Aprendizaje automático:
Clasificación supervisada y su Evaluación

Juan Miguel García Gómez, Carlos Sáez Silvestre
juanmig@upv.es, carsaesi@upv.es, UPV

Diciembre de 2023

Índice

1. Objetivo de los ejercicios

En esta práctica experimentaremos con métodos de aprendizaje automático para obtener modelos predictivos que permitan clasificar nuevas muestras según un conjunto de características. Así mismo, utilizaremos diferentes métricas de evaluación de modelos predictivos y métodos de remuestreo para evaluación.

2. Materiales

- Matlab, con el toolbox Statistics.
- Fichero corpus.mat con 209 muestras supervisadas de entrenamiento y 91 muestras de test

- Xpitraining: 209 muestras simuladas compuestas por 15 variables predictoras correspondientes a concentraciones de sustancias medidas por espectroscopía.
- Ytraining: etiquetas de las 209 muestras correspondientes a tres clases de sustancias.
- Xpitest: 91 muestras de test, procedentes de las mismas distribuciones que Xpitraining
- Ytest: etiquetas de las 91 muestras de test.

- Fichero handson.m es una plantilla de fichero de código matlab para guiar la resolución de la práctica (siéntete libre de utilizarla o no).
- Funciones auxiliares proporcionadas junto al enunciado.

3. Evaluación de la práctica

La evaluación de la práctica consistirá en la entrega del informe que recoja los resultados de la experimentación propuesta por los ejercicios, incluidas las preguntas planteadas.

Responde a las cuestiones de forma específica y concisa, incluyendo el código fuente implementado para resolver la cuestión.

4. Ejercicio 1. Clasificador Gaussiano lineal y cuadrático

4.1. Objetivo

El clasificador generativo Gaussiano asume que las clases están definidas por funciones de densidad de probabilidad Gaussianas D-dimensionales, $p(\mathbf{x}|c) \sim \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c), c = 1, \ldots, C$ con una probabilidades a priori $p(c)$, y por lo tanto con funciones discriminantes proporcionales a las probabilidades a posteri de cada clase c dada la observación x:

$$
\begin{aligned}
g_c(\mathbf{x}) &= p(c|\mathbf{x}) && (1)\\
&\equiv \log p(c) - \frac{1}{2}\mathbf{x}^{\mathrm{T}}\Sigma_c\mathbf{x} - \mathbf{x}^{\mathrm{T}}\Sigma_c^{-1}\boldsymbol{\mu}_c - \frac{1}{2}\log|\Sigma_c| - \frac{1}{2}\boldsymbol{\mu}_c^{\mathrm{T}}\Sigma_c^{-1}\boldsymbol{\mu}_c. && (2)\\
&\equiv \mathbf{x}^{\mathrm{T}}\mathbf{W}_c\mathbf{x} + \mathbf{w}_c^{\mathrm{T}}\mathbf{x} + w_{c0}, && (3)
\end{aligned}
$$

donde

$$
\begin{aligned}
\mathbf{W}_c &= -\frac{1}{2}\Sigma_c^{-1} && (4)\\
\mathbf{w}_c &= \Sigma_c^{-1}\boldsymbol{\mu}_c && (5)\\
w_{c0} &= \log p(c) - \frac{1}{2}\log|\Sigma_c| - \frac{1}{2}\boldsymbol{\mu}_c^{\mathrm{T}}\Sigma_c^{-1}\boldsymbol{\mu}_c. && (6)
\end{aligned}
$$

Cuando las matrices de covarianza son comunes a todas las clases, esto es, $\Sigma_c = \Sigma$, entonces el clasificador gaussiano es lineal ya que el parámetro $\mathbf{W}_c$ también es común y, por tanto, no aporta ninguna información para clasificar.

$$g_c(\mathbf{x}) = \mathbf{w}_c^{\mathrm{T}}\mathbf{x} + w_{c0}, \tag{7}$$

con

$$\mathbf{w}_c = \Sigma_c^{-1}\boldsymbol{\mu}_c \tag{8}$$

$$w_{c0} = \log p(c) - \frac{1}{2}\boldsymbol{\mu}_c^{\mathrm{T}}\Sigma_c^{-1}\boldsymbol{\mu}_c. \tag{9}$$

La estimación de un clasificador Gaussiano viene dada por la estimación por máxima verosimilitud de los parámetros de sus distribuciones de probabilidad.

$$\hat{p}(c) = \frac{N_c}{N} \tag{10}$$

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{N_c}\sum_{n:c_n=c}\mathbf{x}_n \tag{11}$$

$$\hat{\Sigma}_c = \frac{1}{N_c}\sum_{n:c_n=c}(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_c)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_c)^{\mathrm{T}}, \tag{12}$$

donde $c = 1, \ldots, C$, N es el número total de observaciones y N_c es el número total de observaciones de la clase c.

Utiliza la implementación de clasificadores generativos Gaussianos de Matlab (*classify*) para solucionar un problema de clasificación binaria y otro de clasificación de múltiples clases.

4.2. Notas

- Utiliza la documentación de Matlab (*doc classify*) para especificar correctamente los parámetros de entrada y el resultado de la función *classify*

4.3. Desarrollo

1. Clasifica el conjunto de test de las clases 1 y 2 mediante un clasificador Gaussiano lineal entrenado con los datos de entrenamiento.

   a) ¿Qué contiene la variable de salida POSTERIOR?

   b) Observa la salida coeff, ¿cuál es la frontera de decisión obtenida por el clasificador?

2. Clasifica el conjunto de test de las clases 1 y 2 mediante un clasificador Gaussiano cuadrático entrenado con los datos de entrenamiento. ¿Mejora los resultados del clasificador lineal?

3. Clasifica el conjunto de test de las clases 1, 2 y 3 con un clasificador lineal.

   a) Calcula el error de test.

   b) Utiliza la función confus para conseguir la matriz de confusión del clasificador.

4. Clasifica las 3 clases utilizando las 10 primeras componentes PCA. Para ello:

a) Tipifica el conjunto de entrenamiento y guarda las medias y varianzas para aplicarlas posteriormente para tipificar el test manualmente.

b) Estudia la normalidad de algunas de las variables mediante el test de Kolmogorov-Smirnov (*kstest*).

c) Obten mediante el método PCA (*pca*) las componentes principales (PC, loadings), y la proyección (scores) a partir de corpus de entrenamiento tipificado.

d) Proyecta (obten scores) el conjunto de test tipificado mediante la matriz de loadings.

e) Clasifica el conjunto de test proyectado a las 10 primeras PC, mediante un clasificador lineal entrenado con los scores del conjunto de entrenamiento.

f) Calcula el error de test.

5. Ejercicio 2. Clasificador K vecinos

5.1. Objetivo

Los modelos basados en el vecino más próximo (*nearest neighbour* y *K-nearest neighbour*) son modelos no paramétricos basados en distancias. Estos modelos asumen que el espacio muestral es un espacio métrico $\{X, d\}$, donde X es el conjunto de puntos u observaciones y d es una métrica o distancia, definida como $d : X \times X \rightarrow \mathbb{R}$. Además, una métrica debe cumplir las siguientes propiedades para todo $\mathbf{x}_i \in X$:

- No negativa: $d(\mathbf{x}_1, \mathbf{x}_2) \geq 0$. Si y solo si $\mathbf{x}_1 = \mathbf{x}_2$ entonces $d(\mathbf{x}_1, \mathbf{x}_2) = 0$.
- Simétrica: $d(\mathbf{x}_1, \mathbf{x}_2) = d(\mathbf{x}_2, \mathbf{x}_1)$.
- Desigualdad triangular: $d(\mathbf{x}_1, \mathbf{x}_2) + d(\mathbf{x}_2, \mathbf{x}_3) \geq d(\mathbf{x}_1, \mathbf{x}_3)$.

Si se dispone de un conjunto de observaciones $\mathcal{S} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$, donde $\mathbf{x}_i \in X,\ i = 1, \ldots, N$ e y_i es la clase a la que pertenece a observación i-ésima, se puede establecer un espacio métrico $\{X, d\}$. Una vez establecido el espacio métrico, la clase a la que pertenece una nueva muestra $\mathbf{x}$ se calcula en base a la observación u observaciones más cercanas según la distancia $d(\cdot, \cdot)$. Esto es, si los puntos vecinos son de la clase y, entonces se asigna la clase y a la observación nueva $\mathbf{x}$.

5.2. Notas

- Utiliza la función de Matlab (*fitcknn*) para entrenar un modelo de k-vecinos , y posteriormente la función *classify* para clasificar los datos de test con dicho modelo.

5.3. Desarrollo

1. Clasifica el conjunto de test de las clases 1, 2 y 3 con un k=1.

   a) Calcula el error de test.

   b) Utiliza la función confus para conseguir la matriz de confusión del clasificador.

2. Repite el proceso anterior para k desde 1 a 90 y muestra en un gráfico la dependencia del accuracy con el numero de vecinos K. ¿Como ha evolucionado el error de test? ¿Por qué?

6. Ejercicio 3. Redes Neuronales

En este ejercicio vamos a explorar las capacidades de clasificación de las redes neuronales en función de su arquitectura, el tipo de frontera a resolver, la aplicación o no de estrategias de regularización, y los parámetros de entrenamiento. Para ello vamos a utilizar una aplicación web que permite entrenar y testar distintas arquitecturas de redes neuronales para distintos conjuntos de datos y evaluar como son las fronteras de decisión obtenidas en cada caso. Esta aplicación web la podeis encontrar en `http://playground.tensorflow.org/`.

6.1. Desarrollo

1. Con los parámetros existentes por defecto y para cada uno de los 4 conjuntos de datos, evalúa cuantas capas ocultas (hidden layers) son necesarias para poder generar una fronteras de decisión aceptable. Para este apartado utiliza por defecto 4 neuronas por capa.

2. Para un problema complejo como es el del conjunto de datos espiral y utilizando una arquitectura de dos capas, evalúa la influencia del número de neuronas por capa en la habilidad para generar fronteras complejas.

3. Para el segundo conjunto de datos *Exclusive or*, y con la arquitectura por defecto estudia la influencia del ratio de aprendizaje. ¿En qué influye? ¿Qué ventajas e inconveniente presenta el utilizar parámetros demasiado altos o demasiado bajos?

4. ¿Qué ocurre con el error de test, cuando aumentamos el ruido del problema y utilizamos una red con una arquitectura extremadamente sobredimensionada -muchas capas y muchas neuronas por capa-? ¿Se puede solucionar aplicando algún método de regularización? Para este apartado debes dejar entrenar la red con un número de iteraciones alto y puedes probar con el conjunto de datos *Exclusive or* y un nivel de ruido de 50.

Referencias

[1] Richard O Duda, Peter E Hart, David G Stork. Pattern Classification (2nd edition). Wiley Interscience, 2000.

[2] Christopher M Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

[3] Juan M García-Gómez, Salvador Tortajada, Carlos Sáez. Sistemas de ayuda a la decisión médica. Editorial UPV, 2019.