Proyecto 1 - Modelos de clasificación

Abraham Arias Chinchilla, Lenin Torres Valverde, Mauricio Montero Jiménez.
ipseabraham@gmail.com, ttvleninn@gmail.com, maumonteroj@gmail.com

I. 3. Procedimiento

A. 3.1. Función predictora

A

$$\underline{\hat{y}}(\underline{x}; W^{(1)}, W^{(2)} = g\left(W^{(2)}\begin{bmatrix}1\\ g\left(W^{(1)}\begin{bmatrix}1\\ \underline{x}\end{bmatrix}\right)\end{bmatrix}\right)$$

Con

$$W^{(i)} = \begin{bmatrix} W_{10}^{(i)} & W_{11}^{(i)} & W_{12}^{(i)} & \cdots & W_{1n_i}^{(i)} \\ W_{20}^{(i)} & W_{21}^{(i)} & W_{22}^{(i)} & \cdots & W_{2n_i}^{(i)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W_{p_i0}^{(i)} & W_{p_i1}^{(i)} & W_{p_i2}^{(i)} & \cdots & W_{p_in_i}^{(i)} \end{bmatrix}$$

El número de columnas igual a la cantidad de neuronas en la capa (i) y tantas filas como entradas en cada neurona. Por ejemplo en la segunda cada $W^{(2)}$ tiene dimensiones $(3X\#neuronas\_capa\_1 + 1)$, mas 1 por la entrada de "bias".

B. 3.2. Función blanco

$$J(W^{(1)}, W^{(2)}) = \frac{1}{2}\sum_{j=1}^{m}\left(y^{(j)} - \underline{\hat{y}}(\underline{x}^{(j)}; W^{(1)}, W^{(2)})\right)^2$$

C. 3.3. Función gradiente

Para obtener:

$$gW^{(1)} = \begin{bmatrix} \frac{\partial J}{\partial W_{10}^{(i)}} & \frac{\partial J}{\partial W_{11}^{(i)}} & \frac{\partial J}{\partial W_{12}^{(i)}} & \cdots & \frac{\partial J}{\partial W_{1n}^{(i)}} \\ \frac{\partial J}{\partial W_{20}^{(i)}} & \frac{\partial J}{\partial W_{21}^{(i)}} & \frac{\partial J}{\partial W_{22}^{(i)}} & \cdots & \frac{\partial J}{\partial W_{2n}^{(i)}} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \frac{\partial J}{\partial W_{p0}^{(i)}} & \frac{\partial J}{\partial W_{p1}^{(i)}} & \frac{\partial J}{\partial W_{p2}^{(i)}} & \cdots & \frac{\partial J}{\partial W_{pn}^{(i)}} \end{bmatrix}$$

Teniendo en cuenta que:

$$g'(x) = g(x)(1 - g(x))$$

Para obtener por ejemplo $\frac{\partial J}{\partial W_{21}^{1}}$, un peso de la primera capa por medio de error de retropropagación, usando a On como la salida de la enésima neurona de salida con su función antes de activación bOn y para las capas escondidas $H21$ por ejemplo para la salida de la primera neurona de la capa 2 con su respectiva función antes de la activación $bH21$. Sabemos que el MSE (mean square error) tiene en este caso la forma:

$$\frac{\partial J}{\partial W_{21}^{1}} = \left(\frac{(O1 - Y1)^2}{2} + \frac{(O2 - Y2)^2}{2} + \frac{(O3 - Y3)^2}{2}\right)'$$

Derivando cada uno de estos términos tenemos

$$\frac{MSE_{O1}}{\partial W_{21}^{(1)}} = \frac{MSE_{O1}}{\partial O1} * \frac{\partial O1}{\partial bO1} * \frac{\partial bO1}{\partial H21} * \frac{\partial H21}{\partial bH21} * \frac{\partial bH21}{\partial W^{(1)}{}_{21}}$$

$$\frac{MSE_{O2}}{\partial W_{21}^{(1)}} = \frac{MSE_{O2}}{\partial O2} * \frac{\partial O2}{\partial bO2} * \frac{\partial bO2}{\partial H21} * \frac{\partial H21}{\partial bH21} * \frac{\partial bH21}{\partial W^{(1)}{}_{21}}$$

$$\frac{MSE_{O3}}{\partial W_{21}^{(1)}} = \frac{MSE_{O3}}{\partial O3} * \frac{\partial O3}{\partial bO3} * \frac{\partial bO3}{\partial H21} * \frac{\partial H21}{\partial bH21} * \frac{\partial bH21}{\partial W^{(1)}{}_{21}}$$

Nombrando los términos en común para todos los pesos se puede utilizar la siguiente notación

$$\frac{\partial J}{\partial W_{21}^{1}} = (go1 * w11 + go2 * w12 + go3 * w13)(1 - g(x))g(x) * X$$

D. 3.2. Entrenamiento

E. 3.3. Visualización de los datos

F. 3.4. Evaluación de los resultados

1) 3.4.1. Para evaluar el rendimiento de su algoritmo, genere otro conjunto de datos (distinto del conjunto de entrenamiento) con la misma distribución de puntos que los datos de entrenamiento:

2) 3.4.2. Matriz de confusión: Es una técnica para resumir el rendimiento de un algoritmo de clasificación. Para compensar el hecho que aveces se tienen cantidades distintas de datos en cada clase. Obtener esta matriz permite observar que cosas está acertando el modelo y en cuales está fallando. Una matriz de confusión es un resumen de los resultados de la predicción sobre un problema de clasificación. Consiste en contar cuantas veces acertó por cada clase y cuantas veces se equivocó por cada clase.

3) 3.4.3 Utilice su función de predicción para predecir cada punto en el conjunto de prueba y cree la matriz de confusión correspondiente, donde las filas deben tener las clases reales y las columnas las clases predichas:

4) 3.4.4. Investigue qué métricas de evaluación de clasificación se pueden derivar de la matriz de confusión, en particular la sensitividad y la precisión, y calcúlelas para sus datos: Dentro de los valores que permiten entender mejor la matriz de confusión y sus resultados están:

1) Porcentaje de error: se calcula como el número de todas las predicciones incorrectas dividido por el número total del conjunto de datos.
2) Exactitud: el número de todas las predicciones correctas dividido por el número total del conjunto de datos.
3) Sensitividad: el número de predicciones positivas correctas dividido por el número total de positivos.
4) Especificidad: número de predicciones negativas correctas dividido por el número total de negativos.
5) Precisión: el número de predicciones positivas correctas dividido por el número total de predicciones positivas.
6) Tasa de falsos positivos: el número de predicciones positivas incorrectas dividido por el número total de negativos.
7) Coeficiente de correlación de Matthews: es un coeficiente de correlación calculado utilizando todos los valores.
8) La puntuación F: es una media armónica de precisión y Sensitividad.

II. 4.Python, Scikit-Learn y Keras

A. 4.1. Scikit Learn y SVM

Scikit Learn es una biblioteca para Python principalmente escrita en este mismo lenguaje, proporciona al programador herramientas para el desarrollo de aprendizaje automático y reconocimiento de patrones, entre los algoritmos que incluye están:

- Clasificación:
  - SVM.
  - Nearest neighbors.
  - Random forest.
- Regresión:
  - SVR.
  - Ridge regression.
  - Lasso.
- Clustering:
  - K-means.
  - Spectral clustering.
  - Mean-shift.
- Reducción de dimensionalidad:
  - PCA.
  - Feature selection.
- Entre otros.

Para clasificar los datos generados en Octave se utiliza el algoritmo de SVM que ofrece Scikit-Learn, este algoritmo es un método de aprendizaje supervisado utilizado para problemas de clasificación. Las maquinas de soporte vectorial(Support vector machine) son muy efectivos en espacios con muchas dimensiones aunque se cuente con un numero menor de muestras, su funcionamiento puede variar fácilmente modificando el kernel(lineal, polinomial, rbf(Radial Basis Function), sigmoide).

Se analizó el algoritmo con distintas distribuciones de puntos para evaluar en cuales de estos cada kernel logra separar los datos efectivamente. En la Figura 1, Figura 2 y Figura 3, se muestran las clasificaciones realizadas por el algoritmo SVM con kernel lineal, polinomial y rbf respectivamente para una distribución de datos radial con tres clases. Como se logra observar para este tipo de distribución el kernel mas adecuado para la clasificación de los datos es el rbf, para las distintas distribuciones como las verticales y horizontales los tres kernel distintos funcionan bien por ser distribuciones lineales.

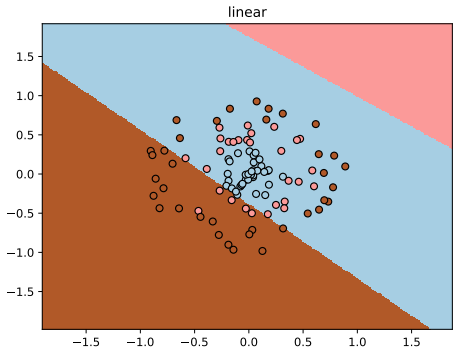

Figure 1: SVM con kernel lineal

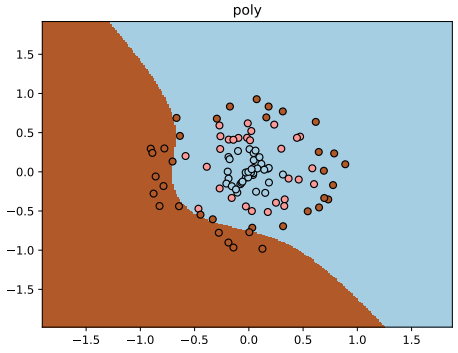

Figure 2: SVM con kernel polinomial

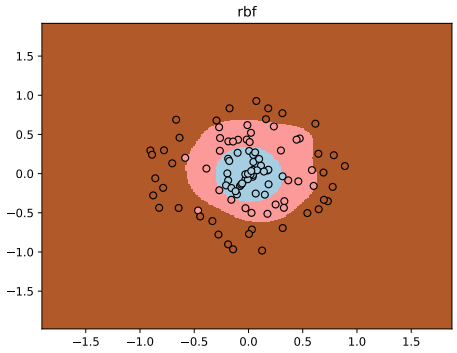

Figure 3: SVM con kernel rbf

Para comprobar la efectividad de las clasificaciones aparte de los gráficos se generaron matrices de confusión mostradas en la figura 4, figura 5 y figura 6, acá también podemos comprobar que para la distribución radial el mejor kernel es el rbf, ya que su matriz contiene la mayor cantidad de coincidencias en la diagonal, lo que indica que los datos predichos son lo esperados.

$$\begin{vmatrix} 34 & 0 & 0 \\ 27 & 0 & 7 \\ 20 & 0 & 12 \end{vmatrix}$$

Figure 4: Matriz de confusión con kernel lineal

$$\begin{vmatrix} 34 & 0 & 0 \\ 34 & 0 & 0 \\ 19 & 0 & 13 \end{vmatrix}$$

Figure 5: Matriz de confusión con kernel polinomial

$$\begin{vmatrix} 33 & 1 & 0 \\ 1 & 29 & 4 \\ 0 & 1 & 31 \end{vmatrix}$$

Figure 6: Matriz de confusión con kernel rbf

B. 4.2. Keras y Deep learning

Keras es un biblioteca para Python diseñada para ser un API de alto nivel para implementar sistemas de redes neuronales y Deep learning, este último es una rama del aprendizaje automático que simula redes neuronales de manera artificial para llegar a aprender de manera no supervisada de una gran cantidad de datos. Para realizar las pruebas de estos algoritmos se implementó una red neuronales que aprendiera a reconocer dígitos del 0 al 9, para esto se entrenó con la base de datos MNIST que cuenta con 60000 números escritos a mano, se implementó una interfaz para dibujar un número y evaluarlo en la red previamente entrenada, además con datos también proporcionados por la base de datos se probó la red neuronal para determinar la matriz de confusión y sus datos derivados.

III. Conclusiones

El descenso de gradiente permite a el conjunto de datos obtener la importancia que tienen los demás datos para solo un dato del conjunto y con ello obtener una forma de clasificar diferentes grupos de datos de entrada en una clase.

El uso de redes neuronales en los problemas computacionales de clasificación permiten que se tengan sistemas que cumplan los requerimientos de clasificar datos de un forma en que el sistema sea simple de comprender y permite mayor flexibilizad de cambio al sistema.