Inteligencia Artificial

CC-421

César Lara Avila

Universidad Nacional de Ingeniería

(actualización: 2020-11-19)

Bienvenidos

Temario de modelos deterministas

- Modelos y datos
- Conceptos y propiedades
- Algoritmos fundamentales
- Anatomía de un algoritmo de aprendizaje automático
- Regularización
- Evaluación de modelos
- Ajuste de hiperparámetros
- Modelos ensamblados
- Transfer Learning

Modelos y datos

¿Por qué un modelo aprendizaje es capaz de predecir correctamente las etiquetas de ejemplos nuevos que no se han visto anteriormente?

Intuitivamente, cuanto más grande es el conjunto de ejemplos de entrenamiento, más improbable es que los nuevos ejemplos sean diferentes de los ejemplos utilizados para el entrenamiento.

PAC

La teoría de aprendizaje PAC (Probably Approximately Correct) ayuda a analizar si, y bajo qué condiciones, un algoritmo de aprendizaje probablemente generará una clasificador correcto.

Además de analizar la capacidad del aprendizaje y la comprensión de la relación entre el error del modelo, el tamaño del conjunto de entrenamiento, la forma de la ecuación matemática que define el modelo y el tiempo que lleva construir el modelo.

Conceptos y propiedades

Estimado insesgado

Se desconoce el PDF f_X y tenemos una muestra $S_X = \{x_i\}_{i=1,\ldots,N}$.

Un estimador insesgado $\hat{\theta}(S_X)$ de alguna estadística θ calculada usando una muestra S_X extraída de una distribución de probabilidad se define como:

$$\mathbf{E}[\hat{\theta}(S_X)] = \theta.$$

donde $\hat{\theta}$ es una estadística muestral obtenida usando una muestra S_X y no la estadística real obtenida desde X.

Intuitivamente, esto significa que si tenemos un número ilimitado de muestras tales como S_X y calculamos algún estimador insesgado, como $\hat{\mu}$, usando cada muestra, entonces el promedio $\hat{\mu}$ es igual a la estadística real μ que se calcularía en X.

Regla de Bayes

La probabilidad condicional $\mathbf{P}(X = x|Y = y)$ es la probabilidad de que la variable aleatoria X tenga un valor específico x dado que otra variable aleatoria Y tiene un valor específico de y.

La Regla de Bayes (o teorema de Bayes) estipula que:

$$\mathbf{P}(X = x|Y = y) = \frac{\mathbf{P}(Y = y|X = x)\mathbf{P}(X = x)}{\mathbf{P}(Y = y)}.$$

La regla de Bayes es útil cuando tenemos un modelo de distribución de X y este modelo f_θ es una función que tiene algunos parámetros en forma de vector θ.

Un ejemplo de tal función puede ser la función Gaussiana (PDF) que tiene dos parametros μ y σ y se define como:

$$f_\theta(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

donde $\theta = [\mu, \sigma]$.

Estimación de parámetros

Esta función tiene todas las propiedades de un PDF. Por lo tanto, podemos usarlo como modelo de una distribución desconocida de X. Podemos actualizar los valores de los parámetros en el vector θ a partir de los datos usando la regla de Bayes:

$$\mathbf{P}(\theta = \hat{\theta}|X = x) \leftarrow \frac{\mathbf{P}(X = x|\theta = \hat{\theta})\mathbf{P}(\theta = \hat{\theta})}{\mathbf{P}(X = x)} = \frac{\mathbf{P}(X = x|\theta = \hat{\theta})\mathbf{P}(\theta = \hat{\theta})}{\sum_{\tilde{\theta}} \mathbf{P}(X = x|\theta = \tilde{\theta})}.$$

donde $\mathbf{P}(X = x|\theta = \hat{\theta}) = f_{\hat{\theta}}$.

Si tenemos una muestra S de X y el conjunto de valores posibles para θ es finito, podemos estimar fácilmente $\mathbf{P}(\theta = \hat{\theta})$ aplicando la regla de Bayes de forma iterativa, un ejemplo $x \in S$ a la vez.

El valor inicial $\mathbf{P}(\theta = \hat{\theta})$ se puede adivinar de manera que $\sum_{\hat{\theta}} \mathbf{P}(\theta = \hat{\theta}) = 1$. Esta suposición de las probabilidades para diferentes $\hat{\theta}$ se llama probabilidad previa (prior).

Estimación de parámetros

Calculamos $\mathbf{P}(\theta = \hat{\theta}|X = x_1)$ para todos los valores posibles $\hat{\theta}$. Antes de actualizar $\mathbf{P}(\theta = \hat{\theta}|X = x)$ esta vez para $x = x_2 \in S$ usando la ecuación anterior reemplazamos $\mathbf{P}(\theta = \hat{\theta})$ por la nueva estimación:

$$\mathbf{P}(\theta = \hat{\theta}) \leftarrow \frac{1}{N} \sum_{x \in S} \mathbf{P}(\theta = \hat{\theta}|X = x).$$

El mejor valor de los parámetros θ^* dado un ejemplo se obtiene utilizando el principio de máxima verosimilitud:

$$\theta^* = \arg\max_\theta \prod_{i=1}^{N} \mathbf{P}(\theta = \hat{\theta}|X = x_i).$$

Si el conjunto de valores posibles para θ no es finito, entonces necesitamos optimizar directamente usando rutinas de optimización numérica como el gradiente de descenso.

Algoritmos fundamentales supervisados

- Regresión lineal
- Regresión logística
- Árboles de decisión
- SVC (no presentado)
- Modelos ensamblados

Regresión lineal

Problema

Tenemos una colección de ejemplos etiquetados $\{(\mathbf{x}_i, y_i)\}, i = 1 \dots N$, donde N es el tamaño de la colección, x_i es un vector de características D-dimensional del ejemplo $i = 1, \dots, N$, y_i es un objetivo de valor real y cada característica $x_i^{(j)}, j = 1, \dots, D$ también es un número real.

Queremos construir un modelo $f_{\mathbf{w},b}(x) = \mathbf{w}\mathbf{x} + b$ como una combinación lineal de características del ejemplo $\mathbf{x}$:

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}\mathbf{x} + b.$$

donde $\mathbf{w}$ es un vector de parámetros D-dimensional y b es un número real. La notación $f_{\mathbf{w},b}$ significa que el modelo f está parametrizado por dos valores: $\mathbf{w}$ y b.

Usaremos el modelo para predecir un nuevo valor y para un $\mathbf{x}$ dado como:

$$y \leftarrow f_{\mathbf{w},b}(\mathbf{x}).$$

Queremos encontrar los valores óptimos $(\mathbf{w}^*, b^*)$.

Regresión lineal

Solución

El procedimiento de optimización que usamos para encontrar los valores óptimos de $\mathbf{w}^*$ y b intenta minimizar la siguiente expresión:

$$\frac{1}{N} \sum_{i=1 \ldots N} (f_{\mathbf{w},b}(\mathbf{x}_i) - y_i)^2.$$

La expresión $(f_{\mathbf{w},b}(\mathbf{x}_i) - y_i)^2$ en el objetivo (función de costo) anterior se llama pérdida de error al cuadrado (MSE). En la regresión lineal, la función de costo viene dada por el riesgo empírico.

La pérdida promedio o riesgo empírico para un modelo, es el promedio de todas las penalizaciones obtenidas al aplicar el modelo a los datos de entrenamiento.

Las soluciones para encontrar el óptimo de una función son expresiones algebraicas y a menudo son preferibles a los métodos complejos de optimización numérica, como el gradiente de descenso.

Regresión logística

```
La regresión logística no es una regresión, sino un algoritmo
de aprendizaje de clasificación.
```

Problema

Si definimos una etiqueta negativa como 0 y una etiqueta positiva como 1, tenemos un codominio como $(0, 1)$. En tal caso, si el valor devuelto por el modelo para la entrada x está más cerca de 0, entonces le asignamos una etiqueta negativa a x, de lo contrario, el ejemplo se etiqueta como positivo.

La función logística estándar cumple las condiciones para este escenario $f(x) = \frac{1}{1+e^{-x}}$ y nuestro modelo de regresión logística se ve así:

$$f_{\mathbf{w},b}(\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}\mathbf{x}+b)}}.$$

Puedes ver el término familiar $\mathbf{w}\mathbf{x} + b$ de la regresión lineal. Ahora bien, ¿cómo encontramos los mejores valores $\mathbf{w}^*$ y b^* para nuestro modelo?

Regresión logística

Solución

En la regresión logística, maximizamos la probabilidad de que nuestro conjunto de entrenamiento se ajuste al modelo. El criterio de optimización en la regresión logística se denomina verosimilitud máxima:

$$L_{\mathbf{w},b} = \prod_{i=1\ldots N} f_{\mathbf{w},b}(\mathbf{x}_i)^{y_i}(1-\mathbf{x}_i)^{1-y_i}.$$

Es más conveniente en la práctica, maximizar la verosimilitud logarítmica que se define de la siguiente manera:

$$\log L_{\mathbf{w},b} = \ln(L_{\mathbf{w},b}(\mathbf{x})) = \sum_{i=1}^{N} y_i \ln f_{\mathbf{w},b}(\mathbf{x}) + (1-y_i)\ln(1-f_{\mathbf{w},b}(\mathbf{x})).$$

Al contrario de la regresión lineal, no hay una solución completa para el problema de optimización anterior. Un procedimiento típico de optimización numérica utilizado en tales casos es el descenso de gradiente.

Árboles de decisión

Es un grafo acíclico que se puede usar para tomar decisiones. En cada nodo de ramificación del grafo, se examina una característica específica j del vector de características.

Se puede aprender un árbol de decisiones a partir de los datos.

Problema

Tenemos una colección de ejemplos etiquetados. Las etiquetas pertenecen al conjunto $\{0, 1\}$.

Queremos construir un árbol de decisiones que nos permita predecir la clase de un ejemplo dado un vector de características.

Solución

Consideramos el algoritmo ID3. El criterio de optimización, en este caso, es la verosimilitud logarítmica promedio:

$$\frac{1}{N} \sum_{i=1}^{N} y_i \ln f_{ID3}(\mathbf{x}_i) + (1 - y_i) \ln(1 - f_{ID3}(\mathbf{x}_i)).$$

Árboles de decisión

Solución

Contrariamente al algoritmo de aprendizaje de regresión logística que construye un modelo paramétrico $f_{\mathbf{w}^*,b^*}$, al encontrar una solución óptima al criterio de optimización, el algoritmo ID3 optimiza aproximadamente al construir un modelo no paramétrico:

$$f_{ID3}(\mathbf{x}) = \mathbf{P}(y = 1|\mathbf{x}).$$

donde f_{ID3} es un árbol de decisión.

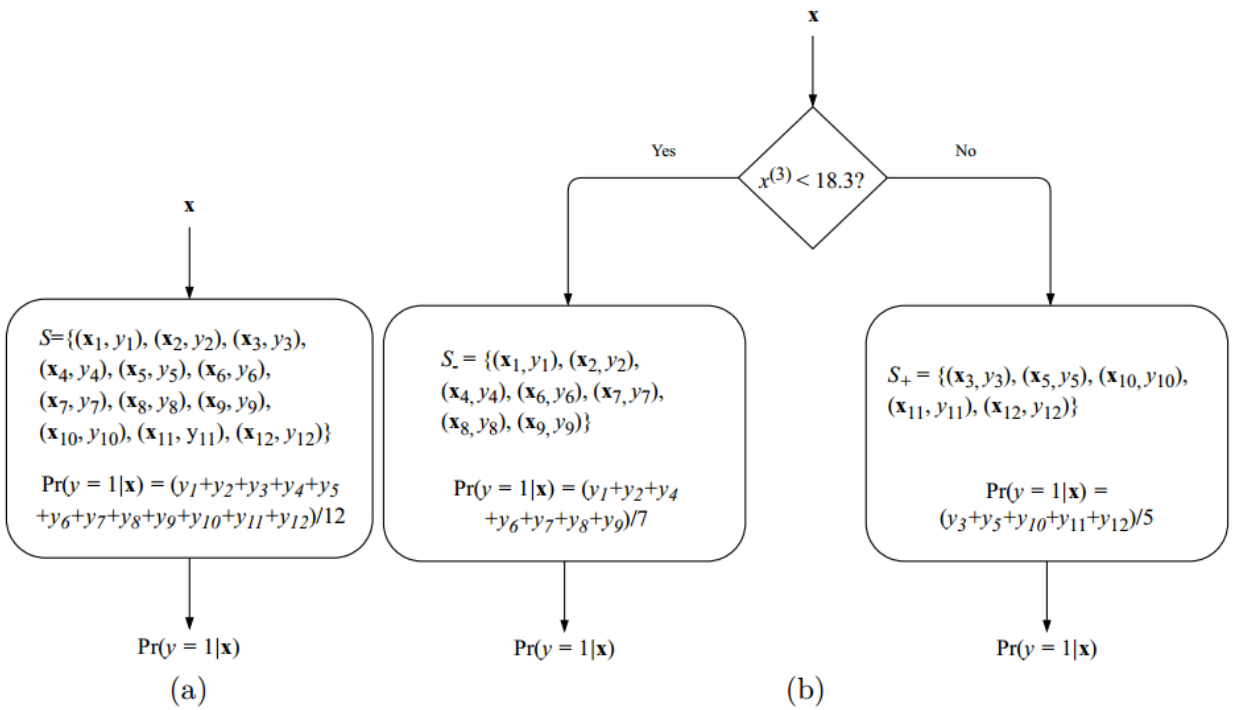
En el algoritmo ID3, la decisión de dividir el conjunto de datos en cada iteración es local (no depende de futuras divisiones), el algoritmo no garantiza una solución óptima.

El modelo se puede mejorar mediante el uso de técnicas como el backtracking durante la búsqueda del árbol de decisiones óptimo al costo de posiblemente tomar más tiempo para construir un modelo.

Algoritmo ID3

Sea S un conjunto de ejemplos etiquetados. Al principio, el árbol de decisión solo tiene un nodo de inicio que contiene todos los ejemplos:
$S = \{(\mathbf{x}_i, y_i)\}\ i = 1 \ldots N.$

Un árbol de decisión después de una divisiones se ilustra en la siguiente figura.



Algoritmo ID3

Comenzamos con un modelo constante f^S_{ID3} dado por:

$$f^S_{ID3} = \frac{1}{N} \sum_{(\mathbf{x},y)\in S} y.$$

La predicción dada por el modelo anterior f^S_{ID3} sería la misma para cualquier entrada $\mathbf{x}$.

Luego buscamos en todas las características $j = 1, \ldots, D$ y todos los umbrales t y dividimos el conjunto S en dos subconjuntos:
$S_- = \{(\mathbf{x}, y) | (\mathbf{x}, y) \in S, x^{(t)} < t\}$ y $S_+ = \{(\mathbf{x}, y) | (\mathbf{x}, y) \in S, x^{(t)} \geq t\}$.

Los dos nuevos subconjuntos irían a dos nuevos nodos hoja y evaluamos para todos los pares posibles (j, t), qué tan buena es la división con las piezas S_- y S_+.

Elegimos los mejores valores (j, t), dividimos S en S_- y S_+, formamos dos nuevos nodos de hoja y continuamos recursivamente en S_- y S_+. (o salimos si no hay una división que produce un modelo que es mejor que el actual).

Algoritmo ID3

En ID3, la bondad de una división se estima utilizando el criterio llamado entropía.

La entropía es una medida de incertidumbre sobre una variable aleatoria y en un conjunto de ejemplos S viene dada por:

$$H(S) = -f_{ID3}^{S} \ln f_{ID3}^{S} - (1 - f_{ID3}^{S}) \ln(1 - f_{ID3}^{S}).$$

Cuando dividimos un conjunto de ejemplos por una determinada característica j y un umbral t, la entropía de una división, $H(S\_S_{+})$, es simplemente una suma ponderada de dos entropías:

$$H(S\_S_{+}) = \frac{|S\_|}{|S|} H(S\_) + \frac{|S_{+}|}{|S|} H(S_{+}).$$

En ID3, en cada paso, en cada nodo de hoja, encontramos una división que minimiza la entropía o nos detenemos en este nodo hoja.

Algoritmo ID3

El algoritmo se detiene en un nodo hoja en cualquiera de las siguientes situaciones:

- Todos los ejemplos en el nodo de hoja se clasifican correctamente por el modelo.
- No podemos encontrar un atributo para dividir.
- La división reduce la entropía menos que algún ϵ (el valor que se debe encontrar experimentalmente).
- El árbol alcanza una profundidad máxima d (también debe encontrarse experimentalmente).

El criterio de división basada en la entropía tiene sentido intuitivamente: la entropía alcanza su mínimo de 0 cuando todos los ejemplos en S tienen la misma etiqueta.

La entropía está en su máximo 1 cuando exactamente la mitad de los ejemplos en S están marcados con 1, lo que hace que esa hoja sea poco útil para la clasificación.

Anatomía de un algoritmo de aprendizaje automático

Cada algoritmo de aprendizaje que vimos constaba de tres partes:

- Una función de pérdida
- Un criterio de optimización basado en la función de pérdida (una función de costo)
- Una rutina de optimización que aprovecha los datos de entrenamiento para encontrar una solución al criterio de optimización.

El descenso de gradiente es un algoritmo de optimización iterativo para encontrar el mínimo de una función y se puede usar para encontrar parámetros óptimos para la regresión lineal y logística, SVM y también redes neuronales.

Para muchos modelos, como la regresión logística o SVM, el criterio de optimización es la convexidad. Las funciones convexas solo tienen un mínimo, que es global.

Descenso de gradiente

Si queremos encontrar un óptimo local $f(\mathbf{x}^*)$ de una función $f : \mathbf{R}^n \to \mathbf{R}$ comenzamos con una estimación inicial x_0 de los parámetros que deseamos optimizar y luego iteramos de acuerdo con:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \lambda_i((\nabla f)(\mathbf{x}_i))^T.$$

Para un tamaño de paso adecuado λ_i, la secuencia $f(\mathbf{x}_0) \geq f(\mathbf{x}_1) \geq \ldots$ converge a un mínimo local.

Descenso de gradiente con momentum

Es un método que introduce un término adicional para recordar lo que sucedió en la iteración anterior. Esta memoria amortigua las oscilaciones y suaviza las actualizaciones de gradiente.

El método basado en momentum recuerda la actualización $\Delta\mathbf{x}_i$ en cada iteración i y determina la siguiente actualización como una combinación lineal del gradientes actual y los anteriores:

$$\begin{aligned}\mathbf{x}_{i+1} &= \mathbf{x}_i - \lambda_i((\nabla f)(\mathbf{x}_i))^T + \alpha\Delta\mathbf{x}_i \\ \Delta\mathbf{x}_i &= \mathbf{x}_i - \mathbf{x}_{i-1} = -\lambda_{i-1}((\nabla f)(\mathbf{x}_{i-1}))^T.\end{aligned}$$

Descenso de gradiente estocástico (SGD)

Es una versión del algoritmo que acelera el cálculo aproximando el gradiente utilizando batches de los datos de entrenamiento.

En el aprendizaje automático dado $n = 1, \ldots, N$ puntos de datos, a menudo consideramos funciones objetivos que son la suma de funciones de pérdidas L_n incurridas por cada ejemplo n. En notación matemática tenemos la forma:

$$L(\theta) = \sum L_n(\theta).$$

donde θ es el vector de parámetros. Es decir queremos encontrar θ que minimize L.

Un ejemplo en regresión es la verosimilitud logarítmica negativa, que se expresa como:

$$L(\theta) = -\sum_{n=1}^{N} \log p(y_n|\mathbf{x}_n, \theta).$$

donde $\mathbf{x}_n \in R^D$ son las entradas de entrenamiento, y_n son los objetivos de entrenamiento y θ son los parámetros del modelo de regresión.

Más de SGD -1

El descenso de gradiente estándar, como se introdujo anteriormente, es un método de optimización "por lotes", es decir, la optimización se realiza utilizando el conjunto de entrenamiento completo actualizando el vector de parámetros:

$$\theta_{i+1} = \theta_i - \lambda_i (\nabla L(\theta_i))^T = \theta_i - \lambda_i \sum_{n=1}^{N} (\nabla L_n(\theta_i))^T.$$

para un parámetro de tamaño de paso adecuado λ_i.

La evaluación del gradiente de la suma puede requerir costosas evaluaciones de los gradientes de todas las funciones individuales L_n. Podemos reducir la cantidad de cálculo tomando una suma sobre un conjunto más pequeño de L_n. Elegimos al azar un subconjunto de L_n para el descenso de gradiente de minibatches.

La idea clave es darse cuenta de que para que el descenso del gradiente converja, solo necesitamos que el gradiente sea una estimación no sesgada del gradiente real.

Más de SGD -2

Dado que el objetivo en el aprendizaje automático no necesita necesariamente una estimación precisa del mínimo de la función objetivo, se utiliza gradientes aproximados que utilizan enfoques de minibatches. El descenso de gradiente estocástico es muy eficaz en problemas de aprendizaje automático a gran escala

SGD tiene varias actualizaciones. Adagrad es una versión de SGD que se escala, para cada parámetro de acuerdo con el historial de gradientes. En el entrenamiento de redes neuronales, las variantes de SGD, como RMSprop y Adam, son las más utilizadas.

Selección de un algoritmo de aprendizaje

Elegir un algoritmo de aprendizaje automático puede ser una tarea difícil. Puedes hacerte varias preguntas antes de comenzar a trabajar sobre el problema. Dependiendo de tus respuestas, puedes hacer una lista de algunos algoritmos y probarlos con tus datos.

- **Explicabilidad**: ¿tu modelo tiene que ser explicable a una audiencia no técnica?.
- **En memoria vs. fuera de memoria**: ¿se puede cargar completamente tu conjunto de datos en la memoria RAM de tu servidor o computadora personal?.
- **Número de características y ejemplos**: ¿cuántos ejemplos de entrenamiento tienes en tu conjunto de datos?.
- **Características categóricas vs. numéricas**: ¿tus datos se componen de características categóricas o solo numéricas, o una combinación de ambas?.
- **No linealidad de los datos**: ¿tus datos se pueden separar linealmente o se pueden modelar utilizando un modelo lineal?.
- **Velocidad de entrenamiento** : ¿cuánto tiempo puedes usar un algoritmo de aprendizaje para construir un modelo?.
- **Velocidad de predicción**: ¿qué tan rápido debe ser el modelo al generar predicciones?, ¿se usará tu modelo en la producción donde se requiere un rendimiento muy alto?.

Tres conjuntos

Los analistas de datos trabajan con tres conjuntos de ejemplos etiquetados:

- Conjunto de entrenamiento,
- Conjunto de validación y
- Conjunto de prueba.

Los conjuntos de validación y prueba a menudo se denominan hold-out sets.

A partir de estos conjuntos queremos que nuestro modelo prediga ejemplos que el algoritmo de aprendizaje no ha visto.

Para ellos usamos el conjunto de validación para 1) elegir el algoritmo de aprendizaje y 2) encontrar los mejores valores de los hiperparámetros y utilizamos el conjunto de pruebas para evaluar el modelo antes de entregarlo al cliente o ponerlo en producción.

Subajuste y sobreajuste

Un modelo tiene un bajo sesgo si predice bien las etiquetas de los datos de entrenamiento. Si el modelo comete muchos errores en los datos de entrenamiento, decimos que el modelo tiene un alto sesgo o que el modelo está subajustado.

Por lo tanto, el **subajuste** es la incapacidad del modelo para predecir bien las etiquetas de los datos en los que se entrenó. Podría haber varias razones para el subajuste, las más importantes de las cuales son:

- Tu modelo es demasiado simple para los datos
- Las características que diseñastes no son lo suficientemente informativas.

  La solución al problema del subajuste es probar un modelo más complejo o diseñar características con mayor poder predictivo.

Subajuste y sobreajuste

El **sobreajuste** es otro problema que un modelo puede exhibir. El modelo que sobreajusta predice muy bien los datos de entrenamiento, pero no los datos de al menos uno de los dos conjunto hould-set. Varias razones pueden llevar al sobreajuste, las más importantes de las cuales son:

- Tu modelo es demasiado complejo para los datos
- Tienes demasiadas características, pero pocos ejemplos de entrenamiento.

La **varianza** es un error del modelo debido a su sensibilidad a pequeñas fluctuaciones en el conjunto de entrenamiento. Si tus datos de entrenamiento se muestrean de manera diferente, el aprendizaje resultaría en un modelo significativamente diferente, por lo que todo modelo que se sobreajusta lleva de forma deficiente el manejo de los datos de prueba.

Al problema de sobreajuste se le denomina el problema de alta varianza. Varias soluciones al problema del sobreajuste son posibles:

- Probar un modelo más simple
- Reducir la dimensionalidad de los ejemplos en el conjunto de datos
- Agregar más datos de entrenamiento si es posible.
- Regularizar el modelo.

Regularización

La regularización es un término general que abarca métodos que obligan a un algoritmo de aprendizaje a construir un modelo menos complejo. En la práctica, eso a menudo conduce a un sesgo ligeramente más alto pero reduce significativamente la varianza.

Este problema se conoce en la literatura como bias-varianza tradeoff.

Los dos tipos de regularización más utilizados se denominan regularización L1 y regularización L2.

`Idea: para crear un modelo regularizado, modificamos la función objetivo agregando un término penalizador cuyo valor es más alto cuando el modelo es más complejo.`

Ejemplos de regularización

Recuerda el objetivo de regresión lineal que queremos minimizar:

$$\min_{\mathbf{w},b} \frac{1}{N} \sum_{i=1}^{N} (f_{\mathbf{w},b}(\mathbf{x}_i) - y_i)^2.$$

Un objetivo regularizado L1 se ve así:

$$\min_{\mathbf{w},b} C|\mathbf{w}| + \frac{1}{N} \sum_{i=1}^{N} (f_{\mathbf{w},b}(\mathbf{x}_i) - y_i)^2.$$

donde $|\mathbf{w}| = \sum_{j=1}^{D} |w^{(j)}|$ y C es un hiperparámetro que controla la importancia de la regularización.

Tu función como analista de datos es encontrar un valor del hiperparámetro C que no aumente demasiado el sesgo, pero reduzca el sobreajuste a un nivel razonable.

Ejemplos de regularización

Un objetivo regularizado L2 se ve así:

$$\min_{\mathbf{w},b} C\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{i=1}^{N}(f_{\mathbf{w},b}(\mathbf{x}_i) - y_i)^2.$$

donde $\|\mathbf{w}\| = \sum_{j=1}^{D}(w^{(j)})^2$ y C es un hiperparámetro que controla la importancia de la regularización.

Comentarios

- L1 produce un modelo disperso y realiza selección de características al decidir qué características son esenciales para la predicción y cuáles no. Eso puede ser útil en caso de que desee aumentar la explicabilidad del modelo.
- Si tu objetivo es maximizar el rendimiento del modelo, L2 suele ofrecer mejores resultados, además que se puede usar el descenso de gradiente para optimizar la función objetivo.

Las redes neuronales también se benefician de otras dos técnicas de regularización: dropout y normalización por lotes.

Evaluación de modelos

Los especialistas en aprendizaje automático utilizan diversas métricas y herramientas formales para evaluar el rendimiento de un modelo.

Regresión

- Se utiliza el error cuadrático medio (MSE) o otras variantes como RMSE, MAE , para el entrenamiento y por separado, para los datos de prueba.

Si el MSE del modelo en los datos de prueba es sustancialmente más alto que el MSE obtenido en los datos de entrenamiento, esto es un signo de sobreajuste.

Clasificación

- Las métricas y herramientas más utilizadas para evaluar un modelo de clasificación son:
    - Matriz de confusión
    - Exactitud
    - Precisión/exhaustividad
    - Área bajo la curva ROC.

Ajuste de hiperparámetros

`Recordar: Los hiperparámetros no están optimizados por el propio algoritmo de aprendizaje.`

Posible solución : Búsqueda grid

- Datos suficientes y el rango de hiperparametros no es muy grande.
- Se basa en probar las combinaciones de hiperparámetros lo que podría llevar mucho tiempo, especialmente para grandes conjuntos de datos.

- Hay más técnicas de ajustes como la búsqueda aleatoria y la optimización de hiperpámetros bayesiana.

Otras soluciones:

- Búsqueda aleatoria proporciona una distribución estadística para cada hiperparámetro desde el cual se muestrean valores aleatoriamente y establece el número total de combinaciones.

Ajuste de hiperparámetros

- Técnicas bayesianas usan los resultados de evaluaciones anteriores para elegir los siguientes valores a evaluar. La idea es limitar la optimización de la función objetivo al elegir los siguientes valores de hiperparámetro basados en aquellos que han funcionado bien en el pasado.
- También existen técnicas basadas en gradientes, técnicas de optimización evolutiva.

Validación cruzada

Cuando no tienes un conjunto de validación decente para ajustar tus hiperparámetros, la técnica común que puede ayudarlo se llama **validación cruzada**.

Cuando tienes pocos ejemplos de entrenamiento, podría ser prohibitivo tener tanto conjuntos de validación y de pruebas. Es preferible usar más datos para entrenar el modelo. En tal caso, solo divide tus datos en un conjunto de entrenamiento y de pruebas. Luego, utiliza la validación cruzada en el conjunto de entrenamiento para simular un conjunto de validación

Ver cuadernos de laboratorio de clase

Sugerencia

Puedes usar una búsqueda grid con validación cruzada para encontrar los mejores valores de hiperparámetros para tu modelo. Una vez que haya encontrado estos valores, usa todo el conjunto de entrenamiento para construir el modelo con estos mejores valores de hiperparámetros que hayas encontrado mediante la validación cruzada.

Finalmente, evalúa el modelo usando el conjunto de prueba.

Modelos ensamblados

Se centran en entrenar una gran cantidad de modelos de baja exactitud o precisión y luego combinar las predicciones dadas por esos modelos débiles para obtener un mejor modelo de alta exactitud o precisión.

Los modelos de baja precisión generalmente son aprendidos por aprendices débiles (árboles de decisión), es decir, algoritmos de aprendizaje que no pueden aprender modelos complejos y por lo tanto, suelen ser rápidos en el entrenamiento y en el tiempo de predicción.

Para obtener la predicción para la entrada $\mathbf{x}$, las predicciones de cada modelo débil se combinan utilizando algún tipo de votación ponderada. La forma específica de ponderación de votos depende del algoritmo, pero, independientemente del algoritmo, la idea es la misma: si el grupo de modelos débiles predice que el mensaje es spam, asignamos la etiqueta spam a $\mathbf{x}$.

Los dos algoritmos de aprendizaje ensamblados más utilizados y eficaces son:

- Bosque aleatorio
- Gradient boosting.

Bosque aleatorio

Hay dos paradigmas de aprendizaje ensamblados: bagging y boosting. El bagging consiste en crear muchas *copias* de los datos de entrenamiento (cada copia es ligeramente diferente de la otra) y luego aplicar un aprendiz débil a cada copia para obtener múltiples modelos débiles y luego combinarlos.

El paradigma bagging está detrás del algoritmo de aprendizaje de bosque aleatorio.

Un algoritmo bagging funciona de la siguiente forma:

- Dado un conjunto de entrenamiento, creamos B muestras aleatorias S_b (para cada $b = 1, \ldots, B$) del conjunto de entrenamiento y construimos un modelo de árbol de decisión f_b usando cada muestra S_b como el conjunto de entrenamiento.
- Para muestrear S_b para algún b, comenzamos con un conjunto vacío y luego elegimos al azar un ejemplo del conjunto de entrenamiento y colocamos su copia exacta en S_b manteniendo el ejemplo original en el conjunto de entrenamiento original (muestreo con reemplazo).
- Seguimos seleccionando ejemplos al azar hasta que $|S_b| = N$.

Después del entrenamiento, tenemos B árboles de decisión.

Bosque aleatorio

La predicción para un nuevo ejemplo x se obtiene como el promedio de B predicciones:

$$y \leftarrow \hat{f}\,(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} f_b(\mathbf{x}).$$

en el caso de regresión o por mayoría de votos en el caso de la clasificación.

El algoritmo de bosque aleatorio es diferente del bagging en una sola forma. Utiliza un algoritmo de aprendizaje de árbol modificado que inspecciona, en cada división del proceso de aprendizaje, un subconjunto aleatorio de características. La razón para hacer esto es evitar la correlación de los árboles.

Los hiperparámetros más importantes para ajustar son el número de árboles, B y el tamaño del subconjunto aleatorio de las características a considerar en cada división.

Gradient Boosting

Para la regresión

- Para construir un regresor fuerte, comenzamos con un modelo constante $f = f_0$ (como hicimos en ID3): $f = f_0(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} y_i$.
- Modificamos las etiquetas de cada ejemplo $i = 1, \ldots, N$ en nuestro conjunto de entrenamiento de la siguiente manera: $\hat{y}_i \leftarrow y_i - f(\mathbf{x}_i)$, donde $\hat{y}_i$ llamado residual es la nueva etiqueta, por ejemplo $\mathbf{x}_i$.
- Ahora usamos el conjunto de entrenamiento modificado, con residuos en lugar de etiquetas originales, para construir un nuevo modelo de árbol de decisión f_1. El modelo boosting ahora se define como $f = f_0 + \alpha f_1$, donde α es la tasa de aprendizaje (un hiperparámetro).
- Luego volvemos a calcular los residuos usando la ecuación anterior y reemplazamos las etiquetas en los datos de entrenamiento una vez más, entrenamos el nuevo modelo de árbol de decisión f_2, redefinimos el modelo boosting como $f = f0 + \alpha f1 + \alpha f2$ y el proceso continúa hasta que el máximo de M (otro hiperparámetro) árboles sea ensamblado.

Gradient Boosting

En el algoritmo, cada árbol adicional agregado al modelo corrige parcialmente los errores cometidos por los árboles anteriores hasta que se combinan el número máximo de árboles.

Los tres hiperparámetros principales para ajustar el gradient boosting son el número de árboles, la tasa de aprendizaje y la profundidad de los árboles. Los tres afectan la precisión del modelo.

La profundidad de los árboles también afecta la velocidad del entrenamiento y la predicción: cuanto más pequeños, más rápidos.

Gradient Boosting

Para la clasificación (caso binario)

- Suponga que tenemos M árboles de decisión de regresión. De manera similar a la regresión logística, la predicción del conjunto de árboles de decisión se modela utilizando la función sigmoidea: $\mathbf{P}(y = 1|\mathbf{x}, f) = \frac{1}{1+e^{-f(\mathbf{x})}}$ donde $f(\mathbf{x}) = \sum_{m=1}^{M} f_m(\mathbf{x})$ y f_m es un árbol de decisión.
- Aplicamos el principio de máxima verosimilitud para encontrar un f que maximiza $L_f = \sum_{i=1}^{N} \ln(\mathbf{P}(y_i = 1|\mathbf{x}_i, f)$.
- El algoritmo comienza con el modelo constante inicial $f = f_0 = \frac{p}{1-p}$, donde $p = \frac{1}{N}\sum_{i=1}^{N} y_i$. Luego, en cada iteración m, se agrega un nuevo árbol f_m al modelo. Para encontrar el mejor f_m primero se calcula la derivada parcial g_i del modelo actual para cada $i = 1, \ldots, N$:

$$g_i = \frac{dL_f}{df},$$

donde f es el modelo de clasificador de conjunto ensamblado en la iteración anterior $m - 1$.

Gradient Boosting

Para la clasificación (caso binario)

- Para calcular g_i necesitamos encontrar las derivadas de $\ln(\mathbf{P}(y_i = 1|\mathbf{x}_i, f)$ con respecto a f para todo i. La derivada con respecto a f es igual a $\frac{1}{e^{f(\mathbf{x}_i)}+1}$.
- Transformamos nuestro conjunto de entrenamiento reemplazando la etiqueta original y_i con la derivada parcial correspondiente g_i y construimos un nuevo árbol f_m usando el conjunto de entrenamiento transformado.
- Luego encontramos el paso de actualización óptimo ρ_m como:

  $$\rho_m = \arg\max_\rho L_{f+\rho f_m}.$$

  Al final de la iteración m, actualizamos el modelo ensamblado f agregando el nuevo árbol f_m:

  $$f \leftarrow f + \alpha\rho_m f_m.$$

- Iteramos hasta $m = M$ luego nos detenemos y devolvemos el modelo de ensamblado f.

Comentarios

- El bosque aleatorio es uno de los algoritmos de aprendizaje por conjuntos más utilizados y la razón es que al usar múltiples muestras del conjunto de datos original, reduce la varianza del modelo final. Una varianza baja significa un bajo sobreajuste.
- El gradient boosting usa residuales que muestran cómo se debe ajustar el modelo para que se reduzca el error (el residual).

- Se puede demostrar que el entrenamiento en residuos optimiza el modelo en regresión general f para el criterio de error cuadrático medio.

- El gradient boosting es uno de los algoritmos de aprendizaje automático más poderosos. No solo porque crea modelos muy precisos, sino también porque es capaz de manejar grandes conjuntos de datos con millones de ejemplos y características. Por lo general, supera al bosque aleatorio en precisión pero, debido a su naturaleza secuencial, puede ser significativamente más lento en el entrenamiento.

Transfer learning

- Elección de un modelo existente entrenado en algún conjunto de datos y lo adaptas para predecir ejemplos de otro conjunto de datos, diferente del que se creó en el modelo.
- El transfer learning en redes neuronales funciona así:
    - Construye un modelo profundo sobre el gran conjunto de datos original.
    - Compila un conjunto de datos etiquetado mucho más pequeño para el segundo modelo.
    - Elimina la última o varias capas del primer modelo. Por lo general, son capas responsables de la clasificación o regresión.
    - Reemplaza las capas eliminadas por capas nuevas adaptadas al nuevo problema.
    - "Congela" los parámetros de las capas restantes del primer modelo.
    - Utiliza tu dataset etiquetado y el descenso de gradiente para entrenar los parámetros solo de las nuevas capas.

Fin!