Aprendizaje de Máquinas, Laboratorio #2

Perceptrón, Redes Neuronales Artificiales y Support Vector Machine

Fecha de entrega a definir

Nota: (1) Estas preguntas requieren pensar, pero no requieren largas respuestas. Por favor se tan conciso como sea posible. (2) Cuando envíes una pregunta al foro, por favor asegúrate de escribir el número de laboratorio y el número del problema, tal como L2 P2. (3) Para problemas que requieran programación, por favor incluye en tu envio el código (con comentarios) y cualquier figura que se haya solicitado graficar. Ten en cuenta que el código debe poder correr desde cualquier máquina (4) Si escribes tus soluciones a mano, por favor escribe claramente y utilizando una birome de color oscuro.

Clasificación de orientación de caras en imágenes[1]

Entre las aplicaciones más comunes de Machine Learning se encuentra la construcción de software que puede resultar demasiado difícil de programar a mano, como por ejemplo software para reconocimiento de voz. Otro ejemplo es el problema de clasificación de imágenes y la tarea de reconocimiento facial. En nuestro caso vamos a utilizar técnicas de Machine Learning en imágenes como las que se muestran en la Figura 1 para clasificar la orientación de caras sobre nuevas imágenes de rostros[2]. Estas imágenes son parte de una colección más grande que contiene 624 imágenes de 20 personas diferentes[3] (correspondiente a estudiantes del curso Machine Learning de la Universidad Carnegie Mellon). Por cada persona se tomaron aproximadamente 32 imágenes en escala de grises, variando la expresión del rostro (neutral, feliz, triste, enojado), la dirección en que él/ella está mirando (al frente, izquierda, derecha, arriba) y su apariencia (con o sin gafas de sol). Teniendo en cuenta estos datos, vamos a usar Redes Neuronales Artificiales (ANN) y Support Vector Machine (SVM) para construir un clasificador que reconozca la orientación del rostro para nuevas imágenes.

Estas imágenes puede encontrarlas en el directorio `data/faces_4`, el cual contiene 20 subdirectorios, uno por cada persona. Cada uno de estos directorios contiene diferentes imágenes del rostro para una misma persona. En el archivo `data/images.txt` se puede encontrar el listado completo de las imágenes con su ruta relativa a `face_4`. El nombre de cada imagen sigue el siguiente formato:

`<userid>_<pose>_<expression>_<eyes>_<scale>.pgm`

- `<userid>` es el user id de la persona en la imagen (an2i, at33, boland, bpm, ch4f, cheyer, choon, danieln, glickman, karyadi, kawamura, kk49, megak, mitchell, night, phoebe, saavik, steffi, sz24 y tammo).
- `<pose>` representa la posición de la cabeza, y puede tomar 4 valores: straight, left, right, up.
- `<expression>` es la expresión del rostro de la persona: neutral, happy, sad, angry.
- `<eyes>` es la apariencia de la persona: open, sunglasses.
- `<scale>` es la escala de la imagen. El 4 indica 1/4 de la resolución original.

Para facilitarle la lectura de estos datos en este laboratorio, hemos codificado el script `src/parser.R` que toma el directorio `data/faces_4` y parsea sus datos a un archivo CSV. La salida del parser puede encontrarlo en `data/faces.csv`. Para los siguientes ejercicios, trabajará directamente con el archivo `data/faces.csv` como dato de entrada.

[1] Ejercicio extraido del curso Machine Learning dictado por Tom Mitchell (Carnegie Mellon University, Octubre 1997).
[2] Si le interesa indagar con mayor profundidad, este problema se presenta en [3] y en [4], sección 4.7, pag. 112-117.
[3] Puede encontrar los datos originales y más información sobre los mismos en `www.cs.cmu.edu/~tom/faces.html` y en `http://archive.ics.uci.edu/ml/datasets/CMU+Face+Images`.

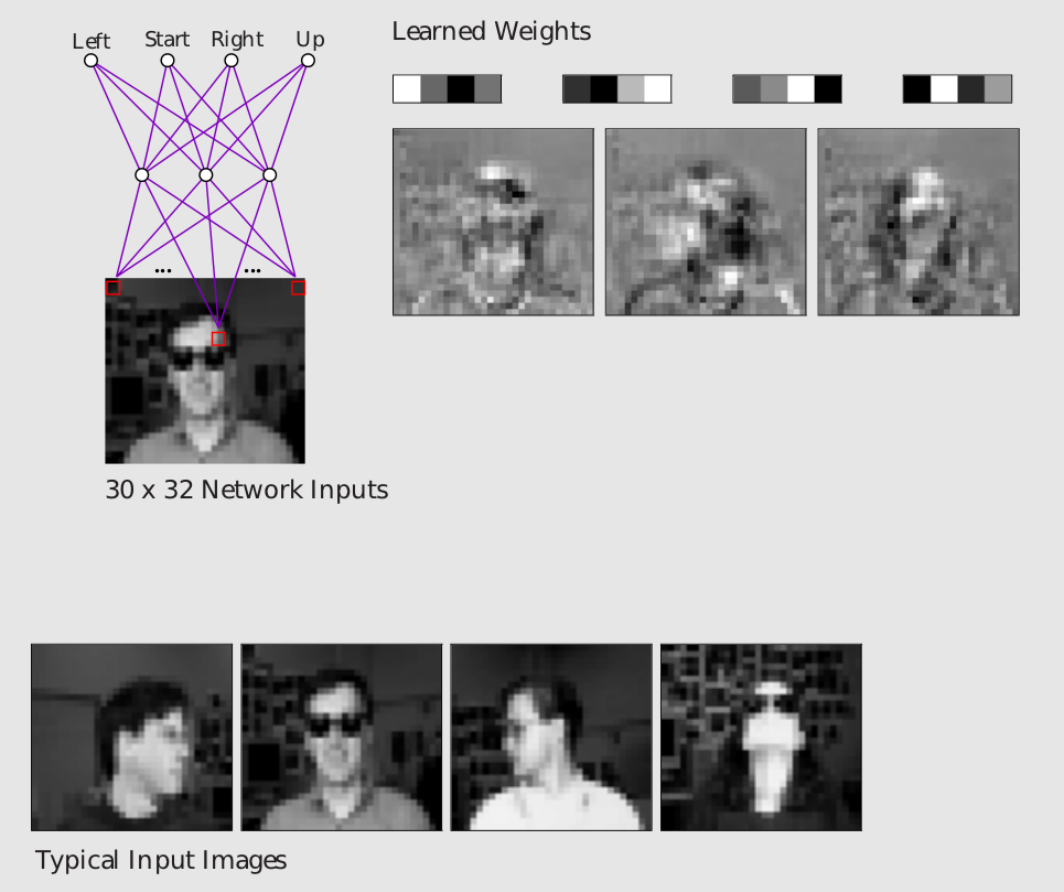

Figura 1: (Imagen extraida de [3]) Red neuronal (arriba a la izquierda) entrenada para reconocer si la persona está mirando a la izquierda, a la derecha, adelante o hacia arriba. Después de entrenar la red con 260 ejemplos, alcanza una precisión del 90 % clasificando nuevas imágenes. Una precisión similar se obtiene cuando se entrena la red para reconocer a una persona individual. Los diagramas en la parte superior derecha representan los valores de los pesos de la red aprendida. El trabajo presentado por Tom Mitchell en [3] puede encontrarlo dentro del directorio `doc`, es el archivo `Does-ML-Mitchell.pdf`.

§ Problema 1. **[10 pts] ANN vs SVM**

`annvssvm.R` Utilizaremos ANN y SVM, dos de las técnicas más difundidas de Machine Learning, para aprender un clasificador que resuelva el problema del reconocimiento de orientación facial.

Haciendo uso del conjunto de datos `faces.csv` lo dividiremos en dos subconjuntos: `train.nset` que consiste en tomar el 70 % de los ejemplos y `test.set` tomando el 30 % restante. Ahora aprenderemos un ANN usando el `trainset` llamando a la función `nnet(formula, data, size, MaxNWts)`. El parámetro `MaxNWts` se establece en 3000, mientras que `size` debe ser igual a 3 (número de neuronas en la capa oculta). Por otro lado aprendemos un SVM también con `trainset` haciendo uso de la función `svm(formula, data)`.

Obtenidos ambos clasificadores, clasificaremos los ejemplos disponibles en `testset`. Para el caso de ANN, es necesario llamar a la función `predict(model, data, type)` donde `type="class"` y para SVM solo es necesario llamar a `predict(model, data)`.

Dadas las salidas de ambos clasificadores, evaluaremos su performance mediantes las siguientes medidas: *Accuracy*, *Precision*, *Recall* y *F-measure*. De este modo construiremos la *confusion matrix* utilizando el método `confusion.matrix()`. Esto devuelve una matrix de $K \times K$ donde cada $k \in K$ representa una de las clases del problema de clasificación. Cada fila i de la columna k-ésima de la matriz, representa el número de instancias que un clasificador clasifico como la clase i siendo que esta pertenece a la clase k. De esta manera, dada una clase k podemos obtener la siguiente información: la celda (k, k) representa el número de *true positives* (TP), la sumatoria $\sum_{i \neq k, j \neq k}(i, j)$ representa el número de *true negatives* (TN), la sumatoria $\sum_{i \neq k, j = k}(i, j)$ representa el número de *false positives* (FP), la sumatoria $\sum_{i = k, j \neq k}(i, j)$ representa el número de *false negatives* (FN). Así podemos calcular nuestras medidas de la siguiente manera:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$precision = \frac{TP}{TP + FP} \tag{2}$$

$$recall = \frac{TP}{TP + FN} \tag{3}$$

$$fmeasure = 2\,\frac{precision \times recall}{precision + recall} \tag{4}$$

§ Problema 2. **[15 pts] Parametrizando SVM con cross-validation**

`svm.R` SVM soporta una gran variedad de parámetros que permiten ajustar el modelo aprendido a las necesidades del problema. En este caso vamos intentar mejorar el clasificador aprendido en el Problema 1 ajustando los valores de los parámetros `cost` y `gamma`[4] según la siguiente tabla:

Parámetros	**Rango de valores**
`gamma`	[0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1]
`cost`	[1, 10, 50, 100, 400, 1000]

Para ello debe trabajar con el template `svm.R` completando el código que se le pide. Empiece a seguir la lógica desde la función `run.experiment.svm.kfold`

(a) Implemente la técnica *5-fold cross-validation* para encontrar el mejor clasificador posible dada la información de la tabla. Tenga en cuenta que el dataset original se debe separar en 70-30, en `trainset` y `testset` respectivamente. Correr *5-fold cross-validacion* sobre el conjunto de entrenamiento.

(b) Explique porque se usa un conjunto de entranamiento para aprender el modelo, es decir, ¿porqué no hacerlo sobre el conjunto completo (`trainset` $\cup$ `testset`)?.

(c) Reportar el error promedio de validación (sobre las 5 corridas del *5-fold cross-validation*) y el error de testeo y comparar este último con los resultados del Problema 1.

[4] La llamada a la función sería `svm(formula, data, gamma, cost)`

§ Problema 3. **[45 pts] Backpropagation**

`backpropagation.R` Implementar el algoritmo Backpropagation para obtener una red neuronal que sirva de clasificador para el problema de orientación de rostros en imágenes. Utilizaremos como guía el pseudocódigo presentado en la sección 5.3 de [1].

Dada un conjunto de ejemplos $\{\mathbf{x}_n\}_n^N$ y una red neuronal de la forma

$$y(\mathbf{x}, \mathbf{w}) = f(a_k) \tag{5}$$

$$\begin{aligned} a_k &= \sum_j^M w_{kj}\ z_j \\ z_j &= h(a_j) \\ a_j &= \sum_i^D w_{j,i} x_i, \end{aligned}$$

donde implementaremos la función $f()$ como la función *softmax*, $h()$ como la función *sigmoid* y $\mathbf{w}$ es el vector de pesos que parametrizan la red, en el cual incorporaremos el parámetro bias. Backpropagation es un algoritmo que a partir del conjunto de datos aprende eficazmente los pesos $\mathbf{w}$ de (5) mediante la minimización de una función de error. Para nuestro problema consideraremos la siguiente función de error, considerada para el caso multiclase (leer en apunte teórico):

$$E_n(\mathbf{w}) = -\sum_{k=1}^{K} t_{kn} \log(y_k(\mathbf{x}_n, \mathbf{w})). \tag{6}$$

donde t_{nk} es la clase para el n-esimo ejemplo del conjunto de ejemplos y puede tomar los valores $\{left, right, straight, up\}$. Aclaramos que se utiliza un esquema de codificación 1-of-K para cada clase t asignándole a cada una los siguientes valores: 1000,0100,0010,0001.

Dado que el gradiente de (**??**) no tiene forma cerrada, debemos recurrir a un método de optimización númerico para minimizarlo. Para nuestro caso implementaremos el método de *gradiente descendente*, esto es:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta\ \nabla E_n(\mathbf{w}^{\tau})$$

Tendremos en cuenta la siguiente arquitectura para la red: $D = 960$ variables de entrada, esto es, una por cada atributo de nuestro conjunto de ejemplos; $M = 3$ variables ocultas y $K = 4$ variables de salida.

Grafique como evolucionan los valores de $E(\mathbf{w})$ durante la ejecución de backpropagation. Note que la función `backprop` regresa el error en cada iteración.

Finalmente, la función `clasificar` debe implementar la clasificación utilizando los pesos de la red neuronal previamente aprendida en `backprop` y los datos del conjunto de testeo.

§ Problema 4. **[30 pts] Generalizando backpropagation a tres capas (no obligatorio)**

Reformular cada uno de los pasos del algoritmo Backpropagation para una red feed-forward pero esta vez con 2 capas ocultas, es decir, una capa de entrada, dos capas ocultas y una capa de salida. Para ello considere leer atentamente como se derivó el algoritmo para una capa oculta en [2], p.227-230, y reproduzca los pasos.

Referencias

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007.

[2] C.M. Bishop. *Pattern recognition and machine learning*, volume 4. Springer New York, 2006.

[3] T.M. Mitchell. Does machine learning really work? *AI magazine*, 18(3):11, 1997. [link].

[4] T.M. Mitchell. *Machine learning*. McGraw Hill, 1997.