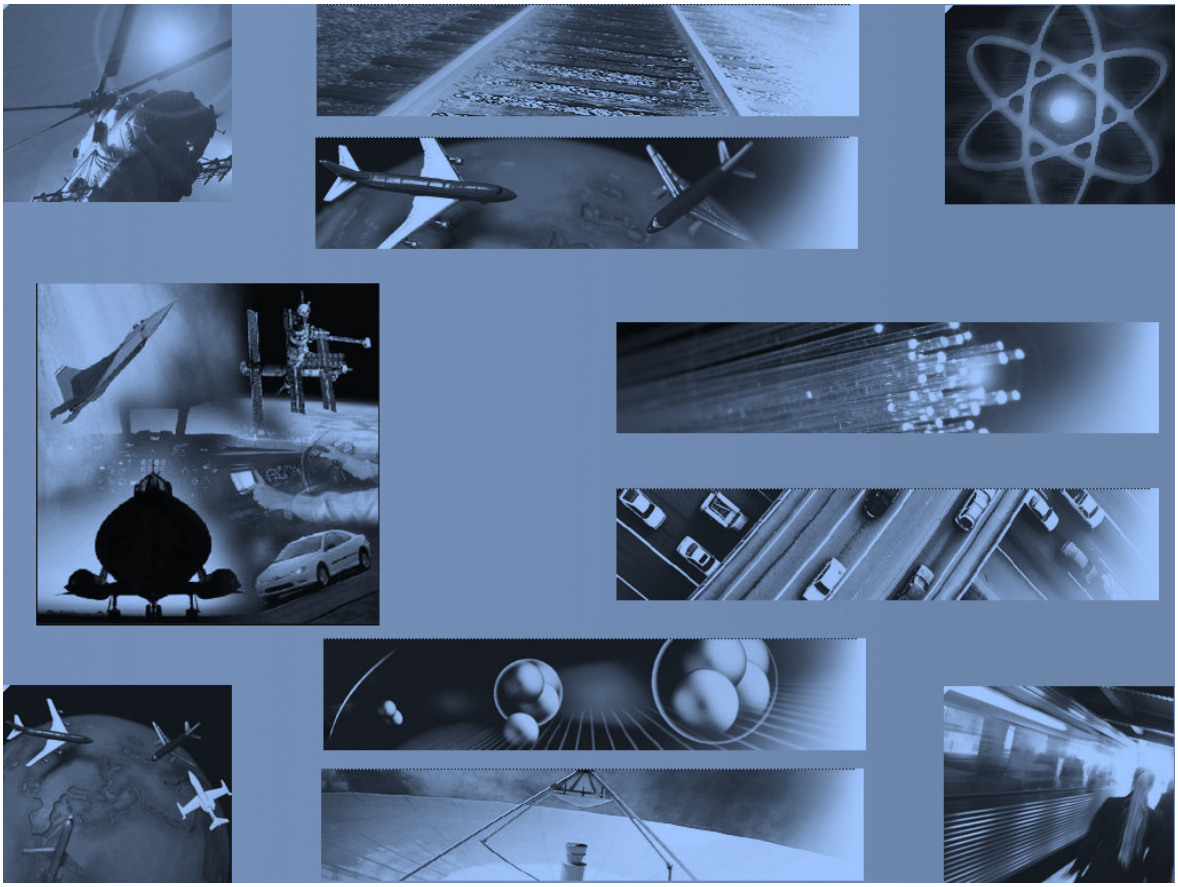
EJERCICIOS SEMANALES DE TIEMPO REAL



GRADO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA

C makes it easy to shoot yourself in the foot;

C++ makes it harder, but when you do, it blows your whole leg off

Bjarne Stroustrup (circa 1986)

EJERCICIOS DE INTELIGENCIA ARTIFICIAL, APRENDIZAJE MÁQUINA

Machine learning, o aprendizaje máquina, en una rama de la inteligencia artificial que desarrolla algoritmos, basados en técnicas estadísticas en su mayoría, que generalizan grandes conjuntos de datos. Esta capacidad de generalización permite que los algoritmos extraigan conclusiones, más o menos acertadas, cuando se les pide que clasifiquen nuevos datos. *Machine learning* es una de las muchas ramas de la Inteligencia Artificial, como redes neuronales, algoritmos genéticos, sistemas basados en lógica difusa, y un largo etcétera. Todos estos algoritmos funcionan en dos etapas: entrenamiento y producción. Primero es necesario utilizar un conjunto de datos para entrenar y preparar el algoritmo, y posteriormente se utiliza para realizar estimaciones.

Existen dos familias de algoritmos encuadrados en *Machine learning*: aprendizaje supervisado y no supervisado. En ambos casos se dispone del conjunto de datos que se quiere generalizar, pero en el primer caso se dispone además de las categorías a que pertenecen dichos datos, mientras que en el segundo no. En el primer caso se quiere ser capaz de estimar la categoría a que pertenece un dato nuevo, mientras que en el segundo se quiere agrupar los datos en conjuntos "con características similares".

Se plantea un ejercicio para implementar un algoritmo de aprendizaje supervisado, el K-Vecinos Cercanos (*K-Nearest Neighbours*, kNN), sobre el conjunto de datos del fichero "*flores.csv*" del Aul@ Virtual. Este fichero contiene los datos de la flor de Iris[1], almacenados en formato CSV (*comma separated values*). Cada línea contiene 4 datos numéricos (longitud y anchura del sépalo, longitud y anchura de los pétalos) y el nombre de la especie de Iris a que corresponden los valores, separados por comas.

Se deja como ejercicio la implementación del algoritmo K-Medias (*K-Means*), otro de los algoritmos clásicos de aprendizaje no supervisado.

IMPLEMENTACIÓN DE KNN, K-VECINOS CERCANOS

El algoritmo kNN es único dentro de los algoritmos de aprendizaje supervisado porque no generaliza el conjunto de datos de entrenamiento, sino que lo utiliza al completo para realizar una estimación. Es el algoritmo más simple de todos. Básicamente, consiste en buscar las 'k' muestras de entrenamiento que se encuentra "más cerca" del valor que queremos clasificar, y luego comprobar cuál es la categoría que más se repite. En caso de empate, se puede escoger la que se encuentra "más cerca" del valor. Otra forma de decidir la categoría es escoger aquella que, de media, tiene los puntos más cercanos. En cuanto a la forma de calcular la distancia, lo habitual es utilizar la distancia Euclídea (raíz cuadrada de la suma de los cuadrados de todas las características), ya que es extrapolable al espacio multidimensional. Por ejemplo, en el caso de Iris tenemos 4 dimensiones (longitud y anchura de pétalo y sépalo). Aunque existen otras formas de calcular la distancia entre muestras. La definición de distancia que se utiliza tiene un gran impacto sobre los resultados del algoritmo.

Para poder comprobar posteriormente que el algoritmo realizar buenas estimaciones, se recomienda al alumno extraer una fila de cada especie (virginica, setosa y versicolor), de forma que el algoritmo disponga de 49 muestras de cada especie en lugar de 50. Una vez "entrenado" el algoritmo (realmente, kNN no se entrena), podrá utilizar las muestras extraídas para comprobar que efectivamente realiza

[1] https://es.wikipedia.org/wiki/Iris_flor_conjunto_de_datos

correctamente la estimación. Este programa tiene ya cierta complejidad, así que recomiendo resolverlo por partes:

1. Defina una clase `C_Datos_Iris` para almacenar los datos de cada muestra, un constructor, métodos de E/S (`operator<<` y `operator>>`), y una función `distancia()` para calcular la distancia Euclídea al objeto de la clase `C_Datos_Iris` que se le pasa como parámetro.
2. Haga un menú en la función `main()` para elegir tres opciones: cargar datos (solicitará nombre del fichero), realizar estimación (solicitará valores para cada una de las 4 características de una flor Iris: longitud y anchura de pétalos y sépalos) y la cantidad de vecinos a utilizar, o terminar el programa. Asegúrese de que hasta que el usuario no haya cargado datos, no pueda realizar estimaciones.
3. Haga una función para leer los datos del fichero CSV y crear (y retornar) un `vector<C_Datos_Iris>` con ellos. Añada también una función para imprimir el contenido de este vector, y poder comprobar que efectivamente tiene todos los datos correctamente almacenados.
4. Haga una función `ejecutarKNN()` a la que se le pasa como argumentos el vector con las muestras de entrenamientos, los valores de la nueva muestra y la cantidad de vecinos que va a buscar, y retorna un `std::string` que representa la categoría de dicha muestra. Utilice como criterio la categoría más repetida y, en caso de empate, la más cercana de entre las más cercanas.

Encontrar los 'k' vecinos más cercanos consiste en calcular la distancia de la muestra nueva a todas las muestras del vector de entrenamiento, ordenarlas de menor a mayor, y buscar entre los 'k' primeros elementos la especia que se repite más veces. Utilice un `map<string, int>` para almacenar el número de veces que se repite cada especie (recuerde el código del histograma). Y pruebe el código conforme los va desarrollando. No espere a tenerlo terminado.
Para que se haga una idea, todo el código ocupa en mi solución unas 150 líneas de C++.

IMPLEMENTACIÓN DE K-MEANS

El algoritmo de K-Medias es un poco más complejo de entender e implementar. El siguiente vídeo en inglés resume su funcionamiento, que es iterativo: https://www.youtube.com/watch?v=zHbxbb2ye3E. Como puede verse en el vídeo, el usuario suministra los datos y el número de grupos o *clusters* ('k') en que quiere que K-Medias reparta las muestras suministradas. El algoritmo elige un centro para cada grupo (denominado *centroide*), y luego asigna cada muestra al grupo del centroide "más cercano". Una vez repartidas las muestras, calcula el nuevo centroide que le correspondería al grupo y vuelve a asignar todas las muestras. Y así se repite hasta que en una iteración no se realice ningún cambio, es decir, no se cambie el grupo al que pertenece ninguna de las muestras. Como en el caso del kNN, la distancia entre muestras se calculará utilizando la distancia Euclídea. En cuanto a la estimación inicial de los centroides de los 'k' grupos, aunque existen varios algoritmos, básicamente se aplican dos:

- Se eligen 'k' muestras al azar.
- Se realizan 'k' agrupaciones al azar, se calculan sus centroides, y se repite el algoritmo.

También existen varias formas para calcular el nuevo centroide a partir de los grupos, pero lo habitual es que el punto se calcule como la media aritmética de cada uno de los valores de las muestras asignadas al grupo.

CONSEJOS IMPLEMENTACION

Es importante notar que (1) el criterio que se aplica es el de homogeneidad de las características de las muestras que se asignan a cada grupo, y no repartir equitativamente las muestras entre grupos; (2) es un algoritmo iterativo que puede tardar mucho tiempo en converger a una solución; (3) la distribución de los grupos está fuertemente condicionada por la elección inicial de sus centroides; (4) los grupos calculados pueden no ser óptimos, por lo que es habitual ejecutar varias veces K-Medias con distintos valores iniciales para los centroides de los grupos, e incluso solicitando la creación de distinto número de grupos. Tenga en cuenta que es un algoritmo de aprendizaje no supervisado: las muestras proporcionadas para el entrenamiento no están categorizadas; no se sabe nada de ellas, tan solo se tienen sus valores. Y el algoritmo intenta realizar agrupaciones en base a la "similitud" entre los valores de las muestras.

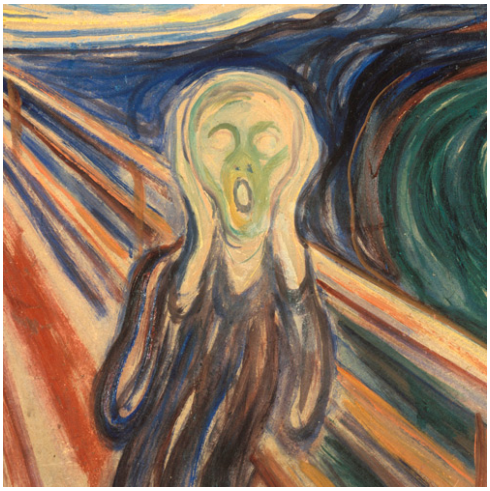
K-Means es un algoritmo no óptimo porque el problema que resuelve es, en general, muy complejo y no abordable computacionalmente cuando el número de muestras es muy grande. Por ejemplo, piense que la solución óptima requeriría calcular las distancias de todos los elementos entre sí, para luego agrupar aquellos que estuvieran más cerca. Y esta operación para un conjunto de datos de tan solo 1000 muestras requeriría 500000 operaciones. Se estima que el arroz (la planta) tiene unos 400 millones de pares de bases químicas (los componentes fundamentales del ADN) y entre 40000 y 60000 genes. Con "tan solo" 40000 genes, el número de distancias que habría que calcular es superior a ochocientos millones.

Piense además que todos los métodos de *machine learning* intentan generalizar el conocimiento que se encuentra en los datos de las muestras, de forma que cuántos más valores de cada muestra se tenga mejor, pero no son infalibles. Dependen mucho de lo representativos que sean dicho valores. Dicho de otra forma, no podemos elegir características cualesquiera de las muestras, sino que hay que intentar seleccionar las que las caracterizan mejor. Aún así, se pueden equivocar. Por ejemplo, la altura, peso y

longitud del cabello pueden parecer buenas características para determinar si una persona es hombre o mujer, pero es cierto que hay 'casos límite' (como mujeres muy altas con pelo corto) que con dichas características podrían no ser clasificados correctamente.

Las soluciones vienen a continuación.

Consultar solo si ya está un poco desesperad@, o para comparar soluciones



IMPLEMENTACIÓN DE KNN, K-VECINOS CERCANOS

main.cpp

```cpp
#include <iostream>
#include <fstream>

#include "kNN.h"

int main() {
   std::cout << "Hello, World!" << std::endl;
   std::ifstream f {"data.csv"};
   kNN mi_knn {f};
   f.close();
   //std::cout << mi_knn;

   //Dato prueba {5.9,3,5.1,1.8, "adios"}; //virginica
   //Dato prueba {5.1,3.5,1.4,0.2, "adios"}; //setosa
   Dato prueba {5.6,2.7,4.2,1.3, "adios"}; //versicolor
   if (mi_knn.clasificar(prueba,5))
      std::cout << prueba << '\n';
   return 0;
}
```

knn.h

```cpp
#ifndef KNN_H
#define KNN_H

#include <iostream>
#include <string>
#include <vector>

//sepal_length,sepal_width,petal_length,petal_width,species

struct Dato {
   double sepal_l, sepal_w, petal_l, petal_w;
   std::string specie;
   double distancia (const Dato &d) const;
};

std::ostream& operator<<(std::ostream& os, const Dato &d);

class kNN {
   std::vector<Dato> datos;
public:
   kNN (std::istream &is);
   friend std::ostream& operator<<(std::ostream& os, const kNN &k);
   bool clasificar(Dato &d, const int vecinos);
};

#endif //KNN_H
```

Knn.cpp

```
#include "kNN.h"

#include <string>
#include <vector>
#include <iostream>
#include <cmath>
#include <utility>
#include <algorithm>
#include <iterator>
#include <map>

double obtener_num (const std::string &s, const bool ultimo) {
   static int p_ini = 0;
   int p_fin;
   double n;
   p_fin = s.find(",", p_ini);

   if (p_fin != std::string::npos) {
      std::string num = s.substr(p_ini, p_fin-p_ini);
      n = std::stod(num);
      //std::cout << num << '[' << p_ini << ", " << p_fin << ") => " << n << '\n';
   }
   p_ini = ultimo? 0 : p_fin+1;
   return n;
}

kNN::kNN(std::istream &is) {
   int tam;
   is >> tam;
   //std::cout << tam << '\n';
   datos.reserve(tam);
   Dato d;
   for (int i=0; i<tam; ++i) {
      std::string linea;
      is >> linea;
      //std::cout << '"' << linea << "\"\n";
      d.sepal_l = obtener_num(linea, false);
      d.sepal_w = obtener_num(linea, false);
      d.petal_l = obtener_num(linea, false);
      d.petal_w = obtener_num(linea, true);
      int p = linea.rfind(",");
      d.specie = linea.substr(p+1);
      datos.push_back(d);
   }
}

std::ostream& operator<<(std::ostream& os, const kNN &k) {
   for (const auto &d : k.datos) {
      os << d << '\n';
   }
   return os;
}

std::ostream& operator<<(std::ostream& os, const Dato &d) {
```

```
    os << d.specie << ": [SL="<<d.sepal_l << ", SW=" << d.sepal_w << "]  (PL=" << d.petal_l<< ", PW=" << d.petal_w << ')';
    return os;
}

double Dato::distancia(const Dato &d) const {
    double pl = petal_l-d.petal_l, pw=petal_w-d.petal_w;
    double sl = sepal_l-d.sepal_l, sw=sepal_w-d.sepal_w;
    return sqrt(pl*pl + pw*pw + sl*sl + sw*sw);
}

bool kNN::clasificar(Dato &d, const int vecinos)   {
    if (!vecinos & 1) return false; // no es impar!
    std::vector<std::pair<Dato*, double>> distancias;
    distancias.reserve(datos.size());
    for (int i=0; i<datos.size(); ++i) {
        distancias.emplace_back(&datos[i], datos[i].distancia(d));
    }
    std::sort(std::begin(distancias), std::end(distancias),
            [](std::pair<Dato*, double> a, std::pair<Dato*, double> b) { return a.second < b.second; }
    );

    std::map<std::string, int> especies;
    for (int i=0; i<vecinos; ++i) {
        ++especies[distancias[i].first->specie];
        //std::cout << *distancias[i].first << '\n';
    }
    auto res = std::max_element(std::begin(especies), std::end(especies),
                        [](std::pair<std::string, int> a, std::pair<std::string, int> b) { return a.second < b.second; }
    );

    //std::cout << "RESULTADO: " << res->first << '\n';
    d.specie = res->first;

    return true;
}
```