Práctica 3 – Reconocimiento de escenas con modelos Bag-of-Words

Moreno Diez, Juan – Pascual Francés, Jaime

– – – – – – – – – – ◆ – – – – – – – – – –

3 PREGUNTASTAREAS OPCIONALES

A. Cree un esquema de clasificación de imágenes completo con los siguientes detalles:

+ Características: HOG con parámetro *tam=100*
+ Modelo: BOW sobre HOG con *max_iter=10*
+ Clasificador: KNN
+ Ratio train_test: 0.20
+ Máx número de ejemplos por categoría: 200

Para construir este esquema puede basarse en:

+ Función *sklearn.model_selection.train_test_split*
+ Función *sklearn.neighbors.KNeighborsClassifier*
+ Función *obtener_features_hog (fichero p3_tarea2.py)*
+ Función *construir_vocabulario y obtener_bags_of_words (fichero p3_tarea1.py)*
+ Función *load_image_dataset (fichero p3_utils.pyc)*

Y aplíquelo sobre el dataset de escenas *scene15* disponible en el material de la práctica. A continuación, responda a las dos siguientes preguntas:

3.1 Varíe el tamaño del diccionario BOW (hasta un valor máximo de 200) y estudie como varía el rendimiento/accuracy de clasificación sobre los datos de entrenamiento y test. Utilice k=5 (1.25 puntos)

Hemos decidido ejecutar el script ***p3_pregunta_31.py*** 5 veces y observar la media de precisión más alta para todos los tamaños del diccionario BOW.

Vamos variando el parámetro ***vocab_size*** de la función ***construir_vocabulario*** desde 25 hasta 200 dando saltos de 25 en 25.

En las filas "Accuracy-x", la x corresponde con el número de la ejecución y usamos **n_neighbours = 5**.

vocab_size	25	50	75	100	125	150	175	200
Accuracy-1	0.415	0.4083	0.4916	0.4583	0.44	0.396	0.4383	0.47
Accuracy-2	0.456	0.4483	0.4316	0.475	0.443	0.445	0.395	0.4716
Accuracy-3	0.415	0.43	0.43	0.425	0.4416	0.4483	0.4883	0.4483
Accuracy-4	0.435	0.4383	0.46	0.4183	0.4683	0.4516	0.4416	0.43
Accuracy-5	0.4216	0.43	0.455	0.4283	0.45	0.485	0.4683	0.415
Media	0.42852	0.43098	**0.45364**	0.44098	0.44858	0.44518	0.4463	0.44698

Tabla 1. Tabla con las distintas precisiones valores de vocab_size en knn

En la tabla 1, podemos observar que el mayor accuracy medio para los distintos tamaños del diccionario BOW es de **0.45364**, que corresponde a vocab_size 75.

El valor de vocab_size está asociado al número de clusters (agrupaciones) que se realizan al construir el vocabulario utilizando el algoritmo k-means. Al ser un problema con 15 clases distintas, para valores más pequeños como 25 o 50, puede que no se esté ajustando bien el modelo a los datos. Por el otro lado, al escoger un valor muy alto, puede que se esté sobre ajustando y no obtiene valores tan buenos como puede ser el de 75.

3.2 Varíe el número de vecinos utilizados en el clasificador KNN (hasta 21) y estudie como varía el rendimiento/accuracy de clasificación sobre los datos de entrenamiento y test. Utilize el tamaño de diccionario BOW que proporciona mejores resultados en la pregunta 3.1. Posteriormente, analize el rendimiento de la configuración que obtenga el mejor resultado y muestre resultados visuales de aciertos/errores (se recomienda utilizar la función *create_webpage_results***) *(*1 puntos)**

Para la realización de esta tarea, hemos ejecutado el script ***p3_pregunta_32.py*** con el parámetro n_neighbours variando desde 1 hasta 21 con saltos de 2 en 2 y vocab_size = 75. Los resultados para los distintos valores de n_neighbours han sido los siguientes:

k=1: **0.423** | k=3: **0.415** | k=5: **0.455**| k=7: **0.43**| k=9: **0.46** | k=11: **0.465** | k=13: **0.4616** | k=15: **0.465** | k=17: **0.483** | k=19: **0.493** | k=21: **0.453**

El mejor resultado al medir la precisión del modelo es de 0.493, con n_neighbours = 19, que es el parámetro que usaremos para analizar el rendimiento de forma más visual.

Se han obtenido los siguientes resultados:

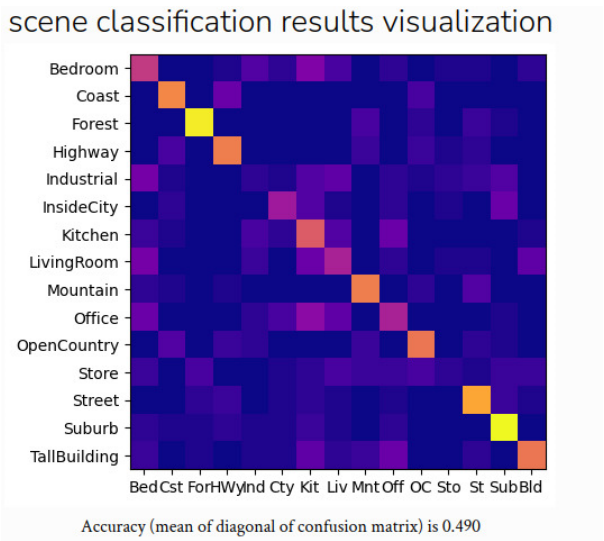

Figura 1. Matriz de confusión con knn, n_neighbours = 19, vocab_size = 75 en knn

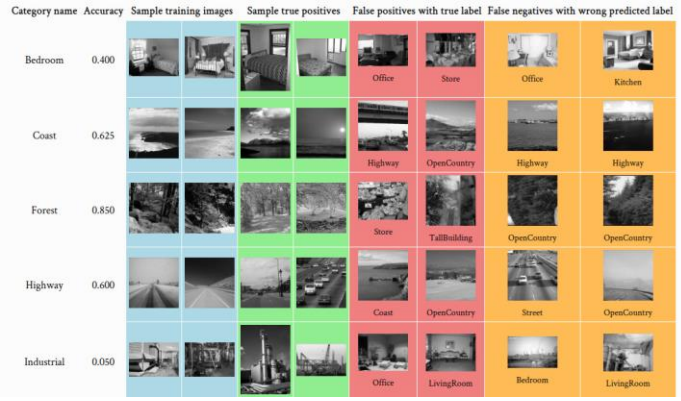

Figura 2. Ejemplos de resultados visuales de las primeras 5 clases para knn.

Lo primero es observar que KNN no saca valores de precisión muy elevados en este ejemplo. Esto se debe a que KNN es un algoritmo que funciona muy bien para problemas binarios, pero en este caso tenemos un problema de 15 clases.

Para valores muy bajos de n_neighbours, se obtienen peores resultados al tener muchas clases. Ponemos un ejemplo: si utilizamos n_neighbours = 3, estamos limitando mucho las opciones del clasificador y no le dejaremos escoger entre muchos ejemplos. En este caso nos interesa darle un valor alto a la k para cubrir el mayor número de clases posibles.

Como se puede observar en la Figura 1, se obtiene un accuracy de la matriz de confusión muy parecido al mejor accuracy obtenido en knn. También se puede ver en la diagonal que algunas de las clases si que se están clasificando de forma más correcta. Por ejemplo la detección de “Forest” y “Suburb” es elevada. Sin embargo, podemos ver que para las clases “Industrial” y “Store” no sacan buena precisión.

En la Figura 2 se pueden ver distintas características para 5 clases de ejemplo. El primer valor que encontramos es la precisión del modelo ajustado a los datos que le pasamos. La segunda columna se refiere a las imágenes de entrenamiento utilizadas, la tercera a las detecciones correctas que ha hecho el modelo. La cuarta columna son imágenes que ha detectado que son de esa clase, pero no debería y la última columna corresponde con imágenes de esa clase, pero que no ha predicho las que debería.

Para el ejemplo “Coast” (2a fila): obtiene un acurracy de 0.625, se le pasan imágenes de entrenamiento que tienen que ver con costas, clasifica correctamente algunas imágenes de costas, detecta “Highway” y “OpenCountry” como “Coast” y por último, detecta “Coast” como “Highway”.

B. Cree un esquema de clasificación de imágenes completo con los siguientes detalles:

+ Características: HOG con parámetro *tam=100*
+ Modelo: BOW sobre HOG con *max_iter=10*
+ Clasificador: SVM (lineal)
+ Ratio train_test: 0.20
+ Máx número de ejemplos por categoría: 200

Para construir este esquema puede basarse en:

+ Función *sklearn.model_selection.train_test_split*
+ Función *sklearn.svm*
+ Función *obtener_features_hog (fichero p3_tarea2.py)*
+ Función *construir_vocabulario y obtener_bags_of_words (fichero p3_tarea1.py)*
+ Función *load_image_dataset (fichero p3_utils.pyc)*

Y aplíquelo sobre el dataset de escenas *scene15* disponible en el material de la práctica. A continuación, responda a la siguiente pregunta:

3.3 Varíe el tamaño del diccionario BOW (hasta un valor máximo de 200) y estudie como varía el rendimiento/accuracy de clasificación sobre los datos de entrenamiento y test. (1.25 puntos)

Hemos decidido ejecutar el script ***p3_pregunta_33.py*** 5 veces y observar la media de precisión más alta para todos los tamaños del diccionario BOW.

Vamos variando el parámetro ***vocab_size*** de la función ***construir_vocabulario*** desde 25 hasta 200 dando saltos de 25 en 25.

Los resultados obtenidos se muestran en la siguiente tabla:

En las filas “Accuracy-x”, la x corresponde con el número de la ejecución.

vocab_size	25	50	75	100	125	150	175	200
Accuracy-1	0.4716	0.496	0.5283	0.506	0.516	0.495	0.526	0.48
Accuracy-2	0.465	0.4816	0.525	0.526	0.546	0.52	0.475	0.4916
Accuracy-3	0.473	0.4783	0.5283	0.535	0.485	0.4716	0.496	0.493
Accuracy-4	0.5216	0.505	0.546	0.54	0.503	0.513	0.476	0.495
Accuracy-5	0.475	0.486	0.546	0.545	0.5116	0.52	0.52	0.48
Media	0.48112	0.48938	**0.5346**	0.5304	0.51232	0.50392	0.4986	0.48792

Tabla 2. Tabla con las distintas precisiones valores de vocab_size en svm lineal

En la tabla 2, podemos observar que el mayor accuracy medio para los distintos tamaños del diccionario BOW es de **0.5356**, que corresponde a vocab_size 75.

En este caso, pasa algo muy parecido al 3.1, solo que obtiene mejores resultados de precisión. Ocurre que para valores o altos o bajos se obtienen peores resultados que para valores más intermedios, como son los casos de 75 y 100.

En este caso, SVM obtiene mejores resultados ya que funciona algo mejor en problemas no binarios comparado con KNN.

Como conclusión de porqué 75 es el mejor valor para el tamaño del diccionario BOW, sacamos la misma conclusión que en el apartado 3.1. Este parámetro está relacionado con el número de clusters utilizados en K-means y por lo tanto, un valor “bajo” está haciendo que no obtenga buenos resultados, y un valor “alto”, está sobre ajustando los datos del vocabulario.

3.4 Investigue y compare los distintos tipos de

kernels no lineales para clasificadores SVM (i.e. *rbf* y *poly*) sobre los datos de entrenamiento y test. Utilize el tamaño de diccionario BOW que proporciona mejores resultados en la pregunta 3.3. Razone porque se obtiene unos resultados iguales o distintos a la pregunta anterior. Posteriormente, analize el rendimiento de la configuración que obtenga el mejor resultado y muestre resultados visuales de aciertos/errores (se recomienda utilizar la función *create_webpage_results*) **(1 puntos)**

Para la realización de esta tarea, hemos ejecutado el script ***p3_pregunta_34.py*** el cual evalúa los datasets de ejemplo para un clasificador SVM con kernels no lineales (poly y rbf). Para ello hemos usado el tamaño de diccionario que nos ha proporcionado el mejor resultado en el 3.3, en concreto vocab_size **75** con una precisión de **0.5346,** como se muestra en la tabla 2.

Los resultados obtenidos al realizar *create_webpage_results* han sido los siguientes:

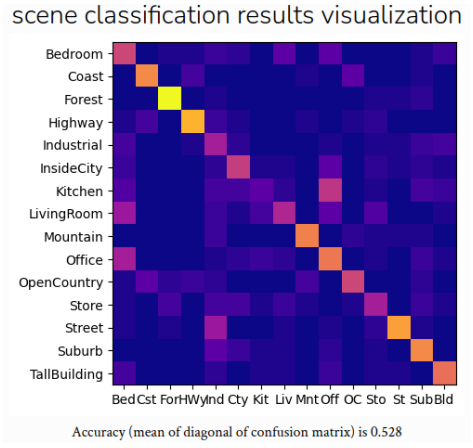

Figura 3. Matriz de confusión con vocab_size = 75 en SVM no lineal (poly).

Figura 4. Ejemplos de resultados visuales de las primeras 5 clases para SVM no lineal (poly).

En este clasificador, como podemos observar en la figura 3, existe una ligera mejora en términos de precisión con respecto al clasificador KNN. Por ejemplo, podemos observar en la diagonal de la matriz de confusión, que hay valores más altos que en la figura 1. También se puede observar que el score mejora con unas centésimas.

Esto se debe a que al utilizar un kernel no lineal, se mapean los datos a un espacio de dimensionalidad mayor donde todos los datos son linealmente separables. Además, las SVM funcionan ligeramente mejor que KNN en problemas no binarios.

En la figura 4, podemos observar las distintas precisiones y métricas para cada una de las clases. Al compararlo con la figura 2, vemos que por ejemplo, la clase Industrial, tiene una mejora considerable. Además podemos ver en la diagonal principal, que hay muchos menos valores cercanos a 0.

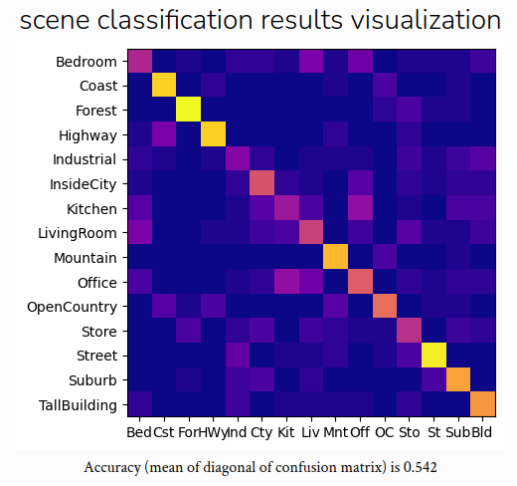

Figura 5. Matriz de confusión con vocab_size = 75 en SVM no lineal (rbf).

Figura 6. Ejemplos de resultados visuales de las primeras 5 clases para SVM no lineal (rbf).

En este clasificador no lineal, con kernel rbf, podemos ver a través de la matriz de confusión de la figura 5, que no varía mucho con respecto al clasificador con kernel poly, de la figura 3. Esto se debe a que ambos clasificadores son no lineales.

La diferencia de precisión entre los dos kernels no lineales no es demasiado notable, siendo esta de un 0,014.

En la figura 6 podemos ver las distintas precisiones y métricas para cada una de las clases. No se aprecia una gran diferencia con respecto a la figura 4 ya que en distintas ejecuciones puede clasificar mejor unas imágenes que otras.

C. Cree un esquema de clasificación de imágenes completo con los siguientes detalles:

+ Características: HOG con parámetro *tam=100*
+ Modelo: BOW sobre HOG con *max_iter=10*
+ Clasificador: Random Forest
+ Ratio train_test: 0.20
+ Máx número de ejemplos por categoría: 200

Para construir este esquema puede basarse en:

+ Función *sklearn.model_selection.train_test_split*
+ Función *sklearn.ensemble.RandomForestClassifier (con valores por defecto)*
+ Función *obtener_features_hog (fichero p3_tarea2.py)*
+ Función *construir_vocabulario y obtener_bags_of_words (fichero p3_tarea1.py)*
+ Función *load_image_dataset (fichero p3_utils.pyc)*

Y aplíquelo sobre el dataset de escenas *scene15* disponible en el material de la práctica. A continuación, responda a la siguiente pregunta:

3.5 Investigue los parámetros de la función *sklearn.ensemble.RandomForestClassifier* y seleccione solo uno de los parámetros disponibles. Compare y razone los resultados para distintos valores del parámetro seleccionado. Utilize el tamaño de diccionario BOW que proporciona mejores resultados en la pregunta 3.2. Posteriormente, analize el rendimiento de la configuración que obtenga el mejor resultado y muestre resultados visuales de aciertos/errores (se recomienda utilizar la función *create_webpage_results*) **(1 puntos)**

Para la realización de esta tarea, hemos ejecutado el script ***p3_pregunta_35.py*** inicialmente con un tamaño del diccionario de 75 e iterando con los distintos número de árboles, en concreto de 60 hasta 130 de 10 en 10 obteniendo los siguientes resultados:

n=60: **0.5083** | n=70: **0.56** | n=80: **0.543** | n=90: **0.5183** | n=100: **0. 52** | n=110: **0.506** | n=120: **0.53** | n=130: **0.5316**

Como podemos comprobar el resultado más alto obtenido ha sido con 70 árboles con una precisión 0.56. La precisión obtenida con los distintos valor de n_estimators se debe a que en este caso, seleccionar un número alto de árboles hace que el modelo se sobreajuste a los datos.

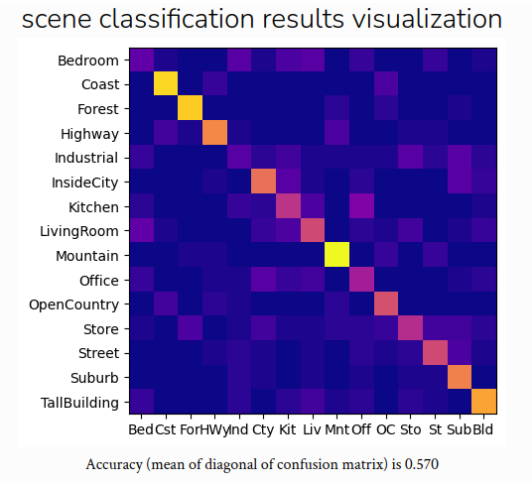

Figura 7. Matriz de confusión con n_estimators = 70 y vocab_size = 75 en random forest.

Figura 8. Ejemplos de resultados visuales de las primeras 5 clases para random forest.

De primeras, podemos observar en la figura 7 que se obtiene una precisión mayor a la de los demás modelos analizados previamente, siendo esta de un 0.57.

El análisis visual es muy parecido al ya descrito en anteriores apartados. La matriz de confusión es correcta ya que en la diagonal principal se observan los valores más grandes para las clases predichas.

En la figura 8 podemos ver las distintas precisiones y métricas para cada una de las clases. No se ven grandes diferencias con respecto a los demás modelos. Simplemente, observar que al contrario de la figura 1, en la matriz de confusión de este clasificador se obtienen valores más próximos a 0 en algunas de las clases.

4 CARGA DE TRABAJO

Indique brevemente la carga (en horas) de cada tarea de esta práctica. Puede utilizar como ejemplo la Tabla 1.

Tarea	Horas dedicadas (Jaime Pascual)	Horas dedicadas (Juan Moreno)
Tarea 1	3.5 h	3.5h
Tarea 2	1.5h	1.5h
P 3.1	3.5h	3.5h
P 3.2	0.5h	0.5h
P 3.3	2h	2h
P 3.4	0.5h	1h
P 3.5	2h	1.5h
TOTAL	13.5h	13.5h

Tabla 13. Tabla con las horas repartidas entre los integrantes de la pareja.