Tarea 10

Programa 1. Clasificador Bayesiano ingenuo

0.1 Metodología

Para este programa se considera el problema de clasificación de correos electrónicos en "spam" y "no spam".

Para resolver esta tarea se parte de un conjunto de datos $\{x_i, y_i\}$ con $i = 1, 2, ..., n$, donde x_i representa algún correo que se quiere clasificar y donde y_i representa la clasificación que se da al correo, esto es, $y_i = 1$ si el correo es spam y $y_i = 0$ si el correo no es spam.

Lo que se quiere estimar es $P(y|x)$, es decir, saber cual es la probabilidad de que nuestro correo x sea clasificado como spam o no spam. Como el clasificador bayesiano ingenuo es un modelo generativo, se modela $P(y|x)$ como $P(x|y) * P(y)$. Por lo que, lo que buscaremos estimar son las distribuciones $P(y)$ y $P(x|y)$.

El modelado de $P(y)$ es el más directo, ya que se puede usar el estimador de máxima verosimilitud, esto es, $P(Y = y) = \frac{\sum_{i=1}^{n} I(y_i=y)}{n}$ donde $I(y_i = y) = 1$ si $y_i = y$ y $I(y_i = y) = 0$ en otro caso.

Debido a que X representa un vector de las palabras presentes en un correo, $P(X|Y = y)$ lo podemos expresar como $P(x_1, x_2, ..., x_m|Y = y)$. Y como la caracteristica principal del clasificador bayesiano ingenuo es asumir la independencia de las palabras $x_1, x_2, ..., x_n$ dada la clase, podemos reescribir $P(x_1, x_2, ..., x_m|Y = y) = P(x_1|Y = y) * P(x_2|Y = y) * ... * P(x_m|Y = y) = \Pi_{i=1}^{m} P(x_i|Y = y)$.

$P(x_i|Y = y)$ es más sencillo de obtener ya que una vez más podemos usar el estimador de máxima verosimilitud, esto es, $P(x_i|Y = y) = \frac{count(w_i,y)}{\sum_{j=1}^{m} count(w_j,y)}$ donde $count(w_j, y)$ regresa el número de veces que se repite la palabra w_j en los correos con clase y.

Una vez que se han estimado $P(Y = 1)$, $P(Y = 0)$ y $P(X|Y = y)$ para cada una de las m palabras, podemos realizar la predicción para cualquier correo nuevo que llegue, regresando:

$$argmax_c\{P(Y = y|X) = P(X|Y = y) * P(Y = y)\} \tag{1}$$

Es decir, regresamos la clase y que maximiza la probabilidad $P(Y = y|X)$.

Finalmente, se hace una ligera modificación al cálculo de $P(x_i|Y = y)$ a la siguiente:

$$P(x_i|Y = y) = \frac{count(w_i, y) + 1}{(\sum_{j=1}^{m} count(w_j, y)) + m} \tag{2}$$

Donde m es el número total de palabras diferentes encontradas (vocabulario). Esto se conoce como "Suavizado de Laplace" y se realiza debido a que si el nuevo correo contiene palabras que no se encontraban a la hora del entrenamiento, la probabilidad daría cero.

0.2 Implementación y Resultados

Para obtener los conteos de palabras se implementaron 2 Hash Maps, uno para las palabras spam y otro para las palabras que no son spam.

También se tienen 2 métodos:

1. eval: Calcula métricas de desempeño del clasificador (accuracy, precision, recall).
2. predict: Dada una cierta cadena pasada por consola, regresa la predicción calculada ("spam", "not spam").

En cuanto al dataset, debido a que los correos normalmente vienen configurados para borrar correos spam después de 30 días, no era viable usar estos datos. En su lugar, se usó un dataset de la plataforma de Kaggle con un total de 5574 correos en inglés.

El dataset se dividió en un 80% para entrenamiento y un 20% para prueba.

Los resultados fueron los siguientes:
Accuracy = 0.961435, Precision = 0.816568 y Recall = 0.92

Se puede observar que no tiene buena medida de precision, lo cual nos dice que de todo lo que predice como spam sólo el 81.6% de las veces es correcto. Mientras que el recall nos dice que de todo lo que en realidad era spam logró clasificar bien el 92% de las veces.

0.3 Conclusión

Como se pudo observar en las métricas obtenidas, el "precision" obtenido no fue muy bueno, lo cual implica que de manera más frecuente, nuestro clasificador mandará correos que no debería a la bandeja de spam, lo cual no es nada deseable para esta aplicación. Aunque a través del "recall" vemos que la mayoría de las cosas que si eran spam las clasificó correctamente.

Una característica con la que se podría experimentar es el uso de n-gramas tanto a nivel palabra como a nivel caracter, ya que se podría extraer información más útil. Por ejemplo, en vez de tomar "coca" y "cola" como dos palabras diferentes, si se usaran n-gramas de ancho de 2 palabras, éstas quedarían como una sóla ("coca cola"), lo cual sería más informativo de

que se trata de propaganda de un refresco y por lo tanto es spam.

Otra modificación que se podría implementar es usar no sólamente conteo de palabras, sino algún método como tf-idf, lo cual dará una ponderación diferente a cada palabra de acuerdo a su importancia en cada una de las clases, en este caso, spam y no spam.

Programa 2. Componentes conexas

0.4 Metodología

Para este programa se considera el problema de identificar y contar el número figuras "aisladas" o "separadas" en una imagen en escala de grises y donde el fondo se considera como el color negro o valor 0 y cualquier pixel mayor a 0 se considerará como parte de alguna figura.

El algoritmo consiste en representar a la imagen como una matriz y recorrerla pixel por pixel hasta encontrar algún elemento con índices i, j que tenga un valor mayor a 0 y proceder a lanzar un recorrido BFS considerando como vecinos a los pixeles con coordenadas $\{(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1), (i-1, j+1), (i+1, j-1), (i+1, j), (i+1, j+1)\}$. Este recorrido BFS se lanza hasta terminar de recorrer la matriz.

0.5 Implementación y Resultados

El manejo de imágenes se hace mediante el formato pgm en ASCII (número mágico P2). Ya en C++, como la manipulación de imágenes se pide que se haga mediante la biblioteca STL,se usa un vector de vectores como una matriz.

También se hace uso de otro vector de vectores de tuplas(i, j), donde cada vector de tuplas representa una componente conexa y sus pixeles asociados.

Para la parte del recorrido BFS se hace uso de la estructura Deque.

A la hora de ejecutar el programa, por consola se muestra el número de componentes conexas que se encontraron y el tiempo de ejecución que tomó realizar la búsqueda de dichas componentes conexas. También se genera un archivo donde se enumera cada una de las componentes conexas indicando el tamaño (en número de pixeles), así como una imagen que contiene las componentes conexas mayor y menor.

En la siguiente imagen se muestra un ejemplo.

Tarea 10

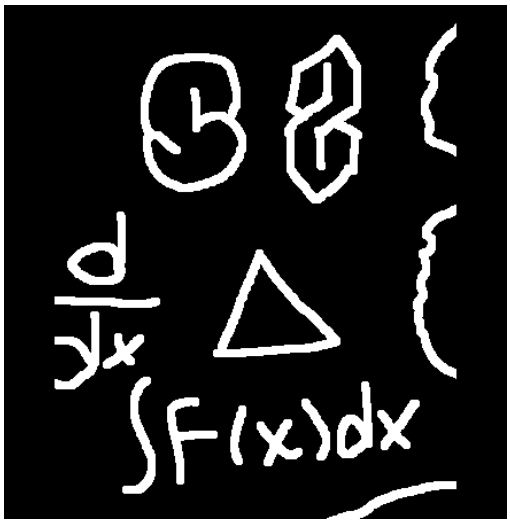

Figure 1: Imagen original

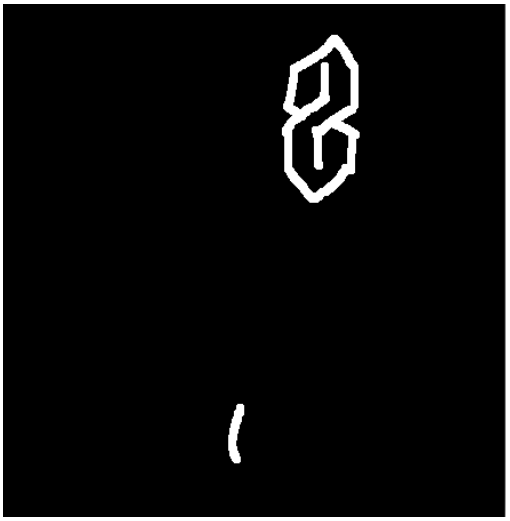

Figure 2: Imagen filtrada con componente conexa más grande y más chica

```
isaac@irb ~/Documents/CIMAT/Programación y Algoritmos/Pro
g++ -Wall -std=c++11 -O2 test.cpp cconexo.cpp -o runTest

Algorithm took 3.80015 ms to find all connected components

Writing image to file min_max_CC5.pgm

Writing sizes to file comp_sizes_CC5.txt

Se encontraron 17 componentes conexas
```

Figure 3: Salida por consola

comp_sizes_CC5.txt (~/Documents/CIMAT/Program

Open

```
1 1094
2 3692
3 3323
4 1475
5 1221
6 2342
7 664
8 975
9 540
10 963
11 1003
12 936
13 476
14 696
15 413
16 713
17 886
```

Figure 4: Archivo con tamaños de todas las componentes

0.6 Conclusiones

Una mejora que se podría implementar es el poder procesar componentes conectadas de diferentes colores, donde cada componente conectada es del mismo color, con lo que ya no sólo se podrían filtrar por tamaño sino de acuerdo a los colores que nos interesan.