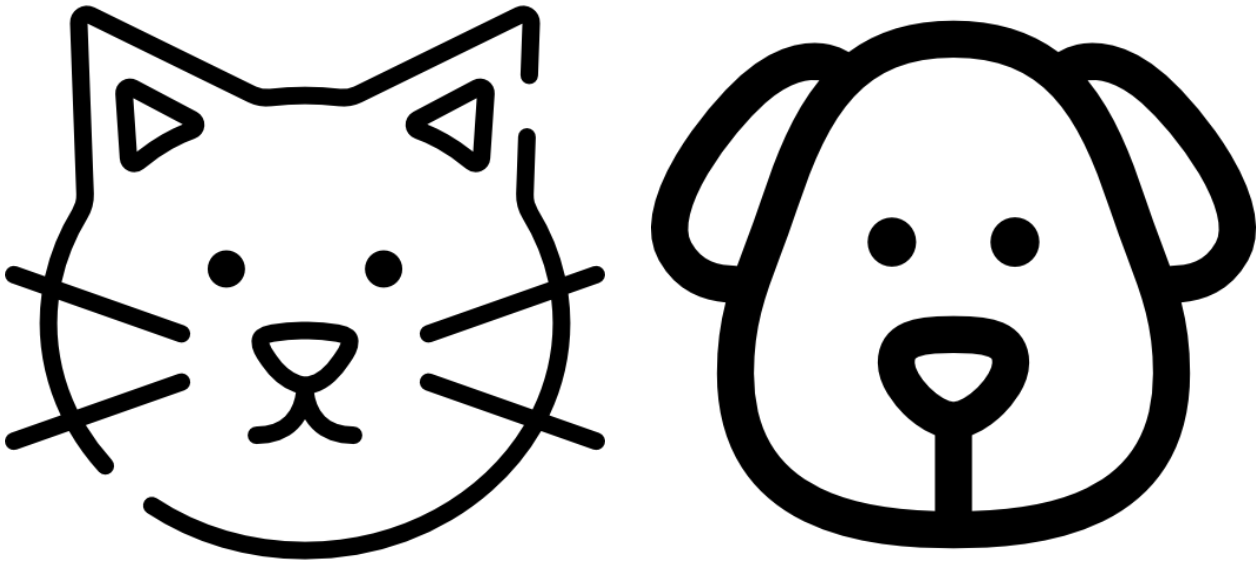
Clasificador de imágenes en perros y gatos



Julen Rodríguez Meneses y Miguel Sagaseta de Ilúrdoz Sánchez

Visión Artificial

Índice

Enlace al vídeo:

https://www.youtube.com/watch?v=QAiyNSAWSSQ

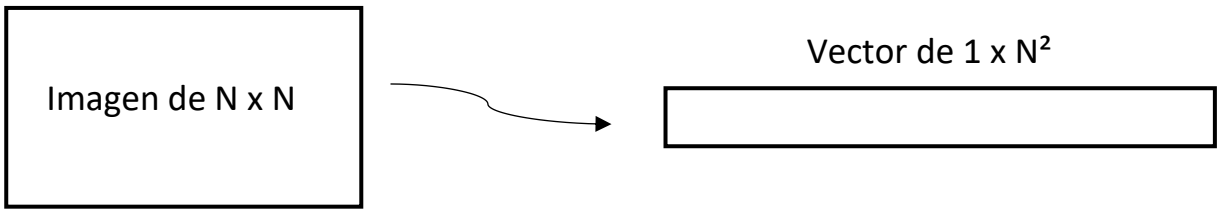
Introducción y objetivos

Para el ojo humano, distinguir un perro de un gato es una tarea simple, casi trivial. No obstante, a la hora de automatizar este proceso, es decir, pedirle a una máquina que lo haga, el problema ya no se convierte en algo tan sencillo. Nosotros vamos a intentar resolver ese problema o, por lo menos, llegar al resultado menos malo.

El objetivo de este trabajo es, por tanto, generar un modelo capaz de clasificar correctamente un conjunto de imágenes de perros y gatos de forma que, dada una nueva imagen de uno de estos animales, sea capaz de indicar qué es. Para llevar esta tarea a cabo hemos dividido el proceso en dos partes, la **extracción de características** y el **entrenamiento de los modelos.**

Extracción de características

La extracción de características consiste en, para cada imagen, extraer un conjunto de datos que sean capaces de "caracterizarla" (distinguirla de forma unívoca) de las demás, para así tratar y entrenar con ella los diversos modelos de aprendizaje. En el caso de las imágenes, sabemos que estas son "vistas" por el ordenador como una matriz de números, con tanta dimensión como píxeles tenga esta, y cuyos valores vienen definidos por la intensidad. No obstante, está claro que, dado un problema con muchas imágenes o donde estás sean de tamaño considerable, no podemos permitirnos usar todos los píxeles de la imagen como ese conjunto de datos característico, pues resultaría cerca de incomputable. En su lugar, vamos a implementar dos métodos de extracción de características con los que obtendremos un **vector característico** para cada imagen.



Modelo de extracción 1: Vamos a omitir, tanto para este modelo como para el siguiente, la parte que ya viene explicada en el enunciado, a fin de ser concisos. Solo como resumen, diremos que el primer modelo se basa en la idea de dividir la imagen en celdas (conjunto de píxeles) disjuntas, crear su histograma y concatenarlo, de forma que eso sea el vector característico de la imagen. Sobre este modelo, nosotros hemos aplicado las siguientes modificaciones:

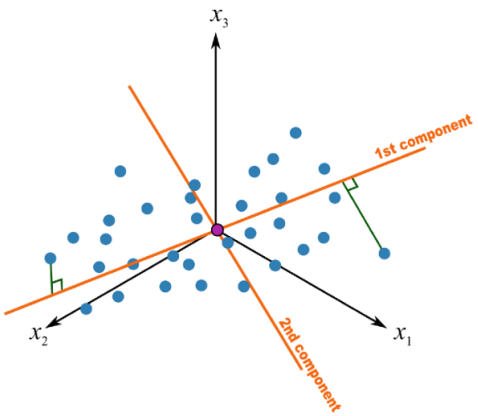
1. **Implementación a color:** En este caso, el algoritmo original no se ve afectado en nada, simplemente, además de hacer lo antes mencionado, iteraremos sobre las tres matrices de color de las imágenes, concatenando los histogramas de color uno detrás de otro. Esto nos creará vectores el tripe de extensos, pero que, presumiblemente, tienen también el tripe de información.

2. **Transformación uniforme:** Antes no lo hemos explicado, pero la forma en que se genera el histograma de cada celda no es simplemente con las intensidades de los píxeles. Primero, se calcula para cada píxel

un valor binario en función de sus vecinos, que serán 0 o 1 según sean mayores o menores que el mismo. En esta modificación, lo que haremos será tratar aquellos valores binarios que no sean uniformes, es decir, que tengan más de dos saltos de 0 a 1 o viceversa. Los valores uniformes seguirán codificándose como siempre, usando su propio valor. A los no uniformes, por el contrario, se les asignará un mismo valor. De esta forma, reducimos considerablemente la longitud del histograma, pues pasamos de 256 valores a apena 59, pues solo existen 58 valores binarios uniformes entre el 0 y el 255. Una reducción del 77%.

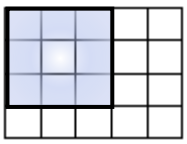
En el valor 58 (el 59º) del histograma caerán todos los no uniformes. A los sí uniformes se les asignan los 58 anteriores de forma arbitraria, por orden creciente.

3. **Principal Component Analisis (PCA):** Por último, para reducir aún más la dimensionalidad y hacer nuestros datos más sencillos de manejar, hemos implementado el PCA. Esta técnica se basa en extraer, como su nombre indica, las componentes principales de los datos, de forma que, por ejemplo, manteniendo solo 10 valores de todo el vector de características, conservemos el 90% de la información de la imagen. El proceso mediante el cual se consigue esto lo omitiremos en esta explicación.

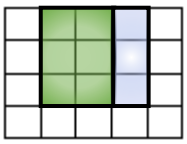


Modelo de extracción 2: En esta segunda aproximación, buscaremos, ante todo, crear vectores característicos más fáciles de manejar. Básicamente, la idea vuelve a ser dividir la imagen en celdas, pero en este caso los vectores de estas serán de tan solo 9 características (bins), que corresponden a una serie de orientaciones del gradiente que hemos definido anteriormente y a los que les vamos sumando la intensidad de los píxeles que se correspondan con ellos. Sobre este modelo, nosotros hemos aplicado las siguientes modificaciones:

1. **Implementación a color:** Nuevamente, repetimos lo ya explicado anteriormente. Ningún cambio en la implementación, simplemente iteramos y concatenamos.
2. **Normalizar el histograma:** En este caso, hemos unido en una sola modificación las dos propuestas en el enunciado del trabajo. Este proceso se divide en dos partes.
   En primer lugar, se normaliza el histograma de cada celda teniendo en cuenta sus vecinos. Esto es, coger un bloque (conjunto de celdas) y normalizarlas entre sí.
   En segundo lugar, hacemos que estos bloques no sean disjuntos, si no que se vayan moviendo por la imagen como si de una convolución se tratara. Con esto tendremos histogramas "repetidos" en tanto a que pertenecientes a una misma celda, pero que también nos aportarán más información.



Genero 9 vectores normalizados en función de los demás.



Genero otros 9, de forma que 6 son "repetidos".

Entrenamiento de modelos

Una vez extraídas las características de las imágenes, llega la parte verdaderamente interesante, el entrenamiento de los modelos, los sistemas que usaremos para tratar de clasificar correctamente las imágenes de perros y de gatos. Antes de comenzar con los modelos en sí, vamos a realizar una serie de aclaraciones.

Como bien sabemos nuestro problema se trata de uno de **aprendizaje supervisado** esto es, uno donde los modelos que entrenamos saben a priori a que clases pertenecen nuestras imágenes. Para simplificar, hemos decidido de forma arbitraria que la **clase 0 son gatos y la 1 son perros.**

También por limitaciones de los modelos se han tomado decisiones a la hora de leer las imágenes originales. En este caso, **estas deben tener la misma dimensión** (mismas filas y columnas) entre sí para poder crear vectores de la misma longitud, del mismo número de características. Para hacer esto, sin entrar en detalles de implementación, hemos usado la función *resize* de la librería CV2.

Para escoger los **hiperparámetros** de los distintos modelos, hemos optado por usar la librería *GridSearchCV* de *sklearn*. Lo que hace esto es, dados una serie de potenciales hiperparámetros, calcular que combinación de estos produce los mejores resultados, certificando los resultados con un conjunto de validación.

A la hora de medir los resultados, vamos a separarlos en función del modelo utilizado para el cálculo de los vectores característicos. El PCA lo dejaremos también aparte, pues puede aplicarse a cualquier caso.

Por último, en caso de que se quiera ejecutar el código correspondiente, se advierte de que el cálculo de los vectores característicos toma un tiempo. Para 240 imágenes, hablamos de unos 7 minutos de ejecución. Para evitar la espera constante, nótese que se adjuntan junto con los *scripts* una serie de ficheros con el siguiente código en el nombre: MODELO_MODIFICACION_CARPETA_TRAIN o TEST.txt, de los cuales se pueden leer rápidamente los vectores previamente calculados. Todo esto está indicado en el código adjunto. Por esta misma razón de lentitud, la mayoría de los resultados expuestos más abajo se han calculado con el conjunto de 100 imágenes.

Modelo 1: Regresión Lineal

El primer modelo que hemos implementado se trata de una regresión lineal. Esta calcula la "tendencia" de los vectores hacia un valor en particular, que en nuestro caso solo podrá ser 0 o 1. Ahora la pregunta es ¿por qué usar una regresión lineal en un problema que claramente no lo es? Naturalmente, salta a la vista que nuestro problema es uno de clasificación, es decir, de regresión logística, y que por tanto usar la lineal no debería servir de mucho o directamente ser inútil. Sin embargo, quisimos probar a implementar este modelo para ver qué resultados era capaz de obtener de entrada, y de ahí ir mejorándolos. Cabe destacar que la regresión lineal no devuelve un valor 0 o 1, sino uno entre ambos, con lo que redondeamos el resultado para acercarlo a la clase a la que más se parezca.

Los resultados obtenidos por la regresión lineal son:

> **0.494 de coeficiente de determinación** (medida típica en regresión lineal que mide la calidad de este para replicar resultados). Luego nuestro modelo no es especialmente bueno generalizando.
>
> **Resultados de precisión:**

	Modelo extracción 1	*Modelo extracción 2*	*PCA*
Versión estándar	85%	82.5%	75%
Versión a color	80%	80%	-
Versión uniforme/regularizada	85%	82.5%	-

Curiosamente, este primer modelo devuelve resultados mucho mejores de los que habríamos pensado. De media, clasifica bien unas 30-34 imágenes de las 40. Ya con este primer modelo, observamos varias cosas interesantes. En primer lugar, las imágenes falladas por el modelo, independientemente del método utilizado para el cálculo de vectores propios son siempre las mismas. Esto nos hace pensar en la posible presencia de **imágenes**

conflictivas. Haremos más hincapié en esto luego. También nos permite observar el funcionamiento del PCA. Salta a la vista que este produce resultados peores, pero cabe destacar que el algoritmo es muchísimo más rápido en este caso. PCA se puede implementar de muchas formas, pero nosotros hemos optado por conservar únicamente las primeras 15 componentes principales de cada imagen. Esto, en base a lo aprendido en ingeniería del conocimiento.

En conclusión, si bien la regresión lineal no es un modelo óptimo para nuestro problema, produce resultados más que aceptables.

Modelo 2: Regresión logística

La regresión logística o clasificación consiste en, como su nombre indica, separar el conjunto de datos en una serie de clases. Estas vienen indicadas previamente, recordemos que estamos en un problema de aprendizaje supervisado. De entrada, se diría que una clasificación es el modelo ideal para nuestro problema, que consiste exactamente en eso. Veamos si es cierto analizando los resultados.

Resultados de precisión:

	Modelo extracción 1	*Modelo extracción 2*	*PCA*
Versión estándar	82.5%	82.5%	70%
Versión a color	80%	80%	-
Versión uniforme/regularizada	85%	82.5%	-

Matriz de confusión (extracción 1, versión uniforme):

		Clase predicha	
		Gatos	Perros
Clase real	Gatos	14	6
	Perros	0	20

Curiosamente, los resultados obtenidos por la regresión logística son iguales (e incluso algo peores) a los de la regresión lineal pese a que este modelo debería producir mejores resultados por la lógica misma del problema. ¿Qué ha ocurrido? Tras un breve análisis de las imágenes que no estamos prediciendo bien (curiosamente siempre son gatos) nos dimos cuenta de que las imágenes falladas en la regresión lineal y las falladas aquí son las mismas. Volvemos a las anteriormente mencionadas **imágenes conflictivas**, aquellas que, por algún motivo, confunden a nuestro clasificador haciéndole pensar que pertenecen a otra clase.



Estas son tres de las imágenes conflictivas, imágenes que, al menos en apariencia, no parecen especialmente extrañas o difíciles de predecir, y, sin embargo, fallan en los dos algoritmos implementados hasta el momento. De momento, no vamos a dar mayor importancia a estas imágenes, continuamos entrenando modelos para ver si alguno es capaz de predecirlas bien.

Modelo 3: Naive Bayes

Vamos a cambiar totalmente de clasificadores. En lugar de emplear regresiones, vamos a utilizar un modelo probabilista como lo es Naive Bayes. Sabemos además que, dentro de los modelos probabilistas, Naive Bayes es uno genenativo, lo propio, de nuevo, para el aprendizaje supervisado. Analicemos los resultados obtenidos.

Resultados de precisión:

	Modelo extracción 1	*Modelo extracción 2*	*PCA*
Versión estándar	77.5%	72.5%	50%
Versión a color	75%	70%	-
Versión uniforme/regularizada	80%	75%	-

Matriz de confusión (extracción 1, versión uniforme):

		Clase predicha	
		Gatos	Perros
Clase real	Gatos	14	6
	Perros	2	18

Así pues, vemos que Naive Bayes obtiene resultados ligeramente peores que los algoritmos anteriores. ¿A qué se puede deber esto? Se puede achacar a diversos motivos. En primer lugar, recordemos que la parte *naive* del nombre del modelo es porque este asume que todas las características de nuestros datos son independientes, y en el caso de las imágenes y más aun teniendo en cuenta como hemos calculado los vectores característicos, esto no es cierto en lo absoluto. La dependencia de un píxel con el siguiente es clara. Existen otros motivos como la flexibilidad del modelo que podrían explicar estos peores resultados, pero no nos vamos a centrar en ellos.

Modelo 4: Red Neuronal

Hasta ahora no hemos sido capaces de mejorar los resultados obtenidos por la regresión lineal. Vamos por tanto a probar una estrategia distinta y entrenar una pequeña red neuronal. A la hora de escoger todos los distintos hiperparámetros que la componen, hemos usado la ya mencionada *GridSearhCV* para calcular los mejores.

Resultados de precisión:

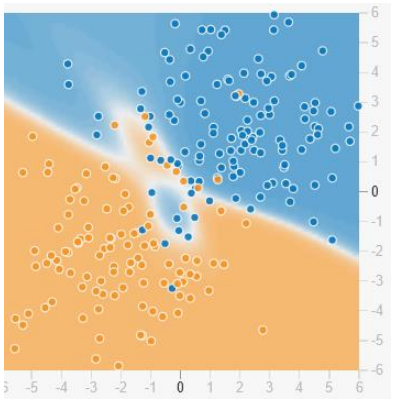
	Modelo extracción 1	*Modelo extracción 2*	*PCA*
Versión estándar	82.5%	87.5%	70%
Versión a color	82.5%	87.5%	-
Versión uniforme/regularizada	87.5%	87.5%	-

Matriz de confusión (extracción 1, versión uniforme):

		Clase predicha	
		Gatos	Perros
Clase real	Gatos	15	5
	Perros	0	20

Ahora sí, hemos superado los resultados obtenidos por la regresión lineal, en este caso, llegando casi al 90% de precisión. ¿A qué se debe esto? Como siempre, no hay una única respuesta. La opción más probable, sin embargo, es que este modelo se trate de una opción mejor para nuestro problema recurrente de imágenes conflictivas. Recordemos que, usando *GridSearchCV* tratamos de no sobreentrenar a nuestros modelos. Cabe suponer, por tanto, que estas imágenes eran *outlaiers* del conjunto al que deberían pertenecer. Así pues, ya que las redes neuronales son capaces de generar estructuras mucho más complejas que otros modelos para agrupar a todos los ejemplos en una misma clase, suponemos que estas estructuras más complejas han permitido que algunas de las imágenes conflictivas (una como mínimo) sean ahora parte del conjunto correspondiente.

Esta idea de los *outlaiers* y de las "estructuras complejas" que hemos mencionado son fácilmente observables en la web https://playground.tensorflow.org/.



En este caso observamos como los valores que no se encuentran en la zona que les correspondería son igualmente "atrapados" dentro de su zona por la propia estructura de la red neuronal, lo que le permite predecirlos bien pese a su complejidad.

Para conseguir esta imagen, se han introducido los siguientes parámetros: Distribución de datos gaussiana con 50 de ruido, dos características, dos capas ocultas con 8 y 4 neuronas respectivamente y dos neuronas de salida.

Modelo 5: SVMs

Seguimos con otros modelos que hemos estudiado. En este caso, los SVMs o *support vector machines*. Se dice que los SVM son uno de los mejores modelos de predicción que existen. La idea detrás de estos es buscar la frontera de decisión de margen máximo con los ejemplos que tenemos. No obstante, aunque no podemos representar gráficamente nuestros datos, está claro que estos no son linealmente separables, con lo que tendremos que utilizar alguna función *kernel* (funciones que permiten la clasificación no lineal). De nuevo, tanto el *kernel* como los diversos hiperparámetros del modelo son seleccionados automáticamente.

Resultados de precisión:

	Modelo extracción 1	*Modelo extracción 2*	*PCA*
Versión estándar	82.5%	82.5%	67.5%
Versión a color	82.5%	82.5%	-
Versión uniforme/regularizada	82.5%	82.5%	-

Matriz de confusión (extracción 1, versión uniforme):

		Clase predicha	
		Gatos	Perros
Clase real	Gatos	13	7
	Perros	0	20

Por desgracia, hemos vuelto a los resultados de la regresión lineal, e incluso los hemos empeorado ligeramente. El porqué de esto ya lo hemos comentado antes, las imágenes conflictivas vuelven a hacer acto de presencia en el clasificador. Si se observa el código, se verá que en la mayoría de las modelos imprime al final cuales han sido estas imágenes conflictivas y, si bien son más o menos comunes, no son siempre las mismas como habíamos

supuesto en un inicio. En cualquier caso, ya hemos dicho antes que el problema con las imágenes conflictivas lo trataríamos al final, así que de momento seguiremos entrenando más modelos.

Modelo 6: Ensembles

Nuestra estrategia hasta el momento ha sido usar un único clasificador cada vez para obtener los resultados. ¿Y si en su lugar utilizásemos varios simultáneamente? Para responder a esta pregunta vamos a utilizar los Ensembles.

Ensemble 1: Adaboost con decision stumps

En primer lugar, usaremos un modelo de Adaboost (o boosting). Este a su vez utilizará *decision stumps* o árboles de un solo nivel como clasificador base. Para todos los ensembles en general, el hiperparámetro por excelencia es el número de estimadores que vamos a utilizar, es decir, cuantas veces vamos a repetir la ejecución para encontrar el resultado óptimo. Este se calcula, de nuevo, con *GridSearchCV*.

Resultados de precisión:

	Modelo extracción 1	*Modelo extracción 2*	*PCA*
Versión estándar	65%	73%	70%
Versión a color	65%	73%	-
Versión uniforme/regularizada	93%	78%	-

Matriz de confusión (extracción 1, versión uniforme):

		Clase predicha	
		Gatos	Perros
Clase real	Gatos	17	3
	Perros	0	20

Finalmente superamos la barrera del 90%, en este caso prediciendo bien 37 de las 40 imágenes. No obstante, lo realmente interesante de este modelo no es su resultado (a fin de cuentas, la gran desventaja de los ensembles es que no son interpretables), si no la diferencia, hasta ahora insólita, del resultado con uno u otro modelo de extracción e incluso entre las versiones de este. El boosting se basa en la idea de dar más peso a aquellos ejemplos que se predicen mal en las repeticiones anteriores. Esto, teniendo en cuenta nuestro problema con las imágenes conflictivas, suena ideal, y ciertamente es con este modelo con el que se obtienen mejores resultados de todos los que hemos probado. Sin embargo, también falla estrepitosamente cuando estos datos no se han regularizado, de lo que podemos inferir que adaboost no trabaja especialmente bien con vectores de características muy largos. Mencionar por curiosidad, que las tres imágenes que fallamos pertenecen a las conflictivas, por supuesto, siendo estas tres las **únicas constantes** en todos los clasificadores. Estas imágenes son:



Y de nuevo sorprenden por no tener nada que llame la atención, ningún humano tendría problema en reconocer estas imágenes como gatos.

Ensemble 2: Bagging con decision stumps

Ahora cambiamos el ensemble principal por el de bagging. La idea de este es coger de una "bolsa" con los ejemplos de entrenamiento, una cantidad aleatoria de datos para entrenar. Después, se usará algún método de voto para decidir a qué clase pertenece cada ejemplo.

Resultados de precisión:

	Modelo extracción 1	*Modelo extracción 2*	*PCA*
Versión estándar	70%	78%	65%
Versión a color	70%	73%	-
Versión uniforme/regularizada	78%	65%	-

Matriz de confusión (extracción 1, versión uniforme):

		Clase predicha	
		Gatos	Perros
Clase real	Gatos	12	8
	Perros	1	19

Una vez más, es difícil interpretar los resultados de un ensemble. En este caso vemos un decrecimiento en la precisión considerable, llegando a algunos de los valores más bajos de todos los modelos. Recordemos que bagging es un modelo que solo puede resolver problemas de varianza alta, es decir, de modelos que se ajustan demasiado. Sin embargo, y en vista de nuestras imágenes conflictivas, podemos decir que, si acaso, nuestro problema está en un bias alto, en la falta de ajuste de los modelos a los datos, con lo que es razonable que bagging nos devuelva peores resultados.

One vs All

Por último, vamos a implementar el modelo One Vs All con un support vector machine como clasificador base. Esto lo hacemos ya que se trata de un modelo muy simple y cómodo de usar, así que nos puede servir para analizar algo más nuestros datos.

Resultados de precisión:

	Modelo extracción 1	*Modelo extracción 2*	*PCA*
Versión estándar	77.5%	82.5%	75%
Versión a color	77.5%	80%	-
Versión uniforme/regularizada	80%	80%	-

One vs All divide el problema multi-clase en tantos problemas como clases tengamos. Teniendo solo dos, utilizar este método es casi de Perogrullo, pero en cualquier caso nos sirve para observar que, al menos en este caso particular, la versión estándar del segundo modelo de extracción es la que arroja mejores resultados.

Modelo 7: Kmeans

Para terminar, vamos a implementar Kmeans. Al igual que cuando implementamos la regresión lineal, aquí, otra pregunta salta a la vista ¿por qué utilizar un modelo propio del aprendizaje no supervisado en un problema que sí lo es? Y de nuevo, la respuesta es la misma, para obtener más información sobre nuestros datos. Kmeans

funciona, como su nombre indica, con las medias, algo que hasta ahora no habíamos tenido en cuenta tan explícitamente. ¿Funcionará Kmeans? Analicemos los resultados.

Resultados de precisión:

	Modelo extracción 1	*Modelo extracción 2*	*PCA*
Versión estándar	57.5%	60%	70%
Versión a color	57.5%	60%	-
Versión uniforme/regularizada	55%	60%	-

Una vez más, podemos sacar varias conclusiones. Primero, lo más destacable de todo, por primera vez PCA es quien obtiene los mejores resultados de todos. Es difícil analizar a qué se puede deber este comportamiento, aunque se lo podemos achacar el propio funcionamiento de Kmeans. Como decíamos antes, este se fundamenta en el cálculo de medias, y de ir agrupando los distintos ejemplos según a qué media se parezcan más. Presumiblemente, al reducir muchísimo la dimensionalidad con PCA (que para calcular sus componentes también utiliza, en parte, medias), creamos vectores característicos que se ven mucho más beneficiados de un algoritmo así mientras que, con los modelos originales, la media no aporta información, pues una imagen no se distingue simplemente por el valor de sus intensidades (un perro no se distingue de un gato por su color de pelo), sino, más importante, por la distribución espacial de los píxeles que la componen. Esto hace que para estos casos Kmeans produzca resultados que están poco por encima del resultado trivial.

Conclusiones

Ya hemos hablado de la precisión de todos nuestros modelos, y tras analizar brevemente los resultados en cada uno de ello hemos concluido lo siguiente:

- El mejor modelo para la extracción de características es, por lo general, el primero, basado en los vecinos y los número binarios.
- De entre las versiones del primer modelo, ofrece mejores resultados en general la versión uniforme, y en los casos donde no son mejores resultados sí resultan considerablemente más rápidos.
- De entre todos los modelos de clasificación que hemos entrenado, el adaboost con decision stumps es el que mejores resultados obtiene para al menos uno de los modelos de extracción, mientras que, de media, es la red neuronal el mejor modelo.
- El PCA ha demostrado ser un modelo menos preciso en todos los modelos salvo en Kmeans, pero en todos los casos las ejecuciones que se han realizado con este modelo han sido mucho más rápidas. No obstante, PCA es una herramienta utilísima, pues reduciendo la dimensionalidad de los vectores en más de un 99% en algunos casos, solo pierde en torno al 10% de precisión.
- Las versiones a color de ambos modelos de extracción devuelven resultados similares a las en blanco y negro, casi siempre los mismos. Por tanto, en el futuro no se recomienda usar versiones a color, pues su ejecución es mucho más lenta.

Estas son las conclusiones generales del problema, pero todavía queda un tema en el tintero desde casi el inicio de la práctica. ¿Y qué pasa con las **imágenes conflictivas**? Para hacer un estudio más profundo, hemos decidido repetir los mejores algoritmos en este caso con el dataset de 500 imágenes.

Con una red neuronal obtenemos una precisión del 75%, y la siguiente matriz de confusión:

		Clase predicha	
		Gatos	Perros
Clase real	Gatos	69	31
	Perros	20	80

Con adaboost obtenemos una precisión del 60%, y la siguiente matriz de confusión:

		Clase predicha	
		Gatos	Perros
Clase real	Gatos	82	18
	Perros	62	38

¿Qué podemos decir de esto? Como podemos ver, en cada caso las imágenes que se fallan son distintas. Más notablemente, con adaboost fallamos más la predicción de los perros que la de los gatos. Siguen estando presentes algunas de las imágenes conflictivas de todas las veces anteriores, pero, en general, parece que no hay nada específico en ellas, más allá de que los propios modelos puedan predecir mejor o peor las imágenes. A fin de cuentas, los propios datasets dan pie a que estás situaciones se produzcan. A continuación, mostramos una serie de ejemplos de imágenes que pueden haber confundido al entrenamiento de nuestros modelos:

Todas estas fotos están en la carpeta de perros, pero vemos que, dadas sus características, pueden no ser adecuadas para nuestros entrenamientos.

Como último apunte, decir que la predicción es muy variable y que cada uno de los predictores es un mundo. No conviene decidir categóricamente cual puede ser el mejor o peor predictor. Así pues, hemos aprendido a clasificar imágenes de perros y gatos con un éxito moderadamente alto.