Sistemas de Aprendizaje Automático

Conforme a contenidos del «Curso de Especialización en Inteligencia Artificial y Big Data»

Universidad de Castilla-La Mancha

Escuela Superior de Informática
Ciudad Real

Índice general

Listado de acrónimos

Capítulo 3

Aprendizaje Supervisado

Francisco Pascual Romero

Existen multitud de problemas que requieren un modelo de predicción sobre una variable objetivo dadas un cierto número de variables de entrada. Estos problemas pueden resolverse mediante técnicas de **aprendizaje supervisado**. El calificativo de **supervisado** viene dado por la disponibilidad de ejemplos que contienen los valores de la salida requerida, por ejemplo, si el paciente ha sido diagnosticado con una enfermedad cardiovascular o no. Estos ejemplos *etiquetados* se obtienen o bien porque estas etiquetas o valores se han proporcionado por un experto humano o bien porque se han extraído mediante algún proceso automático de consulta, por ejemplo, los clientes que no han pagado sus cuotas durante los últimos tres meses pueden ser etiquetados como "morosos". De esta manera, los valores de los campos objetivo, junto con los campos de entrada, deben recogerse para cada instancia o ejemplo en un conjunto de datos estructurado que se utiliza para entrenar el modelo. Los algoritmos aprenden un modelo que asigna los datos de entrada a un valor de campo objetivo predicho.

El principal objetivo del aprendizaje supervisado es inferir un modelo de predicción a partir de un conjunto de datos de entrenamiento que contiene **ejemplos previamente etiquetados**, es decir, instancias sobre las que conocemos la respuesta del problema a resolver. De esta forma se puede construir un modelo que sirva para hacer predicciones a partir de datos aún no existentes y/o futuros.

El aprendizaje supervisado también recibe el nombre de **aproximación de funciones** ya que ante unos datos en los que conocemos tanto la entrada como la salida del proceso, se inferirá la función que dé lugar a la segunda a partir de los primeros. Un algoritmo de aprendizaje supervisado genera una función que asocia las entradas del sistema con las salidas deseadas, para ello, utiliza conocimientos previos

(ejemplos) para obtener resultados futuros. En los problemas de aprendizaje supervisado el sistema intenta aprender (o aproximar el comportamiento de) una función que asocia vectores de información (entradas) a una de entre varias clases (salidas) mirando ejemplos y asociaciones previas.

- En el aprendizaje supervisado se dispone de un conjunto de entrenamiento con ejemplos y las salidas correspondientes a esos ejemplos.
- El algoritmo debe inferir un modelo que prediga la salida de un nuevo ejemplo a partir de el histórico de ejemplos de entrenamiento.

Uno de los ejemplos más clásico de aplicación de técnicas de aprendizaje supervisado es el filtrado de *spam*. Para obtener un modelo que descarte los mensajes etiquetados como *spam* es necesario contar previamente con un conjunto entrenamiento que contenga un volumen significativo de mensajes que estén etiquetados como "*spam*" o "*no spam*".

3.1. Clasificacion vs Regresión

Se distinguen dos tareas principales dentro del aprendizaje supervisado dependiendo de la naturaleza de la variable a predecir: clasificación y regresión.

3.1.1. Clasificación

Los problemas son de clasificación cuando el campo objetivo es categórico. En este caso la etiqueta que acompaña a cada ejemplo, y que es necesario predecir, es discreta, es decir, está compuesta por valores finitos sin necesidad de tener una relación de orden entre ellos, por ejemplo: spam/no spam, bueno/regular/malo, rojo/amarillo, verde; etc.

Para estos problemas se utiliza un algoritmo de aprendizaje automático para construir un modelo que prediga una categoría (etiqueta o clase) para un nuevo ejemplo (instancia). Es decir, çlasifica"las nuevas instancias en un conjunto determinado de categorías (o valores discretos).

En clasificación el objetivo es predecir la categoría o categorías a las que pertenecen nuevas instancias basándonos en los ejemplos de entrenamiento.

Un problema de clasificación puede ser binario, en el caso de que únicamente existan dos alternativas (Si/No, Verdadero/Falso, etc.) o multi-clase, en el que existen más de dos alternativas de clasificación, por ejemplo, cuando se le asignan automáticamente varias etiquetas de las posibles al contenido de una noticia, por ejemplo, *deportes* y *política*.

La validación de los modelos que resuelven este tipo de problemas se realiza habitualmente mediante el porcentaje de elementos clasificados correctamente. Las herramientas más utilizadas son la **matriz de confusión** y la **curva ROC** ((Receiver Operating Characteristic).

La **Matriz de Confusión** es una tabla que permite cruzar las predicciones y los valores reales de las clases del campo objetivo para que se puedan visualizar las decisiones correctas así como los errores cometidos por el clasificador.

La **Curva ROC** es la representación gráfica de la proporción de verdaderos positivos (aciertos) proporción de falsos positivos (fallos) según varía el valor a partir del cual se deduce que una instancia es positiva.

3.1.2. Regresión

El análisis de regresión es la parte del aprendizaje supervisado que se ocupa de la predicción de valores numéricos continuos. En este caso, el objetivo del algoritmo es inferir las relaciones entre las variables, que son previamente conocidas y que permiten ofrecer una predicción sobre la salida requerida. Para estos problemas, se utiliza un algoritmo de aprendizaje automático para construir un modelo que prediga un valor continuo. Es decir, dados los campos que definen una nueva instancia, el modelo predice un número real. Por ejemplo, el precio de una casa, el número de unidades vendidas de un producto, los ingresos potenciales de un posible cliente, el número de horas hasta el próximo fallo del sistema, etc.

La regresión es un problema matemático ampliamente conocido en el que se aplican técnicas que van desde las más simples (regresión lineal o polinómica) hasta las más complejas (*Deep Learning*, etc.) pero que se asemeja a un proceso de aproximación de funciones clásico en el que a partir de unas variables de entrada necesitamos encontrar aquella función que ofrezca la salida requerida (ver Fig 3.1). El proceso de validación de los modelos que resuelven problemas de regresión se basa en el uso de **métricas de error**, por ejemplo el error absoluto medio (MAE), el coeficiente de determinación (R^2), etc.

El coeficiente de determinación R^2 mide la proporción de varianza que corresponde a los residuos (diferencia el valor observado de la variable y el valor predicho por el modelo) de un modelo con respecto a la varianza total de los datos. Este coeficiente determina la calidad de una función o modelo para intentar replicar los resultados

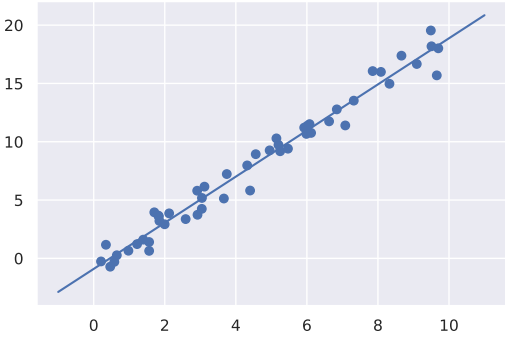

Figura 3.1: Ejemplo de problema de regresión

¿Cuáles de los siguientes problemas son de clasificación y cuáles de regresión?

- ¿Cuánto va a costar una casa?
- Este cliente ¿va a pagar el préstamo?
- ¿Cuántos clientes van a pedir un préstamo el mes próximo?
- ¿Es maligno el tumor?

3.2. Proceso del Aprendizaje Supervisado

Por lo general, el aprendizaje supervisado es un proceso con los siguientes pasos (ver Figura 3.2) [RM19]:

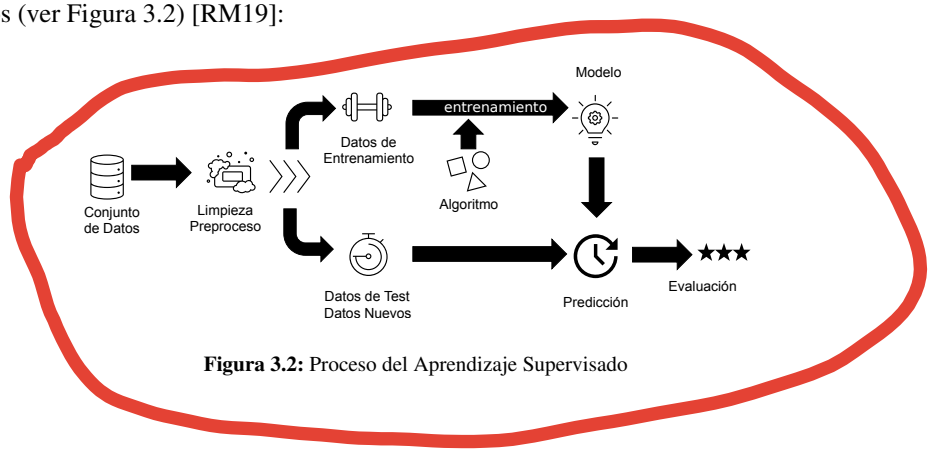

Figura 3.2: Proceso del Aprendizaje Supervisado

1. **Fase de entrenamiento / construcción del modelo**: dado un conjunto de datos de entrada (conjunto de entrenamiento) con ejemplos previamente etiquetados con los valores de salida esperados, el propósito es encontrar un modelo, basado en las características extraídas de las entradas, que explique el valor objetivo.

   El modelo de clasificación resultante es es principalmente una función matemática (o no) que asigna cada ejemplo de entrada a una de las clases predefinidas. Cada clasificador utiliza un algoritmo de aprendizaje que identifica el modelo que mejor encaja con un conjunto de datos específico.

   En los problemas de regresión dispondremos de ejemplos con el valor objetivo conocido, y el modelo obtenido será lo más parecido a una aproximación de una función que asocia los valores de las características con el valor a predecir.

   Normalmente, como conjunto de entrenamiento se utiliza un porcentaje del conjunto de datos disponible de entre un 60 % y un 90 %.

2. **Fase de test / uso del modelo**: Con el fin de evaluar la capacidad de procesamiento de nuevas instancias del clasificador o regresor obtenido se prueba su rendimiento usando un conjunto de datos ya etiquetados, que no se han utilizado en el entrenamiento. Normalmente, para este fin, se utiliza un porcentaje que va desde el 40 % al 10 % del conjunto de datos disponible. El objetivo de esta fase es proporcionar una estimación final e imparcial del rendimiento del modelo obtenido cuando esté preparado para ponerse en producción. Un buen conjunto de pruebas debe arrojar un rendimiento similar al que esperaría al procesar los datos de producción.

Es importante que la división entre datos de entrenamiento y test se haga antes de cualquier transformación (normalización, etc.), si esto no se cumple se provocaría una fuga de datos (*data leakage*). Una fuga de datos es una forma de hacer trampa: estás entrenando o preparando tu modelo con información que no debes conocer. Por lo tanto el consejo es siempre **dividir los datos primero**.

3.2.1. Ajuste y Sobreajuste

Un aspecto muy importante a conocer cuando se está aplicando un algoritmo de aprendizaje supervisado es su capacidad de generalización, es decir, como de flexible va a ser su comportamiento ante el procesamiento de los nuevos casos que se presenten en el futuro.

En la Figura 3.3 se puede ver tres ajustes posibles a un conjunto de datos. En la gráfica de la izquierda, se observa como el modelo representado por una recta produce un ajuste pobre. Muchos puntos quedan alejados de la recta. En la gráfica central, el modelo se ajusta a los puntos. La curva (modelo) pasa por muchos

puntos, pero no por todos. También se observa que este modelo debería predecir correctamente nuevos puntos incluso fuera del rango observado. Finalmente, en la figura de la derecha, se observa un modelo que pasa por casi todos los puntos. Este modelo debe dar el error (*training error*) más bajo de los tres modelos, por lo tanto, debería ser el elegido como el mejor modelo. Sin embargo, este modelo no es bueno en la predicción de nuevos puntos.

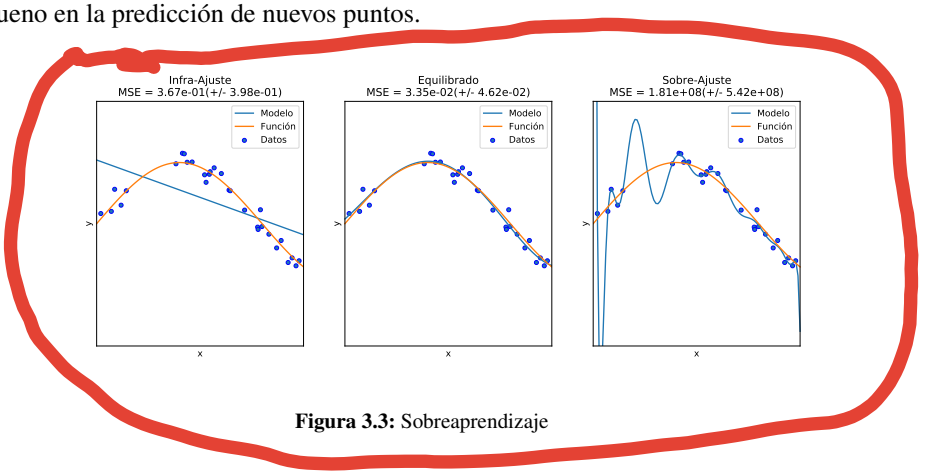

Figura 3.3: Sobreaprendizaje

De esta manera, es necesario resolver cómo obtener un buen ajuste para el conjunto de datos de entrenamiento (ajuste de los parámetros del modelo), y a su vez que sea generalista del comportamiento del fenómeno representado (ajuste de la complejidad del modelo). Para resolver esta cuestión se utiliza la validación cruzada (*Cross-validation*) que consiste en repetir y calcular la media aritmética obtenida de las medidas de evaluación sobre diferentes particiones.

La validación cruzada en el enfoque básico (CV k-fold) consiste en dividir el conjunto de entrenamiento en k subconjuntos más pequeños. Para cada uno de estas divisiones se procede como sigue:

1. Se entrena un modelo utilizando $k - 1$ de los subconjuntos como datos de entrenamiento; 2. el modelo resultante se valida con la parte restante de los datos (es decir, se utiliza como conjunto de pruebas para calcular una medida de rendimiento como la precisión).

2. La medida de rendimiento obtenida mediante la validación cruzada es entonces la media de los valores calculados en el bucle.

La validación cruzada es costosa desde el punto de vista computacional pero ofrece dos ventajas principales: 1) permite estimar como funcionará el modelo en una diversidad de situaciones y 2) aprovecha los datos disponibles, lo que supone una gran ventaja en problemas en el que el número de muestras es muy pequeño y que sucede cuando se fija un conjunto de validación arbitrario.

3.3. Clasificación Bayesiana

Los modelos Naive Bayes son un grupo de algoritmos de clasificación extremadamente rápidos y sencillos que suelen ser adecuados para conjuntos de datos de muy alta dimensión. Como son tan rápidos y tienen tan pocos parámetros ajustables, acaban siendo muy útiles como línea base de elaboración para la resolución un problema de clasificación.

3.3.1. Clasificación Bayesiana

Los clasificadores *Naive Bayes*[WKM10] se basan en métodos de clasificación bayesianos, esto es, están fundamentados matemáticamente en el **Teorema de Bayes**. Este teorema describe mediante una fórmula relación de las probabilidades condicionales de las diferentes valores entre las variables existentes.

Dado un ejemplo x representado por k valores, el clasificador Naïve Bayes se basa en encontrar la hipótesis más probable que describa a ese ejemplo. Si la descripción de ese ejemplo viene dada por los valores $(a_1, \ldots, a_n)$, la hipótesis más probable será aquella que cumpla:

$$v_{MAP} = argmax_(v_j \in V) P(v_j | a_1 \ldots a_n) \tag{3.1}$$

Es decir, la probabilidad de que, conocidos los valores que describen a ese ejemplo, éste pertenezca a la clase v_j, donde v_j es el valor de la función de clasificación $f(x)$ en el conjunto infinito V.

Por el teorema de Bayes:

$$v_{MAP} = argmax_{v_j \in V} \frac{P\left(a_1 \cdots a_n \vdots v_j\right) P\left(v_j\right)}{P\left(a_1 \cdots a_n\right)} \tag{3.2}$$

que es igual a:

$$argmax_{v_j \in V} P\left(a_1 \cdots a_n \vdots v_j\right) P\left(v_j\right) \tag{3.3}$$

Podemos estimar $P(v_j)$ contando las veces que aparece el ejemplo v_j en el conjunto de entrenamiento y dividiéndolo por el número total de ejemplos que forman este conjunto.

Para estimar el término $P(a_1, \ldots, a_n)$, es decir, las veces en que para cada clase aparecen los valores del ejemplo x, se debe recorrer todo el conjunto de entrenamiento. Éste cálculo resulta impracticable para un número suficientemente grande de ejemplos por lo que se hace necesario simplificar la expresión anterior. Para ello se recurre a la hipótesis de independencia condicional con el objeto de poder factorizar la probabilidad.

Esta hipótesis dice lo siguiente: "*Los valores a_j que describen un atributo de un ejemplo cualquiera x son independientes entre sí conocido el valor de la categoría a la que pertenecen*". Así, la probabilidad de observar la conjunción de atributos a_j dada una categoría a la que pertenecen es justamente el producto de las probabilidades de cada valor por separado:

$$P\left(a_1 \cdots a_n \vdots v_j\right) = \prod_i P\left(a_i \vdots v_j\right) \tag{3.4}$$

Resumen En la clasificación bayesiana, el objetivo es encontrar la probabilidad de una etiqueta dada de acuerdo a algunas características observadas, lo cual se podría escribir como

$$P(L \mid features)$$

. El teorema de Bayes nos dice cómo expresar esto en términos de cantidades que se pueden calcular más directamente de la siguiente forma:

$$P(L \mid \text{features}) = \frac{P(\text{features} \mid L)P(L)}{P(\text{features})}$$

Si el objetivo es decir entre dos clases, C_1 y $C2$ por ejemplo, entonces, una forma de poder tomar esta decisión es calcular el ratio de las probabilidades a posteriori para cada una de las clases.

$$\frac{P(C_1 \mid \text{features})}{P(C_2 \mid \text{features})} = \frac{P(\text{features} \mid C_1)}{P(\text{features} \mid C_2)} \frac{P(C_1)}{P(C_2)}$$

Todo el conocimiento que se necesita para efectuar esta clasificación es el modelo de probabilidad de cada clase para cada conjunto de características. Este modelo recibe el nombre de "modelo generativo"porque establece el proceso aleatorio hipotético que genera los datos, es decir, que con las distribuciones especificadas en el modelo podríamos generar el conjunto de datos o uno de similares características estadísticas. Establecer este modelo generativo para cada clase es la labor que se realiza en el proceso de entrenamiento de un clasificación bayesiano.

Esta tarea puede llegar a ser muy compleja pero utilizando ciertas simplificaciones se logra construir un clasificador reducido y eficiente. Estas simplificaciones hacen denominar al clasificador con el calificativo de ingenuo (*naïve*). Si suponemos ïngenuamente"que los datos tienen ciertas características, como la mencionada anteriormente de la independencia, y estas suposiciones las extendemos a los modelos generativos de cada etiqueta, podemos encontrar una aproximación útil del modelo generativo concreto de cada clase. Según que suposiciones se den por ciertas se tendrá un tipo de clasificador bayesiano u otro.

3.3.2. Clasificadores Gaussianos

El clasificador Bayesiano más fácil de entender es aquel que, a partir de características modeladas como números reales utiliza modelos generativos de cada una de las clases están basados en la **distribución gaussiana simple**.

Una forma extremadamente rápida de crear un modelo sencillo es suponer que los datos están descritos por una distribución gaussiana sin covarianza entre dimensiones. Este modelo puede ajustarse simplemente encontrando la media y la desviación estándar de los puntos dentro de cada clase.

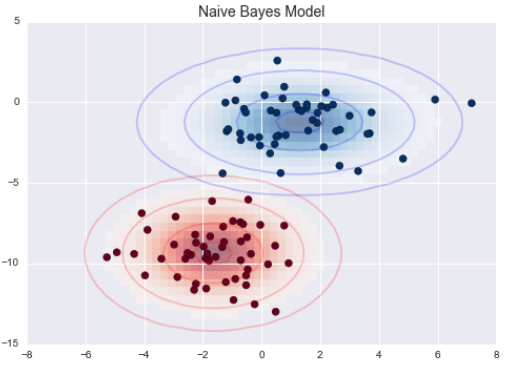

Figura 3.4: .

En la Figura 3.4 las elipses representan aquí el modelo generativo gaussiano para cada clase dentro de un conjunto de datos sintético, con mayor probabilidad hacia el centro de las elipses. Con este modelo generativo para cada clase, tenemos una forma sencilla para calcular la probabilidad $P(features \mid L_1)$ para cualquier punto de datos, y así podemos calcular rápidamente la proporción posterior y determinar qué etiqueta es la más probable para un punto dado.

Un clasificador bayesiano siempre proporcionará las probabilidades a posteriori de las clases que se buscan, de esta manera, resultan ser herramientas muy útiles en el proceso de estimar la incertidumbre en la clasificación. En este caso, la clasificación final sólo será tan buena como las suposiciones del modelo que la conducen, por lo que este clasificador a menudo no produce muy buenos resultados. Sin embargo, en muchos casos -especialmente cuando el número de características es grande- esta suposición no arroja tan malos resultados como para impedir que Gaussian Naive Bayes sea un método útil.

En el siguiente código (Listado 3.1) se puede observar como se implementaría un algoritmo de clasificación NaiveBayes desde cero con la suposición de que todas las variables pueden ser generadas mediante una distribución gaussiana.

Listado 3.1: Clasificador Naive Bayes en Python

```python
import numpy as np

class NaiveBayes:

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # calcular media, varianza y proporciones a priori de cada clase
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors =  np.zeros(n_classes, dtype=np.float64)

        for idx, c in enumerate(self._classes):
            X_c = X[y==c]
            self._mean[idx, :] = X_c.mean(axis=0)
            self._var[idx, :] = X_c.var(axis=0)
            self._priors[idx] = X_c.shape[0] / float(n_samples)

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        posteriors = []

        # se calcular la probabilidad a posteriori de cada clase según
        # teorema de bayes
        for idx, c in enumerate(self._classes):
            prior = np.log(self._priors[idx])
            posterior = np.sum(np.log(self._pdf(idx, x)))
            posterior = prior + posterior
            posteriors.append(posterior)

        # resultado: clase con la probabilidad a posteriori más alta
        return self._classes[np.argmax(posteriors)]


    def _pdf(self, class_idx, x):
        mean = self._mean[class_idx]
        var = self._var[class_idx]
        numerator = np.exp(- (x-mean)**2 / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator
```

En el método fit se procede a la construcción o entrenamiento del modelo. En este caso se aprenden los parámetros que definen cada distribución de probabilidad, por ejemplo, la media, la varianza y la probabilidad a priori de cada una de las clases que existen en los datos. El método $predict$ servirá para, mediante la aplicación del teorema de bayes, obtener la probabilidad a posteriori de cada clase ante un nuevo caso de variables de entrada, ofreciendo como resultado aquella clase que obtengan un valor más alto. La función pdf sirve para calcular el valor procedente de la fórmula que representa a la distribución normal.

3.3.3. Clasificadores Multinomiales

La suposición gaussiana no es la única suposición que puede utilizarse para especificar la distribución generativa de cada clase. Otro ejemplo útil es la implementación basada en la distribución multinomial. En ella se supone que las características se generan a partir de una distribución multinomial simple. La distribución multinomial describe la probabilidad de observar recuentos entre una serie de categorías y, por tanto, el Bayes ingenuo multinomial es el más apropiado para las características que representan recuentos o tasas de recuento.

La distribución multinomial describe la probabilidad de obtener un número específico de conteos para k resultados diferentes, cuando cada resultado tiene una probabilidad fija de ocurrir. Si una variable aleatoria x sigue una distribución multinomial, entonces la probabilidad de que el resultado 1 ocurra exactamente x_1 veces, el resultado 2 ocurra exactamente x_2 veces, el resultado 3 ocurra exactamente x_3 veces, etc. se puede encontrar mediante la siguiente fórmula: $n! * (p_1 x_1 * p_2 x_2 * \cdots * p_k x_k)/(x_1! * x_2! \ldots * x_k!)$

La idea es precisamente la misma que en el clasificador bayesiano gaussiano, salvo que en lugar de modelar la distribución de datos con la distribución gaussiana de mejor ajuste, se modela la distribución de datos con una distribución multinomial de mejor ajuste.

Un problema en el que se utiliza a menudo esta técnica es en la clasificación de textos [MN$^+$98], donde las características están relacionadas con el recuento de palabras o las frecuencias dentro de los documentos a clasificar.

Este clasificador consisten en simplemente en la definición de un modelo de probabilidad para la frecuencia (ponderada) de cada palabra en la cadena; no obstante, el resultado suele ser bastante bueno. De esta manera, se demuestra, que incluso un algoritmo muy tan simple, cuando se utiliza con cuidado y se entrena con un gran conjunto de datos de alta dimensiónalidad (gran número de características), puede ser muy eficaz.

Como ejemplo (ver Listado 3.2) se va a utilizar el conjunto de datos *fetch_20newsgroup* que contiene noticias de 20 grupos diferentes cada uno correspondiente a una categoría ya divididos entre entrenamiento y test. Para construir un clasificador se centrará el ejemplo en las clases relacionadas con el deporte y el motor ('rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey').

Para procesar texto es necesario en primer lugar transformar ese texto en forma de vector numérico por el cual cada característica será una palabra distinta y el valor de esa característica para cada elemento (noticia) será el número de veces (normalizado) que aparece esa palabra en la noticia correspondiente.

Listado 3.2: Clasificador Naive Bayes Multinomial con scikit learn

```
from sklearn.datasets import fetch_20newsgroups
# Carga de Datos y selección de Categorías.
data = fetch_20newsgroups()
categories = ['rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey']

# División entre train y test
train = fetch_20newsgroups(subset='train', categories=categories)
test = fetch_20newsgroups(subset='test', categories=categories)

# Crear Características y establecer modelo clasificador
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline

# Crear el Pipeline: transformación + clasificación
model = make_pipeline(TfidfVectorizer(), MultinomialNB())

# Entrenamiento y Predicción
model.fit(train.data, train.target)
labels = model.predict(test.data)

# Evaluación mediante Accuracy/Exactitud
from sklearn.metrics import accuracy_score
accuracy_score(test.target, labels)
```

Un *Pipeline* es un elemento crucial en cualquier proceso de Machine Learning. Permite aplicar secuencialmente una lista de transformaciones que finaliza en la aplicación de un estimador.

3.3.4. Ventajas e Inconvenientes

Dado que los clasificadores *Naive Bayes* hacen suposiciones muy estrictas sobre los datos, generalmente no obtienen unos resultados de precisión tan altos como modelos más complicados. No obstante, tienen varias ventajas:

- Son extremadamente rápidos tanto para el entrenamiento como para la predicción.
- Proporcionan una predicción probabilística directa.
- Suelen ser muy fáciles de interpretar.
- Tienen muy pocos parámetros ajustables (si es que hay alguno).
- Robustos al posible ruido presente en los ejemplos de entrenamiento y a la posibilidad de tener entre esos ejemplos datos incompletos o posiblemente erróneos.

Estas ventajas implican que un clasificador de este tipo es a menudo una buena opción como clasificación inicial de referencia. Si su rendimiento es adecuado, perfecto, si no funciona bien, entonces se puede empezar a explorar modelos más sofisticados, con algún conocimiento de referencia sobre su rendimiento.

Los clasificadores Naive Bayes tienden a funcionar especialmente bien en una de las siguientes situaciones:

- Cuando los supuestos ingenuos coinciden realmente con los datos (muy raro en la práctica).
- Para categorías muy bien separadas, cuando la complejidad del modelo es menos importante.
- En el caso de datos de muy alta dimensión, cuando la complejidad del modelo es menos importante, dado que los datos por su propia dimensión deben estar naturalmente separados.

Los principales inconvenientes son:

- Requieren eliminar las funciones correlacionadas porque pueden distorsionar el resultado de las probabilidades a priori.
- *Frecuencias cero:* Si una variable categórica tiene una categoría en el conjunto de datos de prueba que no se observó en el conjunto de datos de entrenamiento, entonces el modelo asignará una probabilidad cero. No podrá hacer una predicción. Esto a menudo se conoce como "Frecuencia Cero". Para resolver esto, podemos utilizar la técnica de alisado. Una de las técnicas de suavizado más simples se llama *estimación de Laplace*.

Los parámetros en la distribución multinomial son estimados mediante el recuento de la frecuencia relativa (máxima verosimilitud)

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

donde $N_{yi} = \sum_{x \in T} x_i$ es el número de veces en el que una característica aparece en un elemento de una clase dentro del conjunto de entrenamiento, y $N_y = \sum_{i=1}^{n} N_{yi}$ es el recuento total de elementos de la clase. El parámetro α sirve para evitar el problema de la frecuencia cero. Si $\alpha = 1$ se llama Corrección de Laplace, si $alpha < 1$ se denomina Corrección de Lidstone.

Dada la simplicidad en la implementación un buen ejercicio sería utilizar clasificación bayesiana mediante scikit-learn en los conjuntos de datos de prueba que facilita la librería scikit-learn: *iris, digits, wine y breast_cancer*.

3.4. Vecinos más cercanos

El algoritmo de los vecinos más cercanos (kNN) es uno de los algoritmos de aprendizaje supervisado más sencillos siendo un ejemplo de aprendizaje por analogía.

Aprendizaje por Analogía: El Razonamiento Analógico intenta emular la capacidad humana de recordar la solución de problemas previos ante la aparición de problemas parecidos en los que se llevan a cabo razonamientos análogos para alcanzar sus soluciones respectivas.

Esta estrategia se considera de procesamiento *perezoso*, esta denominación viene dada porque la tarea a realizar, la obtención de una salida a partir de una entrada, se retrasa lo más posible. En este tipo de algoritmos de aprendizaje automático, el modelo son las mismas instancias de entrenamiento, es decir, en vez de inducir un modelo a partir de un conjunto de datos, se utilizan estos mismos datos como modelo sin necesidad de procesamientos adicionales. Se reduce por lo tanto el procesamiento de información Únicamente existe procesamiento al momento de tener que clasificar un nuevo caso, en ese instante se buscan los casos más parecidos y el resultado se obtiene en función de las salidas de esos casos concretos. Los algoritmos más conocidos están basados en la regla del vecino más próximo (NN, $1-NN$ o kNN)

Aprendizaje basado en Instancias (Instance Based Learning) El modelo lo conforman los propios ejemplos de entrenamiento. Para generalizar a nuevas instancias se basa en una medida de **similitud** entre ejemplos. Su denominación proviene del proceso de basar el modelo únicamente en las instancias de entrenamiento. También se conoce como aprendizaje basado en la memoria o aprendizaje perezoso.

3.4.1. El Algoritmo

La técnica más popular dentro de estas estrategias es la Regla del vecino más próximo o Nearest-Neighbour (NN) o su extensión, los k vecinos más cercanos.

La idea básica sobre la que se fundamenta este algoritmo es que un nuevo caso se va a clasificar en la clase más frecuente a la que pertenecen sus k vecinos más cercanos. En el caso de utilizar kNN sobre un problema de clasificación, el algoritmo devolverá la clase a la cual debe pertenecer el caso desconocido. Esta elección se fundamentará en el uso del voto (mayoría) para decidir el valor más adecuado, pudiéndose ser el voto ponderado o no. Por otro lado, si el algoritmo se utiliza para resolver problemas de regresión, el algoritmo devolverá la media de los valores de los vecinos.

Ejemplo

En la Figura 3.5 se observan 10 ejemplos pertenecientes a dos clases distintas, 5 a la clase A en amarillo y 6 a la clase B en morado. Cada ejemplo tiene dos atributos que ayudan a clasificarlo, x_1 y x_2. Como se puede observar, la distancia entre los ejemplos propios de cada clase es por lo general menor que la distancia con los ejemplos de la clase contraria, por lo que aparentan agruparse según su clase. Al clasificar un nuevo ejemplo, representado con la estrella roja del centro. Si se elige $k = 3$, el nuevo ejemplo se clasificará con la clase B, dado que 2 de esos 3 ejemplos más cercanos pertenecen a esa clase. Sin embargo, si elegimos $k = 6$, el ejemplo caerá dentro de la clase A, dado que 4 de los 6 ejemplos pertenecen a esa clase. La k es por lo tanto uno de los parámetros más determinantes del algoritmo

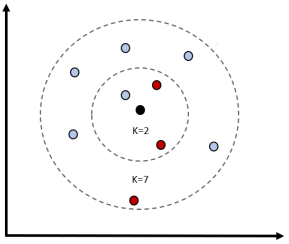

Figura 3.5: Ejemplo de Aplicación de kNN

Función de Distancia

El algoritmo utiliza las distancias entre casos (ejemplos, vectores) para calcular los más cercanos. La elección de la métrica de la distancia también es importante para el rendimiento del algoritmo [AAHL$^+$19]. Algunas de las más populares son las siguientes

1. *Distancía Euclídea*: [1]. Adecuada cuando todas las variables del conjunto de datos son números reales.
2. *Distancia de Manhattan*: Calcula la distancia entre valores numéricos utilizando la suma del valor absoluto de su diferencia. Especialmente indicada cuando el conjunto de datos tiene valores enteros. Tanto la distancia euclídea como la distancia manhattan pueden ser consideradas distancias de Minkovski.
3. *Otras medidas*: Hay muchas otras medidas de distancia que se pueden utilizar, como la distancia Tanimoto, Jaccard, Mahalanobis y coseno, etc..

La elección de la función se distancia se suele hacer a partir de las características del conjunto de datos a analizar. También se puede experimentar con diferentes métricas de distancia ver cual arroja como resultado los modelos más precisos.

[1] Link: http://https://es.wikipedia.org/wiki/Distancia_euclidiana

Parametrización Básica

Uno de los puntos críticos del algoritmo es encontrar el valor adecuado para k, lo cual normalmente se realiza mediante el ajuste o *tuneado* del algoritmo. Para ello, se realiza una prueba iterativa con una cantidad significativa de valores diferentes de k (por ejemplo, valores de 1 a 26) y se comprueba que es lo funciona mejor para la resolución del problema concreto [HAAA14].

En los casos en los que los datos no están muestreados uniformemente, la clasificación de vecinos basada en radios de vecindad ofrece mejores resultados. De esta manera que los puntos de las zonas de vecindad más dispersas utilizan menos vecinos más cercanos para la clasificación. Este método pierde eficacia cuando el conjunto de datos es de alta dimensionalidad ("maldición de la dimensionalidad").

Otro factor importante para considerar como parámetro del algoritmo es el **peso** que se le da para la decisión final a cada uno de los vecinos extraídos como los más cercanos. Este peso puede distribuirse de forma uniforme entre los vecinos, en clasificación sería una votación por mayoría y en la regresión una media clásica, o bien que se tenga en cuenta la distancia de cada caso seleccionado al nuevo caso que pretendemos seleccionar. Por ejemplo, se puede asignar un peso inversamente proporcional a la distancia del mismo al nuevo caso.

Por otro lado, otorgar la misma importancia a todas las variables puede afectar negativamente a los resultados de predicción tras la aplicación del paradigma kNN. Esto será así en el caso de que algunas de las variables sean irrelevantes para la predicción de la variable objetivo. Este es el motivo por el que puede resultar interesante la utilización de funciones de distancia entre casos definidas *ad-hoc* con el fin de ponderar cada variable de una manera adecuada.

Resumen de Parametrización

- **k**: Número de vecinos a considerar en la predicción.
- **radius**: Radio del espacio de parámetros a utilizar para la predicción.
- **weights**: función de ponderación de contribución de los vecinos utilizada en la predicción.
- **distance**: la métrica de distancia a utilizar para determinar los vecinos más cercanos.

Maldición de la dimensionalidad. En un espacio de gran dimensión, las distancias (sobre todo la distancia euclidiana) pierden sentido dado que la distancia entre cada par de puntos es casi la misma, lo cual hace que las funciones de similitud/distancia se comporten mal y perjudica al rendimiento de aquellas técnicas que se basan en su funcionamiento.

Código Fuente

En el siguiente código (Listado 3.3) podemos observar como no hay método de entrenamiento (*fit*) o si lo hubiera no sería de entrenamiento como tal. El método *predict*, permite clasificar un nuevo ejemplo mediante la búsqueda los vecinos más cercanos y tomar la decisión por mayoría (problema de clasificación).

Listado 3.3: Ejemplo de código en python

```python
import numpy as np
from scipy.stats import mode

# Distancia Euclídea
def euclidea(p1,p2):
    dist = np.sqrt(np.sum((p1-p2)**2))
    return dist

# Algoritmo KNN
def knn_predict(x_train, y , x_input, k):
    op_labels = []

    # Recorrer los elementos a clasificar
    for item in x_input:


        point_dist = [] # almacenar distancias

        # para cada elemento de entrenamiento
        for j in range(len(x_train)):
            distances = euclidea(np.array(x_train[j,:]) , item)
            # Calcular la distancia
            point_dist.append(distances)
        point_dist = np.array(point_dist)

        # Ordenar el array manteniendo el índice
        # guardar el top-k de registros
        dist = np.argsort(point_dist)[:k]

        # obtener la labels de cada registro top-k
        labels = y[dist]

        #voto por mayoría
        lab = mode(labels)
        lab = lab.mode[0]
        op_labels.append(lab)

    return op_labels
```

3.4.2. Ventajas e Inconvenientes

Las grandes ventajas de este algoritmo residen en que el coste del entrenamiento es nulo, y además su aplicación no está sujeta a ningúna asumción previa como la independencia, correlación, etc. sobre las características de los elementos a analizar. De esta manera se pueden aprender modelos complejos utilizando funciones sencillas como aproximaciones locales muy tolerantes al ruido sobre todo cuando se utilizan cuando se usan valores de k moderados. Además, es una técnica que sirve tanto para clasificación (por el mecanismo de votación) como para regresión (por la agregación de los valores de los vecinos más cercanos).

El inconveniente principal es el coste de encontrar los k-vecinos más cercanos. Sea cual sea el valor de k la búsqueda requiere explorar todo el conjunto de referencia. Esto significa que el coste de la búsqueda depende linealmente de N. Además, hay que considerar el coste del cálculo de cada distancia. Adicionalmente se debe considerar el espacio de almacenamiento requerido. A esto hay que añadir el proceso de optimización necesario para encontrar el mejor valor de k. Otros inconvenientes son su poca interpretabilidad y su bajo rendimiento si el número de características es muy elevado.

3.4.3. Variantes y Mejoras

Existen diferentes estrategias que permiten la aplicación de la regla del vecino más cercano para grandes volúmenes de datos con un coste computacional menor que el asociado a la fuerza bruta. Estas estrategias pueden agruparse en dos clases, de acuerdo a la filosofía en que se basan:

1. **Reducir el conjunto de referencia**: se basan en la selección de un subconjunto del conjunto de referencia que tenga las mismas propiedades que el conjunto original para, una vez reducido, aplicar la regla del vecino más cercano en su formulación original (fuerza bruta) sobre el nuevo conjunto.

2. **Mejorar la eficiencia computacional**: Consisten en realizar una preindexación del conjunto de datos con el fin de facilitar la localización de los vecinos más cercanos de cada ejemplo. Entre estos métodos están los basados en ordenaciones (se organiza o indexa el conjunto de datos con respecto a una característica) y los jerárquicos (descomposición en árbol de los datos)

De la misma forma que NaiveBayes, kNN se puede utilizar para resolver problemas de clasificación (conjuntos de datos textitiris, digits, wine y breast_cancer) y regresión (boston, diabetes, etc.) utilizando librería scikit-learn. En este caso sería interesante optimizar el rendimiento del algoritmo optimizando sus parámetros.

Bibliografía

[AAHL+19] Haneen Arafat Abu Alfeilat, Ahmad BA Hassanat, Omar Lasassmeh, Ahmad S Tarawneh, Mahmoud Bashir Alhasanat, Hamzeh S Eyal Salman, and VB Surya Prasath. Effects of distance measure choice on k-nearest neighbor classifier performance: a review. *Big data*, 7(4):221–248, 2019.

[HAAA14] Ahmad Basheer Hassanat, Mohammad Ali Abbadi, Ghada Awad Altarawneh, and Ahmad Ali Alhasanat. Solving the problem of the k parameter in the knn classifier using an ensemble learning approach. *arXiv preprint arXiv:1409.0919*, 2014.

[MN+98] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[RM19] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd, 2019.

[WKM10] Geoffrey I Webb, Eamonn Keogh, and Risto Miikkulainen. Naïve bayes. *Encyclopedia of machine learning*, 15:713–714, 2010.