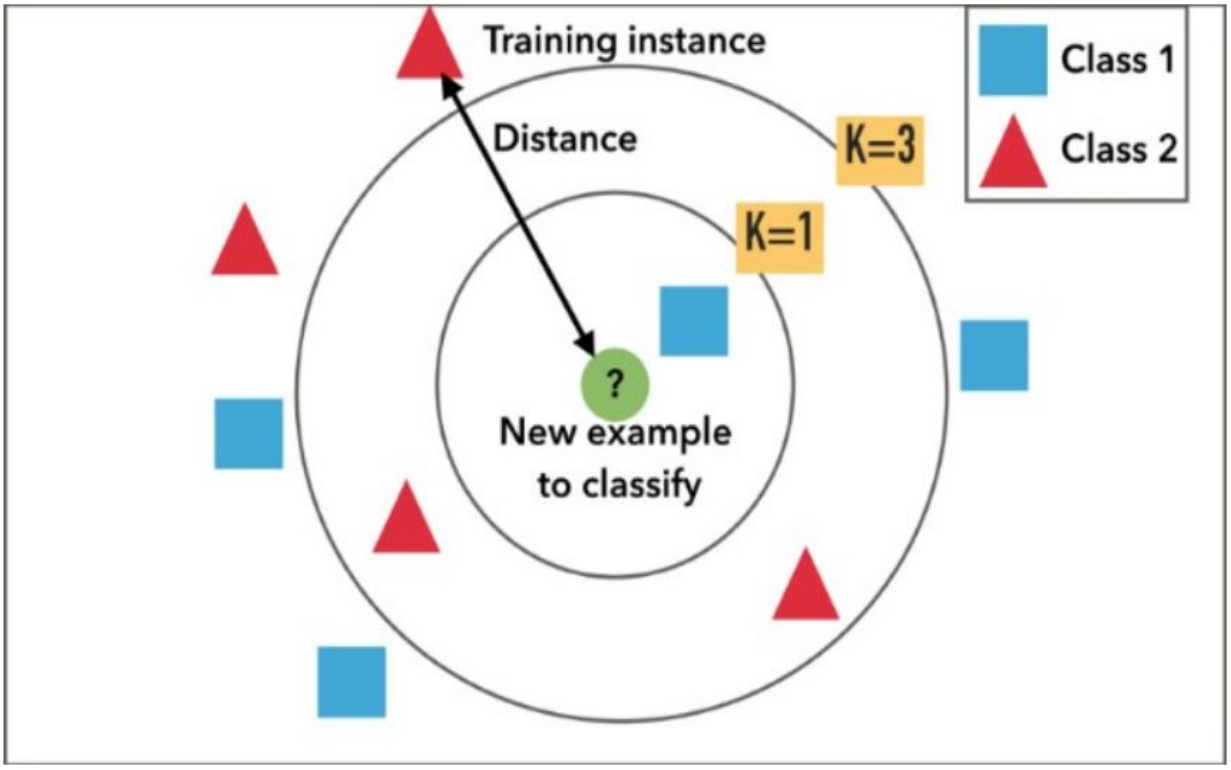


Práctica 1- Problema del Aprendizaje de Pesos en Características

3º Computación y sistemas inteligentes

Adrián Jesús Peña Rodríguez

DNI: 45604927K

Correo: adrianprodri@correo.ugr.es

Prácticas Grupo 1, Lunes 17:30-19:30

KNN Simple, RELIEF y Búsqueda local

1.- Descripción del problema **2**

2.- Descripción general de los algoritmos RELIEF y BL **3**

3.- Pseudocódigo del método de búsqueda **6**
Búsqueda local: 6

4.- Pseudocódigo de los algoritmos de comparación **8**
KNN sin pesos 8
RELIEF 8

5.- Explicación del procedimiento considerado para desarrollar la práctica **9**
Implementación KNN 9
Parámetros modificables 9

6.- Experimentos realizados **10**
1.- Parámetros usados 10
2.- Dataset usados 10
3.- Análisis de los resultados 11

7.- Biografía **13**

1.- Descripción del problema

El problema que hemos elegido es el APC, o lo que es lo mismo el problema de aprendizaje de pesos en características.

Consiste en diseñar un clasificador el cual mediante la asignación de unos determinados pesos se consigue que este mejore en eficiencia y fiabilidad. Para diseñar este clasificador son necesarias dos tareas, aprendizaje y validación.

Los datos a tratar por nuestro clasificador se dividen en dos grupos:

- Entrenamiento: Utilizado para que el clasificador aprenda,
- Prueba: Se usa para validar el aprendizaje de nuestro clasificador. Se calculan el porcentaje de clasificación sobre los ejemplos de este conjunto, los cuales son desconocidos por la tarea de aprendizaje, para así conocer su poder de generalización.

Para determinar con mayor seguridad la fiabilidad de nuestro clasificador, usaremos un método de particiones llamado k-fold cross validation donde nuestra k será 5, esto quiere decir que dividiremos nuestros datos en 5 partes diferentes en las que cada una tendrá un 80% de los datos para aprender y el 20% restante serán para la validación (manteniendo la proporción de las distintas clases que tengan nuestros datasets).

Nuestro clasificador será un K Nearest Neighbors, es un método de clasificación supervisada, que sirve para estimar la función de densidad $F(\frac{x}{C_j})$ de las predicciones x por cada clase C_j.

Una vez conocemos nuestro clasificador, lo que buscaremos es mejorar su capacidad de predicción mediante el uso de una ponderación de características(pesos), la cual iremos obteniendo en las diferentes sesiones de prácticas de la asignatura.

Determinaremos la validez de nuestros algoritmos con la siguiente función de evaluación donde α puede ser cualquier $n \in [0,1]$ aunque por defecto será 0.5:

$$F(W) = \alpha \cdot tasa - clas(W) + (1-\alpha) \cdot tasa - red(W)$$

$$tasa - red = 100 \cdot \frac{nº\ valores\ w_i < 0.2}{nº\ características} \qquad tasa - clas = 100 \cdot \frac{nº\ instancias\ bien\ clasificadas\ en\ T}{nº\ instancias\ en\ T}$$

2.- Descripción general de los algoritmos RELIEF y BL

Para la práctica que nos ocupa en este momento, el objetivo será comparar los resultados de un clasificador K Nearest Neighbors, en adelante KNN, en tres variables:

1. Clasificador sin pesos: Esta variable es básicamente el clasificador KNN sin ningún peso, es decir, tomando todas las características tal cual de los dataset y en base a estas encontrar los K vecinos más cercanos y devolver una predicción.
2. Clasificador con pesos obtenidos mediante RELIEF: para esta variable calcularemos un vector de pesos que multiplicaremos por nuestras características para de este modo priorizar a unas sobre otras o incluso eliminar las que según este algoritmo no son relevantes. Al ser un método greedy, no podemos garantizar que nos mejore la solución sin pesos por lo que veremos su comportamiento en las pruebas que se describen posteriormente.
3. Clasificador con pesos obtenidos mediante búsqueda local: esta variable es la primera que implementaremos que tendrá una heurística, sencilla pero que como veremos nos proporciona unos resultados interesantes.

Cabe decir que la implementación realizada funciona para cualquier K, es decir, es una implementación que va más allá de los pedido en la práctica. Esta implementación que incluye todo sirve para cualquier K vecinos, para cualquier α en la función de evaluación y para cualquier dataset. Como veremos más adelante también se ha ejecutado esta implementación sobre datasets extras.

En el caso RELIEF el K determina también cuantos enemigos y amigos se devuelven respectivamente.

Para la realización de la práctica se ha usado el lenguaje de programación Python, con las librerías:

1. scikit-learn: Para la implementación de nuestro KNN, lo implemente yo mismo completo pero ante la lenta ejecución en determinados datasets decidí usar esta implementación.
2. numpy: Librería para análisis matemático y manejo de datos.
3. scipy: Para tratamiento de archivos ARFF
4. Se importó operator para poder implementar por mi mismo el KNN aunque al final de este uso unas cosas y de la implementación de scikit-learn otras.

5. math: básico para nuestras operaciones.
6. time: Esto nos sirve para medir los tiempos de ejecución.

Para la ejecución de nuestros algoritmos se han usado las mismas particiones para todos ellos, es decir, el método K-fold cross validation con K = 5 y todos ellos se han ejecutado en el mismo orden de particiones.

Para la división de las particiones se ha tenido en cuenta la proporción de clases. Esto es no tener una gran cantidad de una clase en una partición y pocas o ninguna de las otras clases.

Para la representación de nuestras soluciones, tenemos lo siguiente:

1. Pesos: Se usarán para intentar mejorar los resultados de nuestro KNN, para representarlo se ha usado un vector (numpy_array) el cual tendrá misma dimensión que características tiene nuestro conjunto de datos.
2. X_train, X_test: Son dos conjuntos de datos que contienen nuestras características, en X_train tenemos el conjunto de entrenamiento y en X_test, el conjunto con el que evaluaremos el aprendizaje de nuestro KNN que obtuvo con X_train.
3. y_train, y_test: Son los dos conjuntos de etiquetas, y_train contiene las etiquetas de las características de X_train, con este conjunto entrenamos nuestro KNN y hacemos las predicciones cuando cojemos X_test y con y_test comprobamos nuestros aciertos.
4. Para calcular la media de aciertos hacemos la sumatoria de los aciertos de todas las particiones y la dividimos por el número de particiones, hacemos esto mismo para los errores también y para la tasa de reducción.

Nuestra función objetivo es la siguiente:

$$F(W) = \alpha \cdot tasa - clas(W) + (1 - \alpha) \cdot tasa - red(W)$$

$$tasa - red = 100 \cdot \frac{nº\ valores\ w_i < 0.2}{nº\ características} \qquad tasa - clas = 100 \cdot \frac{nº\ instancias\ bien\ clasificadas\ en\ T}{nº\ instancias\ en\ T}$$

Debemos maximizar dicha función , ese es nuestro objetivo y consideraremos una solución mejor que otra según el valor devuelto por esta función.

Puesto que usamos un α de 0.5 quiere decir que damos la misma prioridad a lo bueno que es nuestro KNN para clasificar como a lo sencillo que es (número de características tenidas

en cuenta), tenemos que un clasificador con un 100% de acierto puede ser peor que uno con 90% si este último usa menos características que el primero.

Tanto los valores de α como los valores de K(vecinos) se pueden modificar para cualquier número, la implementación lo permite, para el valor de K se recomienda un valor impar ya que en caso de empate el comportamiento es indeterminado y debido a esto es preferible usar un impar para evitar empates.

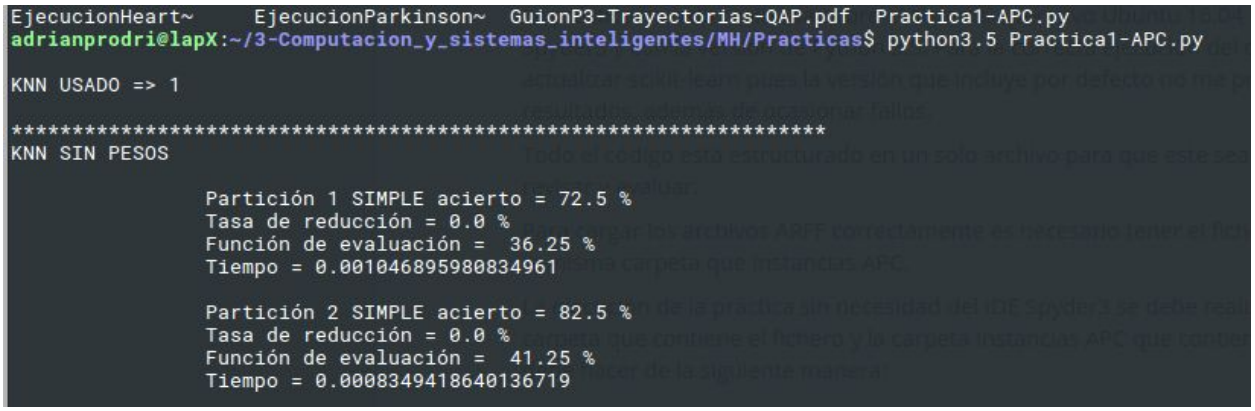
Estos tres algoritmos se han ejecutado sobre tres datasets obligatorios que son, parkinson, ozone y spect-heart. Además de estos se ha ejecutado sobre otros tres dataset que son cáncer de pulmón, diagnóstico de diabetes y clasificación de la flor de iris.

La práctica ha sido realizada sobre el sistema operativo Ubuntu 16.04 LTS con el IDE Spyder3 y con la versión de Python 3.5. Para la correcta ejecución del código es preciso actualizar scikit-learn pues la versión que incluye por defecto no me proporcionaba buenos resultados, además de ocasionar fallos.

Todo el código está estructurado en un solo archivo para que este sea más cómodo de revisar y evaluar.

Para cargar los archivos ARFF correctamente es necesario tener el fichero de la práctica en la misma carpeta que Instancias APC.

La ejecución de la práctica sin necesidad del IDE Spyder3 se debe realizar situándose en la carpeta que contiene el fichero y la carpeta Instancias APC que contiene los datasets, se debe hacer de la siguiente manera:



3.- Pseudocódigo del método de búsqueda

Búsqueda local:

# Inicializamos los contadores necesarios para nuestro algoritmo:
vecinos_generados_tope = 15000
varianza = 0.3
n_caracteristicas = caracteristicas de nuestro X_train
tope = 20 * n_caracteristicas
i = 0
media, media_r, media_f, tiempo_medio, i = 0
##

Empezamos el proceso para cada una de las particiones, este primer bucle se ejecutará en nuestro caso 5 veces ya que haremos 5 particiones distintas:
Para cada i dentro de nuestro numero de particiones (5):
    # Dividimos los datos e iniciamos contadores
    Dividimos nuestro dataset
    vecinos_generados, generadas, media_e = 0
    time_inicial
    # Estaremos intentando mejorar hasta llegar a uno de los topes impuestos en la
    # práctica, aquí exploramos el entorno de soluciones
    mientras vecinos_generados < vecinos_generados_tope y generadas < tope:
        # Generamos la mutación del vector de pesos
        generamos un Zi para mutar nuestros pesos
        # Generamos las componentes del vector a mutar
        ind = generamos la lista de pesos a mutar(aleatoria)
        # Se ejecutará hasta llegar al break o hasta mutar todas las características
        Para cada j en ind:

actual = guardamos peso actual

Mutamos la componente según ind sumandole Zi

Normalizamos y colocamos a cero valores por debajo de 0.2

Asignamos w a nuestras características

Guardamos la predicción

vecinos_generados += 1

Si la funcion de evaluacion > media_e:

    media_e = f_eva

    # Vuelve a poner a 0 el contador pues este contador si llega al

    # tope quiere decir que llevamos 20*n_características son

    # conseguir mejora y cada vez que

    # encontremos una mejora debemos volverlo a poner a 0

    generadas = 0

    break

si no:

    w = actual

    generadas += 1

Comprobamos con nuestro nuevo vector de pesos la eficiencia de nuestro algoritmo

Aumentamos contadores, medias , i y paramos el tiempo para ver el tiempo de ejecución

##

Como podemos apreciar la idea del algoritmo es la de buscar un Zi que al sumarlo a una de las características elegida aleatoriamente mejorará la función de evaluación. Se estará sumando este Zi a las características hasta que se encuentre una que mejore la función de evaluación y en ese momento saldremos y comenzaremos de nuevo el proceso así hasta llegar a ejecutar la función de evaluación 15000 veces o hasta que no se encuentre mejora después de mutar n_características*20 características.

4.- Pseudocódigo de los algoritmos de comparación

I. KNN sin pesos

Inicializamos los contadores i, tiempo_medio, media, media_f = 0

Para cada número de partición:

Divide el dataset en la partición

Define knn

Ajusta el knn

Obtén la predicción

Imprime resultados

Imprime medias

II. RELIEF

Inicializamos los contadores i, tiempo_medio, media, media_r, media_f = 0

Para cada número de partición:

Divide el dataset en la partición

Define knn

W <- {0, 0, ..., 0}
Para cada ei en T
    Buscar el enemigo más cercano de ei: ee
    Buscar el amigo más cercano de ei: ea
    W = W + |ei – ee| – |ei – ea|
    wm = máximo (W)
    Para cada wi en W
        Si wi < 0 entonces
            wi = 0
        Si no
            wi = wi / wm

Ajusta el knn con los pesos obtenidos

Obtén la predicción

Imprime resultados

Imprime medias

Buscar al enemigo y al amigo más cercano es básicamente buscar a todos los vecinos del punto, ordenarlos según la distancia de menor a mayor y devolver el que corresponda.

5.- Explicación del procedimiento considerado para desarrollar la práctica

I. Implementación KNN

En un primer momento decidí implementar el algoritmo KNN por mi mismo y usar mi implementación, para ello me valí de teoría acerca de este método que recopile por internet y di con una implementación que sin ser igual me era de mucha utilidad y base mi código en ella. Más adelante en la biografía colocaré la fuente de dicha implementación.

Deje de usar esta implementación porque al llegar al algoritmo BL era imposible terminar la ejecución del dataset de ozono y del corazón pues para el del parkinson tarda como una hora en terminar para los otros era algo inviable. Debido a esto decidí conservar la implementación pero usar para las predicciones la implementación de scikit-learn.

Aún así la búsqueda de amigos y enemigos la hago básicamente con mi implementación modificada para devolver amigos u enemigos. Debido a esto es por lo que el algoritmo RELIEF da peores resultados en tiempo que BL cuando en principio es menos costoso.

II. Parámetros modificables

Esta práctica la enfoque no solo a ser un 1-NN si no a poder ser un KNN completo y como tal podemos modificar la k y poner el valor que queramos de preferencia un número impar para evitar empates.

También podemos modificar α a nuestro gusto para que nuestro algoritmo BL de más prioridad a acertar o a reducir o a ambos como es el caso por defecto.

A la hora de ejecutar mi práctica, esta ha sido probada con Python3.5 en el IDE Spyder3 en un Ubuntu 16.04 LTS al cual le actualice la librería scikit-learn que trae por defecto.

Para ejecutar la práctica está explicado en el punto 2.

El proceso que seguí para implementarlo todo fue leer los seminarios y poco a poco ir sacando el código a base de ensayo y error. La única documentación que miré fue una implementación de un KNN que colocaré en la biografía.

6.- Experimentos realizados

1.- Parámetros usados

Para la realización de los experimentos he considerado $\alpha = 0.5$ como se indica en la práctica, en cuanto a las semillas solo han sido usadas para el algoritmo BL pues requería generar aleatorios y para este algoritmo se ha usado una semilla diferente para cada partición como se indica en el guión de prácticas. La semilla usada es el x de Partición x.

En el algoritmo RELIEF podemos elegir la cantidad de amigos y enemigos que se nos devolverán, esto por defecto es 1 ya que va en función de K

2.- Dataset usados

Se ha ejecutado el algoritmo sobre tres dataset obligatorios que describiré a continuación y posteriormente describiré los dataset extras que he evaluado:

- Parkinsons: contiene datos que se utilizan para distinguir entre la presencia y la ausencia de la enfermedad de Parkinson en una serie de pacientes a partir de medidas biomédicas de la voz.
    - Consta de :
    - 195 ejemplos
    - 23 atributos (clase incluida)
    - 2 clases. En las particiones se ha respetado el equilibrio de clases en la medida de lo posible
    - Atributos: Frecuencia mínima, máxima y media de la voz, medidas absolutas y porcentuales de variación de la voz, medidas de ratio de ruido en las componentes tonales, ...

- Ozono: es una base de datos para la detección del nivel de ozono según las mediciones realizadas a lo largo del tiempo.
    - Consta de:
        - 320 ejemplos (seleccionados de los 2536 originales, eliminando los ejemplos con valores perdidos y manteniendo la distribución de clases al 50%)
        - 73 atributos (clase incluida)
        - 2 clases (día normal o con una alta concentración de ozono)
        - Atributos: Predicción del nivel de ozono local, Temperatura, Radiación Solar, Velocidad del viento...
- Spectf-heart: contiene atributos calculados a partir de imágenes médicas de tomografía computerizada (SPECT) del corazón de pacientes humanos. La tarea consiste en determinar si la fisiología del corazón analizado es correcta o no.
    - Consta de:
        - 267 ejemplos
        - 45 atributos enteros (clase incluida)
        - 2 clases (paciente sano o patología cardiaca)
        - Atributos: 2 características (en reposo y en esfuerzo) de 22 regiones de interés de las imágenes

Dataset extras:

- Diabetes: dataset orientado a la predicción de padecer diabetes o no, el dataset tiene información acerca de casos en los que el individuo analizado tenía o no diabetes y en base a eso y viendo las características, podemos intentar predecir si un nuevo paciente padecerá o no diabetes.
- Cáncer de pulmón: este dataset contiene información acerca de si con unas determinadas características, un paciente podrá padecer cáncer de pulmón o si por el contrario no. La predicción que podemos hacer gracias a este dataset es porque contiene miles de ejemplos de pacientes a los que se les diagnóstico o no cáncer de pulmón o no.
- Clasificación flor de iris: en este caso el dataset nos proporciona la información necesaria para determinar si una flor es de un tipo u otro de entre tres tipos. La tasa de acierto en este caso es bastante alta en todos los casos como veremos.

3.- Análisis de los resultados

El análisis es válido para todos los datasets pues el comportamiento es el mismo para todos:

- Vemos que con el algoritmo KNN sin pesos y usando un K = 1 es decir un vecino, tenemos que los valores de aciertos son “aceptables“, pero debemos de saber que puesto que aquí no hay ninguna ponderación, no podemos saber si realmente nuestro clasificador es bueno acertando o no pues como vamos depende en gran medida de la calidad y cantidad de datos y puesto que se varía bastante de una partición a otra, podemos determinar que no es una buena solución, hay demasiado margen entre el mínimo de acierto y el máximo. No tiene tasa de reducción pues aquí aún no estamos considerando ninguna ponderación de pesos que nos puede quitar alguna característica.
- En cuanto al algoritmo RELIEF, tenemos que ha habido una pequeña mejora pero aún así no lo suficientemente grande como para considerarlo una buena solución. Esto ocurre porque este algoritmo es un greedy y como tal coje la mejor opción cada momento sin pensar en el futuro y esto ocasiona que lo que en un principio es la mejor solución, posteriormente sea un lastre para la solución global. Para ir mejorando la solución según su criterio, va aplicando la fórmula vista en los seminarios y para ella se basa en la búsqueda de amigos y enemigos.
- Por último, tenemos el algoritmo de búsqueda local, el cual sigue más o menos la tasa de aciertos de los anteriores algoritmos pero sin embargo en este algoritmo conseguimos prácticamente los mismo resultados pero con menos características lo cual es muy positivo pues quiere decir que nuestro KNN es más simple lo que mejora la velocidad de ejecución obteniendo los mismos resultados. Esto se debe a que nuestro algoritmo de búsqueda local busca maximizar la función de evaluación y puesto que se pondera de la misma manera los aciertos que la reducción, el algoritmo tenderá a mejorar esta función y debido a esto tenemos que se pueden eliminar características y así mejorar la evaluación de nuestro KNN. En casos tan masivos como el dataset de ozono el cual tiene un número de características muy alto, es muy conveniente nuestro algoritmo BL en lugar de los anteriores pues conseguimos una tasa de acierto muy similar en todos los casos pero somos capaces de eliminar unas cuantas características y esto beneficia globalmente a nuestro KNN.

Podemos ver que en datasets con un alto número de características, es fácil que incluso el algoritmo RELIEF tenga una reducción de características, debido precisamente al alto número de ejemplos y de características.

Cabe señalar que debido a como es el dataset de cáncer de pulmón, se consigue un porcentaje bastante mayor de acierto con el algoritmo RELIEF que con BL o sin pesos, esto probablemente se deba a que realmente las clases estén bastante bien divididas y debido a esto al aplicar la fórmula de error lo que son los casos del borde de la distribución se mejoren en algunos casos respecto a BL y sin pesos.

7.- Biografía

- Transparencias seminario 2.
- Guión de prácticas
- https://www.kdnuggets.com/2016/01/implementing-your-own-knn-using-python.html