Advanced Techniques in Machine Learning

Experience 2

Due Date: 03/01/2019

Yossi = {Keshet, Adi}

In this exercise you will implement and compare multiclass and structured Perceptron. For that purpose, we will use an OCR dataset containing letter sequences of letters. Each image is composed of 8*16 binary pixels.

You are provided with train and test files, and a README.md file so you can understand the structure of the file.

Models:

1. For the first part you will implement a standard multi-class Perceptron (pseudo code for the update rule can be found at the appendix section).
2. Second, you will implement a multi-class perceptron as a structured perceptron model. For that purpose we need to define the $\emptyset(x_i, y_i)$ (pseudo code for the update rule can be found at the appendix section). We define $\emptyset(x_i, y_i)$ as follows: the dimension of the vector will be the number-of-classes * dimension-of-xi, and the $\emptyset(x_i, y_i)$ function will be: placing the features of xi at the yi-th place in the vector and leave the reset zeros. Another way, to think about it will be flattening the W matrix from part (1) and multiply xi by the relevant Wi by padding it with zeros.
3. Lastly, you will add to the model from part 2, more elements concerning the previous label. In other words, we will add an indicator at the previous prediction. For that purpose, we will add 27*27 (all pairs of English characters plus start character - $), elements to $\emptyset(x_i, y_i)$. These additional features indicate all the possible bi-grams. Now, in order to compute $\emptyset(x_i, y_i)$, we will place xi features at the yi-th place and an indicator at the place of yi-1*yi of the additional features, the rest of the vector will be zeros.

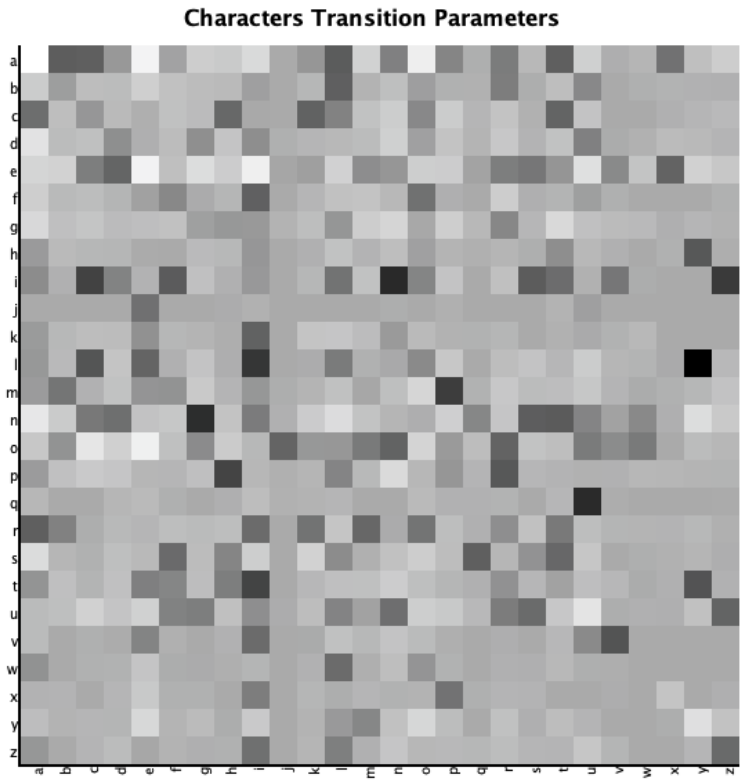
Inference:

1. For the first and second parts you can use a standard inference, i.e. one for loop over all classes.
2. For the last part we cannot use the same inference procedure (why?), we will need to use a dynamic programming algorithm since we have a dependency between time step i and time step i-1 (pseudo code for the update rule can be found at the appendix section).

What to submit?

1. Accuracy on the test set using the three methods.
2. A file with the predictions on the test set (each prediction in a separate line) DO NOT SUFFLE THE TEST SET.
3. A brief PDF report with your conclusion from the comparison between the methods, who performed best, what method is better, etc.
4. Python3.6 code of the project.

5. **Bonus (10 pts)**: plot a Heat-map matrix graph of the bi-gram character features, i.e. the elements in W related to the connection between the characters (y-axis is the prev character and x-axis is the current character). You should get something similar to that (black is high value and white is low value):



For example look at the pairs (q,u), (l,y), (n,g): all of these get high value since they are likely to appear together. On the other hand (u,u), (e,i), (a,a) will get low value.

6. **Bonus (5 pts)**: If we would like to use Structured-SVM instead of Structured Perceptron, what would be the difference, and what constrains would we impose on the task-loss function. Answer on this question in the Report.

Appendix:

1. Multiclass perceptron update rule:

$$\hat{y} = \underset{y \in \{1,\ldots,k\}}{\operatorname{argmax}} \; \boldsymbol{w}^y \cdot \boldsymbol{x}$$

$$\text{if } \hat{y} \neq y_i :$$

$$\boldsymbol{w}^r \leftarrow \boldsymbol{w}^r + \tau_r \boldsymbol{x}_i$$

$$\tau_r = \begin{cases} +1 & \text{if } r = y_i \\ -1 & \text{if } r = \hat{y} \\ 0 & \text{otherwise} \end{cases} .$$

2. Structured perceptron update rule:

$$\hat{\mathbf{y}} = \arg\max_{\hat{\mathbf{y}}} \; \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_i, \hat{\mathbf{y}})$$

$$\boxed{\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\phi}(\mathbf{x}_i, \mathbf{y}_i) - \boldsymbol{\phi}(\mathbf{x}_i, \hat{\mathbf{y}})}$$

3. Dynamic programming for inference:

```
D_S = Matrix[label_size][num_of_eng_char] // use for scores
D_PI = Matrix[label_size][num_of_eng_char] // use to save the prev char index

// ================= INITIALIZATION ================= //
prev_char = '$' // get scores for the first letter where the perv is $
for i = 0; i < num-of-eng-char; i++
    curr_char = get char i+1
    y_hat = prev_char + "" + curr_char
    phi = build_phi(x, y_hat)
    s = W * phi
    D_S[0][i] = s
    D_PI[0][i] = 0
// ===================== RECURSION ===================== //
for i=1 ; i<label_size; i++
    for j=0 ; j<num-of-eng-char; j++
        curr_char = get char j+1
        tmp_d, d_best = -1, i_best = -1;
         d_best, i_best = max_y' (W * phi(x, y' * curr_char) + D_S[i-1][y'])
        D_S[i][j] = d_best;
        D_PI[i][j] = i_best;
// ============================================================ //
// ========================= BACK-TRACK ========================= //
y_hat = Vector[label_size]
d_best = -1
for i=0; i<num_of_eng_char ; i++
    if d_best < D_S[label_size-1][i]
        y_hat[label_size - 1] = i
        d_best = D_S[label_size-1][i]

for i=label_size-2; i>=0 ; i--
    y_hat[i] = D_PI[i + 1][y_hat[i + 1]]
// ============================================================ //
```