eBook Gratuit

APPRENEZ machine-learning

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#machine-learning

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: machine-learning

It is an unofficial and free machine-learning ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official machine-learning.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec l'apprentissage automatique

Remarques

L'apprentissage automatique est la science (et l'art) de la programmation des ordinateurs afin qu'ils puissent apprendre à partir de données.

Une définition plus formelle:

C'est le domaine d'étude qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmés. Arthur Samuel, 1959

Une définition plus orientée vers l'ingénierie:

Un programme informatique est censé tirer des enseignements de l'expérience E en ce qui concerne certaines tâches T et certaines mesures de performance P, si ses performances sur T, mesurées par P, augmentent avec l'expérience. E. Tom Mitchell, 1997

Source: «Apprentissage automatique avec Scikit-Learn et TensorFlow d'Aurélien Géron (O'Reilly). Copyright 2017 Aurélien Géron, 978-1-491-96229-9.

Le machine learning (ML) est un domaine de l'informatique issu de la recherche en intelligence artificielle. La force de l'apprentissage automatique par rapport aux autres formes d'analyse réside dans sa capacité à découvrir des informations cachées et à prédire les résultats d'intrants futurs résultats (généralisation). Contrairement aux algorithmes itératifs où les opérations sont explicitement déclarées, les algorithmes d'apprentissage automatique empruntent des concepts de la théorie des probabilités pour sélectionner, évaluer et améliorer des modèles statistiques.

Examples

Installation ou configuration à l'aide de Python

1) scikit apprendre

scikit-learn est un module Python pour l'apprentissage automatique basé sur SciPy et distribué sous la licence BSD à 3 clauses. Il propose divers algorithmes de classification, de régression et de clustering, notamment des machines à vecteurs de support, des forêts aléatoires, l'amplification du gradient, k-means et DBSCAN, et est conçu pour interagir avec les bibliothèques numériques et scientifiques Python

La version stable actuelle de scikit-learn nécessite :

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

- SciPy (> = 0.9).

Pour la plupart l' installation `pip` gestionnaire de paquets python peut installer python et toutes ses dépendances:

```
pip install scikit-learn
```

Cependant, pour les systèmes Linux, il est recommandé d'utiliser le `conda` paquets `conda` pour éviter les processus de génération possibles.

```
conda install scikit-learn
```

Pour vérifier que vous avez `scikit-learn` , exécutez en shell:

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

Installation de Windows et Mac OSX:

Canopy et Anaconda proposent tous deux une version récente de *scikit-learn* , en plus d'un grand ensemble de bibliothèques scientifiques Python pour Windows, Mac OSX (également pertinentes pour Linux).

Repo officiel du code source: https://github.com/scikit-learn/scikit-learn

---

2) Plate-forme Numenta pour l'informatique intelligente

La plate-forme Numenta pour l'informatique intelligente (NuPIC) est une plateforme d'intelligence machine qui implémente les algorithmes d'apprentissage HTM. HTM est une théorie computationnelle détaillée du néocortex. Au cœur de HTM se trouvent des algorithmes d'apprentissage continu basés sur le temps qui stockent et rappellent les modèles spatiaux et temporels. NuPIC est adapté à une variété de problèmes, en particulier la détection des anomalies et la prévision des sources de données en continu.

Les binaires NuPIC sont disponibles pour:

Linux x86 64 bits
OS X 10.9
OS X 10.10
Windows 64 bits

Les dépendances suivantes sont requises pour installer NuPIC sur tous les systèmes d'exploitation.

- Python 2.7
- pip> = 8.1.2
- setuptools> = 25.2.0
- roue> = 0.29.0
- insipide

- Compilateur C ++ 11 comme gcc (4.8+) ou clang

Configuration OS X supplémentaire:

- Outils de ligne de commande Xcode

Exécutez les opérations suivantes pour installer NuPIC:

```
pip install nupic
```

Repo officiel du code source: https://github.com/numenta/nupic

---

3) nilearn

Nilearn est un module Python pour un apprentissage statistique rapide et facile sur les données NeuroImaging. Il s'appuie sur la boîte à outils Python scikit-learn pour les statistiques multivariées avec des applications telles que la modélisation prédictive, la classification, le décodage ou l'analyse de connectivité.

Les dépendances requises pour utiliser le logiciel sont les suivantes:

- Python> = 2.6,
- setuptools
- Numpy> = 1.6.1
- SciPy> = 0.9
- Scikit-learn> = 0.14.1
- Nibabel> = 1.1.0

Si vous utilisez des fonctionnalités de traçage nilearn ou exécutez les exemples, matplotlib> = 1.1.1 est requis.

Si vous voulez exécuter les tests, vous avez besoin de nez> = 1.2.1 et de couverture> = 3.6.

Assurez-vous d'avoir installé toutes les dépendances énumérées ci-dessus. Ensuite, vous pouvez installer nilearn en exécutant la commande suivante dans une invite de commande:

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

Repo officiel du code source: https://github.com/nilearn/nilearn/

4) Utiliser Anaconda

De nombreuses bibliothèques Python scientifiques sont facilement disponibles dans Anaconda. Vous pouvez obtenir des fichiers d'installation à partir d' ici . D'une part, en utilisant Anaconda, vous ne devez pas installer et configurer de nombreux paquets, c'est une licence BSD, et vous disposez d'un processus d'installation trivial, disponible pour Python 3 et Python 2, tandis que À titre d'exemple, certains paquets python d'apprentissage en profondeur pourraient utiliser une version différente de numpy, puis Anaconda. Cependant, cet inconvénient peut être résolu en

utilisant une autre installation Python séparément (sous Linux et MAC, votre installation par défaut, par exemple).

Le programme d'installation d'Anaconda vous invite à sélectionner l'emplacement d'installation et vous invite également à ajouter l'option PATH. Si vous ajoutez Anaconda à votre PATH, votre système d'exploitation devrait trouver Anaconda Python par défaut. Par conséquent, les modifications et les futures installations seront uniquement disponibles pour cette version de Python.

Pour que ce soit clair, après l'installation d'Anaconda et que vous l'ajoutez à PATH, utilisez Ubuntu 14.04 via le terminal si vous tapez

```
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 est votre Python par défaut, vous pouvez commencer à profiter de nombreuses bibliothèques immédiatement. Cependant, si vous souhaitez utiliser votre ancien 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.
>>>
```

En bref, Anaconda est l'un des moyens les plus rapides de démarrer l'apprentissage automatique et l'analyse des données avec Python.

Installation ou configuration à l'aide du langage R

Les packages sont des collections de fonctions R, de données et de code compilé dans un format bien défini. Les référentiels publics (et privés) sont utilisés pour héberger des collections de packages R. La plus grande collection de packages R est disponible auprès de CRAN. Parmi les paquets d'apprentissage automatique R les plus populaires, on peut citer les suivants:

1) rpart

Description: Partitionnement récursif pour les arbres de classification, de régression et de survie. Une mise en œuvre de la plupart des fonctionnalités du livre de 1984 de Breiman, Friedman, Olshen et Stone.

Il peut être installé à partir de CRAN en utilisant le code suivant:

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

Chargez le paquet:

```
library(rpart)
```

Source officielle: https://cran.r-project.org/web/packages/rpart/index.html

---

2) e1071

Description: Fonctions pour l'analyse des classes latentes, la transformée de Fourier courte, la mise en grappe floue, les machines à vecteurs de support, le calcul du plus court chemin, la mise en grappes ensachée, le classificateur Bayes naïf, etc.

Installation de CRAN:

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

Chargement du paquet:

```
library(e1071)
```

Source officielle: https://cran.r-project.org/web/packages/e1071/index.html

---

3) randomForest

Description: Classification et régression basée sur une forêt d'arbres utilisant des entrées aléatoires.

Installation de CRAN:

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

Chargement du paquet:

```
library(randomForest)
```

Source officielle: https://cran.r-project.org/web/packages/randomForest/index.html

---

4) caret

Description: Fonctions diverses pour l'entraînement et le traçage des modèles de classification et de régression.

Installation de CRAN:

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

Chargement du paquet:

```
library(caret)
```

Source officielle: https://cran.r-project.org/web/packages/caret/index.html

Lire Démarrer avec l'apprentissage automatique en ligne: https://riptutorial.com/fr/machine-learning/topic/1151/demarrer-avec-l-apprentissage-automatique

Chapitre 2: Apprentissage automatique avec Java

Examples

liste d'outils

```
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.
```

Apprentissage par machine à usage général

```
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.
```

Reconnaissance de la parole

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

Analyse de données / visualisation de données

```
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
```

L'apprentissage en profondeur

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

Lire Apprentissage automatique avec Java en ligne: https://riptutorial.com/fr/machine-learning/topic/6175/apprentissage-automatique-avec-java

Chapitre 3: Démarrer avec Machine Learning en utilisant Apache spark MLib

Introduction

Apache spark MLib fournit (JAVA, R, PYTHON, SCALA) 1.) Divers algorithmes d'apprentissage automatique sur la régression, la classification, le clustering, le filtrage collaboratif qui sont principalement utilisés dans l'apprentissage automatique. 2.) Il prend en charge l'extraction de fonctionnalités, la transformation, etc. 3.) Il permet aux professionnels de la gestion de données de résoudre leurs problèmes d'apprentissage automatique (calcul graphique, diffusion en continu et traitement interactif en temps réel).

Remarques

Veuillez vous référer ci-dessous pour en savoir plus sur spark MLib

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

Examples

Écrivez votre premier problème de classification à l'aide du modèle de régression logistique

J'utilise eclipse ici, et vous devez ajouter ci-dessous la dépendance à votre 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 (votre classe d'application)

Nous faisons de la classification en fonction du pays, des heures et nous cliquons sur notre étiquette.

```
    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.) Pour exécuter cette application, exécutez d'abord `mvn-clean-package` sur le projet d'application, cela créerait jar. 4.) Ouvrez le répertoire d'étincelle et soumettez ce travail.

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

5.) Après la soumission, voir les données de formation

```
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.) données de test de la même manière

```
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.) Et voici le résultat de la prédiction sous la colonne de prédiction

```
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
```

Lire Démarrer avec Machine Learning en utilisant Apache spark MLib en ligne: https://riptutorial.com/fr/machine-learning/topic/9854/demarrer-avec-machine-learning-en-utilisant-apache-spark-mlib

Chapitre 4: Enseignement supervisé

Examples

Classification

Imaginez qu'un système souhaite détecter des **pommes** et des **oranges** dans un panier de fruits. Le système peut prélever un fruit, en extraire certaines propriétés (par exemple le poids de ce fruit).

Supposons que le système ait un enseignant! cela enseigne au système quels objets sont des **pommes** et lesquels sont des **oranges** . Ceci est un exemple de problème de **classification supervisé** . Il est supervisé car nous avons des exemples étiquetés. C'est la classification parce que la sortie est une prédiction de la classe à laquelle appartient notre objet.

Dans cet exemple, nous considérons 3 entités (propriétés / variables explicatives):

1. le poids du fruit sélectionné est-il supérieur à 5 grammes
2. est la taille supérieure à 10cm
3. est la couleur est rouge

(0 signifie non et 1 signifie oui)

Donc, pour représenter une pomme / orange, nous avons une série de trois propriétés (appelées vecteur)

(par exemple, [0,0,1] signifie que ce poids de fruit n'est pas supérieur à 0,5 gramme et que sa taille est inférieure à 10 cm et que sa couleur est rouge)

Donc, nous sélectionnons 10 fruits au hasard et mesurons leurs propriétés. L'enseignant (humain) identifie ensuite chaque fruit manuellement comme étant pomme => **[1]** ou orange => **[2]** .

Par exemple, le professeur choisit un fruit qui est pomme. La représentation de cette pomme pour le système pourrait être quelque chose comme ceci: **[1, 1, 1] => [1]** , cela signifie que ce fruit a un **poids supérieur à 0,5 gramme** , une **taille supérieure à 10 cm** et **3. la couleur de ce fruit est rouge** et enfin c'est une **pomme** (=> [1])

Ainsi, pour chacun des 10 fruits, l'enseignant a étiqueté chaque fruit comme étant pomme [=> 1] ou orange [=> 2] et le système a trouvé ses propriétés. comme vous pouvez le deviner, nous avons une série de vecteurs (appelés matrice) pour représenter 10 fruits entiers.

Classification des fruits

Dans cet exemple, un modèle apprend à classer les fruits en fonction de certaines caractéristiques, en utilisant les *étiquettes* pour la formation.

Poids	Couleur	Étiquette
0.5	vert	Pomme
0,6	violet	prune
3	vert	pastèque
0,1	rouge	Cerise
0.5	rouge	Pomme

Ici, le modèle prend comme *poids* et *couleur des* caractéristiques permettant de prédire l'étiquette. Par exemple, [0,15, «rouge»] devrait donner une prédiction «cerise».

Introduction à l'apprentissage supervisé

Il y a un certain nombre de situations dans lesquelles on a énormément de données et qu'il faut classer un objet dans l'une des classes connues. Considérez les situations suivantes:

Opérations bancaires: lorsqu'une banque reçoit une demande de carte bancaire d'un client, celle-ci doit décider d'émettre ou non la carte bancaire, en fonction des caractéristiques de ses clients bénéficiant déjà des cartes pour lesquelles l'historique de crédit est connu.

Médical: On peut être intéressé par la mise au point d'un système médical permettant de diagnostiquer un patient s'il présente ou non une maladie particulière, en fonction des symptômes observés et des tests médicaux effectués sur ce patient.

Finance: Une société de conseil financier souhaite prédire la tendance du prix d'une action qui peut être classée en tendance à la hausse, à la baisse ou nulle selon plusieurs caractéristiques techniques qui régissent les mouvements de prix.

Expression génique: une scientifique analysant les données d’expression génique souhaiterait identifier les gènes les plus pertinents et les facteurs de risque impliqués dans le cancer du sein, afin de séparer les patients sains des patients atteints de cancer du sein.

Dans tous les exemples ci-dessus, un objet est classé dans l'une de plusieurs classes *connues* , sur la base des mesures effectuées sur un certain nombre de caractéristiques, qu'il pense pouvoir distinguer les objets de différentes classes. Ces variables sont appelées variables *prédictives* et l'étiquette de classe est appelée variable *dépendante* . Notez que, dans tous les exemples ci-dessus, la variable dépendante est *catégorique* .

Pour développer un modèle pour le problème de classification, nous avons besoin, pour chaque objet, de données sur un ensemble de caractéristiques prescrites ainsi que des étiquettes de classes auxquelles les objets appartiennent. L'ensemble de données est divisé en deux ensembles dans un rapport prescrit. Le plus grand de ces ensembles de données s'appelle l'ensemble de *données d'apprentissage* et l'autre, *l'* ensemble de *données de test* . L'ensemble de données de formation est utilisé dans le développement du modèle. Comme le modèle est développé à l'aide d'observations dont les étiquettes de classes sont connues, ces modèles sont

appelés modèles d' *apprentissage supervisé* .

Après avoir développé le modèle, le modèle doit être évalué pour ses performances en utilisant l'ensemble de données de test. L'objectif d'un modèle de classification est d'avoir une probabilité minimale de classification erronée sur les observations non vues. Les observations non utilisées dans le développement du modèle sont connues sous le nom d'observations invisibles.

L'induction par arbre de décision est l'une des techniques de construction du modèle de classification. Le modèle d'arbre de décision créé pour la variable dépendante catégorielle s'appelle une *arborescence de classification* . La variable dépendante pourrait être numérique dans certains problèmes. Le modèle d'arbre de décision développé pour les variables dépendantes numériques s'appelle *Arbre de régression* .

Régression linéaire

Puisque l' **apprentissage supervisé** consiste en une variable cible ou une variable de résultat (ou variable dépendante) à prédire à partir d'un ensemble donné de prédicteurs (variables indépendantes). En utilisant cet ensemble de variables, nous générons une fonction qui associe les entrées aux sorties souhaitées. Le processus de formation se poursuit jusqu'à ce que le modèle atteigne le niveau de précision souhaité sur les données de formation.

> Par conséquent, il existe de nombreux exemples d'algorithmes d'apprentissage supervisé. Dans ce cas, je voudrais me concentrer sur la **régression linéaire.**

Régression linéaire Il est utilisé pour estimer les valeurs réelles (coût des maisons, nombre d'appels, ventes totales, etc.) sur la base de variables continues. Ici, nous établissons une relation entre les variables indépendantes et dépendantes en adaptant une meilleure ligne. Cette ligne de meilleur ajustement est appelée ligne de régression et représentée par une équation linéaire Y = a * X + b.

La meilleure façon de comprendre la régression linéaire est de revivre cette expérience de l’enfance. Disons que vous demandez à un enfant de cinquième année de classer les gens dans sa classe en augmentant son poids, sans leur demander leur poids! Que pensez-vous que l'enfant va faire? Il / elle serait probablement regarder (analyser visuellement) à la hauteur et la construction de personnes et les organiser en utilisant une combinaison de ces paramètres visibles.

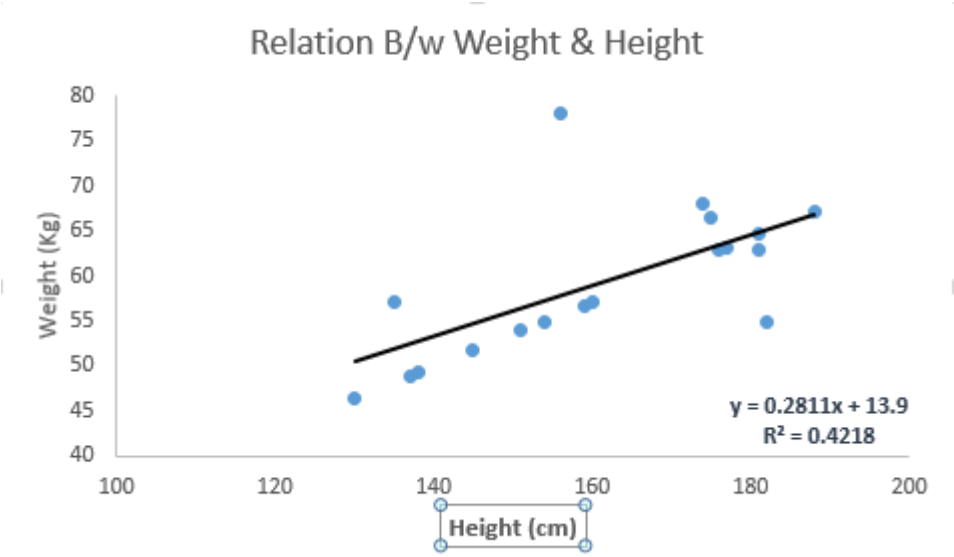
C'est la régression linéaire dans la vie réelle! L'enfant a effectivement compris que la taille et la construction seraient corrélées au poids par une relation, ce qui ressemble à l'équation ci-dessus.

Dans cette équation:

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

Ces coefficients a et b sont calculés en minimisant la somme de la différence au carré de la distance entre les points de données et la ligne de régression.

Regardez l'exemple ci-dessous. Nous avons identifié ici la ligne la mieux ajustée ayant une équation linéaire **y = 0.2811x + 13.9** . Maintenant, en utilisant cette équation, nous pouvons trouver le poids, en sachant la taille d'une personne.



La régression linéaire est principalement de deux types: la régression linéaire simple et la régression linéaire multiple. La régression linéaire simple est caractérisée par une variable indépendante. Et, la régression linéaire multiple (comme son nom l'indique) est caractérisée par plusieurs variables (plus de 1) indépendantes. Tout en trouvant la ligne la mieux adaptée, vous pouvez ajuster une régression polynomiale ou curviligne. Et ceux-ci sont connus sous le nom de régression polynomiale ou curviligne.

Juste un indice sur l'implémentation de la régression linéaire en 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)
```

J'ai donné un aperçu de la compréhension de l'apprentissage supervisé en utilisant l'algorithme de régression linéaire avec un extrait de code Python.

Lire Enseignement supervisé en ligne: https://riptutorial.com/fr/machine-learning/topic/2673/enseignement-supervise

Chapitre 5: L'apprentissage automatique et sa classification

Examples

Qu'est-ce que l'apprentissage automatique?

Deux définitions de l'apprentissage automatique sont proposées. *Arthur Samuel l'a* décrit comme:

> le domaine d'étude qui donne aux ordinateurs la capacité d'apprendre sans être explicitement programmés.

C'est une définition plus ancienne et informelle.

Tom Mitchell fournit une définition plus moderne:

> Un programme informatique est censé apprendre de l'expérience E en ce qui concerne une classe de tâches T et la mesure de la performance P, si sa performance aux tâches dans T, mesurée par P, s'améliore avec l'expérience E.

Exemple: jouer aux dames.

E = l'expérience de jouer à de nombreux jeux de dames

T = la tâche de jouer aux dames.

P = la probabilité que le programme gagne le prochain match.

En général, tout problème d'apprentissage automatique peut être classé dans l'une des deux grandes catégories suivantes:

1. Enseignement supervisé
2. Apprentissage non supervisé

Qu'est-ce que l'apprentissage supervisé?

L'apprentissage supervisé est un type d'algorithme d'apprentissage automatique qui utilise un ensemble de données connu (appelé ensemble de données d'apprentissage) pour effectuer des prédictions.

Catégorie d'apprentissage supervisé:

1. **Régression:** Dans un problème de régression, nous essayons de prédire les résultats dans une sortie continue, ce qui signifie que nous essayons de mapper les variables d'entrée à une fonction continue.
2. **Classification:** Dans un problème de classification, nous essayons plutôt de prédire les résultats dans une sortie discrète. En d'autres termes, nous essayons de mapper les

variables d'entrée en catégories distinctes.

Exemple 1:

En fonction des données sur la taille des maisons sur le marché immobilier, essayez de prévoir leur prix. Le prix en fonction de la taille est une sortie continue, il s'agit donc d'un problème de régression.

Exemple 2:

(a) *Régression* - Pour les valeurs de réponse continue. Par exemple, à partir d'une image d'une personne, il faut prévoir son âge sur la base de l'image donnée

(b) *Classification* - pour les valeurs de réponse catégorielles, où les données peuvent être séparées en «classes» spécifiques. Par exemple, pour un patient atteint d'une tumeur, il faut prédire si la tumeur est maligne ou bénigne.

Qu'est-ce que l'apprentissage non supervisé?

L'apprentissage non supervisé nous permet d'aborder les problèmes avec peu ou pas d'idée de nos résultats. Nous pouvons dériver une structure à partir de données où nous ne connaissons pas nécessairement l'effet des variables.

Exemple:

Clustering: est utilisé pour l'analyse de données exploratoires afin de trouver des modèles cachés ou un regroupement de données. Prenez une collection de 1 000 000 de gènes différents et trouvez un moyen de regrouper automatiquement ces gènes en groupes similaires ou liés par différentes variables, telles que la durée de vie, l'emplacement, les rôles, etc.

Lire L'apprentissage automatique et sa classification en ligne: https://riptutorial.com/fr/machine-learning/topic/9986/l-apprentissage-automatique-et-sa-classification

Chapitre 6: L'apprentissage en profondeur

Introduction

Deep Learning est un sous-domaine de l'apprentissage automatique où les réseaux neuronaux artificiels multicouches sont utilisés à des fins d'apprentissage. Deep Learning a trouvé de nombreuses implémentations géniales, telles que la reconnaissance vocale, les sous-titres sur Youtube, la recommandation Amazon, etc. Pour plus d'informations, il existe un sujet dédié à l'apprentissage en profondeur .

Examples

Bref résumé de l'apprentissage en profondeur

Pour former un réseau de neurones, nous devons tout d'abord concevoir une idée efficace. Il existe trois types de tâches d'apprentissage.

- Enseignement supervisé
- Apprentissage par renforcement
- Apprentissage non supervisé

À l'heure actuelle, l'apprentissage non supervisé est très populaire. L'apprentissage non supervisé est une tâche d'apprentissage en profondeur consistant à inférer une fonction pour décrire une structure cachée à partir de données «non étiquetées» (une classification ou une catégorisation n'est pas incluse dans les observations).

Étant donné que les exemples donnés à l'apprenant ne sont pas étiquetés, il n'y a pas d'évaluation de la précision de la structure produite par l'algorithme pertinent, ce qui constitue une façon de distinguer l'apprentissage non supervisé de l'apprentissage supervisé et de l'apprentissage par renforcement.

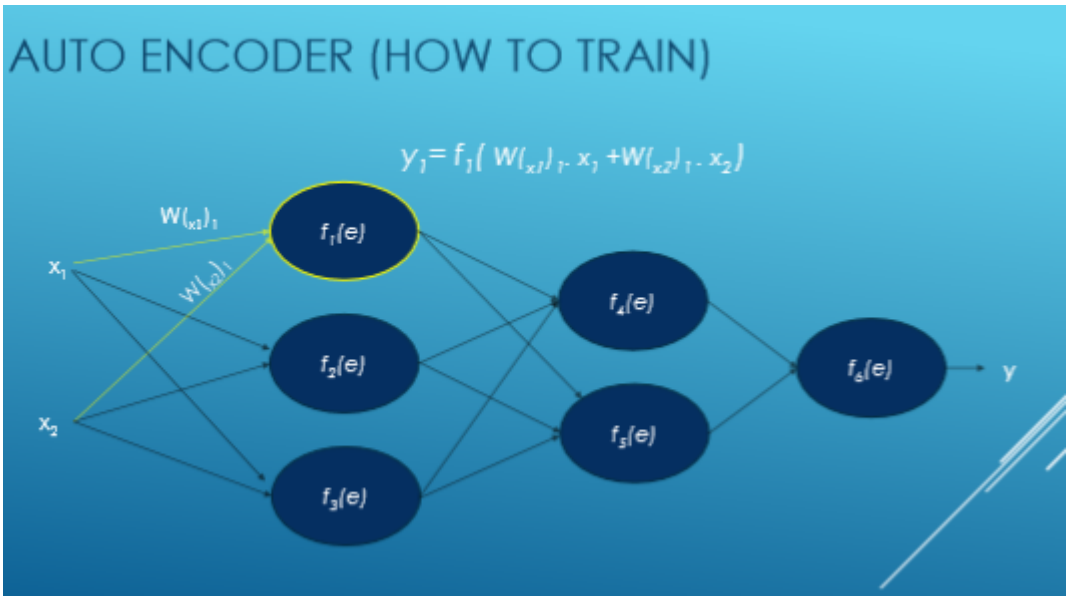
Il existe trois types d’apprentissage non supervisé.

- Machines Boltzmann à usage restreint
- Modèle de codage épars
- Autoencoders Je décrirai en détail l'autoencoder.

Le but d'un autoencodeur est d'apprendre une représentation (encodage) pour un ensemble de données, généralement dans le but de réduire les dimensions.

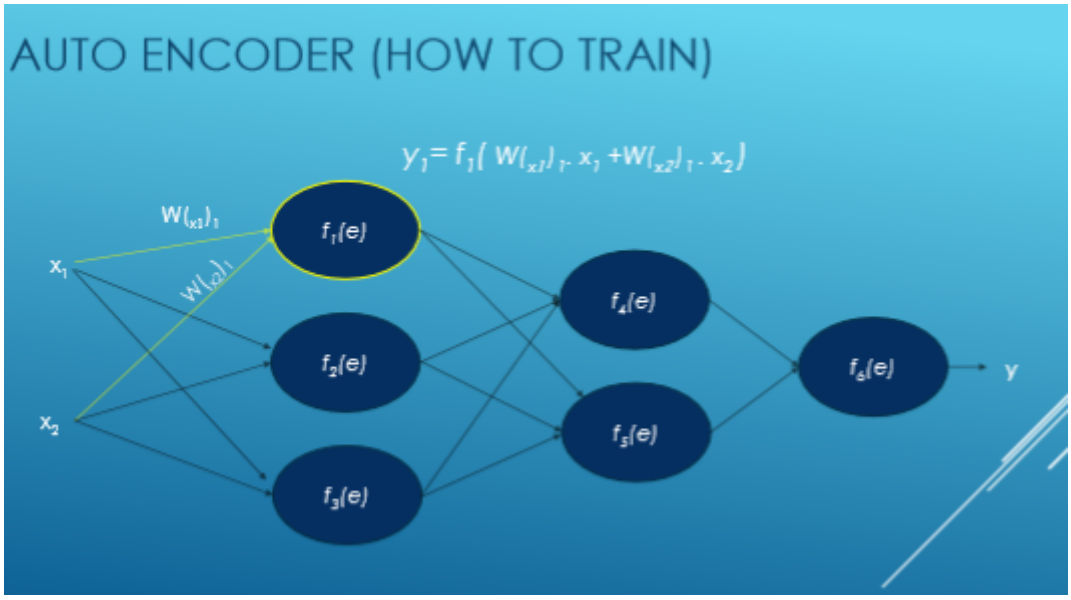
La forme la plus simple d'un autoencodeur est une anticipation, avec une couche d'entrée, une couche de sortie et une ou plusieurs couches cachées qui les connectent. Mais avec la couche de sortie ayant le même nombre de nœuds que la couche d'entrée, et dans le but de reconstruire ses propres entrées, c'est ce qu'on appelle l'apprentissage non supervisé.

Je vais maintenant essayer de donner un exemple de réseau neuronal d'entraînement.

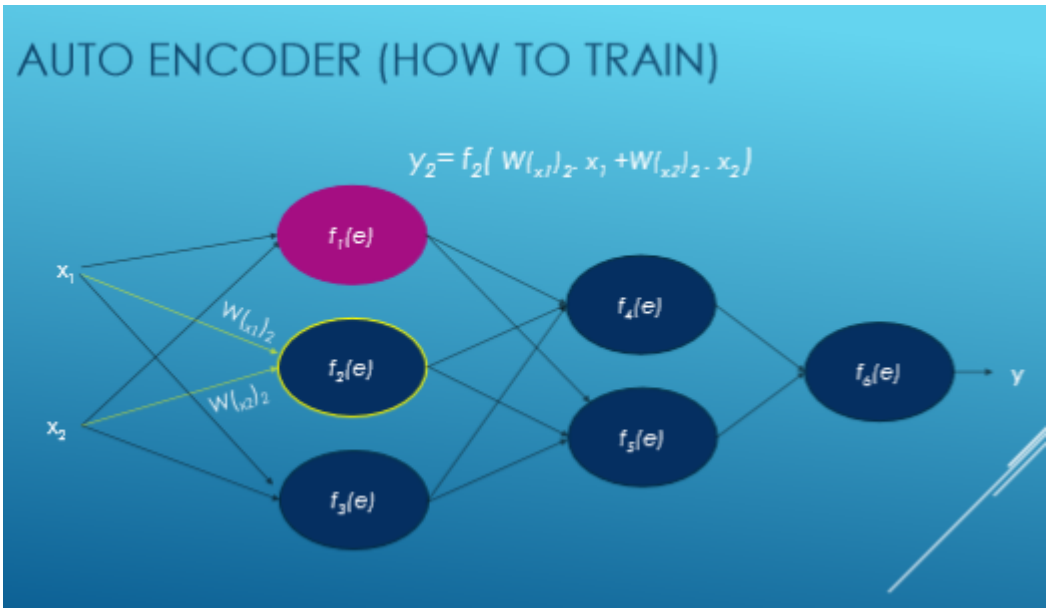


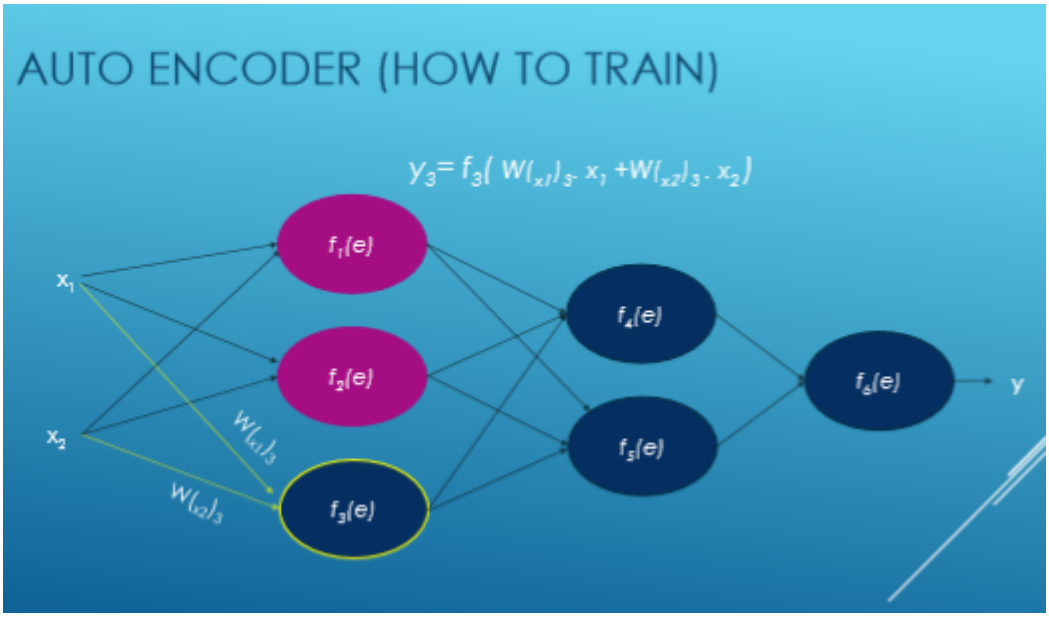
Ici Xi est entré, W est le poids, f (e) est la fonction d'activation et y est sorti.

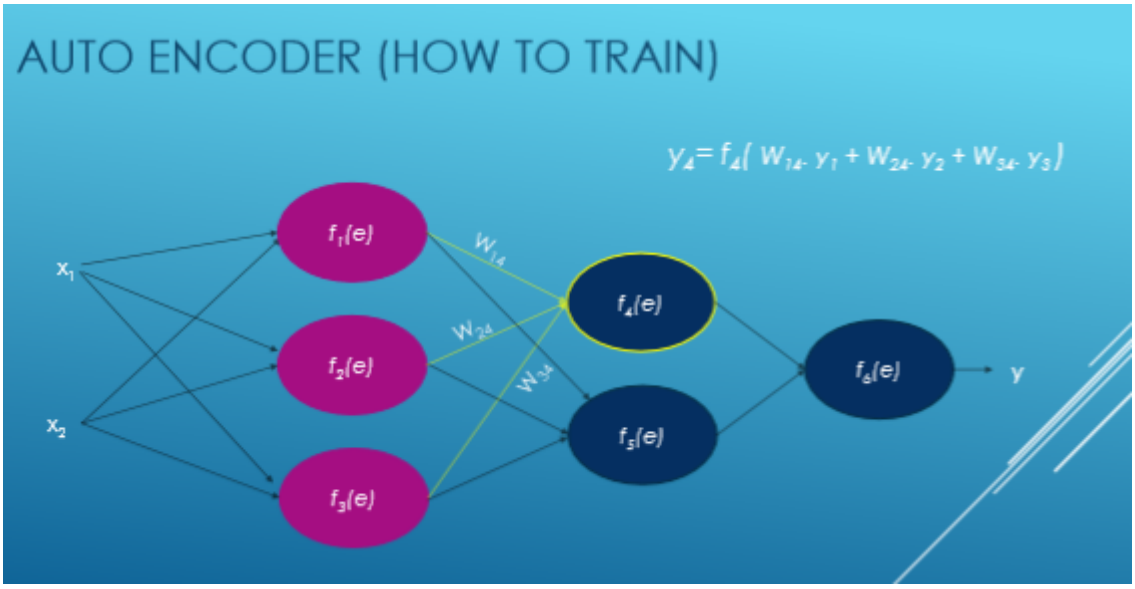
Nous voyons maintenant un flux étape par étape de réseau neuronal d'entraînement basé sur le codage automatique.

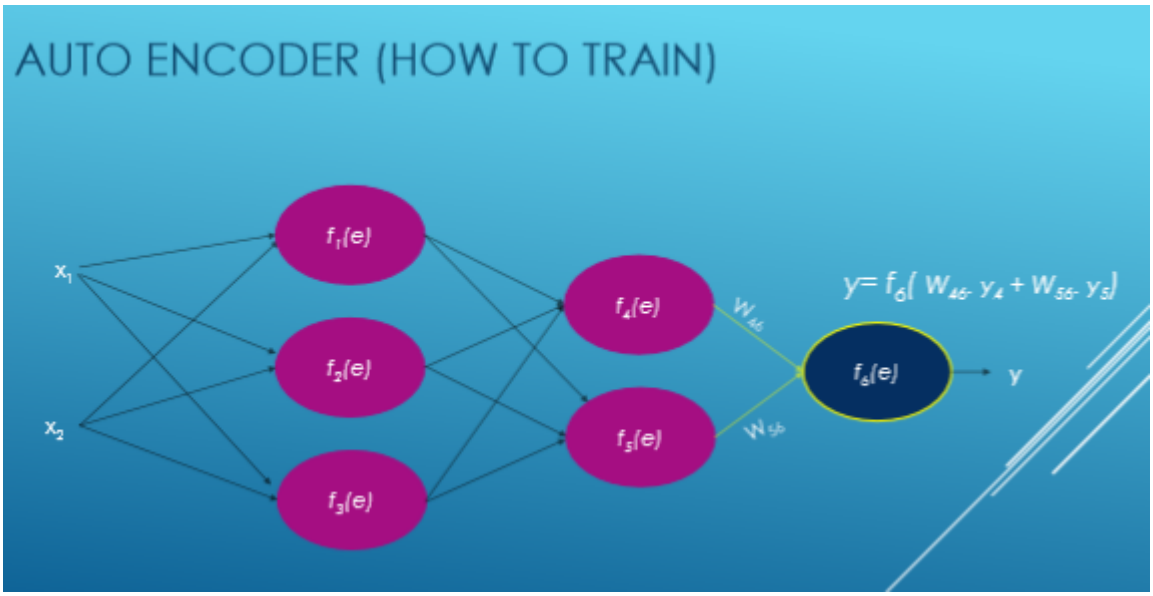


Nous calculons la valeur de chaque fonction d'activation avec cette équation: y = WiXi. Tout d'abord, nous choisissons au hasard des nombres pour les poids et essayons ensuite d'ajuster ces poids.

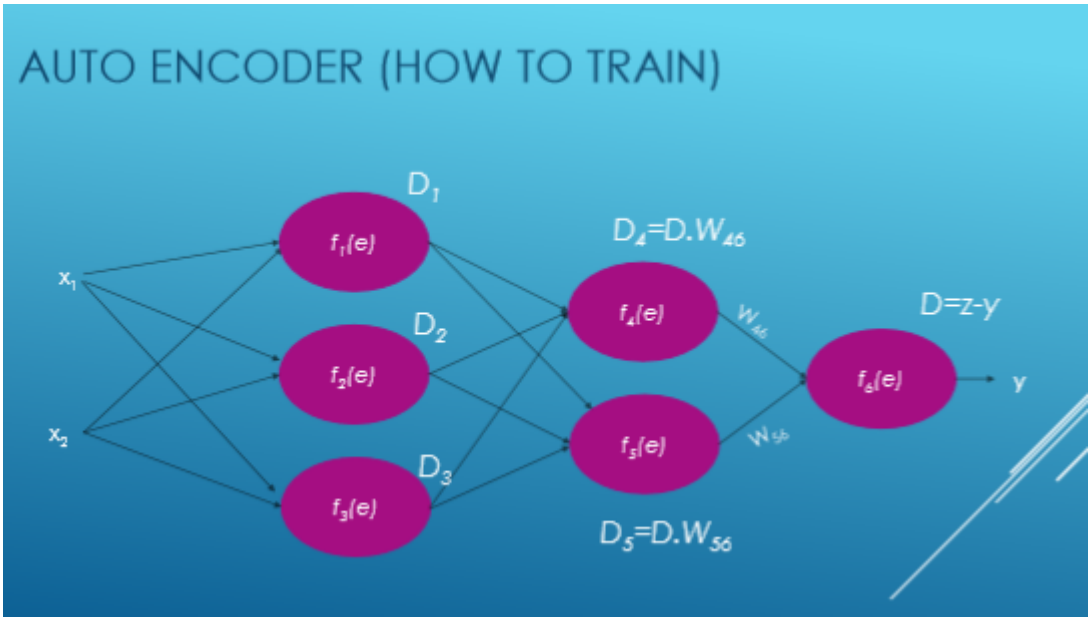

AUTO ENCODER (HOW TO TRAIN)
$y_2 = f_2(W_{(x1)2} \cdot x_1 + W_{(x2)2} \cdot x_2)$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
y
$W_{(x1)2}$
$W_{(x2)2}$


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
$W_{(x1)3}$
$W_{(x2)3}$

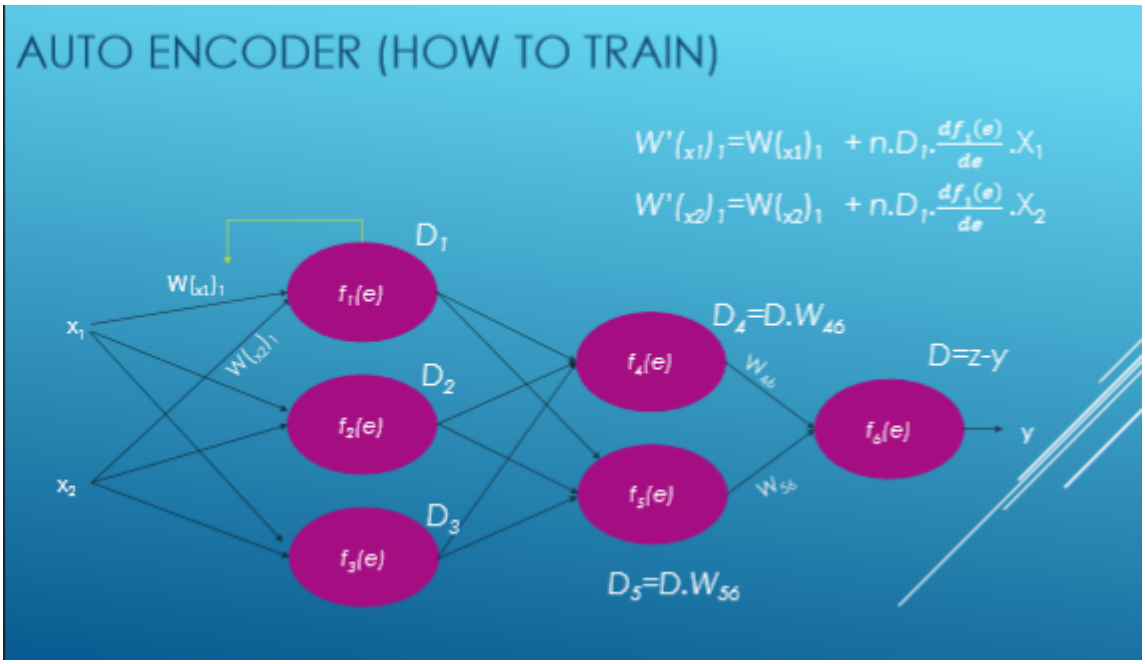

AUTO ENCODER (HOW TO TRAIN)
$y_4 = f_4(W_{14} \cdot y_1 + W_{24} \cdot y_2 + W_{34} \cdot y_3)$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
y
W_{14}
W_{24}
W_{34}



Maintenant, nous calculons l'écart par rapport à notre sortie souhaitée, c'est-à-dire y = zy et calculons les déviations de chaque fonction d'activation.



Ensuite, nous ajustons notre nouveau poids de chaque connexion.



Lire L'apprentissage en profondeur en ligne: https://riptutorial.com/fr/machine-learning/topic/7442/l-apprentissage-en-profondeur

Chapitre 7: Les réseaux de neurones

Examples

Pour commencer: un simple ANN avec Python

La liste de codes ci-dessous tente de classer les chiffres manuscrits de l'ensemble de données MNIST. Les chiffres ressemblent à ceci:

Le code prétraitera ces chiffres, convertissant chaque image en un tableau 2D de 0 et de 1, puis utilisera ces données pour former un réseau neuronal avec une précision pouvant atteindre 97% (50 époques).

```
"""
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)) + "%"
```

Ceci est un exemple de code autonome, et peut être exécuté sans autres modifications. Assurez-vous que `numpy` et `scikit numpy` installés pour votre version de python.

Backpropagation - Le cœur des réseaux de neurones

Le but de la contre-propagande est d'optimiser les poids afin que le réseau neuronal puisse apprendre à mapper correctement les entrées arbitraires aux sorties.

Chaque couche a son propre ensemble de poids, et ces poids doivent être ajustés pour pouvoir prédire avec précision le bon résultat fourni.

Une vue d'ensemble de haut niveau de la propagation arrière est la suivante:

1. Pass en avant - l'entrée est transformée en sortie. À chaque couche, l'activation est calculée avec un produit scalaire entre l'entrée et les poids, suivie de la somme du résultat avec le biais. Enfin, cette valeur est transmise via une fonction d'activation pour obtenir l'activation de cette couche qui deviendra l'entrée de la couche suivante.
2. Dans la dernière couche, la sortie est comparée à l'étiquette réelle correspondant à cette entrée et l'erreur est calculée. Habituellement, c'est l'erreur quadratique moyenne.
3. Passe en arrière - l'erreur calculée à l'étape 2 est renvoyée aux couches internes et les poids de tous les calques sont ajustés pour tenir compte de cette erreur.

1. Initialisation des poids

Un exemple simplifié d'initialisation des poids est présenté ci-dessous:

```
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:]])
```

- La couche cachée 1 a un poids de dimension [64, 784] et un biais de dimension 64.
- La couche de sortie a le poids de la dimension [10, 64] et le biais de la dimension

    10.

Vous vous demandez peut-être ce qui se passe lors de l'initialisation des poids dans le code ci-dessus. Cela s'appelle l'initialisation de Xavier, et c'est une étape meilleure que l'initialisation aléatoire de vos matrices de poids. Oui, l'initialisation est importante. En fonction de votre initialisation, vous pourrez peut-être trouver de meilleurs minima locaux pendant la descente de gradient (la propagation arrière est une version glorifiée de la descente de gradient).

2. Pass en avant

```
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))
```

Ce code effectue la transformation décrite ci-dessus. `hidden_activations[-1]` contient des probabilités softmax - prédictions de toutes les classes, dont la somme est 1. Si nous prédisons des chiffres, alors la sortie sera un vecteur de probabilités de dimension 10, dont la somme est 1.

3. Passage arrière

```
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
```

Les 2 premières lignes initialisent les dégradés. Ces gradients sont calculés et seront utilisés pour mettre à jour les poids et les biais ultérieurement.

Les 3 lignes suivantes calculent l'erreur en soustrayant la prédiction de la cible. L'erreur est ensuite propagée aux couches internes.

Maintenant, tracez soigneusement le fonctionnement de la boucle. Les lignes 2 et 3 transforment l'erreur de la `layer[i]` en `layer[i - 1]` . Tracer les formes des matrices en cours de multiplication pour comprendre.

4. Mise à jour des poids / paramètres

```
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` spécifie la vitesse à laquelle le réseau apprend. Vous ne voulez pas qu'elle apprenne trop vite, car elle risque de ne pas converger. Une descente en douceur est favorisée pour trouver un bon minimum. Généralement, les taux compris entre `0.01` et `0.1` sont considérés comme bons.

Fonctions d'activation

Les fonctions d'activation, également connues sous le nom de fonction de transfert, sont utilisées pour mapper les nœuds d'entrée vers les nœuds de sortie d'une certaine manière.

Ils sont utilisés pour donner une non-linéarité à la sortie d'une couche de réseau neuronal.

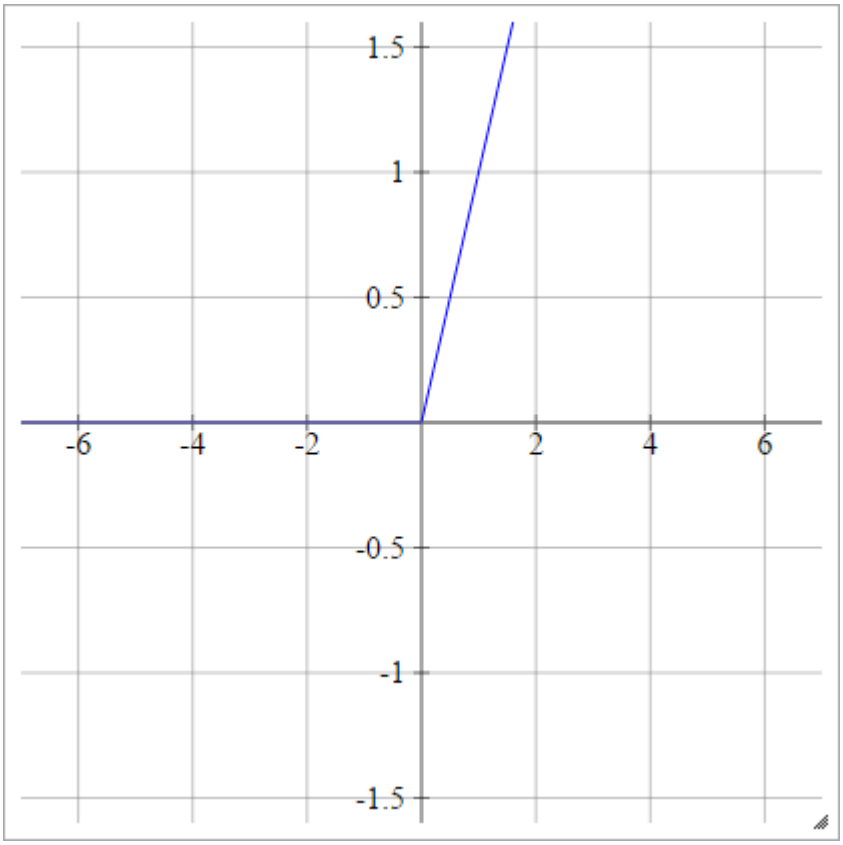
Certaines fonctions couramment utilisées et leurs courbes sont données ci-dessous:

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	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

pour calculer l'activation d'une couche masquée.

Fonction ReLU

Une unité linéaire rectifiée fait simplement `max(0,x)` . C'est l'un des choix les plus courants pour les fonctions d'activation des unités de réseau neuronal.



Les ReLU traitent le problème du gradient de disparition des unités sigmoïdes / tangentes hyperboliques, permettant ainsi une propagation efficace du gradient dans les réseaux profonds.

Le nom ReLU provient de l'article de Nair et Hinton, *Les unités linéaires rectifiées améliorent les machines Boltzmann restreintes* .

Il a quelques variantes, par exemple, les ReLUs qui fuient (LReLUs) et les Unités linéaires exponentielles (ELU).

Le code pour implémenter vanilla ReLU avec son dérivé avec `numpy` est indiqué ci-dessous:

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

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

Fonction Softmax

La régression softmax (ou régression logistique multinomiale) est une généralisation de la régression logistique au cas où nous voulons traiter plusieurs classes. Il est particulièrement utile

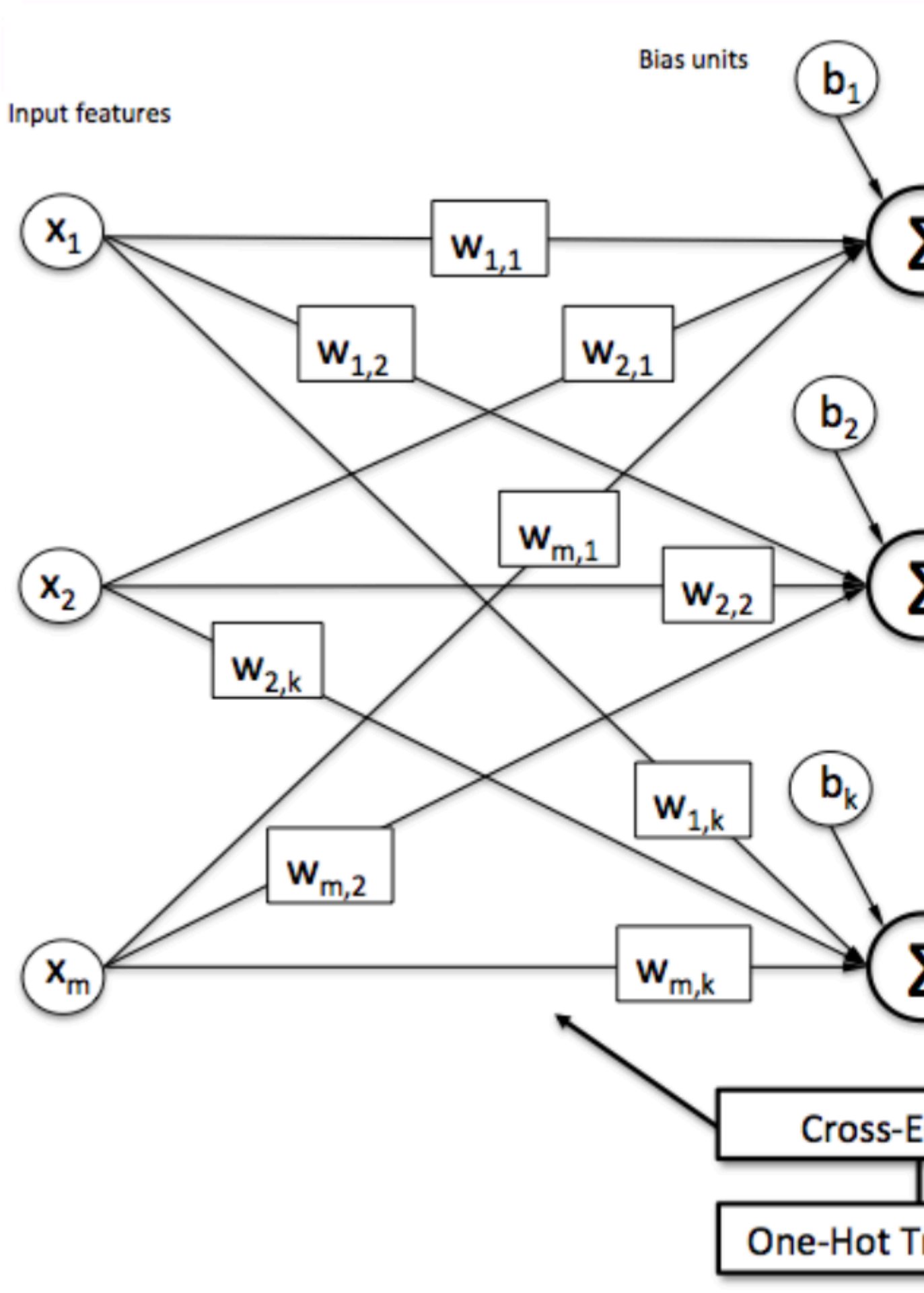
pour les réseaux de neurones où l'on souhaite appliquer une classification non binaire. Dans ce cas, la simple régression logistique ne suffit pas. Nous aurions besoin d'une distribution de probabilités pour toutes les étiquettes, ce que Softmax nous donne.

Softmax est calculé avec la formule ci-dessous:

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

_____Où est-ce que ça correspond?

_____


Bias units
Input features
b_1
x_1
$w_{1,1}$
$w_{1,2}$
$w_{2,1}$
b_2
$w_{m,1}$
x_2
$w_{2,2}$
$w_{2,k}$
b_k
$w_{1,k}$
$w_{m,2}$
x_m
$w_{m,k}$
Cross-E
One-Hot T

Softmax Re

Pour normaliser un vecteur en lui appliquant la fonction softmax avec `numpy` , utilisez:

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

Où `x` est l'activation de la couche finale de l'ANN.

Lire Les réseaux de neurones en ligne: https://riptutorial.com/fr/machine-learning/topic/2709/les-reseaux-de-neurones

Chapitre 8: Mesures d'évaluation

Examples

Zone sous la courbe de la caractéristique de fonctionnement du récepteur (AUROC)

L' **AUROC** est l'une des métriques les plus utilisées pour évaluer les performances d'un classificateur. Cette section explique comment le calculer.

AUC (Area Under the Curve) est utilisé la plupart du temps pour désigner AUROC, ce qui est une mauvaise pratique car AUC est ambigu (peut être n'importe quelle courbe) alors que AUROC ne l'est pas.

Vue d'ensemble - Abréviations

Abréviation	Sens
AUROC	Zone sous la courbe de la caractéristique de fonctionnement du récepteur
AUC	Zone sous la malédiction
ROC	Caractéristique de fonctionnement du récepteur
TP	Vrais positifs
TN	Véritables négatifs
FP	Faux positifs
FN	Faux négatifs
TPR	Vrai taux positif
FPR	Faux Taux positif

Interpréter l'AUROC

L'AUROC a plusieurs interprétations équivalentes :

- L'attente d'un résultat aléatoire aléatoire uniformément établi avant un négatif aléatoire dessiné uniformément.
- La proportion attendue de positifs classés avant un négatif aléatoire uniformément dessiné.
- Le taux positif réel attendu si le classement est divisé juste avant un négatif aléatoire uniformément dessiné.

- La proportion attendue de négatifs classés après un résultat aléatoire aléatoire uniformément dessiné.
- Le taux de faux positif attendu si le classement est divisé juste après un résultat aléatoire aléatoire uniformément dessiné.

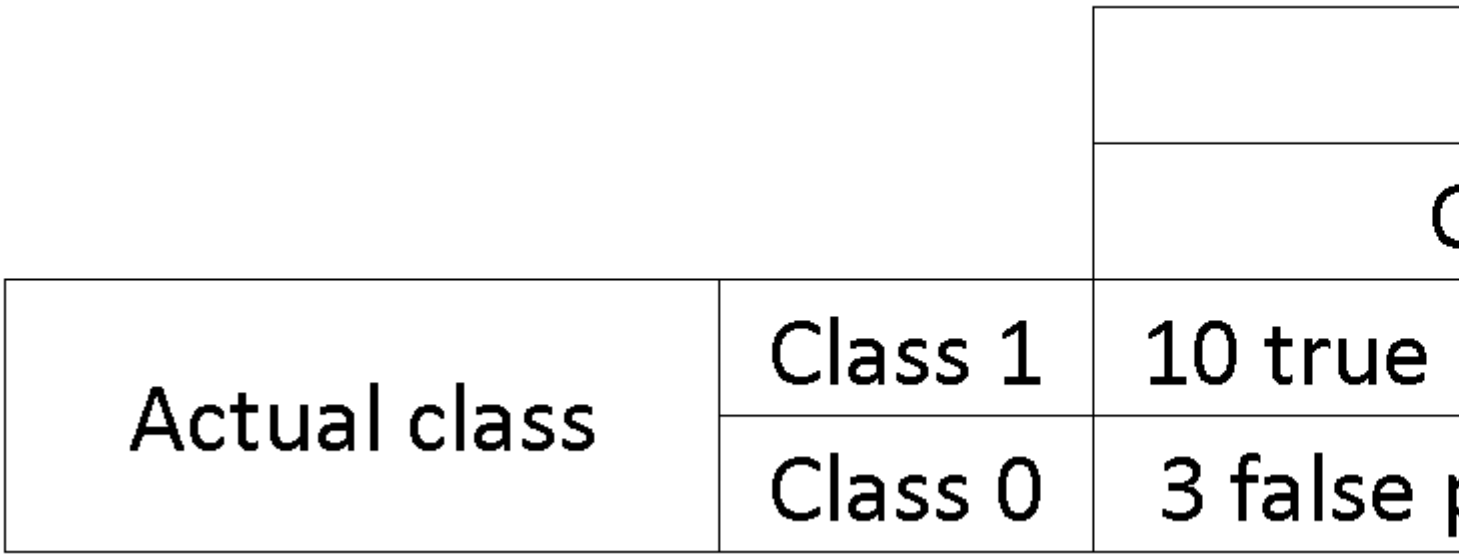
Calculer l'AUROC

Supposons que nous ayons un classificateur binaire probabiliste tel que la régression logistique.

Avant de présenter la courbe **ROC** (= courbe de fonctionnement du récepteur), le concept de **matrice** de **confusion** doit être compris. Lorsque nous faisons une prédiction binaire, il peut y avoir 4 types de résultats:

- Nous prédisons **0** alors que la classe est en fait **0** : cela s'appelle un **vrai négatif** , c'est-à-dire que nous prédisons correctement que la classe est négative (0). *Par exemple, un antivirus n'a pas détecté de fichier inoffensif en tant que virus.*
- Nous prédisons **0** alors que la classe est en fait **1** : cela s'appelle un **faux négatif** , c'est-à-dire que nous prédisons incorrectement que la classe est négative (0). *Par exemple, un antivirus n'a pas réussi à détecter un virus.*
- Nous prédisons **1** alors que la classe est en fait **0** : cela s'appelle un **faux positif** , c'est-à-dire que nous prédisons incorrectement que la classe est positive (1). *Par exemple, un antivirus considère un fichier inoffensif comme un virus.*
- Nous prédisons **1** alors que la classe est en réalité **1** : cela s'appelle un **True Positive** , c'est-à-dire que nous prédisons correctement que la classe est positive (1). *Par exemple, un antivirus a détecté à juste titre un virus.*

Pour obtenir la matrice de confusion, nous passons en revue toutes les prédictions faites par le modèle et comptons le nombre de fois que chacun de ces 4 types de résultats se produit:



Dans cet exemple de matrice de confusion, parmi les 50 points de données classés, 45 sont

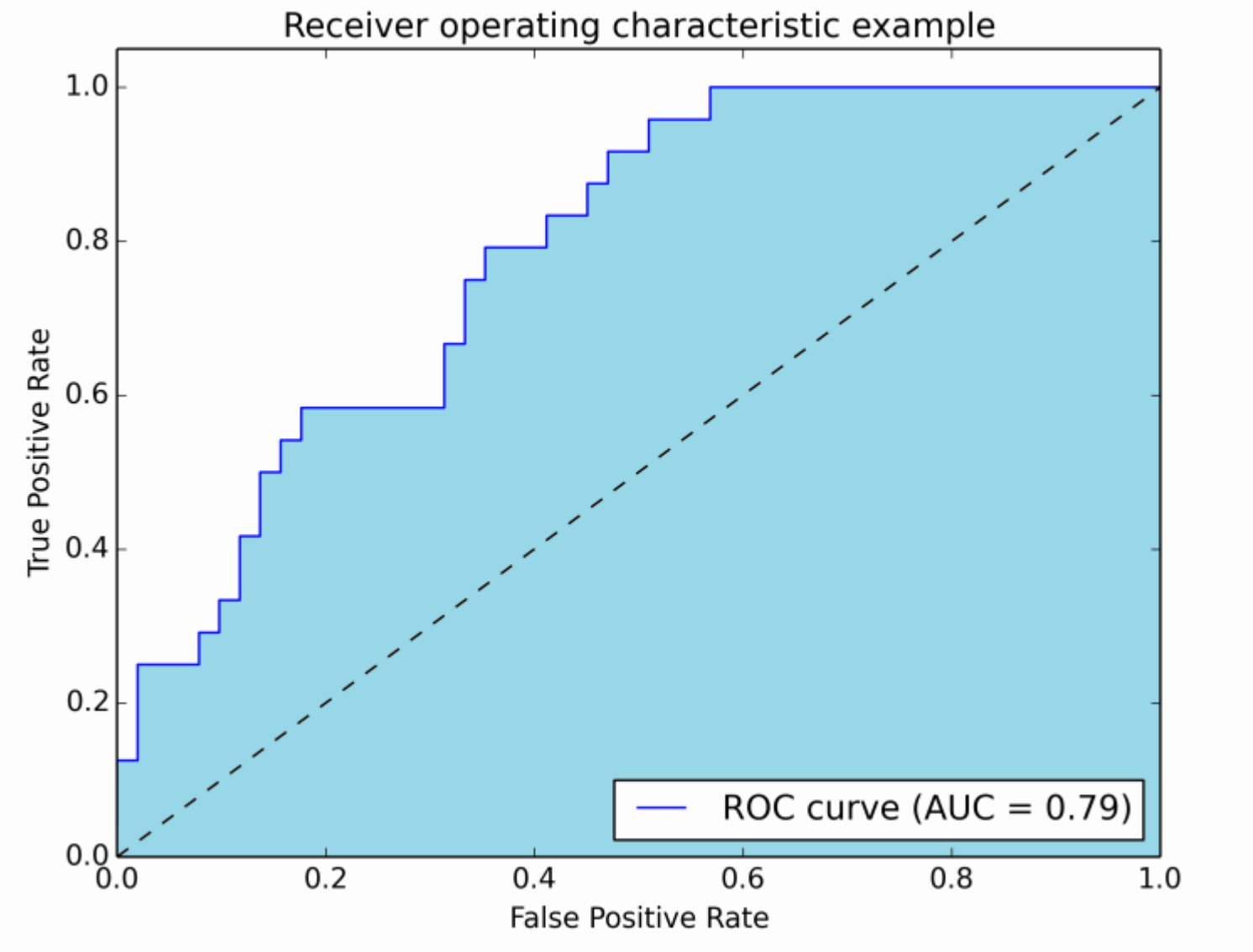
correctement classés et les 5 sont mal classés.

Puisque pour comparer deux modèles différents, il est souvent plus pratique d'avoir une seule métrique plutôt que plusieurs, nous calculons deux métriques à partir de la matrice de confusion, que nous regrouperons plus tard:

- True taux positif (**TPR**), aka. sensibilité, taux de réussite et rappel , défini comme $\frac{TP}{TP+FN}$. Intuitivement, cette métrique correspond à la proportion de points de données positifs correctement considérés comme positifs par rapport à tous les points de données positifs. En d'autres termes, le TPR plus élevé, le moins de points de données positifs nous manquerons.
- Taux de faux positifs (**FPR**), alias. retombées , qui est défini comme $\frac{FP}{FP+TN}$. Intuitivement, cette mesure correspond à la proportion de points de données négatifs considérés à tort comme positifs par rapport à tous les points de données négatifs. En d'autres termes, plus le FPR est élevé, plus les points de données négatifs seront classés.

Pour combiner le FPR et le TPR en une seule mesure, nous calculons d'abord les deux anciennes mesures avec de nombreux seuils différents (par exemple: $0.00,0.01…1.00$) pour la régression logistique, puis les tracer sur un seul graphique, avec les valeurs FPR en abscisse et les valeurs TPR en ordonnée. La courbe résultante est appelée courbe ROC, et la mesure que nous considérons est l'AUC de cette courbe, que nous appelons AUROC.

La figure suivante montre graphiquement l'AUROC:



Dans cette figure, la zone bleue correspond à la zone située sous la courbe de la caractéristique de fonctionnement du récepteur (AUROC). La ligne en tirets dans la diagonale présente la courbe ROC d'un prédicteur aléatoire: elle a un AUROC de 0,5. Le prédicteur aléatoire est couramment utilisé comme base pour voir si le modèle est utile.

Matrice de confusion

Une matrice de confusion peut être utilisée pour évaluer un classificateur, sur la base d'un ensemble de données de test pour lesquelles les vraies valeurs sont connues. C'est un outil simple, qui aide à donner un bon aperçu visuel des performances de l'algorithme utilisé.

Une matrice de confusion est représentée sous forme de tableau. Dans cet exemple, nous allons examiner une **matrice de confusion pour un classificateur binaire** .

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

Sur le côté gauche, on peut voir la classe Actual (appelée *YES* ou *NO*), tandis que la partie supérieure indique la classe prédite et sortie (encore une fois *YES* ou *NO*).

Cela signifie que 50 instances de test - qui ne sont en réalité *AUCUNE* instance, ont été correctement étiquetées par le classificateur comme *NO* . On les appelle les **vrais négatifs (TN)** . En revanche, 100 instances *YES* réelles ont été correctement étiquetées par le classificateur comme des instances *YES* . On les appelle les **vrais positifs (TP)** .

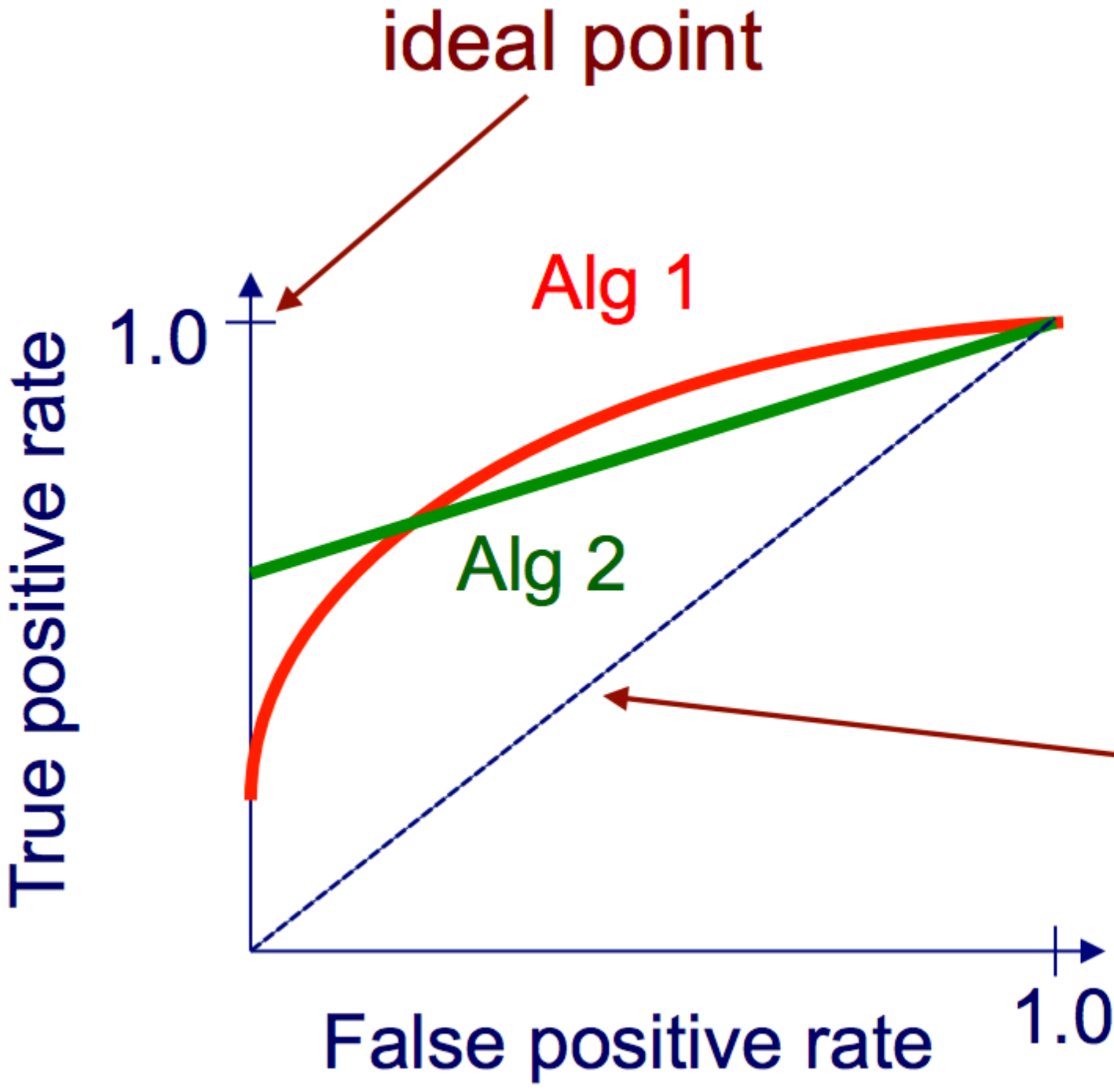
5 cas *OUI* réels ont été mal étiquetés par le classificateur. Ceux-ci s'appellent les **faux négatifs (FN)** . En outre, 10 *NO* cas, ont été considérés comme des instances *OUI* par le classifieur, donc ce sont des **faux positifs (PF)** .

Sur la base de ces **PF** , **TP** , **FN** et **TN** , nous pouvons tirer d'autres conclusions.

- **True Taux positif** :
    - *Essaie de répondre:* Lorsqu'une instance est en fait *OUI* , à quelle fréquence le classificateur prévoit-il *OUI* ?
    - *Peut être calculé comme suit:* **TP** / # instances *YES* réelles = 100/105 = 0,95
- **Faux Taux positif** :
    - *Essaie de répondre:* Quand une instance est en fait *NON* , à quelle fréquence le classificateur prévoit-il *OUI* ?
    - *Peut être calculé comme suit:* **FP** / # *NO* instances réelles = 10/60 = 0.17

Courbes ROC

Une courbe ROC (Receiver Operating Characteristic) trace le taux de TP en fonction du taux de FP, car un seuil de confiance pour une instance positive est varié



Algorithme pour créer une courbe ROC

1. trier les prévisions de test selon la confiance que chaque instance est positive
2. passer à travers la liste triée de confiance élevée à faible

   je. localiser un seuil entre des instances de classes opposées (en gardant les instances avec la même valeur de confiance du même côté du seuil)

ii. calculer TPR, FPR pour les instances au-dessus du seuil

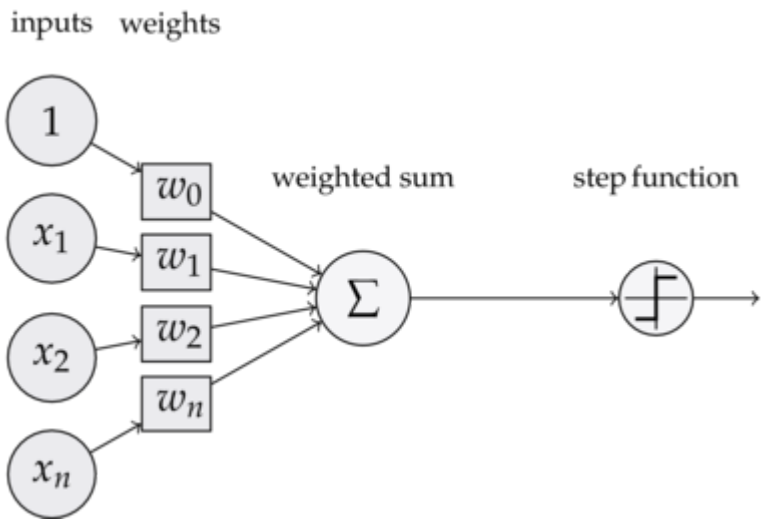
iii. coordonnée de sortie (FPR, TPR)

Lire Mesures d'évaluation en ligne: https://riptutorial.com/fr/machine-learning/topic/4132/mesures-d-evaluation

Chapitre 9: Perceptron

Examples

Qu'est-ce qu'un perceptron exactement?



À la base, un modèle de perceptron est l'un des algorithmes d' **apprentissage supervisé** les plus simples pour la classification binaire . C'est un type de classificateur linéaire , c'est-à-dire un algorithme de classification qui fait ses prédictions sur la base d'une fonction de prédicteur linéaire combinant un ensemble de poids avec le vecteur de caractéristiques. Une manière plus intuitive de penser est comme un **réseau neuronal avec un seul neurone** .

Le fonctionnement est très simple. Il obtient un vecteur de valeurs d'entrée ***x*** dont chaque élément est une caractéristique de notre ensemble de données.

Un exemple:

Dites que nous voulons classer si un objet est un vélo ou une voiture. Par souci de cet exemple, supposons que nous souhaitons sélectionner 2 fonctionnalités. La hauteur et la largeur de l'objet. Dans ce cas, ***x = [x1, x2]*** où ***x1*** est la hauteur et ***x2*** la largeur.

Ensuite, une fois que nous avons notre vecteur d'entrée ***x,*** nous voulons multiplier chaque élément de ce vecteur par un poids. Plus la valeur du poids est élevée, plus la caractéristique est importante. Si, par exemple, nous utilisions la ***couleur*** comme caractéristique ***x3*** et qu'il y a un vélo rouge et une voiture rouge, le perceptron lui attribue un poids très faible, de sorte que la couleur n'affecte pas la prévision finale.

Bon donc nous avons multiplié les 2 vecteurs ***x*** et ***w*** et nous avons récupéré un vecteur. Maintenant, nous devons additionner les éléments de ce vecteur. Une manière intelligente de le faire est de multiplier ***x*** par ***w en*** multipliant ***x*** par ***wT*** où ***T*** signifie transposer. Nous pouvons imaginer la ***transposition*** d'un vecteur en tant que version pivotée du vecteur. Pour plus

d'informations, vous pouvez lire la page Wikipedia . Essentiellement, en prenant la transposition du vecteur ***w,*** nous obtenons un vecteur ***Nx1*** au lieu d'un vecteur ***1xN*** . Ainsi, si nous multiplions maintenant notre vecteur d'entrée de taille ***1xN*** avec ce ***vecteur de*** poids ***Nx1,*** nous obtiendrons un vecteur ***1x1*** (ou simplement une valeur unique) qui sera égal à ***x1 * w1 + x2 * w2 + ... + xn * wn*** . Cela fait, nous avons maintenant notre prédiction. Mais il y a une dernière chose. Cette prédiction ne sera probablement pas simple 1 ou -1 pour pouvoir classer un nouvel échantillon. Nous pouvons donc simplement dire ce qui suit: Si notre prédiction est supérieure à 0, nous disons que l'échantillon appartient à la classe 1, sinon si la prédiction est inférieure à zéro, nous disons que l'échantillon appartient à la classe -1. Cela s'appelle une fonction pas à pas .

Mais comment pouvons-nous avoir les bons poids pour que nous puissions corriger les prédictions? En d' autres termes, comment pouvons-nous ***entraînons*** notre modèle perceptron?

Dans le cas du perceptron, nous n'avons pas besoin d'équations mathématiques sophistiquées pour ***former*** notre modèle. Nos poids peuvent être ajustés par l'équation suivante:

Δw = eta * (y - prédiction) * x (i)

où ***x (i)*** est notre fonctionnalité (x1 par exemple pour le poids 1, x2 pour w2 et ainsi de suite ...).

On a également remarqué qu'il y a une variable appelée ***eta*** qui est le taux d'apprentissage. Vous pouvez imaginer le taux d'apprentissage comme étant la taille que nous voulons que le changement de poids soit. Un bon taux d'apprentissage permet d'obtenir un algorithme d'apprentissage rapide. Une valeur trop élevée de ***eta*** peut entraîner une augmentation du nombre d'erreurs à chaque époque et aboutir à des prédictions vraiment mauvaises pour le modèle et ne jamais converger. Un taux d'apprentissage trop faible peut avoir pour conséquence que le modèle met trop de temps à converger. (Habituellement, une valeur intéressante pour définir ***eta*** pour le modèle perceptron est de 0,1 mais elle peut différer d'un cas à l'autre).

Enfin, certains d'entre vous ont peut-être remarqué que la première entrée est une constante (1) et qu'elle est multipliée par w0. Alors, c'est quoi exactement? Pour avoir une bonne prédiction, nous devons ajouter un biais. Et c'est exactement ce que cette constante est.

Pour modifier le poids du terme de biais, nous utilisons la même équation que pour les autres poids, mais dans ce cas, nous ne le multiplions pas par l'entrée (car l'entrée est une constante 1 et nous n'avons pas à le faire):

Δw = eta * (y - prédiction)

Voilà donc comment fonctionne un modèle simple de perceptron! Une fois que nous formons nos poids, nous pouvons lui donner de nouvelles données et avoir nos prévisions.

REMARQUE:

Le modèle Perceptron a un inconvénient important! Il ne convergera jamais (ei trouvera les poids parfaits) si les données ne sont pas linéairement séparables , ce qui signifie être capable de séparer les 2 classes d'un espace d'entités par une ligne droite. Donc, pour éviter cela, il est

recommandé d'ajouter un nombre fixe d'itérations afin que le modèle ne soit pas bloqué par le réglage des poids qui ne sera jamais parfaitement ajusté.

Implémentation d'un modèle Perceptron en C ++

Dans cet exemple, je vais passer en revue l'implémentation du modèle perceptron en C ++ afin que vous puissiez avoir une meilleure idée de son fonctionnement.

Tout d'abord, il est recommandé de noter un algorithme simple de ce que nous voulons faire.

Algorithme:

1. Faites un vecteur pour les poids et initialisez-le à 0 (n'oubliez pas d'ajouter le terme de biais)
2. Continuez à ajuster les poids jusqu'à ce que nous obtenions 0 erreurs ou un faible nombre d'erreurs.
3. Faites des prédictions sur des données inédites.

Après avoir écrit un algorithme très simple, écrivons maintenant certaines des fonctions dont nous aurons besoin.

- Nous aurons besoin d'une fonction pour calculer l'entrée du réseau (ei ***x * wT*** multipliant le temps d'entrée des poids)
- Une fonction pas à pas pour obtenir une prédiction de 1 ou -1
- Et une fonction qui trouve les valeurs idéales pour les poids.

Donc, sans plus tarder, allons droit au but.

Commençons simple en créant une classe perceptron:

```
class perceptron
{
public:

private:

};
```

Ajoutons maintenant les fonctions dont nous aurons besoin.

```
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:

};
```

Remarquez comment la fonction ***fit*** prendre en argument un vecteur de vecteur <float>. En effet, notre ensemble de données de formation est une matrice d’intrants. Essentiellement, nous

pouvons imaginer cette matrice comme un couple de vecteurs ***x*** empilés les uns sur les autres et chaque colonne de cette matrice étant une caractéristique.

Enfin ajoutons les valeurs que notre classe doit avoir. Comme le vecteur ***w*** pour contenir les poids, le nombre d' ***époques*** qui indique le nombre de passages que nous allons effectuer sur l'ensemble de données d'apprentissage. Et la constante ***eta*** qui est le taux d'apprentissage dont nous multiplierons chaque mise à jour afin de rendre la procédure plus rapide en composant cette valeur ou si ***eta*** est trop élevé, nous pouvons le composer pour obtenir le résultat idéal (pour la plupart des applications). du perceptron je suggérerais une valeur d' ***eta*** de 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;
};
```

Maintenant, avec notre ensemble de classes. Il est temps d'écrire chacune des fonctions.

Nous allons partir du constructeur (**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
}
```

Comme vous pouvez voir ce que nous allons faire, ce sont des choses très simples. Passons donc à une autre fonction simple. La fonction de prédiction (***int prédire (vecteur X);***). Rappelez-vous que la fonction toutes les ***prédictions*** prend l'entrée réseau et renvoie une valeur de 1 si ***netInput*** est supérieur à 0 et -1 otherwhise.

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

Notez que nous avons utilisé une instruction en ligne pour rendre nos vies plus faciles. Voici comment fonctionne la déclaration en ligne:

condition? if_true: sinon

Jusqu'ici tout va bien. Passons à l'implémentation de la fonction ***netInput*** (***float netInput (vector X);***)

Le netInput effectue les opérations suivantes: ***multiplie le vecteur d'entrée par la transposition***

du vecteur de poids

x * wT

En d'autres termes, il multiplie chaque élément du vecteur d'entrée ***x*** par l'élément correspondant du vecteur de poids ***w*** et prend ensuite leur somme et ajoute le biais.

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

biais = 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;
}
```

Bon, nous sommes maintenant à peu près terminés. La dernière chose à faire est d'écrire la fonction d' ***ajustement*** qui modifie les poids.

```
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
        }
    }
}
```

Donc c'était essentiellement ça. Avec seulement 3 fonctions, nous avons maintenant une classe de perceptron qui peut être utilisée pour faire des prédictions!

Si vous souhaitez copier-coller le code et l'essayer. Voici la classe entière (j'ai ajouté des fonctionnalités supplémentaires telles que l'impression du vecteur de poids et les erreurs de chaque époque, ainsi que l'option permettant d'importer / exporter des poids).

Voici le code:

L'en-tête de 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;
};
```

Le fichier de classe .cpp avec les fonctions:

```
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;
}
```

Aussi, si vous voulez essayer un exemple, voici un exemple que j'ai fait:

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;
}
```

La manière dont j'ai importé le jeu de données iris n'est pas vraiment idéale, mais je voulais juste quelque chose qui fonctionne.

Les fichiers de données peuvent être trouvés ici.

J'espère que vous avez trouvé cela utile!

Quel est le biais

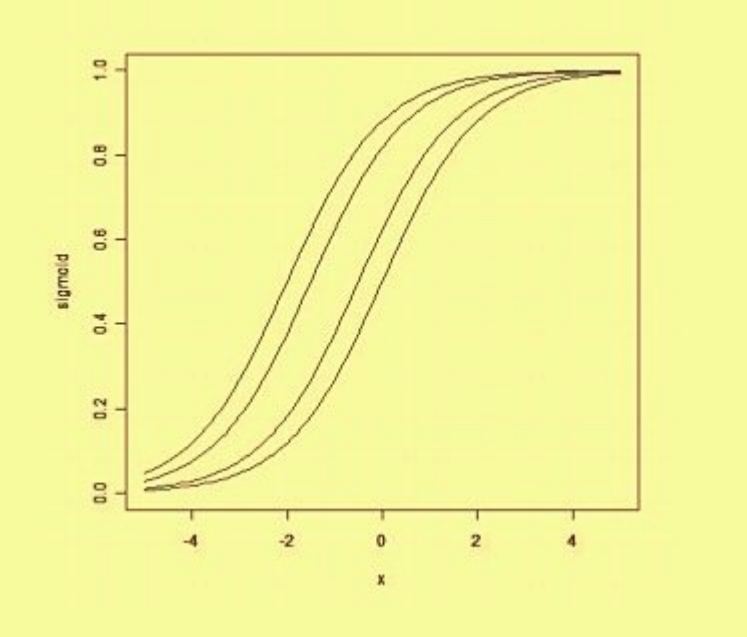
Quel est le biais

Un perceptron peut être vu comme une fonction mappant un vecteur d'entrée (valeur réelle) ***x*** à une valeur de sortie ***f (x)*** (valeur binaire):

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

où ***w*** est un vecteur de poids réels et ***b*** est notre valeur de *biais* . Le biais est une valeur qui décale la limite de décision de l'origine *(0,0)* et qui ne dépend d'aucune valeur d'entrée.

En considérant le biais de manière spatiale, le biais altère la position (mais pas l'orientation) de la limite de décision. Nous pouvons voir ci-dessous un exemple de la même courbe déplacée par le biais:



Lire Perceptron en ligne: https://riptutorial.com/fr/machine-learning/topic/6640/perceptron

Chapitre 10: Scikit Apprendre

Examples

Un problème simple de classification simple (XOR) utilisant l'algorithme du plus proche voisin

Considérez que vous voulez prédire la réponse correcte pour le problème populaire XOR. Vous savez ce qu'est XOR (par exemple [x0 x1] => y). par exemple [0 0] => 0, [0 1] => [1] et ...

```
#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]]))
```

Classification en scikit-learn

1. Arbres de décision mis en sac

L'ensachage donne de meilleurs résultats avec des algorithmes à forte variance. Un exemple populaire est celui des arbres de décision, souvent construits sans élagage.

Dans l'exemple ci-dessous, consultez un exemple d'utilisation de BaggingClassifier avec l'algorithme Classification et arbres de régression (DecisionTreeClassifier). Un total de 100 arbres sont créés.

Ensemble de données utilisé: ensemble de données sur le diabète des Indiens Pima

```
# 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())
```

En utilisant l'exemple, nous obtenons une estimation robuste de la précision du modèle.

```
0.770745044429
```

2. Forêt au hasard

La forêt aléatoire est une extension des arbres de décision ensachés.

Des échantillons du jeu de données d'apprentissage sont pris avec remplacement, mais les arbres sont construits de manière à réduire la corrélation entre les classificateurs individuels. Plus précisément, plutôt que de choisir avec avidité le meilleur point de partage dans la construction de l'arborescence, seul un sous-ensemble aléatoire d'entités est pris en compte pour chaque division.

Vous pouvez construire un modèle de forêt aléatoire pour la classification à l'aide de la classe RandomForestClassifier.

L'exemple ci-dessous fournit un exemple de forêt aléatoire pour la classification avec 100 arbres et des points de partage choisis parmi une sélection aléatoire de 3 entités.

```
# 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'exécution de l'exemple fournit une estimation moyenne de la précision de la classification.

```
0.770727956254
```

3. AdaBoost

AdaBoost était peut-être le premier algorithme de renforcement de l'ensemble réussi. Cela fonctionne généralement en pondérant les instances dans le jeu de données par la facilité ou la difficulté de les classer, ce qui permet à l'algorithme de prêter attention ou moins à leur attention dans la construction des modèles suivants.

Vous pouvez créer un modèle AdaBoost pour la classification à l'aide de la classe AdaBoostClassifier.

L'exemple ci-dessous illustre la construction de 30 arbres de décision en séquence à l'aide de l'algorithme 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'exécution de l'exemple fournit une estimation moyenne de la précision de la classification.

```
0.76045796309
```

4. Stochastic Gradient Boosting

Le Stochastic Gradient Boosting (aussi appelé Gradient Boosting Machines) est l'une des techniques d'ensemble les plus sophistiquées. C'est aussi une technique qui se révèle être l'une des meilleures techniques disponibles pour améliorer les performances via des ensembles.

Vous pouvez construire un modèle d'amélioration de dégradé pour la classification à l'aide de la classe GradientBoostingClassifier.

L’exemple ci-dessous montre l’amplification stochastique du gradient pour la classification avec 100 arbres.

```
# 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'exécution de l'exemple fournit une estimation moyenne de la précision de la classification.

```
0.764285714286
```

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

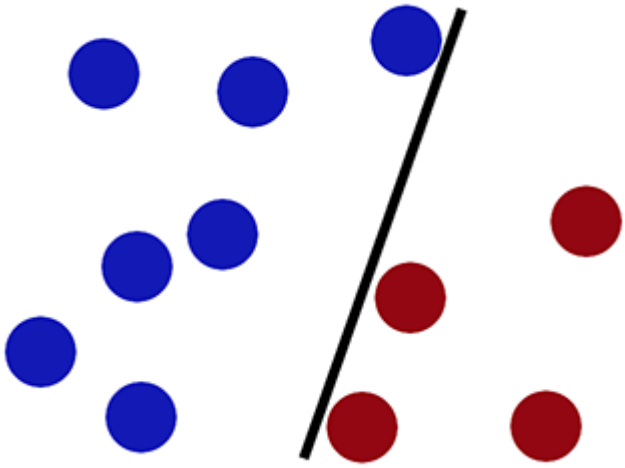
Lire Scikit Apprendre en ligne: https://riptutorial.com/fr/machine-learning/topic/6156/scikit-apprendre

Chapitre 11: SVM

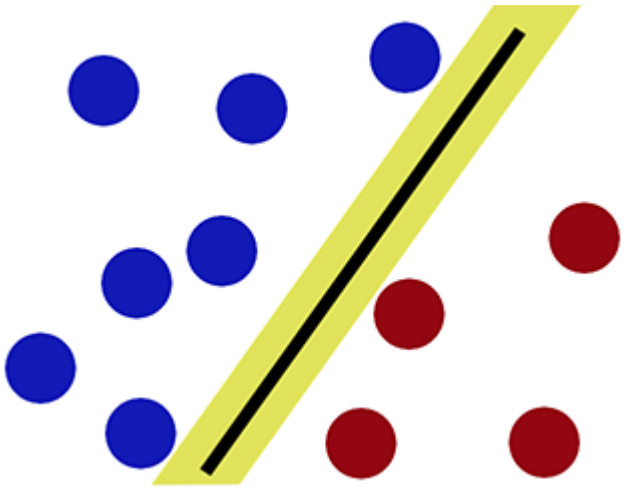
Examples

Différence entre la régression logistique et la SVM

Limite de décision lorsque nous classifions en utilisant **la régression logistique** -



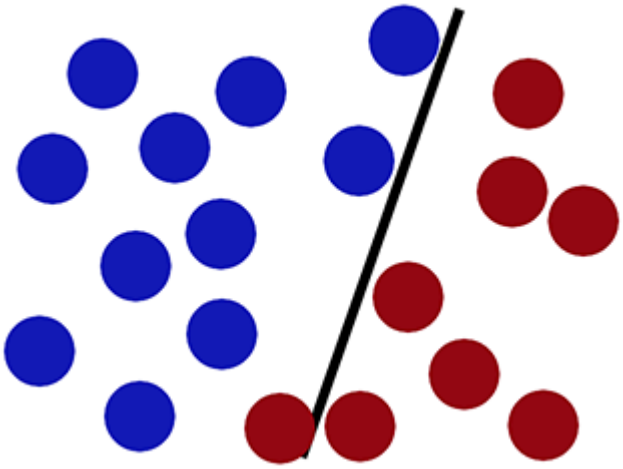
Limite de décision lorsque nous classifions à l'aide de **SVM** -



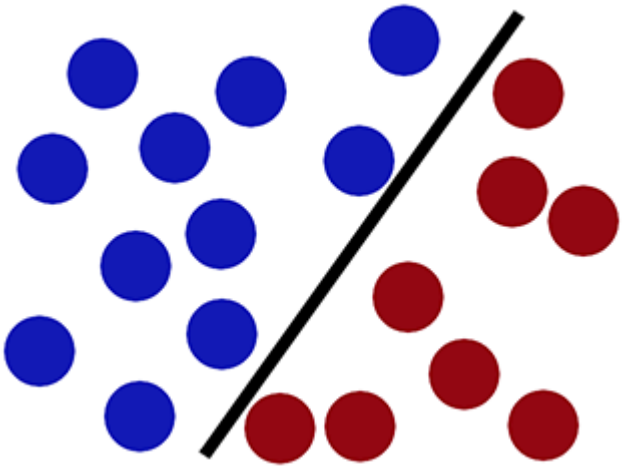
Comme on peut l'observer, SVM essaie de maintenir un «écart» de chaque côté de la limite de décision. Cela s'avère utile lorsque nous rencontrons de nouvelles données.

Avec de nouvelles données

La régression logistique est **médiocre** (le nouveau cercle rouge est classé en bleu) -



Alors que **SVM** peut le classer correctement (le nouveau cercle rouge est classé correctement sur le côté rouge) -



Implémenter le classificateur SVM en utilisant 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()
```

Lire SVM en ligne: https://riptutorial.com/fr/machine-learning/topic/7126/svm

Chapitre 12: Traitement du langage naturel

Introduction

La PNL est un moyen pour les ordinateurs d'analyser, de comprendre et de tirer un sens du langage humain d'une manière intelligente et utile. En utilisant le protocole NLP, les développeurs peuvent organiser et structurer leurs connaissances pour effectuer des tâches telles que la synthèse automatique, la traduction, la reconnaissance d'entités nommées, l'extraction de relations, l'analyse de sentiments, la reconnaissance vocale et la segmentation.

Examples

Correspondance du texte ou similarité

> L'un des domaines importants de la PNL est la correspondance des objets texte pour trouver des similitudes. Les applications importantes de la correspondance de texte incluent la correction automatique de l'orthographe, la déduplication des données et l'analyse du génome, etc. Un certain nombre de techniques de correspondance de texte sont disponibles en fonction des besoins. Alors laisse faire; **Levenshtein Distance**

La distance Levenshtein entre deux chaînes est définie comme le nombre minimum de modifications nécessaires pour transformer une chaîne en une autre, les opérations d'édition autorisées étant l'insertion, la suppression ou la substitution d'un seul caractère.

Voici l'implémentation pour des calculs de mémoire efficaces.

```
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"))
```

Lire Traitement du langage naturel en ligne: https://riptutorial.com/fr/machine-learning/topic/10734/traitement-du-langage-naturel

Chapitre 13: Types d'apprentissage

Examples

Enseignement supervisé

La machine apprend à prédire une sortie lorsqu'elle reçoit une entrée.

Chaque cas d'entraînement comprend une entrée et une sortie cible.

Régression

La sortie cible prend des valeurs continues.

- Prédire le prix d'un stock
- Prédire un prix de maison

Classification

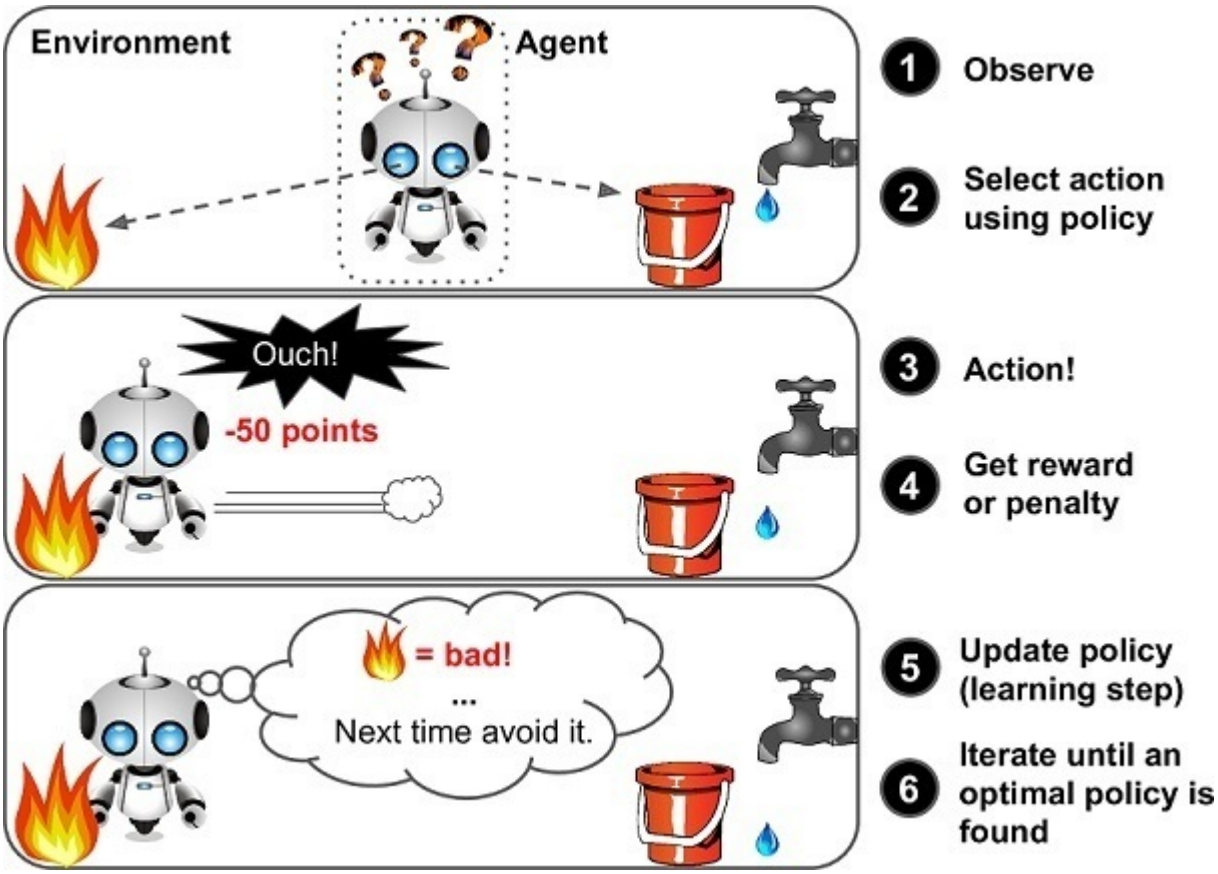
La sortie cible est une étiquette de classe.

- Quel genre de fruit l'apport est
- Quelle langue un mot est

Apprentissage par renforcement

La machine doit automatiquement déterminer le comportement idéal pour optimiser ses performances.

Par exemple:



En utilisant l'apprentissage par renforcement, vous pouvez également créer un programme informatique capable de compléter un niveau Mario (MarI / O-Machine Learning pour les jeux vidéo).

Apprentissage non supervisé

L'apprentissage non supervisé nous permet d'aborder les problèmes avec peu ou pas d'idée de nos résultats. Nous pouvons **dériver une structure à partir de données** où nous ne connaissons pas nécessairement l'effet des variables.

Le type d'apprentissage non supervisé le plus courant est l' **analyse en grappes ou le regroupement** . C'est la tâche de regrouper un ensemble d'objets de telle manière que les objets d'un même groupe (cluster) soient plus proches les uns des autres que ceux des autres groupes.

Il y a aussi l'apprentissage non supervisé sans clustering. Un exemple en est l'identification des voix individuelles et de la musique à partir d'un maillage de sons. C'est ce qu'on appelle "l'algorithme de cocktail".

Lire Types d'apprentissage en ligne: https://riptutorial.com/fr/machine-learning/topic/10912/types-d-apprentissage

Chapitre 14: Une introduction à la classification: générer plusieurs modèles avec Weka

Introduction

Ce tutoriel vous montrera comment utiliser Weka en code JAVA, charger un fichier de données, former des classificateurs et expliquer certains concepts importants de l'apprentissage automatique.

Weka est une boîte à outils pour l'apprentissage automatique. Il comprend une bibliothèque de techniques d'apprentissage et de visualisation et comprend une interface utilisateur conviviale.

Ce tutoriel comprend des exemples écrits en JAVA et inclut des éléments visuels générés avec l'interface graphique. Je suggère d'utiliser l'interface graphique pour examiner les données et le code JAVA pour des expériences structurées.

Examples

Mise en route: Chargement d'un jeu de données à partir d'un fichier

Le jeu de données de fleurs Iris est un ensemble de données largement utilisé à des fins de démonstration. Nous allons le charger, l'inspecter et le modifier légèrement pour une utilisation ultérieure.

```
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();
    }
}
```

Former le premier classificateur: définir une référence avec ZeroR

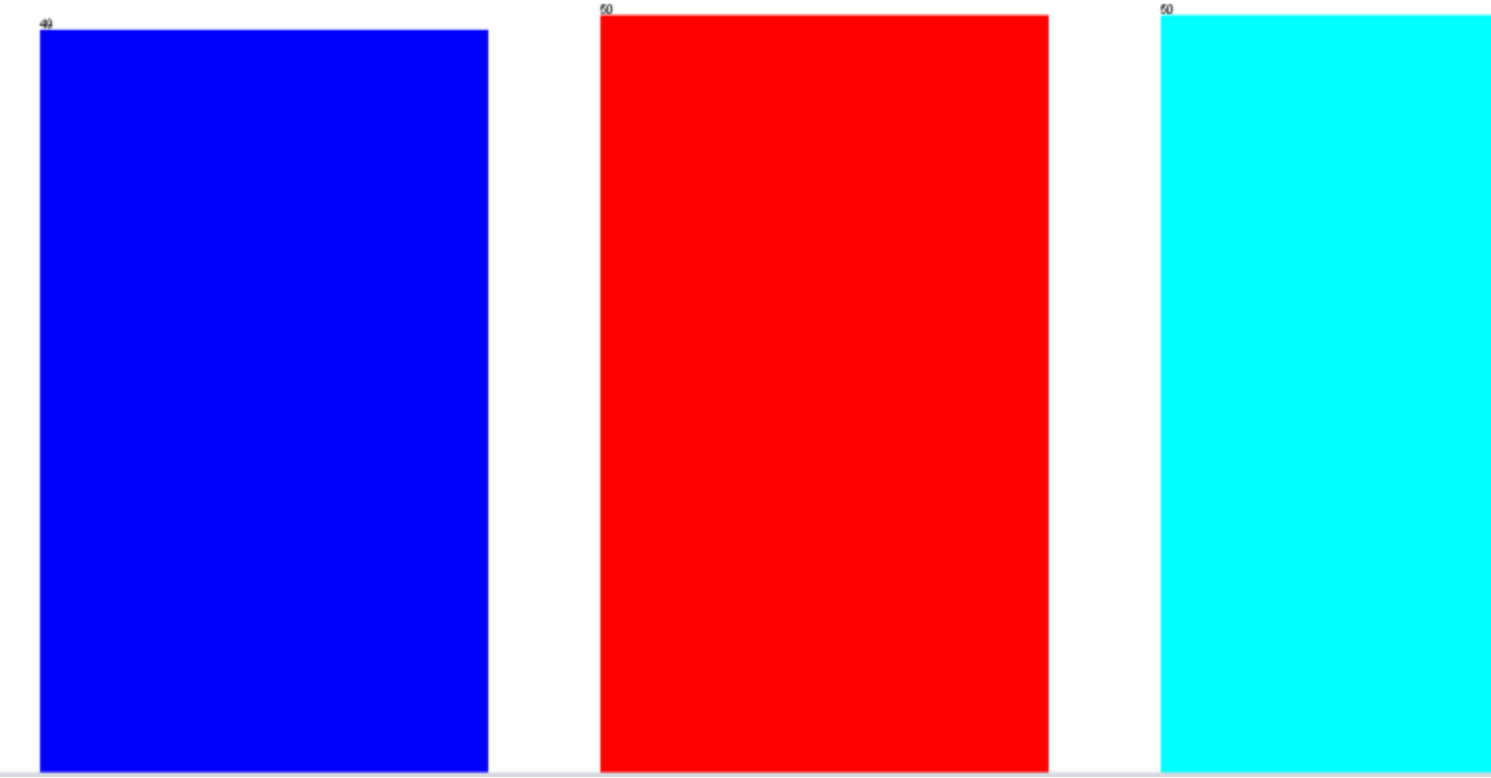
ZeroR est un classificateur simple. Il ne fonctionne pas par instance mais opère sur la distribution générale des classes. Il sélectionne la classe avec la plus grande probabilité a priori. Ce n'est pas un bon classificateur dans le sens où il n'utilise aucune information dans le candidat, mais il est

souvent utilisé comme base. **Remarque: d'autres lignes de base peuvent être utilisées, telles que: Classificateurs standard ou règles artisanales**

```
// 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());
```

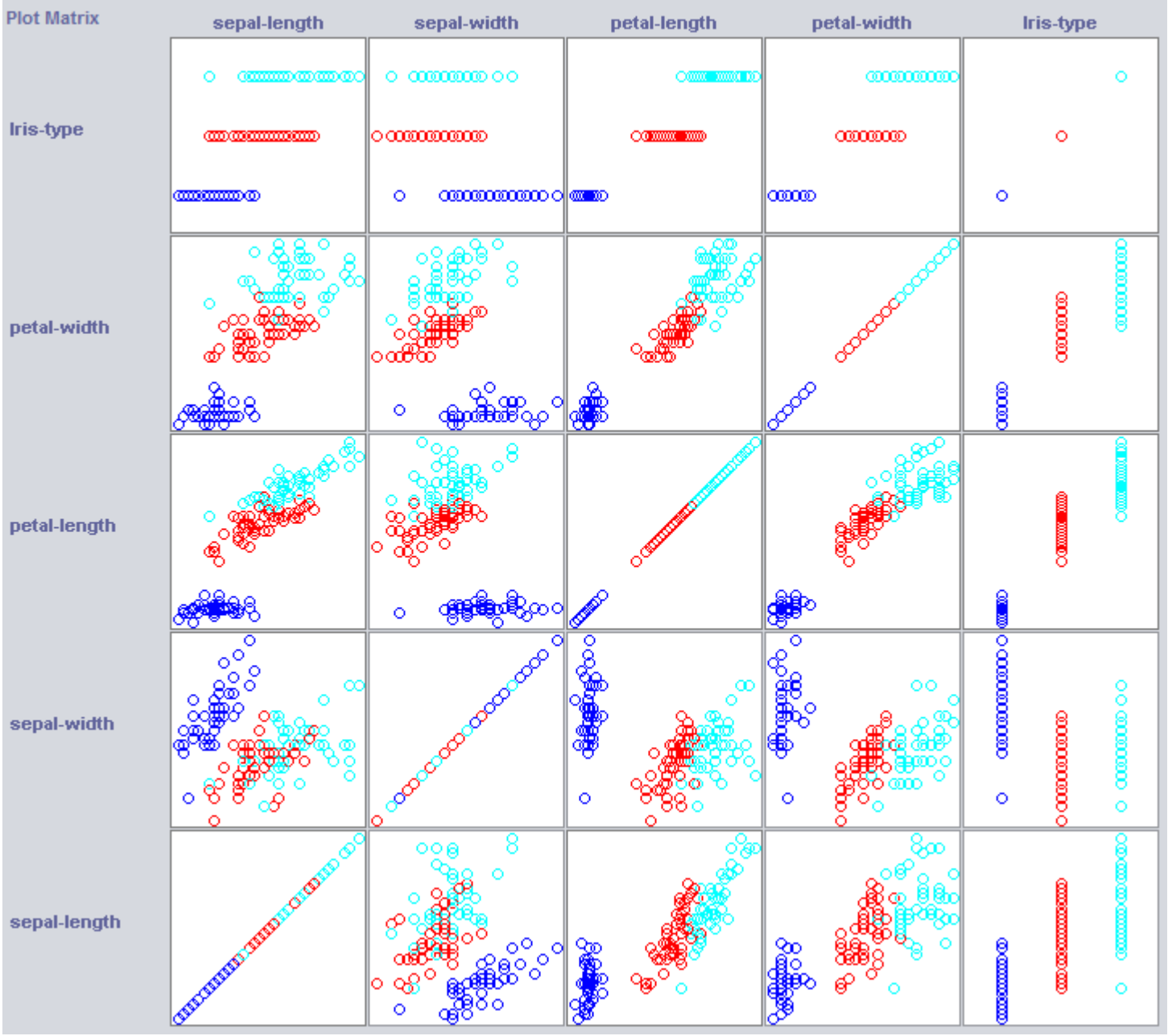
La plus grande classe de l'ensemble vous donne un taux correct de 34%. (50 sur 149)



Remarque: le ZeroR effectue environ 30%. C'est parce que nous avons divisé au hasard dans un train et un ensemble de test. Le plus grand ensemble de la rame sera donc le plus petit de l'ensemble de test. Fabriquer un bon ensemble test / train peut valoir la peine

Avoir une idée des données. Entraînement Naive Bayes et kNN

Afin de construire un bon classificateur, nous devrons souvent avoir une idée de la structure des données dans l'espace des objets. Weka propose un module de visualisation pouvant aider.



Certaines dimensions séparent déjà assez bien les classes. La largeur des pétales ordonne le concept de manière très nette, par exemple par rapport à la largeur des pétales.

La formation de classificateurs simples peut en révéler beaucoup sur la structure des données. J'aime généralement utiliser Nearest Neighbor et Naive Bayes à cette fin. Naive Bayes assume son indépendance, elle fonctionne bien, ce qui indique que les dimensions sur elles-mêmes détiennent des informations. k-Nearest-Neighbor fonctionne en attribuant la classe des k instances les plus proches (connues) dans l'espace des objets. Il est souvent utilisé pour examiner la dépendance géographique locale, nous l'utiliserons pour examiner si notre concept est défini localement dans l'espace des entités.

```
//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 fonctionne beaucoup mieux que notre base de référence fraîchement établie, indiquant que des fonctionnalités indépendantes contiennent des informations (souvenez-vous de la largeur des pétales?).

1NN fonctionne bien aussi (en fait un peu mieux dans ce cas), indiquant que certaines de nos informations sont locales. La meilleure performance pourrait indiquer que certains effets de second ordre contiennent également des informations *(If x et y que la classe z)* .

Rassembler: Former un arbre

Les arbres peuvent créer des modèles qui fonctionnent sur des fonctions indépendantes et sur des effets de second ordre. Ils pourraient donc être de bons candidats pour ce domaine. Les arbres sont des règles qui sont regroupées, une règle divise les instances qui parviennent à une règle dans des sous-groupes, qui passent aux règles de la règle.

Les apprenants arborescents génèrent des règles, les enchaînent et arrêtent de construire des arborescences lorsqu'ils estiment que les règles deviennent trop spécifiques, pour éviter de les dépasser. *Overfitting signifie construire un modèle trop complexe pour le concept que nous recherchons. Les modèles surdimensionnés fonctionnent bien sur les données du train, mais mal sur les nouvelles données*

Nous utilisons J48, une implémentation JAVA de C4.5 un algorithme populaire.

```
//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());
```

L'apprenant en formation formé avec la plus grande confiance génère les règles les plus spécifiques et offre les meilleures performances sur l'ensemble de test, apparemment la spécificité est justifiée.

```
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)
```

Remarque: les deux apprenants commencent par une règle sur la largeur des pétales. Rappelez-vous comment nous avons remarqué cette dimension dans la visualisation?

Lire Une introduction à la classification: générer plusieurs modèles avec Weka en ligne: https://riptutorial.com/fr/machine-learning/topic/8649/une-introduction-a-la-classification--generer-plusieurs-modeles-avec-weka

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec l'apprentissage automatique	Aquib Javed Khan, Community, Dr. Cool, Drew, John Syrinek, Nikos Tavoularis, Piyush, Semih Korkmaz
2	Apprentissage automatique avec Java	Arnab Biswas, Patel Sunil
3	Démarrer avec Machine Learning en utilisant Apache spark MLib	Malav
4	Enseignement supervisé	draco_alpine, L.V.Rao, Martin W, Masoud, plumSemPy, quintumnia, shadowfox
5	L'apprentissage automatique et sa classification	Sirajus Salayhin
6	L'apprentissage en profondeur	Martin W, Minhas Kamal, orbit, Thomas Pinetz
7	Les réseaux de neurones	cldseed, dontloo, Dr. Cool, Franck Dernoncourt
8	Mesures d'évaluation	DJanssens, Franck Dernoncourt, Sayali Sonawane, ScientiaEtVeritas
9	Perceptron	granmirupa, Martin W, Panos
10	Scikit Apprendre	Masoud, RamenChef, Sayali Sonawane
11	SVM	Alekh Karkada Ashok
12	Traitement du langage naturel	quintumnia
13	Types d'apprentissage	Martin W
14	Une introduction à la classification:	S van Balen

	générer plusieurs modèles avec Weka