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

Juez Hernández, Rodrigo – Fernández Freire, Luis

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)

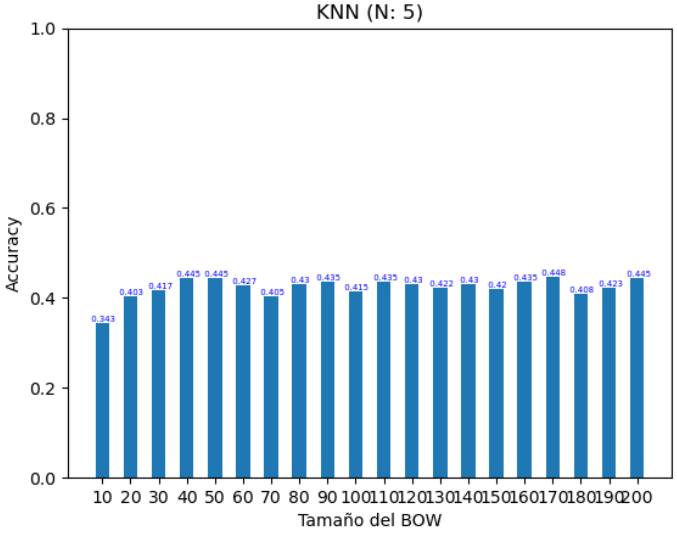

Figura 1. Pruebas de 10 a 200 tamaños.

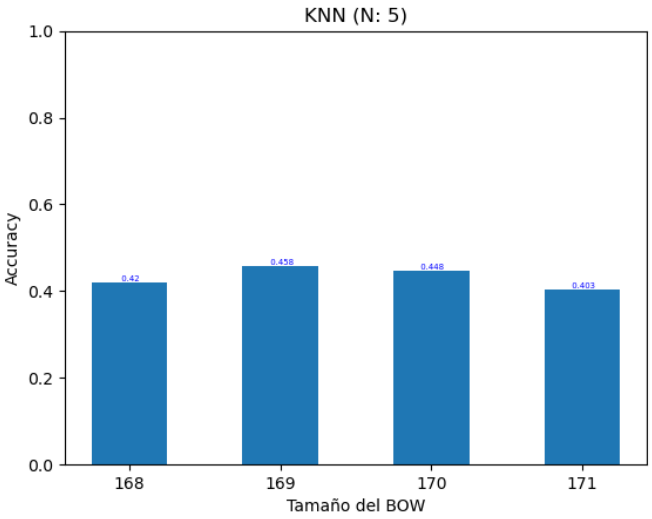

Figura 2. Pruebas de 168 a 171 tamaños.

Tras la realización de varias pruebas, determinamos que el mejor tamaño del diccionario BOW es el 169, con un 0.448 de rendimiento.
Para llegar a este resultado hicimos dos tests, de 10 en 10 para saber que valores daban mejores resultados y una vez vimos que 170 era el mejor, por lo que hicimos pruebas alrededor de 170 para asegurarnos que era el mejor valor y vimos que 169 era el mejor, puede que haya mejores tamaños en 40, 50, sin embargo, probar de 1 en 1 para todo el rango era demasiado tiempo y pensamos que esta aproximación es mejor.

Creemos que la razón para este resultado se debe a que un tamaño del BOW demasiado grande puede provocar que lo que debería ser una sola palabra son varias y cuando se genere el histograma final y se aplique KNN la distancia aumente considerablemente, ya que si falla un bin aumenta uno y se reduce el otro, sin embargo, con números altos de palabras el resultado por lo general es bueno.

Donde más se puede ver una tendencia es en vocabularios pequeños. A partir de 40 para abajo dan los peores resultados, esto es debido a que en el histograma final se combinan demasiados descriptores en pocas palabras, por lo que va a ser difícil distinguir categorías a las que bag-of-words haya asignado la misma palabra.

El número de categorías son 15, por lo que es razonable que como mínimo necesitemos 20 palabras, ya que si no

ni si quiera tendremos una palabra por categoría y dará ese resultado (0.343).

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)

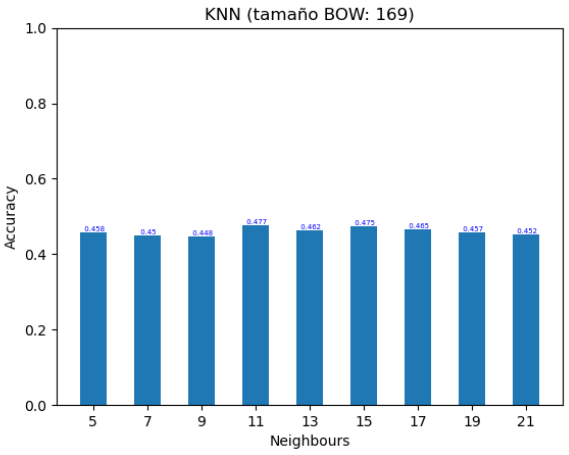

Figura 3. Pruebas de 9 a 21 neighbours.

Utilizando el tamaño del BOW adquirido en el ejercicio anterior del apartado A (169) hemos variado el número de vecino del clasificador de KNN entre 5 y 21. No hemos probado valores impares ya que pueden provocar empates.

El número de vecinos que mejora al máximo el rendimiento de este clasificador es utilizando 11 vecinos.

Por lo general se puede observar que un número bajo de vecinos reduce la puntuación y un número alto también, esto puede ser debido a que con un número muy bajo se introduce ruido y no es capaz de generalizar bien, es decir hay mucho overfitting. De igual manera un número muy alto provoca que pierda el sentido KNN ya que se asume que los puntos más cercanos son los más similares y se le estaría dando tanta importancia a los más lejanos como a los más cercanos.

Sin embargo, la mayoría de las pruebas han obtenido puntuaciones muy estables. Esto puede ser debido a que los puntos estén repartidos claramente en zonas, aunque, dada la baja puntuación, también puede ser que haya muchos fallos en todos, pero varíen mucho entre los que aciertan, pero la puntuación final no lo muestra. Para comprobarlo generamos varias páginas web y vimos los scores de cada clase y que imágenes erán más conflictivas.

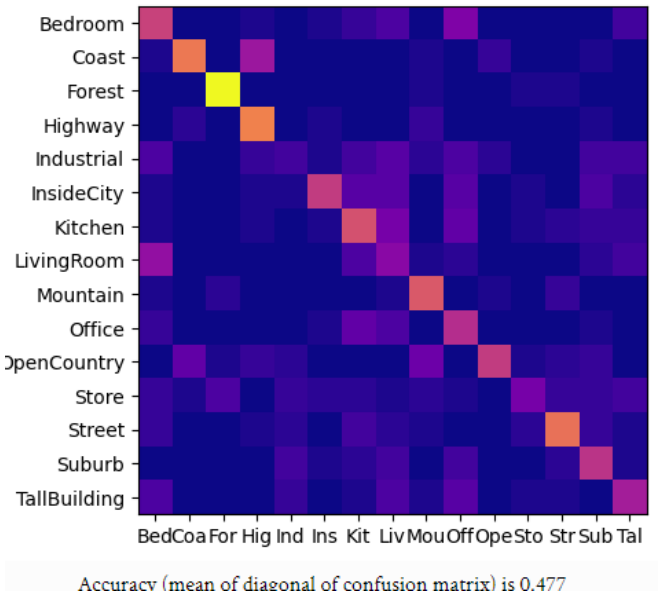

Figura 4. Confusion Matrix 11 neighbours.

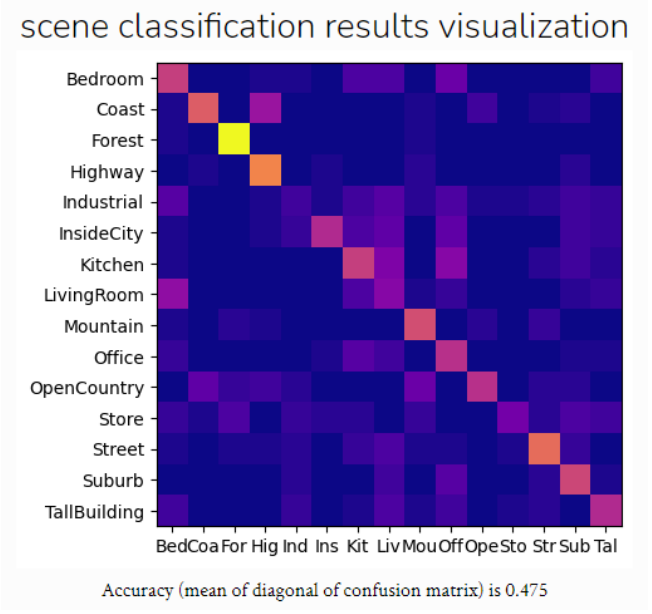

Figura 5. Confusion Matrix 15 neighbours.

Nótese como Forest tiene una puntuación perfecta de casi 1 en ambos, esto es debido a que si vemos las imágenes siempre tienen formas muy simlares de arboles y vegetación:

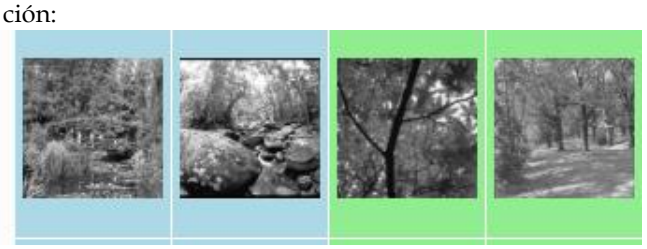

Figura 6. Forest ejemplos clasificados correctamente

También en ambos Industrial funciona de manera incorrecta, esto seguramente sea a que no tiene formas fijas, unas veces son torres, otros mecanismos y otros edificios que parecen casas, entonces siempre se clasifican como otras categorías antes que industrial.

Otras confusiones razonables son Coast y Highway ya que ambos tienen patrones y características muy similares, dos regiones muy planas (cielo y parte de abajo carretera/mar) separadas por el horizonte.

Figura 7. Falsos Positivos y Falsos negativos de highway.

También nos hemos fijado que cuando se comparten varios elementos se confunde, por ejemplo, hay fotos con montañas y carreteras, que a lo mejor clasifica como montaña, pero el dataset indica que es una carretera.

Después de leer el artículo [1] "*Distinctive Image Features from Scale-Invariant Keypoints*" creemos que una técnica que puede mejorar los resultados es una interpolación trilinear en los histogramas de gradients para distribuir los valores de los histogramas a bins adjacentes, esto en principio puede hacer que pequeñas variaciones se filtren y al crear el vocabulario se confunda menos y agrupe mejor. También puede ser que provoque underfitting y se reduzca la puntuación, pero creemos que el problema aquí es que no generaliza bien.

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)

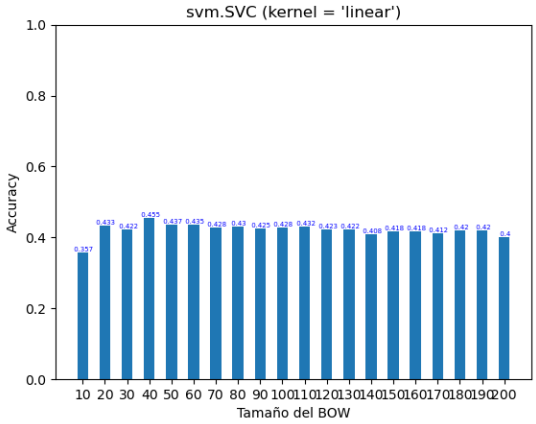

Figura 8. Tamaños de 10 a 200 con SVM linear.

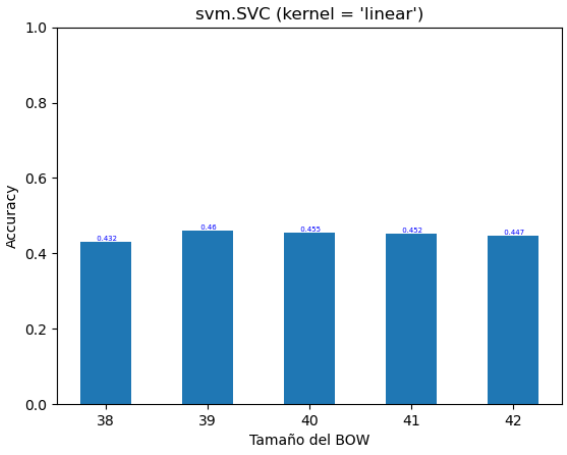

Figura 9. Tamaños de 48 a 42 de 1 en 1 con SVM linear

El mejor resultado ha sido el de 39 con un score de 0.46, de nuevo vemos patrones muy similares a los de KNN, con cantidad de palabras muy bajas como 10, no son suficientes para categorizar bien las imágenes. Además, vuelve a ocurrir que justo empeora notablemente cuando pasa de 20 a 10 palabras.

La razón para los cambios cuando variamos de 1 en 1 la cantidad de palabras puede ser que añadiendo una sola palabra más muchos descriptores pasen de asignarse a un centroide a otro completamente distinto y eso genere que el histograma sea distinto y SVM linear no sea capaz de adaptarlo.

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)**

De la misma forma que operamos en la sección 3.2, los volvemos a hacer en esta sección, pero con el clasificador de SVM.

El tamaño del vocabulario que nos ha proporcionado mejor rendimiento ha sido el 39.

A continuación, hemos aplicado al clasificador SVM dicho tamaño, pero variando su kernel con todos los valores posibles. Lo que nos ha producido el siguiente histograma tras su ejecución:

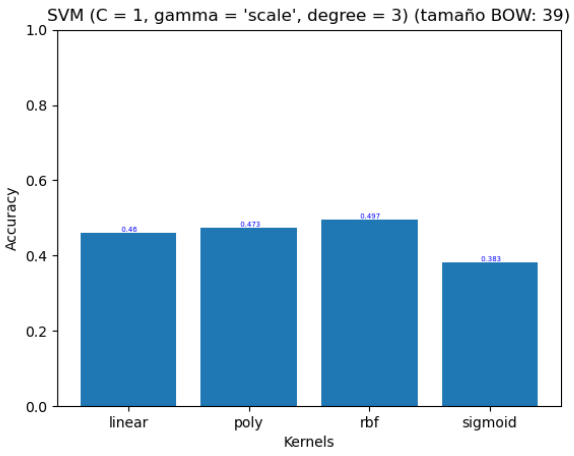

Figura 10. Diferentes kernels de SVM con TAMAÑO 39

Tras una visualización de la gráfica, podemos determinar, que el mejor rendimiento es con **0.497** utilizando el kernel de "**rbf**".

Nota: La razón por la que no hemos utilizado el kernel "**precomputed**" es porque había que preprocesar los datos.

Ahora bien, los resultados del histograma del kernel "**linear**", "**poly**" y "**rbf**" son muy similares, mientras que el kernel de "**sigmoid**" se aleja bastante del resto (siendo el peor de todos). Esto se debe a las ecuaciones utilizadas para el kernel:

Kernel linear (lineal o polinómica de grado 1):
y = ax+b

Kernel poly (polinómico, de más de grado 1):
K(x, y)= x·y +x2·y2

Kernel rbf (Radial Basis Function, Gaussiano):

$$K(\vec{x}^i, \vec{x}^j) = e^{-\left(\frac{\|\vec{x}^i - \vec{x}^j\|^2}{2\sigma^2}\right)} \equiv e^{-\left(\gamma\|\vec{x}^i - \vec{x}^j\|^2\right)}$$

Kernel sigmoid (sigmoidal):

$$h_\theta = \frac{1}{1 + e^{\theta^T x_i}}$$

Es bien conocido que el kernel "sigmoid" funciona mejor con clasificaciones binarias, y aquí estamos trabajando con multiclase, por lo que eso explica su mal resultado, es bastante famoso porque se origina de las redes neuronales. El kernel "poly" se basa en un polinomio y el grado a usar en el, salvo excepciones en la mayoría de datasets suele tardar más en ejecutar y obtener peor puntuación que "rbf". Finalmente, el mejor kernel es "rbf" y esto es porque es el que se suele elegir para separar datos no lineales (como los nuestros) y sin tener conocimientos previos del dataset.

A continuación, generamos la página web para examinar más de cerca la clasificación.

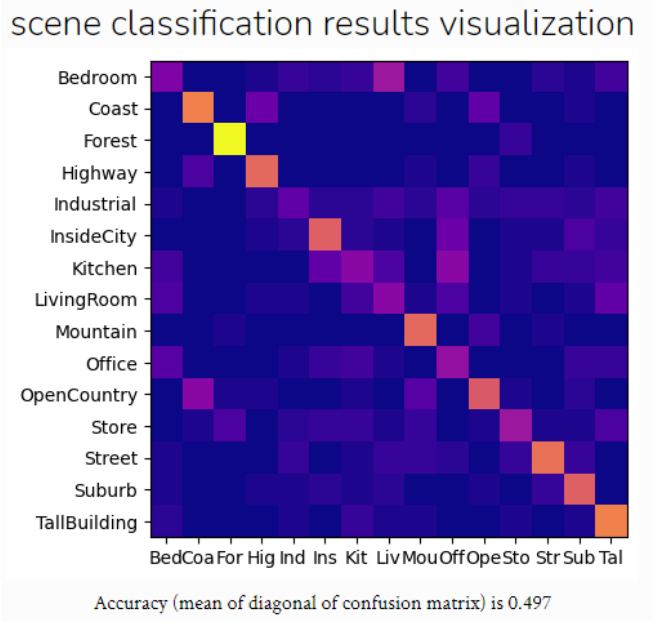

Figura 11. Confusion Matrix BOW 39 y "rbf" kernel.

De primeras nos hemos fijado en la gran mejoría que hay en TallBuilding respecto a otros algoritmos de clasificación, estas fotos suelen tener formas muy verticales y similares, también mejora en suburb, parece que a este clasificador le gustan más los edificios por fuera. Pero no se le dan tan bien los interiores, como kitchen y office.
Esta mejoría puede ser debido a que las formas son más simples y organizadas en los edificios en exteriores mientras que los interiores tienen formas más complejas.

Figura 12. Ejemplo de los patrones verticales de TallBuilding.

Figura 13. Ejemplo de la complejidad de Kitchen

Respecto a otras categorías tenemos que Coast se confunde en los mismos casos que KNN, Forest sigue siendo el que mejor rendimiento da e Industrial no consigue mejorar absolutamente nada.

Ahora vamos a realizar hyperparameter tuning con el fin de obtener los mejores resultados con SVM y acercarnos a la barrera del 70%.
El mejor resultado con los valores por defecto lo conseguimos con "rbf", C = 1 y gamma = "scale". Pero este resultado solo era un 0,497.

Con nuestros experimentos encontramos que el único kernel que merecía la pena probar era "rbf" ya que "poly" no conseguía alcanzar a "rbf" ni tocando el grado del polinomio ni C. Con "rbf" encontramos que el gamma "auto" destrozaba la puntuación y un gamma fijo empeoraba ligeramente la puntuación por debajo del baseline. Por lo que nos limitamos a dejar el gamma por defecto en "scale" y a probar C y el tamaño del BOW. Por casualidad probamos un tamaño de vocabulario distinto al mejor con C = 1 (39) y vimos que 169 funcionaba muy bien con C = 10.

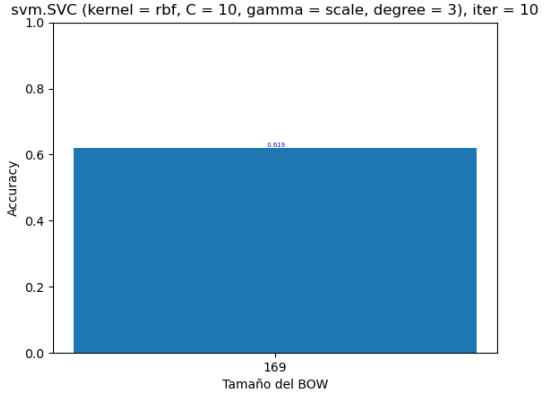

Figura 15. Puntuación de 0.619 con C = 10 y BOW = 169

También observamos que C = 100 no mejoraba el score, asi que ejecutamos el test de BOW con C = 10 para ver cual era el mejor valor pero obtuvimos que de nuevo era el 169, ya que otros valores no llegaban a 61,9% se quedaban en 61,2%.

Para entender porque mejora tanto debemos saber que el valor C = 10 indica que el margen para el hiperplano del SVM, en otras palabras, para mayores valores de C el margen será menor e intentará clasificar todos los puntos del entrenamiento mejor. Creemos que C = 1 provocaba underfitting y C = 100 provoca overfitting porque el score disminuye.

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)

Tras leer el el **Random Forest** de sklearn en https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Hemos determinado que el mejor parámetro a variar en el clasificador es "**n_estimators**". Eso sí, no tenemos en cuenta el parámetro de "**random_state**" debido a que a la hora de representar el rendimiento del clasificador necesitábamos que fuera determinista, pues si fuera al contrario los resultados de cada ejecución con los mismos parámetros serían distintos y nos resultaría difícil estimar una evaluación correcta de la precisión del clasificador.

Una vez aclarado esto, explicaremos por qué hemos elegido **n_estimators** como parámetro para probar. *"Criterion"* no nos permitía mucha comparación además creemos que el que viene por defecto ya es correcto. Todos los que conciernen máximos y mínimos (*max_depth, min_samples_split, …)* tampoco decidimos cambiarlos ya que los valores por defecto suelen permitir que se ejecute hasta que se encuentre un valor óptimo.

Ahora pasamos a explicar y comparar los resultados del rendimiento que hemos obtenido de **Random Forest** con un **tamaño del diccionario** de **169** (que ha sido el que mejor rendimiento nos ha dado en nuestras pruebas de KNN en el apartado 3.1 y utilizado en el 3.2 de dicha memoria):

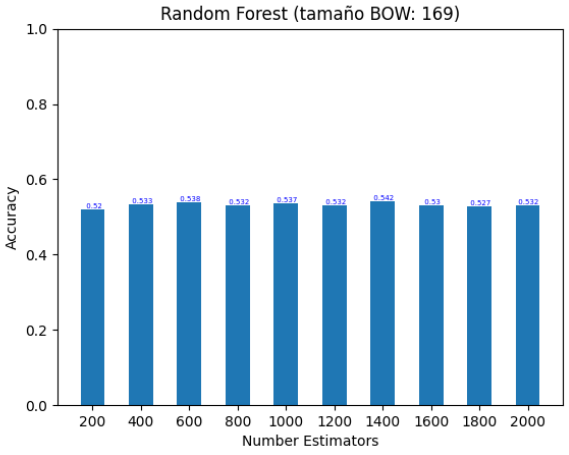

Figura 17. Variación de numero de estimators de 200 a 2000 en RF.

De acuerdo con los resultados que hemos adquirido y representado en la Figura 17 podemos determinar que el mejor **rendimiento** alcanzado es **0,542** (un resultado bastante próximo al indicado en enunciado que era **0.55**), donde el parámetro "**n_estimators**" es igual a **1400**.

Ahora bien, el histograma que se nos presenta proporciona unos rendimientos muy similares, con una diferencia de **0.022** entre el máximo y mínimo valor de los mostrados.

Random Forest es un algoritmo de clasificación que utiliza un número **X** de árboles de decisión paralelizados. Tras la realización de sus cálculos, se escoge de entre todos ellos el más probable.

Debido a esto, si se utilizan pocos árboles la profundidad de cada uno de ellos puede ser muy distinta, por lo que darán resultados muy dispares y la probabilidad de equivocarse en su estimación es más elevada.

Por otro lado, si se utilizan muchos árboles, la mayoría serán bastante similares y con niveles de profundidad de cada árbol equitativos en su mayoría, dando resultados muy parecidos entre todos ellos.

Esto se puede verificar en la siguiente gráfica, donde hemos variado el número de árboles entre el rango [10 y 200] con un incremento en 10.

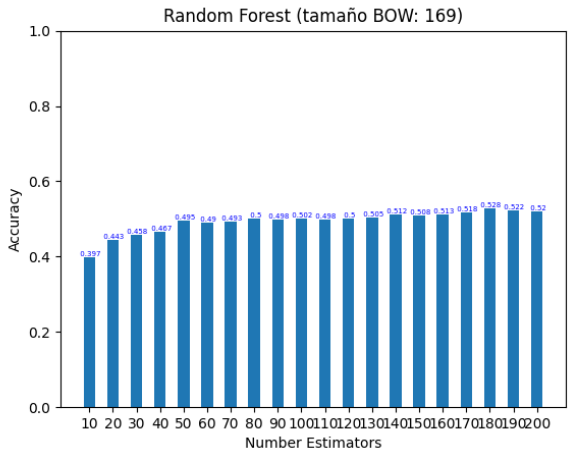

Figura 18. Variación de numero de estimators de 10 a 200 en RF.

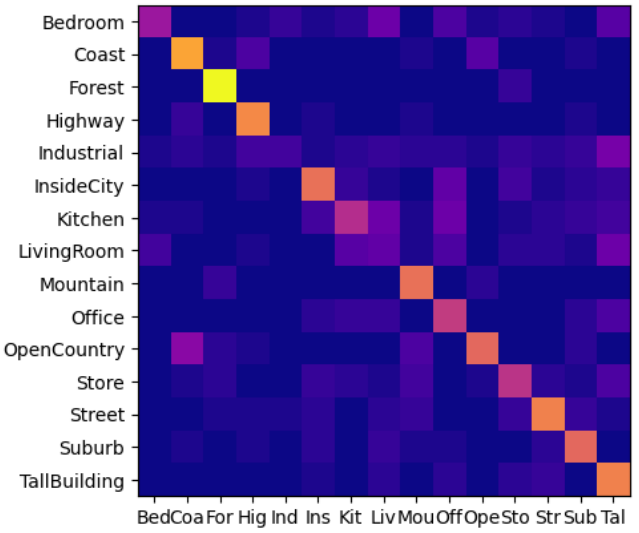

Figura 19. Variación de numero de estimators de 10 a 200 en RF.

Como se puede ver, el rendimiento aumenta considerablemente de **1** a **50** estimaciones. Por encima de **50**, **Random Forest** se empieza ha aproximar y a ajustar a **0.55** de precisión. La diferencia del rendimiento que se puede obtener a partir de estos "**n_estimators**" con respecto a los utilizados en la Figura 17 son bastante bajos (a diferencia de los tiempos de ejecución, que aumentan a medida que se aplica una mayor cantidad de árboles).

De nuevo vemos un comportamiento muy similar al que tenemos con la variación de los tamaños de vocabulario, bajando de 20 el accuracy baja de manera pronunciada, creemos que es porque un número tan bajo de árboles no es capaz de delimitar bien las distintas clases.

Tras esto, hemos utilizado la función "**create_webpage_results**" que se nos ha proporcionado para obtener, entre otras cosas, la **matriz de confusión** del **Random Forest** utilizando un tamaño de diccionario para el BOW de **169** y un **1400** árboles.

Con la matriz de confusión de la Figura 5.5, podemos concluir que el **Random Forest** calcula en su mayoría correctamente. Las secciones donde se asignan más datos se muestran con una tonalidad más amarilla, mientras donde menos datos se detectan se representa con un tono azulado. Es por esto por lo que podemos decir que el conjunto de datos que más falla es el de **Industrial** y el que acierta es el de **Forest**, un comportamiento que se parece mucho a los anteriores tests que hemos hecho, sin embargo, esta vez Coast no se confunde tanto con Highway si no con OpenCountry, que viendo ejemplos se puede entender por qué, la estructura de las imágenes es muy similar, de nuevo son regiones planas delimitadas por el horizonte.

Figura 20. Falso positivo y Falso negativo de Coast. Demuestra la similitud entre OpenCtry y Coast.

4 CARGA DE TRABAJO

Indique brevemente la carga (en horas) de cada tarea de esta práctica.

	Horas dedicadas	
Tarea	**Rodrigo Juez**	**Luis Fernández**
Tarea 1	0.5	0.5
Tarea 2	1	0.5
P 3.1	2	1
P 3.2	1	1
P 3.3	1	1
P 3.4	1	1
P 3.5	0.1	1
TOTAL	6.6	6.5

Tabla 1. Tabla de Horas de trabajo

REFERENCIAS

[1] Lowe, D. G. (2004). "Distinctive image features from scale-invariant keypoints." International journal of computer vision, 60(2), 91-110

[2] Richard Szeliski, "Computer Vision: Algorithms and Applications" 2020, http://szeliski.org/Book/2ndEdition.htm