Computer Vision – 67542

MATLAB Assignment 3 – OCR with Structured Prediction

Preliminary Instructions

General

- Submission deadline: 15.5.2016.
- Late submission is allowed, but every day past the deadline will reduce 5 points from the assignment grade.
- The assignment is individual (i.e. you are not allowed to work in pairs or groups).
- Submit your solution through the course website. All of your files should be zipped in a folder named *matlab3_[ID].zip* ([ID] standing for your 9-digit ID).
- Your submission should include all .m files used during the assignment. Provide a brief description of each code file in a text file named *readme.txt*.
- Your code should be supplemented by a PDF file named *report.pdf*, which will include documentation of your results.
- The assignment consists of several mandatory parts and a bonus part. The number of points a part is worth is listed beside it.
- **Good luck!**

Datasets

In this assignment you will classify handwritten letters, first independently and then as part of multi-letter words (i.e. using context). The letters are supplied to you after going through feature extraction, i.e. they are not represented by images, but rather by vectors of feature measurements. Though outside the scope of this assignment, it is worthwhile mentioning that the feature extraction procedure was based on taking many HOG measurements, and selecting the most informative ones with a forward-greedy scheme.

In the folder *datasets* lie the two datasets which will be used throughout this assignment. Below are descriptions of them both.

letters.mat

This dataset contains single letters for training and testing. Namely, it contains a struct named *letters* which itself contains two structs – *train* and *test*. Each of the latter contains three fields:

- **X**: *m*-by-*d* real matrix where each row holds the feature measurements of a handwritten letter.
- **Y**: *m*-element vector of integers between 1 and 26 (number of letters in the English language). Each entry holds the label of the corresponding row in *X*.
- **char**: *m*-element cell-array where each entry holds the character corresponding to the label in the respective entry of *Y* ('a' if the latter holds 1, 'b' if it holds 2, and so on).

words.mat

This dataset contains multi-letter words for training and testing. Namely, it contains a struct named *words* which itself contains two struct-arrays – *train* and *test*. An entry in any of the latter corresponds to a single word, and holds the following fields:

- **X**: *T*-by-*d* real matrix where row *t* holds the feature measurements of letter *t* in a *T*-letter handwritten word.
- **Y**: *T*-element vector of integers between 1 and 26 (number of letters in the English language). Each entry holds the label of the corresponding row in *X*.
- **text**: String with *T* characters holding the word represented by *X*.

Note that the words in this dataset are synthetic, generally having no linguistic meaning.

Auxiliary Functions

Below are descriptions of the auxiliary functions provided to you, located in the root folder of the assignment.

conf_mat.m

```
cmat = conf_mat(Y, C, names)
```

Inputs

- **Y**: *m*-element vector of integers between 1 and *k* representing true labels of *m* instances.
- **C**: *m*-element vector of integers between 1 and *k* representing predicted labels of the *m* instances *Y* corresponds to.
- **names**: Optional. *k*-element cell-array of strings where each entry holds the name of the corresponding label.

Outputs

- **cmat**: Optional. *k*-by-*k* stochastic matrix where entry (*r*,*l*) holds the portion of instances with label *r* (according to *Y*) that were predicted to have label *l* (according to *C*). The prediction is perfect (*C* is identical to *Y*) if and only if this matrix holds the identity.

Description

According to true labels in *Y* and predicted labels in *C*, return confusion matrix in *cmat*. If the function is called with no output arguments, plot the confusion matrix instead (using label names specified by *names* if the latter is given).

Part 1 (10 pts)

Implement the following function:

Y = classify_letters(X, W)

Inputs

- **X**: *m*-by-*d* real matrix where each row holds the feature measurements of a different handwritten letter to classify.
- **W**: *k*-by-*d* real matrix representing a letter-classifier (rows correspond to classes, columns correspond to feature measurements).

Outputs

- **Y**: *m*-element column vector of integers between 1 and *k*, where each entry holds the predicted label (according to *W*) of the corresponding row in *X*.

Description

The function uses the letter-classifier represented by *W* to predict the labels of the rows in *X*. This is done with the following linear classification rule:

$$\hat{y}(x) = \arg\max_{1 \le r \le k} \langle W_r, x \rangle$$

where $x \in \mathbb{R}^d$ stands for the *X*-row to classify and $W_r \in \mathbb{R}^d$ stands for the *r*'th row of *W*.

Part 2 (15 pts)

Suppose we would like to learn a letter-classifier $W^* \in \mathbb{R}^{k,d}$ via minimization of the regularized hinge-loss:

$$W^* = \arg\min_{W \in \mathbb{R}^{k,d}} \frac{1}{m} \sum_{i=1}^{m} \max_{1 \le r \le k} \left\{ \mathbf{1}[r \ne y_i] + \langle W_r, x_i \rangle - \langle W_{y_i}, x_i \rangle \right\} + \lambda \sum_{r=1}^{k} \|W_r\|^2$$

Here $\{(x_i, y_i)\}_{i=1}^{m} \subset \mathbb{R}^d \times [k]$ are training instances, $W_r \in \mathbb{R}^d$ is the *r*'th row of *W*, and $\lambda \ge 0$ is the regularization parameter.

Write down explicitly the Stochastic Gradient Descent (SGD) algorithm as applied to the above problem. Assume an initialization value of zeros, a step size of $(\lambda t)^{-1}$ at iteration *t*, and *T* overall iterations.

Part 3 (15 pts)

Implement the following function:

W = letter_hinge_SGD(X, Y, lambda, T)

Inputs

- **X**: *m*-by-*d* real matrix where each row holds the feature measurements of a different handwritten letter.
- **Y**: *m*-element vector of integers between 1 and *k*, where each entry holds the label of the respective row in *X*.
- **lambda**: Non-negative scalar. Regularization parameter.
- **T**: Positive integer. Number of SGD iterations to carry out.

Outputs

- **W**: *k*-by-*d* real matrix representing the letter-classifier (rows correspond to classes, columns correspond to feature measurements) learned via SGD-minimization of regularized hinge-loss.

Description

The function implements the algorithm you have specified in Part 2. Namely, it learns a letter-classifier via SGD-minimization of the regularized hinge-loss.

Part 4 (15 pts)

Use the function *letter_hinge_SGD* (from Part 3) to train a letter-classifier on the **training** set of *letters.mat* dataset. Then, use *classify_letters* (Part 1) to measure the error (average 0-1 loss) of the trained classifier on both the training and the testing sets of *letters.mat*.

Repeat this for different values of the regularization parameter λ, and plot the train and test errors as a function of λ. Attach this plot to your report, and explain it in detail. In particular, explain how it can be used to select an optimal value of λ (make sure the values λ ranges over support this selection criteria). For the selected λ-value, use the auxiliary function *conf_mat* to produce a visualization of the confusion matrix on the test set. Attach this visualization to your report and explain it.

Part 5 (30 pts)

Implement the following function:

Y = classify_word(X, W, A)

Inputs

- **X**: *T*-by-*d* real matrix where row *t* holds the feature measurements of letter *t* in a *T*-letter handwritten word.
- **W**: *k*-by-*d* real matrix representing a word-classifier (rows correspond to classes, columns correspond to feature measurements).
- **A**: *k*-by-*k* real (stochastic) matrix holding letter transition probabilities - entry (*r*,*l*) holds the probability that a label of a letter is *l* given that the label of the preceding letter is *r*.

Outputs

- **Y**: *T*-element column vector of integers between 1 and *k*, where each entry holds the predicted label (according to *W*) of the corresponding row in *X*.

Description

The function uses the word-classifier represented by *W* to predict the labels of the rows in *X*. The rows of *X* are classified jointly (the interaction is defined by the transition matrix *A*), with the following prediction rule:

$$\left(\hat{y}_1,\ldots,\hat{y}_T\right)=\underset{1\le r_1,\ldots,r_T\le k}{\arg\max}\left\{\sum_{t=1}^{T}\left\langle W_{r_t},x_t\right\rangle+\sum_{t=1}^{T-1}A_{r_t,r_{t+1}}\right\}$$

where x_t stands for the *t*'th row of *X*, W_{r_t} stands for the r_t'th row of *W*, $A_{r_t,r_{t+1}}$ stands for entry $\left(r_t,r_{t+1}\right)$ of *A* and $\hat{y}_t$ stands for the predicted label of x_t.

Implementation Guidelines

A naïve implementation of the above prediction rule would consider all possibilities for $\left(r_1,\ldots,r_T\right)\in\left[k\right]^T$, i.e. it would go through a discrete search space of size k^T. For a word consisting of eight (*T* = 8) lower case English letters (*k*=26), this amounts to scanning over $2\cdot 10^{11}$ possibilities – clearly not acceptable for classification of a single word. This calls for a more sophisticated approach. One option, which you have seen in class, is dynamic programming. In particular, you have seen a dynamic programming scheme that enables execution of the above prediction rule with time complexity $O\left(T\cdot k^2\right)$. Base your implementation of the function on this scheme.

Part 6 (15 pts)

In this part you will use the letter-classifier learned in Part 4 and a given pre-trained word-classifier, to classify the test instances of *words.mat* dataset. You are not required to train a word-classifier here – this will be done in the Bonus Part.

Denote by W_{ltr} the optimal letter-classifier selected in Part 4, and by W_{wrd} the word-classifier given in the file *W_wrd.mat* (resides in the root folder of the assignment). With each of the following strategies, classify the test instances of *words.mat*:

1. Letter classification with W_{ltr}
2. Letter classification with W_{wrd}
3. Word classification with W_{ltr}
4. Word classification with W_{wrd}

Letter classification is carried out with the function *classify_letters* from Part 1. Word classification is carried out with the function *classify_word* from Part 5, where the transition probability matrix is taken from the file *A.mat* (resides in the root folder of the assignment).

For each of the above classification strategies, measure the average 0-1 loss on **letters**. For example, a word 'station' classified as 'statlqn' will account for 5 correct letter classifications ('s', 't', 'a', 't', 'n') and 2 incorrect letter classifications ('i' classified as 'l', 'o' classified as 'q'). Note that this measure is different from the 0-1 loss on words, which in the latter example would simply count a single mistake (the word 'station' was not correctly classified).

Compare the performance of classification strategies 1 and 2 above. For each strategy, list a few words (if there are any) that were correctly classified with it and were incorrectly classified with the other strategy. Explain the results in detail.
Repeat this process twice more: once for comparing strategy 3 to strategy 4, and once for comparing strategy 1 to strategy 4.

Bonus Part (20 pts)

Task I

Implement the following function:

Z = word_hinge_maximizer(X, Y, W, A)

Inputs

- **X**: *T*-by-*d* real matrix where row *t* holds the feature measurements of letter *t* in a *T*-letter handwritten word.
- **Y**: *T*-element vector of integers between 1 and *k*, where each entry holds the true label of the respective row in *X*.
- **W**: *k*-by-*d* real matrix representing a word-classifier (rows correspond to classes, columns correspond to feature measurements).
- **A**: *k*-by-*k* real (stochastic) matrix holding letter transition probabilities – entry (*r*,*l*) holds the probability that a label of a letter is *l* given that the label of the preceding letter is *r*.

Outputs

- **Z**: *T*-element column vector of integers between 1 and *k*, holding the *X*-rows' labeling for which the word hinge-loss maximum is attained.

Description

The function returns the labeling which attains the maximum in the hinge-loss of the word-classifier *W* with respect to the given labeled word (*X*,*Y*). Namely, it assigns its output as follows:

$$
\begin{aligned}
(z_1,\ldots,z_T) &= \arg \overbrace{\max_{1\le r_1,\ldots,r_T\le k}\left\{\sum_{t=1}^{T}\mathbf{1}[r_t\neq y_t]+\left(\sum_{t=1}^{T}\langle W_{r_t},x_t\rangle+\sum_{t=1}^{T-1}A_{r_t,r_{t+1}}\right)-\left(\sum_{t=1}^{T}\langle W_{y_t},x_t\rangle+\sum_{t=1}^{T-1}A_{y_t,y_{t+1}}\right)\right\}}^{l_A^{hinge}(W;X,Y)} \\
&= \arg\max_{1\le r_1,\ldots,r_T\le k}\left\{\sum_{t=1}^{T}\mathbf{1}[r_t\neq y_t]+\sum_{t=1}^{T}\langle W_{r_t},x_t\rangle+\sum_{t=1}^{T-1}A_{r_t,r_{t+1}}\right\}
\end{aligned}
$$

where:

- x_t - row *t* of *X*
- y_t - entry *t* of *Y*
- z_t - entry *t* of *Z*
- W_r - row *r* of *W*
- $A_{r,l}$ - entry (*r*,*l*) of *A*
- $\sum_{t=1}^{T}\mathbf{1}[r_t\neq y_t]$ - cost of labeling $(r_t)_{t=1}^{T}$ (Hamming distance from true labeling $(y_t)_{t=1}^{T}$)
- $\sum_{t=1}^{T}\langle W_{r_t},x_t\rangle+\sum_{t=1}^{T-1}A_{r_t,r_{t+1}}$ - score *W* gives to labeling *X* as $(r_t)_{t=1}^{T}$
- $\sum_{t=1}^{T}\langle W_{y_t},x_t\rangle+\sum_{t=1}^{T-1}A_{y_t,y_{t+1}}$ - score *W* gives to labeling *X* as $(y_t)_{t=1}^{T}$ (true labeling)

Implementation Guidelines

As in Part 5, a naïve implementation of the function is impractical (requires scanning a discrete search space of size k^T), leading us to adopt a dynamic programming approach. Modify the dynamic programming scheme from Part 5 to adapt to the maximization problem faced here, and base your implementation of the function on this modified scheme.

Task II

Use the function *word_hinge_maximizer* (from Task I) to implement the following function:

W = word_hinge_SGD(X_arr, Y_arr, A, lambda, T)

Inputs

- **X_arr**: *m*-element cell array where cell *i* contains a S_i-by-*d* real matrix with rows holding feature measurements of letters in a S_i-letter handwritten word.
- **Y_arr**: m-element cell array where cell *i* contains a S_i-element vector of integers between 1 and *k* holding the labels of the rows in the matrix *X_arr{i}*.
- **A**: *k*-by-*k* real (stochastic) matrix holding letter transition probabilities – entry (*r*,*l*) holds the probability that a label of a letter is *l* given that the label of the preceding letter is *r*.
- **lambda**: Non-negative scalar. Regularization parameter.
- **T**: Positive integer. Number of SGD iterations to carry out.

Outputs

- **W**: *k*-by-*d* real matrix representing the word-classifier (rows correspond to classes, columns correspond to feature measurements) learned via SGD-minimization of regularized hinge-loss.

Description

The function learns a word-classifier via SGD-minimization of the regularized hinge-loss. Namely, it applies SGD with an initialization value of zeros, a step size of $(\lambda t)^{-1}$ at iteration *t* and *T* overall iterations, to optimize the following objective:

$$\frac{1}{m}\sum_{i=1}^{m}\overbrace{\max_{1\le r_1,\dots,r_{S_i}\le k}\left\{\sum_{s=1}^{S_i}\mathbf{1}\left[r_s\neq y_s^i\right]+\left(\sum_{s=1}^{S_i}\left\langle W_{r_s},x_s^i\right\rangle+\sum_{s=1}^{S_i-1}A_{r_s,r_{s+1}}\right)-\left(\sum_{s=1}^{S_i}\left\langle W_{y_s^i},x_s^i\right\rangle+\sum_{s=1}^{S_i-1}A_{y_s^i,y_{s+1}^i}\right)\right\}}^{l_A^{hinge}\left(W;X\_arr\{i\},Y\_arr\{i\}\right)}+\overbrace{\lambda\sum_{r=1}^{k}\left\|W_r\right\|^2}^{\substack{regularization\\term}}$$

where:

- x_s^i - row *s* of *X_arr{i}*
- y_s^i - entry *s* of *Y_arr{i}*
- W_r - row *r* of *W*
- $A_{r,l}$ - entry (*r*,*l*) of *A*
- λ - *lambda*

Implementation Guidelines

Notice that $l_A^{hinge}\left(W;X\_arr\{i\},Y\_arr\{i\}\right)$ – the hinge-loss of *W* with respect to the labeled word (*X_arr{i}*,*Y_arr{i}*), is defined to be the maximum of affine functions in *W*. Thus, to compute a subgradient (w.r.t. *W*) of this hinge-loss at a specific *W*, one needs only to find the maximizing affine component (via *word_hinge_maximizer* function from Task I), and take the gradient of that component.

Once the computation of a subgradient of $l_A^{hinge}\left(W;X\_arr\{i\},Y\_arr\{i\}\right)$ is implemented, little work is required to complete the SGD algorithm.

Task III

Use the function *word_hinge_SGD* (from Task II) to train a word-classifier on the **training** set of *words.mat* dataset. Then, use *classify_word* (Part 5) to measure the error (average 0-1 loss on **letters** – see Part 6 for details) of the trained classifier on both the training and the testing sets of *words.mat*. The transition probability matrix to use during training and evaluation is given in the file *A.mat* (resides in the root folder of the assignment).

Repeat the above process for different values of the regularization parameter λ, and plot the train and test errors as a function of λ. Attach this plot to your report, and explain it in detail. In particular, explain how it can be used to select an optimal value of λ (make sure the values λ ranges over support this selection criteria). For the selected λ-value, use the auxiliary function *conf_mat* to produce a visualization of the confusion matrix on the letters of the test set. Attach this visualization to your report and explain it. In particular, compare it to the confusion matrix obtained in Part 4.