che dà ai computer la possibilità di apprendere senza essere programmato esplicitamente. Arthur Samuel, 1959

Una definizione più orientata all'ingegneria:

Si dice che un programma per computer apprende dall'esperienza E in relazione a qualche compito T e qualche misura di prestazione P, se la sua prestazione su T, misurata da P, migliora con l'esperienza E. Tom Mitchell, 1997

Fonte: "Hands-On Machine Learning con Scikit-Learn e TensorFlow di Aurélien Géron (O'Reilly). Copyright 2017 Aurélien Géron, 978-1-491-96229-9. "

L'apprendimento automatico (ML) è un campo dell'informatica che è nato dalla ricerca nell'intelligenza artificiale. La forza dell'apprendimento automatico rispetto ad altre forme di analisi è nella sua capacità di scoprire intuizioni nascoste e prevedere i risultati di input futuri non visibili (generalizzazione). A differenza degli algoritmi iterativi in cui le operazioni sono esplicitamente dichiarate, gli algoritmi di machine learning prendono in prestito concetti dalla teoria della probabilità per selezionare, valutare e migliorare i modelli statistici.

## Examples

### Installazione o installazione usando Python

#### 1) Scikit impara

scikit-learn è un modulo Python per l'apprendimento automatico basato su SciPy e distribuito sotto la licenza BSD di 3-Clausole. È dotato di vari algoritmi di classificazione, regressione e clustering che includono macchine vettoriali di supporto, foreste casuali, boosting gradiente, k-means e DBSCAN ed è progettato per interagire con le librerie numeriche e scientifiche Python NumPy e SciPy.

L'attuale versione stabile di scikit-learn richiede :

- Python (> = 2.6 o> = 3.3),
- NumPy (> = 1.6.1),

- SciPy (> = 0,9).

Per la maggior parte di installazione `pip` gestore di pacchetti python può installare python e tutte le sue dipendenze:

```
pip install scikit-learn
```

Tuttavia, per i sistemi Linux si consiglia di utilizzare il gestore pacchetti `conda` per evitare possibili processi di compilazione

```
conda install scikit-learn
```

Per verificare di avere `scikit-learn` , esegui in shell:

```
python -c 'import sklearn; print(sklearn.__version__)'
```

**Installazione di Windows e Mac OSX:**

Canopy e Anaconda hanno entrambi una versione recente di *scikit-learn* , oltre a un ampio set di librerie scientifiche Python per Windows, Mac OSX (anche per Linux).

Repository ufficiale del codice sorgente: https://github.com/scikit-learn/scikit-learn

---

**2) Piattaforma Numenta per l'Intelligent Computing**

La piattaforma Numenta per l'Intelligent Computing (NuPIC) è una piattaforma di machine intelligence che implementa gli algoritmi di apprendimento HTM. L'HTM è una teoria computazionale dettagliata della neocorteccia. Al centro dell'HTM ci sono algoritmi di apprendimento continuo basati sul tempo che memorizzano e richiamano modelli spaziali e temporali. NuPIC è adatto a una varietà di problemi, in particolare il rilevamento di anomalie e la previsione di sorgenti di dati in streaming.

I binari NuPIC sono disponibili per:

Linux x86 a 64 bit
OS X 10.9
OS X 10.10
Windows 64 bit

Le seguenti dipendenze sono necessarie per installare NuPIC su tutti i sistemi operativi.

- Python 2.7
- pip> = 8.1.2
- setuptools> = 25.2.0
- ruota> = 0.29.0
- numpy
- Compilatore C ++ 11 come gcc (4.8+) o clang

Requisiti aggiuntivi per OS X:

- Xcode strumenti da riga di comando

Esegui quanto segue per installare NuPIC:

```
pip install nupic
```

Repository ufficiale del codice sorgente: https://github.com/numenta/nupic

---

**3) nilearn**

Nilearn è un modulo Python per l'apprendimento statistico veloce e facile sui dati NeuroImaging. Sfrutta la toolbox Python per gli scikit-learn per le statistiche multivariate con applicazioni come la modellazione predittiva, la classificazione, la decodifica o l'analisi della connettività.

Le dipendenze richieste per utilizzare il software sono:

- Python> = 2.6,
- setuptools
- Numpy> = 1.6.1
- SciPy> = 0,9
- Scikit-learn> = 0,14,1
- Nibabel> = 1.1.0

Se si utilizzano funzionalità di stampa nilearn o si eseguono gli esempi, è richiesto matplotlib> = 1.1.1.

Se vuoi eseguire i test, hai bisogno di naso> = 1.2.1 e copertura> = 3.6.

Prima assicurati di aver installato tutte le dipendenze elencate sopra. Quindi puoi installare nilearn eseguendo il seguente comando in un prompt dei comandi:

```
pip install -U --user nilearn
```

Repository ufficiale del codice sorgente: https://github.com/nilearn/nilearn/

**4) Utilizzo di Anaconda**

Molte librerie scientifiche Python sono prontamente disponibili in Anaconda. Puoi ottenere i file di installazione da qui . Da una parte, usando Anaconda, non devi installare e configurare molti pacchetti, è concesso in licenza BSD e ha un processo di installazione banale, disponibile per Python 3 e Python 2, mentre, d'altra parte, ti offre meno flessibilità. Ad esempio, alcuni pacchetti python per deep learning all'avanguardia potrebbero utilizzare una versione diversa di numpy e quindi Anaconda. Tuttavia, questo svantaggio può essere risolto usando un'altra installazione python separatamente (in Linux e MAC come quella predefinita, ad esempio).

L'installazione di Anaconda richiede di selezionare la posizione di installazione e richiede anche

l'opzione di aggiunta PATH. Se si aggiunge Anaconda al PATH, si prevede che il sistema operativo troverà Anaconda Python come predefinito. Pertanto, le modifiche e le installazioni future saranno disponibili solo per questa versione di Python.

Per chiarire, dopo l'installazione di Anaconda e aggiungilo a PATH, usando Ubuntu 14.04 tramite terminale se si digita

```
python
```

```
Python 2.7.12 |Anaconda 4.2.0 (64-bit)| (default, Jul  2 2016, 17:42:40)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
```

Voila, Anaconda Python è il tuo Python predefinito, puoi iniziare a divertirti usando molte librerie subito. Comunque, se vuoi usare il tuo vecchio Python

```
/usr/bin/python
```

```
Python 2.7.6 (default, Oct 26 2016, 20:30:19)
[GCC 4.8.4] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

In breve, Anaconda è uno dei modi più veloci per avviare l'apprendimento automatico e l'analisi dei dati con Python.

## Installazione o installazione usando il linguaggio R

I pacchetti sono raccolte di funzioni R, dati e codice compilato in un formato ben definito. Repository pubblici (e privati) vengono utilizzati per ospitare raccolte di pacchetti R. La più grande collezione di pacchetti R è disponibile da CRAN. Alcuni dei pacchetti R machine learning più popolari sono i seguenti, tra gli altri:

**1) rpart**

Descrizione: partizionamento ricorsivo per alberi di classificazione, regressione e sopravvivenza. Un'implementazione della maggior parte delle funzionalità del libro del 1984 di Breiman, Friedman, Olshen e Stone.

Può essere installato da CRAN utilizzando il seguente codice:

```
install.packages("rpart")
```

Carica il pacchetto:

```
library(rpart)
```

Fonte ufficiale: https://cran.r-project.org/web/packages/rpart/index.html

---

**2) e1071**

Descrizione: funzioni per analisi di classi latenti, trasformazioni di Fourier a breve tempo, clustering fuzzy, macchine di supporto vettoriale, calcolo del percorso più breve, clustering insacchettato, classificatore di Bayes ingenuo ecc.

Installazione da CRAN:

```
install.packages("e1071")
```

Caricamento del pacchetto:

```
library(e1071)
```

Fonte ufficiale: https://cran.r-project.org/web/packages/e1071/index.html

---

**3) randomForest**

Descrizione: Classificazione e regressione basate su una foresta di alberi che utilizza input casuali.

Installazione da CRAN:

```
install.packages("randomForest")
```

Caricamento del pacchetto:

```
library(randomForest)
```

Fonte ufficiale: https://cran.r-project.org/web/packages/randomForest/index.html

---

**4) segno di omissione**

Descrizione: funzioni varie per l'addestramento e la tracciatura di modelli di classificazione e regressione.

Installazione da CRAN:

```
install.packages("caret")
```

Caricamento del pacchetto:

```
library(caret)
```

Fonte ufficiale: https://cran.r-project.org/web/packages/caret/index.html

Leggi Iniziare con l'apprendimento automatico online: https://riptutorial.com/it/machine-learning/topic/1151/iniziare-con-l-apprendimento-automatico

# Capitolo 2: Apprendimento approfondito

## introduzione

L'Apprendimento profondo è un sottosettore dell'apprendimento automatico in cui le reti neurali artificiali multistrato sono utilizzate a scopo di apprendimento. L'Apprendimento profondo ha trovato molte grandi implementazioni, ad es. Riconoscimento vocale, Sottotitoli su Youtube, Raccomandazioni Amazon e così via. Per ulteriori informazioni è disponibile un argomento dedicato all'apprendimento approfondito .

## Examples

### Breve riassunto dell'apprendimento profondo

Per addestrare una rete neurale, in primo luogo dobbiamo progettare un'idea valida ed efficiente. Esistono tre tipi di attività di apprendimento.

- Apprendimento supervisionato
- Insegnamento rafforzativo
- Apprendimento senza supervisione

In questo tempo presente, l'apprendimento senza supervisione è molto popolare. L'apprendimento non supervisionato è un compito di apprendimento profondo che deduce una funzione per descrivere la struttura nascosta da dati "senza etichetta" (una classificazione o una categorizzazione non è inclusa nelle osservazioni).

Poiché gli esempi forniti allo studente non sono etichettati, non vi è alcuna valutazione dell'accuratezza della struttura prodotta dal relativo algoritmo, che è un modo per distinguere l'apprendimento non supervisionato dall'apprendimento supervisionato e dall'apprendimento rinforzato.

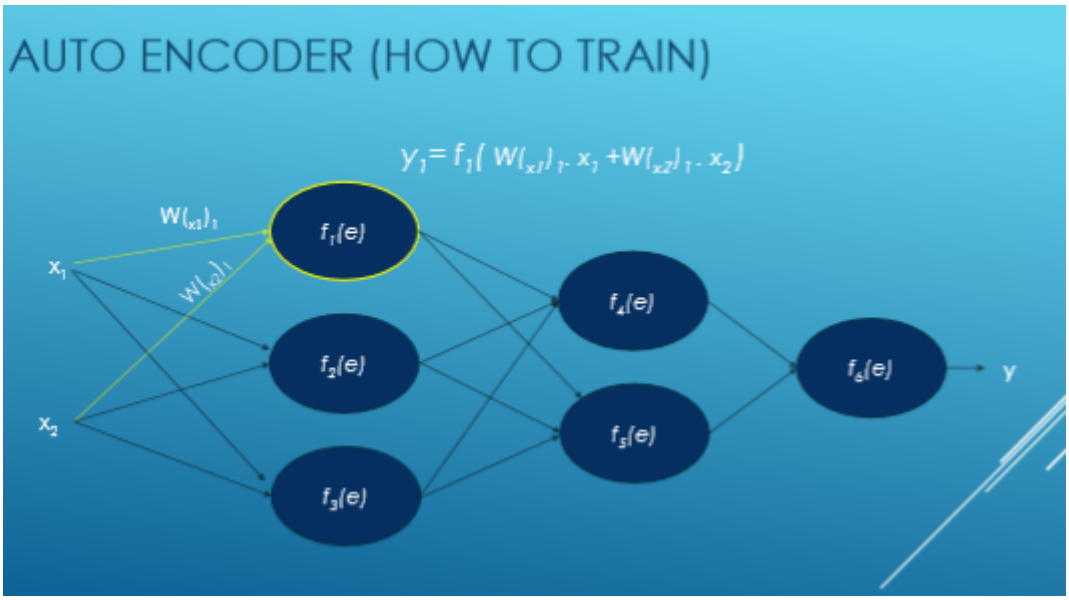
Esistono tre tipi di apprendimento senza supervisione.

- Macchine Boltzmann ristrette
- Modello di codifica sparsi
- Autoencoder Descriverò in dettaglio l'autoencoder.

Lo scopo di un autoencoder è di apprendere una rappresentazione (codifica) per un insieme di dati, tipicamente ai fini della riduzione della dimensionalità.

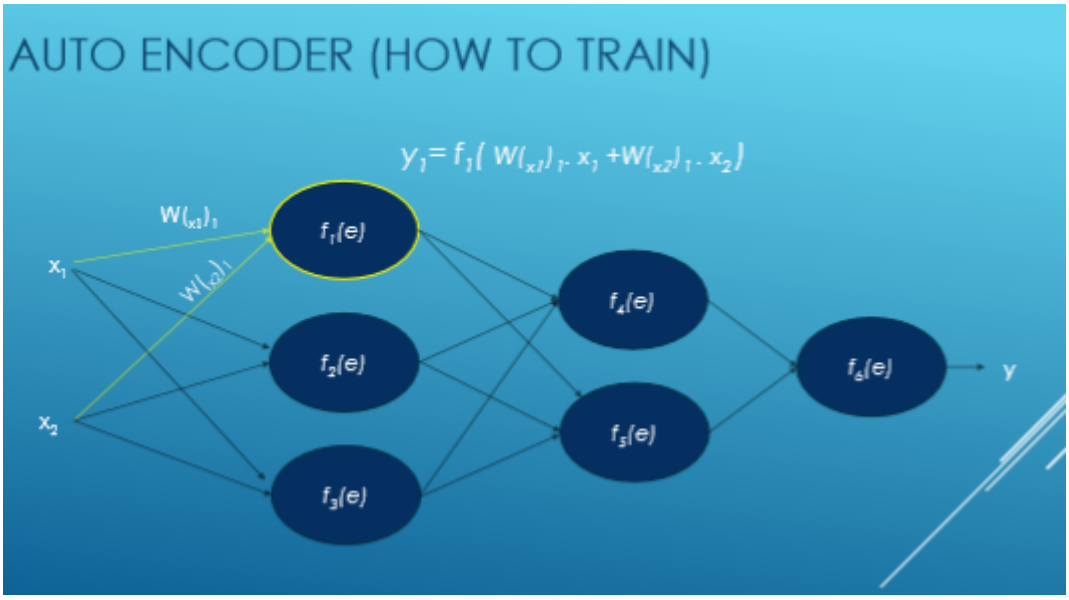
La forma più semplice di un autoencoder è un feedforward, con un livello di input, uno strato di output e uno o più livelli nascosti che li connettono. Ma con lo strato di output che ha lo stesso numero di nodi del livello di input e con lo scopo di ricostruire i propri input ed è per questo che si chiama apprendimento non supervisionato.

Ora proverò a dare un esempio di formazione della rete neurale.

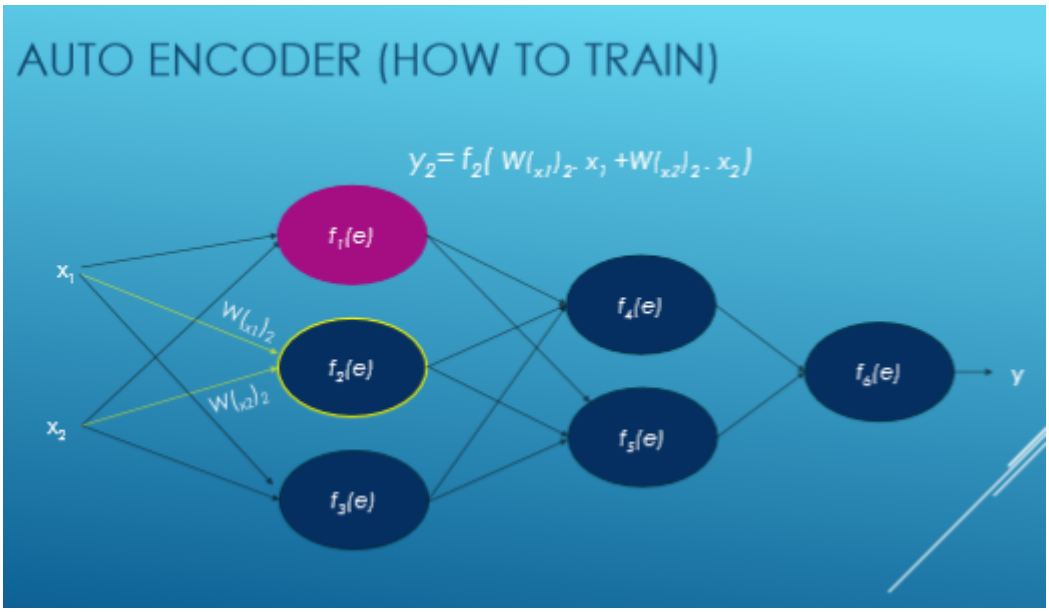


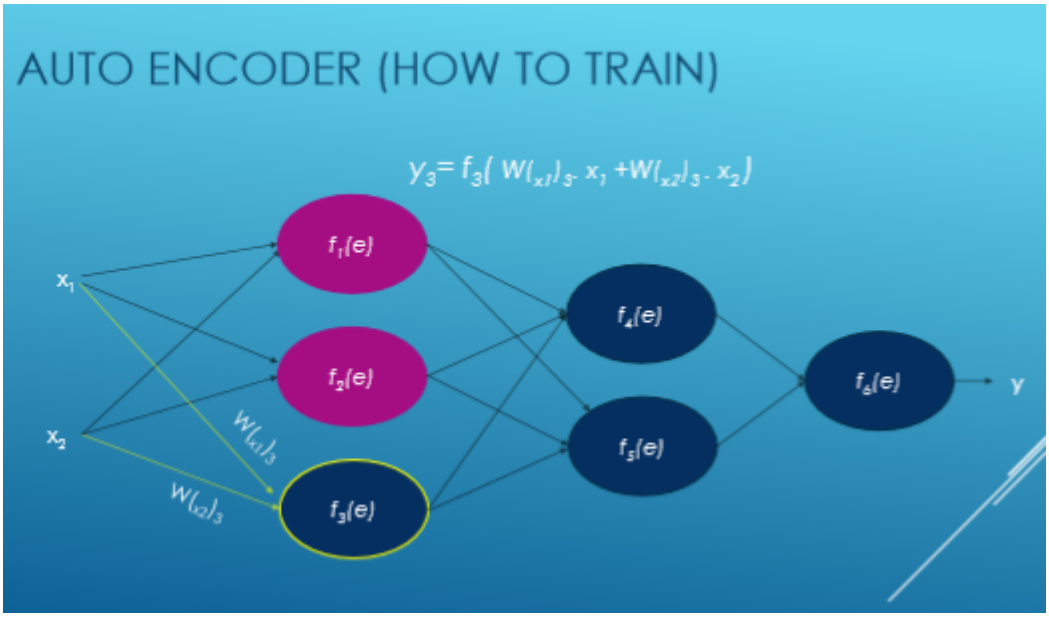

Qui viene inserito Xi, W è il peso, f (e) è la funzione di attivazione e y viene emesso.

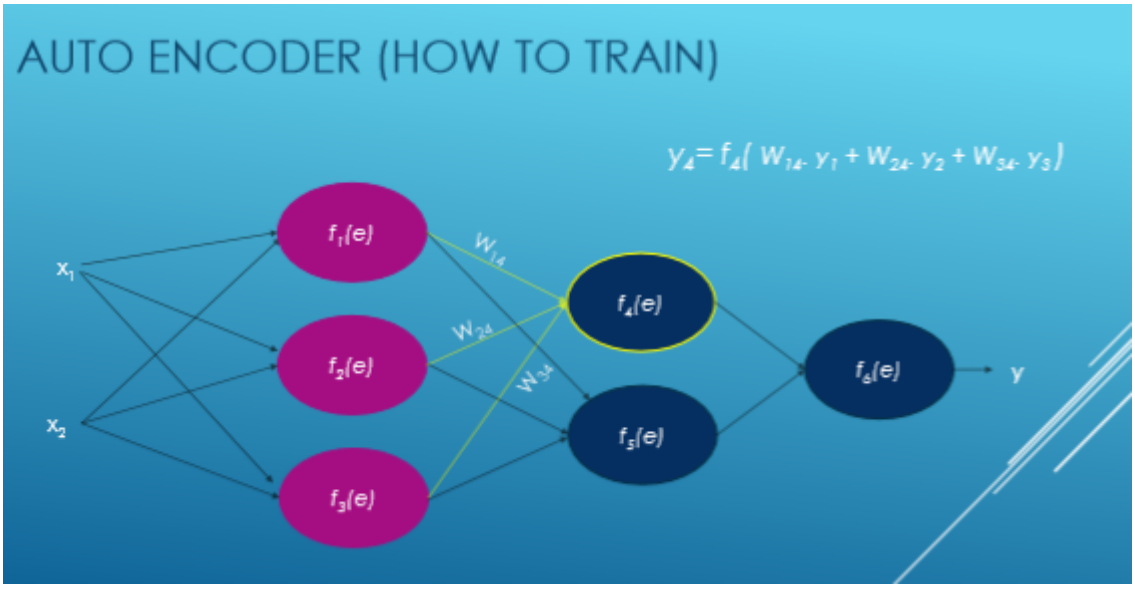
Ora vediamo il flusso passo passo della rete neurale di allenamento basata sul autoencoder.

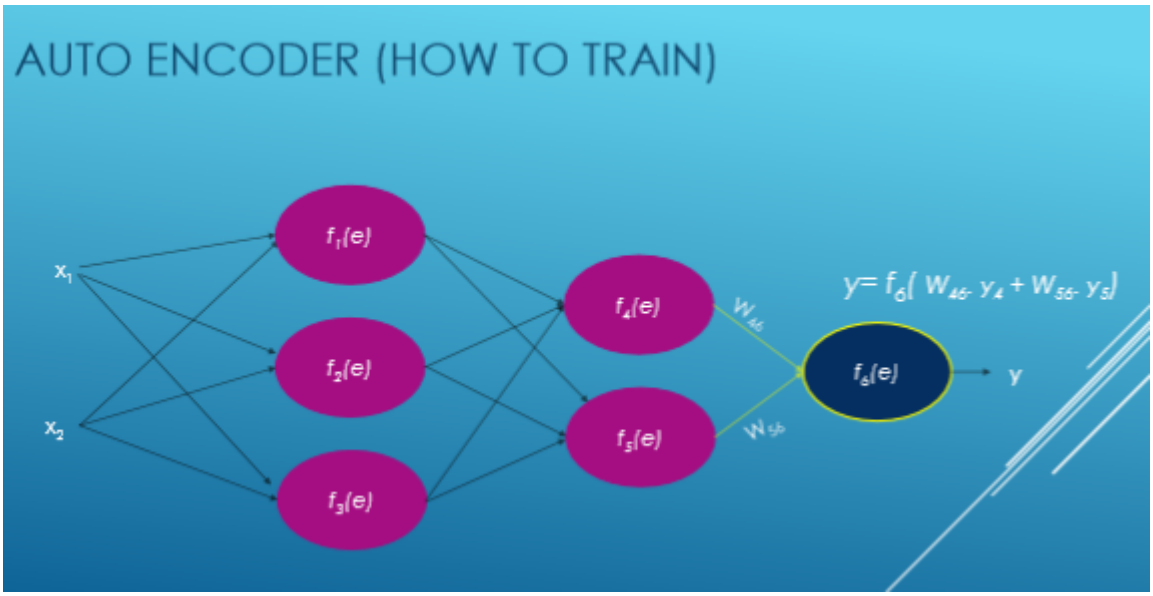



Calcoliamo il valore di ogni funzione di attivazione con questa equazione: y = WiXi. Prima di tutto, selezioniamo i numeri in modo casuale per i pesi e poi proviamo a regolare i pesi.

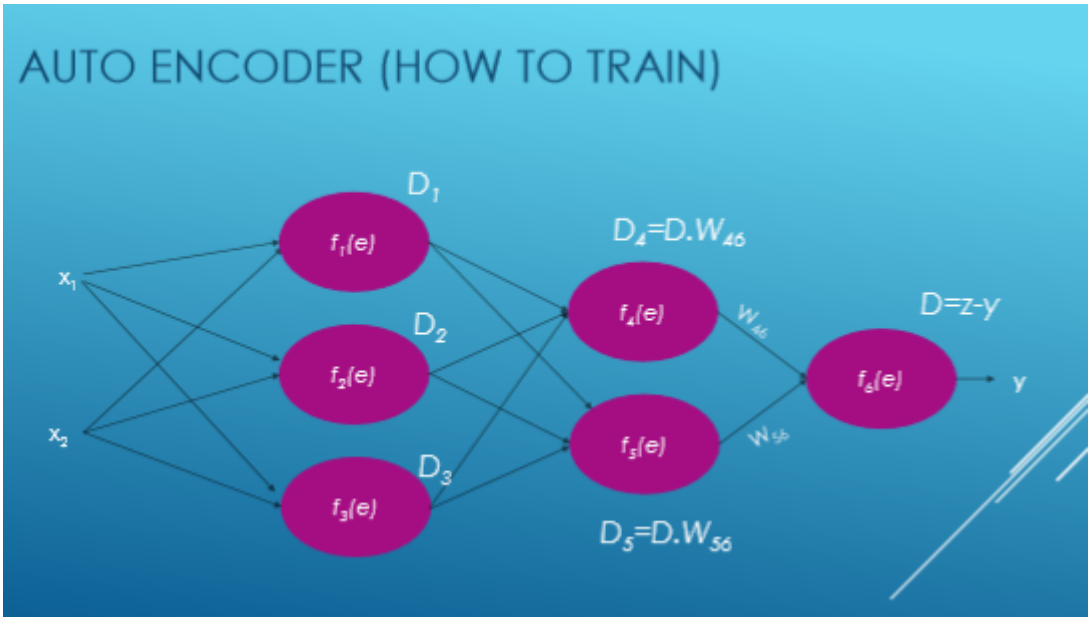

AUTO ENCODER (HOW TO TRAIN)
$y_2 = f_2( W_{(x1)2} \cdot x_1 + W_{(x2)2} \cdot x_2 )$
$x_1$
$x_2$
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
y
$W_{(x1)2}$
$W_{(x2)2}$


AUTO ENCODER (HOW TO TRAIN)
$y_3 = f_3( W_{(x1)3} \cdot x_1 + W_{(x2)3} \cdot x_2 )$
$x_1$
$x_2$
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
y
$W_{(x1)3}$
$W_{(x2)3}$

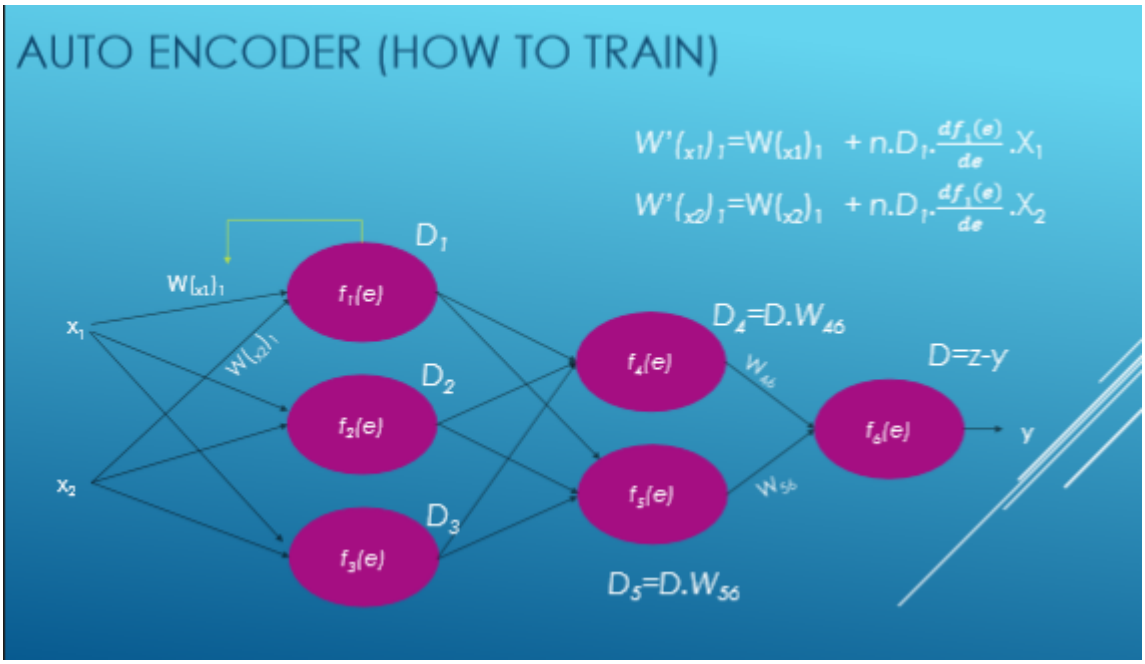

AUTO ENCODER (HOW TO TRAIN)
$y_4 = f_4( W_{14} \cdot y_1 + W_{24} \cdot y_2 + W_{34} \cdot y_3 )$
$x_1$
$x_2$
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
y
$W_{14}$
$W_{24}$
$W_{34}$




Ora calcoliamo la deviazione dall'output desiderato, ovvero y = zy e calcoliamo le deviazioni di ciascuna funzione di attivazione.




Quindi aggiustiamo il nostro nuovo peso di ogni connessione.




Leggi Apprendimento approfondito online: https://riptutorial.com/it/machine-learning/topic/7442/apprendimento-approfondito

# Capitolo 3: Apprendimento automatico e classificazione

## Examples

### Cos'è l'apprendimento automatico?

Sono disponibili due definizioni di Machine Learning. *Arthur Samuel l'ha* descritto come:

> il campo di studio che dà ai computer la possibilità di apprendere senza essere programmato esplicitamente.

Questa è una definizione più vecchia e informale.

*Tom Mitchell* fornisce una definizione più moderna:

> Si dice che un programma per computer apprende dall'esperienza E in relazione ad alcune classi di compiti T e alla misura della prestazione P, se le sue prestazioni nei compiti in T, misurate da P, migliorano con l'esperienza E.

Esempio: giocare a dama.

E = l'esperienza di giocare a molti giochi di dama

T = il compito di giocare a dama.

P = la probabilità che il programma vincerà la prossima partita.

In generale, qualsiasi problema di apprendimento automatico può essere assegnato a una delle due ampie classificazioni:

1. Apprendimento supervisionato
2. Apprendimento senza supervisione.

### Cos'è l'apprendimento supervisionato?

L'apprendimento supervisionato è un tipo di algoritmo di apprendimento automatico che utilizza un set di dati noto (chiamato set di dati di addestramento) per effettuare previsioni.

Categoria di apprendimento supervisionato:

1. **Regressione:** in un problema di regressione, stiamo cercando di prevedere i risultati all'interno di un output continuo, il che significa che stiamo cercando di mappare le variabili di input su una funzione continua.
2. **Classificazione:** in un problema di classificazione, stiamo invece cercando di prevedere i risultati in un output discreto. In altre parole, stiamo cercando di mappare le variabili di input in categorie discrete.

**Esempio 1:**

Dati i dati sulla dimensione delle case sul mercato immobiliare, prova a prevedere il loro prezzo. Il prezzo in funzione delle dimensioni è un risultato continuo, quindi questo è un problema di regressione.

**Esempio 2:**

(a) *Regressione* - Per i valori di risposta continua. Ad esempio, data l'immagine di una persona, dobbiamo prevedere la loro età sulla base dell'immagine data

(b) *Classificazione* - per valori di risposta categoriali, in cui i dati possono essere separati in "classi" specifiche. Ad esempio, dato un paziente con un tumore, dobbiamo prevedere se il tumore è maligno o benigno.

## Cos'è l'apprendimento senza supervisione?

L'apprendimento senza supervisione ci permette di affrontare i problemi con poca o nessuna idea di come dovrebbero essere i nostri risultati. Possiamo ricavare la struttura da dati in cui non conosciamo necessariamente l'effetto delle variabili.

**Esempio:**

*Clustering:* viene utilizzato per l'analisi dei dati esplorativi per trovare schemi nascosti o raggruppare i dati. Prendi una raccolta di 1.000.000 di geni diversi e trova un modo per raggruppare automaticamente questi geni in gruppi che sono in qualche modo simili o correlati da diverse variabili, come durata della vita, posizione, ruoli e così via.

Leggi Apprendimento automatico e classificazione online: https://riptutorial.com/it/machine-learning/topic/9986/apprendimento-automatico-e-classificazione

# Capitolo 4: Apprendimento automatico tramite Java

## Examples

### lista degli strumenti

```
Cortical.io - Retina: an API performing complex NLP operations (disambiguation,
classification, streaming text filtering, etc...) as quickly and intuitively as the brain.
CoreNLP - Stanford CoreNLP provides a set of natural language analysis tools which can take
raw English language text input and give the base forms of words
Stanford Parser - A natural language parser is a program that works out the grammatical
structure of sentences
Stanford POS Tagger - A Part-Of-Speech Tagger (POS Tagger
Stanford Name Entity Recognizer - Stanford NER is a Java implementation of a Named Entity
Recognizer.
Stanford Word Segmenter - Tokenization of raw text is a standard pre-processing step for many
NLP tasks.
Tregex, Tsurgeon and Semgrex - Tregex is a utility for matching patterns in trees, based on
tree relationships and regular expression matches on nodes (the name is short for "tree
regular expressions").
Stanford Phrasal: A Phrase-Based Translation System
Stanford English Tokenizer - Stanford Phrasal is a state-of-the-art statistical phrase-based
machine translation system, written in Java.
Stanford Tokens Regex - A tokenizer divides text into a sequence of tokens, which roughly
correspond to "words"
Stanford Temporal Tagger - SUTime is a library for recognizing and normalizing time
expressions.
Stanford SPIED - Learning entities from unlabeled text starting with seed sets using patterns
in an iterative fashion
Stanford Topic Modeling Toolbox - Topic modeling tools to social scientists and others who
wish to perform analysis on datasets
Twitter Text Java - A Java implementation of Twitter's text processing library
MALLET - A Java-based package for statistical natural language processing, document
classification, clustering, topic modeling, information extraction, and other machine learning
applications to text.
OpenNLP - a machine learning based toolkit for the processing of natural language text.
LingPipe - A tool kit for processing text using computational linguistics.
ClearTK - ClearTK provides a framework for developing statistical natural language processing
(NLP) components in Java and is built on top of Apache UIMA.
Apache cTAKES - Apache clinical Text Analysis and Knowledge Extraction System (cTAKES) is an
open-source natural language processing system for information extraction from electronic
medical record clinical free-text.
ClearNLP - The ClearNLP project provides software and resources for natural language
processing. The project started at the Center for Computational Language and EducAtion
Research, and is currently developed by the Center for Language and Information Research at
Emory University. This project is under the Apache 2 license.
CogcompNLP - This project collects a number of core libraries for Natural Language Processing
(NLP) developed in the University of Illinois' Cognitive Computation Group, for example
illinois-core-utilities which provides a set of NLP-friendly data structures and a number of
NLP-related utilities that support writing NLP applications, running experiments, etc,
illinois-edison a library for feature extraction from illinois-core-utilities data structures
and many other packages.
```

## Machine Learning di uso generale

```
aerosolve - A machine learning library by Airbnb designed from the ground up to be human
friendly.
Datumbox - Machine Learning framework for rapid development of Machine Learning and
Statistical applications
ELKI - Java toolkit for data mining. (unsupervised: clustering, outlier detection etc.)
Encog - An advanced neural network and machine learning framework. Encog contains classes to
create a wide variety of networks, as well as support classes to normalize and process data
for these neural networks. Encog trains using multithreaded resilient propagation. Encog can
also make use of a GPU to further speed processing time. A GUI based workbench is also
provided to help model and train neural networks.
FlinkML in Apache Flink - Distributed machine learning library in Flink
H2O - ML engine that supports distributed learning on Hadoop, Spark or your laptop via APIs in
R, Python, Scala, REST/JSON.
htm.java - General Machine Learning library using Numenta's Cortical Learning Algorithm
java-deeplearning - Distributed Deep Learning Platform for Java, Clojure,Scala
Mahout - Distributed machine learning
Meka - An open source implementation of methods for multi-label classification and evaluation
(extension to Weka).
MLlib in Apache Spark - Distributed machine learning library in Spark
Neuroph - Neuroph is lightweight Java neural network framework
ORYX - Lambda Architecture Framework using Apache Spark and Apache Kafka with a specialization
for real-time large-scale machine learning.
Samoa SAMOA is a framework that includes distributed machine learning for data streams with an
interface to plug-in different stream processing platforms.
RankLib - RankLib is a library of learning to rank algorithms
rapaio - statistics, data mining and machine learning toolbox in Java
RapidMiner - RapidMiner integration into Java code
Stanford Classifier - A classifier is a machine learning tool that will take data items and
place them into one of k classes.
SmileMiner - Statistical Machine Intelligence & Learning Engine
SystemML - flexible, scalable machine learning (ML) language.
WalnutiQ - object oriented model of the human brain
Weka - Weka is a collection of machine learning algorithms for data mining tasks
LBJava - Learning Based Java is a modeling language for the rapid development of software
systems, offers a convenient, declarative syntax for classifier and constraint definition
directly in terms of the objects in the programmer's application.
```

## Riconoscimento vocale

```
CMU Sphinx - Open Source Toolkit For Speech Recognition purely based on Java speech
recognition library.
```

## Analisi dei dati / visualizzazione dei dati

```
Flink - Open source platform for distributed stream and batch data processing.
Hadoop - Hadoop/HDFS
Spark - Spark is a fast and general engine for large-scale data processing.
Storm - Storm is a distributed realtime computation system.
Impala - Real-time Query for Hadoop
DataMelt - Mathematics software for numeric computation, statistics, symbolic calculations,
data analysis and data visualization.
Dr. Michael Thomas Flanagan's Java Scientific Library
```

## Apprendimento approfondito

```
Deeplearning4j – Scalable deep learning for industry with parallel GPUs
```

Leggi Apprendimento automatico tramite Java online: https://riptutorial.com/it/machine-learning/topic/6175/apprendimento-automatico-tramite-java

# Capitolo 5: Apprendimento supervisionato

## Examples

### Classificazione

Immagina che un sistema voglia rilevare **mele** e **arance** in un cesto di frutta. Il sistema può raccogliere un frutto, estrarne alcune proprietà (ad es. Il peso di quel frutto).

Supponiamo che il sistema abbia un insegnante! che insegna al sistema quali oggetti sono le **mele** e quali sono le **arance** . Questo è un esempio di un problema di **classificazione supervisionato** . È supervisionato perché abbiamo esempi etichettati. È una classificazione perché l'output è una previsione di quale classe appartiene anche il nostro oggetto.

In questo esempio consideriamo 3 caratteristiche (proprietà / variabili esplicative):

1. il peso del frutto selezionato è maggiore di 0,5gram
2. è una dimensione superiore a 10 cm
3. è il colore è rosso

(0 significa No, e 1 significa Sì)

Quindi per rappresentare una mela / arancione abbiamo una serie (chiamata vettore) di 3 proprietà (spesso chiamata un vettore di funzionalità)

(es. [0,0,1] significa che questo peso di frutta non è superiore a quello di .5gram, e la sua dimensione non è superiore a 10 cm e il colore è rosso)

Quindi, prendiamo 10 frutti a caso e misuriamo le loro proprietà. L'insegnante (umano) etichetta quindi ogni frutto manualmente come mela => **[1]** o arancione => **[2]** .

es.) L'insegnante seleziona un frutto che è una mela. La rappresentazione di questa mela per il sistema potrebbe essere qualcosa del genere: **[1, 1, 1] => [1]** , Ciò significa che questo frutto ha **1.piccolo maggiore di .5gram** , **2.size maggiore di 10cm** e **3. il colore di questo frutto è rosso** e infine è una **mela** (=> [1])

Quindi per tutti i 10 frutti, l'insegnante etichetta ciascun frutto come mela [=> 1] o arancione [=> 2] e il sistema ha trovato le loro proprietà. come indovinate abbiamo una serie di vettori (che chiamano matrice) per rappresentare interi 10 frutti.

### Classificazione dei frutti

In questo esempio, un modello imparerà a classificare i frutti in base a determinate caratteristiche, utilizzando le *etichette* per l'allenamento.

| Peso | Colore | Etichetta |
|---|---|---|
| 0.5 | verde | Mela |
| 0.6 | viola | prugna |
| 3 | verde | anguria |
| 0.1 | rosso | ciliegia |
| 0.5 | rosso | Mela |

Qui un modello prenderà *Peso* e *Colore* come caratteristiche per prevedere l'Etichetta. Ad esempio [0.15, 'red'] dovrebbe dare come risultato una previsione 'cherry'.

## Introduzione all'apprendimento supervisionato

Ci sono molte situazioni in cui si hanno enormi quantità di dati e l'uso di cui deve classificare un oggetto in una delle diverse classi conosciute. Considera le seguenti situazioni:

*Banking:* quando una banca riceve una richiesta da un cliente per una carta bancaria, la banca deve decidere se emettere o meno la carta bancaria, in base alle caratteristiche dei suoi clienti che già godono delle carte per le quali è nota la storia creditizia.

*Medico:* uno potrebbe essere interessato a sviluppare un sistema medico che diagnostica un paziente sia che stia avendo o meno una particolare malattia, in base ai sintomi osservati e agli esami medici condotti su quel paziente.

*Finanza:* una società di consulenza finanziaria vorrebbe prevedere l'andamento del prezzo di uno stock che può essere classificato verso l'alto, verso il basso o senza tendenza in base a diverse caratteristiche tecniche che regolano il movimento dei prezzi.

*Espressione genica:* uno scienziato che analizza i dati dell'espressione genica vorrebbe identificare i geni e i fattori di rischio più rilevanti coinvolti nel cancro al seno, al fine di separare i pazienti sani da pazienti affetti da cancro al seno.

In tutti gli esempi precedenti, un oggetto è classificato in una delle varie classi *conosciute* , in base alle misurazioni fatte su un numero di caratteristiche, che può ritenere discriminare gli oggetti di classi diverse. Queste variabili sono chiamate variabili di *predittore* e l'etichetta di classe è chiamata variabile *dipendente* . Si noti che, in tutti gli esempi precedenti, la variabile dipendente è *categoriale* .

Per sviluppare un modello per il problema di classificazione, richiediamo, per ogni oggetto, dati su un insieme di caratteristiche prescritte insieme alle etichette di classe, a cui gli oggetti appartengono. Il set di dati è diviso in due set in un rapporto prescritto. Il più grande di questi set di dati è chiamato set di *dati di allenamento* e l'altro set di *dati di test* . Il set di dati di addestramento viene utilizzato nello sviluppo del modello. Poiché il modello viene sviluppato utilizzando osservazioni le cui etichette di classe sono note, questi modelli sono noti come modelli di *apprendimento supervisionato* .

Dopo aver sviluppato il modello, il modello deve essere valutato per le sue prestazioni utilizzando il set di dati di test. L'obiettivo di un modello di classificazione è di avere una probabilità minima di errata classificazione delle osservazioni non osservate. Le osservazioni non utilizzate nello sviluppo del modello sono note come osservazioni invisibili.

*L'induzione dell'albero decisionale* è una delle tecniche di costruzione del modello di classificazione. Il modello dell'albero decisionale costruito per la variabile dipendente categoriale è chiamato *albero di classificazione* . La variabile dipendente potrebbe essere numerica in alcuni problemi. Il modello dell'albero decisionale sviluppato per le variabili dipendenti numeriche è chiamato *Albero di regressione* .

## Regressione lineare

Poiché l' **apprendimento supervisionato** consiste in una variabile obiettivo o risultato (o variabile dipendente) che deve essere prevista da un determinato insieme di predittori (variabili indipendenti). Usando questi set di variabili, generiamo una funzione che mappa gli input alle uscite desiderate. Il processo di addestramento continua fino a quando il modello raggiunge il livello desiderato di accuratezza sui dati di allenamento.

> Pertanto, ci sono molti esempi di algoritmi di apprendimento supervisionato, quindi in questo caso vorrei concentrarmi sulla **regressione lineare**

Regressione lineare Viene utilizzato per stimare valori reali (costo di case, numero di chiamate, vendite totali, ecc.) In base a variabili continue. Qui stabiliamo una relazione tra variabili indipendenti e dipendenti adattando una linea migliore. Questa linea di adattamento ottimale è nota come linea di regressione e rappresentata da un'equazione lineare Y = a * X + b.

Il modo migliore per comprendere la regressione lineare è rivivere questa esperienza dell'infanzia. Diciamo, chiedi a un bambino in quinta elementare di organizzare le persone della sua classe aumentando l'ordine di peso, senza chiedere loro i loro pesi! Cosa pensi che farà il bambino? Lui / lei probabilmente guarderebbe (visivamente analizzando) all'altezza e alla costruzione di persone e li sistemerebbe usando una combinazione di questi parametri visibili.

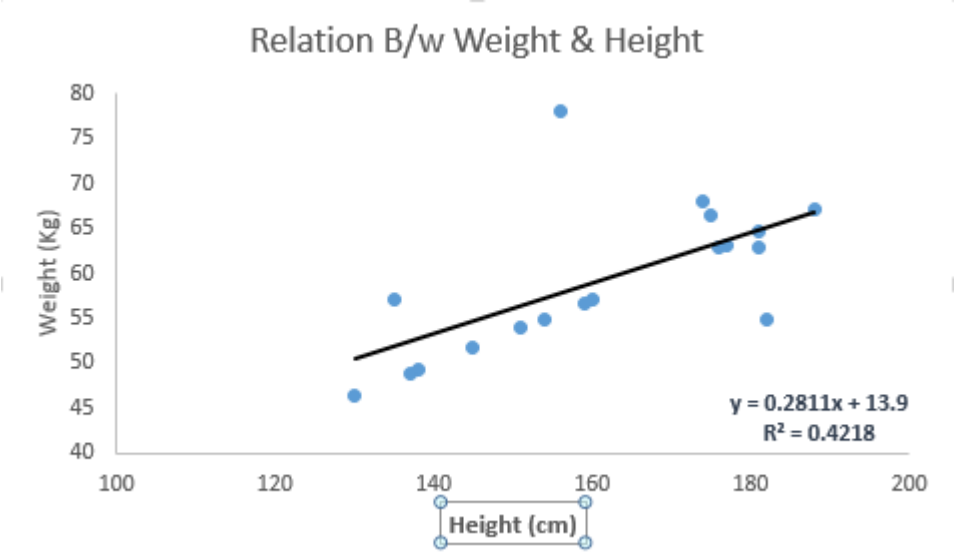
Questa è una regressione lineare nella vita reale! Il bambino ha effettivamente capito che l'altezza e la struttura sarebbero correlate al peso da una relazione, che somiglia all'equazione di cui sopra.

In questa equazione:

```
Y – Dependent Variable
a – Slope
X – Independent variable
b – Intercept
```

Questi coefficienti a e b sono derivati basati sulla minimizzazione della somma della differenza quadratica della distanza tra i punti dati e la linea di regressione.

Guarda l'esempio qui sotto. Qui abbiamo identificato la migliore linea di adattamento con equazione lineare **y = 0,2811x + 13,9** . Ora usando questa equazione, possiamo trovare il peso, conoscendo l'altezza di una persona.




La regressione lineare è principalmente di due tipi: regressione lineare semplice e regressione lineare multipla. La semplice regressione lineare è caratterizzata da una variabile indipendente. E, la regressione lineare multipla (come suggerisce il nome) è caratterizzata da più variabili (più di 1) indipendenti. Mentre trovi la migliore linea di adattamento, puoi adattare una regressione polinomiale o curvilinea. E questi sono noti come regressione polinomiale o curvilinea.

**Solo un suggerimento sull'implementazione della regressione lineare in Python**

```
#Import Library
#Import other necessary libraries like pandas, numpy...
from sklearn import linear_model

#Load Train and Test datasets
#Identify feature and response variable(s) and values must be numeric and numpy arrays

x_train=input_variables_values_training_datasets
y_train=target_variables_values_training_datasets
x_test=input_variables_values_test_datasets

# Create linear regression object

linear = linear_model.LinearRegression()

# Train the model using the training sets and check score

linear.fit(x_train, y_train)
linear.score(x_train, y_train)

#Equation coefficient and Intercept

print('Coefficient: \n', linear.coef_)
print('Intercept: \n', linear.intercept_)

#Predict Output

predicted= linear.predict(x_test)
```

Ho fornito uno spaccato sulla comprensione dell'apprendimento supervisionato scavando verso il basso all'algoritmo di regressione lineare insieme a uno snippet di codice Python.

Leggi Apprendimento supervisionato online: https://riptutorial.com/it/machine-learning/topic/2673/apprendimento-supervisionato

# Capitolo 6: Elaborazione del linguaggio naturale

## introduzione

La PNL è un modo per i computer di analizzare, comprendere e derivare il significato dal linguaggio umano in modo intelligente e utile. Utilizzando la PNL, gli sviluppatori possono organizzare e strutturare le conoscenze per eseguire attività quali riepilogo automatico, traduzione, riconoscimento di entità con nome, estrazione delle relazioni, analisi dei sentimenti, riconoscimento vocale e segmentazione degli argomenti.

## Examples

### Corrispondenza del testo o somiglianza

> Una delle aree importanti della PNL è la corrispondenza degli oggetti di testo per trovare somiglianze. Importanti applicazioni di corrispondenza testuale includono la correzione automatica dello spelling, la deduplicazione dei dati e l'analisi del genoma, ecc. A seconda del requisito sono disponibili diverse tecniche di abbinamento del testo. Quindi, abbiamo; **Levenshtein Distance**

La distanza di Levenshtein tra due stringhe è definita come il numero minimo di modifiche necessarie per trasformare una stringa nell'altra, con le operazioni di modifica consentite che sono l'inserimento, la cancellazione o la sostituzione di un singolo carattere.

Di seguito è riportata l'implementazione per calcoli di memoria efficienti.

```
def levenshtein(s1,s2):

 if len(s1) > len(s2):
    s1,s2 = s2,s1
distances = range(len(s1) + 1)

for index2,char2 in enumerate(s2):
    newDistances = [index2+1]
    for index1,char1 in enumerate(s1):
        if char1 == char2:
            newDistances.append(distances[index1])
        else:
             newDistances.append(1 + min((distances[index1], distances[index1+1],
newDistances[-1])))
             distances = newDistances

             return distances[-1]

print(levenshtein("analyze","analyse"))
```

Leggi Elaborazione del linguaggio naturale online: https://riptutorial.com/it/machine-

learning/topic/10734/elaborazione-del-linguaggio-naturale

# Capitolo 7: Iniziare con Machine Learning utilizzando Apache spark MLib

## introduzione

Apache spark MLib fornisce (JAVA, R, PYTHON, SCALA) 1.) Vari algoritmi di apprendimento automatico su regressione, classificazione, clustering, filtraggio collaborativo che sono approcci maggiormente utilizzati nell'apprendimento automatico. 2.) Supporta l'estrazione delle caratteristiche, la trasformazione, ecc. 3.) Permette ai professionisti dei dati di risolvere i loro problemi di apprendimento automatico (nonché calcolo grafico, streaming e elaborazione interattiva interattiva in tempo reale) in modo interattivo e su scala molto più ampia.

## Osservazioni

Si prega di fare riferimento qui sotto per saperne di più su scintilla MLib

1. http://spark.apache.org/docs/latest/ml-guide.html
2. https://mapr.com/ebooks/spark/

## Examples

### Scrivi il tuo primo problema di classificazione usando il modello di regressione logistica

Sto usando eclipse qui e devi aggiungere la dipendenza fornita sotto al tuo pom.xml

1.) **POM.XML**

```
    <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.predection.classification</groupId>
  <artifactId>logisitcRegression</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>logisitcRegression</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
     <!-- Spark -->
```

```
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.11</artifactId>
            <version>2.1.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-mllib_2.10</artifactId>
            <version>2.1.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-sql_2.11</artifactId>
            <version>2.1.0</version>
        </dependency>
    </dependencies>
</project>
```

2.) APP.JAVA (la tua classe di applicazione)

Stiamo facendo una classificazione in base al paese, alle ore e si fa clic sulla nostra etichetta.

```
    package com.predection.classification.logisitcRegression;

import org.apache.spark.SparkConf;
import org.apache.spark.ml.classification.LogisticRegression;
import org.apache.spark.ml.classification.LogisticRegressionModel;
import org.apache.spark.ml.feature.StringIndexer;
import org.apache.spark.ml.feature.VectorAssembler;
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;
import org.apache.spark.sql.types.StructField;
import org.apache.spark.sql.types.StructType;
import java.util.Arrays;
import java.util.List;
import org.apache.spark.sql.RowFactory;
import static org.apache.spark.sql.types.DataTypes.*;

/**
 * Classification problem using Logistic Regression Model
 *
 */

public class App
{
    public static void main( String[] args )
    {
        SparkConf sparkConf = new SparkConf().setAppName("JavaLogisticRegressionExample");

        // Creating spark session
        SparkSession sparkSession = SparkSession.builder().config(sparkConf).getOrCreate();

        StructType schema = createStructType(new StructField[]{
                  createStructField("id", IntegerType, false),
                  createStructField("country", StringType, false),
                  createStructField("hour", IntegerType, false),
```

```
                  createStructField("clicked", DoubleType, false)
                });

                List<Row> data = Arrays.asList(
                  RowFactory.create(7, "US", 18, 1.0),
                  RowFactory.create(8, "CA", 12, 0.0),
                  RowFactory.create(9, "NZ", 15, 1.0),




 RowFactory.create(10,"FR", 8, 0.0),
                  RowFactory.create(11, "IT", 16, 1.0),
              RowFactory.create(12, "CH", 5, 0.0),
              RowFactory.create(13, "AU", 20, 1.0)
            );


Dataset<Row> dataset = sparkSession.createDataFrame(data, schema);

// Using stringindexer transformer to transform string into index
 dataset = new
StringIndexer().setInputCol("country").setOutputCol("countryIndex").fit(dataset).transform(dataset);


// creating feature vector using dependent variables countryIndex, hours are features and
clicked is label
VectorAssembler assembler = new VectorAssembler()
        .setInputCols(new String[] {"countryIndex", "hour"})
        .setOutputCol("features");

    Dataset<Row> finalDS = assembler.transform(dataset);

    // Split the data into training and test sets (30% held out for
    // testing).
        Dataset<Row>[] splits = finalDS.randomSplit(new double[] { 0.7, 0.3 });
        Dataset<Row> trainingData = splits[0];
        Dataset<Row> testData = splits[1];
        trainingData.show();
        testData.show();
        // Building LogisticRegression Model
        LogisticRegression lr = new
LogisticRegression().setMaxIter(10).setRegParam(0.3).setElasticNetParam(0.8).setLabelCol("clicked");


        // Fit the model
        LogisticRegressionModel lrModel = lr.fit(trainingData);

        // Transform the model, and predict class for test dataset
        Dataset<Row> output = lrModel.transform(testData);
        output.show();
    }
}
```

3.) Per eseguire questa applicazione, prima eseguire `mvn-clean-package` sul progetto di applicazione, creerebbe jar. 4.) Aprire la directory spark root e inviare questo lavoro

```
bin/spark-submit --class com.predection.regression.App --master local[2] ./regression-0.0.1-
SNAPSHOT.jar(path to the jar file)
```

5.) Dopo averlo inviato, visualizza i dati di allenamento

```
17/05/13 11:38:36 INFO CodeGenerator: Code generated in 24.888221 ms
+---+-------+----+-------+------------+----------+
| id|country|hour|clicked|countryIndex|  features|
+---+-------+----+-------+------------+----------+
|  7|     US|  18|    1.0|         1.0|[1.0,18.0]|
|  8|     CA|  12|    0.0|         6.0|[6.0,12.0]|
|  9|     NZ|  15|    1.0|         0.0|[0.0,15.0]|
| 10|     FR|   8|    0.0|         4.0| [4.0,8.0]|
| 13|     AU|  20|    1.0|         2.0|[2.0,20.0]|
+---+-------+----+-------+------------+----------+

17/05/13 11:38:36 INFO CodeGenerator: Code generated in 31.184275 ms
17/05/13 11:38:36 INFO SparkContext: Starting job: show at App.java:67
```

6.) allo stesso modo i dati di test

```
17/05/13 11:38:36 INFO TaskSchedulerImpl: Removed TaskSet 3.0, whose tasks have all completed, from pool
17/05/13 11:38:36 INFO DAGScheduler: ResultStage 3 (show at App.java:67) finished in 0.016 s
17/05/13 11:38:36 INFO DAGScheduler: Job 2 finished: show at App.java:67, took 0.027935 s
+---+-------+----+-------+------------+----------+
| id|country|hour|clicked|countryIndex|  features|
+---+-------+----+-------+------------+----------+
| 11|     IT|  16|    1.0|         5.0|[5.0,16.0]|
| 12|     CH|   5|    0.0|         3.0| [3.0,5.0]|
+---+-------+----+-------+------------+----------+

17/05/13 11:38:36 INFO CodeGenerator: Code generated in 27.316114 ms
17/05/13 11:38:36 INFO Instrumentation: LogisticRegression-logreg_3fdceb703058-1998603857-1: training: num
as)
17/05/13 11:38:36 INFO Instrumentation: LogisticRegression-logreg_3fdceb703058-1998603857-1: {"regParam":0
17/05/13 11:38:36 INFO SparkContext: Starting job: treeAggregate at LogisticRegression.scala:352
17/05/13 11:38:36 INFO DAGScheduler: Got job 3 (treeAggregate at LogisticRegression.scala:352) with 1 outp
17/05/13 11:38:36 INFO DAGScheduler: Final stage: ResultStage 4 (treeAggregate at LogisticRegression.scala
17/05/13 11:38:36 INFO DAGScheduler: Parents of final stage: List()
17/05/13 11:38:36 INFO DAGScheduler: Missing parents: List()
17/05/13 11:38:36 INFO DAGScheduler: Submitting ResultStage 4 (MapPartitionsRDD[27] at treeAggregate at Lo
```

7.) Ed ecco il risultato della previsione sotto la colonna di predizione

```
17/05/13 11:38:37 INFO CodeGenerator: Code generated in 22.111904 ms
+---+-------+----+-------+------------+----------+--------------------+--------------------+----------+
| id|country|hour|clicked|countryIndex|  features|       rawPrediction|         probability|prediction|
+---+-------+----+-------+------------+----------+--------------------+--------------------+----------+
| 11|     IT|  16|    1.0|         5.0|[5.0,16.0]|[-0.0683991645753...|[0.48290687244237...|       1.0|
| 12|     CH|   5|    0.0|         3.0| [3.0,5.0]|[0.38550601723144...|[0.59520039795209...|       0.0|
+---+-------+----+-------+------------+----------+--------------------+--------------------+----------+

17/05/13 11:38:37 INFO SparkContext: Invoking stop() from shutdown hook
17/05/13 11:38:37 INFO SparkUI: Stopped Spark web UI at http://192.168.1.7:4041
17/05/13 11:38:37 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
17/05/13 11:38:37 INFO MemoryStore: MemoryStore cleared
17/05/13 11:38:37 INFO BlockManager: BlockManager stopped
```

Leggi Iniziare con Machine Learning utilizzando Apache spark MLib online: https://riptutorial.com/it/machine-learning/topic/9854/iniziare-con-machine-learning-utilizzando-apache-spark-mlib

# Capitolo 8: Metriche di valutazione

## Examples

### Area Under the Curve of the Receiver Operating Characteristic (AUROC)

L' **AUROC** è una delle metriche più comunemente utilizzate per valutare le prestazioni di un classificatore. Questa sezione spiega come calcolarlo.

**L'AUC** (Area Under the Curve) è usata per la maggior parte del tempo per indicare AUROC, che è una cattiva pratica in quanto l'AUC è ambigua (potrebbe essere qualsiasi curva) mentre AUROC non lo è.

### Panoramica - Abbreviazioni

| Abbreviazione | Senso |
|---|---|
| AUROC | Area Under the Curve of Receiver Operating Characteristic |
| AUC | Area sotto il Curce |
| ROC | Caratteristica operativa del ricevitore |
| TP | Veri positivi |
| TN | Veri negativi |
| FP | Falsi positivi |
| FN | Falsi negativi |
| TPR | True Positive Rate |
| FPR | False Positive Rate |

---

## Interpretazione di AUROC

L'AUROC ha diverse interpretazioni equivalenti :

- L'aspettativa che un positivo casuale estratto in modo uniforme sia classificato prima di un negativo casuale disegnato in modo uniforme.
- La proporzione prevista di positivi è stata classificata prima di un negativo casuale uniformemente tracciato.
- L'aspettativa vero tasso positivo se la classifica è divisa appena prima di un negativo casuale disegnato in modo uniforme.

- La proporzione attesa di negativi si è classificata dopo un positivo casuale uniformemente tracciato.
- Il tasso falso positivo atteso se la classifica viene divisa subito dopo un positivo casuale tracciato uniformemente.

---

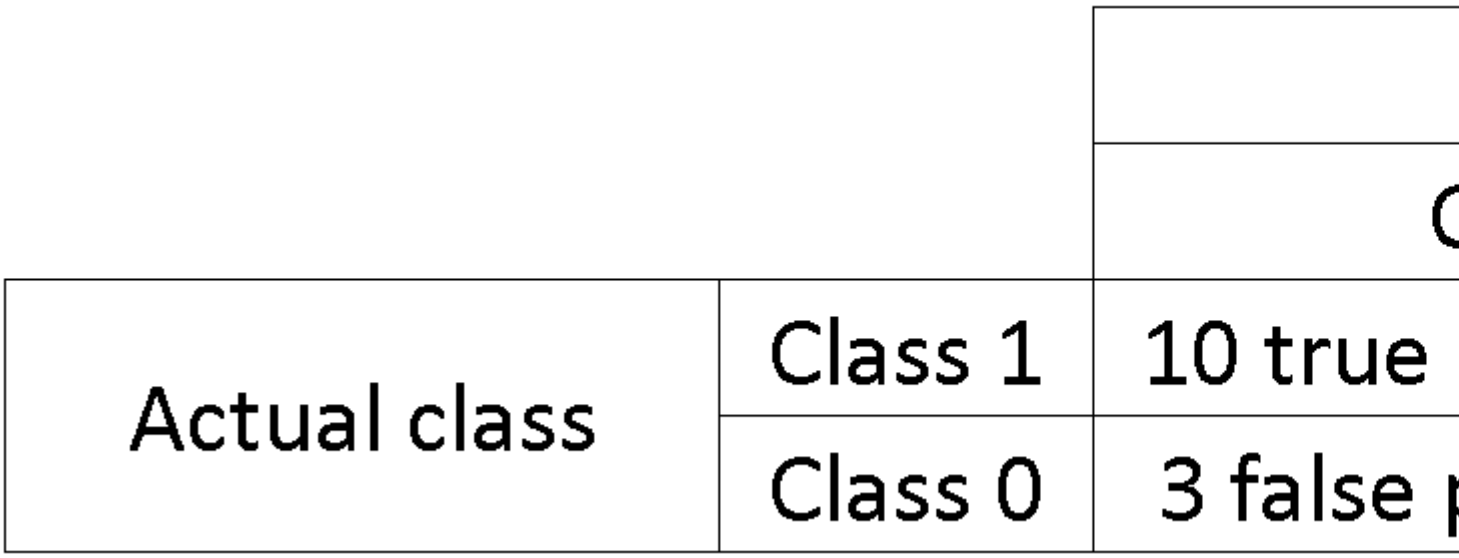
## Calcolare l'AUROC

Supponiamo di avere un classificatore probabilistico, binario come la regressione logistica.

Prima di presentare la curva **ROC** (= curva caratteristica operativa del ricevitore), deve essere compreso il concetto di **matrice** di **confusione** . Quando facciamo una previsione binaria, ci possono essere 4 tipi di risultati:

- Prevediamo **0** mentre la classe è in realtà **0** : questo è chiamato **True Negative** , ovvero prevediamo correttamente che la classe sia negativa (0). *Ad esempio, un antivirus non ha rilevato un file innocuo come un virus.*
- Prevediamo **0** mentre la classe è in realtà **1** : questo è chiamato **False Negative** , ovvero prevediamo erroneamente che la classe sia negativa (0). *Ad esempio, un antivirus non è riuscito a rilevare un virus.*
- Prevediamo **1** mentre la classe è in realtà **0** : questo è chiamato **False Positive** , ovvero prevediamo erroneamente che la classe sia positiva (1). *Ad esempio, un antivirus considera un file innocuo come un virus.*
- Prevediamo **1** mentre la classe è in realtà **1** : questo è chiamato **True Positive** , ovvero prevediamo correttamente che la classe sia positiva (1). *Ad esempio, un antivirus ha rilevato correttamente un virus.*

Per ottenere la matrice di confusione, esaminiamo tutte le previsioni fatte dal modello e contiamo quante volte si verificano ciascuno di questi 4 tipi di risultati:




In questo esempio di una matrice di confusione, tra i 50 punti dati classificati, 45 sono classificati

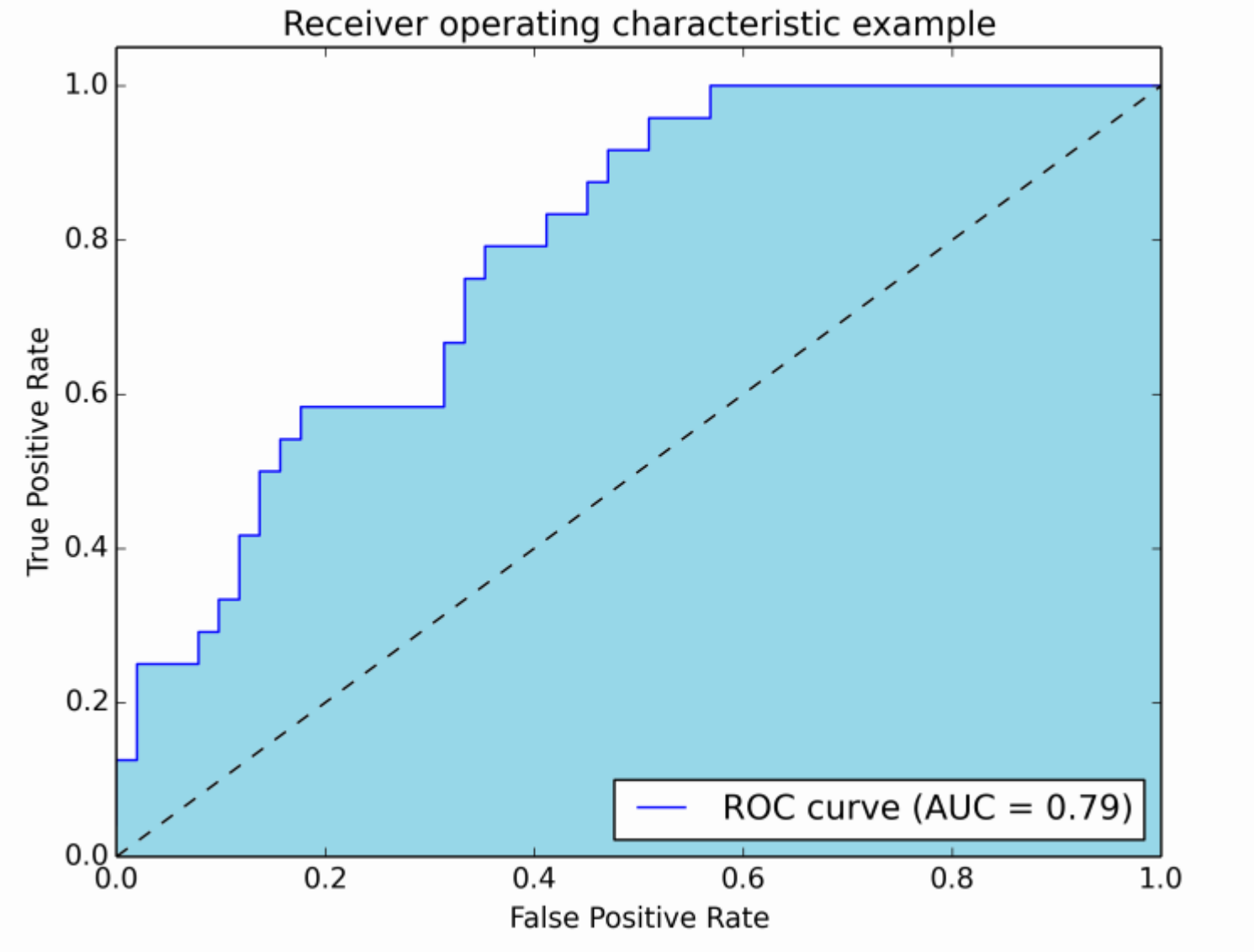
correttamente e 5 sono classificati in modo errato.

Dal momento che per confrontare due diversi modelli è spesso più conveniente avere una singola metrica piuttosto che più, calcoliamo due parametri dalla matrice di confusione, che successivamente combineremo in uno:

- Vero tasso positivo ( **TPR** ), alias. sensibilità, hit rate e recall , che è definito come $\frac{TP}{TP+FN}$ . Intuitivamente, questa metrica corrisponde alla proporzione di punti dati positivi che sono correttamente considerati positivi, rispetto a tutti i punti dati positivi. In altre parole, più alto TPR, meno punti dati positivi ci mancheranno.
- False positive rate ( **FPR** ), aka. fall-out , che è definito come $\frac{FP}{FP+TN}$ . Intuitivamente, questa metrica corrisponde alla proporzione di punti di dati negativi erroneamente considerati positivi rispetto a tutti i punti di dati negativi. In altre parole, il più alto FPR, più punti di dati negativi ci mancheranno.

Per combinare FPR e TPR in una singola metrica, calcoliamo prima le due precedenti metriche con molte soglie diverse (ad esempio $0.00, 0.01 \ldots 1.00$ ) per la regressione logistica, quindi tracciarli su un singolo grafico, con i valori FPR sull'asse delle ascisse e i valori TPR sull'ordinata. La curva risultante è chiamata curva ROC e la metrica che consideriamo è l'AUC di questa curva, che chiamiamo AUROC.

La figura seguente mostra graficamente AUROC:




In questa figura, l'area blu corrisponde all'Area sotto la curva del Receiver Operating Characteristic (AUROC). La linea tratteggiata nella diagonale presenta la curva ROC di un predittore casuale: ha un AUROC di 0,5. Il predittore casuale è comunemente usato come base per vedere se il modello è utile.

## Matrice di confusione

Una matrice di confusione può essere utilizzata per valutare un classificatore, in base a una serie di dati di test per i quali sono noti i valori reali. È uno strumento semplice, che aiuta a fornire una buona panoramica visiva delle prestazioni dell'algoritmo in uso.

Una matrice di confusione è rappresentata come una tabella. In questo esempio vedremo una **matrice di confusione per un classificatore binario** .

| n=165 | Predicted: NO | Predicted: YES | |
|---|---|---|---|
| **Actual: NO** | TN = 50 | FP = 10 | 60 |
| **Actual: YES** | FN = 5 | TP = 100 | 105 |
| | 55 | 110 | |

Sul lato sinistro, si può vedere la classe Actual (che viene etichettata come *YES* o *NO* ), mentre la parte superiore indica la classe che viene predetta e emessa (di nuovo *YES* o *NO* ).

Ciò significa che 50 istanze di test, che in realtà *NON sono* istanze, sono state correttamente etichettate dal classificatore come *NO* . Questi sono chiamati **True Negatives (TN)** . Al contrario, 100 istanze *YES* effettive, sono state correttamente etichettate dal classificatore come istanze *YES* . Questi sono chiamati i **veri positivi (TP)** .

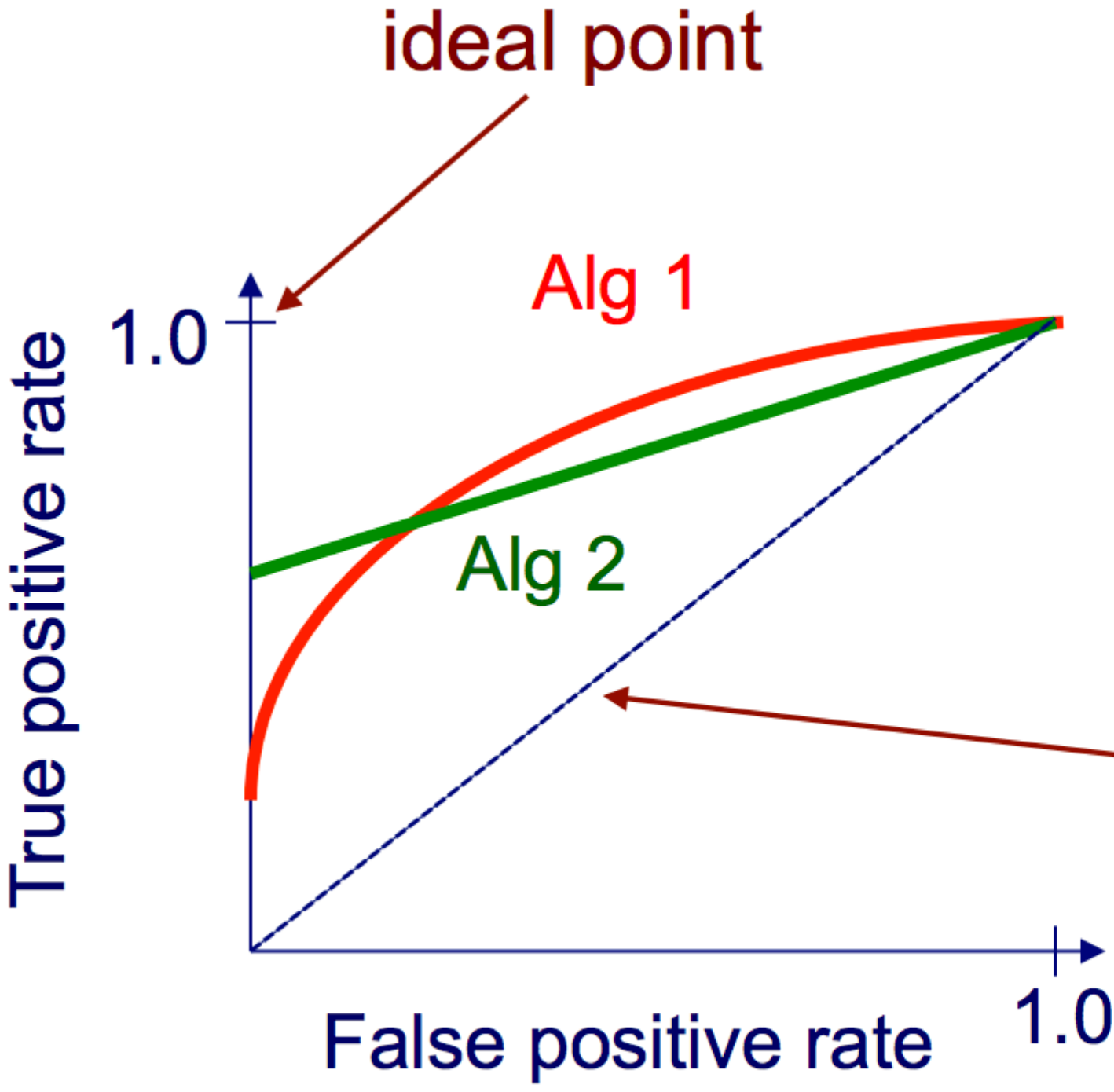
5 istanze *YES* effettive, sono state indicate erroneamente dal classificatore. Questi sono chiamati i **falsi negativi (FN)** . Inoltre 10 istanze *NO* , sono state considerate istanze *SÌ* dal classificatore, quindi questi sono **falsi positivi (FP)** .

Sulla base di questi **FP** , **TP** , **FN** e **TN** , possiamo trarre ulteriori conclusioni.

- **True Positive Rate** :
    - *Cerca di rispondere:* quando un'istanza è effettivamente *SÌ* , quanto spesso il classificatore prevede *SÌ* ?
    - *Può essere calcolato come segue:* **TP** / # istanze *YES* effettive = 100/105 = 0,95
- **False Positive Rate** :
    - *Cerca di rispondere:* quando un'istanza è effettivamente *NO* , quanto spesso il classificatore prevede *SÌ* ?
    - *Può essere calcolato come segue:* **FP** / # effettivi *NO* istanze = 10/60 = 0,17

## Curve ROC

Una curva ROC (Receiver Operating Characteristic) traccia la velocità TP rispetto alla frequenza FP come una soglia sulla sicurezza di un'istanza che è positiva.




**Algoritmo per la creazione di una curva ROC**

1. ordina le previsioni del test in base alla certezza che ogni istanza è positiva
2. scorrere l'elenco ordinato da alta a bassa confidenza

   io. individuare una soglia tra istanze con classi opposte (mantenendo le istanze con lo stesso valore di confidenza sullo stesso lato della soglia)

ii. calcolare TPR, FPR per le istanze al di sopra della soglia

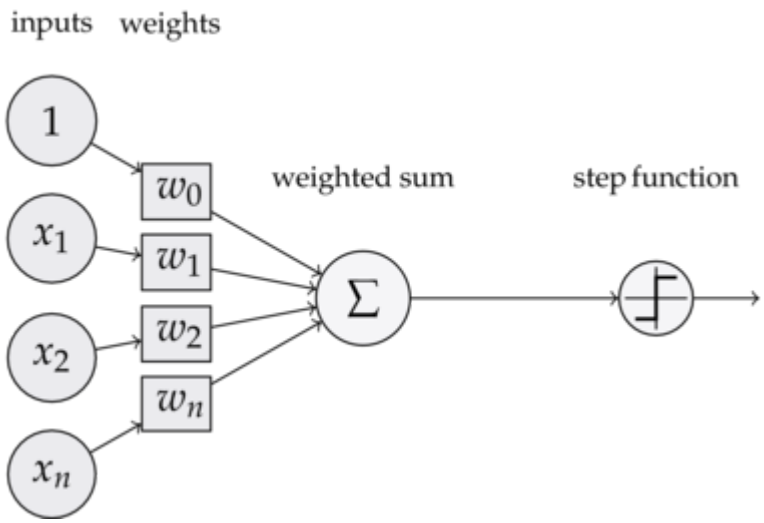
iii. uscita (FPR, TPR) coordinata

Leggi Metriche di valutazione online: https://riptutorial.com/it/machine-learning/topic/4132/metriche-di-valutazione

# Capitolo 9: Perceptron

## Examples

### Cos'è esattamente un perceptron?




Al suo interno un modello perceptron è uno dei più semplici algoritmi di **apprendimento supervisionato** per la classificazione binaria . È un tipo di classificatore lineare , ovvero un algoritmo di classificazione che fa le sue previsioni basate su una funzione di predittore lineare che combina un insieme di pesi con il vettore di caratteristiche. Un modo più intuitivo di pensare è come una **rete neurale con un solo neurone** .

Il modo in cui funziona è molto semplice. Ottiene un vettore di valori di input ***x*** di cui ogni elemento è una caratteristica del nostro set di dati.

# Un esempio:

Diciamo che vogliamo classificare se un oggetto è una bicicletta o una macchina. Per il gusto di questo esempio, diciamo che vogliamo selezionare 2 funzioni. L'altezza e la larghezza dell'oggetto. In questo caso ***x = [x1, x2]*** dove ***x1*** è l'altezza e ***x2*** è la larghezza.

Quindi, una volta ottenuto il vettore di input ***x*** , vogliamo moltiplicare ogni elemento di quel vettore con un peso. Di solito più alto è il valore del peso, più importante è la caratteristica. Se per esempio abbiamo usato il ***colore*** come caratteristica ***x3*** e c'è una bicicletta rossa e una rossa, il percettore imposterà un peso molto basso in modo che il colore non influenzi la previsione finale.

Bene, abbiamo moltiplicato i 2 vettori ***x*** e ***w*** e abbiamo ottenuto un vettore. Ora abbiamo bisogno di sommare gli elementi di questo vettore. Un modo intelligente per farlo è invece di moltiplicare semplicemente ***x*** per ***w*** possiamo moltiplicare ***x*** per ***wT*** dove ***T*** sta per transpose. Possiamo immaginare la ***trasposizione*** di un vettore come una versione ruotata del vettore. Per maggiori informazioni puoi leggere la pagina di Wikipedia . Essenzialmente prendendo la trasposizione del

vettore ***w*** otteniamo un vettore ***Nx1*** invece di un ***1xN*** . Quindi, se ora moltiplichiamo il nostro vettore di input con dimensione ***1xN*** con questo vettore di peso ***Nx1*** otterremo un vettore ***1x1*** (o semplicemente un singolo valore) che sarà uguale a ***x1 * w1 + x2 * w2 + ... + xn * wn*** . Fatto ciò, ora abbiamo la nostra previsione. Ma c'è un'ultima cosa. Questa previsione probabilmente non sarà una semplice 1 o -1 per essere in grado di classificare un nuovo campione. Quindi quello che possiamo fare è semplicemente dire quanto segue: Se la nostra previsione è maggiore di 0, allora diciamo che il campione appartiene alla classe 1, altrimenti se la previsione è minore di zero diciamo che il campione appartiene alla classe -1. Questa è chiamata una funzione di passaggio .

Ma come possiamo ottenere i pesi corretti in modo da correggere le previsioni? In altre parole, come possiamo ***addestrare il*** nostro modello perceptron?

Bene, nel caso del perceptron non abbiamo bisogno di fantasiose equazioni matematiche per ***addestrare il*** nostro modello. I nostri pesi possono essere regolati con la seguente equazione:

***Δw = eta * (y - predizione) * x (i)***

dove ***x (i)*** è la nostra caratteristica (x1 per esempio per peso 1, x2 per w2 e così via ...).

Inoltre ho notato che c'è una variabile chiamata ***eta*** che è il tasso di apprendimento. Potete immaginare il tasso di apprendimento quanto grande vogliamo il cambiamento dei pesi. Un buon tasso di apprendimento si traduce in un algoritmo di apprendimento veloce. Un valore troppo elevato di ***eta*** può comportare una quantità crescente di errori ad ogni epoca e comporta che il modello faccia delle previsioni davvero sbagliate e non converga mai. Troppo basso di un tasso di apprendimento può avere come risultato che il modello impiega troppo tempo per convergere. (Di solito un buon valore per impostare ***eta*** su per il modello perceptron è 0.1 ma può differire da caso a caso).

Infine alcuni di voi potrebbero aver notato che il primo input è una costante (1) e viene moltiplicato per w0. Quindi cos'è esattamente? Per ottenere una buona previsione, dobbiamo aggiungere un pregiudizio. E questo è esattamente ciò che è costante.

Per modificare il peso del termine di bias usiamo la stessa equazione che abbiamo fatto per gli altri pesi, ma in questo caso non lo moltiplichiamo per l'input (perché l'input è una costante 1 e quindi non dobbiamo):

***Δw = eta * (y - previsione)***

Quindi questo è fondamentalmente come funziona un semplice modello perceptron! Una volta addestrati i nostri pesi, possiamo dargli nuovi dati e avere le nostre previsioni.

# NOTA:

Il modello Perceptron ha un importante svantaggio! Non convergerà mai (e troverà i pesi perfetti) se i dati non sono linearmente separabili , il che significa che è in grado di separare le 2 classi in uno spazio di caratteristiche da una linea retta. Quindi, per evitare che sia una buona pratica aggiungere un numero fisso di iterazioni in modo che il modello non sia bloccato a regolare i pesi

che non saranno mai perfettamente sintonizzati.

## Implementazione di un modello Perceptron in C ++

In questo esempio passerò attraverso l'implementazione del modello perceptron in C ++ in modo che tu possa avere un'idea migliore di come funziona.

Per prima cosa è una buona pratica scrivere un semplice algoritmo di ciò che vogliamo fare.

Algoritmo:

1. Crea il vettore per i pesi e inizializza a 0 (Non dimenticare di aggiungere il termine di bias)
2. Continua a regolare i pesi finché non riceviamo 0 errori o un conteggio di errori basso.
3. Fai previsioni su dati invisibili.

Avendo scritto un algoritmo semplicissimo, scriviamo ora alcune delle funzioni di cui avremo bisogno.

- Avremo bisogno di una funzione per calcolare l'input della rete (ei **** x wT*** moltiplicando gli input tempo i pesi)
- Una funzione di passaggio in modo da ottenere una previsione di 1 o -1
- E una funzione che trova i valori ideali per i pesi.

Quindi, senza ulteriori indugi, andiamo subito al suo interno.

Iniziamo semplice creando una classe perceptron:

```
class perceptron
{
public:

private:

};
```

Ora aggiungiamo le funzioni di cui avremo bisogno.

```
class perceptron
{
public:
    perceptron(float eta,int epochs);
    float netInput(vector<float> X);
    int predict(vector<float> X);
    void fit(vector< vector<float> > X, vector<float> y);
private:

};
```

Si noti come la funzione ***adatta*** prende come argomento un vettore di vettore <float>. Questo perché il nostro set di dati di formazione è una matrice di input. Essenzialmente possiamo immaginare che la matrice come una coppia di vettori ***x*** impilasse l'una sopra l'altra e ogni colonna di quella matrice fosse una caratteristica.

Infine aggiungiamo i valori che la nostra classe deve avere. Come il vettore ***w*** per contenere i pesi, il numero di ***epoche*** che indica il numero di passaggi che faremo sopra il set di dati di addestramento. E l' ***eta*** costante che è il tasso di apprendimento di cui moltiplicheremo ogni aggiornamento di peso per rendere più veloce la procedura di allenamento componendo questo valore o se ***eta*** è troppo alto possiamo comporlo per ottenere il risultato ideale (per la maggior parte delle applicazioni del perceptron suggerirei un valore ***eta*** di 0,1).

```
class perceptron
{
public:
    perceptron(float eta,int epochs);
    float netInput(vector<float> X);
    int predict(vector<float> X);
    void fit(vector< vector<float> > X, vector<float> y);
private:
    float m_eta;
    int m_epochs;
    vector < float > m_w;
};
```

Ora con il nostro set di lezioni. È tempo di scrivere ognuna delle funzioni.

Inizieremo dal costruttore ( **perceptron (float eta, int epochs);** )

```
perceptron::perceptron(float eta, int epochs)
{
    m_epochs = epochs; // We set the private variable m_epochs to the user selected value
    m_eta = eta; // We do the same thing for eta
}
```

Come puoi vedere, quello che faremo è roba molto semplice. Quindi passiamo ad un'altra semplice funzione. La funzione di previsione ( ***int prevedere (vettore X);*** ). Ricordare che ciò che tutto ***predire*** funzione fa sta prendendo l'ingresso rete e restituisce un valore 1 se il ***netInput*** è maggiore di 0 e -1 otherwhise.

```
int perceptron::predict(vector<float> X)
{
    return netInput(X) > 0 ? 1 : -1; //Step Function
}
```

Si noti che abbiamo usato un'istruzione inline se per semplificarci la vita. Ecco come funziona la dichiarazione inline if:

***condizione? if_true: else***

Fin qui tutto bene. Passiamo ad implementare la funzione ***netInput*** ( ***float netInput (vector X);*** )

Il netInput fa quanto segue; ***moltiplica il vettore di input per la trasposizione del vettore di pesi***

***x * wT***

In altre parole, moltiplica ciascun elemento del vettore di input ***x*** dall'elemento corrispondente del vettore di pesi ***w,*** quindi prende la loro somma e aggiunge il bias.

***(x1 * w1 + x2 * w2 + ... + xn * wn) + bias***

***bias = 1 * w0***

```
float perceptron::netInput(vector<float> X)
{
    // Sum(Vector of weights * Input vector) + bias
    float probabilities = m_w[0]; // In this example I am adding the perceptron first
    for (int i = 0; i < X.size(); i++)
    {
        probabilities += X[i] * m_w[i + 1]; // Notice that for the weights I am counting
        // from the 2nd element since w0 is the bias and I already added it first.
    }
    return probabilities;
}
```

Bene, ora siamo praticamente all'ultima cosa che dobbiamo fare è scrivere la funzione di ***adattamento*** che modifica i pesi.

```
void perceptron::fit(vector< vector<float> > X, vector<float> y)
{
    for (int i = 0; i < X[0].size() + 1; i++) // X[0].size() + 1 -> I am using +1 to add the
bias term
    {
        m_w.push_back(0); // Setting each weight to 0 and making the size of the vector
        // The same as the number of features (X[0].size()) + 1 for the bias term
    }
    for (int i = 0; i < m_epochs; i++) // Iterating through each epoch
    {
        for (int j = 0; j < X.size(); j++) // Iterating though each vector in our training
Matrix
        {
            float update = m_eta * (y[j] - predict(X[j])); //we calculate the change for the
weights
            for (int w = 1; w < m_w.size(); w++){ m_w[w] += update * X[j][w - 1]; } // we
update each weight by the update * the training sample
            m_w[0] = update; // We update the Bias term and setting it equal to the update
        }
    }
}
```

Quindi era essenzialmente questo. Con solo 3 funzioni ora abbiamo una classe perceptron funzionante che possiamo usare per fare previsioni!

Nel caso in cui si desideri copiare e incollare il codice e provarlo. Ecco l'intera classe (ho aggiunto alcune funzionalità extra come la stampa del vettore di pesi e gli errori in ogni epoca e ho aggiunto l'opzione per importare / esportare pesi).

Ecco il codice:

L'intestazione della classe:

```
class perceptron
{
public:
    perceptron(float eta,int epochs);
    float netInput(vector<float> X);
    int predict(vector<float> X);
    void fit(vector< vector<float> > X, vector<float> y);
    void printErrors();
    void exportWeights(string filename);
    void importWeights(string filename);
    void printWeights();
private:
    float m_eta;
    int m_epochs;
    vector < float > m_w;
    vector < float > m_errors;
};
```

Il file class.cpp con le funzioni:

```
perceptron::perceptron(float eta, int epochs)
{
    m_epochs = epochs;
    m_eta = eta;
}

void perceptron::fit(vector< vector<float> > X, vector<float> y)
{
    for (int i = 0; i < X[0].size() + 1; i++) // X[0].size() + 1 -> I am using +1 to add the
bias term
    {
        m_w.push_back(0);
    }
    for (int i = 0; i < m_epochs; i++)
    {
        int errors = 0;
        for (int j = 0; j < X.size(); j++)
        {
            float update = m_eta * (y[j] - predict(X[j]));
            for (int w = 1; w < m_w.size(); w++){ m_w[w] += update * X[j][w - 1]; }
            m_w[0] = update;
            errors += update != 0 ? 1 : 0;
        }
        m_errors.push_back(errors);
    }
}

float perceptron::netInput(vector<float> X)
{
    // Sum(Vector of weights * Input vector) + bias
    float probabilities = m_w[0];
    for (int i = 0; i < X.size(); i++)
    {
        probabilities += X[i] * m_w[i + 1];
    }
    return probabilities;
}

int perceptron::predict(vector<float> X)
{
```

```
    return netInput(X) > 0 ? 1 : -1; //Step Function
}

void perceptron::printErrors()
{
    printVector(m_errors);
}

void perceptron::exportWeights(string filename)
{
    ofstream outFile;
    outFile.open(filename);

    for (int i = 0; i < m_w.size(); i++)
    {
        outFile << m_w[i] << endl;
    }

    outFile.close();
}

void perceptron::importWeights(string filename)
{
    ifstream inFile;
    inFile.open(filename);

    for (int i = 0; i < m_w.size(); i++)
    {
        inFile >> m_w[i];
    }
}

void perceptron::printWeights()
{
    cout << "weights: ";
    for (int i = 0; i < m_w.size(); i++)
    {
        cout << m_w[i] << " ";
    }
    cout << endl;
}
```

Anche se vuoi provare un esempio qui è un esempio che ho fatto:

main.cpp:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <fstream>
#include <string>
#include <math.h>

#include "MachineLearning.h"

using namespace std;
using namespace MachineLearning;

vector< vector<float> > getIrisX();
vector<float> getIrisy();
```

```
int main()
{
    vector< vector<float> > X = getIrisX();
    vector<float> y = getIrisy();
    vector<float> test1;
    test1.push_back(5.0);
    test1.push_back(3.3);
    test1.push_back(1.4);
    test1.push_back(0.2);

    vector<float> test2;
    test2.push_back(6.0);
    test2.push_back(2.2);
    test2.push_back(5.0);
    test2.push_back(1.5);
    //printVector(X);
    //for (int i = 0; i < y.size(); i++){ cout << y[i] << " "; }cout << endl;

    perceptron clf(0.1, 14);
    clf.fit(X, y);
    clf.printErrors();
    cout << "Now Predicting: 5.0,3.3,1.4,0.2(CorrectClass=-1,Iris-setosa) -> " <<
clf.predict(test1) << endl;
    cout << "Now Predicting: 6.0,2.2,5.0,1.5(CorrectClass=1,Iris-virginica) -> " <<
clf.predict(test2) << endl;

    system("PAUSE");
    return 0;
}

vector<float> getIrisy()
{
    vector<float> y;

    ifstream inFile;
    inFile.open("y.data");
    string sampleClass;
    for (int i = 0; i < 100; i++)
    {
        inFile >> sampleClass;
        if (sampleClass == "Iris-setosa")
        {
            y.push_back(-1);
        }
        else
        {
            y.push_back(1);
        }
    }

    return y;
}

vector< vector<float> > getIrisX()
{
    ifstream af;
    ifstream bf;
    ifstream cf;
    ifstream df;
    af.open("a.data");
```

```
    bf.open("b.data");
    cf.open("c.data");
    df.open("d.data");

    vector< vector<float> > X;

    for (int i = 0; i < 100; i++)
    {
        char scrap;
        int scrapN;
        af >> scrapN;
        bf >> scrapN;
        cf >> scrapN;
        df >> scrapN;

        af >> scrap;
        bf >> scrap;
        cf >> scrap;
        df >> scrap;
        float a, b, c, d;
        af >> a;
        bf >> b;
        cf >> c;
        df >> d;
        X.push_back(vector < float > {a, b, c, d});
    }

    af.close();
    bf.close();
    cf.close();
    df.close();

    return X;
}
```

Il modo in cui ho importato il set di dati dell'iride non è proprio l'ideale, ma volevo solo qualcosa che funzionasse.

I file di dati possono essere trovati qui.

Spero che tu l'abbia trovato utile!

### Qual è il pregiudizio

---

## Qual è il pregiudizio

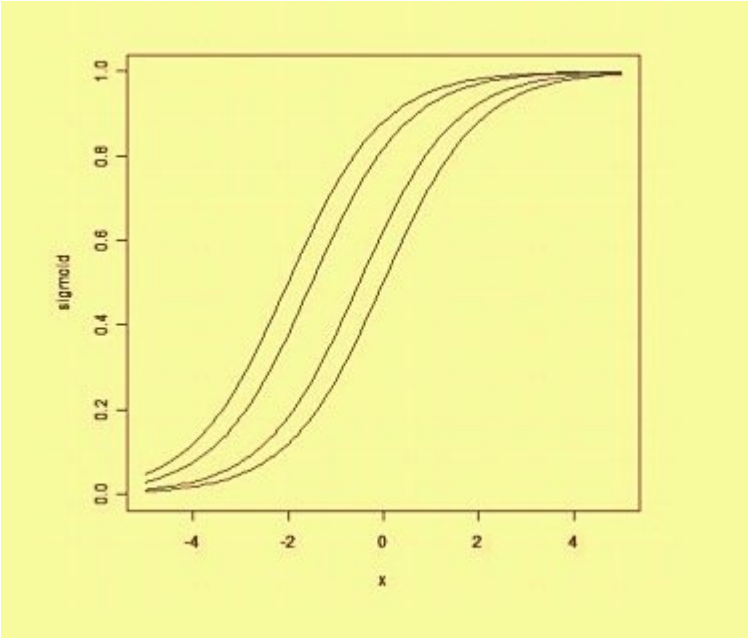
Un perceptron può essere visto come una funzione che mappa un vettore di input (valore reale) ***x*** su un valore di uscita ***f (x)*** (valore binario):

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

dove ***w*** è un vettore di pesi a valori reali e ***b*** è un nostro valore di *bias* . Il bias è un valore che

sposta il limite di decisione lontano dall'origine *(0,0)* e che non dipende da alcun valore di input.

Pensando al pregiudizio in modo spaziale, il pregiudizio altera la posizione (sebbene non l'orientamento) del confine della decisione. Possiamo vedere sotto un esempio della stessa curva spostata dal bias:




Leggi Perceptron online: https://riptutorial.com/it/machine-learning/topic/6640/perceptron

# Capitolo 10: Reti neurali

## Examples

### Per iniziare: una semplice ANN con Python

Il seguente elenco di codici tenta di classificare le cifre scritte a mano dal set di dati MNIST. Le cifre assomigliano a questo:

Il codice eseguirà il preprocesso di queste cifre, convertendo ciascuna immagine in una matrice 2D di 0 e 1 e quindi utilizzando questi dati per addestrare una rete neurale con un'accuratezza del 97% (50 epoche).

```
"""
Deep Neural Net

(Name: Classic Feedforward)

"""

import numpy as np
import pickle, json
import sklearn.datasets
import random
import time
import os


def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))

def sigmoid_prime(z):
    return sigmoid(z) * (1 - sigmoid(z))

def relU(z):
    return np.maximum(z, 0, z)

def relU_prime(z):
    return z * (z <= 0)

def tanh(z):
```

```
    return np.tanh(z)

def tanh_prime(z):
    return 1 - (tanh(z) ** 2)

def transform_target(y):
    t = np.zeros((10, 1))
    t[int(y)] = 1.0
    return t

"""-------------------------------------------------------------------------------"""

class NeuralNet:

    def __init__(self, layers, learning_rate=0.05, reg_lambda=0.01):
        self.num_layers = len(layers)
        self.layers = layers
        self.biases = [np.zeros((y, 1)) for y in layers[1:]]
        self.weights = [np.random.normal(loc=0.0, scale=0.1, size=(y, x)) for x, y in
zip(layers[:-1], layers[1:])]
        self.learning_rate = learning_rate
        self.reg_lambda = reg_lambda
        self.nonlinearity = relU
        self.nonlinearity_prime = relU_prime

    def __feedforward(self, x):
        """ Returns softmax probabilities for the output layer """
        for w, b in zip(self.weights, self.biases):
            x = self.nonlinearity(np.dot(w, np.reshape(x, (len(x), 1))) + b)

        return np.exp(x) / np.sum(np.exp(x))

    def __backpropagation(self, x, y):
        """
        :param x: input
        :param y: target
        """
        weight_gradients = [np.zeros(w.shape) for w in self.weights]
        bias_gradients = [np.zeros(b.shape) for b in self.biases]

        # forward pass
        activation = x
        hidden_activations = [np.reshape(x, (len(x), 1))]
        z_list = []

        for w, b in zip(self.weights, self.biases):
            z = np.dot(w, np.reshape(activation, (len(activation), 1))) + b
            z_list.append(z)
            activation = self.nonlinearity(z)
            hidden_activations.append(activation)

        t = hidden_activations[-1]
        hidden_activations[-1] = np.exp(t) / np.sum(np.exp(t))

        # backward pass
        delta = (hidden_activations[-1] - y) * (z_list[-1] > 0)
        weight_gradients[-1] = np.dot(delta, hidden_activations[-2].T)
        bias_gradients[-1] = delta

        for l in range(2, self.num_layers):
            z = z_list[-l]
```

```
            delta = np.dot(self.weights[-l + 1].T, delta) * (z > 0)
            weight_gradients[-l] = np.dot(delta, hidden_activations[-l - 1].T)
            bias_gradients[-l] = delta

        return (weight_gradients, bias_gradients)

    def __update_params(self, weight_gradients, bias_gradients):
        for i in xrange(len(self.weights)):
            self.weights[i] += -self.learning_rate * weight_gradients[i]
            self.biases[i] += -self.learning_rate * bias_gradients[i]

    def train(self, training_data, validation_data=None, epochs=10):
        bias_gradients = None
        for i in xrange(epochs):
            random.shuffle(training_data)
            inputs = [data[0] for data in training_data]
            targets = [data[1] for data in training_data]

            for j in xrange(len(inputs)):
                (weight_gradients, bias_gradients) = self.__backpropagation(inputs[j],
targets[j])
                self.__update_params(weight_gradients, bias_gradients)

            if validation_data:
                random.shuffle(validation_data)
                inputs = [data[0] for data in validation_data]
                targets = [data[1] for data in validation_data]

                for j in xrange(len(inputs)):
                    (weight_gradients, bias_gradients) = self.__backpropagation(inputs[j],
targets[j])
                    self.__update_params(weight_gradients, bias_gradients)

            print("{} epoch(s) done".format(i + 1))

        print("Training done.")

    def test(self, test_data):
        test_results = [(np.argmax(self.__feedforward(x[0])), np.argmax(x[1])) for x in
test_data]
        return float(sum([int(x == y) for (x, y) in test_results])) / len(test_data) * 100

    def dump(self, file):
        pickle.dump(self, open(file, "wb"))

"""-----------------------------------------------------------------------------"""

if __name__ == "__main__":
    total = 5000
    training = int(total * 0.7)
    val = int(total * 0.15)
    test = int(total * 0.15)

    mnist = sklearn.datasets.fetch_mldata('MNIST original', data_home='./data')

    data = zip(mnist.data, mnist.target)
    random.shuffle(data)
    data = data[:total]
    data = [(x[0].astype(bool).astype(int), transform_target(x[1])) for x in data]

    train_data = data[:training]
```

```
    val_data = data[training:training+val]
    test_data = data[training+val:]

    print "Data fetched"

    NN = NeuralNet([784, 32, 10]) # defining an ANN with 1 input layer (size 784 = size of the
image flattened), 1 hidden layer (size 32), and 1 output layer (size 10, unit at index i will
predict the probability of the image being digit i, where 0 <= i <= 9)

    NN.train(train_data, val_data, epochs=5)

    print "Network trained"

    print "Accuracy:", str(NN.test(test_data)) + "%"
```

Questo è un esempio di codice autonomo e può essere eseguito senza ulteriori modifiche. Assicurati di aver installato `numpy` e `scikit` per la tua versione di python.

### Backpropagation - The Heart of Neural Networks

L'obiettivo di backpropagation è ottimizzare i pesi in modo che la rete neurale possa imparare come mappare correttamente gli input e gli output arbitrari.

Ogni livello ha il proprio set di pesi e questi pesi devono essere calibrati per poter prevedere con precisione il giusto input dato in input.

Una panoramica di alto livello della propagazione del retro è la seguente:

1. Forward pass - l'input viene trasformato in un output. Ad ogni livello, l'attivazione viene calcolata con un prodotto punto tra l'input e i pesi, seguito dalla somma del risultante con il bias. Infine, questo valore viene passato attraverso una funzione di attivazione, per ottenere l'attivazione di quel livello che diventerà l'input per il livello successivo.
2. Nell'ultimo livello, l'output viene confrontato con l'etichetta effettiva corrispondente a quell'input e l'errore viene calcolato. Di solito, è l'errore quadratico medio.
3. Passaggio all'indietro: l'errore calcolato nel passaggio 2 viene propagato ai livelli interni e i pesi di tutti i livelli vengono adattati per tenere conto di questo errore.

## 1. Inizializzazione dei pesi

Di seguito è riportato un esempio semplificato di inizializzazione dei pesi:

```
layers = [784, 64, 10]
weights = np.array([(np.random.randn(y, x) * np.sqrt(2.0 / (x + y))) for x, y in zip(layers[:-
1], layers[1:])])
biases = np.array([np.zeros((y, 1)) for y in layers[1:]])
```

- Il livello nascosto 1 ha il peso della dimensione [64, 784] e il bias della dimensione 64.
- Il livello di output ha il peso della dimensione [10, 64] e il bias della dimensione

    10.

Ci si potrebbe chiedere cosa sta succedendo quando si inizializzano i pesi nel codice sopra. Questa è chiamata inizializzazione Xavier ed è un passo migliore dell'inizializzazione casuale delle matrici di peso. Sì, l'inizializzazione conta. In base alla tua inizializzazione, potresti essere in grado di trovare un minimo locale migliore durante la discesa del gradiente (la propagazione posteriore è una versione glorificata della discesa del gradiente).

## 2. Passaggio in avanti

```
activation = x
hidden_activations = [np.reshape(x, (len(x), 1))]
z_list = []

for w, b in zip(self.weights, self.biases):
    z = np.dot(w, np.reshape(activation, (len(activation), 1))) + b
    z_list.append(z)
    activation = relu(z)
    hidden_activations.append(activation)

t = hidden_activations[-1]
hidden_activations[-1] = np.exp(t) / np.sum(np.exp(t))
```

Questo codice esegue la trasformazione sopra descritta. `hidden_activations[-1]` contiene le probabilità softmax - le previsioni di tutte le classi, la cui somma è 1. Se si prevedono cifre, l'output sarà un vettore di probabilità della dimensione 10, la cui somma è 1.

## 3. Passaggio a ritroso

```
weight_gradients = [np.zeros(w.shape) for w in self.weights]
bias_gradients = [np.zeros(b.shape) for b in self.biases]

delta = (hidden_activations[-1] - y) * (z_list[-1] > 0) # relu derivative
weight_gradients[-1] = np.dot(delta, hidden_activations[-2].T)
bias_gradients[-1] = delta

for l in range(2, self.num_layers):
    z = z_list[-l]
    delta = np.dot(self.weights[-l + 1].T, delta) * (z > 0) # relu derivative
    weight_gradients[-l] = np.dot(delta, hidden_activations[-l - 1].T)
    bias_gradients[-l] = delta
```

Le prime 2 righe inizializzano i gradienti. Questi gradienti sono calcolati e verranno utilizzati per aggiornare i pesi e i bias successivi.

Le 3 linee successive calcolano l'errore sottraendo la previsione dalla destinazione. L'errore viene quindi nuovamente propagato ai livelli interni.

Ora, traccia attentamente il funzionamento del ciclo. Le righe 2 e 3 trasformano l'errore dal `layer[i]` al `layer[i - 1]` . Traccia le forme delle matrici moltiplicate per capire.

## 4. Aggiornamento pesi / parametri

```
for i in xrange(len(self.weights)):
    self.weights[i] += -self.learning_rate * weight_gradients[i]
    self.biases[i] += -self.learning_rate * bias_gradients[i]
```

`self.learning_rate` specifica la velocità con cui la rete impara. Non vuoi che impari troppo velocemente, perché potrebbe non convergere. Una discesa morbida è favorita per trovare una buona minima. Di solito, i tassi tra `0.01` e `0.1` sono considerati buoni.

## Funzioni di attivazione

Le funzioni di attivazione, note anche come funzione di trasferimento, vengono utilizzate per mappare i nodi di input per i nodi di output in un determinato modo.

Sono utilizzati per impartire non linearità all'output di uno strato di rete neurale.

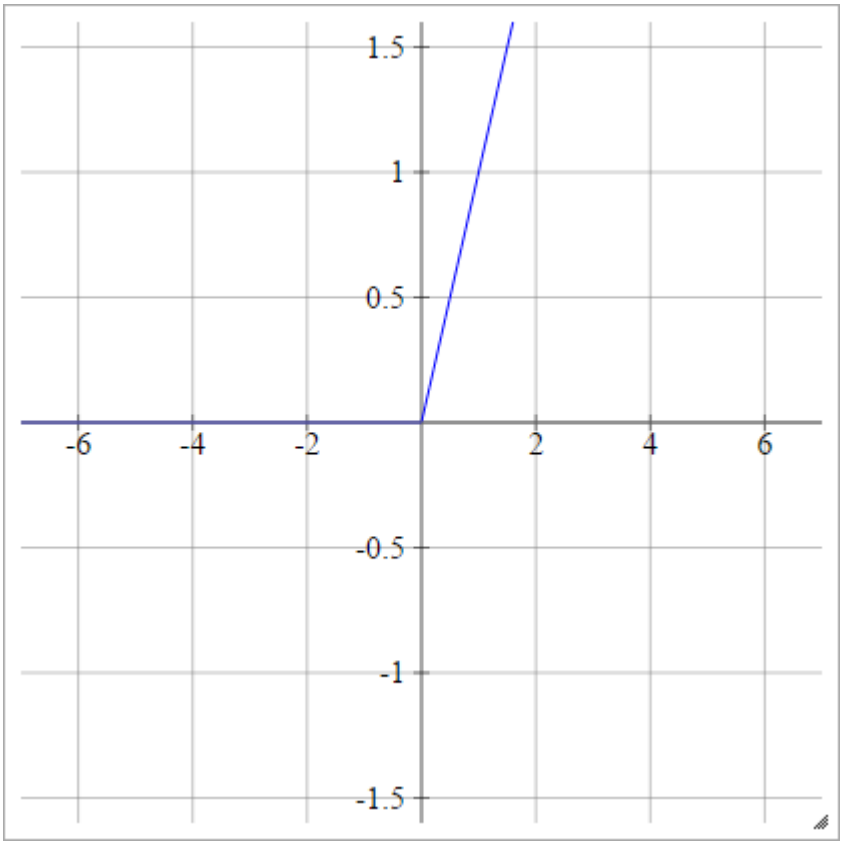
Di seguito alcune funzioni comunemente utilizzate e le loro curve:

| Activation function | Equation |
| --- | --- |
| Unit step (Heaviside) | $\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$ |
| Sign (Signum) | $\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$ |
| Linear | $\phi(z) = z$ |
| Piece-wise linear | $\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \\ 0, & z \leq -\frac{1}{2}, \end{cases}$ |
| Logistic (sigmoid) | $\phi(z) = \frac{1}{1 + e^{-z}}$ |
| Hyperbolic tangent | $\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$ |

per calcolare l'attivazione di un livello nascosto.

## Funzione ReLU

Un'unità lineare rettificata non fa altro che `max(0,x)` . È una delle scelte più comuni per le funzioni di attivazione delle unità di rete neurali.




Le ReLU risolvono il problema del gradiente di fuga delle unità tangenti sigmoide / iperboliche, consentendo una propagazione graduale efficiente nelle reti profonde.

Il nome ReLU viene dalla carta di Nair e Hinton, le *unità* rettificate rettificate *migliorano le macchine Boltzmann con restrizioni* .

Ha alcune variazioni, ad esempio, riLU (leali) (LReLU) e unità lineari esponenziali (ELU).

Il codice per l'implementazione di ReLU vanilla insieme alla sua derivata con `numpy` è mostrato di seguito:

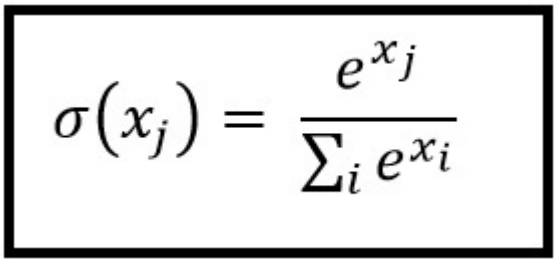
```
def relU(z):
    return z * (z > 0)

def relU_prime(z):
    return z > 0
```

## Funzione Softmax

La regressione di Softmax (o regressione logistica multinomiale) è una generalizzazione della regressione logistica nel caso in cui vogliamo gestire più classi. È particolarmente utile per le reti

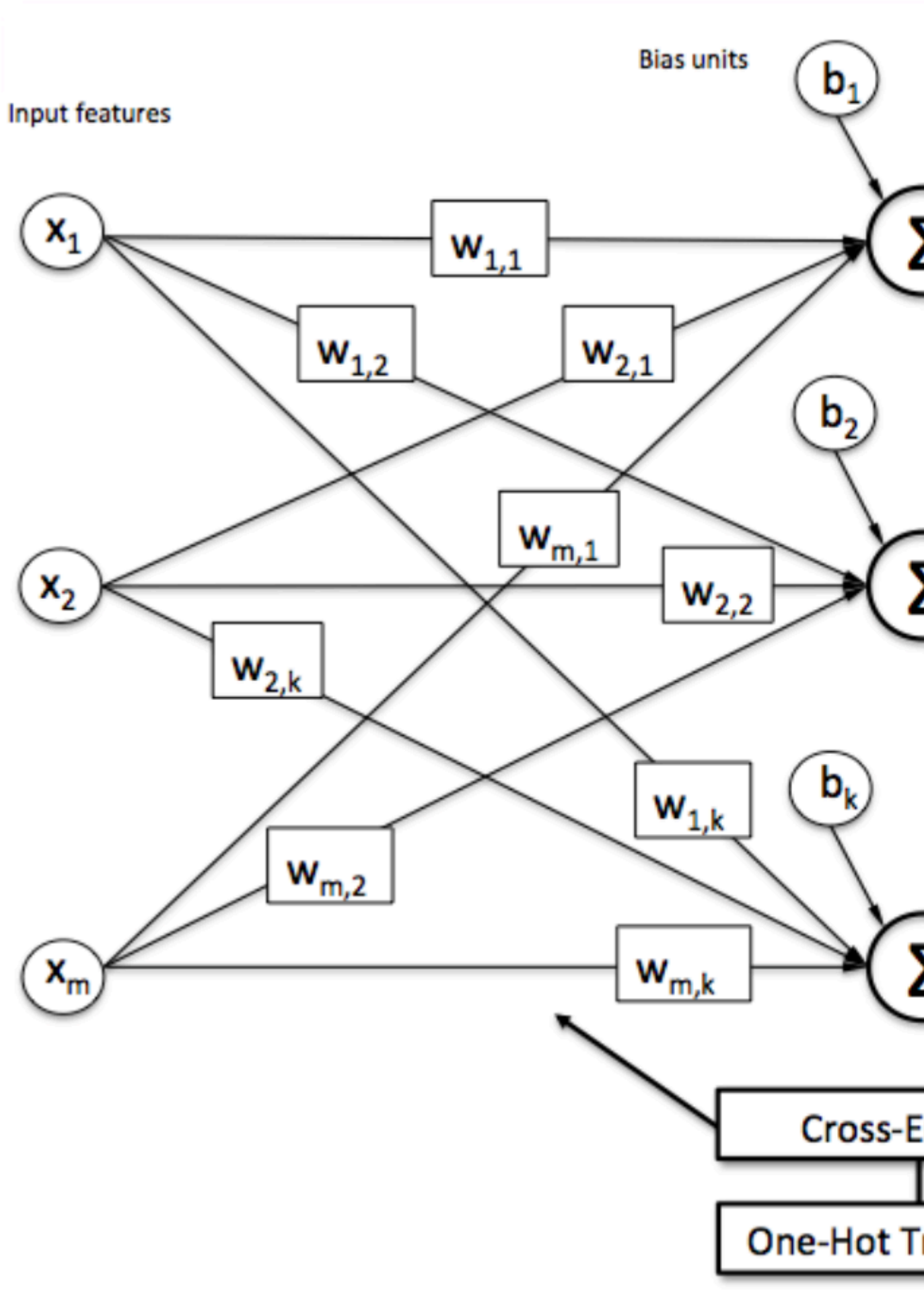
neurali in cui vogliamo applicare la classificazione non binaria. In questo caso, la semplice regressione logistica non è sufficiente. Avremmo bisogno di una distribuzione di probabilità su tutte le etichette, che è ciò che ci offre Softmax.

Softmax è calcolato con la seguente formula:




### _____Dove si adatta?

_____


Bias units
Input features
b1
x1
w1,1
w1,2
w2,1
b2
wm,1
x2
w2,2
w2,k
w1,k
bk
wm,2
xm
wm,k
Cross-E
One-Hot T

Softmax Re

Per normalizzare un vettore applicando la funzione softmax ad esso con `numpy` , utilizzare:

```
np.exp(x) / np.sum(np.exp(x))
```

Dove `x` è l'attivazione dallo strato finale della RNA.

Leggi Reti neurali online: https://riptutorial.com/it/machine-learning/topic/2709/reti-neurali

# Capitolo 11: Scikit impara

## Examples

### Un semplice problema di classificazione semplice (XOR) utilizzando l'algoritmo k neighbor neighbor

Considera che vuoi pronosticare la risposta corretta per il problema popolare XOR. Sapevi che cos'è XOR (es. [X0 x1] => y). ad esempio [0 0] => 0, [0 1] => [1] e ...

```
#Load Sickit learn data
from sklearn.neighbors import KNeighborsClassifier

#X is feature vectors, and y is correct label(To train model)
X = [[0, 0],[0 ,1],[1, 0],[1, 1]]
y = [0,1,1,0]

#Initialize a Kneighbors Classifier with K parameter set to 2
KNC = KNeighborsClassifier(n_neighbors= 2)

#Fit the model(the KNC learn y Given X)
KNC.fit(X, y)

#print the predicted result for [1 1]
print(KNC.predict([[1 1]]))
```

### Classificazione in scikit-learn

#### 1. Alberi decisionali insaccati

L'insaccamento si comporta meglio con algoritmi con varianza elevata. Un esempio popolare sono gli alberi decisionali, spesso costruiti senza potatura.

Nell'esempio seguente si vede un esempio di utilizzo del BaggingClassifier con l'algoritmo Classification and Regression Trees (DecisionTeamClassifier). Vengono creati un totale di 100 alberi.

Set di dati utilizzato: set di dati sul diabete Pima Indians

```
# Bagged Decision Trees for Classification
import pandas
from sklearn import cross_validation
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-
indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
```

```
num_folds = 10
num_instances = len(X)
seed = 7
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 100
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=seed)
results = cross_validation.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

Eseguendo l'esempio, otteniamo una stima attendibile dell'accuratezza del modello.

```
0.770745044429
```

**2. Foresta casuale**

La foresta casuale è un'estensione degli alberi decisionali insaccati.

I campioni dell'insieme di dati sull'addestramento vengono presi con la sostituzione, ma gli alberi sono costruiti in modo da ridurre la correlazione tra i singoli classificatori. In particolare, anziché scegliere avidamente il miglior punto di divisione nella costruzione dell'albero, solo un sottoinsieme casuale di funzionalità viene preso in considerazione per ogni suddivisione.

È possibile costruire un modello Foresta casuale per la classificazione utilizzando la classe RandomForestClassifier.

L'esempio seguente fornisce un esempio di Foresta casuale per la classificazione con 100 alberi e punti di divisione scelti da una selezione casuale di 3 caratteristiche.

```
# Random Forest Classification
import pandas
from sklearn import cross_validation
from sklearn.ensemble import RandomForestClassifier
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-
indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_folds = 10
num_instances = len(X)
seed = 7
num_trees = 100
max_features = 3
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_state=seed)
model = RandomForestClassifier(n_estimators=num_trees, max_features=max_features)
results = cross_validation.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

L'esecuzione dell'esempio fornisce una stima media dell'accuratezza della classificazione.

```
0.770727956254
```

### 3. AdaBoost

AdaBoost è stato forse il primo algoritmo di ensemble boosting di successo. Funziona generalmente ponderando le istanze nel set di dati in base alla loro facilità o difficoltà di classificazione, consentendo all'algoritmo di pagare o meno attenzione a loro nella costruzione dei modelli successivi.

È possibile costruire un modello AdaBoost per la classificazione utilizzando la classe AdaBoostClassifier.

L'esempio seguente mostra la costruzione di 30 alberi decisionali in sequenza usando l'algoritmo AdaBoost.

```
# AdaBoost Classification
import pandas
from sklearn import cross_validation
from sklearn.ensemble import AdaBoostClassifier
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-
indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_folds = 10
num_instances = len(X)
seed = 7
num_trees = 30
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_state=seed)
model = AdaBoostClassifier(n_estimators=num_trees, random_state=seed)
results = cross_validation.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

L'esecuzione dell'esempio fornisce una stima media dell'accuratezza della classificazione.

```
0.76045796309
```

### 4. Aumento graduale stocastico

Stochastic Gradient Boosting (chiamato anche Gradient Boosting Machines) è una delle tecniche di ensemble più sofisticate. È anche una tecnica che sta dimostrando di essere forse delle migliori tecniche disponibili per migliorare le prestazioni tramite ensemble.

È possibile costruire un modello Gradient Boosting per la classificazione utilizzando la classe GradientBoostingClassifier.

L'esempio seguente mostra l'aumento graduale stocastico per la classificazione con 100 alberi.

```
# Stochastic Gradient Boosting Classification
import pandas
from sklearn import cross_validation
from sklearn.ensemble import GradientBoostingClassifier
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-
```

```
indians-diabetes.data"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_folds = 10
num_instances = len(X)
seed = 7
num_trees = 100
kfold = cross_validation.KFold(n=num_instances, n_folds=num_folds, random_state=seed)
model = GradientBoostingClassifier(n_estimators=num_trees, random_state=seed)
results = cross_validation.cross_val_score(model, X, Y, cv=kfold)
print(results.mean())
```

L'esecuzione dell'esempio fornisce una stima media dell'accuratezza della classificazione.

```
0.764285714286
```

Fonte: http://machinelearningmastery.com/ensemble-machine-learning-algorithms-python-scikit-learn/

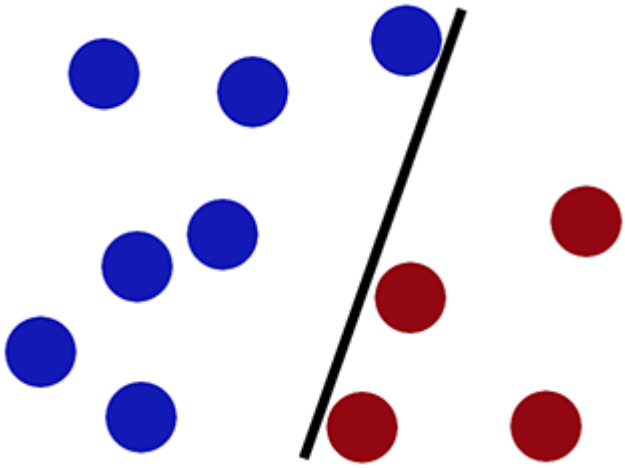
Leggi Scikit impara online: https://riptutorial.com/it/machine-learning/topic/6156/scikit-impara

# Capitolo 12: SVM

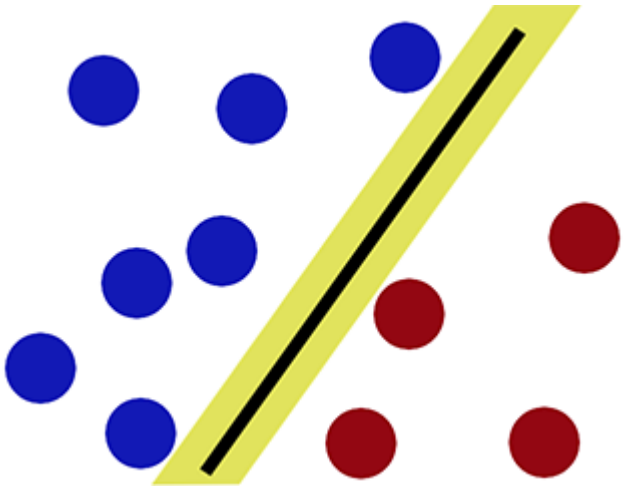
## Examples

### Differenza tra regressione logistica e SVM

Confini decisionali quando classifichiamo usando **la regressione logistica** -



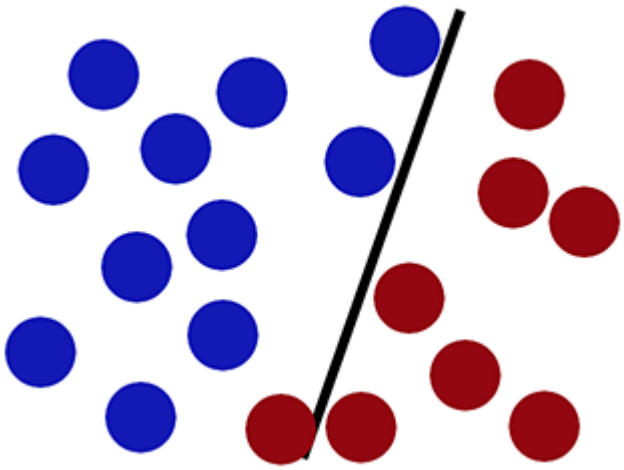
Confini decisionali quando classifichiamo usando **SVM** -



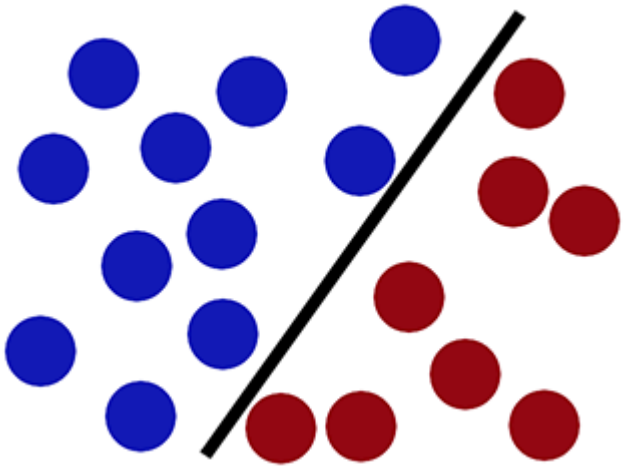
Come si può osservare, SVM cerca di mantenere una "distanza" su entrambi i lati del confine della decisione. Ciò si rivela utile quando incontriamo nuovi dati.

**Con i nuovi dati**

La regressione logistica si comporta **male** (il nuovo cerchio rosso è classificato come blu) -



Mentre **SVM** può classificarlo correttamente (il nuovo cerchio rosso è classificato correttamente nel lato rosso) -



### Implementare il classificatore SVM usando Scikit-learn:

```
from sklearn import svm
X = [[1, 2], [3, 4]] #Training Samples
y = [1, 2] #Class labels
model = svm.SVC() #Making a support vector classifier model
model.fit(X, y) #Fitting the data

clf.predict([[2, 3]]) #After fitting, new data can be classified by using predict()
```

Leggi SVM online: https://riptutorial.com/it/machine-learning/topic/7126/svm

# Capitolo 13: Tipi di apprendimento

## Examples

### Apprendimento supervisionato

La macchina impara a prevedere un output quando viene fornito un input.

Ogni caso di formazione consiste in un input e un output target.

# Regressione

L'uscita obiettivo assume valori continui.

- Prevedere il prezzo di un titolo
- Prevedere un prezzo di casa

# Classificazione

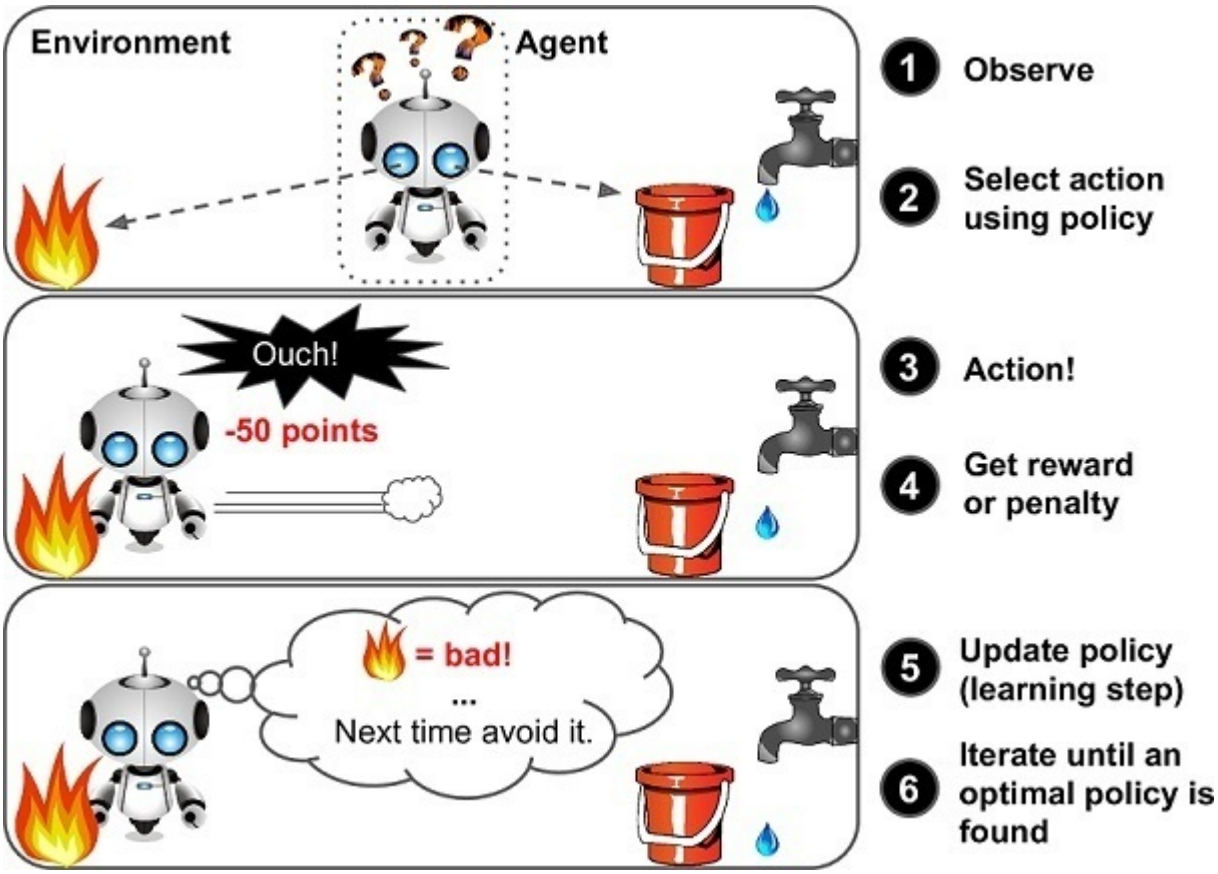
L'output di destinazione è un'etichetta di classe.

- Che tipo di frutto è l'input
- Che lingua è una parola

### Insegnamento rafforzativo

La macchina deve determinare automaticamente il comportamento ideale per massimizzare le sue prestazioni.

Per esempio:




Utilizzando l'apprendimento di rinforzo puoi anche creare un programma per computer che possa completare un livello Mario ( MarI / O - Machine Learning per videogiochi ).

## Apprendimento senza supervisione

L'apprendimento senza supervisione ci permette di affrontare i problemi con poca o nessuna idea di come dovrebbero essere i nostri risultati. Possiamo **ricavare la struttura da dati in** cui non conosciamo necessariamente l'effetto delle variabili.

Il tipo più comune di apprendimento non supervisionato è l' **analisi cluster o il clustering** . È il compito di raggruppare un insieme di oggetti in modo tale che gli oggetti nello stesso gruppo (cluster) siano più simili tra loro che con quelli di altri gruppi.

Esiste anche l'apprendimento non controllato in clustering. Un esempio di ciò è l'identificazione di singole voci e musica da una trama di suoni. Questo è chiamato "Cocktail Party Algorithm".

Leggi Tipi di apprendimento online: https://riptutorial.com/it/machine-learning/topic/10912/tipi-di-apprendimento

# Capitolo 14: Un'introduzione alla classificazione: generare diversi modelli usando Weka

## introduzione

Questo tutorial ti mostrerà come usare Weka nel codice JAVA, caricare il file di dati, formare i classificatori e spiegare alcuni concetti importanti alla base dell'apprendimento automatico.

Weka è un kit di strumenti per l'apprendimento automatico. Include una libreria di tecniche di machine learning e di visualizzazione e presenta una GUI user friendly.

Questo tutorial include esempi scritti in JAVA e include immagini generate con la GUI. Suggerisco di utilizzare la GUI per esaminare i dati e il codice JAVA per esperimenti strutturati.

## Examples

### Per iniziare: caricamento di un set di dati dal file

Il set di dati del fiore di Iris è un set di dati ampiamente utilizzato a scopo dimostrativo. Lo caricheremo, lo ispezioneremo e lo modificheremo leggermente per un uso successivo.

```
import java.io.File;
import java.net.URL;
import weka.core.Instances;
import weka.core.converters.ArffSaver;
import weka.core.converters.CSVLoader;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.RenameAttribute;
import weka.classifiers.evaluation.Evaluation;
import weka.classifiers.rules.ZeroR;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.lazy.IBk;
import weka.classifiers.trees.J48;
import weka.classifiers.meta.AdaBoostM1;

public class IrisExperiments {
    public static void main(String args[]) throws Exception
     {
        //First we open stream to a data set as provided on http://archive.ics.uci.edu
         CSVLoader loader = new CSVLoader();
         loader.setSource(new URL("http://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data").openStream());
         Instances data = loader.getDataSet();

         //This file has 149 examples with 5 attributes
         //In order:
         //  sepal length in cm
         //  sepal width in cm
```

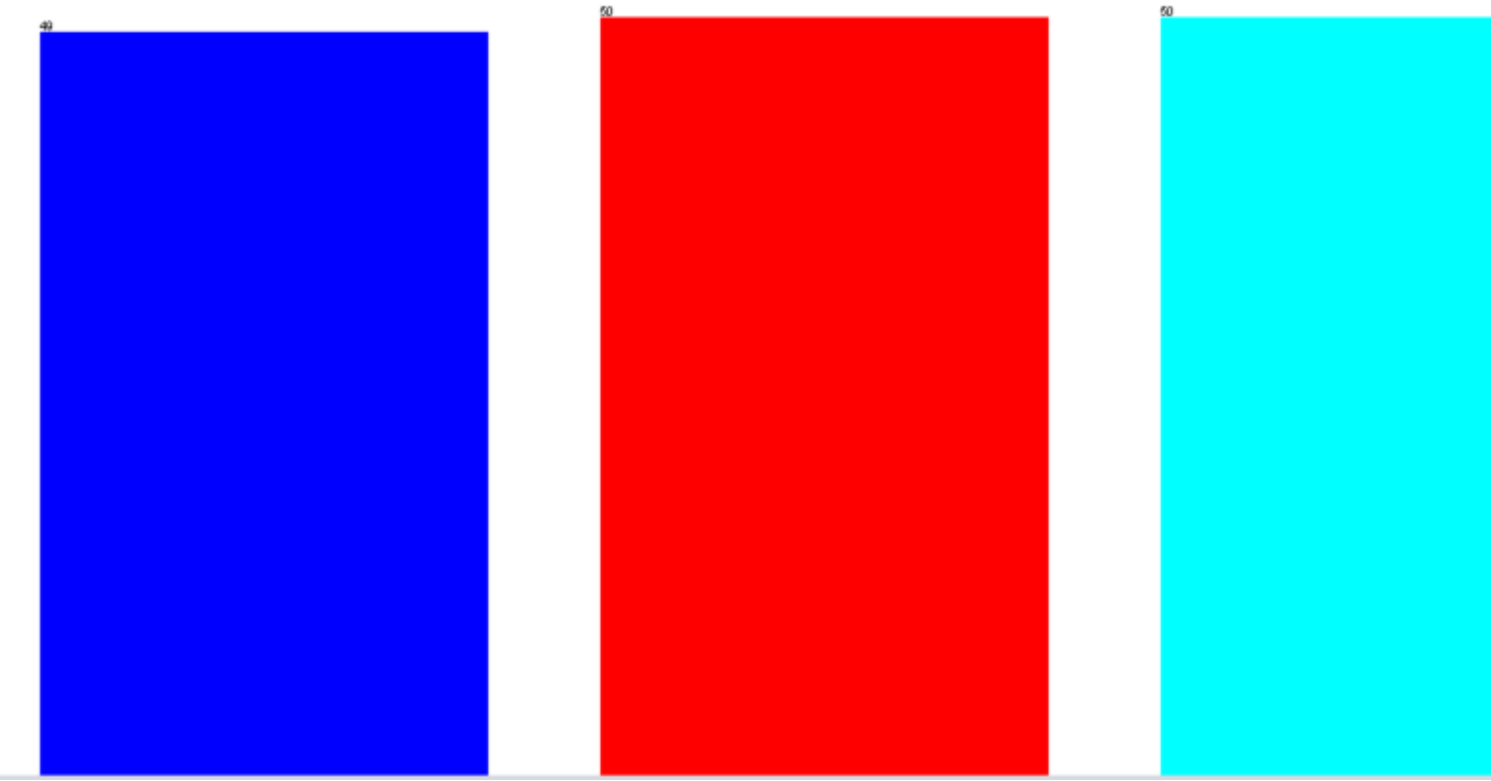
```
        //  petal length in cm
        //  petal width in cm
        //  class ( Iris Setosa , Iris Versicolour, Iris Virginica)

        //Let's briefly inspect the data
        System.out.println("This file has " + data.numInstances()+" examples.");
        System.out.println("The first example looks like this: ");
        for(int i = 0; i < data.instance(0).numAttributes();i++ ){
            System.out.println(data.instance(0).attribute(i));
        }

        // NOTE that the last attribute is Nominal
        // It is convention to have a nominal variable at the last index as target variable

        // Let's tidy up the data a little bit
        // Nothing too serious just to show how we can manipulate the data with filters
        RenameAttribute renamer = new RenameAttribute();
        renamer.setOptions(weka.core.Utils.splitOptions("-R last -replace Iris-type"));
        renamer.setInputFormat(data);
        data = Filter.useFilter(data, renamer);

        System.out.println("We changed the name of the target class.");
        System.out.println("And now it looks like this:");
        System.out.println(data.instance(0).attribute(4));

        //Now we do this for all the attributes
        renamer.setOptions(weka.core.Utils.splitOptions("-R 1 -replace sepal-length"));
        renamer.setInputFormat(data);
        data = Filter.useFilter(data, renamer);

        renamer.setOptions(weka.core.Utils.splitOptions("-R 2 -replace sepal-width"));
        renamer.setInputFormat(data);
        data = Filter.useFilter(data, renamer);

        renamer.setOptions(weka.core.Utils.splitOptions("-R 3 -replace petal-length"));
        renamer.setInputFormat(data);
        data = Filter.useFilter(data, renamer);

        renamer.setOptions(weka.core.Utils.splitOptions("-R 4 -replace petal-width"));
        renamer.setInputFormat(data);
        data = Filter.useFilter(data, renamer);

        //Lastly we save our newly created file to disk
        ArffSaver saver = new ArffSaver();
        saver.setInstances(data);
        saver.setFile(new File("IrisSet.arff"));
        saver.writeBatch();
    }
}
```

## Allena il primo classificatore: imposta una linea di base con ZeroR

ZeroR è un semplice classificatore. Non funziona per istanza, invece funziona sulla distribuzione generale delle classi. Seleziona la classe con la più grande probabilità a priori. Non è un buon classificatore nel senso che non usa alcuna informazione nel candidato, ma è spesso usato come base di riferimento. **Nota: è possibile utilizzare altre linee di base come: classificatori standard di settore o regole artigianali**

```
// First we tell our data that it's class is hidden in the last attribute
data.setClassIndex(data.numAttributes() -1);
// Then we split the data in to two sets
// randomize first because we don't want unequal distributions
data.randomize(new java.util.Random(0));
Instances testset = new Instances(data, 0, 50);
Instances trainset = new Instances(data, 50, 99);

// Now we build a classifier
// Train it with the trainset
ZeroR classifier1 = new ZeroR();
classifier1.buildClassifier(trainset);
// Next we test it against the testset
Evaluation Test = new Evaluation(trainset);
Test.evaluateModel(classifier1, testset);
System.out.println(Test.toSummaryString());
```

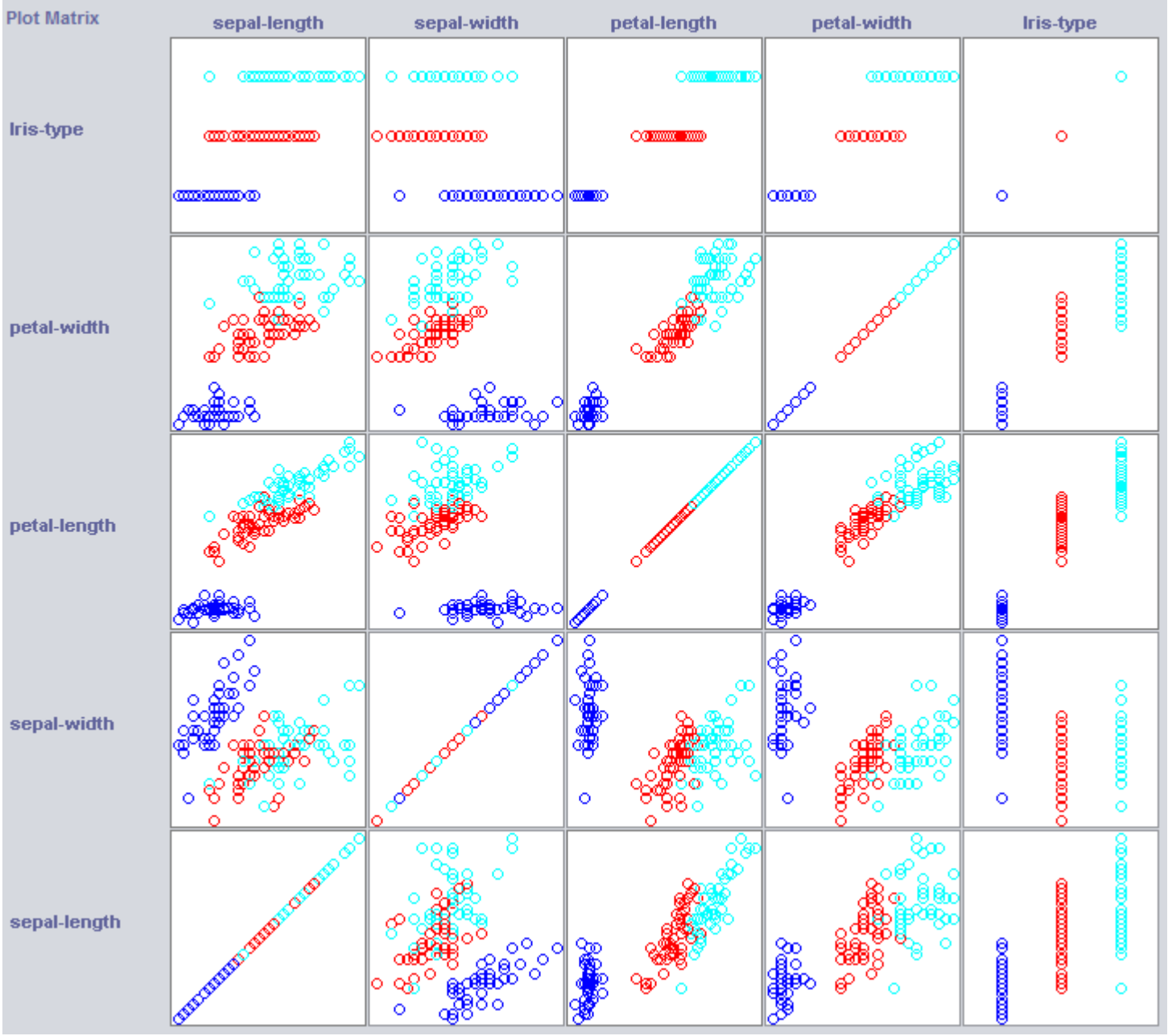
La più grande classe nel set ti dà un tasso corretto del 34%. (50 su 149)



**Nota: lo ZeroR si attesta attorno al 30%. Questo perché abbiamo suddiviso casualmente un treno e un set di prova. Il set più grande nel set di treni, sarà quindi il più piccolo nel set di prova. Preparare un buon test / set di treni può valerne la pena**

## Ottenere una sensazione per i dati. Allenamento Naive Bayes e kNN

Per costruire un buon classificatore, spesso abbiamo bisogno di avere un'idea di come i dati sono strutturati nello spazio delle feature. Weka offre un modulo di visualizzazione che può aiutare.




Alcune dimensioni già separano abbastanza bene le classi. La larghezza del petalo ordina il concetto in modo abbastanza preciso, se confrontato con la larghezza del petalo, ad esempio.

La formazione di semplici classificatori può rivelare un bel po 'anche sulla struttura dei dati. Solitamente mi piace usare il vicino più vicino e Naive Bayes per quello scopo. Naive Bayes assume l'indipendenza, il suo rendimento è indicativo del fatto che le dimensioni su se stesse contengono informazioni. k-Nearest-Neighbor funziona assegnando la classe delle istanze k più vicine (note) nello spazio delle feature. È spesso usato per esaminare la dipendenza geografica locale, lo useremo per esaminare se il nostro concetto è definito localmente nello spazio delle funzioni.

```
//Now we build a Naive Bayes classifier
NaiveBayes classifier2 = new NaiveBayes();
classifier2.buildClassifier(trainset);
// Next we test it against the testset
Test = new Evaluation(trainset);
Test.evaluateModel(classifier2, testset);
```

```
 System.out.println(Test.toSummaryString());

//Now we build a kNN classifier
IBk classifier3 = new IBk();
// We tell the classifier to use the first nearest neighbor as example
classifier3.setOptions(weka.core.Utils.splitOptions("-K 1"));
classifier3.buildClassifier(trainset);
// Next we test it against the testset
Test = new Evaluation(trainset);
Test.evaluateModel(classifier3, testset);
System.out.println(Test.toSummaryString());
```

Naive Bayes si comporta molto meglio della nostra linea di base appena stabilita, indicando che le caratteristiche indipendenti contengono informazioni (ricorda la larghezza del petalo?).

Anche 1NN si comporta bene (in effetti un po 'meglio in questo caso), indicando che alcune delle nostre informazioni sono locali. Le migliori prestazioni potrebbero indicare che alcuni effetti del secondo ordine contengono anche informazioni *(se x e y rispetto alla classe z)* .

## Metterlo insieme: formare un albero

Gli alberi possono costruire modelli che funzionano su funzioni indipendenti e su effetti di second'ordine. Quindi potrebbero essere buoni candidati per questo dominio. Gli alberi sono regole che sono chaind insieme, una regola divide le istanze che arrivano a una regola in sottogruppi, che passano alle regole sotto la regola.

Gli Tree Learners generano regole, li incatenano e smettono di costruire alberi quando sentono che le regole diventano troppo specifiche, per evitare il sovradattamento. *Il sovradimensionamento significa costruire un modello troppo complesso per il concetto che stiamo cercando. I modelli con sovralimentazione funzionano bene sui dati del treno, ma scarsamente sui nuovi dati*

Usiamo J48, un'implementazione JAVA di C4.5 un algoritmo popolare.

```
//We train a tree using J48
//J48 is a JAVA implementation of the C4.5 algorithm
J48 classifier4 = new J48();
//We set it's confidence level to 0.1
//The confidence level tell J48 how specific a rule can be before it gets pruned
classifier4.setOptions(weka.core.Utils.splitOptions("-C 0.1"));
classifier4.buildClassifier(trainset);
// Next we test it against the testset
Test = new Evaluation(trainset);
Test.evaluateModel(classifier4, testset);
System.out.println(Test.toSummaryString());

System.out.print(classifier4.toString());

//We set it's confidence level to 0.5
//Allowing the tree to maintain more complex rules
classifier4.setOptions(weka.core.Utils.splitOptions("-C 0.5"));
classifier4.buildClassifier(trainset);
// Next we test it against the testset
Test = new Evaluation(trainset);
Test.evaluateModel(classifier4, testset);
System.out.println(Test.toSummaryString());
```

```
System.out.print(classifier4.toString());
```

Lo studente dell'albero addestrato con la massima sicurezza genera le regole più specifiche e ha le migliori prestazioni sul set di prova, apparentemente la specificità è giustificata.

```
J48 pruned tree
------------------

petal-width <= 0.6: Iris-setosa (34.0)
petal-width > 0.6
|   petal-length <= 4.7: Iris-versicolor (27.0/1.0)
|   petal-length > 4.7: Iris-virginica (38.0/4.0)

J48 pruned tree
------------------

petal-width <= 0.6: Iris-setosa (34.0)
petal-width > 0.6
|   petal-length <= 5
|   |   sepal-width <= 3
|   |   |   petal-width <= 1.7: Iris-versicolor (28.0/2.0)
|   |   |   petal-width > 1.7: Iris-virginica (6.0)
|   |   sepal-width > 3: Iris-versicolor (4.0)
|   petal-length > 5: Iris-virginica (27.0)
```

**Nota: entrambi gli studenti iniziano con una regola sulla larghezza del petalo. Ricorda come abbiamo notato questa dimensione nella visualizzazione?**

Leggi Un'introduzione alla classificazione: generare diversi modelli usando Weka online: https://riptutorial.com/it/machine-learning/topic/8649/un-introduzione-alla-classificazione--generare-diversi-modelli-usando-weka

# Titoli di coda

| S. No | Capitoli | Contributors |
|---|---|---|
| 1 | Iniziare con l'apprendimento automatico | Aquib Javed Khan, Community, Dr. Cool, Drew, John Syrinek, Nikos Tavoularis, Piyush, Semih Korkmaz |
| 2 | Apprendimento approfondito | Martin W, Minhas Kamal, orbit, Thomas Pinetz |
| 3 | Apprendimento automatico e classificazione | Sirajus Salayhin |
| 4 | Apprendimento automatico tramite Java | Arnab Biswas, Patel Sunil |
| 5 | Apprendimento supervisionato | draco_alpine, L.V.Rao, Martin W, Masoud, plumSemPy, quintumnia, shadowfox |
| 6 | Elaborazione del linguaggio naturale | quintumnia |
| 7 | Iniziare con Machine Learning utilizzando Apache spark MLib | Malav |
| 8 | Metriche di valutazione | DJanssens, Franck Dernoncourt, Sayali Sonawane, ScientiaEtVeritas |
| 9 | Perceptron | granmirupa, Martin W, Panos |
| 10 | Reti neurali | cLDSEED, dontloo, Dr. Cool, Franck Dernoncourt |
| 11 | Scikit impara | Masoud, RamenChef, Sayali Sonawane |
| 12 | SVM | Alekh Karkada Ashok |
| 13 | Tipi di apprendimento | Martin W |
| 14 | Un'introduzione alla classificazione: generare diversi modelli usando | S van Balen |

| | |
|---|---|
| | Weka |