CS584 – MACHINE LEARNING

FALL 2016

Image Classifier

Group Members: Daksh Gupta

Table of Contents

Image Classifier

Group Members: Daksh Gupta

Task

The task is to classify images as ‘spam’ or ‘notspam’. This is a common problem faced by many people using apps like WhatsApp, Snapchat etc. Garbage images such as memes, text images, etc., fill up your memory. So if there was something that could help you automatically separate the useless from the useful images then that would help solve this problem.

In this project I am trying to form a classifier that can predict the class and image is most likely to be a member of.

Dataset

My dataset is made up of real world images.

Data source

I used images from my personal clicks and WhatsApp folder for real world examples and also several memes downloaded from Google Images. I labeled all the 3,426 images manually as either spam or notspam.

Target variable

The target variable is the label of the image. It can either be labeled as ‘spam’ or ‘notspam’. Below are a few examples for each class

NOTSPAM:




SPAM

Features

I have used 13 features to represent the images, they are as follows:

- Number of faces in the image
- Number of distinct color in the image
- Percentage of area of image covered with text
- Percentage of area of image covered by top color 1
- Percentage of area of image covered by top color 2
- Percentage of area of image covered by top color 3
- Percentage of area of image covered by top color 4
- Percentage of area of image covered by top color 5
- Percentage of area of image covered by top color 6
- Percentage of area of image covered by top color 7
- Percentage of area of image covered by top color 8
- Percentage of area of image covered by top color 9
- Percentage of area of image covered by top color 10

Here, ‘top color 1’ means the color that has the maximum number of pixels covered in the image.

Data size

There are 3,426 instances in the dataset, 1 for each image.

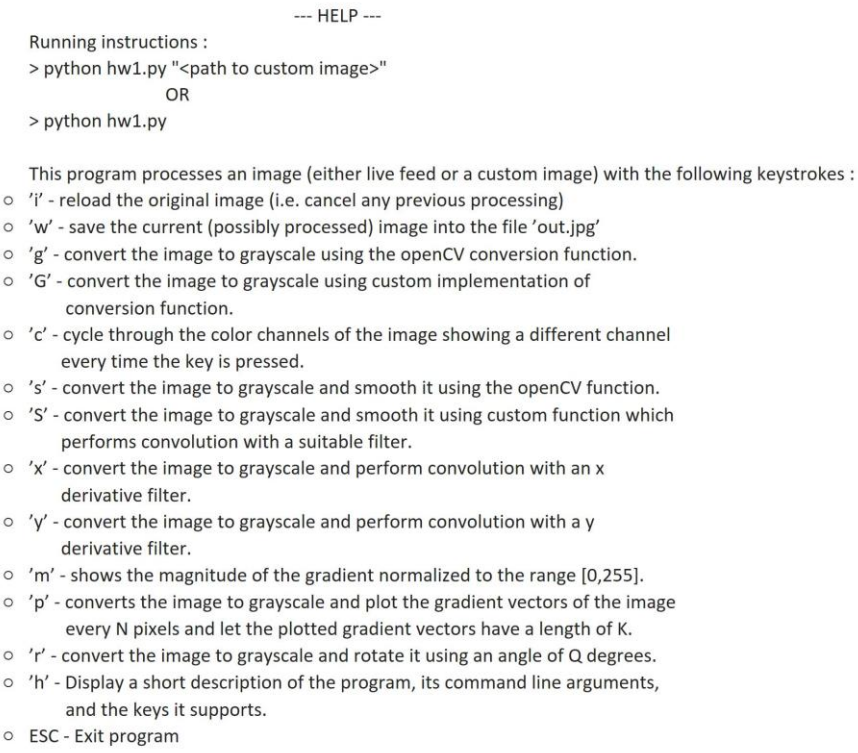
Preprocessing

I extracted the features from every image using OpenCV 3.1.0.

For finding the number of faces in an image, I used the CascadeClassifier (opencv) to detect how many objects in the image matched the haar cascade for the default frontal face.

To perform a perfect text detection/recognition, libraries like Google Tesseract, that perform Optical Character Recognition (OCR), are required. And even then they are language dependent. So, to avoid the language barrier, I used edge detection to get an approximation of the area covered with text. Since edges can be produced in the image from objects other than text, this technique sometimes produces erroneous results.

The number of distinct colors cannot be found out simply by using the histogram of the image. This is because of the presence of noise in the image. For example:



This image has just 2 colors right now, but if I take a screenshot of this and try to find out all the shades of gray in that, then it has almost all the 256 shades.

To fix this I set a threshold on the percentage of number of pixels filled with a particular shade. I set that threshold to 0.5%.

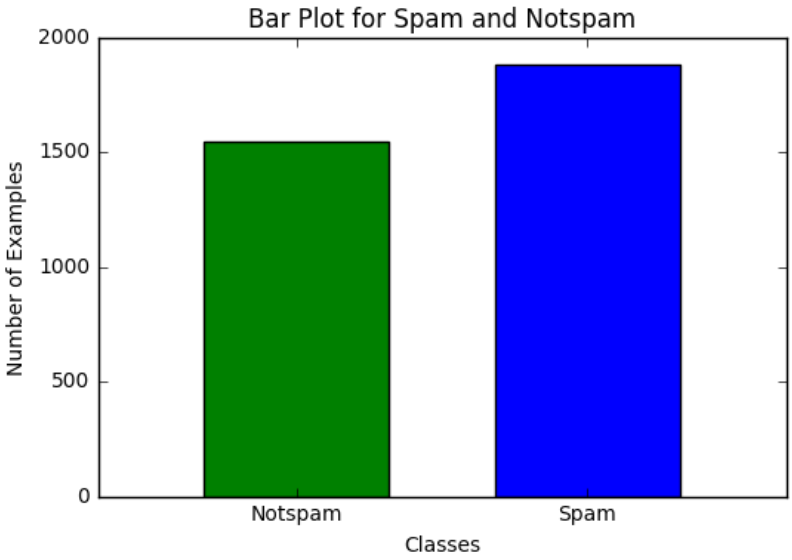
I used the percentage of number of pixels filled with a particular shade to find out the top colors.

The preprocessing results can be loaded from the files (using pickle) 'data.pckl', which contains a numpy matrix of 3,426 rows and 13 columns, and 'labels.pckl', which contains a numpy array with 3,426 rows.

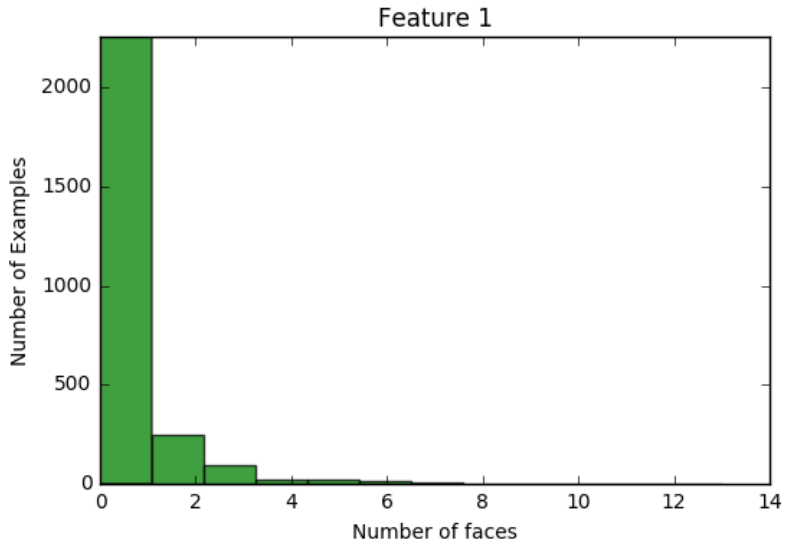
Visualization

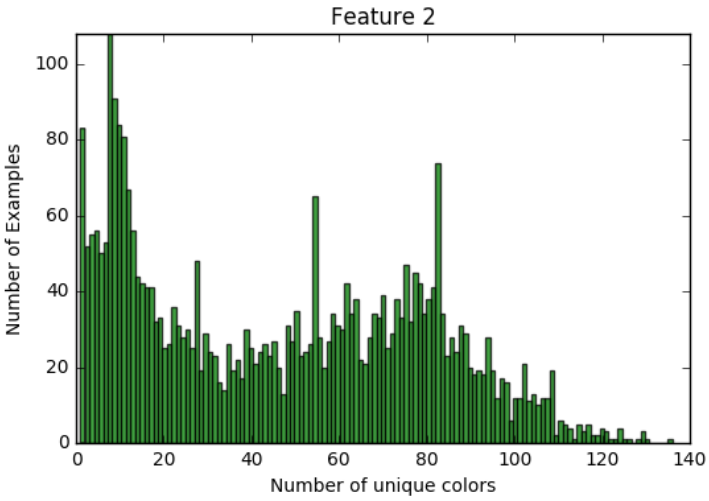
Target

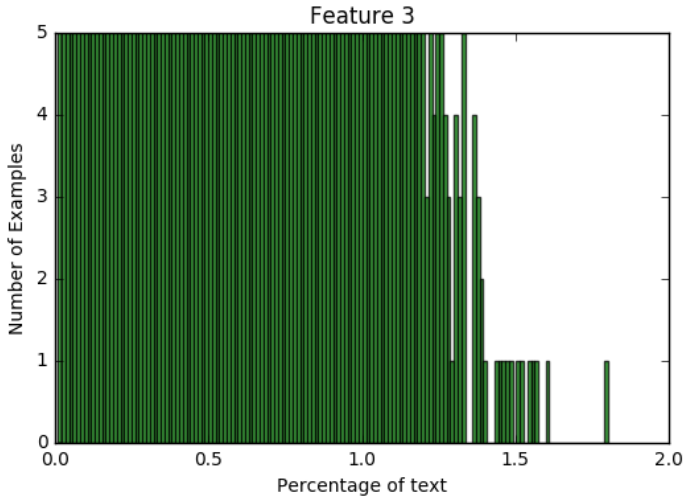
Class ‘notspam’ has 1,548 instances and class ‘spam’ has 1,878 instances.

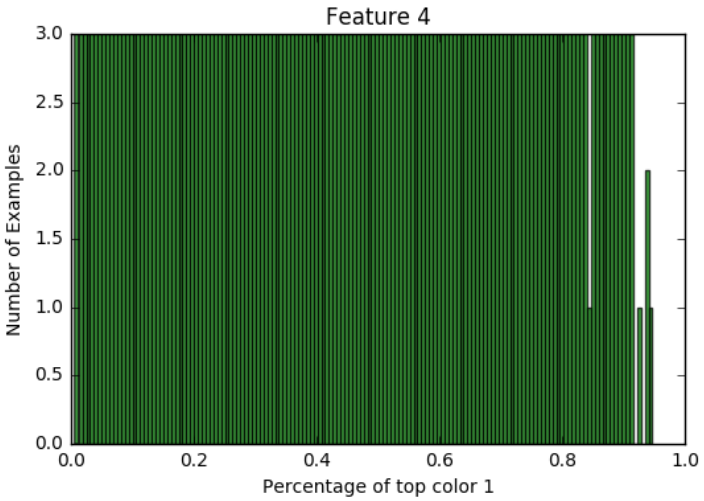


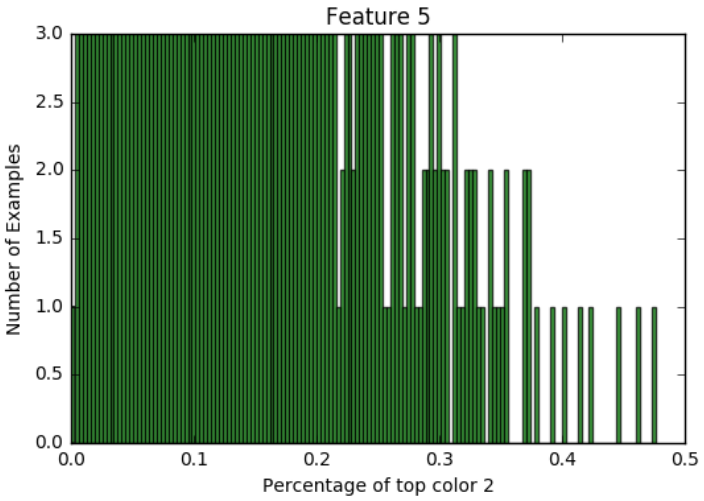
Features

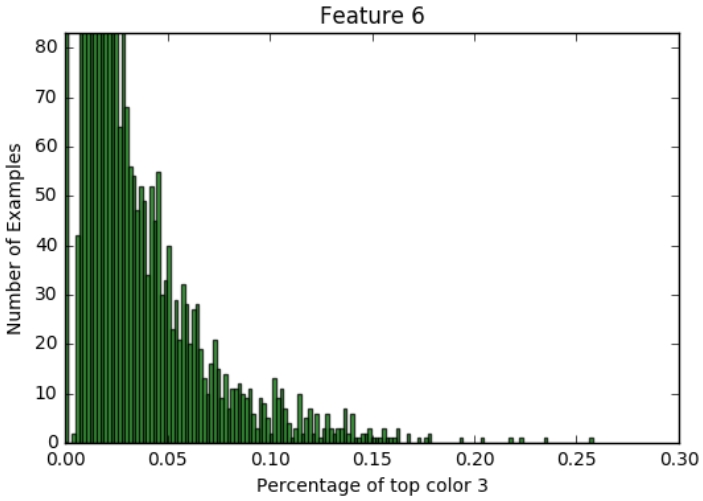


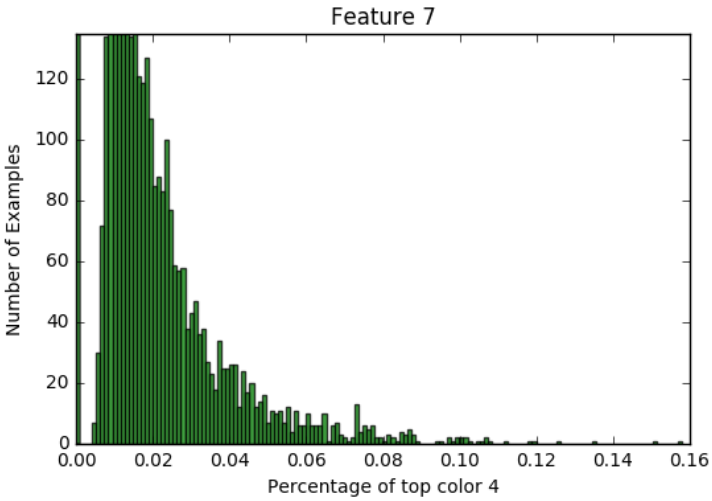

Feature 2
Number of Examples
Number of unique colors
0
20
40
60
80
100
120
140

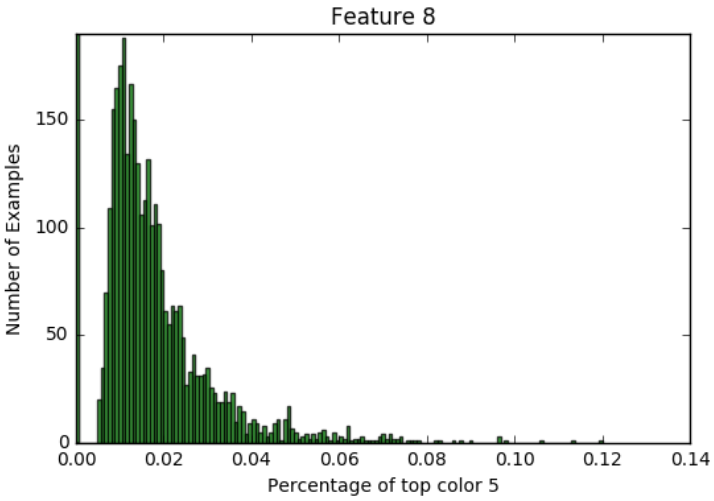

Feature 3
Number of Examples
Percentage of text
0.0
0.5
1.0
1.5
2.0

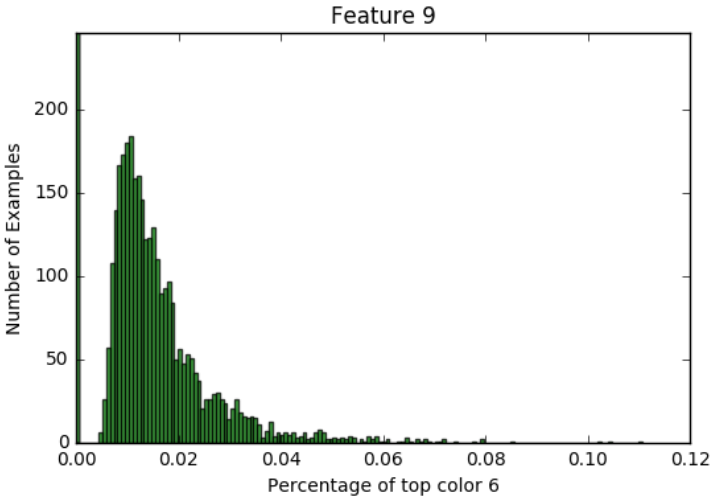

Feature 4
Number of Examples
Percentage of top color 1
0.0
0.2
0.4
0.6
0.8
1.0

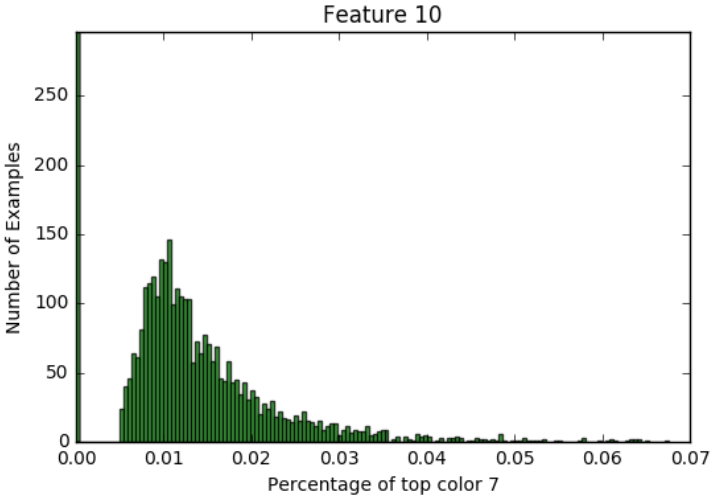

Feature 5
Number of Examples
Percentage of top color 2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0.1
0.2
0.3
0.4
0.5


Feature 6
Number of Examples
Percentage of top color 3
0
10
20
30
40
50
60
70
80
0.00
0.05
0.10
0.15
0.20
0.25
0.30


Feature 7
Number of Examples
Percentage of top color 4
0
20
40
60
80
100
120
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16


Feature 8
Number of Examples
0
50
100
150
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
Percentage of top color 5


Feature 9
Number of Examples
0
50
100
150
200
0.00
0.02
0.04
0.06
0.08
0.10
0.12
Percentage of top color 6


Feature 10
Number of Examples
0
50
100
150
200
250
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
Percentage of top color 7

Evaluation

Performance Measure

I chose F1 score as the performance measure. F1 score is the harmonic mean of precision and recall, so maximizing this means that the number of true positives for a binary target variable is more reliable than maximizing precision or recall alone.

$$F_1 = 2.\frac{precision.recall}{precision + recall}$$

Classifiers

I tried a coarse comparison between 5 classifiers:

- DecisionTreeClassifier with default parameters
- BernoulliNB with default parameters
- SVC (Support Vector Machine) with 100 different values for C parameter (0.01 to 1.00)
- LogisticRegression with 100 different values for C parameter (0.01 to 1.00)
- MLP (Multilayer Perceptron) with 100 different values for alpha parameter, adaptive learning rate, and no shuffling (0.01 to 1.00).

I used these classifiers because they are supposed to perform very good for binary classification. I did not change the parameters for DecisionTreeClassifier and BeroulliNB because they require a prior belief as to which class is an image most likely to be in, which I don't have. I changed the C/alpha for the other 3 classifiers because these parameters determine how big of penalty is going to be applied to bigger weights (regularization). 'C' parameter for LogisticRegression and SVC means the smaller the value the bigger the penalty. 'alpha' for MLP parameter means the bigger the value, the bigger the penalty.

I also tried the more parameter tweaking for MLP and I further tried different values for the following parameters:

- random_state: 50 different values (0-49)
- activation: 4 possible values (relu, tanh, identity, logistic)
- solver: 3 different values (lbfgs, adam, sgd)
- hiddel_layer_sizes: 4 different values ((100,), (50,50,), (33,33,33,), (25,25,25,25,))

Evaluation Strategy

I used 100 random test train splits with 67% training data and 33% testing data to perform a coarse model selection using 10-fold cross validation on 5 classifiers with either default parameters or very less tweaking. The total number of combinations was 30,200. It took more than 12 hours to perform this comparison. (Results are stored in 'modelparams.pckl', contains a list of tuples in the format (F1 score, random_state, string identifier for classifier, C/alpha value))

The purpose of this comparison was to find the classifier that could perform well with little tweaking in in a particular train test split state.

After this I would further tweak the parameters of the classifier that won in the coarse comparison for that state. In my case MLP was that classifier and I tweaked 4 more parameters, and tried 2,400

combinations. This this comparison took about 5 hours. (Results are stored in 'MLPparams.pckl', contains a list of tuples in the format (F1 score, random_state, activation, solver, hidden_layer_sizes))

Performance Results

Top results for the each classifier in the coarse model selection

Model	Parameters	train_test_split random state	Performance
Baseline	Majority class	-	0.7081
	Random class	-	0.5300
Logistic regression	C=0.03	97	0.8505
	C=0.05	97	0.8502
MLP	alpha=0.51	81	0.8518
	alpha=0.64	81	0.8515
SVC	C=0.03	42	0.8461
	C=0.03	97	0.8449
BernoulliNB	default	79	0.4606
	default	56	0.4589
DecisionTreeClassifier	default	0	0.8442
	default	18	0.8379

Top results for the fine model selection or further tweaking of MLP

random_state	activation	hidden_layer_sizes	solver	Performance
29	relu	(33,33,33)	lbfgs	0.8712
9	relu	(100)	lbfgs	0.8693
26	tanh	(50,50)	lbfgs	0.8689
45	tanh	(100)	lbfgs	0.8689
30	relu	(25,25,25,25)	lbfgs	0.8656
40	logistic	(33,33,33)	lbfgs	0.8571
25	tanh	(25,25,25,25)	adam	0.8546
48	relu	(50,50)	adam	0.8542
39	identity	(33,33,33)	lbfgs	0.8533
41	identity	(100)	adam	0.8533
7	tanh	(25,25,25,25)	sgd	0.8523

Top Features

Since my model was a neural network, I cannot judge what the top features according to my model are without indulging into a lot of mathematics. I can on the other hand tell that my model has 3 hidden layers, with 33 neurons in each layer with the activation function as relu (REctified Linear Unit).

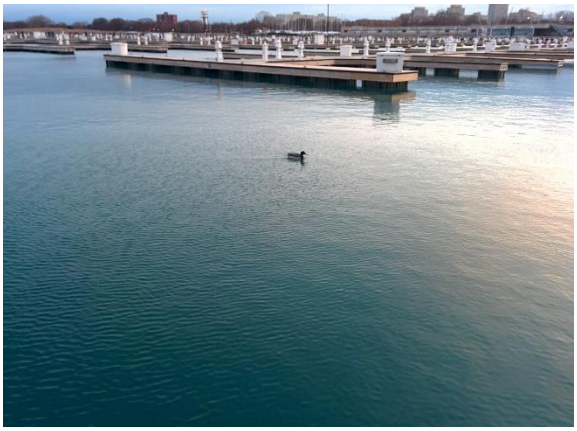
Discussion

The F1 score for the testing data was 0.8402 with 88.75% precision and 79.77% recall. Even though the F1 score for pretty satisfying for my classifier, the classification in itself was not what I wanted. My classifier was correctly classifying all the spam images but it was incorrectly classifying notspam images.

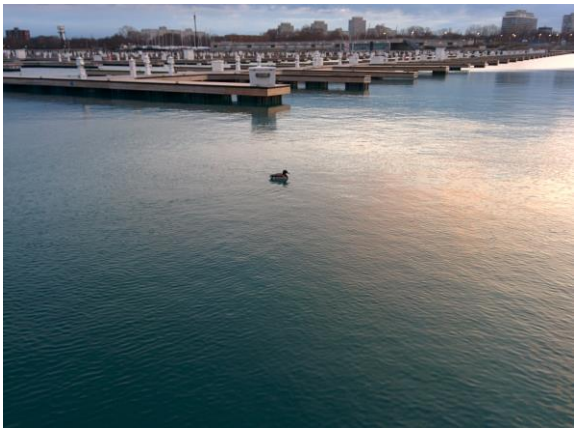
Out of 1,131 testing instances, there were 184 mismatches and all of them were 'notspam' or important images which were classified as 'spam'. The number of 'spam' instances in the testing data was 613 and that of 'notspam' instances was 518. Getting a few spam images in the important images folder is fine, but losing important images to the spam folder in a way defeats the purpose of this classification.

Interesting/Unexpected Results

These cases are interesting because all of them have very similar training examples



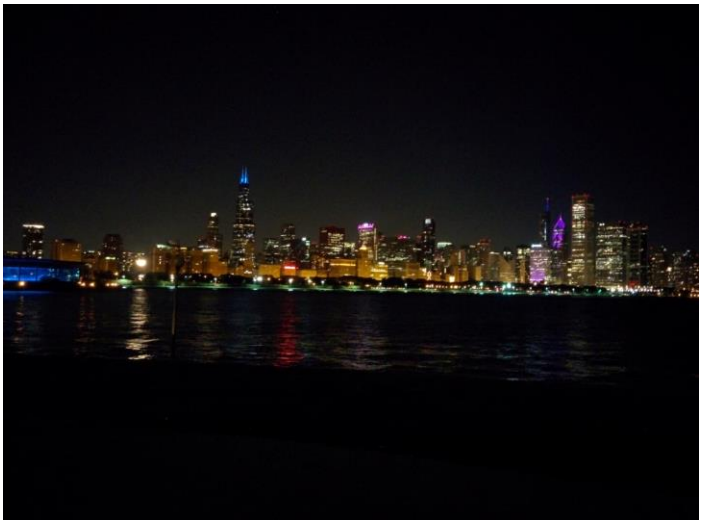
This image was labelled as 'spam'. This is an image from the 31st street beach.



I took this image a few minutes later and this image is present in the training data with the label 'notspam', but still the above was classified as 'spam'.

This image was classified as ‘spam’ even though it has no text, many shades of colors, maybe the face wasn’t detected, but this still could be treated as a landscape image.

This is an image of some protest in Chicago, classified as ‘spam’. This image is hard to classify by a classifier because it has text, there is probably one shade that has large area coverage, but I would classify it as ‘notspam’ because it has all the natural elements of a photograph. So it was a failure case for the classifier.



Tis image was classified as ‘spam’. I have provided many training images of downtown skyline at night. This should not have happened. But the reasons for misclassification are similar to the above image, a large are of this image is black, which could have been a deciding factor for the classification.

This image was also classified as ‘spam’. This is an important image in my point of view. This is a personal click, it has a clear frontal face, lots of different colors, but still it was misclassified as ‘spam’. Losing these types of images is a big problem.

Conclusion

My classifier can classify images without having to use advanced techniques like deep learning or convolutional neural networks. Improvement in performance can be achieved by:

- Making classifier language dependent, to perform OCR, instead of using edge detection
- Adding additional features to define what each top color is. This will provide not only the area coverage but also the information about the top color for spam and notspam images.
- Using area coverage for all 256 shades rather than using area coverage of only top 10 colors. This will help in defining what the characteristic colors for spam and notspam images are and what is the area coverage for each of them.

Without the proposed improvements above, the classifier is not yet ready to be used since, there is a danger of losing sensitive images.