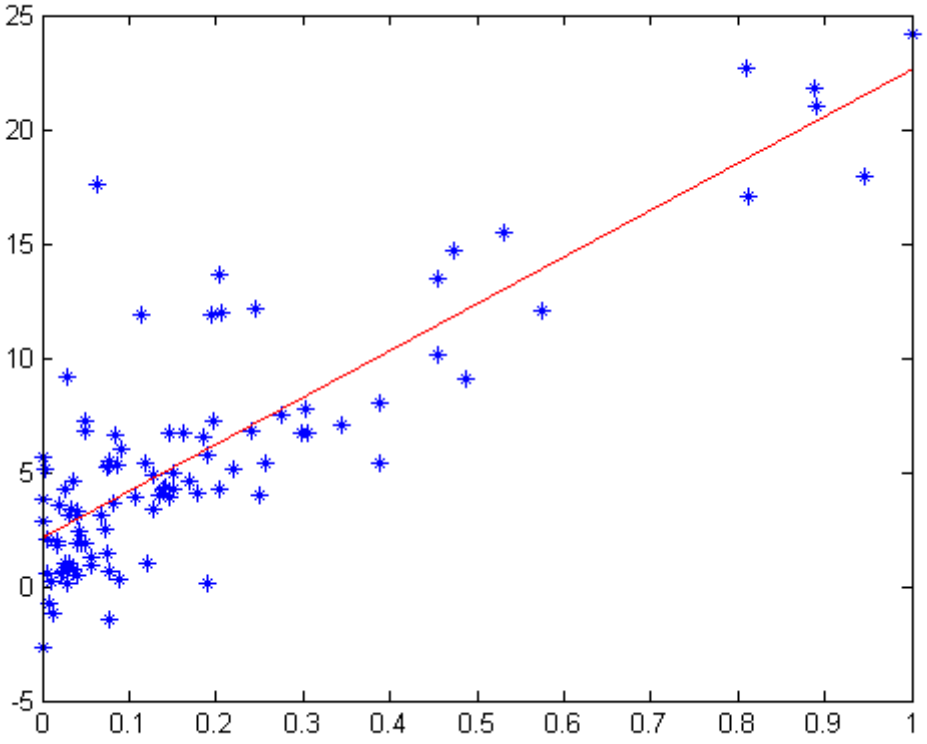
Tercera Tarea [OPCIONAL]: (Reynaldo Alfonte Zapana)

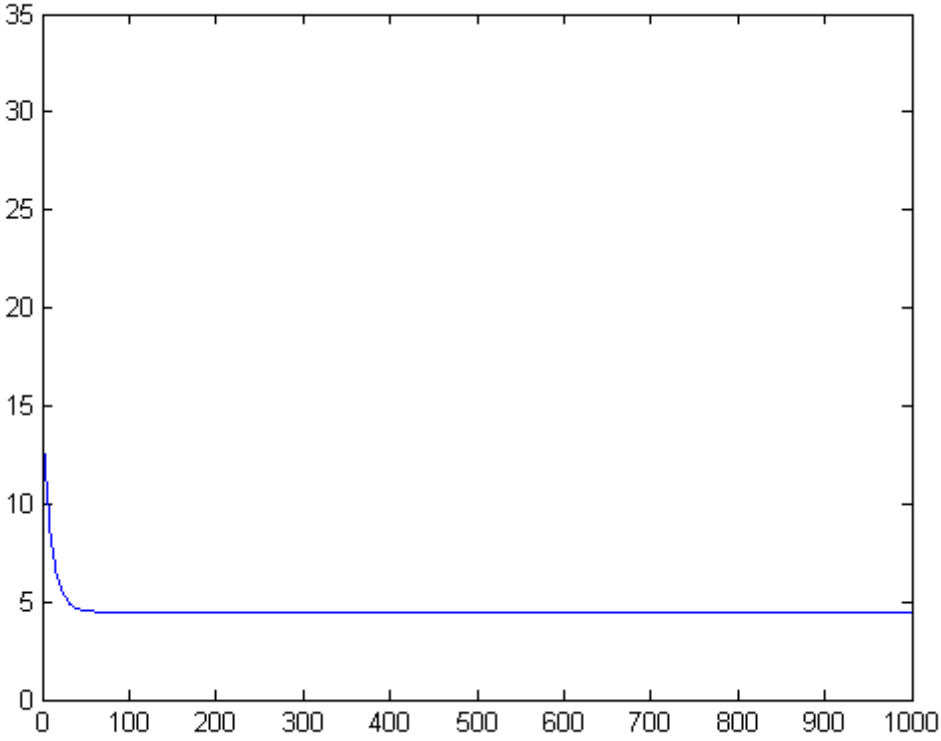
Consiste en probar la regularización con un parámetro muy pequeño, normal, muy grande en los mismos archivos de datos de la tarea 1 y 2.

Que sucede cuando tenemos un lambda muy pequeño?, y con uno muy grande?. Mostrar los graficos de regresion lineal y regresion logistica (frontera de decisión) obtenidos.

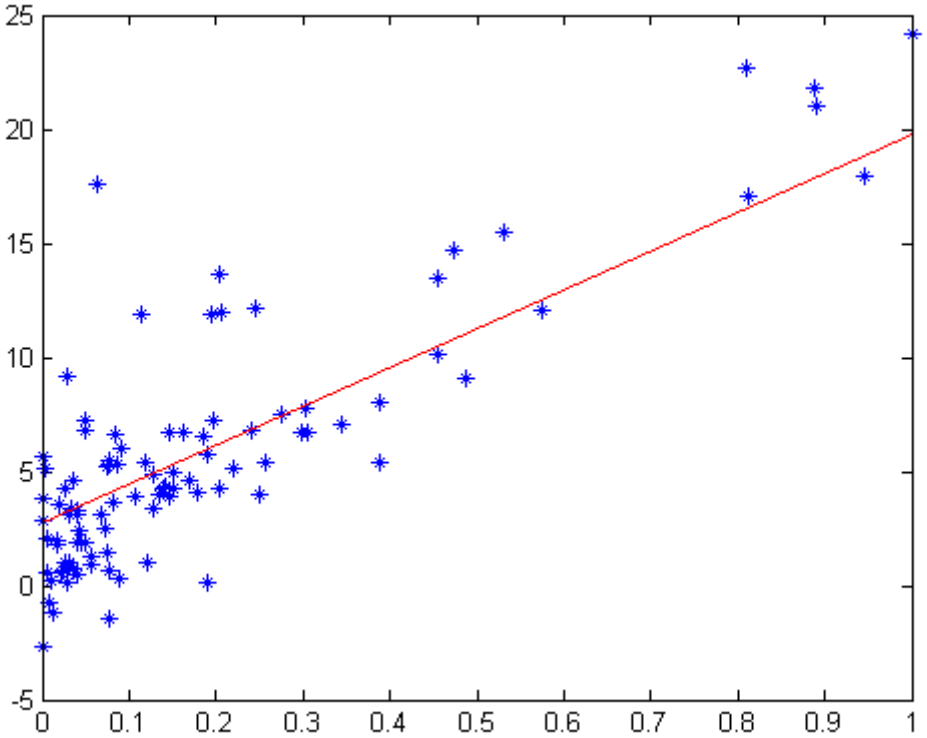
Solucion

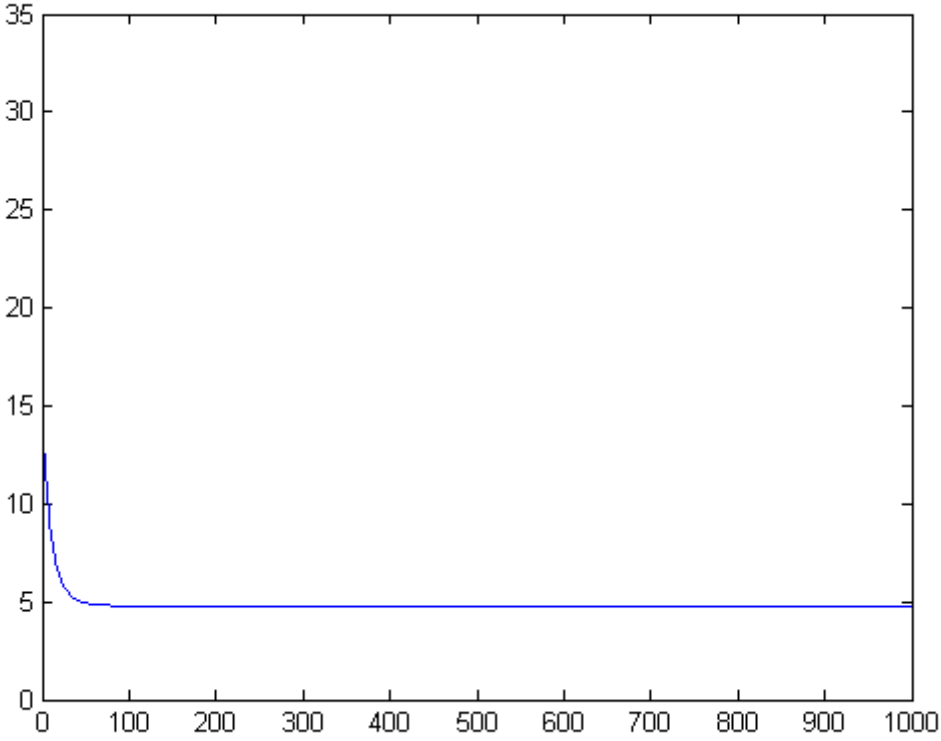
Labmda=0



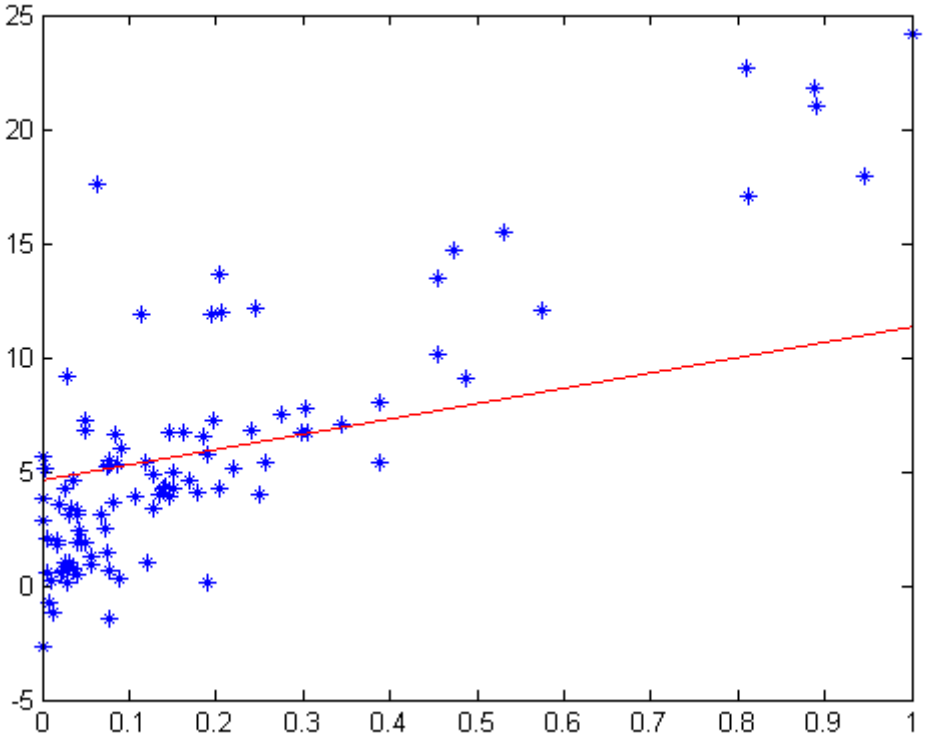

35
30
25
20
15
10
5
0
0
100
200
300
400
500
600
700
800
900
1000

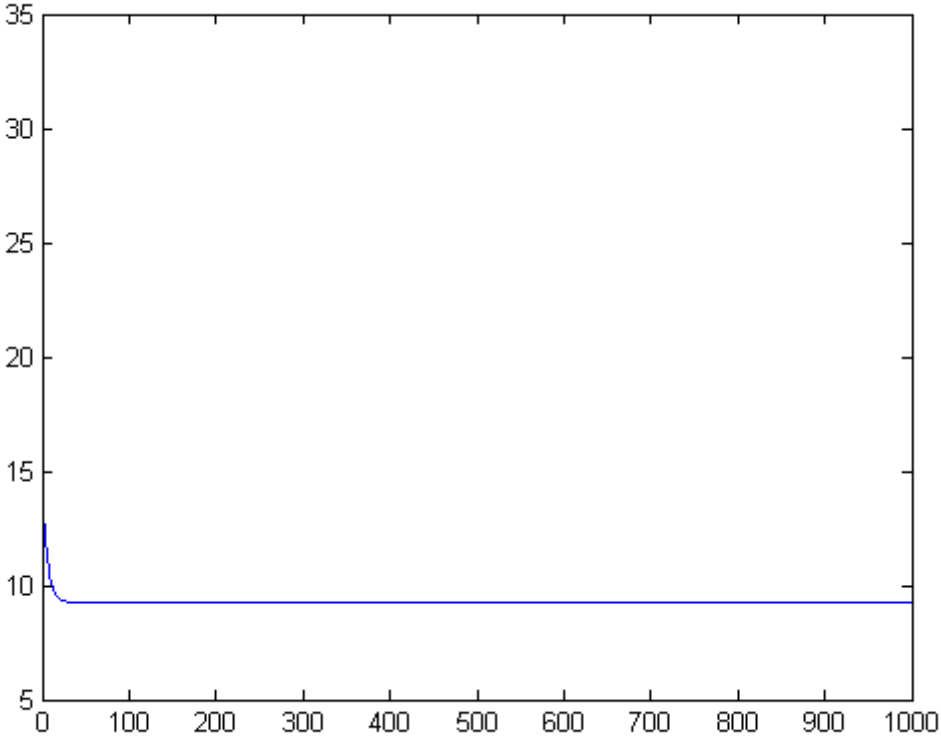
Lambda=1


25
20
15
10
5
0
-5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

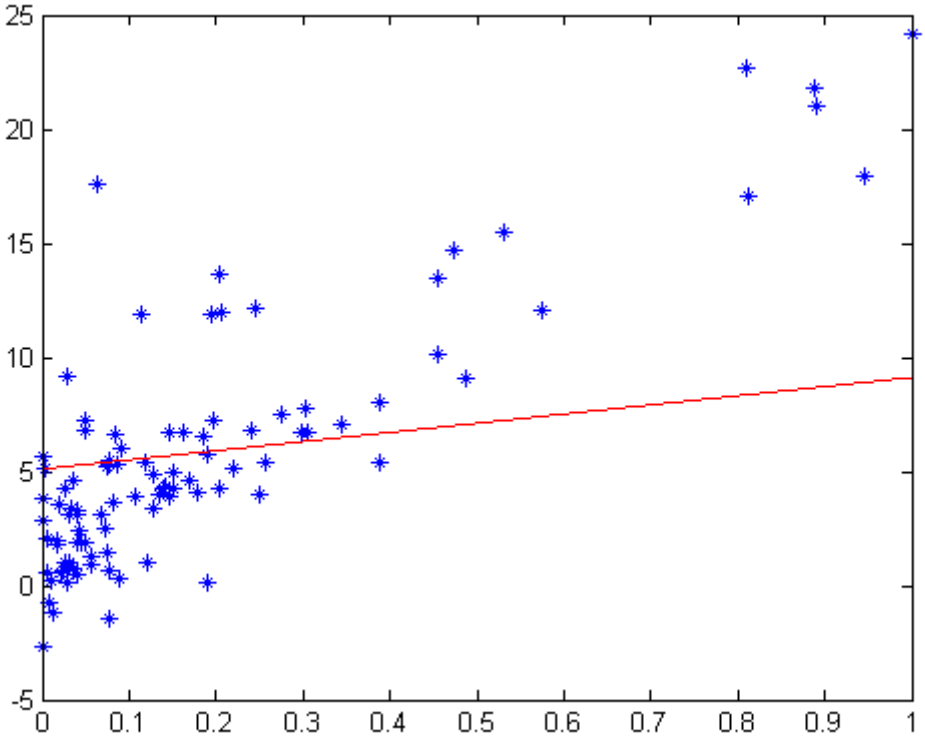

35
30
25
20
15
10
5
0
0
100
200
300
400
500
600
700
800
900
1000

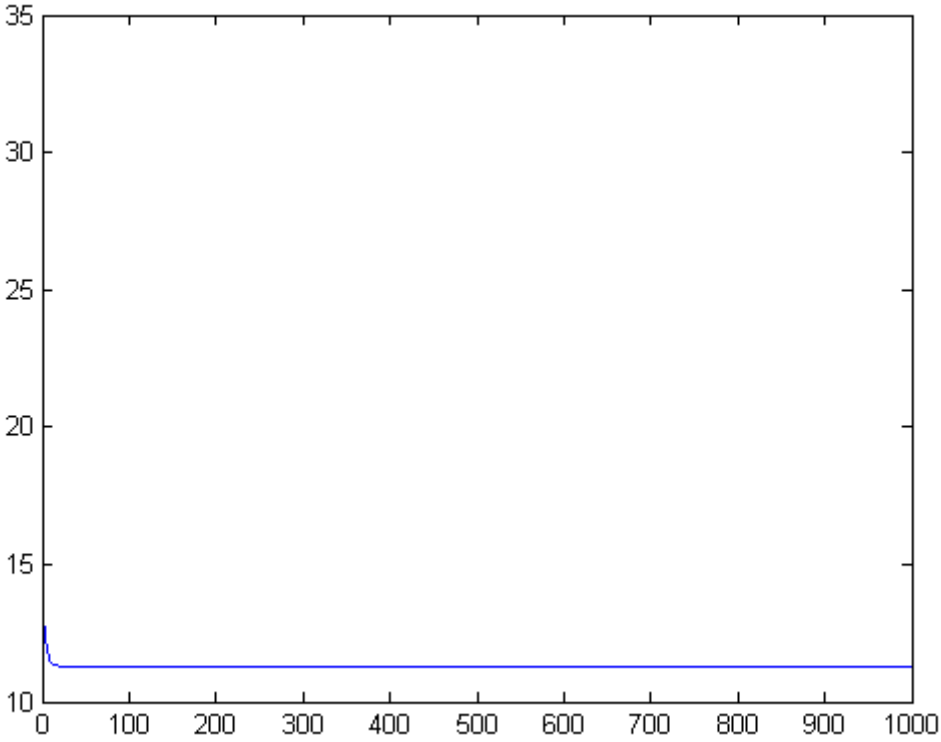
Lambda=10


25
20
15
10
5
0
-5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

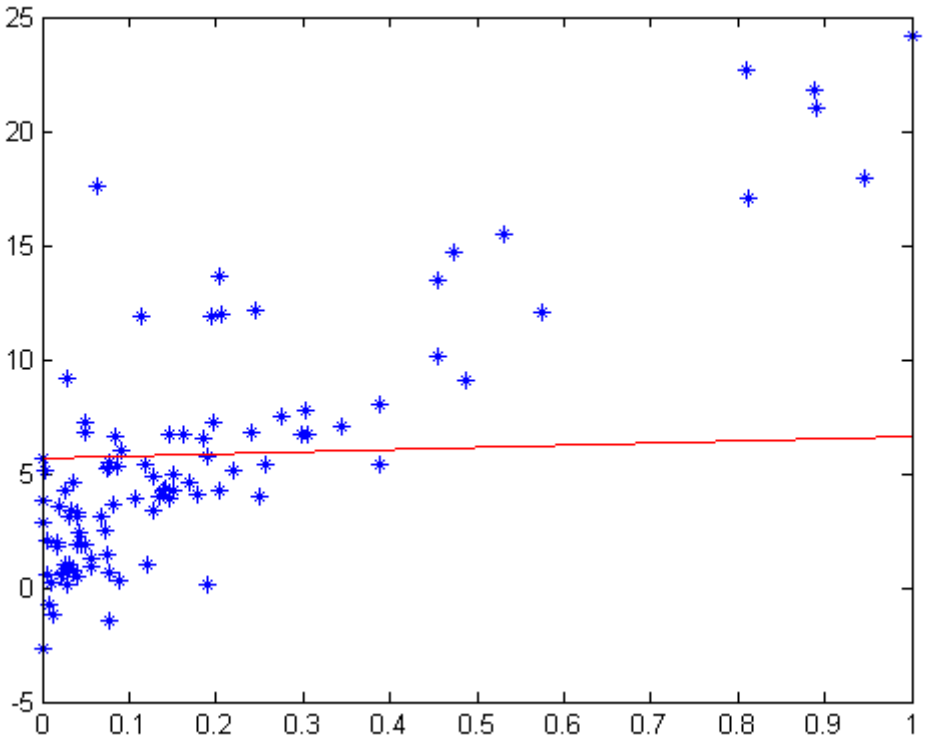

35
30
25
20
15
10
5
0
100
200
300
400
500
600
700
800
900
1000

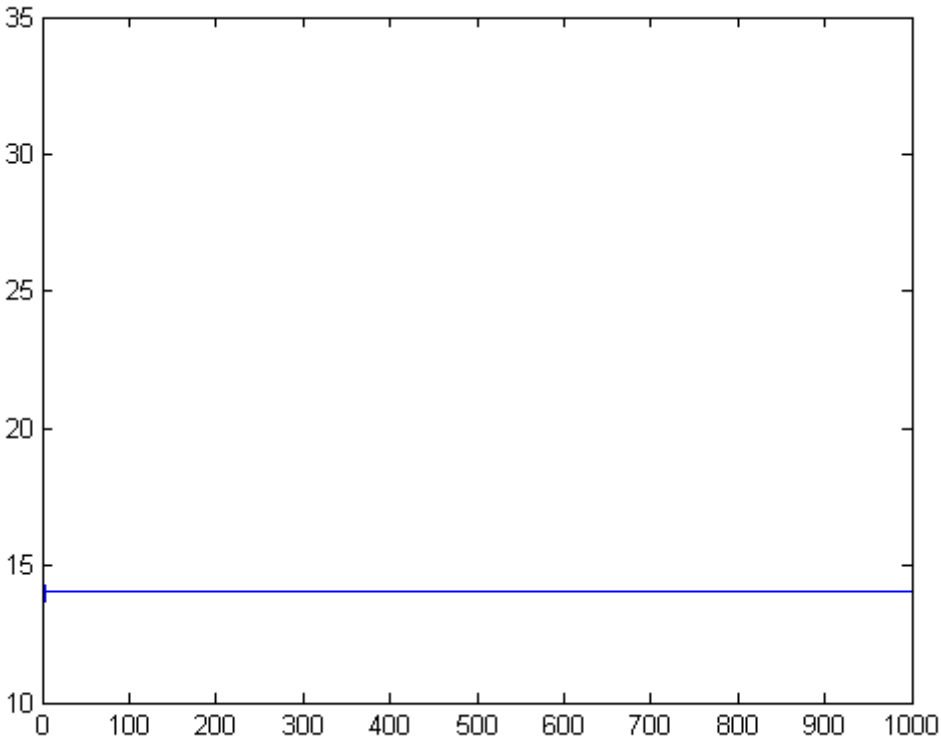
Lambda=20


25
20
15
10
5
0
-5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


35
30
25
20
15
10
0
100
200
300
400
500
600
700
800
900
1000

Lambda=100


25
20
15
10
5
0
-5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



En conclusión se observa que a mayor lambda la función de hipótesis converge al óptimo rápidamente pero la regresión no es buena (los parámetros theta no son adecuados).

Cuarta Tarea:

a) En la carpeta SegundaSemana/Codigo/Matlab Se encuentra el código de redes neuronales, pertenece a un ejercicio de programación de Andrew Ng (http://www.andrewng.org/). El mismo se encuentra casi resuelto. El archivo principal esta en el archivo ex4.m, el cual está dividido por una série de pasos.

Su tarea en implementar el paso 8 (Implement Regularization). Que consiste en modificar el archivo nnCostFunction.m (donde se implementa el algoritmo back Propagation y se obtiene la función Costo) y agregarle el término de regularización a las variables Theta1_grad(:) y Theta2_grad(:). La fórmula de ese término se encuentra en la última diapositiva del archivo: "SegundaSemana/Contenido/1-Redes Neuronales.pdf"

El código está bien documentado y en ex4.pdf está descrito lo que hace cada función especificamente. Y dá ayudas para obtener una correcta implementación.

Algunas cosas que quiero aclarar son las siguientes:

- Para probar si su regularización esta bien hecha,
- El archivo fmincg.m contiene un algoritmo similar al Gradient Descent que utiliza la función nnCostFunction.m, lo que devuelve esta función obviamente son los pesos Theta optimos que minimizan la función Costo (J).
- Como lo dije en clases, al Gradient Descent u otros algoritmos de optimización similares solo les interesa la versión linearizada de los pesos Theta con sus respectivas gradientes. Por lo tanto, en casi todo el código se trabaja de esa forma. Por otro lado el algoritmo Back Propagation (implementado en nnCostFunction.m) utiliza una versión matricial de los pesos Theta, por lo que al inicio de esa función aparece el método 'reshape', que recibe una version linearizada y lo transforma a su forma matricial. Finalmente esa misma función retorna los gradientes de THeta linearizados.
- Nuevamente mas informaciones en el archivo ex4.pdf.
- Entiendan el código antes de hacer alguna cosa.

Solucion

Lo que se agrego fue el siguiente código:

```
Theta2_grad = Theta2_grad / m + lambda/m * [zeros(size(Theta2,1),1),Theta2(:,2:end)];

Theta1_grad = Theta1_grad / m + lambda/m * [zeros(size(Theta1,1),1),Theta1(:,2:end)];
```

Después del bucle for y antes de calcular la función J.

b) Implementar el paso 12 (70% training, 30% testing) y 13 (Cross Validation) en ex4.m. En el mismo archicvo ex4.m se detalla lo que tiene que ser hecho.

Hacer un relatorio para la tarea a) y b) escribiendo al menos el código utilizado, y para b) la exactitud(accuracy) obtenida.

Solucion

Para el paso 12

Se agrego el siguiente código, que selecciona aleatoriamente el 70% de los datos para entrenamiento y 30% para test.

```
percent = 0.7;
select = randperm(m);
train_size = round(m * percent);
%test_size = m - train_size;
X_train = X(select(1:train_size),:);
X_test = X(select(train_size+1:end),:);
y_train = y(select(1:train_size),:);
y_test = y(select(train_size+1:end),:);
```

y ase agrego el código de la parte 9 (for training) y de la parte 11(to predict)

Para el paso 13

Antes del bucle se agregó lo siguiente

```
kfold = 10;
select = randperm(m);
fold_size = round(m/kfold);
```

Donde *select* es un arreglo que tiene aleatoriamente permutado los índices de X, *fold_size* define el tamaño aproximado de cada *fold*.

Dentro del bucle se agregó el siguiente código

```
if k ~= kfold
    X_test = X(select((k-1)*fold_size+1:k*fold_size),:);
    y_test = y(select((k-1)*fold_size+1:k*fold_size),:);
    X_train = X([select(1:(k-1)*fold_size),select(k*fold_size+1:end)],:);
    %X_train = [X_train; X(k*fold_size+1:end,:)];
    y_train = y([select(1:(k-1)*fold_size), select(k*fold_size+1:end)],:);
    %y_train = [y_train; y(k*fold_size+1:end,:)];
else
    X_test = X(select((k-1)*fold_size + 1:end),:);
    y_test = y(select((k-1)*fold_size + 1:end),:);
    X_train = X(select(1:(k-1)*fold_size),:);
    y_train = y(select(1:(k-1)*fold_size),:);
end
```

lo que hace es seleccionar fold_size = round(m/kfold) = 500 datos aleatoriamente y lo demás datos para entrenamiento (for training), después ejecuta el código del paso 9 y del paso 11, todo este proceso se da en cada iteración.

c) [Opcional] Para el paso 13 crear una matriz de confusión. Y responder la pregunta: Que clase (1-10) es la que falla mas veces?.

Solucion

Para obtener la matriz de confusión se hizo lo siguiente.

```
confusion=zeros(10,10);
size_test=size(y_test);
for i=1:size_test
   confusion(pred(i),y_test(i))=confusion(pred(i),y_test(i))+1;
end
disp(confusion);
```

Para determinar que clase falla más veces, se agregó

```
disp(sum(confusion - confusion.* eye(10)));
```

se obtuve la siguiente tabla en una de las iteraciones

1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
1	2	6	3	8	1	4	6	5	4
1	7	8	5	4	3	7	8	6	0
0	6	4	3	5	1	2	4	6	1
4	1	4	8	5	0	3	4	3	0
1	8	7	1	8	4	1	5	4	1
0	3	6	1	7	4	5	7	2	2
1	6	10	4	6	2	5	2	5	1
3	11	7	1	7	1	5	5	2	1
2	8	4	2	9	2	6	4	4	1
2	10	7	3	4	3	0	5	5	1
15	62	63	31	63	21	38	50	42	12

Las clases que fallan más son la clase 5, clase 3, clase 2 y clase 8.