Hüseyin Utku ASLAN
h.utku.aslan@hotmail.com

Human Detection-Final Report

26th January 2018

OVERVIEW

Purpose of this project is detecting pictures which has humans in it and hasn't. Name of the application is "bye cruel world". There are no particular reason for me choosing it.

STEPS

1. Extract features with Histogram of Gradients method.
2. Flatten the data.
3. Train classifier or create clusters.
4. Test and compare the results.

SPECIFICATIONS

To run this program user needs some dependencies installed on their machine. These are python3, python3-opencv, python3-numpy and python-sklearn.

Images should be stored in "Images" file. Inside that folder there are 2 others which represents classes; positives and negatives as "pos" and "neg" To run the program user should enter *"python main.py -e True -c SVM"* to terminal. Script works both with python 2 and 3. If user wants to run feature extraction before classification, user needs to enter *"-e True"*. If not, *"-e False"* should be entered. At the end of extraction, program saves the flattened data to a ".npy" file. After running it once, program will use saved data for training classifier. There are 2 classification and 1 clustering method. These are Support Vector Machine(SVM), neural network and k-means algorithms. To run SVM, user needs to enter *"-c SVM"*. To run neural network, user needs to enter *"-c MLP"*. To run k-means, user needs to enter *"-c kmeans"*.

Examples : *python main.py -e True -c SVM*

python main.py -e False -c MLP

python main.py -e True -c kmeans

PROJECT DESCRIPTION

Description of Project

This project is made for detecting images with humans in it. I used INRIA Person Dataset to train and test the program. There are positive and negative pictures. This dataset was collected as part of research work on detection of upright people in images and video. The research is described in detail in CVPR 2005 paper Histograms of Oriented Gradients for Human Detection. The dataset is divided in two formats: (a) original images with corresponding annotation files, and (b) positive images in normalized 64x128 pixel format (as used in the CVPR paper) with original negative images. Our purpose is to detect humans as many as we can.

Description of Solution[1]

I used Histograms of Oriented Gradients for Human Detection(HOG). HOG is a feature descriptor. A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information.

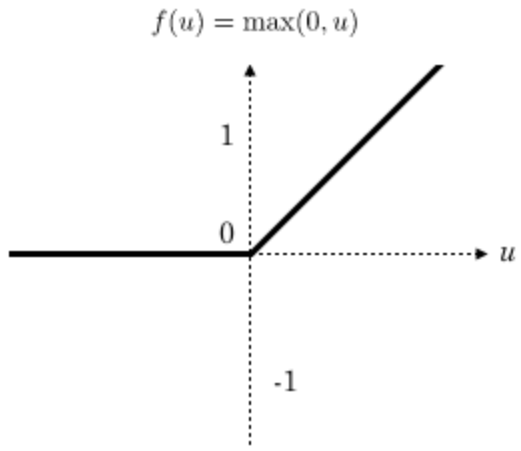
In the HOG feature descriptor, the distribution (histograms) of directions of gradients (oriented gradients) are used as features. Gradients (x and y derivatives) of an image are useful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes) and we know that edges and corners pack in a lot more information about object shape than flat regions.

For more details, please follow the link below.

For classification, I used Support Vector Machine(SVM), and a neural network. For clustering, I used k-means algorithm.

Scikit-learn's functions are used in the program for SVM. For testing, last "n" numbers of inputs are cut from training data and fed to prediction algorithm of SVM. It created a basis to compare classification method I wrote.

K-means algorithm is an unsupervised learning algorithm. It divide data to clusters according to a distance function. We start by choosing a k. For our example we picked 2 because we already know how many classes there are. Algorithm initialize k number of centroids and put them in random places. After that, for every node in training set, we assign them to closest centroid. Then we calculate the mean value of every point that is connected to same centroid. After that, we move centroid to that position. We repeat this process till centroids stop moving or reach a predefined number of iterations.



Neural network in this project uses multilayers, dropout method, Relu activation function and backpropagation. To increase generality in network, I used dropout method. While training the network, 1/k of random weights are closed(k ∈ (0, 1)). While prediction, we close this dropout. I choose 0.5 as k for this example. This way, user can get more generalized predictions. I picked Relu activation function. Because of needing few calculations, Relu is a great option for fast training.

Things Have Changed

Since the initial report many things have changed in project. Firstly, I added 2 more classification techniques. A multi layer neural network with dropout method for generalization and ReLU activator for classification purposes. Other method is k-means. I used it for clustering data. Even though we have tags for each training and testing data, I used k-means with k=2 to create a model to whether or not a unsupervised learning method will also work for these examples. Results we interesting.

HOW TO HOG WORKS? OVERVIEW OF METHOD[2]

The method is based on evaluating well-normalized local histograms of image gradient orientations in a dense grid. Similar features have seen increasing use over the past decade. The basic idea is that local object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions, even without precise knowledge of the corresponding gradient or edge positions. In practice this is implemented by dividing the image window into small spatial regions (“cells”), for each cell accumulating a local 1-D histogram of gradient directions or edge orientations over the pixels of the cell. The combined histogram entries from the representation. For better invariance to illumination, shadowing, etc., it is also useful to contrast-normalize the local responses before using them. This can be done by accumulating a measure of local histogram “energy” over somewhat larger spatial regions (“blocks”) and using the results to normalize all of the cells in the block. We will refer to the normalized descriptor blocks as Histogram of Oriented Gradient (HOG) descriptors. Tiling the detection window with a dense (in fact, overlapping) grid of HOG descriptors and using the combined feature vector in a conventional SVM based window classifier gives our human detection chain.

Figure 1. An overview of our feature extraction and object detection chain. The detector window is tiled with a grid of overlapping blocks in which Histogram of Oriented Gradient feature vectors are extracted. The combined vectors are fed to a linear SVM for object/non-object classification. The detection window is scanned across the image at all positions and scales, and conventional non-maximum suppression is run on the output pyramid to detect object instances, but this paper concentrates on the feature extraction process.

DESCRIPTION OF INPUT AND OUTPUT

Description of Input

Input images are in "./Images/pos" and "./Images/neg" folders. There are 2416 images for positives and 1218 negatives. For every single positive image there is also 180 degrees rotated one so actual number of positives is 1208.

If user enters *"-e True"* for running the program, program first starts HOG, extracts features, flattens it to a vector and saves it to a ".npy" file. After those runs training and testing. If user runs *"-e False"*, program will jump extraction steps,load data that is saved to ".npy" file and trains classifier. Feature extraction takes too much time. That is why we used this kind of distinction.

There are 3 options for classifier; SVM, k-means, neural network. To pick a classifier, user needs to enter 1 of 3 code words; "SVM" for support vector machine, "kmeans" for k-means clustering, and "MLP" for neural network. All of them gives different results.

Description of Output

Steps that are completed are printed to terminal. If program run with extraction option, it is printed the shape of extracted positives and negatives which represent number of examples and number of features in certain array. After extraction, program prints loading steps. Prints how many features are loaded, shape of training set and shape of testing set. After those information about classifier is printed and the error rate is calculated and printed. Error rate is division between number of errors made throughout testing and number of testing examples. At the end of program name of the program is printed.

Examples : *python main.py -e True -c SVM*

python main.py -e False -c MLP

python main.py -e True -c kmeans

Hard Examples

For testing purposes, I added files called “hard-examples”. In these files, there are only 1 picture. These are for testing images outside the dataset.

Input Examples

Positive samples :

Negative samples :

Sample Input and Output

python3 main.py -e True -c SVM

```
(cv) huseyin@huseyin-ABRA-A5-V1 ~/Desktop/OpenCV/HOG $ python3 main.py -e True -
c SVM

==========HOG==========


----------------------

        Extracting features

        HOG Positives
         (2416, 243)  positives.

        HOG Negatives
         (1218, 243)  negatives.
        Positives HOG vectors are saved
        Negatives HOG vectors are saved

----------------------


Loading features :
positives :  (2116, 243) (300, 243)
negatives :  (918, 243) (300, 243)

Training data :  (3034, 243)
Test data :  (600, 243)

 SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Error Rate :  12 / 600

Hard Example 0:  [ 1.]

Hard Example 1:  [ 2.]

Hard Example 2:  [ 1.]

Hard Example 3:  [ 2.]

======================

bye cruel world
```

python3 main.py -e False -c SVM

```
(cv) huseyin@huseyin-ABRA-A5-V1 ~/Desktop/OpenCV/HOG $ python3 main.py -e False
-c SVM

==========HOG==========

Loading features :
positives :  (2116, 243) (300, 243)
negatives :  (918, 243) (300, 243)

Training data :  (3034, 243)
Test data :  (600, 243)

 SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)

Error Rate :  12 / 600

Hard Example 0:  [ 1.]

Hard Example 1:  [ 2.]

Hard Example 2:  [ 1.]

Hard Example 3:  [ 2.]

======================

bye cruel world
```

python3 main.py -e False -c kmeans

```
(cv) huseyin@huseyin-ABRA-A5-V1 ~/Desktop/OpenCV/HOG $ python3 main.py -e False
-c kmeans

==========HOG==========

Loading features :
positives :  (2116, 243) (300, 243)
negatives :  (918, 243) (300, 243)

--------kMeans--------

-----Class1-----
Number of positives in cluster 0 :  267
Number of positives in cluster 1 : 33
-----Class2-----
Number of positives in cluster 0 :  16
Number of positives in cluster 1 : 284

Hard Example  0 :  2

Hard Example  1 :  1

Hard Example  2 :  2

Hard Example  3 :  1

======================

bye cruel world
```

python3 main.py -e False -c MLP

```
(cv) huseyin@huseyin-ABRA-A5-V1 ~/Desktop/OpenCV/HOG $ python3 main.py -e False
-c MLP

==========HOG==========

Loading features :
positives :  (2116, 243) (300, 243)
negatives :  (918, 243) (300, 243)

Training ...
Training ended

False negative :  10    False positive :  0

Hard Example  0 : positive

Hard Example  1 : negative

Hard Example  2 : positive

Hard Example  3 : positive

======================

bye cruel world
```

Hard Examples Results

I used a washing machine which has a funny human like state, my profile picture from facebook, my old profile picture from facebook, and a cute cat.

Hard Example 0 :

SVM	positive
kmeans	positive
MLP	positive

Reason of this image classified as a human might be picture having human like features.

Hard Example 1 :

SVM	negative
kmeans	negative
MLP	negative

To be honest, being classified as a non-human by my own software is a little bit ironic.

Hard Example 2 :

SVM	positive
kmeans	positive
MLP	positive

Due to positioning, results of this image is exactly what we want. This test proves that, I'm a human being.

Hard Example 3 :

SVM	negative
kmeans	negative
MLP	positive

From this data, we can assume that, algorithm might give different results for different images.

CLASSIFICATION PARAMETERS

SVM

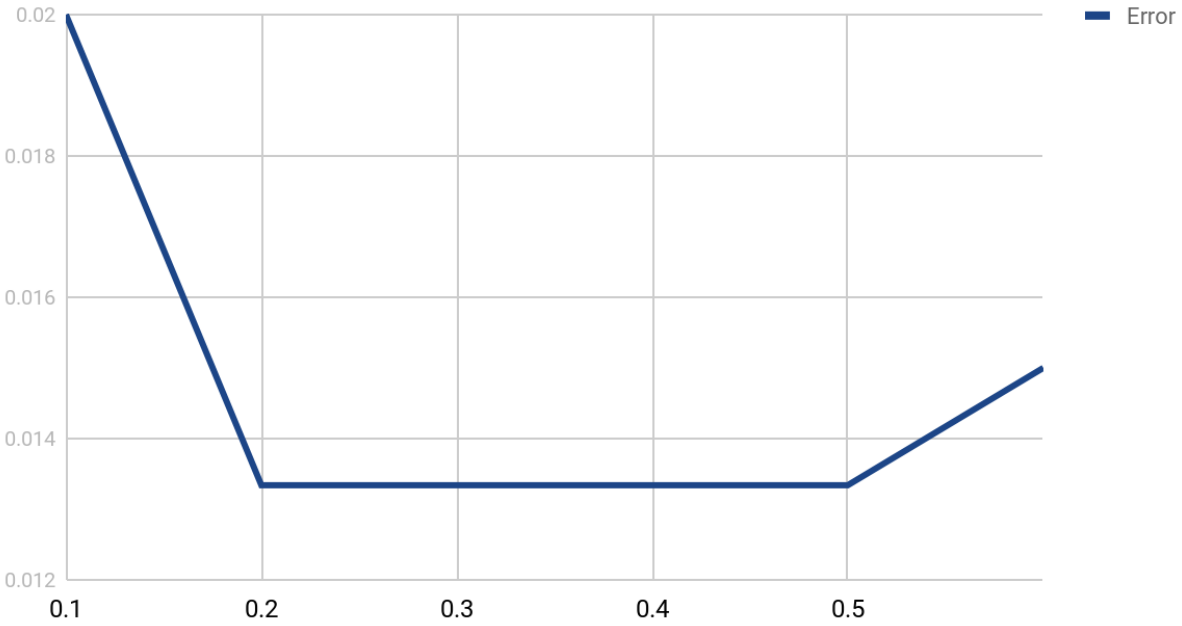
For SVM there are 3 different parameters we can change. These are C, gamma and kernel. C is penalty parameter of the error term. Kernel specifies the kernel type to be used in the algorithm. It must be one of ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable. If none is given, ‘rbf’ will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n_samples, n_samples). Gamma is kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. If gamma is ‘auto’ then 1/n_features will be used instead.

We test them 1 at a time. We only change 1 parameter and keep remaining same.

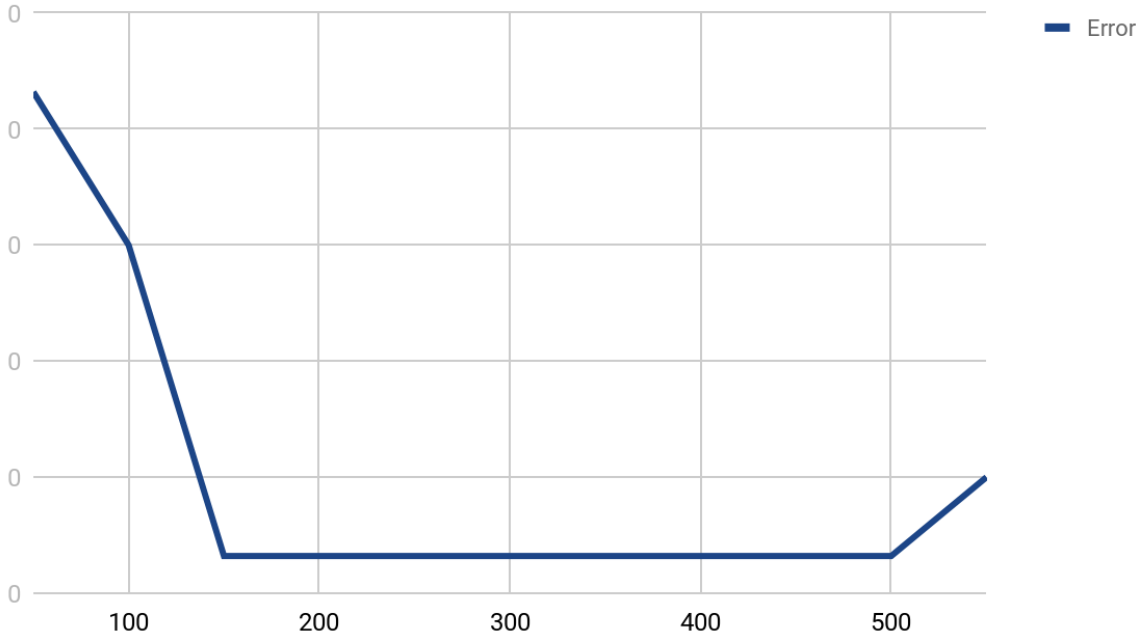
Default values are :

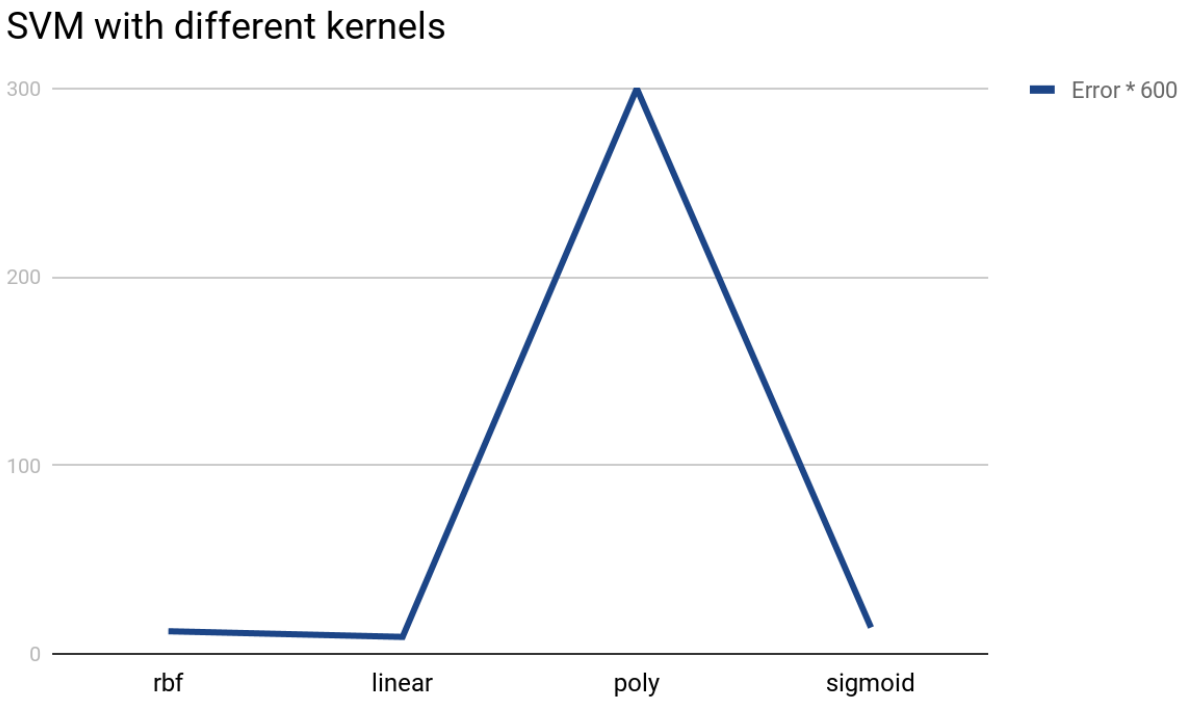
C	100
gamma	0.1
kernel	rbf

SVM gamma value changes



SVM C value changes





In individual tests; gamma rate between 0.2, 0.5, c value between 150, 500 gives the minimum error. For kernels, using linear kernel seems like giving the optimal solution. When we combine them all like gamma = 0.3, c = 300 and with linear kernel we decrease error rate up to 9/600 but we we test with hard examples, we can observe that all of them gives same result. From these results, we can suggest that, our optimal range causes **overfitting**.

K-means

In k-means, there are not many different parameters we can change. We can always change k, but for this example with 2 unique results, it will be unnecessary.

Neural Network

In neural network, parameters we can change are number of epochs, dropout rate, number of hidden layers, number of neurons in hidden layers. To keep analysis short, we assume there are same amount of neurons in each hidden layer. As we did in SVM, we keep default values same while changing other values.

Default values are :

epoch	375
Dropout rate	0.5
Number of hidden layers	2
Neuron in each hidden layer	400

To remind, hard example 0 and 3 should be negative while 2 and 3 should be positive

For number of layers :

N. layers	False pos.	False neg.	Hard exp 0	Hard exp 1	Hard exp 2	Hard exp 3
1	0/50	11/50	positive	negative	positive	positive
2	0/50	10/50	positive	negative	positive	positive
3	0/50	50/50	negative	negative	negative	negative

For number of epochs:

N. epochs	False pos.	False neg.	Hard exp 0	Hard exp 1	Hard exp 2	Hard exp 3
100	8/50	2/50	positive	negative	positive	positive
200	0/50	50/50	positive	positive	positive	positive
300	0/50	50/50	positive	positive	positive	positive
375	0/50	10/50	positive	negative	positive	positive
380	0/50	28/50	positive	positive	positive	positive
400	1/50	0/50	positive	error	error	error

For number of neurons in each hidden layer :

N. neuron	False pos.	False neg.	Hard exp 0	Hard exp 1	Hard exp 2	Hard exp 3
100	0/50	13/50	positive	negative	positive	negative
200	50/50	50/50	negative	negative	negative	negative
300	0/50	11/50	positive	negative	positive	negative
400	0/50	10/50	positive	negative	positive	negative
500	0/50	25/50	positive	negative	positive	negative

For dropout rate :

dropout	False pos.	False neg.	Hard exp 0	Hard exp 1	Hard exp 2	Hard exp 3
0.1	0/50	20/50	positive	negative	positive	negative
0.25	error	error	error	error	error	error
0.5	0/50	10/50	positive	negative	positive	positive
0.75	0/50	50/50	negative	negative	negative	negative

CONCLUSION

Because of distinct structure of training data, it is easy for any classification method to return an acceptable error rate and easy to create a 2 cluster with k-means algorithm. According to our tests, default values of the algorithms are close to optimal with this training data. In some examples error rate was very low but this time problem was overfitting. In this project, we proved that, HOG can be used to detect features from an image and create an acceptable training and testing data. But in some extreme cases, this is far from true. According to our hard examples, HOG with these classifiers, extreme cases causes problems to HOG feature extractor.

SOURCES

1. https://www.learnopencv.com/histogram-of-oriented-gradients/
2. Histograms of Oriented Gradients for Human Detection (Navneet Dalal and Bill Triggs) (INRIA Rhˆone-Alps, 655 avenue de l'Europe, Montbonnot 38334, France {Navneet.Dalal,Bill.Triggs}@inrialpes.fr, http://lear.inrialpes.fr)
3. Dropout: A Simple Way to Prevent Neural Networks from Overfitting (Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov) (Department of Computer Science
   University of Toronto
   10 Kings College Road, Rm 3302
   Toronto, Ontario, M5S 3G4, Canada)