Textones y clasificadores

Silvana Castillo
Universidad de los Andes
Cra 1 #18a-12
ls.castillo332@uniandes.edu.co

Laura Daza
Universidad de los Andes
Cra 1 #18a-12
la.daza10@uniandes.edu.co

Abstract

En el siguiente articulo, se utilizan textones para solucionar un problema de clasificación con la base de datos de texturas recopilada por "The Ponce Group". Se evalúa el desempeño de esta tarea por medio de dos clasificadores (Nearest Neighbor y Random Forest), calculando su porcentaje de acierto, tiempo de ejecución y matrices de confusión. Adicionalmente, se plantean mejoras para implementaciones futuras.

1.. Introducción

En el siguiente articulo se realiza la creación de un diccionario de textones y se analiza el desempeño de dos clasificadores diferentes: Nearest Neighbor y Random Forest, en un problema de clasificación de textura de la imagen. A continuación se presentara la base de datos, y la metodología para la construccion del diccionario de textones, los clasificadores.

2.. Base de datos

La base de datos usada fue compilada por *"The Ponce Group- Beckman Institute, University of Illinois at Urbana-Champaign"* y consiste de 25 clases diferentes de texturas con 30 ejemplos para cada clase en la parte de entrenamiento (750 imágenes) y 10 imágenes para cada clase en la parte de evaluación (250 imágenes). Todas las imágenes se encuentran en formato JPG, en escala de grises y con un tamaño de 640x480 pixels.[1]

3.. Metodo y filtros

El primer paso, para la creación del diccionario de textones, es crear un banco de filtros. Para esto se utilizó la función *fbCreate (numOrient,startSigma,numScales,scaling,elong)*, la cual crea diversos filtros gaussianos con base a los parámetros iniciales, formando de esta manera el banco de filtros. En este caso particular, los valores de parámetros se dejaron en los valores de la función por default, mostrados a continuación:

- **numOrient:** Cantidad de orientaciones, 8.
- **startSigma:** Valor inicial de sigma, 1.
- **numScales:** Numero de escalas, 2.
- **scaling:** Factor de escalamiento, $\sqrt{2}$.
- **elong:** Longitud, 2.

Luego de crear el banco de filtros, se seleccionaron 50 imagen de los datos de entrenamiento, 2 imagenes por categoria escogidas aleatoriamente, por lo tanto cada categoría se veía representada por dos ejemplos. Estas imagenes se concatenaron una seguida de la otra horizontalmente.

Se decidió que usaríamos 50 textones, de la misma manera que con las imágenes esperábamos que cada categoría fuera representada por dos textones. Ya con esto se utilizo la función *fbRun(fb,im)* para aplicar el banco de filtros sobre la imagen concatenada. El resultado de esto, se coloco como primer parámetro en la función *computeTextons(fim,k)* con *k=50*, la cantidad de textones que queríamos; esta función aplica k-means a *fim* (respuesta a los filtros) creando así el diccionario de textones.

Los filtros que mas discriminan son los que están orientados diagonalmente (hacia derecha e izquierda), debido a que la mayoría de las imágenes de la base de datos presenta de una u otra manera una textura con dirección diagonal (con cambios pequeños de inclinación) y las que no, son fácilmente reconocidas por su falta de "reacción" con estos filtros.

4.. Clasificadores

El único ajuste que se le hizo a la base de datos fue la normalización de las imágenes con el fin de que los valores quedaran entre 0 y 1 y fuera posible comparar.

Los clasificadores implementados fueron:

- Nearest neighbour: Usando chi-cuadrado y la funcion de Matlab fitcknn.

- Random forest: Usando la función de Matlab TreeBagger.

Antes de la implementación de ambos clasificadores, fue necesario usar la función *assignTextons(fim,textons)* con el fin de crear un mapa de textones para cada imagen de entrenamiento (la distribucion de textones en cada imagen), luego se aplico *histcounts(tmap,k)* y se guardaron también las etiquetas correspondientes, todo esto con el fin de generar un diccionario de categorías, es decir, una matriz de 750x51, donde las filas corresponden a cada imagen, las primeras 50 columnas contienen una representación de los textones como histograma y la ultima columna contiene los valores de las categorías semánticas. Se realiza el mismo procedimiento con la base de datos del test, tambien guardando las anotaciones (etiquetas de las categorías), de esta manera se obtiene una matriz de 250x51 que contiene una representación de los textones como histograma para cada imagen y sus respectivas categorías.

4.1.. Random Forest

Para la aplicación de este clasificador se utiliza la función *TreeBagger(nTrees,features, classLabels, 'Method', 'classification')* para la parte de entrenamiento, con el fin de crear el modelo de predicción. A continuación una descripción de los parámetros utilizados:

- **ntrees:** Se sabe que entre mas arboles mejores son los resultados. Sin embargo hay un punto en el que si se aumentan demasiado los arboles el costo computacional aumenta tanto que el desempeño del algoritmo disminuye incluso si las predicciones son mejores. Por lo cual, decidimos usar 20 arboles, debido a que es el doble del valor por default (10) pero tampoco es un numero lo suficientemente alto.
- **features:** Corresponde los histogramas de los textones que se encuentran en el diccionario de categorías (primeras 50 columnas).
- **classLabels:** Corresponden a las etiquetas de cada categoría, que se encuentran en la ultima columna del diccionario de categorías.
- **'Method':** Este parámetro se agrego con el fin de especificar que el metodo seria de clasificación (no regresion).
- Los demas parametros se dejaron con el default de la función, ya que no vimos ninguna necesidad de cambiarlos.

Se tomo el diccionario de categorías del test y a cada imagen (filas) se le aplico *predict* con modelo de predicción antes generado. Despues estas predicciones son comparadas con las categorías correspondientes de cada imagen con el fin de obtener un porcentaje de acierto del clasificador (usando la función *precisión* creada solo para este propósito).

4.2.. Nearest Neighbor

Para la implementacion de este clasificador se utilizo como distancia métrica (comparación entre histogramas) la distancia *Chi-square*, donde entre menor sea el valor resultante entre histogramas, mas similares son entre si. Luego se aplica Nearest Neighbor con la función *fitcknn(feature,labels,'Distance',@(x,Z)chiSqrDist(x,Z,w), 'NumNeighbors',k=1);* que es modelo de clasificación de k-Nearest Neighbor (KNN); donde cada parámetro esta descrito a continuación:

- **features:** Corresponde los histogramas de los textones que se encuentran en el diccionario de categorías (primeras 50 columnas).
- **labels:** Corresponden a las etiquetas de cada categoría, que se encuentran en la ultima columna del diccionario de categorías.
- **'Distance':** Este parámetro se agrego con el fin de especificar que la distancia para realizar la comparacion sera *Chi-square*.
- **'NumNeighbors':** Este parámetro se agrego con el fin de especificar se realizara con solo 1 vecino mas cercano.
- Los demas parametros se dejaron con el default de la función, ya que no vimos ninguna necesidad de cambiarlos.

Nuevamente se tomó el diccionario de categorías del test y a cada imagen (filas) se le aplico *predict* con modelo de predicción de NN antes generado. Despues estas predicciones son comparadas con las categorías correspondientes de cada imagen con el fin de obtener un porcentaje de acierto del clasificador (usando la función *precisión* creada solo para este propósito).

5.. Resultados

A la hora de correr los clasificadores se uso la función *tic toc* tanto en etapa de entrenamiento como evaluación, para poder hacer una comparación mas precisa del gasto computacional de cada clasificador, estos valores se pueden apreciar en la tabla 1.

Como se menciono antes las predicciones obtenidas en cada clasificador se compararon con las anotaciones correspondientes por la función *precisión*, que los compara, obtiene la cantidad de verdaderos positivos y los divide por la cantidad predicciones, para así obtener el porcentaje de

Cuadro 1. Tiempos para cada etapa

Clasificador	Entrenamiento (s)	Evaluación (s)
Random Forest	2.3524	3.3520
Nearest Neighbor	1.1006	1.3698

Cuadro 2. Porcentaje de acierto de cada clasificador

Clasificador	Random Forest	Nearest Neighbor
% de acierto	65 %	60.65 %

acierto del método. Estos valores pueden verse en la tabla 2.

Adicionalmente, para medir el desempeño de los metodos se realizaron las matrices de confusión para cada clasificador, pueden visulizarse los resultados en la figura 1.

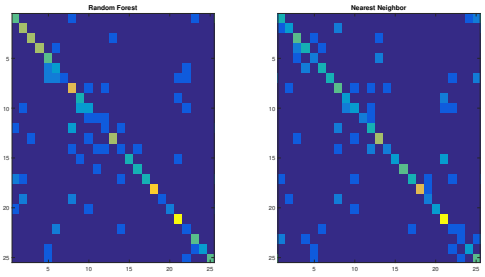

Figura 1. Matrices de confusión para Random Forest y Nearest Neighbor

6.. Discusión

En resultados no se menciono, no obstante es importante mencionar que una de las limitaciones mas grandes del método es que la etapa de creación del diccionario de textones es bastante demorada y consume mucha memoria, lo que produce un gran costo computacional (mucho mayor al dado por cualquiera de los clasificadores). Debido a que esta etapa es afin para ambos clasificadores, no se toma en cuenta a la hora de compararlos, pero como se puede ver es una etapa esencial no muy eficiente.

En relación al tiempo de ejecución de cada clasificaron (1), se obtuvo que Nearest Neighbor realiza tanto el entrenamiento como la prediccion en tiempo record, teniendo en cuenta que se estaba tomando toda la base de datos. por otro lado, Random forest se demora un poco mas del doble en el entrenamiento ($\tilde{2}$.4 segundos), y casi el tiple en la etapa de evaluacion ($\tilde{3}$.4 segundos). Esta característica permite que Nearest Neighbor pueda extrapolarse a bases de datos mas grandes, ya que tiene un costo computacional pequeño, o también permite ajustes y mejoras (que probablemente aumenten el tiempo de ejecución), por ejemplo aumentar numero de textones, y aun así converger a un resultado en poco tiempo.

En relación al porcentaje de aciertos de cada clasificador (2), se obtuvo que Random Forest tiene un $\tilde{5}$ % de desempeño mayor que Nearest Neighbor. En ambos cados el porcentaje supera el 60 %, lo que nos indica que esta prediciendo bien mas de la mitad de las categorías para cada imagen, pero también indica que hay muchas categorias en las que se confunde bastante, es necesario mejorar esto. Random Forest es mejor en este caso (no por mucho), sin embargo ambos métodos presentan un amplio margen de mejora.

En relación a la confusión entre categorías para cada clasificaron (1), encontramos que Random Forest obtiene mejores resultados en las categorías granite, wall, carpet1 y wallpaper (respectivamente 8,13,18 y 21), mientras que Nearest Neighbor en las categorias wall, carpet1 y wallpaper (respectivamente 13,18,21); a simple vista se ve que Random Forest tiene mejor desempeño en este aspecto.

En la figura 1, también se puede ver que para Random Forest las categorías que mas se confunden son floor2, pebbles, brick1 y fur (respectivamente 11, 12, 14 y 22), mientras que en Nearest Neighbor las que mas se confunden son fur y Knit (respectivamente 22 y 23). Estas confusiones están directamente relacionadas con la orientación de las categorías, ya que a la hora de clasificar las imagen con texturas muy diferentes pero que tienden en la misma dirección tienen histogramas muy similares.

Una de las limitaciones de la base de datos, es la cantidad de categorías semánticas, estas no logran representar la gran cantidad de patrones que existen, seria necesario aumentarla. Otra limitación es que las imágenes se encuentran en escala de grises, por lo que se esta perdiendo mucha información de color que podría ser relevante a la hora de clasificar la textura.

7.. Mejoras

Una posible mejora seria aumentar el numero de textones, esto para poder obtener unos resultados mas detallados para cada imagen, aumentando la información y por lo tanto aumentando las diferencias comparativas entre diferentes categorías, de esta manera el resultado final seria mas robusto y menos dado a confundir entre categorías. Se realizarían múltiples experimentos buscando encontrar el valor limite de textones que ya no logre mejorar el desempeño y a la vez comparar los tiempos de ejecución tanto de los clasificadores como de la etapa de creación de diccionario de textones, para encontrar un balance apropiado entre el costo computacional y el desempeño del algoritmo.

En el caso de Random forest, una posible mejora seria daría aumentando el numero de arboles, como se menciono antes. También se realizarían experimentos aumentando este valor hasta que el desempeño deje de mejorar o teniendo en cuanta como esto afecta el tiempo de ejecución, se llegue a un punto donde el método es demasiado costoso compu-

tacionalmente.

En el caso de Nearest Neighbor, se podria aplicar K-Neighbors con el fin de obtener resultados mas robustos y viables.

Referencias

[1] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. *A Sparse Texture Representation Using Local Affine Regions*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 8, pp. 1265-1278, August 2005.

[2] Matworks and Simulink *Funciones de Matlab*