Diego Fernando Montaño Pérez A01282875

AprendizajeAutomáticoTC3020

Reporte Práctica 5

Dr. Jesús Guillermo Falcón Cardona

Introducción

La practica 5 consistió en tres partes, en la primera se puso en practica lo visto sobre optimización de enjambre de partículas en la cual el profesor proporciono un código en C el cual tuvo que se modificado y los resultados que se obtenían se debían animar.

La segunda parte fue poner en práctica la teoría vista sobre Naïve Bayes en donde se tenían que obtener la probabilidad de sucesos de un dataset dado.

La última parte consistió en hacer un programa usando la librería de Naïve Bayes de skicit learn para clasificar el dataset de Iris, obtener su precisión y después graficar las instancias clasificadas.

Reporte

Parte 1: Particle Swarm Optimization

1) La primera parte solo cambie dentro de la función f() la formula dentro del ciclo tal y como se indicaba en las instrucciones, se borro también el aumento que se hizo en la variable temp. El código quedo de la siguiente manera:

```
double f(double *x){
    double tmp = 0.0;
    int i;
    for(i = 0; i < n; i++)
        tmp += x[i]*x[i];
    return tmp;
}
```

Mientras que los limites puestos en los DEFINE se les cambio el valor a -5 y 5.

```
#define PI 3.141592654
#define UB 5
#define LB -5
```

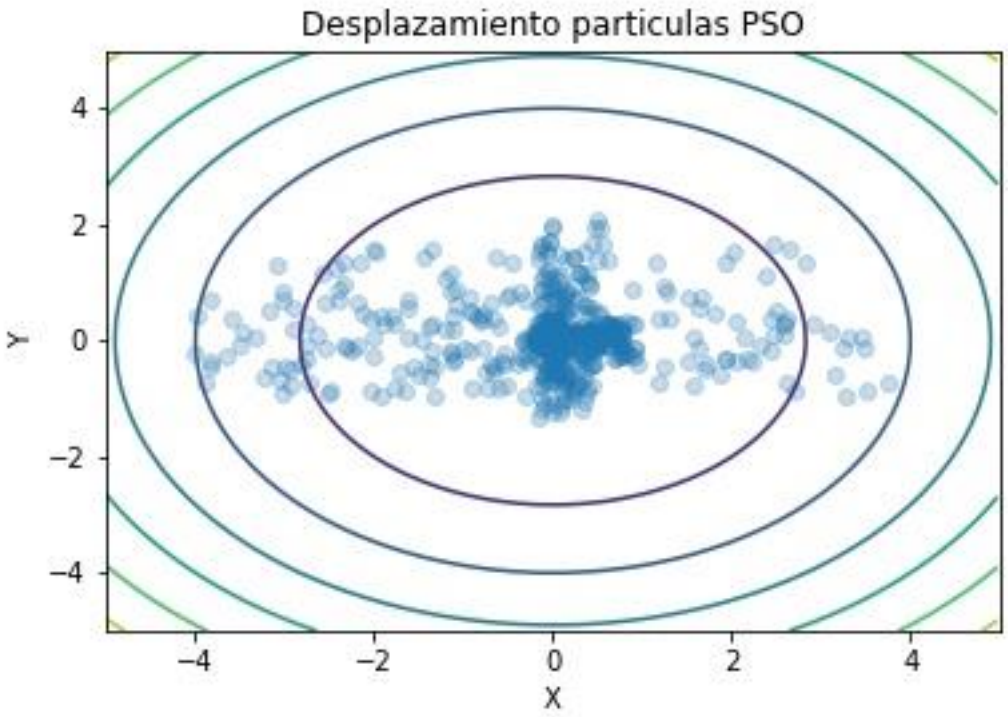
2) Se creo un nuevo archivo llamado answer.txt en donde se guardó la posición de las partículas por cada iteración del algoritmo.
   Para obtener resultados exitosos se usaron los siguientes parámetros al ejecutar el programa:

```
./PSO 2 100 100 3.5 2.3 0.9 1
```

```
97          for(int i = 0; i < N; i++){
98              // Set values for vectors r1 and r2 from a uniform distribution in range [0, 1]
99              for(int j = 0; j < n; j++){
100                 Swarm[i].v[j] = w*Swarm[i].v[j] + c1*r1[j]*(Swarm[i].y[j] - Swarm[i].x[j]) + c2*r2[j]*(gBest.x[j] - Swarm[i].x[j]);
101                 Swarm[i].x[j] = Swarm[i].x[j] + Swarm[i].v[j];
102                 // If out of bounds, force x_j to be inside the feasible region
103                 if(Swarm[i].x[j] < LB)
104                     Swarm[i].x[j] = LB;
105                 else if(Swarm[i].x[j] > UB)
106                     Swarm[i].x[j] = UB;
107             //Save in file position of particle
108             fprintf(fileMov,"%lf " ,Swarm[i].x[j]);
109             }
110         }
111         fprintf(file, "%lf\n", gBest.f);
112         fprintf(fileMov,"\n ");
113         t++;
114     }
115     fclose(file);
116     fclose(fileMov);
117 }
```

3) Se genero una animación para mostrar el desplazamiento de las partículas en el espacio dado, debido a que por obvias razones la animación no la puedo mostrar en el reporte, esta se encontrara disponible en la carpeta donde se entrega el trabajo bajo el formato de MP4, adjunto fotografía de como se ve un

frame de la animación:



Parte 2: Ejercicio Manual de Naïves Bayes

La tabla para hacer los operaciones fue obtenida de las instrucciones del reporte, en la parte de abajo se dejan los cálculos y pasos que seguir para llegar al resultado de la clasificación.

P(Beach = Yes) = 4/10

P(Beach = No) = 6/10

Outlook

	Yes	No	P(Yes)	P(No)
Sunny	3	1	¾	1/6
Rain	0	3	0	3/6
Cloudly	1	2	¼	2/6
Total	4	6	100%	100%

Temp

	Yes	No	P(Yes)	P(No)
High	3	2	¾	2/6
Mild	1	2	¼	2/6
Low	0	2	0	2/6
Total	4	6	100%	100%

Humidity

	Yes	No	P(Yes)	P(No)
High	2	2	2/4	2/6
Normal	2	4	2/4	4/6
Total	4	6	100%	100%

P(X|C)P(C=Yes) = ¼ * ¼ * 2/4 * 4/10 = 0.0125

(C=No) = 1/6 * 2/6 * 2/6 * 6/10 = 0.0111

Se divide por ambos lados P(x) para normalizar

P(x) = 2/10 * 3/10 * 4/10 = 0.024

Se dividen los resultados

P(C=Yes|X) = 0.0125/0.024 = 0.5208

P(C=No|X) = 0.0111/0.024 = 0.4625

R-. No, no se podrá ir a la playa

Parte 3: Uso de Naïves Bayes con Scikit-Learn

1-2) Se utilizo la base de datos de Iris para crear el programa, este dataset se importo de sklearn. Para la división del training set y testing set se utilizó la misma técnica de las otras prácticas, pero en este caso se usó un 75% para el training y un 25% para el test set, en el siguiente código se muestra como se hizo este paso:

```
[ ]  #Modelando arbol de decision y dividiendo dataset
     iris = load_iris()
     X = iris.data[:,:4]
     y = iris.target
     X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.25, random_state=25)
```

3) Para hacer la clasificación se usaron dos diferentes funciones que maneja scikit learn, GaussianNB y Multinomial. Ambas se manejaron de la misma manera pues recibían los mismos parámetros, en el siguiente código se muestra como se implementaron y como se obtuvo la precisión del modelo usando ambas funciones y por último de obtuvo las matrices de confusión de cada caso:

```
#GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
#Prediccion
acurracy = accuracy_score(y_test, y_pred)
print("Precision GaussianNB : ", acurracy)
#Creando matriz de confusion
ConfMtx = confusion_matrix(y_test, y_pred)
print("Matriz Confusion GaussianNB: ",ConfMtx)

# Grafica ilustrada
plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

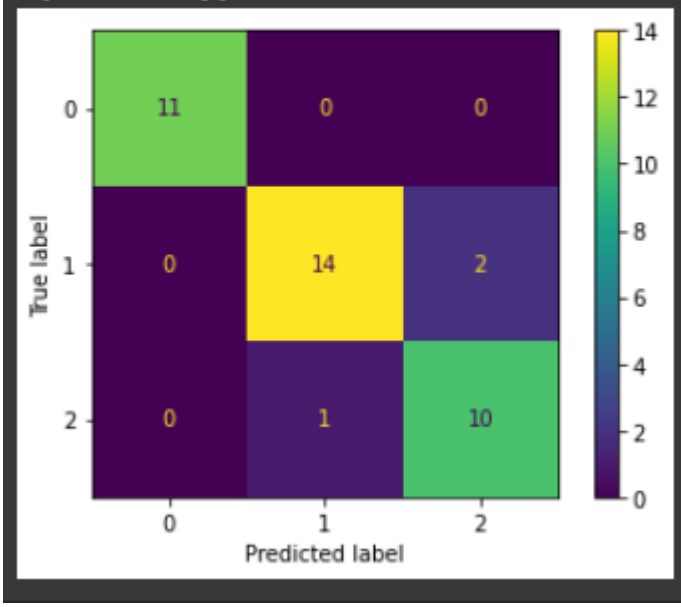
NB

```
#MultinomialDB
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
#Prediccion
acurracy = accuracy_score(y_test, y_pred)
print("Precision MultinomialNB : ", acurracy)
#Creando matriz de confusion
ConfMtx = confusion_matrix(y_test, y_pred)
print("Matriz Confusion MultinomialNB: ",ConfMtx)

# Grafica ilustrada

plot_confusion_matrix(classifier, X_test, y_test)
plt.show()
```

4) Los resultados obtenidos se mostrarán en la parte inferior, pero se puedo concluir que la función de GaussianNB tiene mejor precisión que la multinomial, por lo tanto, para categorizar este dataset es mejor usar la función de GaussianNB.

Diego Fernando Montaño Pérez A01282875

```
Precision GaussianNB :  0.9210526315789473
Matriz Confusion GaussianNB:  [[11  0  0]
 [ 0 14  2]
 [ 0  1 10]]
```




```
Precision MultinomialNB :  0.7105263157894737
Matriz Confusion MultinomialNB:  [[11  0  0]
 [ 0  5 11]
 [ 0  0 11]]
```