UNIVERSIDAD DE GRANADA

METAHEURÍSTICAS

Prácticas Grupo 1, Lunes 17:30-19:30
Enfriamiento Simulado, Búsqueda Local Reiterada y Evolución Diferencial para el Problema del Aprendizaje de Pesos en Características.

Nombre: Adrián Jesús
Apellidos: Peña Rodríguez
Correo: adrianprodri@correo.ugr.es
DNI: 45604927-K
Fecha de entrega: 09/06/18

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 ∈ [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.- Introducción a la práctica

Como sabemos la práctica que nos ocupa tiene como objetivo comparar los resultados de un clasificador K Nearest Neighbors, en adelante KNN, con las tres variables vistas en la práctica 1 además de con los algoritmos que hemos desarrollado en esta práctica 2:

1. Clasificador con todos los pesos a 1: 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 enfriamiento simulado: Para esta variable usaremos una estrategia basada en trayectorias como es enfriamiento simulado. A grandes rasgos esta técnica se usa en problemas de optimización global, debido a esto es una buena técnica para nuestro ejercicio. Este algoritmo intenta encontrar una buena aproximación al valor óptimo de una función en un espacio de búsqueda grande como es el propuesto en los datasets.
4. Clasificador con pesos obtenidos mediante búsqueda local iterativa, este método es un método de la familia de búsqueda por trayectorias múltiples. Básicamente es la aplicación de un algoritmo de búsqueda local a una solución inicial que se obtiene por mutación de un óptimo local previamente encontrado.
5. Clasificador con pesos obtenidos mediante Differential Evolution, este es un método perteneciente a la categoría de algoritmos evolutivos. Al igual que otros algoritmos de esta categoría, la evolución diferencial mantiene una población de soluciones candidatas, las cuales se recombinan y mutan para producir nuevos individuos los cuales serán elegidos de acuerdo al valor de su función fitness. Lo que caracteriza a DE es el uso de vectores de prueba, los cuales compiten con los individuos de la población actual a fin de sobrevivir.

Esta práctica se basa en búsqueda por trayectorias, usaremos este tipo de métodos y los compararemos entre ellos, veremos que su comportamiento es un tanto especial en comparación con los algoritmos vistos hasta ahora, sobre todo con respecto a los evolutivos de la práctica anterior.
Por otro lado también experimentaremos con un algoritmo de la categoría de evolucionales, este es differential evolution (evolución diferencial), este pertenece a la categoría de evolucionales pero con una característica clave dicha anteriormente (vectores de prueba), compararemos sus resultados directamente con los anteriores y veremos cual es su comportamiento.

Para la realización de esta tercera y última práctica hemos usados Python como lenguaje principal con las siguientes 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. time: Esto nos sirve para medir los tiempos de ejecución.

En esta ocasión hemos dividido en varios archivos nuestra práctica para así poder mantener y actualizar el código de una manera más simple, además de ayudar a la corrección de esta.

Los parámetros de ejecución de la práctica son los mismos que en la práctica 1 y 2 y para los nuevos algoritmos se han usado los parámetros que se indican en la práctica como valores por defecto pero pueden cambiarse a conveniencia para así poder investigar de manera más exhaustiva el comportamiento de estos algoritmos.

Para la representación de nuestras soluciones, tenemos lo siguiente:

1. Pesos: Se usarán para intentar mejorar los resultados de nuestro KNN, para representarlo hemos optado por cambiar la representación en numpy por listas de python las cuales nos permiten un manejo más sencillo de los datos, además de una alta velocidad de modificado. Aún así seguimos usando la librería numpy pues nos sirve para multitud de tareas.
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.

En cuanto a operadores comunes tenemos que los tres algoritmos tienen un operador de cálculo de función fitness y un operador de generador de solución. Los otros operadores son específicos de cada algoritmo y por tanto los definiremos en los apartados siguientes.

La práctica ha sido realizada sobre el sistema operativo Elementary OS (basada en Ubuntu 16.04 LTS) con el IDE PyCharm 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.
El código a sido reestructurado en más de un archivo para así poder mantenerlo y actualizarlo de manera más simple.
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:

```
EjecucionHeart~     EjecucionParkinson~  GuionP3-Trayectorias-QAP.pdf  Practica1-APC.py
adrianprodri@lapX:~/3-Computacion_y_sistemas_inteligentes/MH/Practicas$ python3.5 Practica1-APC.py

KNN USADO => 1

*****************************************************************
KNN SIN PESOS

               Partición 1 SIMPLE acierto = 72.5 %
               Tasa de reducción = 0.0 %
               Función de evaluación =  36.25 %
               Tiempo = 0.010046895980834961

               Partición 2 SIMPLE acierto = 82.5 %
               Tasa de reducción = 0.0 %
               Función de evaluación =  41.25 %
               Tiempo = 0.0008349418640136719
```

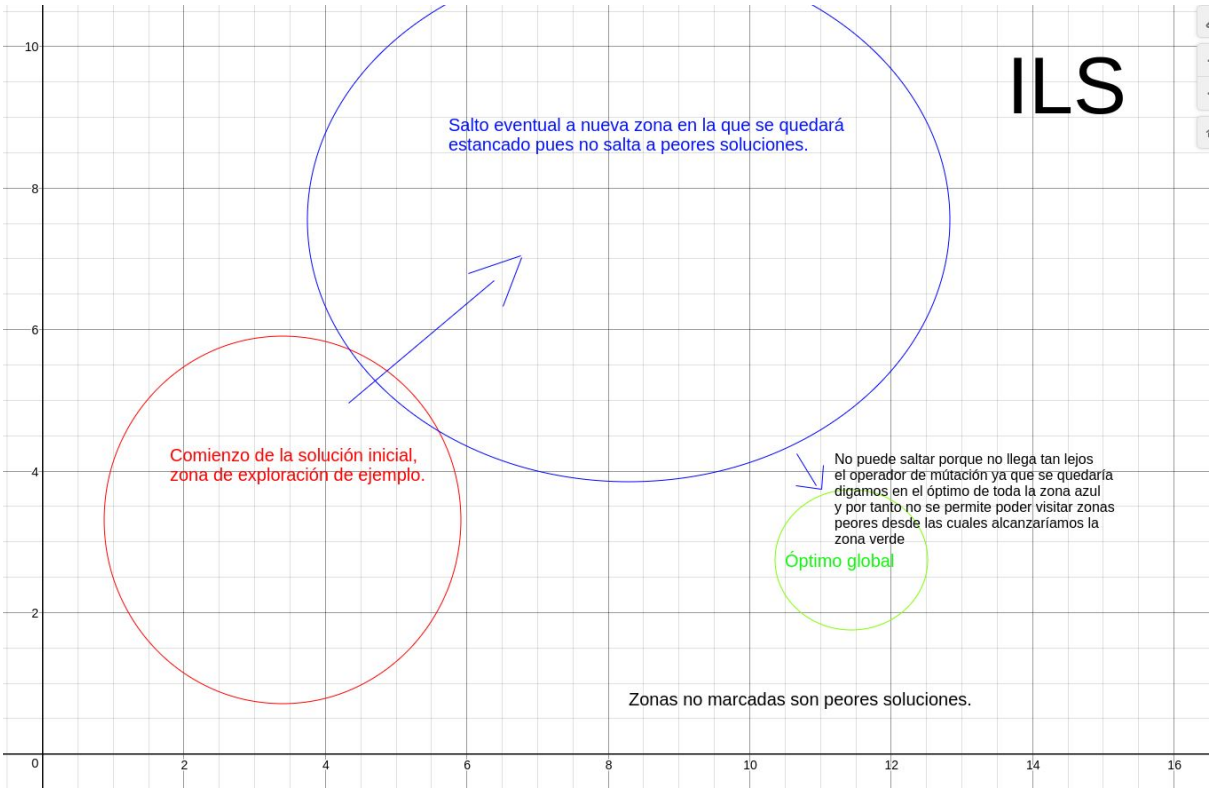
3.- Descripción general de los algoritmos Simulated annealing, ILS y Differential Evolution

Comenzaremos indicando cuál ha sido el esquema de representación de las soluciones, para representar las soluciones se ha usado una lista de python en la que se encuentran todas las características de la solución (característica = peso específico para la solución que ocupa ese lugar).

También tenemos que tener en cuenta que el algoritmo DF habla de genes, esto no deja de ser una nomenclatura diferente pero que viene a ser lo mismo que característica, cada característica de un individuo es un gen de ese individuo.

Por tanto para nosotros, una generación de individuos será una lista de listas del tamaño que corresponda según el número de características y el número de individuos por generación, esto es en cuanto a differential evolution ya que en ILS y en ES no trabajaremos con poblaciones pues en la búsqueda por trayectorias no se trata de evolucionar de población a población sino en digamos comenzar con una única solución y a partir de esta trazar trayectorias a vecinos de esta para así conseguir tener una alta exploración y así en caso de entrar en un óptimo local poder salir a un óptimo global o local más prometedor. Como particularidad tenemos que ILS es capaz de salir de óptimos locales y eventualmente encontrar el global pero esto es por puro azar ya que es en base a una distribución normal, así que puede que lo encontremos o no, lo que está claro es que nos quedaremos con el mejor óptimo encontrado.

En cuanto a ES, tenemos que en un principio el algoritmo tiene una alta tasa de exploración que poco a poco a medida que la temperatura baja se va tornando en explotación. Tiene sus virtudes y desventajas, permite ir a soluciones peores para luego quizás encontrar una mejor, en ILS no tenemos la opción de ir a ninguna solución peor por lo que digamos la exploración corre el riesgo de quedarse en una determinada zona del espacio de búsqueda y no llegar a una nueva zona que aunque en principio puede ser menos prometedora no tiene porque serlo a la larga. Este es un dibujo aclaratorio de lo explicado:



En cuanto a nuestra función objetivo tenemos que es la siguiente:
Nuestra función objetivo es la siguiente:

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

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

Por tanto una solución X será mejor que otra Y si al aplicarle esta función, F(X) > F(Y), a esto lo denominaremos fitness.
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.
Para la realización de la práctica hemos dejado los dataset obligatorios que son Parkinsons, ozone-320 y spect-heart y hemos tenido que eliminar los añadidos a parte excepto el de diabetes debido a que en sus filas contenían errores y caracteres raros que provocan un fallo que impedía su ejecución.

A la hora de generar la solución inicial, usamos un esquema de generación aleatorio con una distribución normal entre un mínimo y un máximo elegidos (por defecto mínimo = 0.0 y máximo = 0.0), además tenemos una cota inferior mediante la cual si un gen del individuo

está por debajo de esta lo colocamos a 0, al igual que una cota superior para colocarle el valor máximo, el pseudocódigo de esto es el siguiente:

```
def generaSolucion:
        generados<-distribución uniforme entre min y max
        for n, i en aleatorios:
                si i < cota-inferior
                        generados<-0.0
                si i > cota-superior
                        generados = max
```

Podemos observar que la cota superior es prescindible pero esta puesta por si se da el caso de que queremos generar entre un min y un max más elevado y queremos que los valores no pasen de 0.0 y la cota superior.

En cuanto a la mutación tenemos que simplemente es una mutación de una característica para así obtener un vecino de la solución que estamos mutando. Esto solo ocurre en ILS y ES, en differential evolution no hay operador de mutación. En ES podemos mutar a una solución peor, en ILS no, esto es lo explicado anteriormente.

Por lo tanto tenemos que los algoritmos funcionan de manera diferente aunque entre ILS y ES hay similitudes pero con DE no hay similitudes, son diferentes por eso son de categorías diferentes.

Luego tenemos los operadores específicos de cada algoritmo, los cuales veremos en el siguiente apartado.

Con esto tenemos descritos los operadores básicos en estas técnicas de búsqueda.

4.- Pseudocódigos y explicación de los algoritmos

En este apartado se describirán los algoritmos desarrollados, así como sus peculiaridades y similitudes.
Comenzaremos con el algoritmo SE:

def simannealing:
    Generamos una solución que actualmente será la mejor
    Iniciamos contador
    Mientras no condiciones de acabado (Podemos salir por no tener exitos en la iteración i por llegar al máximo de evaluaciones) o la temperatura baje un cierto umbral (Esto es lo que puede hacer que no lleguemos al máximo de 15000 evaluaciones):
        Colocamos a cero el numero de exitos
        Aumentamos iteración
        Bucle desde 0 al máximo de vecinos:
            Generamos vecino
            Evaluamos al vecino generado
            Guardamos evaluación actual
            Calculamos diferencias entre las evaluaciones
            Generamos aleatorio (El cual permite que junto con la temperatura y la diferencia nos permite ir a soluciones peores)
            Comprobamos condiciones (Si diferencia es negativa entramos o si se cumple condición de la temperatura y el aleatorio)
                Asignamos a la solución actual el vecino
                Aumentamos éxito
                Si evaluación de vecino mejor que mejor actual:
                    mejor actual = solución actual
                Si el número de éxitos supera el permitido:
                    break

        Enfriamos

    Devolvemos la mejor solución

Como podemos observar en el pseudocódigo y como hemos explicado antes, tenemos que el algoritmo se basa en la generación de una solución aleatoria y a partir de esta comenzar a explorar el espacio de soluciones a raíz de trayectorias que se basan en mutaciones a las características.

El algoritmo enfría la temperatura en base a la iteración en la que estamos, de tal manera que cuando la temperatura es alta es más probable saltar a una solución peor que la actual, es decir, esto favorece la exploración, a medida que enfriamos pasamos de una alta exploración a una alta explotación, lo cual se traduce en un equilibrio bastante bueno.

Puesto que el algoritmo se basa en la metalurgia, al igual que está, es posible llegar a un estado peor que el actual para eventualmente llegar a una solución óptima o al menos intentarlo, en caso de no ser óptimo global, al menos será un local.

Generaremos la temperatura inicial con los parámetros que nos dicen en el guión de prácticas y esta fórmula:

$$T_0 = \frac{\mu \cdot C(S_0)}{-\ln(\phi)}$$

Enfriaremos en base a la fórmula de Cauchy modificado según la iteración en la que estemos (M):

$$\beta = \frac{T_0 - T_f}{M \cdot T_0 \cdot T_f}$$

Y lo haremos como se ve en el pseudocódigo del algoritmo.

Ahora procederemos con ILS, empezaremos explicando su método de búsqueda local, el cual es una búsqueda local como la conocemos solo que ahora estaremos haciendo búsqueda local hasta llegar al máximo de evaluaciones de BL permitidas por iteración, en cada iteración, las cuales como peculiaridad tenemos que son 15 solamente pues en cada una de ellas se aplicará el algoritmo BL con 1000 evaluaciones.

En cuanto al pseudocódigo, tenemos lo siguiente:

```
def ILS:
        Generamos solución actual
        Aplicamos BL a la solución actual
        Mejor solución = solución actual
        iteración = 1

        Mientras iteracion < max_iteraciones (15):
                iteración += 1
                Si fitness(actual) > fitness(mejor_solución):
```

```
                Mutamos actual
                Aplicamos BL a la actual
        Si no:
                Actual = mejor
                Mutamos actual
                Aplicamos BL a la actual

        Si fitness(actual) > fitness(best):
                best = actual

    return best
```

Como vemos el objetivo de este algoritmo se basa en aplicación de BL a la mejor solución de tal manera que no visitaremos nunca soluciones peores a la actual como hemos descrito anteriormente en el apartado de descripción de algoritmos (Apartado 3).

Por último tenemos Differential Evolution, el algoritmo que se diferencia de los anteriores pues es de la categoría de evolutivos, tiene un buen equilibrio entre exploración y explotación siendo teniendo una gran exploración al inicio y una convergencia hacia el final aunque tenemos una componente de descendencia con tres padres aleatorios que permite que se explore muy bien y hacia el final se converja lo suficiente. El método de este algoritmo a diferencia de los anteriores de esta práctica si trabaja con poblaciones.

Como característica principal, este algoritmo se basa en la competición con vectores de prueba de los individuos de las poblaciones para determinar si se quedan o no. Con esto completamos la explicación empezada en el apartado 3.

Su pseudocódigo es el siguiente:

```
def differentialEvolution:
    Generamos población actual
    Mientras no superemos el máximo de evaluaciones:
        Para cada j en el tamaño de la población:
            Creamos lista de candidatos
            Quitamos al índice j de la lista de candidatos
            Generamos tres índices para los padres
            Tenemos los tres padres
            Objetivo = población_actual[j]
            Creamos lista vacía de descendientes
            Para cada k en las características:
                Generamos una cota aleatoria
                Si la cota < Probabilidad de cruce:
                    Aplicamos fórmula en función de la variante de DF que
estemos haciendo
                    Ajustamos si superamos la cota inferior o superior
```

Añadimos a descendientes la característica k de el nuevo individuo sacado a través de la fórmula

sí no:

Añadimos a descendientes la característica k de el individuo actual de la población pues el nuevo no ha superado a este

Calculamos el fitness del descendiente
Calculamos el fitness del objetivo

Si el fitness del descendiente > fitness del individuo actual

Devolver el mejor individuo de la población

5.- Procedimiento seguido para la realización de la práctica

Para la realización de esta práctica hemos estudiado las diapositivas tanto de clase como de las clases de prácticas. También hemos realizado una investigación para la creación de los algoritmos puesto que no terminaba de quedar claro con los pseudocódigos de las transparencias.

6.- Experimentación y explicación de los resultados

Para la realización de los experimentos se han usados los parámetros establecidos en el guión de prácticas.
Se ha ejecutado sobre los tres datasets obligatorios, por falta de tiempo no hemos podido realizar experimentos extras.

A continuación vamos a comentar los resultados obtenidos viendo los resultados obtenidos en las tablas de resultados pero sin hacer una simple enumeración de los resultados.

Viendo las tablas podemos ver una clase de características clave en cada algoritmo, podemos apreciar como ES tiene una buena tasa de reducción y de clasificación sin destacar en ninguna de las 2 pero consiguiendo unos resultados realmente buenos. En cuanto al tiempo de ejecución es más elevado que los otros algoritmos de esta práctica por lo que aumentar las iteraciones para poder conseguir unos resultados mejores en principio no es una buena opción.

El siguiente algoritmo a analizar es ILS el cual de media nos da unos resultados similares a ES pero con un tiempo mucho menor. Como característica destacable podemos apreciar que la tasa de reducción que consigue este algoritmo es muy acusada y debido a esto la tasa de clasificación se resiente. Esto puede ocurrir porque llegando a un óptimo puede ser que no se pueda llegar a una tasa de clasificación que consiga aumentar el fitness lo suficiente como lo hace la reducción masiva de características.

En el siguiente, tenemos el algoritmo DE el cual tiene un comportamiento muy diferente con respecto a sus dos variables, la variante de aleatorios es similar a ILS en cuanto a tasas de reducción pero es capaz de mejorar la tasa de clasificación y por tanto dar una mejor media que el algoritmo ES y ILS, además de su otra otra variable. En cuanto a la variante de elegir al tercer padre como el mejor individuo de la población, tenemos que se consigue unas peores medias, esto puede deberse a que eliminamos parte de la exploración de la variante RAND. Y como ya ocurría con los algoritmos meméticos la variante de elegir el mejor no daba unos resultados mejores a las otras opciones, esto se debe a que también una solución que en un momento X es la mejor, en un momento Y puede ser terriblemente mala o incluso que no hubiese sobrevivido.

Por último, tenemos que computacionalmente el más costoso con diferencia es el algoritmo ES, por lo que podemos determinar que no es el mejor para nada pues tenemos cierto margen para aumentar las iteraciones de los otros algoritmos para así poder conseguir mejores resultados a costa de igual el coste computacional del algoritmo ES. Así que en los otros algoritmos se podrían aumentar las iteraciones y evaluaciones, para igualar el coste computacional a ES y posiblemente obtener mejores resultados a los actuales.