Character Recognition using Artificial Neural Networks

Abstract – This report focuses on the development of a character recognition system based on existing artificial classification algorithms. There are three different implementations. The first recognizes printed letters and makes use of a Support Vector Machine. The second implementation compares two different algorithms and determines which one is more efficient for identifying handwritten characters. The third and final implementation is a system that recognizes and classifies signatures. Each of these implementations was written in "python" and tested extensively.

I. Introduction

Character recognition is a huge part of the machine learning world and is often referred to as an exciting but challenging field, which I soon found out when it came to the implementation side of this project. This research area falls under the machine learning category. If you are not aware of what machine learning is, it is an application of artificial intelligence (AI) that allows a system to learn and improve without being explicitly programmed[1]. It focuses on the development of computer programs which can access given sets on input data and learn from them. There are two different forms of machine learning algorithms, Supervised and Unsupervised which will be elaborated on in the research section of this report. The primary objects of this report are to "understand and develop a useful computer vision system," "extract appropriate features for recognition" and to "develop classification techniques and evaluate their performance. We will first explore the use of a Support Vector Machine (SVM) machine when it comes to the recognition "printed pages." We will then test it on "handwritten notes" and compare it to another classifier K-Nearest Neighbor (KNN). Finally, we will train a classifier to recognize different peoples signatures.

II. Literature Review

[5] In this paper the performance of SVM's and ANN are compared for handwritten characters. There is a discussion on how the slopes and slants of handwritten text significantly affect the performance of the program. This is somewhat appropriate when comparing it to my report as this is one of the leading issues I faced during implementation of the handwritten task. I felt this paper discussed the different classifiers well by giving advantages and distadvantages. However I do not not think it covers enough on the mathmetical side of the project.

[6] This paper discusses different concepts that are used from using SVM classifiers. It talks about the theoretical side of the SVM and how its algorithm works. It also gives a brief insight into the history of the algorithm itself. It has a thorough explanation of the maths side of the SVM algorithm which helped me understand how it worked. The other strategies it discusses also helped me when it came to the research side of the report. It also explains the use of this concept in the real world which is a more exciting side to the project.

[7] This paper focuses on an entirely different way of classifying characters. It compares the use of decision trees and neural network classifiers in recognition research. This paper explains the design system of the character recognition well. It goes into depth explaining things such as bounding boxes and centroids which are useful seeing as I will be using both in my implementation. The only downside to this paper was the lack of testing and results to prove specific methods!

III. Analysis Of The Problem and Challenges

The first problem is that we need to correctly extract letters and words from an image of a printed page. Fortunately, the first task is on printed pages so "letters" will not deviate in size and will not deviate from a given line, by this I mean there will be no "human" error. Once we've built a suitable dataset, we need to train a letter recognizer on the given dataset. For this, we can choose from a range of Supervised/Unsupervised learning algorithms. For the first one, I will use the SVM algorithm. This will, however, generate a couple of problems, one being that the dots on "i" and "j" will be missed and the second being that punctuation will be missed. To overcome this issue the extract letters method needs to accommodate "X" and "Y" boundaries so that when it comes across an "i" or "j" if the dot is within a given distance it recognizes it as part of the letter. The main problem comes in task two when we are analyzing handwritten notes. Extracting the letters, in this case, is no simple feat. We have to take into account that the height of letters will deviate, the distance between letters will vary, and the distance between lines will also vary. We're also going to have to make significant changes to the SVM code from part 1 as that relied on letters being on the same line, but due to human error, this will not be the case. However, once we have generated sufficient training sets teaching the SVM should be easy. The next step is then comparing the accuracy of the SVM against another algorithm. In this case, I will be testing it against the KNN algorithm. We will evaluate the two algorithms by looking at the error margins when reading in pieces of text.

IV. Research

An SVM is a classifier that makes use of basic linear classification and kernel methods[4], it also falls under the bracket of unsupervised learning. Unsupervised learning is a process which involves making use of unclassified or unlabeled data to train an artificial intelligence algorithm to act on that information without any guidance[2]. Basic linear classification works by averaging the data of each class and constructing a point that represents the center of that class(Centroid), new points positions are then determined depending on how close they are to a given centroid. To determine how close a point is to another we make use of the "Euclidean distance" algorithm shown below[3].

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Fig. 1. Euclidean distance

A kernel method is an algorithm that uses pattern analysis to find and study general relations, and these relations can range from clusters to classifications. With this background information, we can now move onto how the SVM itself operates.

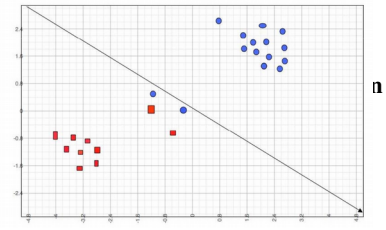


Fig. 2. Basic Linear classification

Figure2 represents a common problem that is found when making use of basic linear classification which is that the dividing line misclassifies some points as they are too close to the line comparred to the rest of the data, this problem is generated as most of the data is much further away. An SVM deals with this issue by generating a line that is as far away as possible from each of the classes known as a "maximum-margin" hyperplane. **Figure3** represents this hyperplane for a given set of classes and is determined by using the two parallel margins.

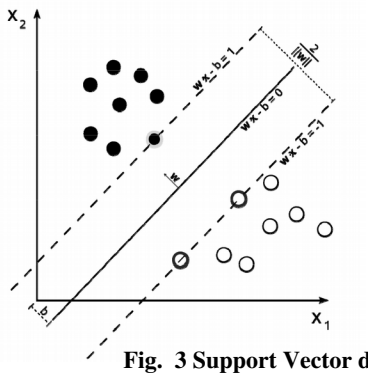


Fig. 3 Support Vector diagram

This is useful as you can determine which class a new set of data points fits in by seeing which side of the line it lies on, The points near the line are known as support vectors (SV). The algorithm which makes use of these vectors to find the dividing line is known as the SVM, which shows the relevance of this research to my stated topics. SVM's work well with high dimensional datasets which means they are often applied to problems with elaborate sets of data which is this case is our "character recognition problem"[4]. There are few advantages when it comes to the use of SVM. One is that it has a regularisation parameter which informs the user to avoid over-fitting. Secondly, we've previously explained that it uses kernel methods which we can engineer to increase the programs knowledge. Finally, after a little research, I found out that most people state it has a performance advantage over most other techniques. One of its main disadvantages is the time it takes to train on more massive datasets. Another problem is that the kernel needs a lot of modification to get perfect results.

I decided that to fully understand classification I would use a technique that falls under the supervised learning (SL) bracket. SL makes use of an external supervisor which presents training sets with known inputs and desired outputs to the network, KNN falls under this description. It is a lot more flexible but is an expensive classification system which classifies an unknown feature vector into the class that is closest to it, also known as the nearest neighbor rule (NN). It can be useful when classes overlap, for example, letters with similarities, e.g., "i and j" who can be easily distinguished. Again KNN makes use of the "Euclidean distance" algorithm to determine the distance between elements. A downside to this algorithm is that you need a large sample of training data for it to function effectively. Also, you must make sure you do not use the test set in the training phase as it will null the effects of the test itself. By using the test set once at the end, we have a good proxy for measuring the generalization of our classifier[4]. The KNN classifier uses a hyperparameter to smooth out the overall process and remove outliers.

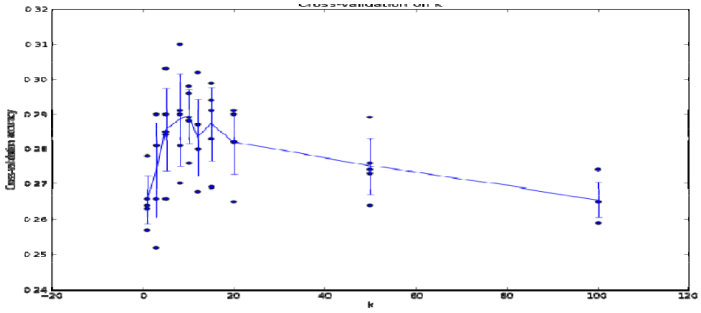


Fig. 4. KNN Clustering Optimal K value

Figure4 shows the results of attunement when it comes to finding the optimal value for K (Hyperparameter[4]. The trend line is an average of all the k values and indicates the error margin. The hyperparameter with the smallest error margin is the most optimal value. In relevance to our topics, we will have an extensive training set which will consist of multiple samples for each character. This means that the network should have a good understanding when it comes to the testing phase. With this research, We can now move onto implementing a solution. The main advantage of using KNN is how useful it is with enormous training datasets, It is also very robust when it comes to noisy training data, and by noisy I mean data that is corrupted. The main disadvantage is that we need to determine the value of K. And unlike the SVM the computational cost if very high as we have to calculate the "Euclidean distance" between every point.

Finally, we need to understand the basic components that a typical optical character recognition (OCR) system consists of. Which are:

1. Capturing and inputting data
2. Pre-Procssing
3. Feature Extraction
4. Classification and recognition
5. Post-Processing
6. Labeling

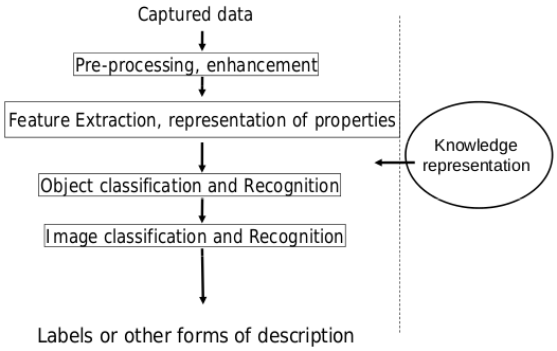


Fig. 5. OCR System Overview

Firstly the image is scanned, and specific dimensions are set so that each image is the same, once this is complete it is sent to the pre-processing phase. Pre-processing improves the overall quality of the image by using segmentation, boundary detection, thinning is also often used to delete dark spots from the image. The next step and the most important step is “feature extraction” which involves getting features like “edges” or “corners.” Feature extraction varies depending on the technique you are using (SVM, KNN…). In the classification phase, individual characters are extracted and processed. Post-processing includes methods such as grouping and error detection. After this process has finished the image can be classified and outputted.

V. Implementation And Technical Issues

The first step when it came to developing a solution for recognizing printed pages was to choose a classifier, in this case, I decided on the SVM classifier. There are a few justifications to this choice, the first being that it is the simplest to engineer out of the three options and the second being that we’re using print pages so there isn't much of an error margin. The first step was to safely extract the characters from all of the training files provided (sorting_character.py). The university provided most of the code for this task, so there wasn't much of a challenge, the only issues with this code are that it could not distinguish between “i” and “j” because the dot was not picked up. This issue is a lot more complex than we think and I fixed it during the handwriting implementation. Once we had all of the letters extracted the next step was to train the classifier. This is fairly simple as we load in each character one by one, extract the HOG features for that specific image and then give it a label(train_classifier.py). Once we’ve done this for every letter, we train the SVM and pickle it. The final stage is to test the classifier on a piece of text(place_detector.py). For this, I simply read in a test file, printed out how many characters were recognized and then checked that each character correctly matched the text file. The overall implementation was accurate to about 90% with only “j’s” and “i's in the wrong place sometimes. With this accuracy, I thought It was justifiable to move on the next task.

The handwritten task was the most challenging implementation out of the three. Straight away I noticed that the letter extractor that I made use of in part 1 was not going to work, due to the fact handwriting deviates so much and does not stay strictly uniform to a line. My first challenge was to overcome this issues, and I started off by printing all the rectangle objects around each letter and stored them within an array. The next step was to loop through this array and run a few checks. I created a few logic gates that checked whether the rectangles were overlapping or within each other or a certain distance. If any of these cases were met, I would create a new rectangle with a size of the two previous rectangles combined. This process was continued until I had fully iterated through the entire array. The next step involved removing any non-letter objects, and these could range from scribbles on a letter to the lines on a page. I did this by removing any objects from the array with a size smaller than a given amount. I then found an issue with the previous method, and that was that I had duplicates of each rectangle on top of each other, so each letter was being recognized multiple times. To overcome this issue I created a new array to store unique rectangles. A simple for loop and a couple of statements allowed me to do this. Numpy has a built-in function that lets you remove duplicate objects and leave unique ones. After lots of trial and error, the method finally works, and it extracts handwritten data correctly, all you have to do is changed the boundaries depending on the distance between lines and letters. Now I have the letters the rest of the SVM implementation is the same as task1 but uses handwriting as the training data. I created a file called (letter_detector) which checks a test file and prints the number of characters. I had a large amount of training data, so the testing phase had reasonably promising results.

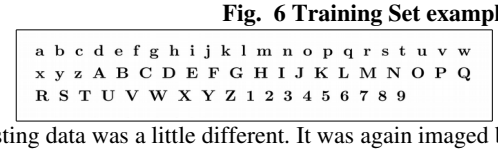
As task two required me to experiment with different classifiers, I decided to test it with a KNN classifier. As I already had a system for extracting handwritten data I only had to work on the classifier itself. I first went through the labs again to understand how it works when reading in images of digits. The issue I found was that KNN only works on 1-dimensional arrays, to overcome this I converted my images to 1d using a built in numpy function. To train the classifier I looped through all of my training data adding them to a dataset and labeling (similar to the SVM implementation but with a few tweaks). I then called the function knn_trainer which was taken from the labs, in short, it trains the knn_classifier on the data I provided. The final step was to test the implementation on a piece of test data. Again I had to follow the same steps when passing in images, but apart from that, it's as simple as comparing the prediction with the probability of a given letter. If they match, then it will print out that particular letter. I didn't have enough training data to get conclusive results on this implementation but from what I found it was less accurate than the SVM implementation.

The final task which was about recognizing signatures was probably the easiest of the three to implement. This is because I could reuse most of my code from the first task and change the classifier slightly. I first collect signatures from three different people before moving onto the implementation side, I always find its easier to test the code if you have a good set of training data. The reason I only chose three people was to make sure the classifier itself worked. The classifier file (train_classifier.py) contains all of the logic for the SVM. It's very similar to the SVM in the first task in which it loads in all the signatures, gets the hog features and saves them to a data list along with a label. Like before once this step is completed I train the SVM and pickle it. The next step is to simply test the SVM by running the file "signature_detection." I pass in a test image, one of the three peoples signatures and output the result type. This proved to be successful for every different person. I didn't see the need to test this what lots more training data as I knew it already worked for the data I had. One thing to note is that with more and more training data the SVM will struggle to distinguish between people with very similar signatures. Now that the implementation side of the coursework is done its time to move onto the testing and evaluation phase.

VI. Evaluation And Results

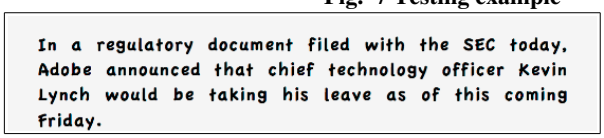
For the first task, I had a vast range of training data. I had six separate images which each had nine sets of lower and upper case characters along with numbers, the only thing that changed within these sets was the font. This means that for each letter there were 108 variants between lower and upper case. And for numbers, there were 54 variations. Figure 5 represents an example of one of these sets of letters.

Fig. 6 Training Set example



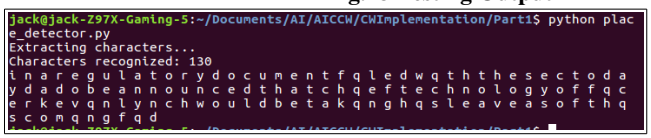
The testing data was a little different. It was again imaged but this time composed of pieces of text(paragraphs), I had a total of 10 images to test it on. I tested the accuracy of each file by printing the number of characters recognized, printing each of these characters to see if it represented the text and then adding up how many errors there were. Figure 6 depicts a short bit of text that I can test the classifier on.

Fig. 7 Testing example



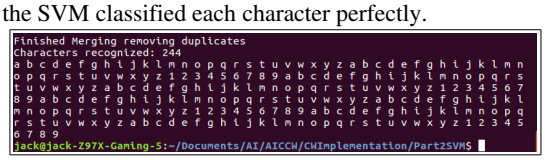
The final figure represents the test run on this piece of text. The number of characters recognized is 130 which is out by one when taking into account punctuation. This gives us a recognition accuracy 130/131 (99%). However the "i's" have been incorrectly determined as "q's" which reduces the accuracy of this SVM. If we total up the number of incorrectly recognized characters, we get 10. This means that the overall recognition accuracy is 121/131 (92%). I tested this on multiple documents (10), and the average recognition accuracy was 90%, which seems like a pretty reasonable result!

Fig. 8 Testing Output



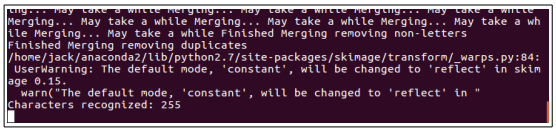
My testing on task two was quite extensive. As I spent so long on the extracting characters side of the implementation it meant I had a large amount of training data. I may have had a little too much which could mean my results were slightly overfitted. For the training data itself I wrote out each letter a numerous number of times and extracted them one by one. This meant the classifier would have a decent amount of examples to work off. In total I had about 1600 characters for testing, about 40 for each character. For the testing side I had a range of documents, some consisted of written paragraphs and others consisted of the alphabet written out similar to task1. When testing the SVM implmentation my results ranged and accuracy of "80 to 100%" after conclusive tests my overall accuracy worked out to be about (88%) which is very similar to my results in task1 recognition accuracy. The figure below represents the results for a test on the alphabet. The document consisted of the alphabet written out 6 times with digits aswell, you can see that in this case the SVM classified each character perfectly.

Fig. 9 SVM handwriting Test output



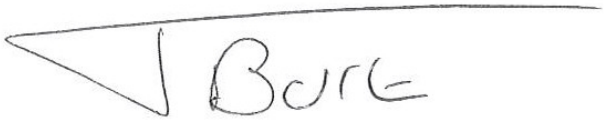
When it came to testing the KNN classifier I used the identical training set data that I used for the SVM. I then tested the KNN on the same testing data that I used previously so that I could compare the two classifiers. The average accuracy recognition for the KNN classifier worked out to be around (80%), a little lower than the SVM implementation. Interesting when I tested the KNN classifier on the text file used in figure 8 it returned an accurace of 98% which was still less than the SVM implementation, this can be seen in figure 9.

Fig. 10 KNN handwriting Test output



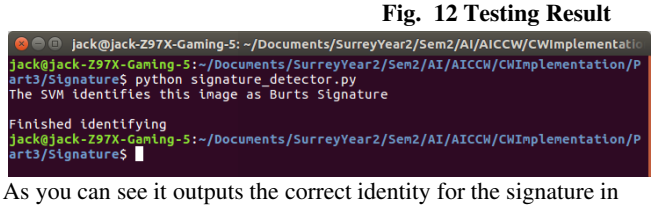
In the final task when it came to testing and training the SVM signature classifier, I had a slightly different technique. My training set consisted of 27 signatures, nine from each person. I understand that when it comes to having multiple peoples signatures, this will not be thorough enough to get conclusive results. The figure below represents an example of a given persons signature.

Fig. 11 Training Set example



For the testing side, again I had nine different signatures from each person, this allowed for me to test each person signatures on a wide range of human error. The figure below is the output of checking one of these signatures on the SVM.

Fig. 12 Testing Result



As you can see it outputs the correct identity for the signature in question. For small inputs, the SVM is never wrong when it comes to identifying the person linked with the signature giving it a recognition accuracy of 100%. However, this will not be the

case with more massive datasets. As we increase the dataset more signatures will have similar hog features. This means that the implementation will lose its accuracy. In this case, I do not think my evaluation of the system is thorough enough. In the future, I would test this on 100 different peoples signatures. This would give me a conclusive recognition accuracy.

VII. Discussion

The implementation side of the project was very successful. I think that I produced robust implementations for all aspects of the project. However, I believe that the testing and evaluation phase has a lot of room for improvement and I would go back and work on this in the future. Most of the work involved playing around with rectangles and there bounding box attribute, which I knew would be an issue from the report on neural trees[7]. Once I had this working correctly I could work on the actual training side of the classifiers. One of the main things I discovered is that the SVM classifier struggles to produce valid results for smaller training sets. Without extensive training, the accuracy recognition of the classifier is extremely low with it not working at all sometimes. Without extensive practice, there will not be much deviation between hog values for individual letters, and at some points, they will probably share the same values due to human error with handwritten scripts. A Histogram of Oriented Gradients (HOG) is a feature extraction method used by the SVM to extract features from all locations in a given image as opposed to KNN which uses local neighborhood points. From what I found it divides the image into small blocks of cells, and then these cells vote on the gradient for that block. Finally, the histogram is normalized so that the values are more spread out and not just focused on one value. After testing a couple of different classifiers, I found that the SVM algorithm was more accurate than the KNN classifier with smaller training data sets. I could not check this with large training sets as I did not have time sit there and write handwriting out due to time constraints. Another problem I found when reading in characters was that the distance between each character often changed (human error). To overcome this issue I came up with some logic gates that make use of X and Y bounds. These bounds will change depending on the specifics of the document. The other issue was that the characters were not on the same y plane, so when it comes to extracting the characters, I had to sort them by y and x to make sure they were stored in their corresponding folder. Another issue is that my program cannot pick up the dots on “i” and “j.” This is because the program struggles to see the dot as a letter due to its small area. In the future to fix this issue, I would tweak my logic gates to take into account specific areas when they are within a particular distance of a letter. The main thing I found is that its pretty much impossible to get a correctly working piece of code as there is so much deviation in character recognition!

VIII. Conclusion

This paper discusses different classification techniques when it comes to recognizing characters. It describes the performance of these classifiers and how accurate they are with different training data sizes. The recognition accuracy of the classifier can be improved by extensively training it and testing it, the same goes for the reliability of the system. The accuracy itself is greatly affected by the type of font and the quality of the writing itself. I would conclude that the SVM classifier is more accurate with smaller training input sizes but as we reach larger training sets neural networks will take over.

VIIII. References

[1] Varone, M. V. What is Machine Learning? A definition. Retrieved from http://www.expertsystem.com/machine-learning-definition/
[2]Rouse, M. unsupervised learning. Retrieved from https://whatis.techtarget.com/definition/unsupervised-learning
[3]Retrieved from https://en.wikipedia.org/wiki/Euclidean_distance
[4] Clark, J. Surrey Artificial Intelligence Lectures. Retrieved from https://surreylearn.surrey.ac.uk/d2l/le/content/154064/Home
[5] Arora#, S. Performance Comparison of SVM and ANN for Handwritten Devnagari Character Recognition. Retrieved from https://arxiv.org/pdf/1006.5902.pdf
[6] Thome, A. C. G. SVM Classifiers – Concepts and Applications to Character Recognition . Retrievedfrom https://pdfs.semanticscholar.org/d563/9ebc1168793d72394a672e032aa57c9a1f7c.pdf
[7] Htike, T. Handwritten Character Recognition Using Competitive Neural Trees . Retrieved from http://www.ijetch.org/papers/574-ST0017.pdf

Appendix – Source Code

Task 1 - Recognising printed pages

File – sorting_character.py

```
import time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from scipy.misc import imread,imresize,imsave
from skimage.segmentation import clear_border
#from skimage.morphology import label
from skimage.measure import regionprops
from skimage.measure import label
from skimage.transform import resize

class Extract_Letters:
    def extractFile(self, filename):
        image = imread(filename,1)

        #apply threshold in order to make the image binary
        bw = image < 120

        # remove artifacts connected to image border
        cleared = bw.copy()
        #clear_border(cleared)

        # label image regions
        label_image = label(cleared,neighbors=8)
        borders = np.logical_xor(bw, cleared)
        label_image[borders] = -1


        fig = plt.figure()
        #ax = fig.add_subplot(131)
        #ax.imshow(bw, cmap='jet')

        letters = list()
        order = list()

        for region in regionprops(label_image):
            minc, minr, maxc, maxr = region.bbox
            # skip small images
            if maxc - minc > len(image)/250: # better to use height rather than area.
                rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                          fill=False, edgecolor='red', linewidth=2)
                order.append(region.bbox)


        #sort the detected characters left->right, top->bottom
```

```
        lines = list()
        first_in_line = ''
        counter = 0

        #worst case scenario there can be 1 character per line
        for x in range(len(order)):
            lines.append([])

        for character in order:
            if first_in_line == '':
                first_in_line = character
                lines[counter].append(character)
            elif abs(character[0] - first_in_line[0]) < (first_in_line[2] - first_in_line[0]):
                lines[counter].append(character)
            elif abs(character[0] - first_in_line[0]) > (first_in_line[2] - first_in_line[0]):
                first_in_line = character
                counter += 1
                lines[counter].append(character)


        for x in range(len(lines)):
            lines[x].sort(key=lambda tup: tup[1])

        final = list()
        prev_tr = 0
        prev_line_br = 0

        for i in range(len(lines)):
            for j in range(len(lines[i])):
                tl_2 = lines[i][j][1]
                bl_2 = lines[i][j][0]
                if tl_2 > prev_tr and bl_2 > prev_line_br:
                    tl,tr,bl,br = lines[i][j]
                    letter_raw = bw[tl:bl,tr:br]
                    letter_norm = resize(letter_raw ,(20 ,20))
                    final.append(letter_norm)
                    prev_tr = lines[i][j][3]
                if j == (len(lines[i])-1):
                    prev_line_br = lines[i][j][2]
            prev_tr = 0
            tl_2 = 0
        print 'Characters recognized: ' + str(len(final))
        return final


    def __init__(self):
        print "Extracting characters..."

start_time = time.time()
extract = Extract_Letters()
```

```
training_files = ['./ocr/training/training1.png',
'./ocr/training/training2.png','./ocr/training/training3.png','./ocr/training/training4.png','./ocr/training/
training6.png']

folder_string = 'abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz123456789'
name_counter = 600
for files in training_files:
        letters = extract.extractFile(files)
        string_counter = 0

        for i in letters:
                if string_counter > 60:
                        string_counter = 0
                imsave('./training_type/' + str(folder_string[string_counter]) + '/' + str(name_counter)
+ '_snippet.png', i)
                print 'training character: ' + str(folder_string[string_counter]) + ' (' +
str(name_counter) + '/' + str(len(letters)) + ')'
                string_counter += 1
                name_counter += 1
print time.time() - start_time, "seconds"
```

Results

Training files loaded in all have the same format. Each letter is extracted and placed in the appropirate folder, most code supplied by lecturer.

```
ack-Z97X-Gaming-5: ~/Documents/AI/AICCW/CWImplementation/Part1
jack@jack-Z97X-Gaming-5:~/Documents/AI/AICCW/CWImplementation/Part1$ python sorting_character.py
Extracting characters...
/home/jack/anaconda2/lib/python2.7/site-packages/skimage/transform/_warps.py:84: UserWarning: The de
  warn("The default mode, 'constant', will be changed to 'reflect' in "
Characters recognized: 549
training character: a (600/549)
training character: b (601/549)
training character: c (602/549)
training character: d (603/549)
training character: e (604/549)
training character: f (605/549)
training character: g (606/549)
training character: h (607/549)
training character: i (608/549)
training character: j (609/549)
training character: k (610/549)
training character: l (611/549)
training character: m (612/549)
training character: n (613/549)
```

File – train_classifier..py

```
import numpy as np
import os
import itertools
import operator
import random
```

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from skimage.feature import hog
from skimage import color, exposure
from scipy.misc import imread,imsave,imresize
import numpy.random as nprnd
from sklearn.svm import SVC
from sklearn import linear_model
from sklearn.svm import LinearSVC
import matplotlib
import pickle


if __name__ == '__main__':


    #format of folders
    folder_string = 'abcdefghijklmnopqrstuvwxyz123456789'


    #create the list that will hold ALL the data and the labels
    #the labels are needed for the classification task:
    data = []
    labels = []
    #Lets me know how far through a given array I am.
    string_counter = 0


    #fill the training dataset
    # the flow is
    # 1) load sample
    # 2) resize it to (200,200) so that we have same size for all the images
    # 3) get the HOG features of the resized image
    # 4) save them in the data list that holds all the hog features
    # 5) also save the label (target) of that sample in the labels list
    # we go through each folder one by one
    while string_counter < 35:
        # within that folder we get all of the files.
        path = './training_type/' + str(folder_string[string_counter]) + '/'
        print 'Number of training images -> 1: ' + str((path))
        filenames = sorted([filename for filename in os.listdir(path) if (filename.endswith('.jpg') or
 filename.endswith('.png') or (filename.endswith('.bmp'))) ])
        #print 'Number of training images -> 2: ' + str((filenames))
        filenames = [path+filename for filename in filenames]
        #print 'Number of training images -> 3: ' + str((filenames))
        for filename in filenames:
            #read the images
            image = imread(filename,1)
            #flatten it
            image = imresize(image, (200,200))
            hog_features = hog(image, orientations=12, pixels_per_cell=(16, 16),
                    cells_per_block=(1, 1))
```

```
        data.append(hog_features)
        labels.append(str(folder_string[string_counter]))
    #print 'Filename print' + str(filenames) + '/'
    string_counter += 1
```

```
print 'Training the SVM'
    #create the SVC
clf = LinearSVC(dual=False,verbose=1)
    #train the svm
clf.fit(data, labels)

#pickle it - save it to a file
pickle.dump( clf, open( "letter.detector", "wb" ) )
```

Results

It goes through each folder one by 1 and loads in all the image samples with lables

```
2.53281497955 seconds
jack@jack-Z97X-Gaming-5:~/Documents/AI/AICCW/CWImplementation/Part1$ python train_classifier.py
  File "train_classifier.py", line 26
    labels =
           ^
SyntaxError: invalid syntax
jack@jack-Z97X-Gaming-5:~/Documents/AI/AICCW/CWImplementation/Part1$ python train_classifier.py
Number of training images -> 1: ./training_type/a/
/home/jack/anaconda2/lib/python2.7/site-packages/skimage/feature/_hog.py:119: skimage_deprecation: Def
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)
Number of training images -> 1: ./training_type/b/
Number of training images -> 1: ./training_type/c/
Number of training images -> 1: ./training_type/d/
Number of training images -> 1: ./training_type/e/
Number of training images -> 1: ./training_type/f/
Number of training images -> 1: ./training_type/g/
```

We then train the svm on this list of data and labels and save it to a pickle file

```
iter 15 act 4.071e-03 pre 5.506e-03 delta 1.969e-02 f 8.981e+00 |g| 8.975e-01 CG   7
cg reaches trust region boundary
iter 16 act 3.584e-03 pre 4.490e-03 delta 1.969e-02 f 8.977e+00 |g| 7.881e-01 CG   7
cg reaches trust region boundary
iter 17 act 1.910e-03 pre 3.672e-03 delta 1.371e-02 f 8.974e+00 |g| 8.688e-01 CG   7
cg reaches trust region boundary
iter 18 act 3.176e-03 pre 3.288e-03 delta 1.386e-02 f 8.972e+00 |g| 1.055e+00 CG   7
cg reaches trust region boundary
iter 19 act 1.600e-03 pre 1.614e-03 delta 1.499e-02 f 8.968e+00 |g| 3.117e-01 CG   6
cg reaches trust region boundary
iter 20 act 1.412e-03 pre 1.774e-03 delta 1.499e-02 f 8.967e+00 |g| 3.238e-01 CG   7
cg reaches trust region boundary
iter 21 act 1.582e-03 pre 1.855e-03 delta 1.499e-02 f 8.965e+00 |g| 5.090e-01 CG   7
cg reaches trust region boundary
```

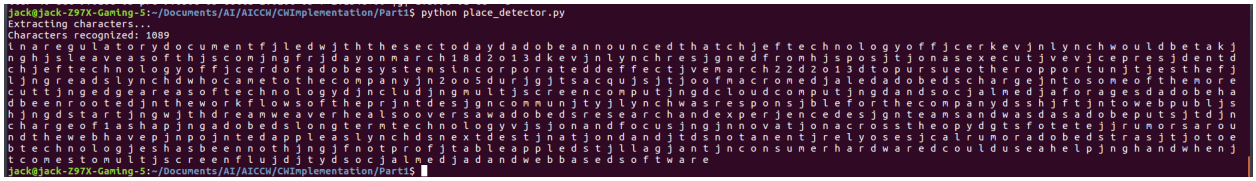
File – letter_detector.py

```
from skimage.feature import hog
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
import pickle
import time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from scipy.misc import imread,imresize,imsave
from skimage.segmentation import clear_border
#from skimage.morphology import label
from skimage.measure import regionprops
from skimage.measure import label
from skimage.transform import resize
### Remove forced-depreciation warnings about outdated python modules
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
### End warning removal



class Extract_Letters:
    def extractFile(self, filename):
        image = imread(filename,1)


        #apply threshold in order to make the image binary
        bw = image < 120


        # remove artifacts connected to image border
        cleared = bw.copy()
        #clear_border(cleared)


        # label image regions
        label_image = label(cleared,neighbors=8)
        borders = np.logical_xor(bw, cleared)
        label_image[borders] = -1



        fig = plt.figure()
        #ax = fig.add_subplot(131)
        #ax.imshow(bw, cmap='jet')
```

```
letters = list()
order = list()


for region in regionprops(label_image):
    minc, minr, maxc, maxr = region.bbox
    # skip small images
    if maxc - minc > len(image)/250: # better to use height rather than area.
        rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr,
                                  fill=False, edgecolor='red', linewidth=2)
        order.append(region.bbox)



#sort the detected characters left->right, top->bottom
lines = list()
first_in_line = ''
counter = 0


#worst case scenario there can be 1 character per line
for x in range(len(order)):
    lines.append([])


for character in order:
    if first_in_line == '':
        first_in_line = character
        lines[counter].append(character)
    elif abs(character[0] - first_in_line[0]) < (first_in_line[2] - first_in_line[0]):
        lines[counter].append(character)
    elif abs(character[0] - first_in_line[0]) > (first_in_line[2] - first_in_line[0]):
        first_in_line = character
        counter += 1
        lines[counter].append(character)



for x in range(len(lines)):
    lines[x].sort(key=lambda tup: tup[1])


final = list()
prev_tr = 0
prev_line_br = 0


for i in range(len(lines)):
    for j in range(len(lines[i])):
        tl_2 = lines[i][j][1]
        bl_2 = lines[i][j][0]
        if tl_2 > prev_tr and bl_2 > prev_line_br:
```

```
                tl,tr,bl,br = lines[i][j]
                letter_raw = bw[tl:bl,tr:br]
                letter_norm = resize(letter_raw ,(20 ,20))
                final.append(letter_norm)
                prev_tr = lines[i][j][3]
            if j == (len(lines[i])-1):
                prev_line_br = lines[i][j][2]
        prev_tr = 0
        tl_2 = 0
    print 'Characters recognized: ' + str(len(final))
    return final



  def __init__(self):
    print "Extracting characters..."


extract = Extract_Letters()
letters = extract.extractFile('adobe.png')
letter_count = 0


for i in letters:
  imsave('./data/' + str(letter_count) + '_snippet.png', i)
  letter_count += 1



clf = pickle.load( open("letter.detector","rb"))


letter_count = 0
for j in letters:
  #now load a test image and get the hog features.
  test_image = imread('./data/' + str(letter_count) + '_snippet.png',1) # you can modify which image
 is tested by changing the filename here
  test_image = imresize(test_image, (200,200))



  hog_features = hog(test_image, orientations=12, pixels_per_cell=(16, 16),
      cells_per_block=(1, 1))
  hog_features = hog_features.reshape(1, -1)


  result_type = clf.predict(hog_features)
  print str(result_type)[2],


  letter_count += 1
```

Results



Task 2 – Handwritten notes

SVM Implementation
File – extract_letters.py

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from scipy.misc import imread,imresize,imsave
from skimage.segmentation import clear_border
#from skimage.morphology import label
from skimage.measure import label
from skimage.measure import regionprops
from skimage.transform import resize

class Extract_Letters:
	def extractFile(self, filename):
		image = imread(filename,1)
		#apply threshold in order to make the image binary
		bw = image < 120

		# remove artifacts connected to image border
		cleared = bw.copy()
		clear_border(cleared)

		# label image regions
		label_image = label(cleared,neighbors=8)
		borders = np.logical_xor(bw, cleared)
		label_image[borders] = -1

		print label_image.max()

		fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(6, 6))
		ax.imshow(bw, cmap='jet')
```

```
            #Setting Up variables
            letters = list()
            order = list()
            #if the area of the object/letter is less than 255 remove it from the array
            invalidAreaObjects = 255
            #store a list of all merged rectangle regions
            mergedRectangles = np.array([])
            #array to store non_duplicate rectangles
            uniqueArray = np.array([])
            #array which stores all non letters (e.g. scribbles on page)
            nonElements = np.array([])
            #declaring the distance between each bbox (cannot be any bigger or all letters
rectangles(bbox) would merge)
            bbox_xBoundary = 40
            bbox_yBoundary = 140


            #Loop which finds all rectangles for a given image and appends them to an array
            for region in regionprops(label_image):
                  minr, minc, maxr, maxc = region.bbox
                  # skip small images
                  if region.area > 40:
                        # draw rectangle around segmented objects
                        selectedRectangle = mpatches.Rectangle((minc, minr), maxc - minc,
maxr - minr,
                                                               fill=False,
edgecolor='red', linewidth=2)
                        # add new rectangle to array
                        mergedRectangles = np.append(mergedRectangles, selectedRectangle)


            # This is where all the logic happens when it comes to merging overalpping/close
rectangles around a letter
            for i in range(0,mergedRectangles.size):
                  #Select a rectangle from the array to compare
                  selectedRectangle = mergedRectangles[i]
                  for j in range(0, mergedRectangles.size):
                        # go through every other rectangle and run logic gate checks
                        previousRectangle = mergedRectangles[j]
                        # Check to see whether the two rectangles overalap


overlap=(selectedRectangle.get_bbox()).fully_overlaps(previousRectangle.get_bbox())or(selectedR
ectangle.get_bbox().x1-previousRectangle.get_bbox().x1<bbox_xBoundary and
selectedRectangle.get_bbox().x1-previousRectangle.get_bbox().x1>-
bbox_xBoundary)and(previousRectangle.get_bbox().y0<selectedRectangle.get_bbox().y0<previous
Rectangle.get_bbox().y1<selectedRectangle.get_bbox().y1)
                        # The next two checks are to see whether the rectangles are within a
certain distance of eachother i.e. less than the boundaries.
```

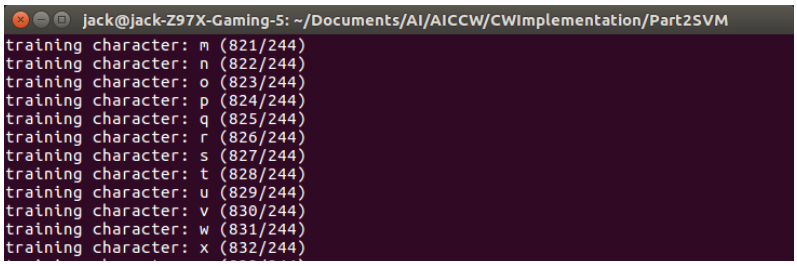
```
                                checkOne=(selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y0<bbox_yBoundary and selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y0>-bbox_yBoundary)and(selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x0<bbox_xBoundary and selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x0>-bbox_xBoundary)or(selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y1<bbox_yBoundary and selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y1>-bbox_yBoundary)and(selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x1<bbox_xBoundary and selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x1>-bbox_xBoundary)
                                checkTwo=(selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y1<bbox_yBoundary and selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y1>-bbox_yBoundary)and(selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x1<bbox_xBoundary and selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x1>-bbox_xBoundary)or(selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y0<bbox_yBoundary and selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y0>-bbox_yBoundary)and(selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x0<bbox_xBoundary and selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x0>-bbox_xBoundary)
                                #If statement to  check logic gates
                                if overlap or checkOne or checkTwo:
                                        print"Merging... May take a while",
                                        if
previousRectangle.get_bbox().x0>selectedRectangle.get_bbox().x0:minc=selectedRectangle.get_bb
ox().x0
                                        else:minc=previousRectangle.get_bbox().x0
                                        if
previousRectangle.get_bbox().x1>selectedRectangle.get_bbox().x1:maxc=previousRectangle.get_b
box().x1
                                        else:maxc=selectedRectangle.get_bbox().x1
                                        if
previousRectangle.get_bbox().y0>selectedRectangle.get_bbox().y0:minr=selectedRectangle.get_bb
ox().y0
                                        else:minr=previousRectangle.get_bbox().y0
                                        if
previousRectangle.get_bbox().y1>selectedRectangle.get_bbox().y1:maxr=previousRectangle.get_b
box().y1
                                        else:maxr=selectedRectangle.get_bbox().y1
                                        #create and store a new rectangle which consists of both
previous rectangles
                                        newRectangle=mpatches.Rectangle((minc,minr),maxc-
minc,maxr-minr,fill=False,edgecolor='red',linewidth=2)
                                        # store new rectangle
                                        mergedRectangles[j] = newRectangle
                print "Finished Merging removing non-letters"

                # This checks for objects whose areas are smaller than a given amount
                for i in range(0,mergedRectangles.size):
                        #if the object is smaller than the given area store it in a new array
```

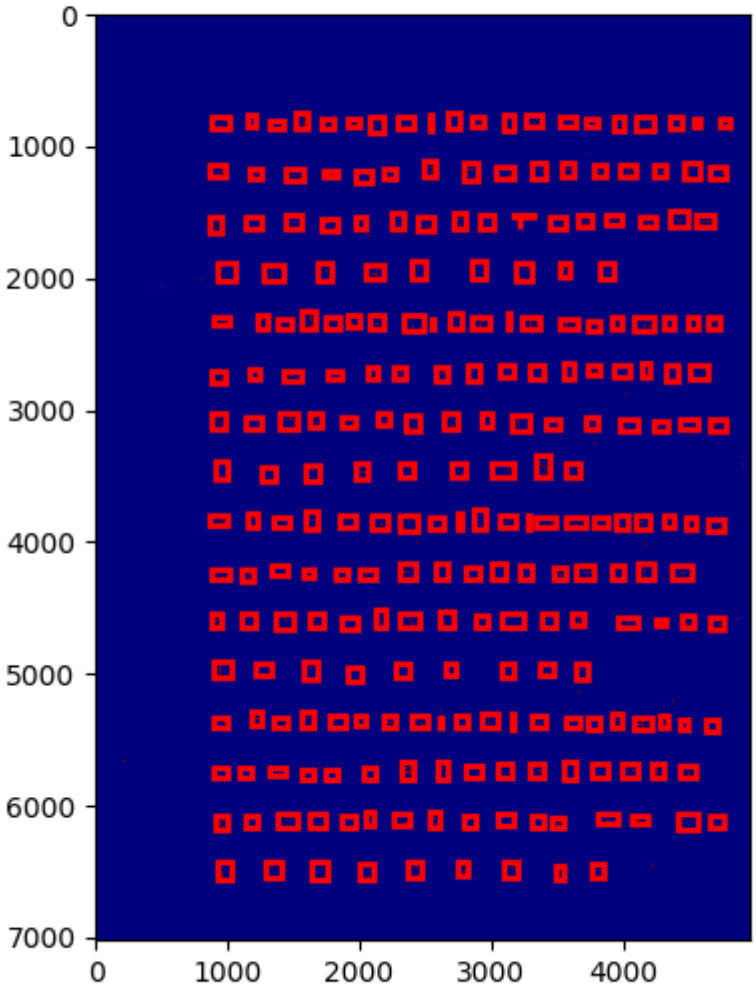
```
                if mergedRectangles[i].get_width() * mergedRectangles[i].get_height() < invalidAreaObjects: nonElements = np.append(nonElements,mergedRectangles[i])

        # This line removes all the nonElements from the main array
        mergedRectangles = np.setdiff1d(mergedRectangles,nonElements)

        print "Finished Merging removing duplicates"
        # Storing only unique rectangles in a new array (removing duplicates)
        for i in reversed(range(0,mergedRectangles.size)):
                #check if value is unqiue or not
                noDup = False
                for j in range (0,i):
                        # If the two rectangles are equal move onto the next one
                        if str(mergedRectangles[j].get_bbox()) == str(mergedRectangles[i].get_bbox()):
                                noDup = True
                                break
                if noDup == False:
                        uniqueArray = np.append(uniqueArray, mergedRectangles[i])

        # Going through the sorted array and adding the rectangles to the image (Checking system)
        for i in range(0,uniqueArray.size):
                ax.add_patch(uniqueArray[i])
                # getting the bbox of that rectangle and storing it in the list order for chracter extraction
                bbox = long(uniqueArray[i].get_bbox().y0), long(uniqueArray[i].get_bbox().x0), long(uniqueArray[i].get_bbox().y1), long(uniqueArray[i].get_bbox().x1)
                order.append(bbox)


        lines = list()
        first_in_line = ''
        counter = 0


        # I need to sort by y value as the bboxs are scattered randomly within the array for example (800,1500,700)
        # This will not function correctly when using your extract letters code.
        # By ordering the values correcly i can simply import your code to extract the letters
        order.sort(key=lambda tup: tup[0])
        #worst case scenario there can be 1 character per line
        for x in range(len(order)):
                lines.append([])

        for character in order:
                if first_in_line == '':
                        first_in_line = character
```

```
                        lines[counter].append(character)
                elif abs(character[0] - first_in_line[0]) < (first_in_line[2] - first_in_line[0]):
                        lines[counter].append(character)
                elif abs(character[0] - first_in_line[0]) > (first_in_line[2] - first_in_line[0]):
                        first_in_line = character
                        counter += 1
                        lines[counter].append(character)


        for x in range(len(lines)):
                lines[x].sort(key=lambda tup: tup[1])

        final = list()
        prev_tr = 0
        prev_line_br = 0

        for i in range(len(lines)):
                for j in range(len(lines[i])):
                        tl_2 = lines[i][j][1]
                        bl_2 = lines[i][j][0]
                        if tl_2 > prev_tr and bl_2 > prev_line_br:
                                tl,tr,bl,br = lines[i][j]
                                letter_raw = bw[tl:bl,tr:br]
                                letter_norm = resize(letter_raw ,(20 ,20))
                                final.append(letter_norm)
                                prev_tr = lines[i][j][3]
                        if j == (len(lines[i])-1):

                                prev_line_br = lines[i][j][2]
                prev_tr = 0
                tl_2 = 0
        print 'Characters recognized: ' + str(len(final))
        return final


        def __init__(self):
                print "Extracting characters..."


extract = Extract_Letters()
training_files = ['1.png']

folder_string = 'abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz123456789'
name_counter = 600
for files in training_files:
        letters = extract.extractFile(files)
        string_counter = 0
```

```
        for i in letters:
                if string_counter > 60:
                        string_counter = 0
                imsave('./training_type/' + str(folder_string[string_counter]) + '/' + str(name_counter)
+ '_snippet.png', i)
                print 'training character: ' + str(folder_string[string_counter]) + ' (' +
str(name_counter) + '/' + str(len(letters)) + ')'
                string_counter += 1
                name_counter += 1

plt.show()
```

Results This was the most challenging part of the coursework but as you can see it works perfectly!





File – train_classifier.py

```
import numpy as np
import os
import itertools
import operator
import random
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from skimage.feature import hog
from skimage import color, exposure
from scipy.misc import imread,imsave,imresize
import numpy.random as nprnd
from sklearn.svm import SVC
from sklearn import linear_model
from sklearn.svm import LinearSVC
import matplotlib
import pickle

if __name__ == '__main__':



    #Same technique as 1 and 2.
    folder_string = 'abcdefghijklmnopqrstuvwxyz123456789'
    print 'folderstring -> 1: ' + str(len(folder_string))
    data = []
    labels = []
    string_counter = 0


    #fill the training dataset
    # the flow is
    # 1) load sample
    # 2) resize it to (200,200) so that we have same size for all the images
    # 3) get the HOG features of the resized image
    # 4) save them in the data list that holds all the hog features
    # 5) also save the label (target) of that sample in the labels list
    # we go through each folder one by one
    while string_counter < 35:
        path = './training_type/' + str(folder_string[string_counter]) + '/'
        print 'Number of training images -> 1: ' + str((path))
        filenames = sorted([filename for filename in os.listdir(path) if (filename.endswith('.jpg') or
 filename.endswith('.png') or (filename.endswith('.bmp'))) ])
        #print 'Number of training images -> 2: ' + str((filenames))
        filenames = [path+filename for filename in filenames]
        #print 'Number of training images -> 3: ' + str((filenames))
        for filename in filenames:
            #read the images
```

```
        image = imread(filename,1)
        #flatten it
        image = imresize(image, (200,200))
        hog_features = hog(image, orientations=12, pixels_per_cell=(16, 16),
                cells_per_block=(1, 1))
        data.append(hog_features)
        labels.append(str(folder_string[string_counter]))
    #print 'Filename print' + str(filenames) + '/'
    string_counter += 1




print 'Training the SVM'
    #create the SVC
clf = LinearSVC(dual=False,verbose=1)
    #train the svm
clf.fit(data, labels)


#pickle it - save it to a file
pickle.dump( clf, open( "letter.detector", "wb" ) )
```

Results

```
jack@jack-Z97X-Gaming-5: ~/Documents/AI/AICCW/CWImplementation/Part2SVM
image_deprecation: Default value of `block_norm`==`L1` is deprecated and will be
 changed to `L2-Hys` in v0.15
  'be changed to `L2-Hys` in v0.15', skimage_deprecation)
Number of training images -> 1: ./training_type/b/
Number of training images -> 1: ./training_type/c/
Number of training images -> 1: ./training_type/d/
Number of training images -> 1: ./training_type/e/
Number of training images -> 1: ./training_type/f/
Number of training images -> 1: ./training_type/g/
Number of training images -> 1: ./training_type/h/
Number of training images -> 1: ./training_type/i/
Number of training images -> 1: ./training_type/j/
Number of training images -> 1: ./training_type/k/
Number of training images -> 1: ./training_type/l/
Number of training images -> 1: ./training_type/m/
Number of training images -> 1: ./training_type/n/
Number of training images -> 1: ./training_type/o/
Number of training images -> 1: ./training_type/p/
Number of training images -> 1: ./training_type/q/
Number of training images -> 1: ./training_type/r/
Number of training images -> 1: ./training_type/s/
Number of training images -> 1: ./training_type/t/
Number of training images -> 1: ./training_type/u/
```

File – letter_detector.py

```
from skimage.feature import hog
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
import pickle
import time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from scipy.misc import imread,imresize,imsave
from skimage.segmentation import clear_border
#from skimage.morphology import label
from skimage.measure import regionprops
from skimage.measure import label
from skimage.transform import resize
### Remove forced-depreciation warnings about outdated python modules
def warn(*args, **kwargs):
	pass
import warnings
warnings.warn = warn
### End warning removal


class Extract_Letters:
	def extractFile(self, filename):
		image = imread(filename,1)
		#apply threshold in order to make the image binary
		bw = image < 120

		# remove artifacts connected to image border
		cleared = bw.copy()
		clear_border(cleared)

		# label image regions
		label_image = label(cleared,neighbors=8)
		borders = np.logical_xor(bw, cleared)
		label_image[borders] = -1

		print label_image.max()

		fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(6, 6))
		ax.imshow(bw, cmap='jet')

		#Setting Up variables
		letters = list()
		order = list()
		#if the area of the object/letter is less than 255 remove it from the array
```

```
            invalidAreaObjects = 255
            #store a list of all merged rectangle regions
            mergedRectangles = np.array([])
            #array to store non_duplicate rectangles
            uniqueArray = np.array([])
            #array which stores all non letters (e.g. scribbles on page)
            nonElements = np.array([])
            #declaring the distance between each bbox (cannot be any bigger or all letters
rectangles(bbox) would merge)
            bbox_xBoundary = 40
            bbox_yBoundary = 140


            #Loop which finds all rectangles for a given image and appends them to an array
            for region in regionprops(label_image):
                  minr, minc, maxr, maxc = region.bbox
                  # skip small images
                  if region.area > 40:
                        # draw rectangle around segmented objects
                        selectedRectangle = mpatches.Rectangle((minc, minr), maxc - minc,
maxr - minr,
                                                               fill=False,
edgecolor='red', linewidth=2)
                        # add new rectangle to array
                        mergedRectangles = np.append(mergedRectangles, selectedRectangle)


            # This is where all the logic happens when it comes to merging overalpping/close
rectangles around a letter
            for i in range(0,mergedRectangles.size):
                  #Select a rectangle from the array to compare
                  selectedRectangle = mergedRectangles[i]
                  for j in range(0, mergedRectangles.size):
                        # go through every other rectangle and run logic gate checks
                        previousRectangle = mergedRectangles[j]
                        # Check to see whether the two rectangles overalap


overlap=(selectedRectangle.get_bbox()).fully_overlaps(previousRectangle.get_bbox())or(selectedR
ectangle.get_bbox().x1-previousRectangle.get_bbox().x1<bbox_xBoundary and
selectedRectangle.get_bbox().x1-previousRectangle.get_bbox().x1>-
bbox_xBoundary)and(previousRectangle.get_bbox().y0<selectedRectangle.get_bbox().y0<previous
Rectangle.get_bbox().y1<selectedRectangle.get_bbox().y1)
                        # The next two checks are to see whether the rectangles are within a
certain distance of eachother i.e. less than the boundaries.
                        checkOne=(selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y0<bbox_yBoundary and selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y0>-bbox_yBoundary)and(selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x0<bbox_xBoundary and selectedRectangle.get_bbox().x0-
```

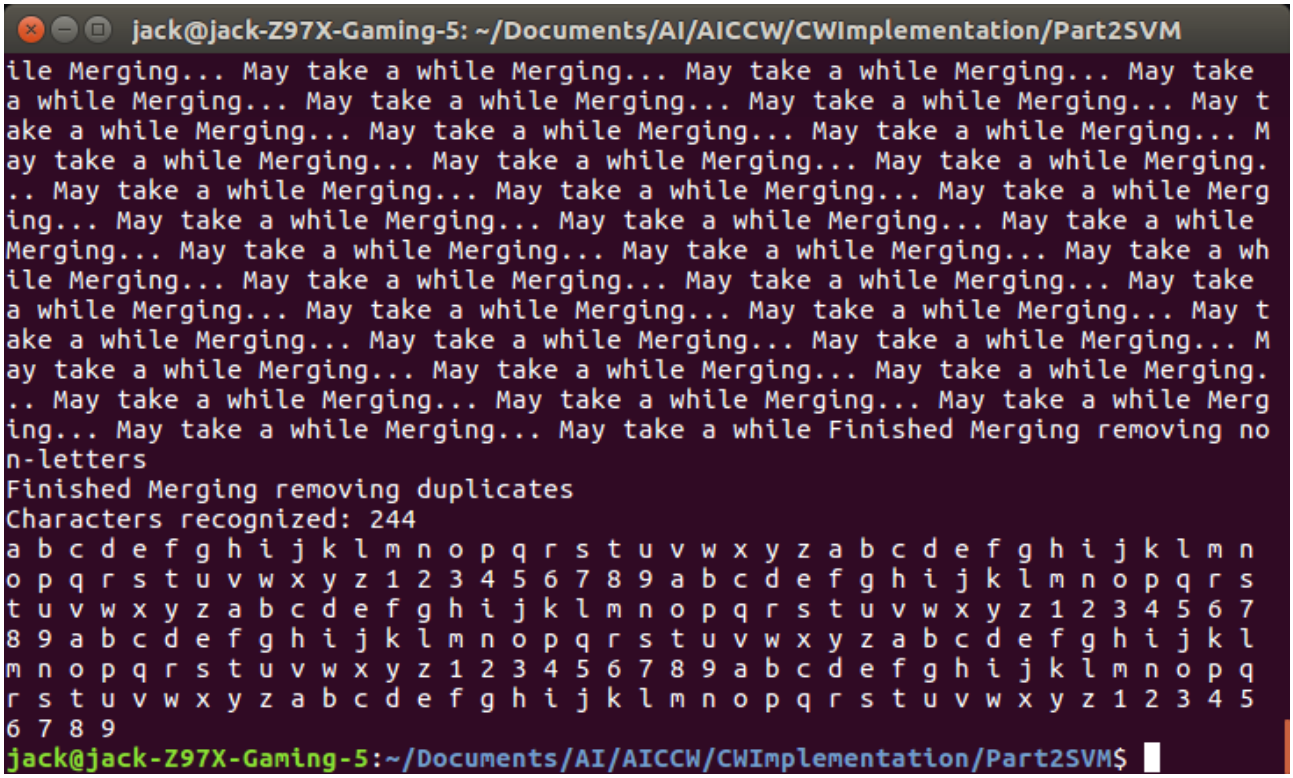
```
previousRectangle.get_bbox().x0>-bbox_xBoundary)or(selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y1<bbox_yBoundary and selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y1>-bbox_yBoundary)and(selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x1<bbox_xBoundary and selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x1>-bbox_xBoundary)
                                checkTwo=(selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y1<bbox_yBoundary and selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y1>-bbox_yBoundary)and(selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x1<bbox_xBoundary and selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x1>-bbox_xBoundary)or(selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y0<bbox_yBoundary and selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y0>-bbox_yBoundary)and(selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x0<bbox_xBoundary and selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x0>-bbox_xBoundary)
                                #If statement to  check logic gates
                                if overlap or checkOne or checkTwo:
                                        print"Merging... May take a while",
                                        if
previousRectangle.get_bbox().x0>selectedRectangle.get_bbox().x0:minc=selectedRectangle.get_bb
ox().x0
                                        else:minc=previousRectangle.get_bbox().x0
                                        if
previousRectangle.get_bbox().x1>selectedRectangle.get_bbox().x1:maxc=previousRectangle.get_b
box().x1
                                        else:maxc=selectedRectangle.get_bbox().x1
                                        if
previousRectangle.get_bbox().y0>selectedRectangle.get_bbox().y0:minr=selectedRectangle.get_bb
ox().y0
                                        else:minr=previousRectangle.get_bbox().y0
                                        if
previousRectangle.get_bbox().y1>selectedRectangle.get_bbox().y1:maxr=previousRectangle.get_b
box().y1
                                        else:maxr=selectedRectangle.get_bbox().y1
                                        #create and store a new rectangle which consists of both
previous rectangles
                                        newRectangle=mpatches.Rectangle((minc,minr),maxc-
minc,maxr-minr,fill=False,edgecolor='red',linewidth=2)
                                        # store new rectangle
                                        mergedRectangles[j] = newRectangle
                print "Finished Merging removing non-letters"

                # This checks for objects whose areas are smaller than a given amount
                for i in range(0,mergedRectangles.size):
                        #if the object is smaller than the given area store it in a new array
                        if mergedRectangles[i].get_width() * mergedRectangles[i].get_height() <
invalidAreaObjects: nonElements = np.append(nonElements,mergedRectangles[i])

                # This line removes all the nonElements from the main array
                mergedRectangles = np.setdiff1d(mergedRectangles,nonElements)
```

```
print "Finished Merging removing duplicates"
# Storing only unique rectangles in a new array (removing duplicates)
for i in reversed(range(0,mergedRectangles.size)):
	#check if value is unqiue or not
	noDup = False
	for j in range (0,i):
		# If the two rectangles are equal move onto the next one
		if str(mergedRectangles[j].get_bbox()) == 
str(mergedRectangles[i].get_bbox()):
			noDup = True
			break
	if noDup == False:
		uniqueArray = np.append(uniqueArray, mergedRectangles[i])

# Going through the sorted array and adding the rectangles to the image (Checking 
system)
for i in range(0,uniqueArray.size):
	ax.add_patch(uniqueArray[i])
	# getting the bbox of that rectangle and storing it in the list order for chracter 
extraction
	bbox = long(uniqueArray[i].get_bbox().y0), 
long(uniqueArray[i].get_bbox().x0), long(uniqueArray[i].get_bbox().y1), 
long(uniqueArray[i].get_bbox().x1)
	order.append(bbox)


lines = list()
first_in_line = ''
counter = 0


# I need to sort by y value as the bboxs are scattered randomly within the array for 
example (800,1500,700)
# This will not function correctly when using your extract letters code.
# By ordering the values correcly i can simply import your code to extract the letters
order.sort(key=lambda tup: tup[0])
#worst case scenario there can be 1 character per line
for x in range(len(order)):
	lines.append([])

for character in order:
	if first_in_line == '':
		first_in_line = character
		lines[counter].append(character)
	elif abs(character[0] - first_in_line[0]) < (first_in_line[2] - first_in_line[0]):
		lines[counter].append(character)
	elif abs(character[0] - first_in_line[0]) > (first_in_line[2] - first_in_line[0]):
		first_in_line = character
```

```
                        counter += 1
                        lines[counter].append(character)


        for x in range(len(lines)):
                lines[x].sort(key=lambda tup: tup[1])

        final = list()
        prev_tr = 0
        prev_line_br = 0

        for i in range(len(lines)):
                for j in range(len(lines[i])):
                        tl_2 = lines[i][j][1]
                        bl_2 = lines[i][j][0]
                        if tl_2 > prev_tr and bl_2 > prev_line_br:
                                tl,tr,bl,br = lines[i][j]
                                letter_raw = bw[tl:bl,tr:br]
                                letter_norm = resize(letter_raw ,(20 ,20))
                                final.append(letter_norm)
                                prev_tr = lines[i][j][3]
                        if j == (len(lines[i])-1):

                                prev_line_br = lines[i][j][2]
                prev_tr = 0
                tl_2 = 0
        print 'Characters recognized: ' + str(len(final))
        return final


        def __init__(self):
                print "Extracting characters..."


extract = Extract_Letters()
letters = extract.extractFile('test1.jpg')
letter_count = 0

for i in letters:
        imsave('./data/' + str(letter_count) + '_snippet.png', i)
        letter_count += 1


        #load the detector
clf = pickle.load( open("letter.detector","rb"))

letter_count = 0
for j in letters:
        #now load a test image and get the hog features.
```

```
        test_image = imread('./data/' + str(letter_count) + '_snippet.png',1) # you can modify which image is tested by changing the filename here
        test_image = imresize(test_image, (200,200))


        hog_features = hog(test_image, orientations=12, pixels_per_cell=(16, 16),
                           cells_per_block=(1, 1))
        hog_features = hog_features.reshape(1, -1)

        result_type = clf.predict(hog_features)
        print str(result_type)[2],
        letter_count += 1
```

Results



KNN Implementation

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from skimage.segmentation import clear_border
from skimage.morphology import label
from skimage.measure import label
from scipy.misc import imread,imresize,imsave
from sklearn.cluster import KMeans
import pickle
```

```
import os
from skimage.measure import regionprops
from skimage.transform import resize
from sklearn.neighbors import KNeighborsClassifier

#Same method as before...
class Extract_Letters:
	def extractFile(self, filename):
		image = imread(filename,1)
		#apply threshold in order to make the image binary
		bw = image < 120

		# remove artifacts connected to image border
		cleared = bw.copy()
		clear_border(cleared)

		# label image regions
		label_image = label(cleared,neighbors=8)
		borders = np.logical_xor(bw, cleared)
		label_image[borders] = -1

		print label_image.max()

		fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(6, 6))
		ax.imshow(bw, cmap='jet')

		#Setting Up variables
		letters = list()
		order = list()
		#if the area of the object/letter is less than 255 remove it from the array
		invalidAreaObjects = 255
		#store a list of all merged rectangle regions
		mergedRectangles = np.array([])
		#array to store non_duplicate rectangles
		uniqueArray = np.array([])
		#array which stores all non letters (e.g. scribbles on page)
		nonElements = np.array([])
		#declaring the distance between each bbox (cannot be any bigger or all letters
rectangles(bbox) would merge)
		bbox_xBoundary = 40
		bbox_yBoundary = 70


		#Loop which finds all rectangles for a given image and appends them to an array
		for region in regionprops(label_image):
			minr, minc, maxr, maxc = region.bbox
			# skip small images
			if region.area > 40:
				# draw rectangle around segmented objects
```

```
                        selectedRectangle = mpatches.Rectangle((minc, minr), maxc - minc,
maxr - minr,
                                                               fill=False,
edgecolor='red', linewidth=2)
                        # add new rectangle to array
                        mergedRectangles = np.append(mergedRectangles, selectedRectangle)


        # This is where all the logic happens when it comes to merging overalpping/close
rectangles around a letter
        for i in range(0,mergedRectangles.size):
            #Select a rectangle from the array to compare
            selectedRectangle = mergedRectangles[i]
            for j in range(0, mergedRectangles.size):
                # go through every other rectangle and run logic gate checks
                previousRectangle = mergedRectangles[j]
                # Check to see whether the two rectangles overalap


overlap=(selectedRectangle.get_bbox()).fully_overlaps(previousRectangle.get_bbox())or(selectedR
ectangle.get_bbox().x1-previousRectangle.get_bbox().x1<bbox_xBoundary and
selectedRectangle.get_bbox().x1-previousRectangle.get_bbox().x1>-
bbox_xBoundary)and(previousRectangle.get_bbox().y0<selectedRectangle.get_bbox().y0<previous
Rectangle.get_bbox().y1<selectedRectangle.get_bbox().y1)
                # The next two checks are to see whether the rectangles are within a
certain distance of eachother i.e. less than the boundaries.
                checkOne=(selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y0<bbox_yBoundary and selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y0>-bbox_yBoundary)and(selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x0<bbox_xBoundary and selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x0>-bbox_xBoundary)or(selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y1<bbox_yBoundary and selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y1>-bbox_yBoundary)and(selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x1<bbox_xBoundary and selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x1>-bbox_xBoundary)
                checkTwo=(selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y1<bbox_yBoundary and selectedRectangle.get_bbox().y0-
previousRectangle.get_bbox().y1>-bbox_yBoundary)and(selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x1<bbox_xBoundary and selectedRectangle.get_bbox().x0-
previousRectangle.get_bbox().x1>-bbox_xBoundary)or(selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y0<bbox_yBoundary and selectedRectangle.get_bbox().y1-
previousRectangle.get_bbox().y0>-bbox_yBoundary)and(selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x0<bbox_xBoundary and selectedRectangle.get_bbox().x1-
previousRectangle.get_bbox().x0>-bbox_xBoundary)
                #If statement to  check logic gates
                if overlap or checkOne or checkTwo:
                        print"Merging... May take a while",
```

```
                                        if
previousRectangle.get_bbox().x0>selectedRectangle.get_bbox().x0:minc=selectedRectangle.get_bb
ox().x0
                                        else:minc=previousRectangle.get_bbox().x0
                                        if
previousRectangle.get_bbox().x1>selectedRectangle.get_bbox().x1:maxc=previousRectangle.get_b
box().x1
                                        else:maxc=selectedRectangle.get_bbox().x1
                                        if
previousRectangle.get_bbox().y0>selectedRectangle.get_bbox().y0:minr=selectedRectangle.get_bb
ox().y0
                                        else:minr=previousRectangle.get_bbox().y0
                                        if
previousRectangle.get_bbox().y1>selectedRectangle.get_bbox().y1:maxr=previousRectangle.get_b
box().y1
                                        else:maxr=selectedRectangle.get_bbox().y1
                                        #create and store a new rectangle which consists of both
previous rectangles
                                        newRectangle=mpatches.Rectangle((minc,minr),maxc-
minc,maxr-minr,fill=False,edgecolor='red',linewidth=2)
                                        # store new rectangle
                                        mergedRectangles[j] = newRectangle
                print "Finished Merging removing non-letters"

                # This checks for objects whose areas are smaller than a given amount
                for i in range(0,mergedRectangles.size):
                        #if the object is smaller than the given area store it in a new array
                        if mergedRectangles[i].get_width() * mergedRectangles[i].get_height() <
invalidAreaObjects: nonElements = np.append(nonElements,mergedRectangles[i])

                # This line removes all the nonElements from the main array
                mergedRectangles = np.setdiff1d(mergedRectangles,nonElements)

                print "Finished Merging removing duplicates"
                # Storing only unique rectangles in a new array (removing duplicates)
                for i in reversed(range(0,mergedRectangles.size)):
                        #check if value is unqiue or not
                        noDup = False
                        for j in range (0,i):
                                # If the two rectangles are equal move onto the next one
                                if str(mergedRectangles[j].get_bbox()) ==
str(mergedRectangles[i].get_bbox()):
                                        noDup = True
                                        break
                        if noDup == False:
                                uniqueArray = np.append(uniqueArray, mergedRectangles[i])

                # Going through the sorted array and adding the rectangles to the image (Checking
system)
```

```
        for i in range(0,uniqueArray.size):
                ax.add_patch(uniqueArray[i])
                # getting the bbox of that rectangle and storing it in the list order for chracter extraction
                bbox = long(uniqueArray[i].get_bbox().y0), long(uniqueArray[i].get_bbox().x0), long(uniqueArray[i].get_bbox().y1), long(uniqueArray[i].get_bbox().x1)
                order.append(bbox)


        lines = list()
        first_in_line = ''
        counter = 0


        # I need to sort by y value as the bboxs are scattered randomly within the array for example (800,1500,700)
        # This will not function correctly when using your extract letters code.
        # By ordering the values correcly i can simply import your code to extract the letters
        order.sort(key=lambda tup: tup[0])
        #worst case scenario there can be 1 character per line
        for x in range(len(order)):
                lines.append([])

        for character in order:
                if first_in_line == '':
                        first_in_line = character
                        lines[counter].append(character)
                elif abs(character[0] - first_in_line[0]) < (first_in_line[2] - first_in_line[0]):
                        lines[counter].append(character)
                elif abs(character[0] - first_in_line[0]) > (first_in_line[2] - first_in_line[0]):
                        first_in_line = character
                        counter += 1
                        lines[counter].append(character)


        for x in range(len(lines)):
                lines[x].sort(key=lambda tup: tup[1])

        final = list()
        prev_tr = 0
        prev_line_br = 0

        for i in range(len(lines)):
                for j in range(len(lines[i])):
                        tl_2 = lines[i][j][1]
                        bl_2 = lines[i][j][0]
                        if tl_2 > prev_tr and bl_2 > prev_line_br:
                                tl,tr,bl,br = lines[i][j]
```

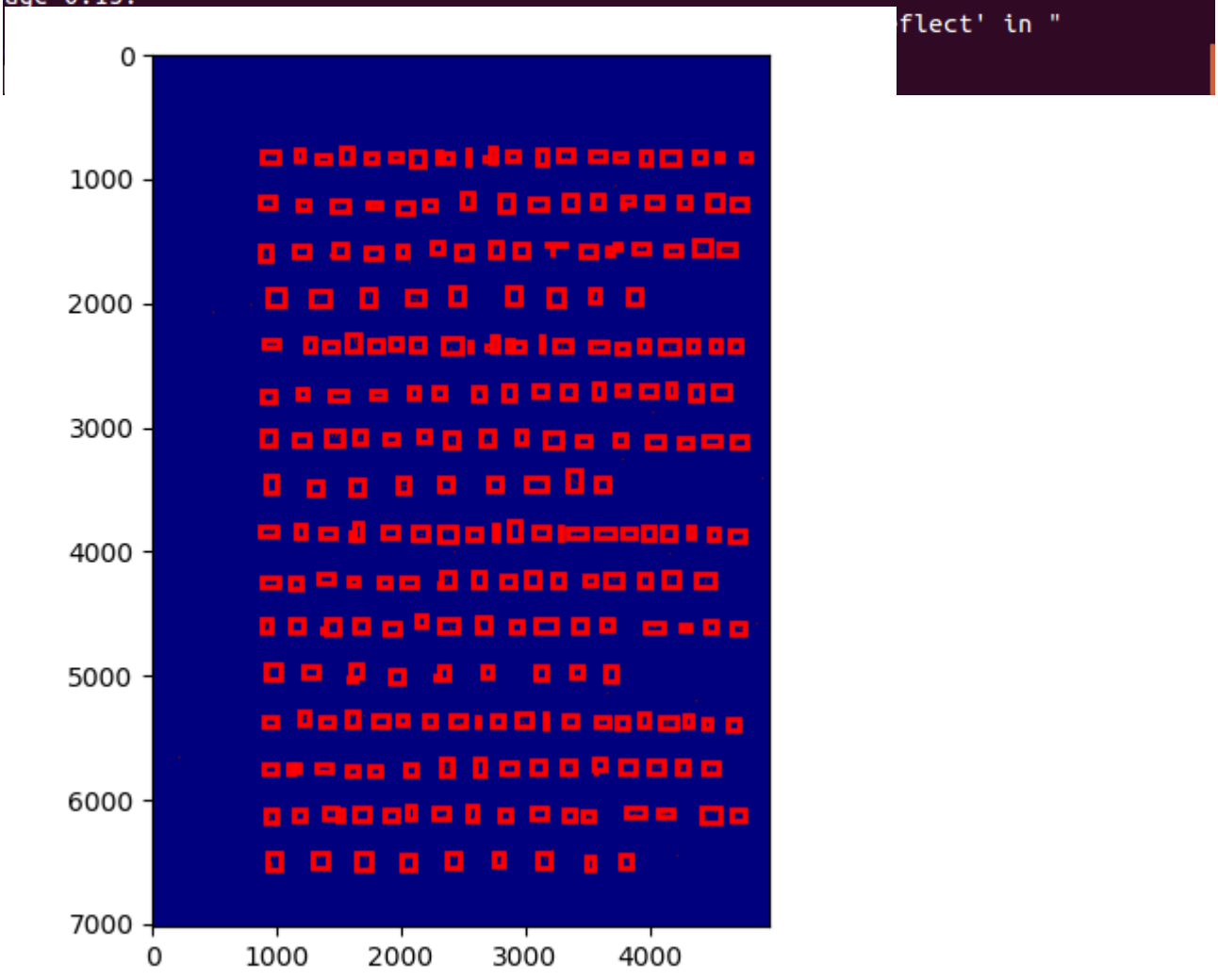
```
                                letter_raw = bw[tl:bl,tr:br]
                                letter_norm = resize(letter_raw ,(20 ,20))
                                final.append(letter_norm)
                                prev_tr = lines[i][j][3]
                        if j == (len(lines[i])-1):

                                prev_line_br = lines[i][j][2]
                prev_tr = 0
                tl_2 = 0
        print 'Characters recognized: ' + str(len(final))
        return final


        def __init__(self):
                print "Extracting characters..."

class knn_trainer:
        def train_knn_classifier(self):
                k=10
                knn_classifier = KNeighborsClassifier(n_neighbors=k)
                print knnData.shape
                knn_classifier.fit(knnData, imageLabels)
                print 'Trained the knn-classifier'
                return knn_classifier


folder_string = 'abcdefghijklmnopqrstuvwxyz123456789'
print 'folderstring -> 1: ' + str(len(folder_string))
knnData = []
imageLabels = []
string_counter = 0


#Same loop as before with a few minor changes
while string_counter < 35:
        #get path of current character
        path = './training_type/' + str(folder_string[string_counter]) + '/'
        print 'Number of training images -> 1: ' + str((path))
        filenames = sorted([filename for filename in os.listdir(path) if (filename.endswith('.jpg') or
filename.endswith('.png') or (filename.endswith('.bmp'))) ])
        filenames = [path+filename for filename in filenames]
        for filename in filenames:
                #read the images
                image = imread(filename,1)
                #flatten it
                image = imresize(image, (200,200))
                #Reszing the image to a 1 dimensional array
                imageRavel = image.ravel()
                #append 1d array
```

```
                knnData.append(imageRavel)
                #add corresponding label to other array
                imageLabels.append(str(folder_string[string_counter]))
        string_counter += 1
# Store knnData as a numpy array
knnData = np.asarray(knnData)
# Store label as a numpy array
imageLabels = np.asarray(imageLabels)

# Similar to SVM but training KNN instead
knn_trainer = knn_trainer()
knnclf = knn_trainer.train_knn_classifier()

extract = Extract_Letters()
letters = extract.extractFile('./ocr/testing/testing2.jpg')
letter_count = 0

for i in letters:
        imsave('./data/' + str(letter_count) + '_snippet.png', i)
        letter_count += 1

folder_string = 'abcdefghijklmnopqrstuvwxyz123456789'

letter_count = 0
for j in letters:
        try:
                image = imread('./data/' + str(letter_count) + '_snippet.png',1) # you can modify
which image is tested by changing the filename here
                image = imresize(image, (200,200))

                # conver to 1 d array
                testingLetter = image.ravel()
                # reshape to make it a 2 dimensional array
                testingLetter = testingLetter.reshape(1,-1)
                # conver to numpy array
                testingLetter = np.asarray(testingLetter)
                # get the knn value from that array for the given letter
                knn_results = knnclf.predict_proba(testingLetter)[0]
                prev_value = 0
                prev_index = 0
                # calls the classifier to predict the probability of what letter it is
                for k,result in enumerate(knn_results):
                        if result > prev_value:
                                prev_index = k
                                prev_value = result # actual probability it is k

                #determine what letter it is in the string
                print folder_string[prev_index],
                letter_count += 1
```

```
        except Exception,e:
                print "Caught an Exception with details: %s" % e

plt.show()
```

Results

```
jack@jack-Z97X-Gaming-5: ~/Documents/AI/AICCW/CWImplementation/Part2KNN
ing... May take a while Merging... May take a while Merging... May take a while
Merging... May take a while Merging... May take a while Merging... May take a wh
ile Merging... May take a while Merging... May take a while Merging... May take
a while Merging... May take a while Merging... May take a while Merging... May t
ake a while Merging... May take a while Merging... May take a while Merging... M
ay take a while Merging... May take a while Merging... May take a while Merging.
.. May take a while Merging... May take a while Merging... May take a while Merg
ing... May take a while Merging... May take a while Merging... May take a while
Merging... May take a while Merging... May take a while Merging... May take a wh
ile Merging... May take a while Merging... May take a while Merging... May take
a while Merging... May take a while Merging... May take a while Merging... May t
ake a while Merging... May take a while Merging... May take a while Merging... M
ay take a while Merging... May take a while Merging... May take a while Merging.
.. May take a while Merging... May take a while Merging... May take a while Merg
ing... May take a while Merging... May take a while Merging... May take a while
Merging... May take a while Merging... May take a while Merging... May take a wh
ile Merging... May take a while Finished Merging removing non-letters
Finished Merging removing duplicates
/home/jack/anaconda2/lib/python2.7/site-packages/skimage/transform/_warps.py:84:
 UserWarning: The default mode, 'constant', will be changed to 'reflect' in skim
age 0.15.
                                                                    flect' in "
```



Task 3 – Signature recognition

File – sorting_character.py

```
import numpy as np
import os
import itertools
import operator
import random
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from skimage.feature import hog
from skimage import color, exposure
from scipy.misc import imread,imsave,imresize
import numpy.random as nprnd
from sklearn.svm import SVC
from sklearn import linear_model
from sklearn.svm import LinearSVC
import matplotlib
import pickle

if __name__ == '__main__':

    #Same code as part 1 pretty much
    data = []
    labels = []



    #At the moment only testing it on 3 different people
    folder_string = 'bhs'
    print 'folderstring -> 1: ' + str(len(folder_string))
    string_counter = 0
    string_countertest = 0


    # 3 people so loop through 3 times
    # Exact same process as part 1 when it comes to extracting hog_features
    #fill the training dataset
    # the flow is
```

```
    # 1) load sample
    # 2) resize it to (200,200) so that we have same size for all the images
    # 3) get the HOG features of the resized image
    # 4) save them in the data list that holds all the hog features
    # 5) also save the label (target) of that sample in the labels list
    # we go through each folder one by one
    while string_counter < 3:
        path = './training_type/' + str(folder_string[string_counter]) + '/'
        filenames = sorted([filename for filename in os.listdir(path) if (filename.endswith('.jpg') or
filename.endswith('.png') or (filename.endswith('.bmp'))) ])
        filenames = [path+filename for filename in filenames]
        for filename in filenames:
            #read the images
            print (string_countertest)
            image = imread(filename,1)
            #flatten it
            image = imresize(image, (200,200))
            hog_features = hog(image, orientations=12, pixels_per_cell=(16, 16),
                    cells_per_block=(1, 1))
            data.append(hog_features)
            labels.append(str(folder_string[string_counter]))
            string_countertest += 1
        #print 'Filename print' + str(filenames) + '/'
        string_counter += 1


    print 'Training the SVM'
    #create the SVC
    clf = LinearSVC(dual=False,verbose=1)
    #train the svm
    clf.fit(data, labels)


    #pickle it - save it to a file
    pickle.dump( clf, open( "signature.detector", "wb" ) )
```

Results

The screenshot below shows the hog features of each signature being saved into the svm.

```
ack-Z97X-Gaming-5: ~/Documents/AI/AICCW/CWImplementation/Part3/Signature
iter 13 act 7.444e-05 pre 8.489e-05 delta 6.582e-03 f 1.955e-01 |g| 5.420e-02 CG   6
cg reaches trust region boundary
iter 14 act 4.260e-05 pre 4.260e-05 delta 6.600e-03 f 1.954e-01 |g| 4.830e-02 CG   9
iter  1 act 2.652e+01 pre 2.640e+01 delta 6.778e-01 f 2.700e+01 |g| 8.622e+01 CG   3
iter  2 act 2.162e-01 pre 2.182e-01 delta 6.778e-01 f 4.845e-01 |g| 4.659e+00 CG   4
iter  3 act -5.487e-01 pre 2.325e-02 delta 5.093e-02 f 2.683e-01 |g| 5.283e-01 CG   7
cg reaches trust region boundary
iter  3 act -5.793e-03 pre 9.822e-03 delta 1.882e-02 f 2.683e-01 |g| 5.283e-01 CG   4
cg reaches trust region boundary
iter  3 act 4.795e-03 pre 4.942e-03 delta 2.108e-02 f 2.683e-01 |g| 5.283e-01 CG   3
cg reaches trust region boundary
iter  4 act 1.992e-03 pre 2.074e-03 delta 2.445e-02 f 2.635e-01 |g| 1.905e-01 CG   3
cg reaches trust region boundary
iter  5 act -1.455e-03 pre 2.468e-03 delta 9.000e-03 f 2.615e-01 |g| 2.380e-01 CG   4
cg reaches trust region boundary
iter  5 act 9.013e-04 pre 1.121e-03 delta 9.000e-03 f 2.615e-01 |g| 2.380e-01 CG   3
cg reaches trust region boundary
iter  6 act 6.490e-04 pre 6.490e-04 delta 1.324e-02 f 2.606e-01 |g| 1.325e-01 CG   3
cg reaches trust region boundary
iter  7 act 2.874e-04 pre 7.889e-04 delta 8.904e-03 f 2.600e-01 |g| 9.872e-02 CG   3
cg reaches trust region boundary
iter  8 act 6.130e-04 pre 6.182e-04 delta 9.314e-03 f 2.597e-01 |g| 2.122e-01 CG   4
cg reaches trust region boundary
iter  9 act 1.822e-04 pre 2.690e-04 delta 7.766e-03 f 2.591e-01 |g| 5.981e-02 CG   4
cg reaches trust region boundary
iter 10 act 1.843e-04 pre 1.843e-04 delta 8.426e-03 f 2.589e-01 |g| 1.027e-01 CG   6
cg reaches trust region boundary
iter 11 act -2.442e-04 pre 7.766e-05 delta 2.107e-03 f 2.587e-01 |g| 3.537e-02 CG   6
cg reaches trust region boundary
iter 11 act 3.106e-05 pre 3.326e-05 delta 2.110e-03 f 2.587e-01 |g| 3.537e-02 CG   3
cg reaches trust region boundary
iter 12 act 1.248e-05 pre 1.248e-05 delta 2.166e-03 f 2.587e-01 |g| 1.825e-02 CG   5
cg reaches trust region boundary
iter 13 act 1.393e-05 pre 1.393e-05 delta 2.632e-03 f 2.586e-01 |g| 2.156e-02 CG   5
cg reaches trust region boundary
```

File – signatute_detector.py

```
from skimage.feature import hog
from scipy.misc import imread,imresize
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
import pickle

### Remove forced-depreciation warnings about outdated python modules
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
### End warning removal


if __name__ == '__main__':
    #load the detector
    clf = pickle.load( open("signature.detector","rb"))

    #now load a test image and get the hog features.
    test_image = imread('./testing/s/sh.jpg',1) # you can modify which image is tested by changing the filename here
    test_image = imresize(test_image, (200,200))

    hog_features = hog(test_image, orientations=12, pixels_per_cell=(16, 16),
```

```
                cells_per_block=(1, 1))
    hog_features = hog_features.reshape(1, -1)

    # result_type returns a number based on whether the image predicted is a b h or s depending on the person
    result_type = clf.predict(hog_features)


    # we now translate the above result into a string, making the result easier to understand
    if result_type == 'b':
            print 'The SVM identifies this image as Burts Signature'
    elif result_type == 'h':
            print "The SVM identifies this image as  Hawkins Signature"
    elif result_type == 's':
            print "The SVM identifies this image as a Brians Signature"
    else:
            print "Something went wrong"

print '\nFinished identifying'
```

Results

The screenshot below confirms that the screenshot passed in belongs to Brian which is correct