02807: Week 10 - Feature hashing and LSH

Code with output (use this when reading report):
https://gist.github.com/anonymous/9dd9c4c42eec26848a646e54bc930ba6
Scripts: https://gist.github.com/anonymous/b1d085292d7e71dbf7e48e7aae46d99c

Exercise 10.1 - Feature Hashing:

Here we should find both a bag-of-words representation and a hashed version of this representation for the articles to show we can shrink our feature space without loosing information.

The overall idea of the script is:

- Load in all the articles using **glob** and **json**, append all valid articles to the list **cleaned_data**, confirm the length of this as 10377
- Make a dictionary of the unique words in all articles combined, for this is use the **set-**type in python that automatically only holds unique elements. The number of unique words (without removing punctuation) is 70793.
- To make fast lookups, convert the set into a dictionary, where each key is a word that has a value which corresponds to an index. This will represent the columns in the bag-of-words.
- Construct the bag-of-words by simply looping through all the words in an article, looking up the index in the dictionary and increment this index in the bag-of-words matrix for this specific article. Asserting that the dimensions is truly
  (articles, feautres) = (10377, 70793)
- Now we construct a **X** (the bag-of-words representation, each row representing one observation) and a **y,** which is simply 1 if the article contains the subject **'earn**' else 0. Thus we have a case of binary classification. Use sklearn build in functionality to split the data into a test and train set
- Use a random forest classifier with 50 estimators and finally use the **score** to predict performance on the test and train-set.

```
#Test performance
print("Accuracy on train-set =", clf.score(X_train, y_train))
print("Accuracy on test-set =", clf.score(X_test, y_test))
```

```
Accuracy on train-set = 1.0
Accuracy on test-set = 0.946531791908
```

- Now using feature hashing, I define that I want to reduce the feature space to 1000 dimensions. We simply just take the modulus 1000 to the index, so if a word has index 1100, it will be put in bin 100 instead of 1100.
- Assert that the dimensions of the hashed articles are indeed (10377 x 1000)
- Construct a new random tree classifier in the same manner as before and test the performance.

```
#Test performance
print("Accuracy on train-set =", clf.score(X_train, y_train))
print("Accuracy on test-set =", clf.score(X_test, y_test))
```

```
Accuracy on train-set = 1.0
Accuracy on test-set = 0.946531791908
```

- From the score on the test-set, we do indeed confirm, that we have been able to reduce the dimensionality of the data greatly without loosing accuracy of our classifier.

Exercise 10.2 - LSH:

For differencing the images, I simply import them, convert to grayscale and compare a shifted edition of the image to itself using the **">"** operator with numpy. This is the most efficient way to do this compared to doing it in a for-loop.

The overall idea with the rest of the script is:

- Define a height and width, which will determine how much we resize the images and thus how many features we will have in the end.
- For every **.jpeg**-image in the folder:
    - Open the image and convert it to grayscale using the **convert('L')**-function from the Pillow module
    - Do the differencing as explain above
    - Using Davids script found on Aula, convert the row to a hex-value depending on how many **Trues** and their position that is presented in the row.
    - Append the hex-value to the string **hex_string**
    - For the images getting reshaped to 9x8, we will end up with 8 rows. Since each **hex**_value is defined over two characters, we will end up with strings with the length of 16 characters:

```
banana.jpeg: 0000212139331e07
banana2.jpeg: 808141312123260c
orange1.jpeg: 000e2d0d4f0f060e
orange2.jpeg: 0613160747070e0c
```

This is the end of the required exercise, but for the sake of curiosity i also tried to find the cosine distance between images. For two images of bananas and oranges, each just being on a white background, I get the following correlation matrix for the resize of (9x8):

	banana.jpeg	banana2.jpeg	orange1.jpeg	orange2.jpeg
banana.jpeg	1.000000	0.437564	0.825922	0.776200
banana2.jpeg	0.437564	1.000000	0.447148	0.496058
orange1.jpeg	0.825922	0.447148	1.000000	0.961906
orange2.jpeg	0.776200	0.496058	0.961906	1.000000

This is not great, notice how the banana.jpeg is more similar to an orange than the other banana.

Increasing the resizing to (25x24) gives us more features and it is seen, that this help a lot in clustering the images:

	banana.jpeg	banana2.jpeg	orange1.jpeg	orange2.jpeg
banana.jpeg	1.000000	0.874059	0.517271	0.494726
banana2.jpeg	0.874059	1.000000	0.518566	0.457681
orange1.jpeg	0.517271	0.518566	1.000000	0.901867
orange2.jpeg	0.494726	0.457681	0.901867	1.000000