CSE415
Final Project
June 2nd 2015
Yu Fu, Yinghe Chen

License Plate Recognition

This program is made by Yinghe Chen, and Yu Fu. Yinghe is responsible for implement perceptron classifier, and part of license plate data collection. Yu Fu is in charge of making naïve Bayes classifier, plate segmentation, and bagging two classifiers together.

This program is designed to recognize the license plate picture. Plate segmentation.py process the picture into a folder which contains every single letter or number. Two different classifiers are ready to read the folder not only independently but also in bagging condition.

Perceptron classifier takes folder of single letters, which generated by plate segmentation.py, as input, and divided picture into 20 x 20 cells. Compare each cells to our data set, and returned each most similar letter from dataset back.

One interesting thing I find in this program is that it may take lots of time for traditional perceptron training algorithm to change each cell's weight. Since the size of picture of single letter is set by plate segmentation.py. I changed my training algorithm target at center of the picture from 5 to 15 columns, and 5 to 15 rows. This reduces the running time of program, and accuracy does not change at all.

Yinghe Chen learned about perceptron classifier implementation, data collection, work as a team (how to overcome time conflict). More importantly, I feel happy to make something that people can used in daily life, and see the significance of machine learning, and if I have more time, I will reduce my training algorithms complexity, and dataset collection so that the license plate will be read much faster and accurately.
I learned perceptron from CSE415 website Logic 1: Propositional calculus and resolution, and most of dataset pictures are collected from google image. Thank both partners to get us knowledge, and data to make this program come true.

The purpose of Image segmentation step is to extract the images of characters in a license plate image. The ideas used here is that the characters on the license plate are connected and have the same color. And they are usually have high contrast with background color. Therefore, the processing has following steps:

1. Threshold
2. Find and label connected components
3. Calculate height and width of character blocks
4. Crop the characters and output

Yu uses the package scikit-image to perform the image process. For the thresholding part, Yu use threshold_otsu() function, which will perform Otsu's method to threshold the image to binary image. The method will search for the threshold that minimizes the intra-class variance, which is useful in separate the foreground characters with the background. Following, I use the label() method to find the connected components. Then for each connected component, there is a box containing the component. I calculated the height and width range of such components that they should be the characters. The algorithm I use here is to sort the heights of all the components from large to small values, and start searching from the max height. If there are at least 5 components with similar heights, then take them as the candidates of characters. Then I exclude the boxes that is too wide, and the rest are the characters. After getting the boxes, I crop the box, and padding with white background to make the image square. Then the image of character can be used in the classifiers.

There are some drawbacks about the algorithm on deciding which boxes are the characters, and during tests it will not successfully identify all the images. However, we are expecting about 70% successful rate. Also, adjusting some parameters in the program can lead to correct results, but we still fail to find parameters that will work for all the test cases. Also, the threshold algorithm will not work for plate with shadows on it. Therefore the best image used here should be high-resolution, sharp, and with good lighting condition. Also for best results in the following image recognition algorithms, the image should not be tilting.

If Yu Fu has more time, he will work on the algorithm to make it more accurate and better adapted for different images. Also, I might implement my own threshold algorithm to see if I can get better result.

The following code gives the algorithm of deciding which boxes are the characters. When determining if boxes have similar height, I take variable eps, which is set to be 1% of the image height here. The percentage can be changed, and the result will be affected as well. For the width part, I sort the width, and if the width are 1.5 times wider than other ones, it will be excluded. Also, the parameter here can be changed to get different results.

```
# Calculate height range of character
heights = []
for region in regionprops(label_image):
    minr, minc, maxr, maxc = region.bbox
    height = maxr - minr
    heights.append(height)

list.sort(heights)
list.reverse(heights)

max_height = heights[0]
min_height = max_height + 1
eps = img.shape[1] * 0.01
count = 0
for i in range(len(heights) - 1):
    min_height = heights[i+1]
    if max_height-min_height >= eps:
        count = 0
        max_height = min_height
    else:
        count = count + 1
        if count == 5:
            break
max_height = max_height + eps
min_height = min_height - eps
```

```
# Calculate width range of character
widths = []
for region in regionprops(label_image):
    minr, minc, maxr, maxc = region.bbox
    height = maxr - minr
    width = maxc - minc
    if height <= max_height and height >= min_height:
        widths.append(width)
list.sort(widths)
list.reverse(widths)
max_width = widths[0]
count = 1
while widths[count] * 1.5 < max_width:
    max_width = widths[count]
    count = count + 1
```

References:
[1] http://scikit-image.org/docs/dev/auto_examples/plot_label.html
[2] http://en.wikipedia.org/wiki/Otsu%27s_method

Part II: Naive Bayes Classifier By Yu Fu

The Naive Bayes Classifier takes an image of a character generated by the image segmentation algorithm described in Part I, and output the character as text. For each image, the classifier will read resize the image using bicubic interpolation to 30x30, and read it as a binary matrix, and reshape it into an array of length 900. Each number here represent a pixel of image after resize, and is treated as one attribute in Naive Bayes Classifier. Therefore there are total 900 attributes. Each attribute are assumed to be independent. Then we read the training set, and calculate the priors and individual features. For an input array, the classifier will calculate the character with highest likelihood given the input data. The classifier can also read all the images in a directory and output a string of all characters it recognizes.

I tried to reduce the attribute sizes by projecting to x-axis and y-axis. In this way it will only have 60 attributes. However, the result does not perform well in the test, because some characters will have similar projections on both axes so it does not work. The large attribute size causes some time issues. The classifier runs relatively slow (6 chars for about 20-30s). Also, it still have some trouble distinguish between similar characters such as ‘2’ and ‘Z’, ‘M’ and ‘N’.

Below is the code to compute the likelihood of character c by given input. Since each probability is small, I use log-likelihood here, and add them together.

```
def get_likelihood(input_list, c, indi_prob, prior_char):
    '''Calculate and return P(c|input)'''
    patterns = "0123456789abcdefghijklmnpqrstuvwxyz"
    char_index = patterns.index(c)
    log_p_input_c = 0
    prob_list = indi_prob[char_index][0]

    for i in range(900):
        if input_list[i] == 0:
            log_p_input_c += math.log(1 - prob_list[i])
        else:
            log_p_input_c += math.log(prob_list[i])
    p_c = prior_char[char_index]
    return log_p_input_c + math.log(p_c)
```

Here is the code that for each image, it will iterate through all possible characters, and find the one with largest (log-)likelihood and use that as the result.

```
def get_result(imgpath, indi_prob, prior_char):
    '''Use Naive Bayes classifier to compute
       the character with max likihood of given image array.'''
    input = img_to_array(imgpath)
    max_log_p = -sys.maxsize - 1
    max_c = ''
    for c in "0123456789abcdefghijklmnpqrstuvwxyz":
        log_p = get_likelihood(input, c, indi_prob, prior_char)
        if log_p > max_log_p:
            max_log_p = log_p
            max_c = c
    return (max_c, max_log_p)
```

If time allows, I will increase the training set to see if the result will be more accurate. Also, maybe introduce some new attributes such as curvature, number of closed circles, etc. Hopefully I can reduce the attribute size and have a better timing performance.

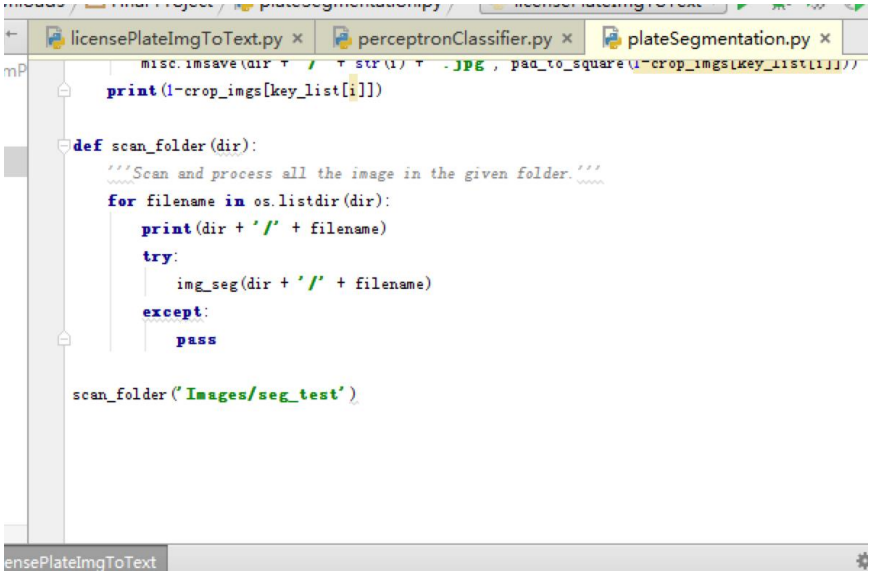
Reference:

[1] https://courses.cs.washington.edu/courses/cse415/15sp/slides/17-Naive-Bayes-2-up.pdf
[2] STAT 391 Lecture Notes, Marina Meila

Sample:

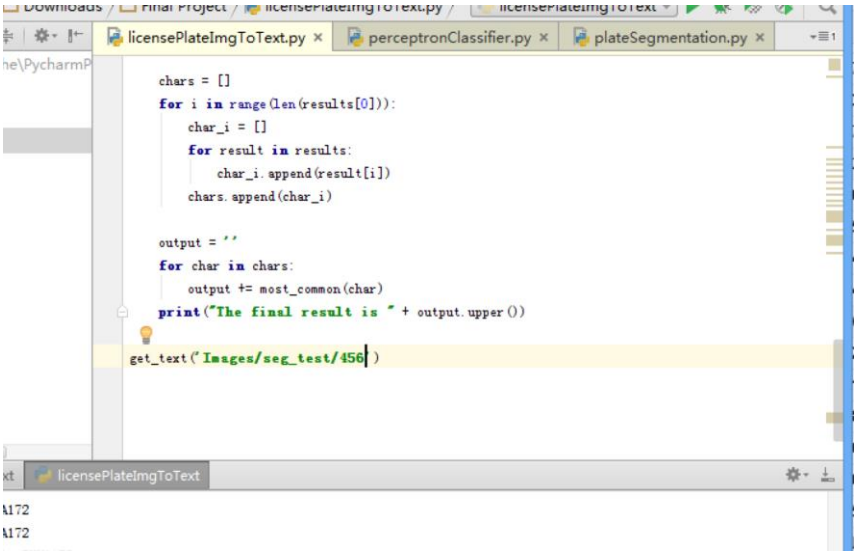

```
C:\Python34\python.exe "C:/Users/Yinghe/Downloads/Final Proje
c43974
Classifier 1: 043974
Classifier 2: C43974
Classifier 3: C43974
Classifier 4: C43974
Classifier 5: C43974
Classifier 6: C43974
Classifier 7: C43974
Classifier 8: C43974
Classifier 9: C43974
The final result is C43974
```

Demon Instructions:



1. put the license plate picture you want to recognize as path “Image/seg_test” (same folder as py file) in plateSegmentation.py

2. after you run the plateSegmentation.py. According folder should be generated in path “Image/seg_test/?”

3. 

   In file licensePlateImgtoText.py, change get_text(‘Image/seg_test/?’) Replace ? as the folder generated in second step, and run the code.

4. Now see the results as print lines from py file.