Universidad Nacional
Autónoma de México

Facultad de Ingeniería

Aprendizaje 2019-2

Algoritmos genéticos

M.C. Hugo E. Estrada León

Padilla Herrera Carlos Ignacio

Ramírez Ancona Simón Eduardo

16 de mayo del 2019

Índice

Objetivo

El alumno programará un algoritmo genético con la finalidad de maximizar una función y el alumno conocerá los efectos de la tasa de cruza y mutación.

Introducción

Los Algoritmos Genéticos (AG) pueden usarse en distintas áreas para resolver problemas de búsqueda y optimización. Estos se basan en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin (1859). Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas.

Los principios básicos de los Algoritmos Genéticos fueron establecidos por Holland (1975), y se encuentran bien descritos en varios textos.

En la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagarán en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes “súperindividuos”, cuya adaptación es mucho mayor que la de cualquiera de sus ancestros. De esta manera, las especies evolucionan logrando unas caracteríısticas cada vez mejor adaptadas al entorno en el que viven.

Los Algoritmos Genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor o puntuación, relacionado con la bondad de dicha solución. En la naturaleza esto equivaldría al grado de efectividad de un organismo para competir por unos determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos – descendientes de los anteriores – los cuales comparten algunas de las características de sus padres. Cuanto menor sea la adaptación de un individuo, menor será la probabilidad de que dicho individuo sea seleccionado para la reproducción, y por tanto de que su material genético se propague en sucesivas generaciones.

De esta manera se produce una nueva población de posibles soluciones, la cual

reemplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. A lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, van siendo exploradas las áreas más prometedoras del espacio de búsqueda. Si el Algoritmo Genético ha sido bien diseñado, la población convergerá hacia una solución óptima del problema.

El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el Algoritmo Genético encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al Algoritmo Genético, tanto en rapidez como en eficacia. El gran campo de aplicación de los Algoritmos Genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas brindándoles con los Algoritmos Genéticos.

Desarrollo

Realizamos la implementación del ejercicio 1 que consiste en la programación de un algoritmo genético para maximizar una función. Usamos el lenguaje de programación Python en su versión 2.7

Implementación del programa de algoritmos genéticos

Ejercicio 1

Un grupo de amigos decide invertir su dinero en un producto novedoso. El producto se venderá en 4 tiendas (T1, T2, T3 y T4). Un estudio de mercado ha sido realizado en cada una de las tiendas y han sido establecidas las ganancias medias en función de las inversiones totales. Estos datos se ilustran en la Tabla (1). Para simplificar los cálculos, supondremos que las asignaciones de inversiones deben hacerse por unidades de 1 millón de pesos. La pregunta es: ¿Cuál es la distribución óptima del dinero para que la ganancia total sea máxima? Programe un algoritmo genético para poder contestar a la pregunta anterior.

Millones Invertidos	Ganancia T1	Ganancia T2	Ganancia T3	Ganancia T4
0	0.00	0.00	0.00	0.00
1	0.28	0.25	0.15	0.20
2	0.45	0.41	0.25	0.33
3	0.65	0.55	0.40	0.42
4	0.78	0.65	0.50	0.48
5	0.90	0.75	0.62	0.53
6	1.02	0.80	0.73	0.56
7	1.13	0.85	0.82	0.58
8	1.23	0.88	0.90	0.60
9	1.32	0.90	0.96	0.60
10	1.38	0.90	1.00	0.60

Tabla 1. Datos obtenidos en el estudio de mercado

```
enigmak9@EnigmaUnit: (master *)AG-> python ag.py
Mejor individuo=[4, 1, 2, 0]
enigmak9@EnigmaUnit: (master *)AG->
```

Figura 1. Salida del programa 'AG.py' mostrando al mejor individuo

Programa 1

Código asociado al programa 1

```
import random

def compLis(l):
    while len(l)!=4:
        l.append(0)

def convBin(coef):
    valores = []
    while coef/2 != 0:
        valores.append(coef%2)
        coef=coef/2
    if coef > 0:
        valores.append(coef)
    compLis(valores)
    valores.reverse()
    return valores

#Funcion para seleccionar números correctos
a=0
b=0
c=0
d=0
t1=0
t2=0
```

```
t3=0
t4=0
gan=0
sumaAptitudes=0
f_tot=0
pobBin=[]
pobDec=[]
listaEvaluacion=[]
listaEsperado=[]
listaAcumulado=[]

def valNums(a,b,c,d):
    return False if ((a+b+c+d)<=10 and (a+b+c+d)!=0) else True

def asignaVal():
    global a,b,c,d
    while valNums(a,b,c,d):
        a=random.randint(0,11)
        b=random.randint(0,11-a)
        c=random.randint(0,11-a-b)
        d=random.randint(0,11-a-b-c)
    return a+b+c+d

def ganancia():
    global t1,t2,t3,t4,a,b,c,d,gan
    ga=[0,0.28,0.45,0.65,0.78,0.90,1.02,1.13,1.23,1.32,1.38]
    gb=[0,0.25,0.41,0.55,0.65,0.75,0.80,0.85,0.88,0.90,0.90]
    gc=[0,0.15,0.25,0.40,0.50,0.62,0.73,0.82,0.90,0.96,1.00]
    gd=[0,0.20,0.33,0.42,0.48,0.53,0.56,0.58,0.60,0.60,0.60]
    t1=ga[a]
    t2=gb[b]
    t3=gc[c]
    t4=gd[d]
    return t1+t2+t3+t4

def f_objetivo():
    global gan,a,b,c,d
    gan=ganancia()
    v=abs(a+b+c+d-10)
    return gan/((500*v)+1)

#Crea individuos
def creaIndividuoBin():
    global a,b,c,d
    l=[]
    a=0
    b=0
    c=0
```

```
    d=0
    asignaVal()
    l.append(convBin(a))
    l.append(convBin(b))
    l.append(convBin(c))
    l.append(convBin(d))
    return l

def creaIndividuoDec():
    global a,b,c,d
    l=[]
    l.append(a)
    l.append(b)
    l.append(c)
    l.append(d)
    return l

#Inicializa poblaci'on, 50 individuos.
def creaPoblacion():
    global pobBin,pobDec
    for i in range(50):
        pobBin.append(creaIndividuoBin())
        #print "a={},b={},c={},d={}".format(a,b,c,d)
        pobDec.append(creaIndividuoDec())
        #print "a={},b={},c={},d={}".format(a,b,c,d)
        evaluaIn()
    #print "Taman'o poblacion={}".format(len(pobDec))

#Evaluar a los individuos
def evaluaIn():
    global pobDec,listaEvaluacion
    listaEvaluacion.append(f_objetivo())

#Selecciona individuos Ruleta
def sumaApt():
    global listaEvaluacion,sumaAptitudes
    for i in range(len(listaEvaluacion)):
        sumaAptitudes+=listaEvaluacion[i]


def valorEsperado():
    global listaEsperado,listaEvaluacion,sumaAptitudes,f_tot
    f_tot=sumaAptitudes/50
    for i in range(len(listaEvaluacion)):
        listaEsperado.append(listaEvaluacion[i]/f_tot)
    x=0
    for i in range(len(listaEsperado)):
        x+=listaEsperado[i]
```

```
    #print "Suma lista valor esperado={}".format(x)

def valorAcumulado():
    global listaEsperado,listaAcumulado
    for i in range(len(listaEsperado)):
        listaAcumulado.append(listaEsperado[0]) if i==0 else
listaAcumulado.append(listaAcumulado[i-1]+listaEsperado[i])
        #print listaAcumulado[i]
    #print "Ultimo valor listaAcumulado={}".format(listaAcumulado[49])


def seleccion():
    global listaAcumulado,pobDec
    pobNue=[]
    l=[random.uniform(0,50) for i in range(50)]
    for i in range(len(l)):
        for j in range(len(l)):
            if listaAcumulado[j]>l[i]:
                pobNue.append(pobDec[j])
                break
    pobDec=pobNue
    #print pobDec
    #print "Tamaño población en selección={}".format(len(pobDec))
    #print l

#Operador cruza: 2 puntos
#Siempre se utilizaran los mismos 2 puntos para garantizar que se
#generen 3 segmentos a intercambiar.

def cruza():
    global  listaAcumulado,pobDec
    pc=0.8
    l=[random.random() for i in range(50)]
    padres=[]
    hijos=[]
    posicionesHijos=[]
    #print "Taman'o poblacion={}".format(len(pobDec))
    for i in range(len(pobDec)):
        if pc>l[i]:
            padres.append(pobDec[i])
            posicionesHijos.append(i)
    #print pobDec
    #print padres
    posicionesHijos.reverse()
    for i in range(len(posicionesHijos)):
        pobDec.pop(posicionesHijos[i])
    for i in range(len(padres)):
        pad1=padres[i]
```

```
        for j in range(len(padres)-1):
            pad2=padres[j]
            #print "pad1={} pad2={}".format(pad1,pad2)
            if not valNums(pad1[0],pad2[1],pad2[2],pad1[3]):
                hijo=[pad1[0],pad2[1],pad2[2],pad1[3]]
                hijos.append(hijo)
                #print "hijo {}".format(hijo)
            else:
                hijo=[pad2[0],pad1[1],pad1[2],pad2[3]]
                hijos.append(hijo)
                #print "hijo {}".format(hijo)
    posicionesHijos.reverse()
    #print pobDec
    for i in range(len(posicionesHijos)):
        if len(pobDec)>posicionesHijos[i]:
            pobDec.insert(posicionesHijos[i],hijos[random.randint(0,len(hijos)-1)])
    #print "Separacion"
    #print pobDec
#realiza todas las combinaciones posibles de hijos y despues selecciona al azar
#cuales seran seleccionados.
    while len(pobDec)!=50:
        pobDec.append(hijos[random.randint(0,len(hijos)-1)])

    #print hijos
    #print "Taman'o Hijos={}".format(len(hijos))
    #print "Taman'o posicionesHijos={}".format(posicionesHijos)
    #print "Taman'o padres={}".format(len(padres))
    #print "Taman'o pobDec en cruza antes de elementos
nuevos={}".format(len(pobDec))

#Se utilizara' mutacio'n uniforme
def mutacion():
    global pobDec
    total_genes=len(pobDec)*4
    mr=int(0.1*total_genes)
    posicionesMutacion=[]
    for i in range(mr):
        posicionesMutacion.append(random.randrange(0,total_genes))
    #print "Total de genes={}".format(total_genes)
    #print "posicionesMutacion={}".format(posicionesMutacion)
    for i in range(len(posicionesMutacion)):
        individuo=posicionesMutacion[i]/4
        #print "posicionesMutacion=={}".format(posicionesMutacion[i]/4)
        posicionInd=posicionesMutacion[i]%4
        listaElemRango=[0,1,2,3]
        #print "Posicion de individuo={}".format(posicionInd)
        listaElemRango.remove(posicionInd)
        #print "listaElemRango={}".format(listaElemRango)
```

```
        UB=10
        #print "Elemento={} PosicionAMutar={}".format(individuo,posicionInd)
        for i in range(len(listaElemRango)):
            UB=UB-pobDec[individuo][listaElemRango[i]]
        #print pobDec[individuo]
        #print "UB={}".format(UB)
        if UB>0:
            pobDec[individuo][posicionInd]=random.randrange(0,UB)
            #print pobDec[individuo]

def mejorIndividuo():
    mejInd=0
    posMejInd=0
    for i in range(len(listaEvaluacion)):
        if listaEvaluacion[i]>mejInd:
            mejInd=listaEvaluacion[i]
            posMejInd=i
    print "Mejor individuo={}".format(pobDec[posMejInd])

def main():
    #asignaVal()
    #print "a={},b={},c={},d={}".format(a,b,c,d)
    #print "Millones invertidos:{}".format(a+b+c+d)
    #print f_objetivo()
    #print "Crea individuo:{}".format(creaIndividuoDec())
    creaPoblacion()
    #print
    #Se realiza 20 veces la ejecuci'on
    for i in range(20):
        evaluaIn()
        sumaApt()
        #print "Suma aptitudes={}".format(sumaAptitudes)
        valorEsperado()
        #print listaEsperado
        valorAcumulado()
        #print listaAcumulado
        seleccion()
        cruza()
        mutacion()
        #mejorIndividuo()
    mejorIndividuo()
    #print pobDec
    #print "listaEvaluacionFinal={}".format(listaEvaluacion)     #for i in
range(len(pobDec)):


main()
```

Código disponible en:

https://github.com/EnigmaK9/id3-aprendizaje/blob/master/AG/ag.py

Manual de usuario disponible en:

Conclusiones

Padilla Herrera Carlos Ignacio

Me parece sorprendente que la concepción de los algoritmos genéticos esté basada en la teoría de la evolución de Darwin, y que tenga un amplio rango de aplicaciones no es menos admirable. Estas aplicaciones que van desde lo comercial, educacional y científico y estás áreas van cada vez siendo más dependientes de estos. Su utilidad y elegancia a la hora de resolver problemas ha hecho que su popularidad crezca a comparación de otros métodos más tradicionales. Me parece que los algoritmos genéticos son muy útiles cuando el desarrollador no tiene un conocimiento amplio del campo, porque los algoritmos genéticos tienen la habilidad de explorar y aprender del dominio aplicado.

A través del desarrollo del algoritmo logramos captar la esencia más fácilmente de los AG, y de la incertidumbre que me generan acerca de que si la vida generada por computadora puede existir como una forma de vida verdadera.

Ramírez Ancona Simón Eduardo

Se ha mostrado que los algoritmos genéticos se desempeñan muy bien en áreas como el complejo mundo real. Estos son adaptativos al ambiente y son muy atractivos para todas las comunidades de cualquier campo que trabajan con un ambiente dinámico. Sin embargo, considero que varias mejoras puedan ser implementadas para que sean aplicadas generalmente, como el hecho de que varias generaciones puedan evolucionar en paralelo al mismo tiempo.

Durante el desarrollo del programa logré entender los efectos de la taza de cruza y mutación para poder maximizar la función objetivo que se nos solicitó.

Referencias

Hagan, Demuth, Beale & De Jésus. (s.f.). *Neural Network Design.* 2nd Edition

Mitchell, T. M. (1997). *Machine Learning*.

R. Rojas. *Neural Networks: A Systematic Introduction,* Springer, 1996.