EBook Gratis

APRENDIZAJE machine-learning

Free unaffiliated eBook created from
Stack Overflow contributors.

#machine-learning

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con el aprendizaje automático.

Observaciones

El aprendizaje automático es la ciencia (y el arte) de la programación de computadoras para que puedan aprender de los datos.

Una definición más formal:

Es el campo de estudio que le da a las computadoras la capacidad de aprender sin ser programadas explícitamente. Arthur Samuel, 1959

Una definición más orientada a la ingeniería:

Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna tarea T y alguna medida de desempeño P, si su desempeño en T, medido por P, mejora con la experiencia E. Tom Mitchell, 1997

Fuente: “Aprendizaje automático con Scikit-Learn y TensorFlow por Aurélien Géron (O'Reilly). Derechos de autor 2017 Aurélien Géron, 978-1-491-96229-9. "

El aprendizaje automático (MA) es un campo de la informática que surgió de la investigación en inteligencia artificial. La fuerza del aprendizaje automático sobre otras formas de análisis radica en su capacidad para descubrir ideas ocultas y predecir resultados de futuros, insumos ocultos (generalización). A diferencia de los algoritmos iterativos donde las operaciones se declaran explícitamente, los algoritmos de aprendizaje automático toman prestados conceptos de la teoría de la probabilidad para seleccionar, evaluar y mejorar los modelos estadísticos.

Examples

Instalación o configuración utilizando Python

1) scikit aprender

scikit-learn es un módulo de Python para aprendizaje automático construido sobre SciPy y distribuido bajo la licencia BSD de 3 cláusulas. Cuenta con varios algoritmos de clasificación, regresión y agrupamiento, que incluyen máquinas de vectores de soporte, bosques aleatorios, aumento de gradiente, k-means y DBSCAN, y está diseñado para interactuar con las bibliotecas numéricas y científicas de Python, NumPy y SciPy.

La versión estable actual de scikit-learn requiere :

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

- Ciencia ficción (> = 0.9).

Durante la mayor instalación `pip` gestor de paquetes Python puede instalar Python y todas sus dependencias:

```
pip install scikit-learn
```

Sin embargo, para los sistemas Linux, se recomienda utilizar el `conda` paquetes `conda` para evitar posibles procesos de compilación.

```
conda install scikit-learn
```

Para verificar que tiene `scikit-learn` , ejecute en shell:

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

Instalación de Windows y Mac OSX:

Canopy y Anaconda *entregan* una versión reciente de *scikit-learn* , además de un amplio conjunto de bibliotecas científicas de Python para Windows, Mac OSX (también relevante para Linux).

Repo de código fuente oficial: https://github.com/scikit-learn/scikit-learn

---

2) Plataforma Numenta para Computación Inteligente.

La Plataforma Numenta para Computación Inteligente (NuPIC) es una plataforma de inteligencia de máquina que implementa los algoritmos de aprendizaje HTM. HTM es una teoría computacional detallada del neocórtex. En el núcleo de HTM se encuentran los algoritmos de aprendizaje continuo basados en el tiempo que almacenan y recuperan patrones espaciales y temporales. NuPIC es adecuado para una variedad de problemas, en particular la detección de anomalías y la predicción de fuentes de datos de transmisión.

Los binarios de NuPIC están disponibles para:

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

Se requieren las siguientes dependencias para instalar NuPIC en todos los sistemas operativos.

- Python 2.7
- pip> = 8.1.2
- setuptools> = 25.2.0
- rueda> = 0.29.0
- adormecido
- Compilador de C ++ 11 como gcc (4.8+) o clang

Requisitos adicionales de OS X:

- Herramientas de línea de comandos de Xcode

Ejecute lo siguiente para instalar NuPIC:

```
pip install nupic
```

Repo de código fuente oficial: https://github.com/numenta/nupic

---

3) nilearn

Nilearn es un módulo de Python para el aprendizaje estadístico rápido y fácil de los datos de NeuroImaging. Aprovecha la caja de herramientas Python de scikit-learn para estadísticas multivariadas con aplicaciones tales como modelado predictivo, clasificación, decodificación o análisis de conectividad.

Las dependencias requeridas para utilizar el software son:

- Python> = 2.6,
- herramientas de configuración
- Numpy> = 1.6.1
- Ciencia ficción> = 0.9
- Scikit-learn> = 0.14.1
- Nibabel> = 1.1.0

Si está utilizando las funcionalidades de trazado de nilearn o ejecutando los ejemplos, se requiere matplotlib> = 1.1.1.

Si desea ejecutar las pruebas, necesita nose> = 1.2.1 y cobertura> = 3.6.

Primero asegúrese de haber instalado todas las dependencias enumeradas anteriormente. Luego, puede instalar nilearn ejecutando el siguiente comando en un símbolo del sistema:

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

Repo de código fuente oficial: https://github.com/nilearn/nilearn/

4) Usando Anaconda

Muchas bibliotecas científicas de Python están disponibles en Anaconda. Puedes obtener los archivos de instalación desde aquí . Por un lado, al usar Anaconda, no tiene que instalar ni configurar muchos paquetes, tiene licencia BSD y tiene un proceso de instalación trivial, disponible para Python 3 y Python 2, mientras que, por otro lado, le brinda menos flexibilidad. Como ejemplo, algunos paquetes de Python de aprendizaje profundo del estado de la técnica podrían usar una versión diferente de numpy que Anaconda instalada. Sin embargo, este inconveniente puede tratarse utilizando otra instalación de Python por separado (por ejemplo, en Linux y MAC, su predeterminada).

La configuración de Anaconda le solicita la selección de la ubicación de la instalación y también la opción de adición PATH. Si agrega Anaconda a su RUTA, se espera que su sistema operativo encuentre Anaconda Python como predeterminado. Por lo tanto, las modificaciones y futuras instalaciones estarán disponibles solo para esta versión de Python.

Para dejarlo claro, después de la instalación de Anaconda y lo agrega a PATH, use Ubuntu 14.04 a través de la terminal si escribe

```
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 es su Python predeterminado, puede comenzar a disfrutar de muchas bibliotecas de inmediato. Sin embargo, si quieres usar tu antiguo 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 resumen, Anaconda es una de las maneras más rápidas de comenzar el aprendizaje automático y el análisis de datos con Python.

Instalación o configuración utilizando R Language

Los paquetes son colecciones de funciones R, datos y código compilado en un formato bien definido. Los repositorios públicos (y privados) se utilizan para alojar colecciones de paquetes R. La mayor colección de paquetes de R está disponible en CRAN. Algunos de los paquetes de aprendizaje automático de R más populares son, entre otros, los siguientes:

1) rpart

Descripción: Partición recursiva para árboles de clasificación, regresión y supervivencia. Una implementación de la mayor parte de la funcionalidad del libro de 1984 de Breiman, Friedman, Olshen y Stone.

Se puede instalar desde CRAN usando el siguiente código:

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

Cargue el paquete:

```
library(rpart)
```

Fuente oficial: https://cran.r-project.org/web/packages/rpart/index.html

---

2) e1071

Descripción: Funciones para el análisis de clases latentes, transformada de Fourier de tiempo corto, agrupamiento difuso, máquinas de vectores de soporte, cálculo de la ruta más corta, agrupamiento empaquetado, clasificador de Bayes ingenuo, etc.

Instalación desde CRAN:

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

Cargando el paquete:

```
library(e1071)
```

Fuente oficial: https://cran.r-project.org/web/packages/e1071/index.html

---

3) randomForest

Descripción: Clasificación y regresión basada en un bosque de árboles usando entradas aleatorias.

Instalación desde CRAN:

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

Cargando el paquete:

```
library(randomForest)
```

Fuente oficial: https://cran.r-project.org/web/packages/randomForest/index.html

---

4) caret

Descripción: Funciones misceláneas para entrenamiento y trazado de modelos de clasificación y regresión.

Instalación desde CRAN:

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

Cargando el paquete:

```
library(caret)
```

Fuente oficial: https://cran.r-project.org/web/packages/caret/index.html

Lea Empezando con el aprendizaje automático. en línea: https://riptutorial.com/es/machine-learning/topic/1151/empezando-con-el-aprendizaje-automatico-

Capítulo 2: Aprendizaje automático utilizando Java

Examples

lista de herramientas

```
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.
```

Aprendizaje automático de propósito general

```
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.
```

Reconocimiento de voz

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

Análisis de datos / Visualización de datos

```
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
```

Aprendizaje profundo

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

Lea Aprendizaje automático utilizando Java en línea: https://riptutorial.com/es/machine-learning/topic/6175/aprendizaje-automatico-utilizando-java

Capítulo 3: Aprendizaje profundo

Introducción

El aprendizaje profundo es un subcampo del aprendizaje automático en el que se utilizan redes neuronales artificiales de múltiples capas para fines de aprendizaje. Deep Learning ha encontrado muchas implementaciones excelentes, por ejemplo, Reconocimiento de voz, Subtítulos en Youtube, Recomendaciones de Amazon, etc. Para información adicional hay un tema dedicado al aprendizaje profundo .

Examples

Breve breve de Aprendizaje profundo.

Para entrenar una red neuronal, primero debemos diseñar una idea buena y eficiente. Hay tres tipos de tareas de aprendizaje.

- Aprendizaje supervisado
- Aprendizaje reforzado
- Aprendizaje sin supervisión

En este momento actual, el aprendizaje no supervisado es muy popular. El aprendizaje no supervisado es una tarea de aprendizaje profundo de inferir una función para describir una estructura oculta a partir de datos "no etiquetados" (no se incluye una clasificación o categorización en las observaciones).

Dado que los ejemplos proporcionados al aprendiz no están marcados, no se evalúa la precisión de la estructura que genera el algoritmo relevante, que es una forma de distinguir el aprendizaje no supervisado del aprendizaje supervisado y el aprendizaje por refuerzo.

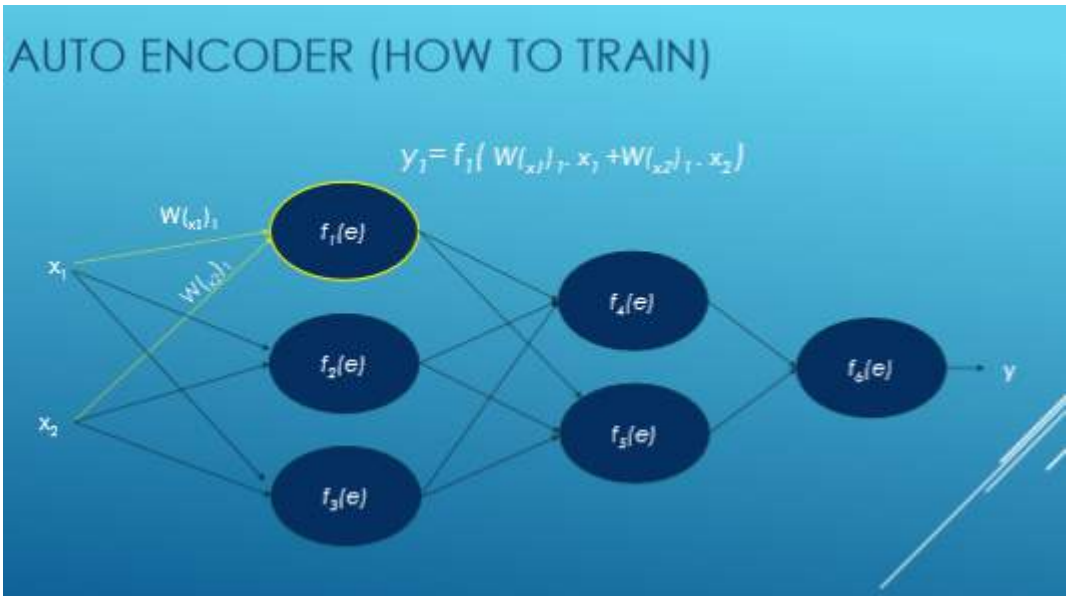
Hay tres tipos de aprendizaje no supervisado.

- Máquinas de Boltzmann restringidas
- Modelo de codificación escasa
- Autoencoders que describiré en detalle de autoencoder.

El objetivo de un autocodificador es aprender una representación (codificación) para un conjunto de datos, generalmente con el propósito de reducir la dimensionalidad.

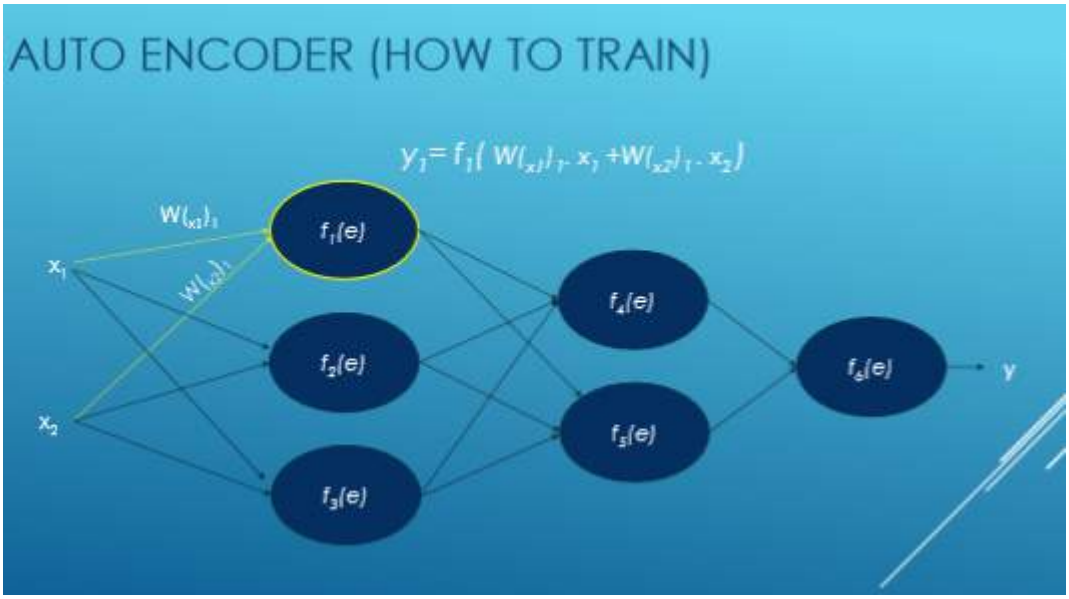
La forma más simple de un autocodificador es un feedforward, que tiene una capa de entrada, una capa de salida y una o más capas ocultas que los conectan. Pero con la capa de salida que tiene el mismo número de nodos que la capa de entrada, y con el propósito de reconstruir sus propias entradas, es por eso que se llama aprendizaje no supervisado.

Ahora voy a tratar de dar un ejemplo de red neuronal de entrenamiento.

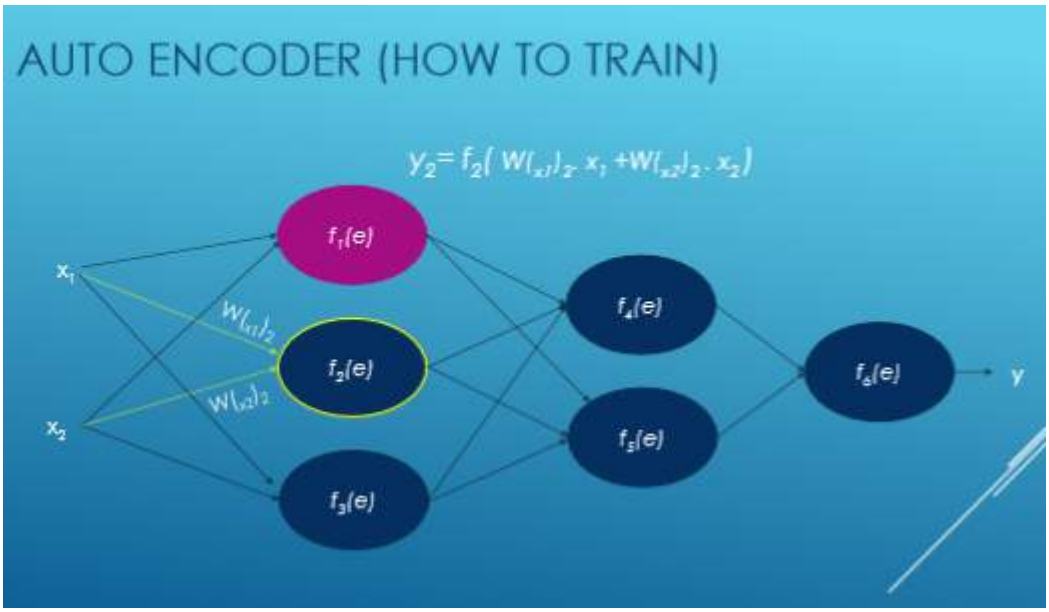


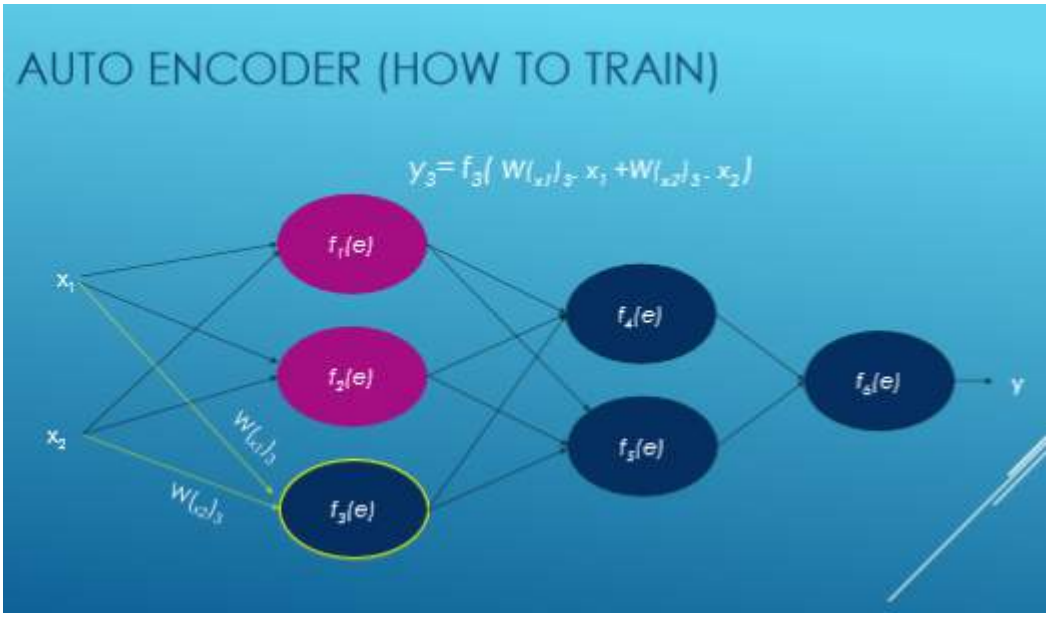
Aquí Xi es entrada, W es peso, f (e) es función de activación y y es salida.

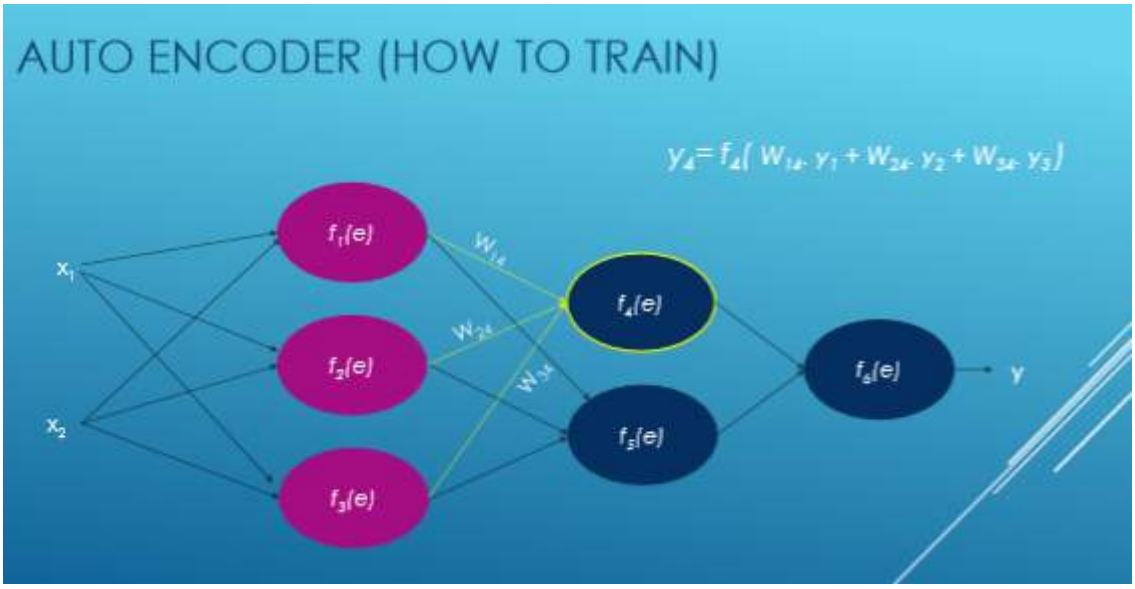
Ahora vemos paso a paso el flujo de la red neuronal de entrenamiento basada en el autoencoder.

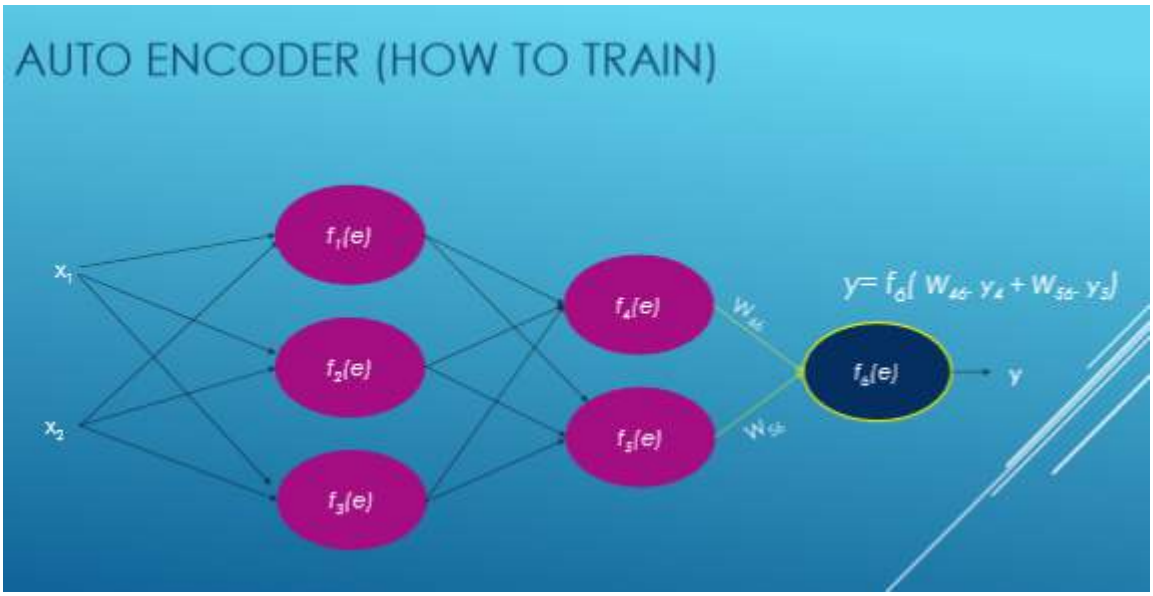


Calculamos el valor de cada función de activación con esta ecuación: y = WiXi. Primero que nada, elegimos números al azar para los pesos y luego intentamos ajustarlos.

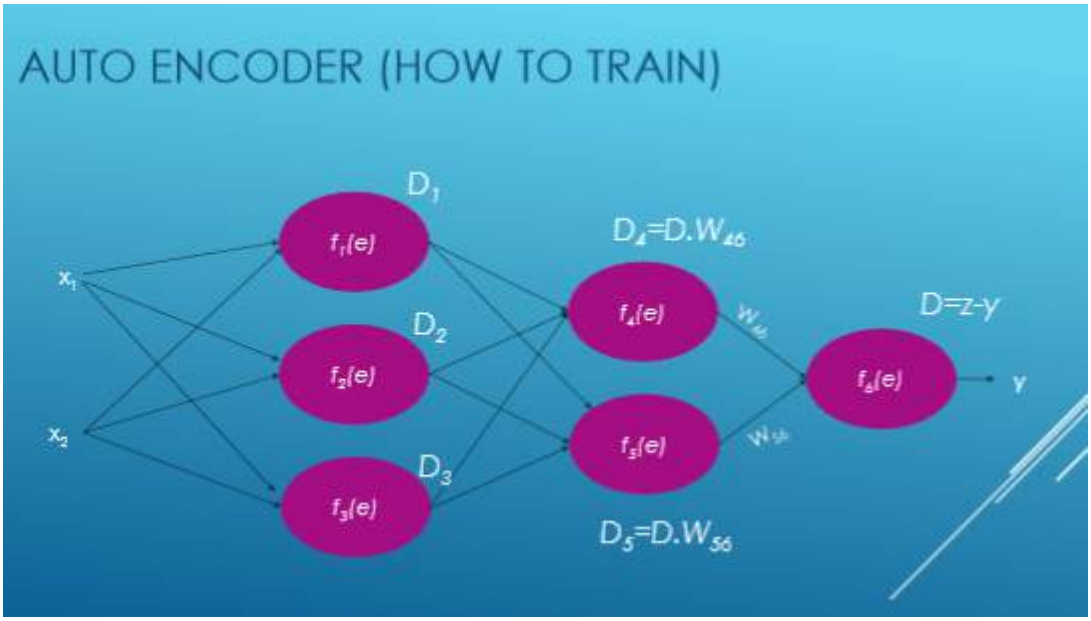

AUTO ENCODER (HOW TO TRAIN)
$y_2 = f_2(W_{(x1)2} \cdot x_1 + W_{(x2)2} \cdot x_2)$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
y


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

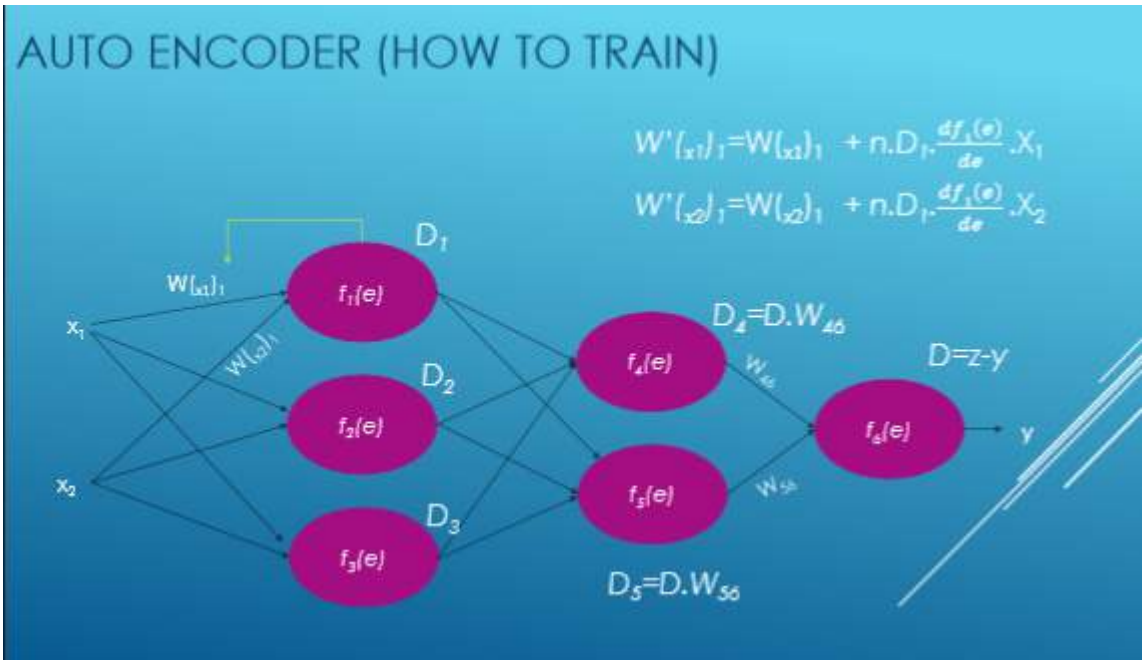

AUTO ENCODER (HOW TO TRAIN)
$y_4 = f_4(W_{14} \cdot y_1 + W_{24} \cdot y_2 + W_{34} \cdot y_3)$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
y



Ahora, calculamos la desviación de nuestra salida deseada, que es y = zy, y calculamos las desviaciones de cada función de activación.



Luego ajustamos nuestro nuevo peso de cada conexiones.



Lea Aprendizaje profundo en línea: https://riptutorial.com/es/machine-learning/topic/7442/aprendizaje-profundo

Capítulo 4: Aprendizaje supervisado

Examples

Clasificación

Imagina que un sistema quiere detectar **manzanas** y **naranjas** en una cesta de frutas. El sistema puede recoger una fruta, extraer alguna propiedad de ella (por ejemplo, el peso de esa fruta).

Supongamos que el sistema tiene un maestro! que enseña al sistema qué objetos son **manzanas** y cuáles son **naranjas** . Este es un ejemplo de un problema de **clasificación supervisada** . Es supervisado porque hemos etiquetado ejemplos. Es una clasificación porque la salida es una predicción de a qué clase pertenece nuestro objeto también.

En este ejemplo consideramos 3 características (propiedades / variables explicativas):

1. es el peso de la fruta seleccionada mayor que .5gram
2. es el tamaño mayor de 10 cm
3. el color es rojo

(0 significa No, y 1 significa Sí)

Entonces, para representar una manzana / naranja, tenemos una serie (llamada vector) de 3 propiedades (a menudo llamada vector característica)

(por ejemplo, [0,0,1] significa que el peso de esta fruta no es mayor que .5 gramos, y su tamaño no es mayor de 10 cm y el color es rojo)

Entonces, escogemos 10 frutas al azar y medimos sus propiedades. El maestro (humano) luego etiqueta cada fruta manualmente como manzana => **[1]** o naranja => **[2]** .

Ej.) El profesor selecciona una fruta que es manzana. La representación de esta manzana por sistema podría ser algo como esto: **[1, 1, 1] => [1]** , Esto significa que, esta fruta tiene **1.weight mayor que .5gram** , **2.size mayor que 10cm** y **3. el color de esta fruta es rojo** y finalmente es una **manzana** (=> [1])

Entonces, para todas las 10 frutas, el maestro etiqueta cada fruta como manzana [=> 1] o naranja [=> 2] y el sistema encuentra sus propiedades. Como supones, tenemos una serie de vectores (que se llama matriz) para representar 10 frutas enteras.

Clasificación de frutas

En este ejemplo, un modelo aprenderá a clasificar frutas dadas ciertas características, utilizando las *Etiquetas* para entrenamiento.

Peso	Color	Etiqueta
0.5	verde	manzana
0.6	púrpura	ciruela
3	verde	sandía
0.1	rojo	Cereza
0.5	rojo	manzana

Aquí el modelo tomará el *peso* y el *color* como características para predecir la etiqueta. Por ejemplo, [0.15, 'rojo'] debería resultar en una predicción 'cherry'.

Introducción al aprendizaje supervisado

Hay bastantes situaciones en las que uno tiene enormes cantidades de datos y el uso que tiene para clasificar un objeto en una de varias clases conocidas. Considere las siguientes situaciones:

Banca: cuando un banco recibe una solicitud de un cliente para una tarjeta bancaria, el banco debe decidir si emite o no la tarjeta bancaria, según las características de los clientes que ya disfrutan de las tarjetas para las cuales se conoce el historial de crédito.

Médico: Uno podría estar interesado en desarrollar un sistema médico que diagnostique a un paciente si tiene o no una enfermedad en particular, según los síntomas observados y las pruebas médicas realizadas en ese paciente.

Finanzas: una empresa de consultoría financiera desea predecir la tendencia del precio de una acción que puede clasificarse en una tendencia ascendente, descendente o nula basada en varias características técnicas que rigen el movimiento de precios.

Expresión génica: un científico que analiza los datos de la expresión génica quisiera identificar los genes más relevantes y los factores de riesgo involucrados en el cáncer de mama, a fin de separar a los pacientes sanos de los pacientes con cáncer de mama.

En todos los ejemplos anteriores, un objeto se clasifica en una de varias clases *conocidas* , en función de las mediciones realizadas en una serie de características, que puede pensar que discriminan los objetos de diferentes clases. Estas variables se denominan variables *predictoras* y la etiqueta de clase se denomina variable *dependiente* . Tenga en cuenta que, en todos los ejemplos anteriores, la variable dependiente es *categórica* .

Para desarrollar un modelo para el problema de clasificación, requerimos, para cada objeto, datos sobre un conjunto de características prescritas junto con las etiquetas de clase, a las que pertenecen los objetos. El conjunto de datos se divide en dos conjuntos en una proporción prescrita. El mayor de estos conjuntos de datos se denomina conjunto de *datos de entrenamiento* y el otro, conjunto de *datos de prueba* . El conjunto de datos de entrenamiento se utiliza en el desarrollo del modelo. A medida que el modelo se desarrolla utilizando observaciones cuyas etiquetas de clase son conocidas, estos modelos se conocen como modelos de *aprendizaje*

supervisado .

Después de desarrollar el modelo, el modelo se evaluará por su rendimiento utilizando el conjunto de datos de prueba. El objetivo de un modelo de clasificación es tener una probabilidad mínima de errores de clasificación en las observaciones invisibles. Las observaciones que no se usan en el desarrollo del modelo se conocen como observaciones invisibles.

La inducción de árboles de decisión es una de las técnicas de construcción de modelos de clasificación. El modelo de árbol de decisión creado para la variable dependiente categórica se denomina *Árbol de clasificación* . La variable dependiente podría ser numérica en ciertos problemas. El modelo de árbol de decisión desarrollado para las variables dependientes numéricas se llama *árbol de regresión* .

Regresión lineal

El aprendizaje supervisado consiste en un objetivo o variable de resultado (o variable dependiente) que se debe predecir a partir de un conjunto dado de predictores (variables independientes). Usando este conjunto de variables, generamos una función que mapea las entradas a las salidas deseadas. El proceso de capacitación continúa hasta que el modelo alcanza el nivel deseado de precisión en los datos de capacitación.

> Por lo tanto, hay muchos ejemplos de algoritmos de aprendizaje supervisado, por lo que en este caso me gustaría centrarme en la **regresión lineal**

Regresión lineal Se utiliza para estimar valores reales (costo de casas, número de llamadas, ventas totales, etc.) en función de variables continuas. Aquí, establecemos la relación entre variables independientes y dependientes ajustando una mejor línea. Esta línea de mejor ajuste se conoce como línea de regresión y se representa mediante una ecuación lineal Y = a * X + b.

La mejor manera de entender la regresión lineal es revivir esta experiencia de la infancia. Digamos que le pide a un niño de quinto grado que organice a las personas en su clase aumentando el orden de peso, ¡sin preguntarles su peso! ¿Qué crees que hará el niño? Es probable que él (ella) mire (analice visualmente) la altura y la formación de las personas y las organice utilizando una combinación de estos parámetros visibles.

¡Esto es regresión lineal en la vida real! El niño realmente ha descubierto que la altura y la estructura se correlacionarían con el peso por una relación, que se parece a la ecuación anterior.

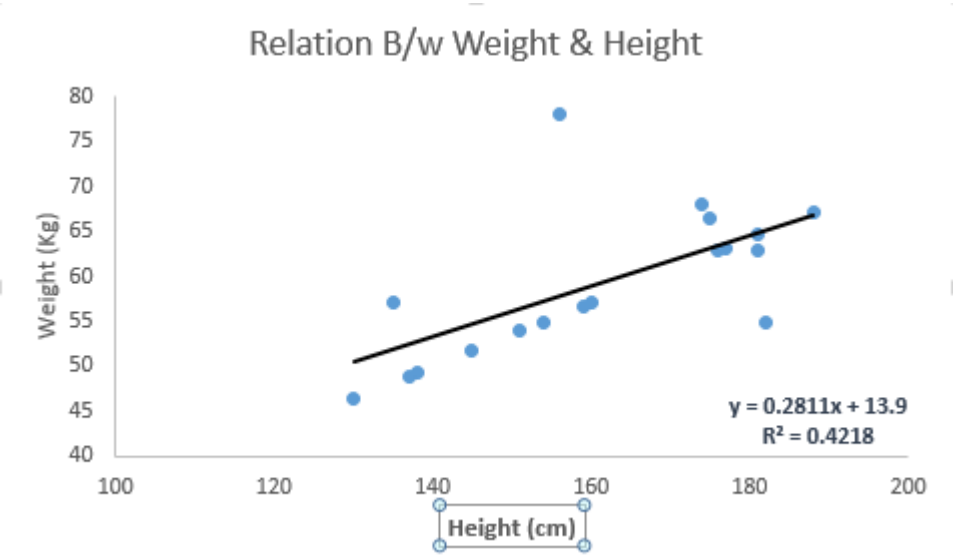
En esta ecuación:

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

Estos coeficientes a y b se derivan de la minimización de la suma de la diferencia al cuadrado de la distancia entre los puntos de datos y la línea de regresión.

Mira el siguiente ejemplo. Aquí hemos identificado la línea de mejor ajuste que tiene la ecuación

lineal **y = 0.2811x + 13.9** . Ahora usando esta ecuación, podemos encontrar el peso, sabiendo la altura de una persona.



La regresión lineal es principalmente de dos tipos: regresión lineal simple y regresión lineal múltiple. La regresión lineal simple se caracteriza por una variable independiente. Y, la regresión lineal múltiple (como sugiere su nombre) se caracteriza por múltiples variables independientes (más de 1). Mientras encuentra la mejor línea de ajuste, puede ajustar una regresión polinomial o curvilínea. Y estos son conocidos como regresión polinomial o curvilínea.

Solo una sugerencia sobre la implementación de regresión lineal 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)
```

He brindado un vistazo a la comprensión del aprendizaje supervisado que se adentra en el algoritmo de regresión lineal junto con un fragmento de código Python.

Lea Aprendizaje supervisado en línea: https://riptutorial.com/es/machine-learning/topic/2673/aprendizaje-supervisado

Capítulo 5: Comenzando con Machine Learning utilizando Apache spark MLib

Introducción

Apache spark MLib proporciona (JAVA, R, PYTHON, SCALA) 1.) Varios algoritmos de aprendizaje automático en regresión, clasificación, agrupación, filtrado colaborativo que se utilizan principalmente en el aprendizaje automático. 2.) Es compatible con la extracción de características, la transformación, etc. 3.) Permite a los profesionales de datos resolver sus problemas de aprendizaje automático (así como el cálculo de gráficos, la transmisión y el procesamiento de consultas interactivas en tiempo real) de manera interactiva y a una escala mucho mayor.

Observaciones

Por favor, refiérase a continuación para saber más sobre spark MLib

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

Examples

Escribe tu primer problema de clasificación usando el modelo de Regresión Logística

Estoy usando eclipse aquí, y necesita agregar la dependencia dada a su pom.xml a continuación

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 (tu clase de aplicación)

Estamos haciendo clasificación según el país, las horas y se hace clic en nuestra etiqueta.

```
    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.) Para ejecutar esta aplicación, primero realice `mvn-clean-package` en el proyecto de la aplicación, se crearía un `mvn-clean-package` jar. 4.) Abra el directorio raíz de chispa, y envíe este trabajo

```
bin/spark-submit --class com.predection.regression.App --master local[2] ./regression-0.0.1-
```

```
SNAPSHOT.jar(path to the jar file)
```

5.) Después de enviar ver, construye datos de entrenamiento

```
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.) datos de prueba de la misma manera

```
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.) Y aquí está el resultado de la predicción en la columna de predicción

```
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
```

Lea Comenzando con Machine Learning utilizando Apache spark MLib en línea: https://riptutorial.com/es/machine-learning/topic/9854/comenzando-con-machine-learning-utilizando-apache-spark-mlib

Capítulo 6: El aprendizaje automático y su clasificación.

Examples

¿Qué es el aprendizaje automático?

Se ofrecen dos definiciones de aprendizaje automático. *Arthur Samuel lo* describió como:

> el campo de estudio que otorga a las computadoras la capacidad de aprender sin ser explícitamente programadas.

Esta es una definición más antigua, informal.

Tom Mitchell ofrece una definición más moderna:

> Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas T y la medida de rendimiento P, si su desempeño en las tareas en T, medido por P, mejora con la experiencia E.

Ejemplo: jugar a las damas.

E = la experiencia de jugar muchos juegos de damas

T = la tarea de jugar a las damas.

P = la probabilidad de que el programa gane el siguiente juego.

En general, cualquier problema de aprendizaje automático puede asignarse a una de dos clasificaciones generales:

1. Aprendizaje supervisado
2. Aprendizaje sin supervisión.

¿Qué es el aprendizaje supervisado?

El aprendizaje supervisado es un tipo de algoritmo de aprendizaje automático que utiliza un conjunto de datos conocido (denominado conjunto de datos de entrenamiento) para hacer predicciones.

Categoría de aprendizaje supervisado:

1. **Regresión:** en un problema de regresión, estamos tratando de predecir los resultados dentro de una salida continua, lo que significa que estamos tratando de asignar las variables de entrada a alguna función continua.
2. **Clasificación:** en un problema de clasificación, en cambio, estamos tratando de predecir los resultados en un resultado discreto. En otras palabras, estamos tratando de asignar las

variables de entrada en categorías discretas.

Ejemplo 1:

Dados los datos sobre el tamaño de las casas en el mercado inmobiliario, intente predecir su precio. El precio en función del tamaño es una salida continua, por lo que este es un problema de regresión.

Ejemplo 2:

(a) *Regresión* - Para valores de respuesta continua. Por ejemplo, dada la imagen de una persona, tenemos que predecir su edad en función de la imagen dada

(b) *Clasificación* : para valores de respuesta categóricos, donde los datos se pueden separar en "clases" específicas. Por ejemplo, dado un paciente con un tumor, tenemos que predecir si el tumor es maligno o benigno.

¿Qué es el aprendizaje no supervisado?

El aprendizaje no supervisado nos permite abordar los problemas con poca o ninguna idea de cómo deberían ser nuestros resultados. Podemos derivar la estructura de datos donde no necesariamente sabemos el efecto de las variables.

Ejemplo:

Agrupación en clúster: se utiliza para el análisis exploratorio de datos para encontrar patrones ocultos o agrupación en datos. Tome una colección de 1,000,000 de genes diferentes, y encuentre una manera de agrupar automáticamente estos genes en grupos que de alguna manera sean similares o estén relacionados por diferentes variables, como la duración de la vida, la ubicación, los roles, etc.

Lea El aprendizaje automático y su clasificación. en línea: https://riptutorial.com/es/machine-learning/topic/9986/el-aprendizaje-automatico-y-su-clasificacion-

Capítulo 7: Métricas de evaluación

Examples

Área bajo la curva de la característica de operación del receptor (AUROC)

El **AUROC** es una de las métricas más utilizadas para evaluar las actuaciones de un clasificador. Esta sección explica cómo computarlo.

AUC (Área bajo la curva) se usa la mayor parte del tiempo para significar AUROC, lo que es una mala práctica, ya que AUC es ambigua (podría ser cualquier curva) mientras que AUROC no lo es.

Resumen - Abreviaturas

Abreviatura	Sentido
AUROC	Área bajo la curva de la característica de operación del receptor
AUC	Área bajo la curva
ROC	Característica de funcionamiento del receptor
TP	Verdaderos positivos
Tennesse	Verdaderos negativos
FP	Falsos positivos
FN	Falsos negativos
TPR	Tasa positiva verdadera
FPR	Tasa positiva falsa

Interpretando el AUROC

El AUROC tiene varias interpretaciones equivalentes :

- La expectativa de que un positivo aleatorio dibujado uniformemente se clasifica antes que un negativo aleatorio dibujado uniformemente.
- La proporción esperada de positivos clasificó antes de un negativo aleatorio uniformemente extraído.
- La tasa positiva verdadera esperada si la clasificación se divide justo antes de un negativo aleatorio de forma uniforme.

- La proporción esperada de negativos se clasificó después de un resultado positivo aleatorio uniformemente dibujado.
- La tasa de falsos positivos esperada si la clasificación se divide justo después de un positivo aleatorio extraído uniformemente.

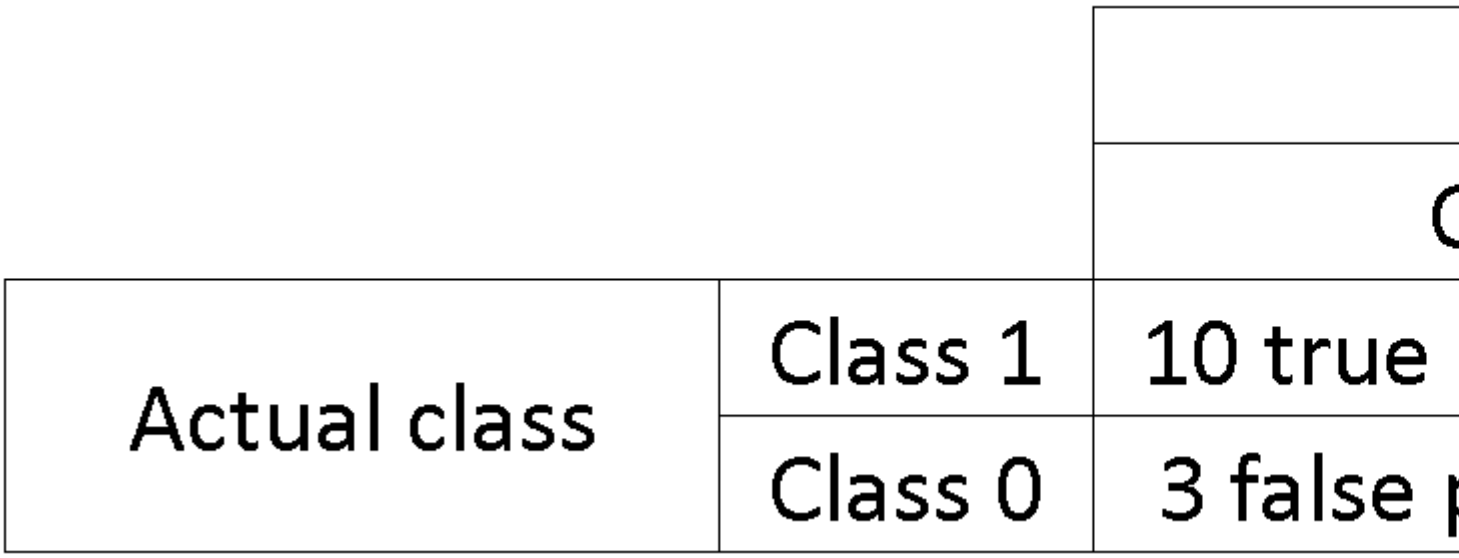
Calculando el auroc

Supongamos que tenemos un clasificador binario probabilístico, como la regresión logística.

Antes de presentar la curva **ROC** (= curva característica de funcionamiento del receptor), debe entenderse el concepto de **matriz** de **confusión** . Cuando hacemos una predicción binaria, puede haber 4 tipos de resultados:

- Predecimos **0** mientras la clase es en realidad **0** : esto se llama **Negativo Verdadero** , es decir, predecimos correctamente que la clase es negativa (0). *Por ejemplo, un antivirus no detectó un archivo inofensivo como un virus.*
- Predecimos **0** mientras la clase es en realidad **1** : esto se llama **Falso Negativo** , es decir, predecimos incorrectamente que la clase es negativa (0). *Por ejemplo, un antivirus no pudo detectar un virus.*
- Predecimos **1** mientras la clase es en realidad **0** : esto se denomina **Falso Positivo** , es decir, predecimos incorrectamente que la clase es positiva (1). *Por ejemplo, un antivirus considera que un archivo inocuo es un virus.*
- Predecimos **1** mientras la clase es en realidad **1** : esto se denomina **Verdadero Positivo** , es decir, predecimos correctamente que la clase es positiva (1). *Por ejemplo, un antivirus detectó con razón un virus.*

Para obtener la matriz de confusión, repasamos todas las predicciones hechas por el modelo y contamos cuántas veces ocurren cada uno de esos 4 tipos de resultados:



En este ejemplo de una matriz de confusión, entre los 50 puntos de datos que están clasificados,

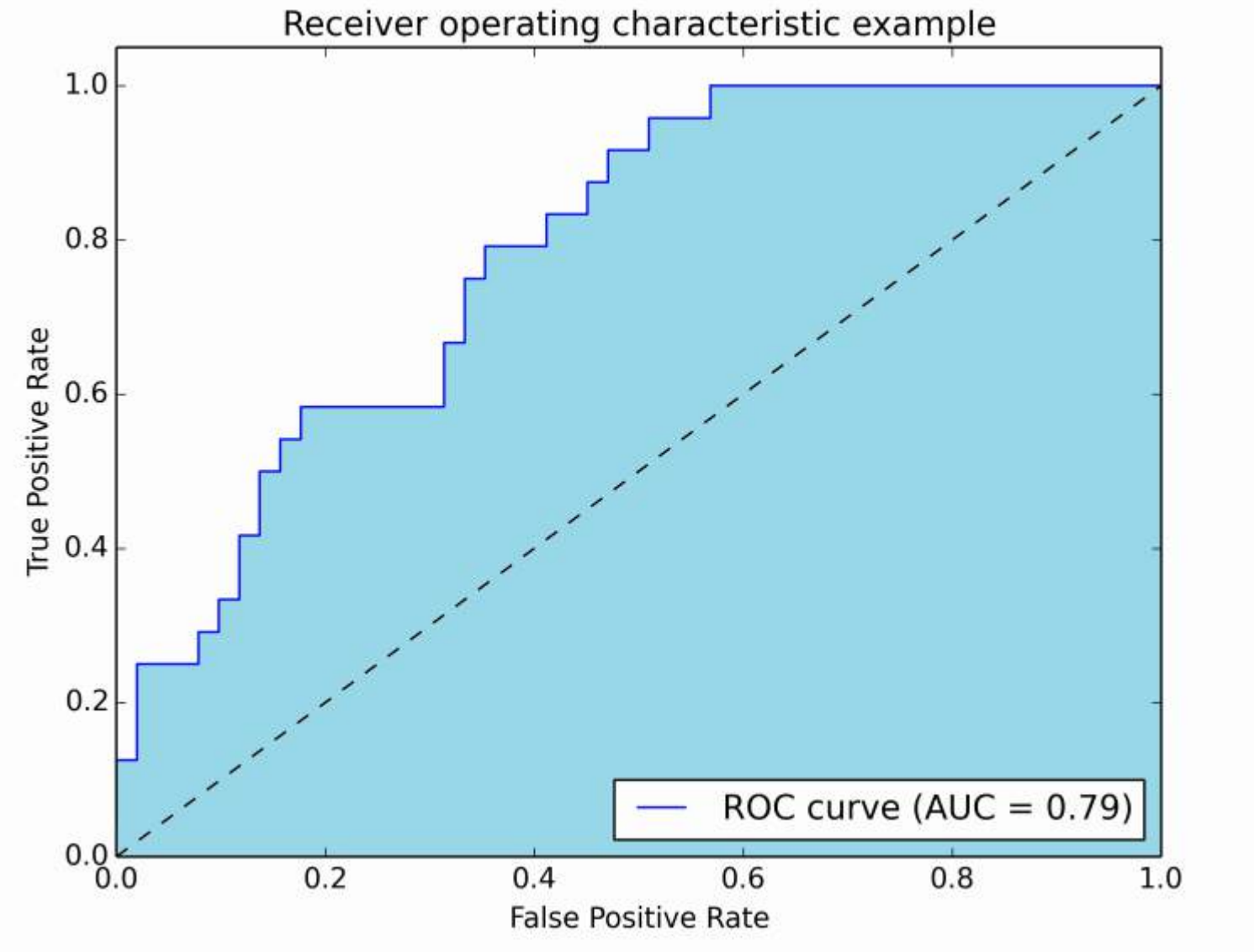
45 están clasificados correctamente y los 5 están clasificados erróneamente.

Como para comparar dos modelos diferentes, a menudo es más conveniente tener una sola métrica en lugar de varias, calculamos dos métricas de la matriz de confusión, que luego combinaremos en una:

- Tasa positiva verdadera (**TPR**), aka. sensibilidad, tasa de aciertos y recuperación , que se define como
  $\frac{TP}{TP+FN}$
  . Intuitivamente, esta métrica corresponde a la proporción de puntos de datos positivos que se consideran correctamente como positivos, con respecto a todos los puntos de datos positivos. En otras palabras, cuanto mayor sea el TPR, menos puntos de datos positivos perderemos.
- Tasa de falsos positivos (**FPR**), también conocido como. caída , que se define como
  $\frac{FP}{FP+TN}$
  . Intuitivamente, esta métrica corresponde a la proporción de puntos de datos negativos que se consideran erróneamente positivos con respecto a todos los puntos de datos negativos. En otras palabras, a mayor FPR, más puntos de datos negativos se clasificarán por error.

Para combinar el FPR y el TPR en una sola métrica, primero calculamos las dos métricas anteriores con muchos umbrales diferentes (por ejemplo, $0.00, 0.01 \ldots 1.00$) para la regresión logística, luego trácelas en un solo gráfico, con los valores de FPR en la abscisa y los valores de TPR en la ordenada. La curva resultante se llama curva ROC, y la métrica que consideramos es el AUC de esta curva, que llamamos AUROC.

La siguiente figura muestra gráficamente el AUROC:



En esta figura, el área azul corresponde al área bajo la curva de la característica de operación del receptor (AUROC). La línea discontinua en la diagonal presenta la curva ROC de un predictor aleatorio: tiene un AUROC de 0.5. El predictor aleatorio se usa comúnmente como una línea de base para ver si el modelo es útil.

Matriz de confusión

Se puede usar una matriz de confusión para evaluar un clasificador, en base a un conjunto de datos de prueba para los cuales se conocen los valores reales. Es una herramienta simple que ayuda a proporcionar una buena visión general del rendimiento del algoritmo que se está utilizando.

Una matriz de confusión se representa como una tabla. En este ejemplo veremos una **matriz de confusión para un clasificador binario** .

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

En el lado izquierdo, se puede ver la clase Real (etiquetada como *SÍ* o *NO*), mientras que la parte superior indica la clase que se está pronosticando y emitiendo (nuevamente *SÍ* o *NO*).

Esto significa que 50 clasificaciones de prueba, que en realidad *NO son* instancias, fueron clasificadas correctamente por el clasificador como *NO* . Estos se llaman los **verdaderos negativos (TN)** . En contraste, a 100 casos *SI* real, se marcaron correctamente por el clasificador como instancias *SÍ.* Estos se llaman los **verdaderos positivos (TP)** .

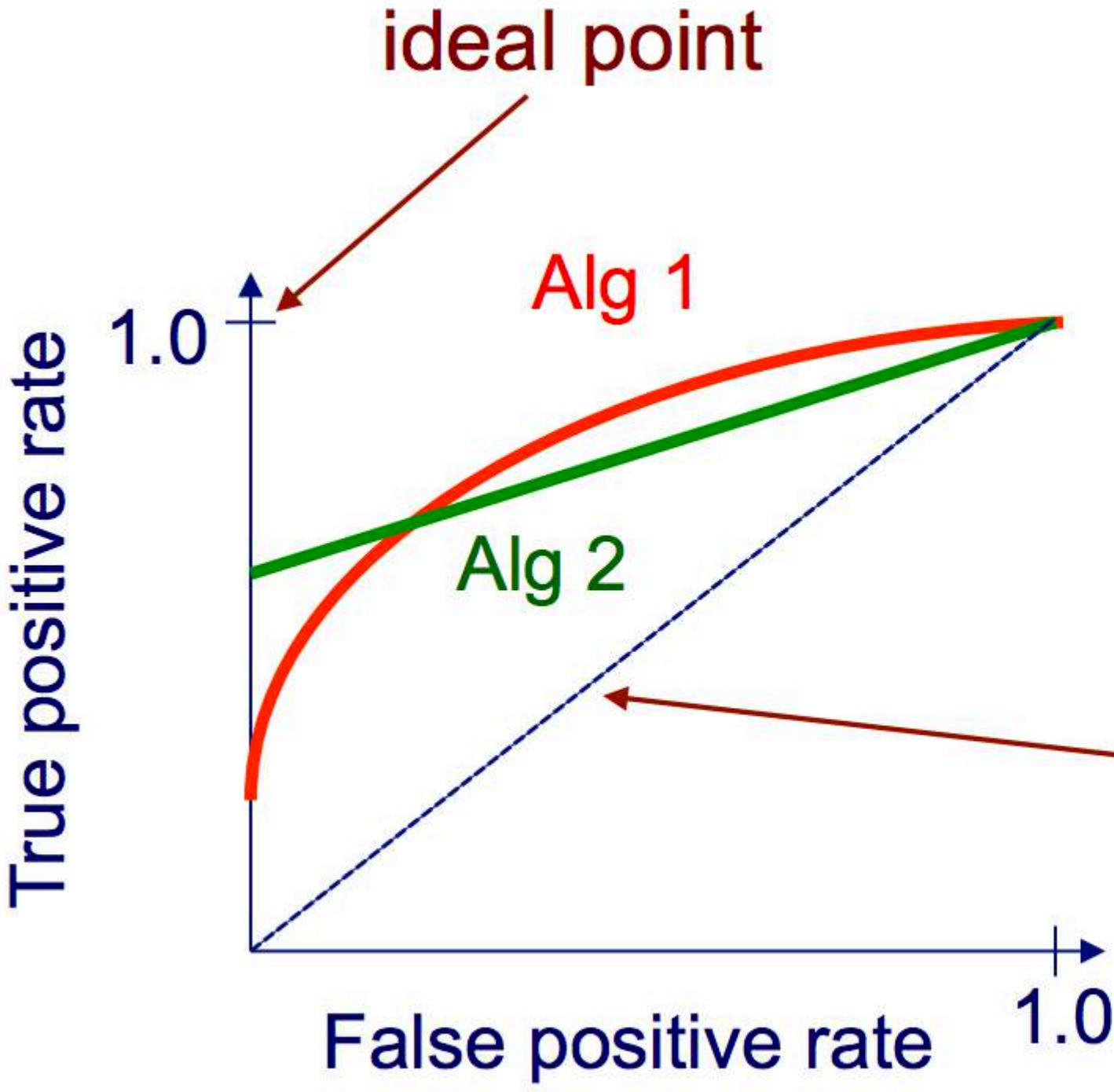
5 instancias reales de *SÍ* , fueron mal etiquetadas por el clasificador. Estos se llaman los **falsos negativos (FN)** . Además, 10 clasificaciones *NO* , fueron consideradas *SI* por el clasificador, por lo que son **falsos positivos (FP)** .

Sobre la base de estos **FP** , **TP** , **FN** y **TN** , podemos sacar conclusiones adicionales.

- **Tasa positiva verdadera** :
    - *Intenta responder:* cuando una instancia es realmente *SÍ* , ¿con qué frecuencia el clasificador predice *SÍ* ?
    - *Se puede calcular de la siguiente manera:* **TP** / # instancias reales *SÍ* = 100/105 = 0,95
- **Tasa positiva falsa** :
    - *Intenta responder:* cuando una instancia es realmente *NO* , ¿con qué frecuencia el clasificador predice *SÍ* ?
    - *Se puede calcular de la siguiente manera:* **FP** / # instancias reales *NO* = 10/60 = 0.17

Curvas ROC

Una curva de Característica operativa del receptor (ROC) traza la tasa de TP frente a la tasa de PF, ya que se varía un umbral sobre la confianza de que una instancia sea positiva



Algoritmo para crear una curva ROC

1. ordene las predicciones del conjunto de pruebas según la confianza de que cada instancia es positiva

2. paso a través de la lista ordenada de alta a baja confianza

   yo. localice un umbral entre instancias con clases opuestas (manteniendo las instancias con el mismo valor de confianza en el mismo lado del umbral)

ii. calcular TPR, FPR para instancias por encima del umbral

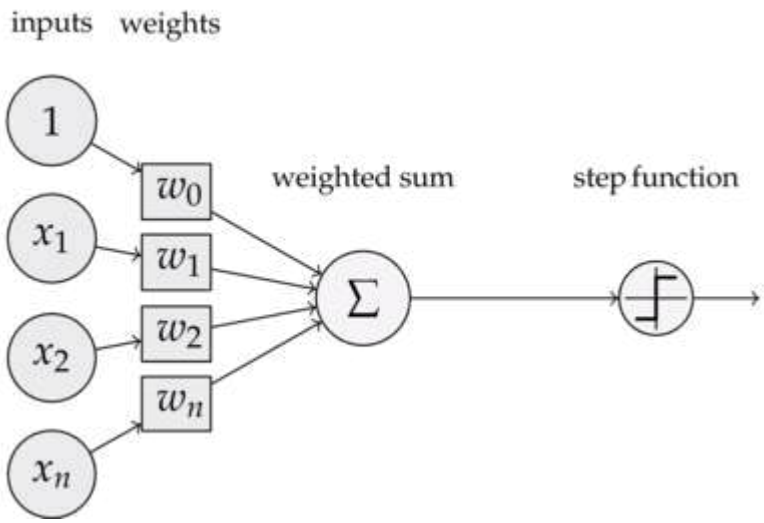
iii. coordenada de salida (FPR, TPR)

Lea Métricas de evaluación en línea: https://riptutorial.com/es/machine-learning/topic/4132/metricas-de-evaluacion

Capítulo 8: Perceptron

Examples

¿Qué es exactamente un perceptrón?



En su núcleo, un modelo de perceptrón es uno de los algoritmos de **aprendizaje supervisado** más simples para la clasificación binaria . Es un tipo de clasificador lineal , es decir, un algoritmo de clasificación que realiza sus predicciones basadas en una función de predictor lineal que combina un conjunto de pesos con el vector de características. Una forma más intuitiva de pensar es como una **red neuronal con una sola neurona** .

La forma en que funciona es muy simple. Obtiene un vector de valores de entrada ***x*** de los cuales cada elemento es una característica de nuestro conjunto de datos.

Un ejemplo:

Digamos que queremos clasificar si un objeto es una bicicleta o un automóvil. Por el bien de este ejemplo, digamos que queremos seleccionar 2 funciones. La altura y el ancho del objeto. En ese caso ***x = [x1, x2]*** donde ***x1*** es la altura y ***x2*** es el ancho.

Luego, una vez que tengamos nuestro vector de entrada ***x*** , queremos multiplicar cada elemento en ese vector con un peso. Por lo general, cuanto más alto es el valor del peso, más importante es la característica. Si, por ejemplo, utilizamos el ***color*** como característica ***x3*** y hay una bicicleta roja y un coche rojo, el perceptrón le asignará un peso muy bajo para que el color no afecte la predicción final.

Muy bien, así que hemos multiplicado los 2 vectores ***x*** y ***w*** y obtuvimos un vector. Ahora necesitamos sumar los elementos de este vector. Una forma inteligente de hacer esto es en lugar de simplemente multiplicar ***x*** por ***w*** podemos multiplicar ***x*** por ***wT*** donde ***T*** significa transposición. Podemos imaginar la ***transposición*** de un vector como una versión rotada del vector. Para más

información puedes leer la página de Wikipedia . Esencialmente, al tomar la transposición del vector ***w*** obtenemos un vector ***Nx1 en*** lugar de un ***1xN*** . Por lo tanto, si ahora multiplicamos nuestro vector de entrada con tamaño ***1xN*** con este vector de peso ***Nx1*** obtendremos un vector ***1x1*** (o simplemente un solo valor) que será igual a ***x1 * w1 + x2 * w2 + ... + xn * wn*** . Habiendo hecho eso, ahora tenemos nuestra predicción. Pero hay una última cosa. Esta predicción probablemente no será un simple 1 o -1 para poder clasificar una nueva muestra. Entonces, lo que podemos hacer es simplemente decir lo siguiente: si nuestra predicción es mayor que 0, entonces decimos que la muestra pertenece a la clase 1, de lo contrario, si la predicción es menor que cero, diremos que la muestra pertenece a la clase -1. Esto se llama una función de paso .

Pero, ¿cómo obtenemos las ponderaciones correctas para hacer predicciones correctas? En otras palabras, ¿cómo ***entrenamos*** nuestro modelo de percepción?

Bueno, en el caso del perceptrón, no necesitamos ecuaciones matemáticas sofisticadas para ***entrenar*** nuestro modelo. Nuestros pesos pueden ser ajustados por la siguiente ecuación:

Δw = eta * (y - predicción) * x (i)

donde ***x (i)*** es nuestra función (x1 por ejemplo para peso 1, x2 para w2 y así sucesivamente ...).

También notó que hay una variable llamada ***eta*** que es la tasa de aprendizaje. Puedes imaginarte la tasa de aprendizaje como cuán grande queremos que sea el cambio de los pesos. Una buena tasa de aprendizaje resulta en un algoritmo de aprendizaje rápido. Un valor demasiado alto de ***eta*** puede resultar en una cantidad creciente de errores en cada época y los resultados en el modelo hacen predicciones realmente malas y nunca convergen. Una tasa de aprendizaje demasiado baja puede tener como resultado que el modelo tome demasiado tiempo para converger. (Por lo general, un buen valor para establecer ***eta*** para el modelo de perceptrón es 0.1 pero puede diferir de un caso a otro).

Finalmente, algunos de ustedes habrán notado que la primera entrada es una constante (1) y se multiplica por w0. Entonces, ¿qué es exactamente eso? Para obtener una buena predicción necesitamos agregar un sesgo. Y eso es exactamente lo que es esa constante.

Para modificar el peso del término de sesgo usamos la misma ecuación que hicimos para los otros pesos, pero en este caso no lo multiplicamos por la entrada (porque la entrada es una constante 1 y no tenemos que hacerlo):

Δw = eta * (y - predicción)

¡Así es básicamente como funciona un modelo simple de perceptrón! Una vez que entrenamos nuestros pesos, podemos darle nuevos datos y tener nuestras predicciones.

NOTA:

El modelo Perceptron tiene una desventaja importante! Nunca convergerá (ei encontrará los pesos perfectos) si los datos no se pueden separar linealmente , lo que significa poder separar las 2 clases en un espacio de características mediante una línea recta. Por lo tanto, para evitar eso,

es una buena práctica agregar un número fijo de iteraciones para que el modelo no se quede atascado al ajustar los pesos que nunca se ajustarán perfectamente.

Implementando un modelo de Perceptron en C ++

En este ejemplo, pasaré por la implementación del modelo de perceptrón en C ++ para que pueda tener una mejor idea de cómo funciona.

Primero lo primero es una buena práctica escribir un algoritmo simple de lo que queremos hacer.

Algoritmo:

1. Haga un vector para los pesos e inicialícelo a 0 (No olvide agregar el término de sesgo)
2. Siga ajustando los pesos hasta que obtengamos 0 errores o un recuento bajo de errores.
3. Hacer predicciones sobre datos invisibles.

Habiendo escrito un algoritmo súper simple, ahora escribamos algunas de las funciones que necesitaremos.

- Necesitaremos una función para calcular la entrada de la red (ei ***x * wT*** multiplicando las entradas por el tiempo de los pesos)
- Una función escalonada para que podamos obtener una predicción de 1 o -1
- Y una función que encuentra los valores ideales para los pesos.

Así que sin más preámbulos, entremos en esto.

Comencemos simple creando una clase de perceptron:

```
class perceptron
{
public:

private:

};
```

Ahora agreguemos las funciones que necesitaremos.

```
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:

};
```

Observe cómo el ***ajuste de*** la función toma como argumento un vector del vector <float>. Esto se debe a que nuestro conjunto de datos de capacitación es una matriz de entradas. Esencialmente, podemos imaginar que la matriz como un par de vectores ***x*** apiladas una encima de otra y cada

columna de esa Matriz es una característica.

Finalmente, agreguemos los valores que nuestra clase necesita tener. Como el vector ***w*** para mantener los pesos, el número de ***épocas*** que indica el número de pases que realizaremos sobre el conjunto de datos de entrenamiento. Y la constante ***eta,*** cuya velocidad de aprendizaje multiplicaremos cada actualización de peso para acelerar el procedimiento de entrenamiento al aumentar este valor o si ***eta*** es demasiado alto, podemos marcarlo para obtener el resultado ideal (para la mayoría de las aplicaciones del perceptrón sugeriría un valor ***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;
};
```

Ahora con nuestro conjunto de clases. Es hora de escribir cada una de las funciones.

Comenzaremos desde el constructor (**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
}
```

Como pueden ver, lo que haremos es algo muy simple. Así que vamos a pasar a otra función simple. La función de predicción (***int predecir (vector X);***). Recuerde que lo que hace la función de ***predicción*** total es tomar la entrada neta y devolver un valor de 1 si el ***netInput*** es mayor que 0 y -1 en cualquier otro lugar.

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

Observe que usamos una declaración en línea si para hacer nuestras vidas más fáciles. Así es como funciona la instrucción inline if:

condición? if_true: else

Hasta ahora tan bueno. Continuemos con la implementación de la función ***netInput*** (***float netInput (vector X);***)

El netInput hace lo siguiente; ***multiplica el vector de entrada por la transposición del vector de pesos***

x * wT

En otras palabras, multiplica cada elemento del vector de entrada ***x*** por el elemento correspondiente del vector de ponderaciones ***w*** y luego toma su suma y agrega el sesgo.

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

sesgo = 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;
}
```

Bien, ahora ya casi hemos terminado. Lo último que tenemos que hacer es escribir la función de ***ajuste*** que modifica los pesos.

```
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
        }
    }
}
```

Así que eso era esencialmente eso. ¡Con solo 3 funciones ahora tenemos una clase de percepción de trabajo que podemos usar para hacer predicciones!

En caso de que quieras copiar y pegar el código y probarlo. Aquí está la clase completa (agregué algunas funciones adicionales, como la impresión del vector de pesos y los errores en cada época, así como la opción de importar pesos de exportación).

Aquí está el código:

El encabezado de la clase:

```
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;
};
```

El archivo de clase .cpp con las funciones:

```
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;
}
```

Además, si quieres probar un ejemplo, aquí hay un ejemplo que hice:

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 forma en que importé el conjunto de datos del iris no es realmente ideal, pero solo quería algo que funcionara.

Los archivos de datos se pueden encontrar aquí.

Espero que hayas encontrado esto útil!

Que es el sesgo

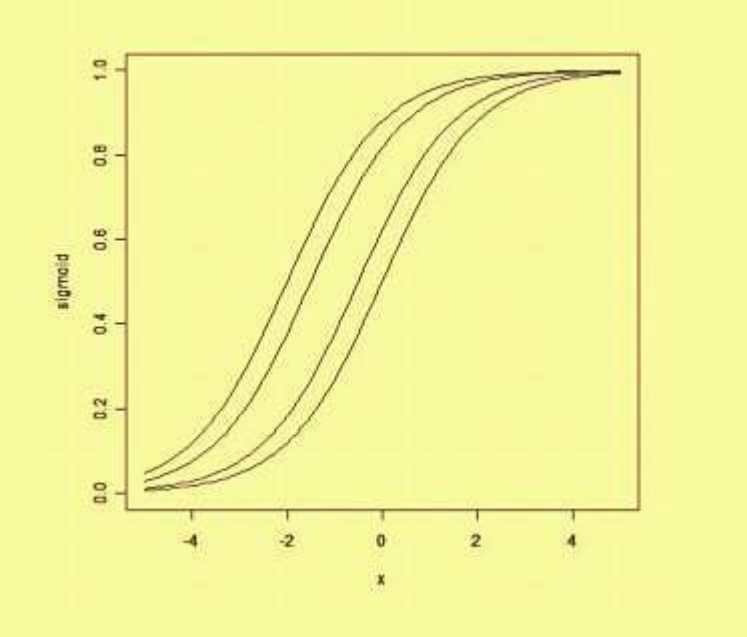
Que es el sesgo

Un perceptrón se puede ver como una función que mapea un vector de entrada (valor real) ***x*** a un valor de salida ***f (x)*** (valor binario):

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

donde ***w*** es un vector de pesos en valores reales y ***b*** es nuestro valor de *sesgo* . El sesgo es un valor que aleja el límite de decisión del origen *(0,0)* y no depende de ningún valor de entrada.

Pensando en el sesgo de una manera espacial, el sesgo altera la posición (aunque no la orientación) del límite de decisión. A continuación podemos ver un ejemplo de la misma curva desplazada por el sesgo:



Lea Perceptron en línea: https://riptutorial.com/es/machine-learning/topic/6640/perceptron

Capítulo 9: Procesamiento natural del lenguaje

Introducción

La PNL es una forma en que las computadoras analizan, comprenden y derivan el significado del lenguaje humano de una manera inteligente y útil. Al utilizar la PNL, los desarrolladores pueden organizar y estructurar el conocimiento para realizar tareas como el resumen automático, la traducción, el reconocimiento de entidades nombradas, la extracción de relaciones, el análisis de sentimientos, el reconocimiento de voz y la segmentación de temas.

Examples

Coincidencia de texto o similitud

> Una de las áreas importantes de la PNL es la comparación de objetos de texto para encontrar similitudes. Las aplicaciones importantes de la correspondencia de texto incluyen la corrección automática de la ortografía, la deduplicación de datos y el análisis del genoma, etc. Hay varias técnicas de correspondencia de texto disponibles según el requisito. Así que vamos a tener; **Levenshtein Distancia**

La distancia de Levenshtein entre dos cadenas se define como el número mínimo de ediciones necesarias para transformar una cadena en otra, con las operaciones de edición permitidas que son la inserción, eliminación o sustitución de un solo carácter.

A continuación se muestra la implementación para cálculos de memoria eficientes.

```
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"))
```

Lea Procesamiento natural del lenguaje en línea: https://riptutorial.com/es/machine-

learning/topic/10734/procesamiento-natural-del-lenguaje

Capítulo 10: Redes neuronales

Examples

Comenzando: Una ANN simple con Python

El código que figura a continuación intenta clasificar los dígitos escritos a mano a partir del conjunto de datos MNIST. Los dígitos se ven así:

El código preprocesará estos dígitos, convirtiendo cada imagen en una matriz 2D de 0s y 1s, y luego utilizará estos datos para entrenar una red neuronal con una precisión de hasta el 97% (50 épocas).

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

Este es un ejemplo de código autocontenido, y puede ejecutarse sin más modificaciones. Asegúrate de que tienes `numpy` y `scikit` learn para tu versión de python.

Backpropagation - El corazón de las redes neuronales

El objetivo de la propagación hacia atrás es optimizar los pesos para que la red neuronal pueda aprender cómo asignar correctamente las entradas arbitrarias a las salidas.

Cada capa tiene su propio conjunto de pesos, y estos pesos deben ajustarse para poder predecir con precisión la salida correcta dada la entrada.

Una visión general de alto nivel de la propagación hacia atrás es la siguiente:

1. Pase hacia adelante: la entrada se transforma en alguna salida. En cada capa, la activación se calcula con un producto de punto entre la entrada y los pesos, seguido de la suma de la resultante con el sesgo. Finalmente, este valor se pasa a través de una función de activación, para obtener la activación de esa capa que se convertirá en la entrada a la siguiente capa.
2. En la última capa, la salida se compara con la etiqueta real correspondiente a esa entrada, y se calcula el error. Por lo general, es el error cuadrático medio.
3. Paso hacia atrás: el error calculado en el paso 2 se propaga de nuevo a las capas internas y los pesos de todas las capas se ajustan para dar cuenta de este error.

1. Inicialización de pesas

A continuación se muestra un ejemplo simplificado de inicialización de pesos:

```
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 capa 1 oculta tiene un peso de dimensión [64, 784] y sesgo de dimensión 64.
- La capa de salida tiene un peso de dimensión [10, 64] y un sesgo de dimensión

    10.

Es posible que se pregunte qué sucede al inicializar los pesos en el código anterior. Esto se llama inicialización de Xavier, y es un paso mejor que la inicialización aleatoria de sus matrices de peso. Sí, la inicialización sí importa. Según su inicialización, es posible que pueda encontrar mejores mínimos locales durante el descenso del gradiente (la propagación hacia atrás es una versión glorificada del descenso del gradiente).

2. Pase hacia adelante

```
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))
```

Este código realiza la transformación descrita anteriormente. `hidden_activations[-1]` contiene probabilidades de softmax: predicciones de todas las clases, cuya suma es 1. Si estamos pronosticando dígitos, la salida será un vector de probabilidades de dimensión 10, cuya suma es 1.

3. Pase hacia atrás

```
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
```

Las 2 primeras lineas inicializan los gradientes. Estos gradientes se calculan y se utilizarán para actualizar las ponderaciones y los sesgos más adelante.

Las siguientes 3 líneas calculan el error restando la predicción del objetivo. El error se propaga de nuevo a las capas internas.

Ahora, sigue cuidadosamente el funcionamiento del bucle. Las líneas 2 y 3 transforman el error de la `layer[i]` a la `layer[i - 1]` . Traza las formas de las matrices que se multiplican para entender.

4. Pesos / Actualización de Parámetros

```
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` especifica la velocidad a la que la red aprende. No quieres que aprenda demasiado rápido, porque puede que no converja. Se favorece un suave descenso para encontrar un buen mínimo. Generalmente, las tasas entre `0.01` y `0.1` se consideran buenas.

Funciones de activacion

Las funciones de activación también conocidas como función de transferencia se utilizan para asignar los nodos de entrada a los nodos de salida de cierta manera.

Se utilizan para impartir no linealidad a la salida de una capa de red neuronal.

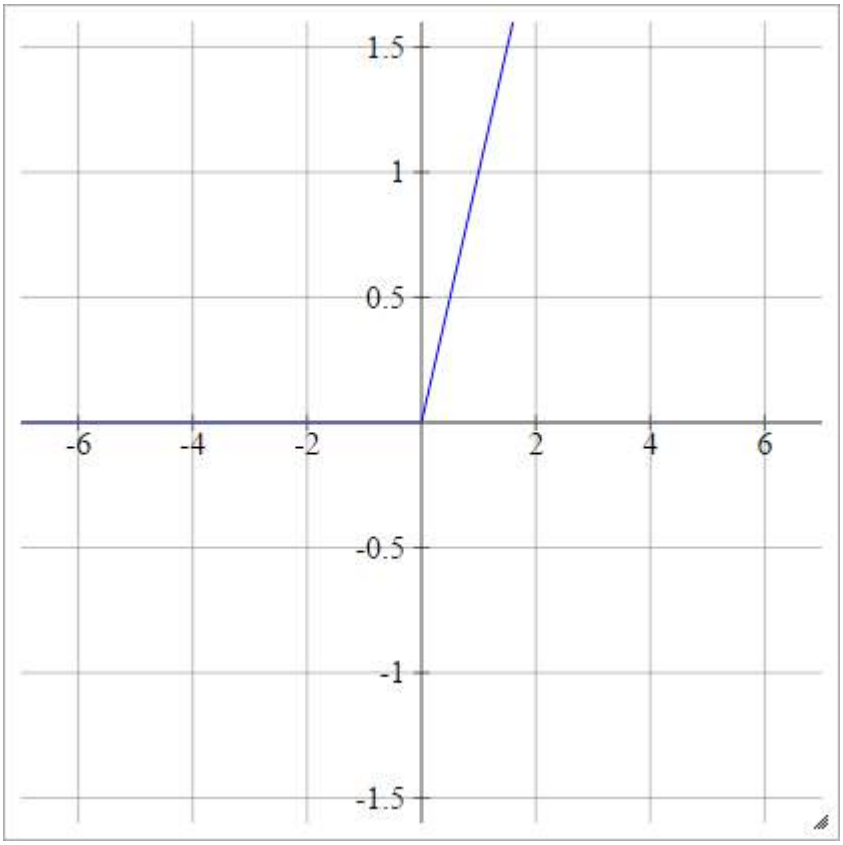
Algunas funciones de uso común y sus curvas se dan a continuación:

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

para calcular la activación de una capa oculta.

Función ReLU

Una unidad lineal rectificada hace simplemente `max(0,x)` . Es una de las opciones más comunes para las funciones de activación de las unidades de red neuronal.



Las ReLU abordan el problema de la degradación de la degradación de las unidades tangentes sigmoideas / hiperbólicas, lo que permite una propagación eficiente de las degradaciones en redes profundas.

El nombre ReLU proviene del documento de Nair y Hinton, *Unidades lineales rectificadas que mejoran las máquinas de Boltzmann restringidas* .

Tiene algunas variaciones, por ejemplo, ReLU (LReLU) con fugas y Unidades lineales exponenciales (ELU).

El código para implementar ReLU de vainilla junto con su derivado con `numpy` se muestra a continuación:

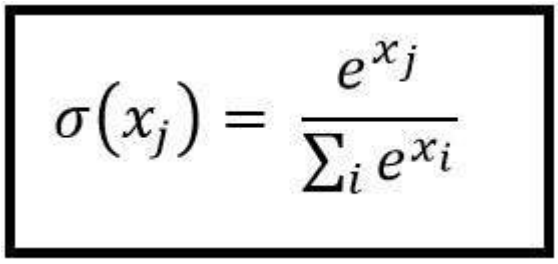
```
def relU(z):
    return z * (z > 0)

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

Función softmax

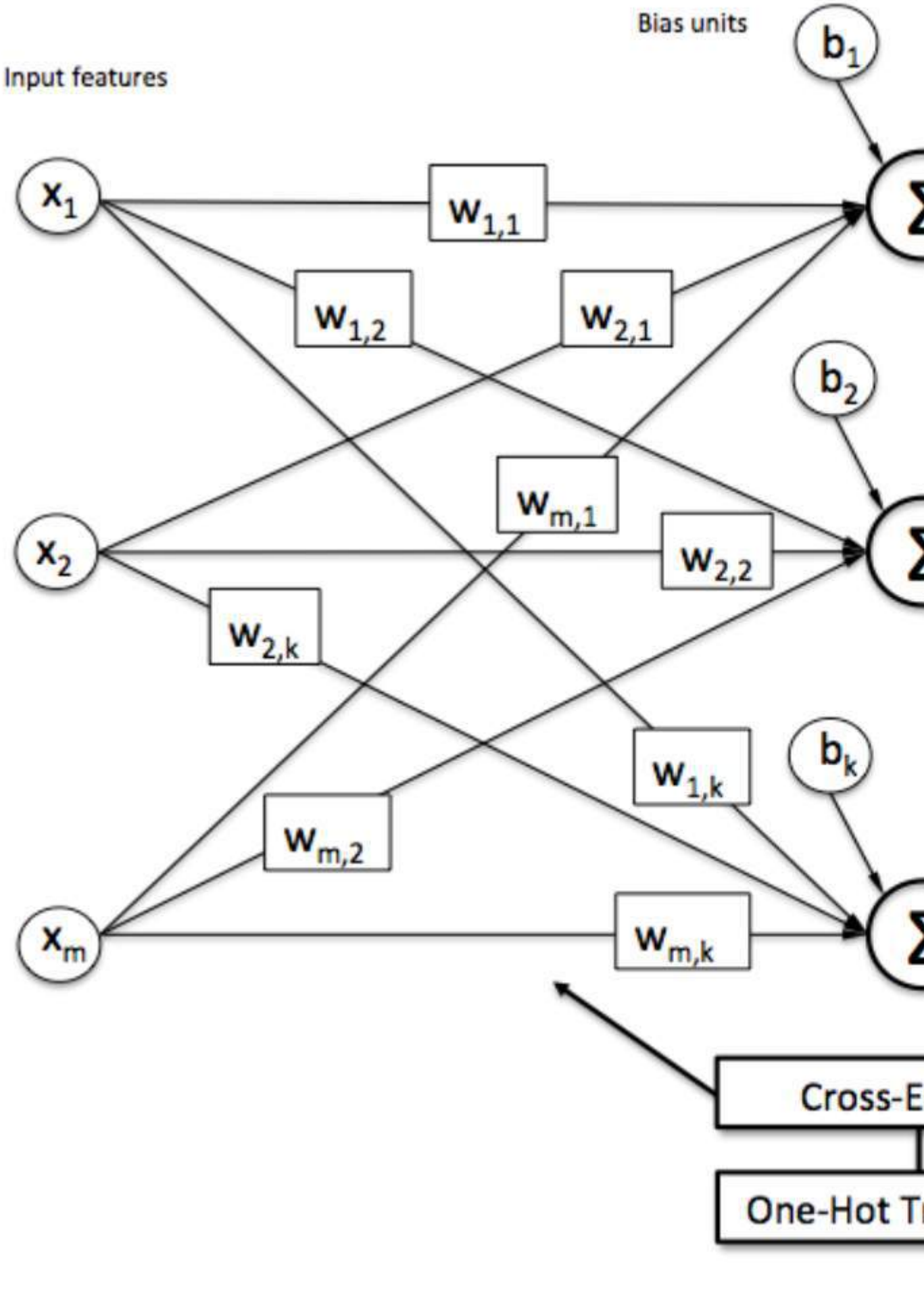
La regresión de Softmax (o regresión logística multinomial) es una generalización de la regresión logística al caso en el que queremos manejar múltiples clases. Es particularmente útil para redes neuronales donde queremos aplicar una clasificación no binaria. En este caso, la regresión logística simple no es suficiente. Necesitaríamos una distribución de probabilidad en todas las etiquetas, que es lo que nos brinda softmax.

Softmax se calcula con la siguiente fórmula:



_____ ¿Dónde encaja?

_____


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

Para normalizar un vector aplicándole la función `numpy` con `numpy` , use:

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

Donde `x` es la activación de la capa final de la ANN.

Lea Redes neuronales en línea: https://riptutorial.com/es/machine-learning/topic/2709/redes-neuronales

Capítulo 11: Scikit Learn

Examples

Un problema de clasificación simple simple (XOR) que usa k el algoritmo del vecino más cercano

Considera que quieres predecir la respuesta correcta para el problema popular de XOR. Sabías lo que es XOR (por ejemplo, [x0 x1] => y). por ejemplo [0 0] => 0, [0 1] => [1] y ...

```
#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]]))
```

Clasificación en scikit-learn

1. Árboles de decisión embolsados

El ensacado se realiza mejor con algoritmos que tienen una gran variación. Un ejemplo popular son los árboles de decisión, a menudo construidos sin poda.

En el siguiente ejemplo, vea un ejemplo del uso de BaggingClassifier con el algoritmo de árboles de clasificación y regresión (DecisionTreeClassifier). Se crean un total de 100 árboles.

Conjunto de datos utilizado: conjunto de datos de diabetes de los indios 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())
```

Siguiendo el ejemplo, obtenemos una estimación robusta de la precisión del modelo.

```
0.770745044429
```

2. Bosque aleatorio

El bosque aleatorio es una extensión de los árboles de decisión embolsados.

Las muestras del conjunto de datos de entrenamiento se toman con reemplazo, pero los árboles se construyen de una manera que reduce la correlación entre los clasificadores individuales. Específicamente, en lugar de elegir con avidez el mejor punto de división en la construcción del árbol, solo se considera un subconjunto aleatorio de características para cada división.

Puede construir un modelo de bosque aleatorio para la clasificación utilizando la clase RandomForestClassifier.

El siguiente ejemplo proporciona un ejemplo de Random Forest para la clasificación con 100 árboles y puntos de división seleccionados de una selección aleatoria de 3 características.

```
# 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())
```

Ejecutar el ejemplo proporciona una estimación media de la precisión de la clasificación.

```
0.770727956254
```

3. AdaBoost

AdaBoost fue quizás el primer algoritmo conjunto exitoso. Generalmente funciona al ponderar las instancias en el conjunto de datos por lo fácil o difícil que es clasificarlas, lo que permite que el algoritmo les preste más atención a la construcción de modelos posteriores.

Puede construir un modelo de AdaBoost para la clasificación utilizando la clase AdaBoostClassifier.

El siguiente ejemplo muestra la construcción de 30 árboles de decisión en secuencia utilizando el algoritmo AdaBoost.

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

Ejecutar el ejemplo proporciona una estimación media de la precisión de la clasificación.

```
0.76045796309
```

4. Incremento del gradiente estocástico

El aumento de gradiente estocástico (también llamado máquinas de aumento de gradiente) es una de las técnicas de conjunto más sofisticadas. También es una técnica que está demostrando ser quizás una de las mejores técnicas disponibles para mejorar el rendimiento a través de conjuntos.

Puede construir un modelo de Gradient Boosting para la clasificación utilizando la clase GradientBoostingClassifier.

El siguiente ejemplo muestra el aumento de gradiente estocástico para la clasificación con 100 árboles.

```
# 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())
```

Ejecutar el ejemplo proporciona una estimación media de la precisión de la clasificación.

```
0.764285714286
```

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

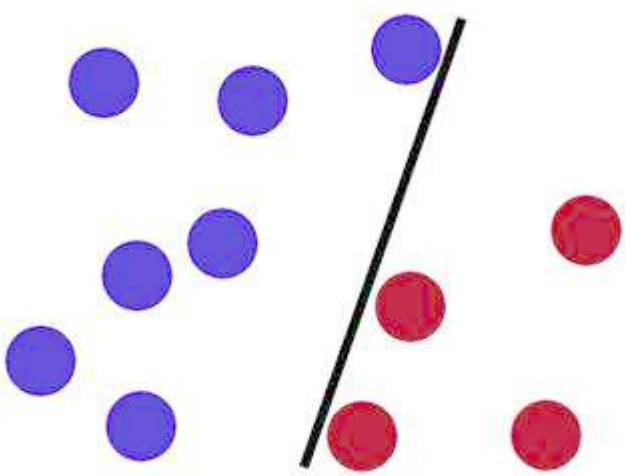
Lea Scikit Learn en línea: https://riptutorial.com/es/machine-learning/topic/6156/scikit-learn

Capítulo 12: SVM

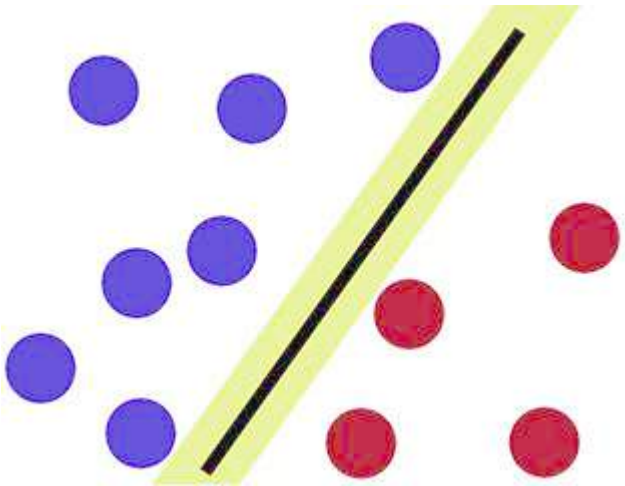
Examples

Diferencia entre regresión logística y SVM.

Límite de decisión cuando clasificamos usando **regresión logística** -



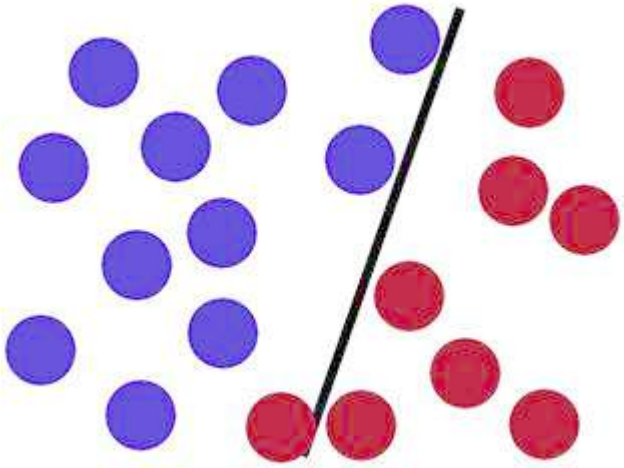
Límite de decisión cuando clasificamos usando **SVM** -



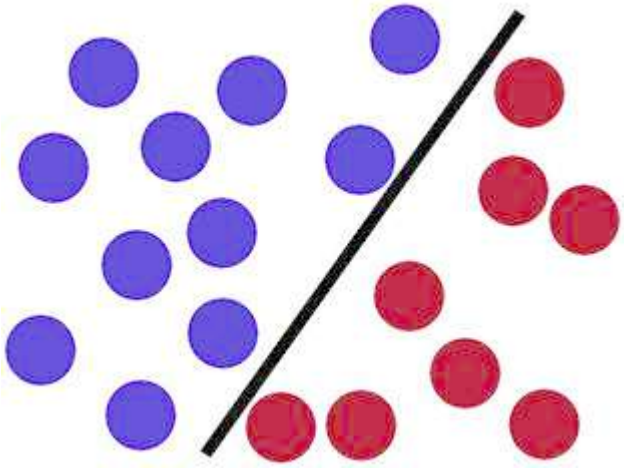
Como se puede observar, la SVM intenta mantener un "vacío" en ambos lados del límite de decisión. Esto resulta útil cuando nos encontramos con nuevos datos.

Con nuevos datos

La regresión logística funciona **mal** (el nuevo círculo rojo se clasifica como azul)



Mientras que **SVM** puede clasificarlo correctamente (el nuevo círculo rojo se clasifica correctamente en el lado rojo) -



Implementando el clasificador SVM usando Scikit-learn:

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

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

Lea SVM en línea: https://riptutorial.com/es/machine-learning/topic/7126/svm

Capítulo 13: Tipos de aprendizaje

Examples

Aprendizaje supervisado

La máquina aprende a predecir una salida cuando se le da una entrada.

Cada caso de entrenamiento consiste en una entrada y una salida objetivo.

Regresión

La salida de destino toma valores continuos.

- Predecir el precio de una acción.
- Predecir el precio de una casa

Clasificación

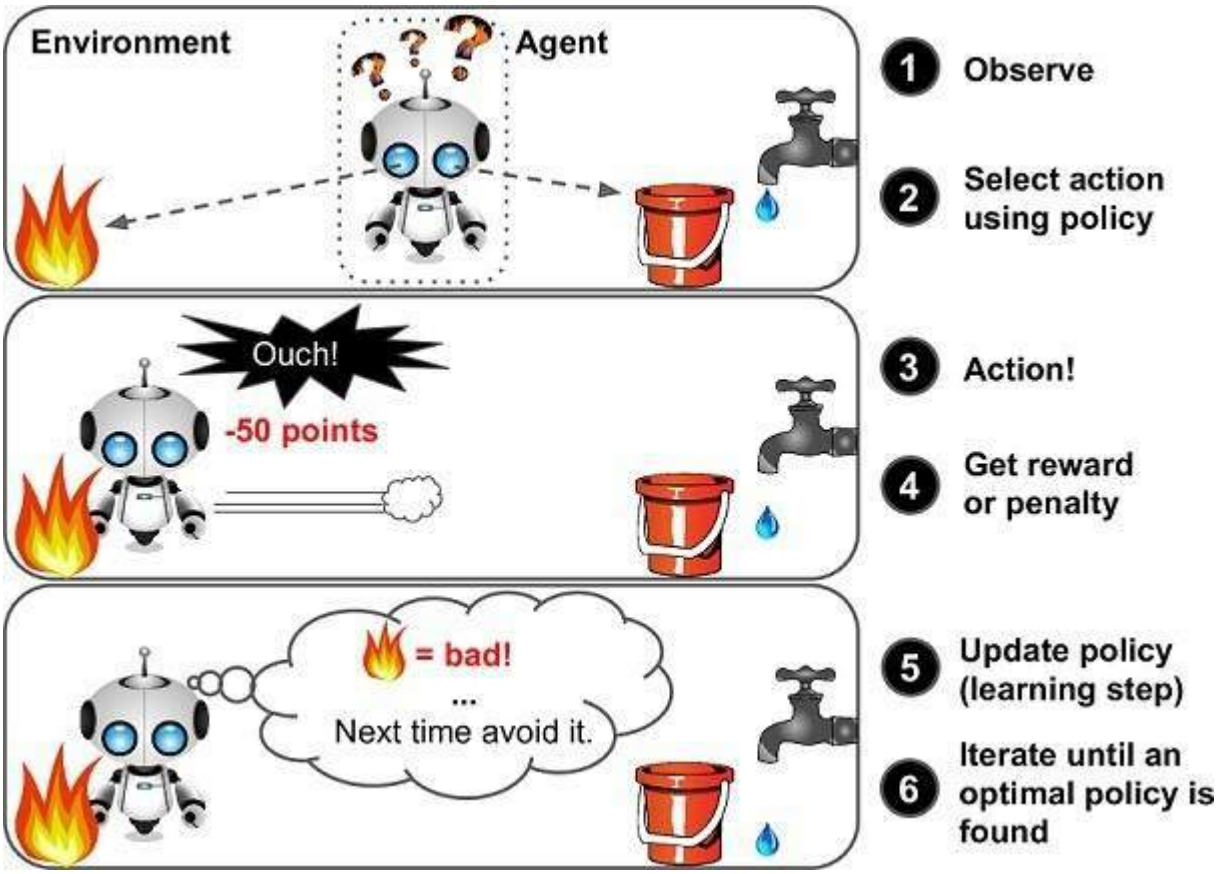
La salida de destino es una etiqueta de clase.

- ¿Qué tipo de fruta es la entrada?
- Que idioma es una palabra

Aprendizaje reforzado

La máquina tiene que determinar automáticamente el comportamiento ideal para maximizar su rendimiento.

Por ejemplo:



Con el aprendizaje por refuerzo también puede crear un programa de computadora que pueda completar un nivel de Mario (MarI / O - Aprendizaje automático para videojuegos).

Aprendizaje sin supervisión

El aprendizaje no supervisado nos permite abordar los problemas con poca o ninguna idea de cómo deberían ser nuestros resultados. Podemos **derivar la estructura de datos** donde no necesariamente sabemos el efecto de las variables.

El tipo más común de aprendizaje no supervisado es el **análisis de cluster o clustering** . Es la tarea de agrupar un conjunto de objetos de tal manera que los objetos en el mismo grupo (grupo) sean más similares entre sí que a los de otros grupos.

También hay aprendizaje no supervisado no agrupado. Un ejemplo de ello es identificar voces individuales y música de una malla de sonidos. Esto se llama el "algoritmo del cóctel".

Lea Tipos de aprendizaje en línea: https://riptutorial.com/es/machine-learning/topic/10912/tipos-de-aprendizaje

Capítulo 14: Una introducción a la clasificación: Generando varios modelos usando Weka

Introducción

Este tutorial le mostrará cómo usar Weka en el código JAVA, cargar archivos de datos, entrenar clasificadores y explica algunos de los conceptos importantes detrás del aprendizaje automático.

Weka es un conjunto de herramientas para el aprendizaje automático. Incluye una biblioteca de técnicas de aprendizaje automático y visualización y presenta una GUI fácil de usar.

Este tutorial incluye ejemplos escritos en JAVA e incluye imágenes generadas con la GUI. Sugiero usar la GUI para examinar los datos y el código JAVA para experimentos estructurados.

Examples

Comenzando: Cargando un conjunto de datos desde el archivo

El conjunto de datos de la flor del iris es un conjunto de datos ampliamente utilizado para fines de demostración. Lo cargaremos, lo inspeccionaremos y lo modificaremos ligeramente para su uso posterior.

```
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();
    }
}
```

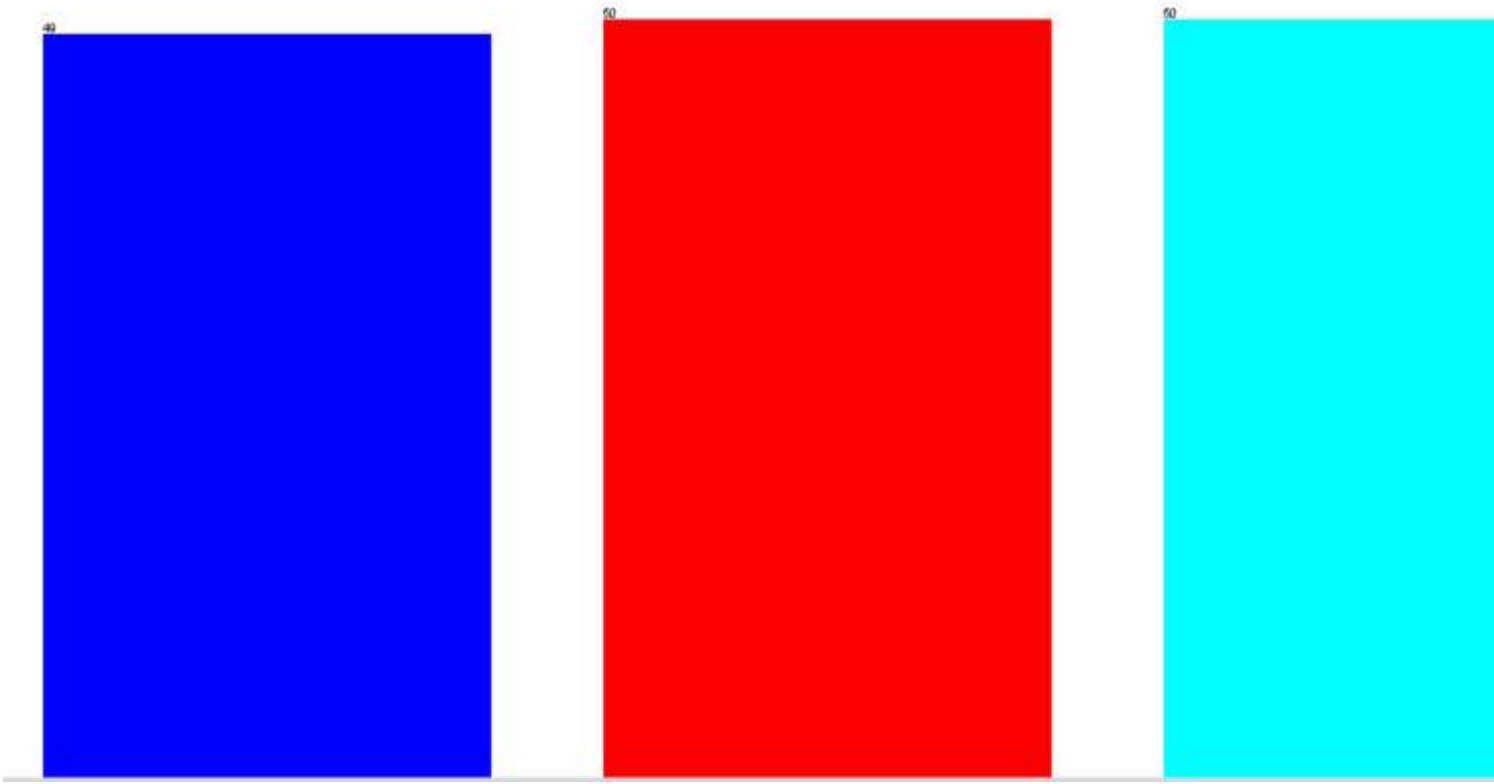
Entrene al primer clasificador: establecer una línea de base con ZeroR

ZeroR es un clasificador simple. No opera por instancia, sino que opera en la distribución general de las clases. Selecciona la clase con la mayor probabilidad a priori. No es un buen clasificador en el sentido de que no utiliza ninguna información en el candidato, pero a menudo se utiliza como una línea de base. **Nota: Se pueden usar otras líneas de base como, por ejemplo: clasificadores estándar de la industria o reglas artesanales**

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

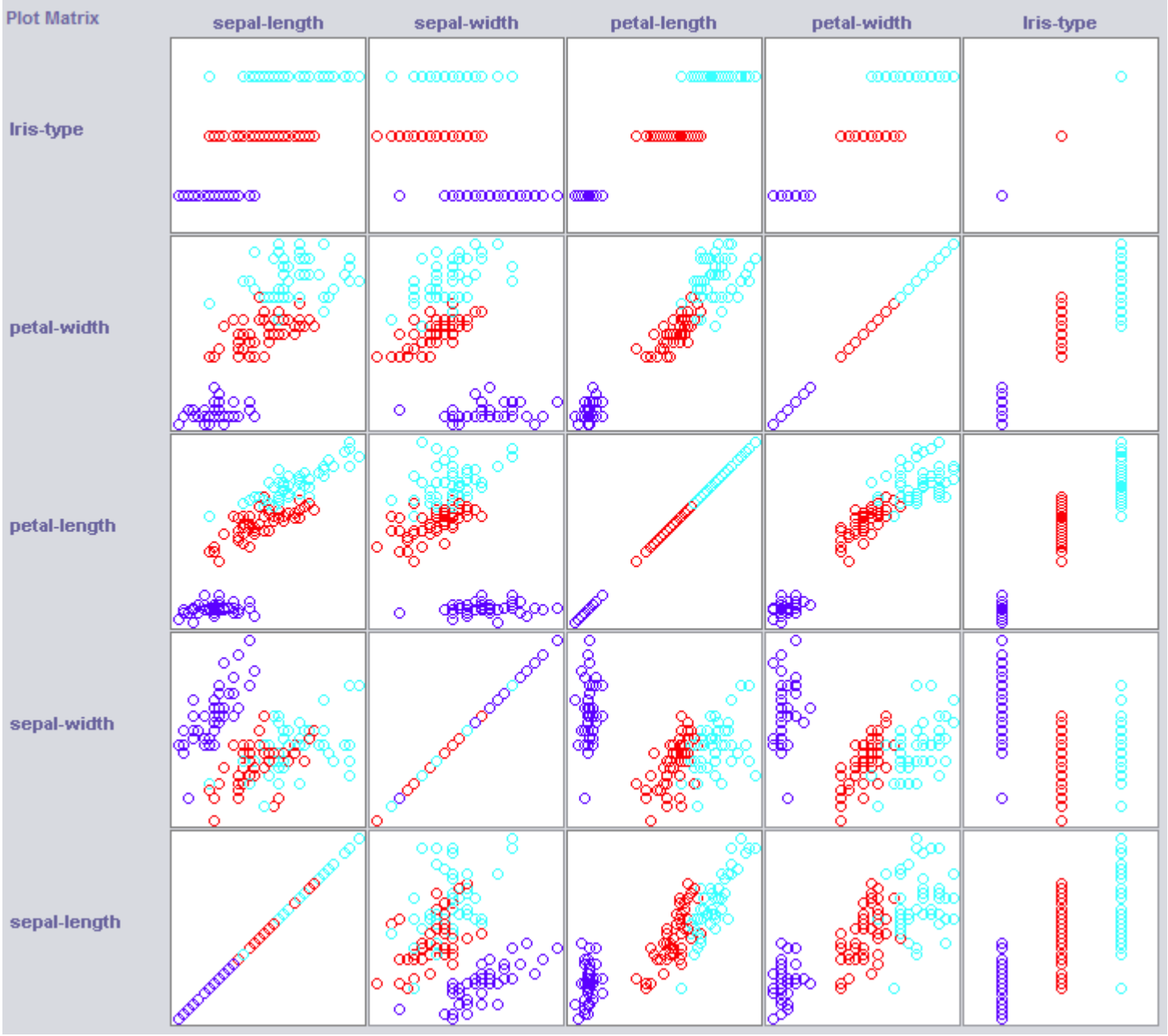
La clase más grande en el conjunto te da una tasa correcta del 34%. (50 de 149)



Nota: El ZeroR realiza alrededor del 30%. Esto se debe a que nos dividimos aleatoriamente en un conjunto de pruebas y trenes. El conjunto más grande en el conjunto de trenes, por lo tanto, será el más pequeño en el conjunto de prueba. Hacer un buen set de prueba / tren puede valer la pena

Tener una idea de los datos. Entrenamiento Naive Bayes y kNN

Para construir un buen clasificador, a menudo necesitaremos tener una idea de cómo se estructuran los datos en el espacio de características. Weka ofrece un módulo de visualización que puede ayudar.



Algunas dimensiones ya separan bastante bien las clases. El ancho de los pétalos ordena el concepto con bastante claridad, en comparación con el ancho de los pétalos, por ejemplo.

La capacitación de clasificadores simples también puede revelar bastante sobre la estructura de los datos. Por lo general, me gusta usar Nearest Neighbor y Naive Bayes para ese propósito. Naive Bayes asume independencia, su buen desempeño es una indicación de que las dimensiones en sí mismas contienen información. k-Nearest-Neighbor funciona asignando la clase de las k instancias más cercanas (conocidas) en el espacio de características. A menudo se usa para examinar la dependencia geográfica local, lo usaremos para examinar si nuestro concepto está definido localmente en el espacio de características.

```
//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 se desempeña mucho mejor que nuestra línea de base recién establecida, indicando que las características independientes contienen información (¿recuerda el ancho de los pétalos?).

1NN también tiene un buen desempeño (de hecho, un poco mejor en este caso), lo que indica que parte de nuestra información es local. El mejor rendimiento podría indicar que algunos efectos de segundo orden también contienen información *(si x e y que clase z)* .

Juntándolo: entrenando un árbol

Los árboles pueden construir modelos que funcionan en características independientes y en efectos de segundo orden. Así que podrían ser buenos candidatos para este dominio. Los árboles son reglas que están juntas, una regla divide las instancias que llegan a una regla en subgrupos, que pasan a las reglas según la regla.

Los Tree Learners generan reglas, los encadenan y dejan de construir árboles cuando sienten que las reglas se vuelven demasiado específicas para evitar el sobreajuste. *El ajuste excesivo significa construir un modelo que es demasiado complejo para el concepto que estamos buscando. Los modelos sobre ajustados tienen un buen desempeño en los datos del tren, pero mal en los nuevos datos*

Utilizamos J48, una implementación JAVA de C4.5, un algoritmo popular.

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

El aprendiz de árbol entrenado con la mayor confianza genera las reglas más específicas y tiene el mejor rendimiento en el conjunto de pruebas; al parecer, la especificidad está justificada.

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

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

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

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

Nota: ambos alumnos comienzan con una regla sobre el ancho de los pétalos. ¿Recuerdas cómo notamos esta dimensión en la visualización?

Lea Una introducción a la clasificación: Generando varios modelos usando Weka en línea: https://riptutorial.com/es/machine-learning/topic/8649/una-introduccion-a-la-clasificacion--generando-varios-modelos-usando-weka

Creditos

S. No	Capítulos	Contributors
1	Empezando con el aprendizaje automático.	Aquib Javed Khan, Community, Dr. Cool, Drew, John Syrinek, Nikos Tavoularis, Piyush, Semih Korkmaz
2	Aprendizaje automático utilizando Java	Arnab Biswas, Patel Sunil
3	Aprendizaje profundo	Martin W, Minhas Kamal, orbit, Thomas Pinetz
4	Aprendizaje supervisado	draco_alpine, L.V.Rao, Martin W, Masoud, plumSemPy, quintumnia, shadowfox
5	Comenzando con Machine Learning utilizando Apache spark MLib	Malav
6	El aprendizaje automático y su clasificación.	Sirajus Salayhin
7	Métricas de evaluación	DJanssens, Franck Dernoncourt, Sayali Sonawane, ScientiaEtVeritas
8	Perceptron	granmirupa, Martin W, Panos
9	Procesamiento natural del lenguaje	quintumnia
10	Redes neuronales	CLDSEED, dontloo, Dr. Cool, Franck Dernoncourt
11	Scikit Learn	Masoud, RamenChef, Sayali Sonawane
12	SVM	Alekh Karkada Ashok
13	Tipos de aprendizaje	Martin W
14	Una introducción a la clasificación: Generando varios modelos usando	S van Balen

	Weka