75.06 ORGANIZACIÓN DE DATOS
CLASIFICACIÓN BINARIA DE TEXTOS UTILIZANDO TÉCNICAS DE COMPRESIÓN

NOMBRE DEL GRUPO EN KAGGLE: "LOS BORBOTONES"

Resumen. En este trabajo estudiaremos la aproximación a la clasificación binaria de textos utilizando técnicas de compresión. Se propone también un posible diseño de un clasificador AMDL, y se dejan formuladas las herramientas matemáticas necesarias para implementar un Perceptrón con kernel que utiliza técnicas de compresión para obtener una medida de similitud entre archivos clasificables.

Índice

1. Fundamentos y Antecedentes

La idea del equipo de utilizar compresión para la clasificación de textos surge tras la aplicación de la misma a la búsqueda de imágenes similares en un conjunto de aproximadamente 50000 archivos PNG. Una búsqueda posterior llevó a un paper de Marton, Wu y Hellerstein [1]. En dicho artículo se describen los resultados de un experimento en el que se utilizan diferentes herramientas de compresión para clasificar, con buenos resultados, textos extraidos de variados conjuntos de datos. Las técnicas de clasificación descriptas en el artículo de Marton et al. son tres: SMDL (standard minimum description length), AMDL (approximate minimum description length), y BCN (best compression neighbor). Estos métodos se describen en más detalle en la sección 2.

La idea intuitiva detrás del uso de la compresión para clasificar textos es la siguiente: la longitud en bits del string que se obtiene al comprimir dos documentos concatenados será menor si éstos presentan similitudes al nivel de la información que contienen. Esto implica que la tasa de compresión podría utilizarse como una medida de similitud. Existen clasificadores como el Perceptrón con Kernel y las Máquinas de Soporte Vectorial que utilizan medidas de similitud para los procesos de entrenamiento y clasificación. En [2] se describen los requisitos que deben cumplir las denominadas funciones núcleo o kernel. Dichas funciones calculan productos internos en forma implícita y son utilizadas por los ya mencionados clasificadores para evaluar el parecido entre elementos clasificables. Una idea que surge es entones la siguiente: dados dos textos, quizás sea posible utilizar la medida denominada *distancia normalizada de compresión* para cuantificar la similitud entre los mismos. Más detalles sobre esta distancia se dan en la sección 2. En dicha sección se presentan también los fundamentos matemáticos necesarios para implementar un Perceptrón con Kernel.

Las técnicas y herramientas estudiadas en este documento se aplicarán a la clasificación de un conjunto de 25000 reseñas de películas extraídas de IMDB (Internet Movie Database). Se cuenta, en adición, con un conjunto de 25000 reseñas de entrenamiento, de las cuáles 12500 se consideran positivas, y 12500 se consideran negativas.

2. Bases Matemáticas

2.1. Minimum Description Length

Los tres métodos de clasificación por compresión mencionados en [1] se describen a continuación.

2.1.1. SMDL (Standard Minimum Description Length)

La idea del método es la siguiente: dado un conjunto de documentos de entrenamiento $S_1, S_2, \ldots, S_m$, categorizados en clases $c_1, c_2, \ldots c_m$, SMDL crea un archivo A_i para cada clase c_i, $i = 1, 2, \ldots, m$. Este archivo A_i contiene la concatenación de todos los documentos de entrenamiento que pertenezcan a la clase c_i. Utilizando un algoritmo de compresión, el clasificador crea un modelo o diccionario M_i a partir de cada archivo A_i. Dado entonces un documento de prueba T, SMDL comprime T una vez con cada modelo M_i y asigna T a la clase cuyo modelo haya obtenido la mejor tasa de compresión.

2.1.2. AMDL (Approximate Minimum Description Length)

AMDL es una aproximación a SMDL. AMDL genera en base a un conjunto de documentos de entrenamiento clasificados una secuencia de archivos A_i tal como lo hace SMDL. En vez de utilizar un algoritmo de compresión para generar un modelo, AMDL comprime y toma la longitud en bits $|A_{ic}|$ cada archivo A_i comprimido, A_{ic}. Dado un archivo de prueba T, el algoritmo concatena T con cada uno de los archivos comprimidos, obteniendo la secuencia $A_{ic}T$. Para medir la similitud entre T y cada uno de los archivos A_i, AMDL toma la longitud en bits $|(A_{ic}T)_c| - |A_{ic}|$, donde $(A_{ic}T)_c$ es la concatenación $A_{ic}T$ comprimida. T será asignado a la clase cuyo archivo minimice esta diferencia.

2.1.3. BCN (Best Compression Neighbor)

BCN es similar a AMDL, salvo que cada documento de entrenamiento S_i se mantiene en un archivo separado. En vez de utilizar archivos representativos de cada clase, BCN concatena el archivo de prueba T a cada archivo S_i, y toma la diferencia $|(S_{ic}T)_c| - |S_{ic}|$. A T se le asigna la clase del documento S_i que minimice esta diferencia.

2.2. Métodos con Kernel

Existen muchos métodos de clasificación que utilizan funciones núcleo (o kernels) para medir la similitud entre dos piezas de información. Estas piezas de información pueden ser vectores, strings, matrices, grafos, etc. La ventaja de utilizar kernels es que no es necesario obtener una representación vectorial explícita de la información que se desea clasificar.

Procederemos ahora a dar una definición más formal de una función kernel k. La definición dada a continuación es similar a la propuesta en [2, p.3] por Hoffman et al.

Definición 1. Sea $\mathcal{S}$ la unión entre el conjunto de los objetos de entrenamiento y el conjunto de los objetos a ser clasificados. Sean s_1, s_2 elementos de $\mathcal{S}$. Una función kernel $k : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ es una función semidefinida positiva tal que

$$k(s_1, s_2) = \langle \Phi(s_1), \Phi(s_2) \rangle ,$$

donde Φ es una transformación de $\mathcal{S}$ hacia algún espacio con producto interno $\mathcal{V}$, y $\langle ., . \rangle$ es un producto interno en $\mathcal{V}$.

Es decir, k calcula un producto interno transformando implícitamente elementos de $\mathcal{S}$ en elementos de $\mathcal{V}$. No es necesario entonces calcular (o conocer) la transformación Φ para efectuar la operación.

2.3. El Perceptrón

Perceptrón es el nombre de un algoritmo de clasificación lineal supervisada que data del año 1950. En base a un conjunto de vectores de entrenamiento, pertenecientes al espacio de características $\mathcal{V}$, el Perceptrón busca determinar un hiperplano en $\mathcal{V}$ tal que aquellos elementos que se encuentren de un lado del hiperplano sean de la clase 1, y aquellos que se encuentren del otro lado del mismo sean de la clase -1.

Más formalmente, definimos la siguiente regla de clasificación: sea $\mathbf{x}$ un vector del espacio de características $\mathcal{V}$, y sea $y \in \{-1, 1\}$ la clase binaria asociada a $\mathbf{x}$. El algoritmo del Perceptrón busca determinar un vector de pesos $\mathbf{w} \in \mathcal{V}$ tal que, para todo $\mathbf{x}$,

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b \geq 0 \Leftrightarrow y = 1 \tag{1}$$

La idea intuitiva detrás del método es la siguiente: reemplazando el producto escalar en (1) por una suma, tal que

$$f(\mathbf{x}) = \sum_{i=1}^{n} w_i \, x_i + b \geq 0 \Leftrightarrow y = 1 \, ,$$

podemos ver que la idea del perceptrón es asignar a cada característica x_i un cierto peso. Algunas características tendrán pesos negativos, otras tendrán pesos positivos. Es más probable que aquellos elementos que presenten características positivas de mayor valor tengan asociada la clase positiva 1, y aquellos elementos que presenten características negativas de mayor valor absoluto tengan asociada la clase -1.

Se puede lograr que el Perceptrón aprenda el sesgo b y eliminar el término de (1) agregando una componente adicional al vector $\mathbf{w}$, cuyo valor es b. A cada uno de los vectores del espacio de características se les agrega una componente adicional cuyo valor es idénticamente 1,

para todos y cada uno de ellos. La ecuación de clasificación (1) puede escribirse entonces como

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} \geq 0 \quad \Leftrightarrow \quad y = 1\,.$$

La predicción de y, la cuál denominaremos $\tilde{y}$, es entonces

$$\tilde{y} = \text{signo}(\mathbf{w}^{\mathrm{T}}\mathbf{x})\,. \tag{2}$$

Para entrenar al Perceptrón se requiere una secuencia de vectores de entrenamiento $\mathbf{x}_i$, y sus correspondientes clasificaciones $y_i \in \{-1, 1\}$. Se comienza inicialmente con un vector $\mathbf{w}$ nulo. El algoritmo a ejecutar es entonces el siguiente:

para *i desde 1 hasta n* **hacer**
  $\tilde{y}_i \leftarrow$ *predecir* y_i *aplicando* (2);
  si $\tilde{y}_i$ *no es igual a* y_i **entonces**
    $\mathbf{w} \leftarrow \mathbf{w} + y_i\,\mathbf{x}_i$;
    # $b \leftarrow b + y_i$ *implícito en la coordenada adicional*;
  fin
fin

Se puede demostrar que la repetición iterativa de este algoritmo de entrenamiento lleva a la convergencia al plano que separa a los elementos positivos de los negativos, si es que éste existe. Con cada iteración, el perceptrón se adapta mejor al set de entrenamiento. Para evitar sesgar los resultados en forma contraproducente, es conveniente reordenar secuencia de entrenamiento en forma aleatora con cada iteración.

2.4. El Perceptrón con Kernel

Podemos escribir el vector de pesos $\mathbf{w}$ como una combinación lineal de los vectores de entrenamiento, tal que

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i\, y_i\, \mathbf{x}_i\,. \tag{3}$$

Reemplazando en (2) tenemos lo siguiente:

$$\tilde{y} = \text{signo}\left(\sum_{i=1}^{n} \alpha_i\, y_i\, \langle \mathbf{x}_i, \mathbf{x}\rangle\right) \tag{4}$$

Aplicando una transformación Φ a cada uno de los vectores del espacio de características, se puede reemplazar el producto interno en (4) por una función kernel, definida en 2.2.

La intuición detras del método es la siguiente. Para clasificar un vector $\mathbf{x}$, tomamos una medida de similitud dada por el producto interno con cada uno de los vectores de entrenamiento $\mathbf{x}_i$. Si $y_i = 1$ y los vectores $\mathbf{x}$ y $\mathbf{x}_i$ son similares, esto aportará de forma tal que la suma neta tienda a ser positiva. Lo contrario ocurre si $y_i = -1$. El objetivo del entrenamiento es determinar el vector $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ tal

que los vectores con mayor peso (en valor absoluto) sean los que mejor caracterizan a cada una de las clases.

Reemplazando (3) en la regla de actualización del Perceptrón y sacando factor común, obtenemos la regla de actualización del Perceptrón kernelizado: $\alpha_i \leftarrow \alpha_i + y_i$. Comenzando con un vector α nulo, el algoritmo de entrenamiento del Perceptrón con kernel es

para *i desde 1 hasta n* **hacer**
  $\tilde{y}_i \leftarrow$ *predecir* y_i *aplicando* (4);
  si $\tilde{y}_i$ *no es igual a* y_i **entonces**
    $\alpha_i \leftarrow \alpha_i + y_i$;
  fin
fin

Nuevamente, este procedimiento puede repetirse en forma iterativa, adaptando así al Perceptrón al set de datos de entrenamiento. Notese que cada iteración requiere calcular n^2 productos internos. Un paso previo al entrenamiento del Perceptrón suele ser computar y almacenar la matriz de Gram G del producto interno utilizado en (4); esta tarea resulta ser paralelizable. En adición, es conveniente normalizar la matriz para evitar darle peso adicional a los vectores de mayor magnitud. La matriz de Gram normalizada G' es aquella tal que

$$G'_{ij} = \frac{G_{ij}}{\sqrt{G_{ii}\, G_{jj}}}$$

2.5. Distancia Entre Información

Definición 2. Sea s un string (una cadena de bits). Definimos la complejidad de Kolmogorov $K(s)$ como la longitud en bits del programa más corto que genera s.

Definición 3. Sean s_1 y s_2 dos strings, y sea $K(s_i)$ con $i = 1, 2$ la complejidad de Kolmogorov del string s_i. Definimos la *distancia normalizada de información* como

$$\mathrm{NID}(s_1, s_2) = \frac{K(s_1 s_2) - \min\{K(s_1), K(s_2)\}}{\max\{K(s_1), K(s_2)\}}$$

donde $s_1 s_2$ es la concatenación de s_1 con s_2.

Intuitivamente, la distancia normalizada de información (NID, por sus siglas en inglés) da una medida de la similitud entre dos cadenas de bits. Mientras mayor sea la redundancia en la cadena concatenada (es decir, mientras más información haya en común entre s_1 y s_2), $\mathrm{NID}(s_1, s_2)$ sera menor.

Si bien la medida provista por la distancia normalizada resulta conveniente, se ha demostrado que la complejidad de Kolmogorov K no es

computable. Para los fines prácticos, se suele aproximar $K(s)$ mediante la longitud del string que se obtiene al comprimir s con un compresor adecuado. A esta longitud la denominaremos $C(s)$. Mientras mejor sea el compresor utilizado para el tipo de secuencia que se desea comprimir, mejor será la aproximación a la complejidad de la misma. A la distancia normalizada que se obtiene aproximando K mediante C la denominaremos *distancia normalizada de compresión*, o NCD, por sus siglas en ingles. Damos entonces la siguiente definición:

Definición 4. Sean s_1 y s_2 dos strings, y sea $C(s_i)$ con $i = 1, 2$ la aproximación por compresión a la complejidad de Kolmogorov del string s_i. Definimos la *distancia normalizada de compresión* como

$$\mathrm{NCD}(s_1, s_2) \;=\; \frac{C(s_1 s_2) - \min\{C(s_1), C(s_2)\}}{\max\{C(s_1), C(s_2)\}}$$

donde $s_1 s_2$ es la concatenación de s_1 con s_2.

2.6. Un Kernel de Compresión

Sean s_1 y s_2 dos elementos del conjunto $\mathcal{S}$ de las secuencias finitas de bits. Definimos el kernel de compresión $k : \mathcal{S} \times \mathcal{S} \to [0, 1]$ como

$$k(s_1, s_2) \;=\; 1 - \mathrm{NCD}(s_1, s_2) \tag{5}$$

En la expresión, $C(s_i)$ es la longitud en bits del string que se obtiene al comprimir s_i con un compresor adecuado, y $s_1 s_2$ es la concatenación de las secuencias s_1 y s_2.

Si bien la aplicación de la distancia normalizada en experimentos previos dio resultados interesantes, es necesario destacar que (5) no cumple las propiedades necesarias para ser un producto interno. En el caso ideal en el que la compresión es perfecta, (5) cumple únicamente que su matriz de Gram es simétrica y definida positiva. Cuando se utilizan compresores reales, sin embargo, la propiedad de simetría se pierde. En adición, el cálculo de dicha matriz con una cantidad elevada de reviews se vuelve rápidamente intractable. Sin embargo, incluso con las cuestiones mencionadas, la idea de aplicar NCD a los problemas de clasificación resulta interesante e ituitivamente convincente, por lo que no descartamos todavía la posibilidad de poder adaptarla de alguna forma.

3. Diseño del Clasificador

A continuación se describen los componentes de un clasificador que utiliza AMDL para clasificar el set de datos de pruebas mencionado en la sección 1.

3.1. Clasificador AMDL

3.1.1. Módulo de Entrenamiento

- **Preprocesador:** El preprocesador recibe como entrada el set de datos de entrenamiento como un único archivo A_t en formato tsv. La tarea de este módulo es separar las distintas columnas que componen al archivo, y filtrar el contenido HTML de las reseñas. En adición, el preprocesador también puede eliminar caractéres no alfanuméricos en función de un parámetro dado. En [1], Marton et al. reportan que el rendimiento del clasificador AMDL con y sin puntuación depende del set de datos con el que se esté trabajando. Los datos han de ser levantados y procesados en memoria principal (donde asumimos que hay suficiente espacio para almacenarlos). La salida del módulo es una tupla de punteros a arreglos; estos arreglos constituyen la representación en memoria de las columnas de A_t.
- **Concatenador:** El concatenador recibe como entrada la tupla de punteros a arreglos generada por el preprocesador, y emite dos punteros a buffers en memoria principal: un buffer b^+ que contiene la concatenación de todas las reseñas positivas, y otro buffer b^-, que contiene la concatenación de todas las reseñas negativas.
- **Compresor:** En la fase de entrenamiento, el submódulo de compresión debe comprimir los datos almacenados en los buffers creados por el concatenador. La salida en esta etapa es un par de punteros a buffers b_c^+ y b_c^-, donde se almacenan comprimidos los contenidos de b^+ y b^- respectivamente.
- **Módulo de Almacenamiento:** El paso previo concluye la fase de entrenamiento del clasificador. Un paso adicional es volcar al disco el contenido de los buffers b_c^+ y b_c^-. Nótese que el proceso de clasificación es paralelizable; es recomendable entonces distribuir los datos comprimidos para llevar a cabo dicho proceso en forma más eficiente.

3.1.2. Módulo de Clasificación

Dado un subconjunto $\{r_1, r_2, \ldots, r_n\}$ del total de las reseñas a ser clasificadas, el módulo de clasificación debe tomar los buffers b_c^+ y b_c^- (levantando los datos a memoria principal de ser necesario), y proceder según el siguiente algoritmo:

para *i desde 1 hasta n* **hacer**
  # Concatenar los datos comprimidos y la reseña r_i;
  $b_c^+ r_i \leftarrow$ *Concatenación entre* b_c^+ *y* r_i;
  $b_c^- r_i \leftarrow$ *Concatenación entre* b_c^- *y* r_i;
  # Comprimir las concatenaciones recién generadas;
  $(b_c^+ r_i)_c \leftarrow$ *Comprimir* $b_c^+ r_i$;
  $(b_c^- r_i)_c \leftarrow$ *Comprimir* $b_c^- r_i$;
  # Comparar las longitudes en bits de $(b_c^+ r_i)_c$ *y* $(b_c^- r_i)_c$*;*
  si $|(b_c^+ r_i)_c| \leq |(b_c^- r_i)_c|$ **entonces**
    # Predecir r_i de clase $+1$;
  en otro caso
    # Predecir r_i de clase -1;
  fin
fin

El conjunto de reseñas a clasificar junto con los correspondientes identificadores debe ser cargado del disco. Para realizar la compresión se puede utilizar el mismo módulo utilizado en la fase de entrenamiento. Un último submódulo debe guardar los resultados de las predicciones en el formato adecuado.

3.2. Algoritmos de Compresión

En [1] los autores reportan haber utilizado tres compresores diferentes: gzip, LZW y RAR. El primero está basado en LZ77, el segundo es un método conocido basado en diccionarios, y el tercero utiliza, usualmente, PPMII para la compresión de textos. Esta información fue recopilada por los autores de [1], y probablemente esté actualmente desactualizada.

Nosotros probaremos algoritmos más recientes como LZMA y PPMC. Se tiene pensado implementar una versión básica de alguno de los algoritmos de compresión a evaluar, y comparar los resultados obtenidos con los que se obtienen utilizando bibliotecas externas populares.

3.3. Otras Posibilidades

Como se mencionó en la sección 2.6, consideramos la posibilidad de implementar un clasificador que utilice técnicas de compresión para evaluar la similitud entre objetos clasificables. El Perceptrón con Kernel y las Máquinas de Soporte Vectorial son alternativas simples que soportan funciones kernel. Una vez evaluado el desempeño y el consumo temporal del método AMDL, se tiene planeado evaluar el desempeño del kernel de compresión descripto en la sección 2.6.

Referencias

[1] Yuval Marton, Ning Wu & Lisa Hellerstein, *On Compression Based Text Classification.*

[2] Thomas Hoffman, Bernard Schölkopf & Alexander J. Smola, *Kernel Methods in Machine Learning*, 2008.