Exercise 5/6: Classification with Logistic Regression

Lecture Information Processing and Communication

Jörn Anemüller, May 2022

Submit solutions until Tuesday 2022-05-31, 23:59h, by uploading to your group's exercise folder on cs.uol.de. You may submit your solutions in groups of at most two students. You are free to write your code in matlab or in python (but we provide the example functions in matlab only). Note that this exercise covers two weeks due to Christ Himmelfahrt public holiday on May 26th. Also note that this exercise gives double points.

Note: Part 9 of this exercise sheet –regularization– has been postponed to one of the following exercise sessions (due to illness of faculty). You do not need to work on part 9 for the present solutions!

Summary of exercise 5/6

The goal of this exercise is to implement a logistic regression classification algorithm. Use the method from the lecture to implement the logistic regression algorithm, using the gradient descent method with the gradient equation provided. The functions are organized largely in analogy to exercise 4 (least-squares method), thus, you may wish to build on your previous code to fill in the gaps in the provided matlab skeleton.

List of functions to be completed

`ex5_script.m`: main script for completion of exercise 5/6

`add_dummy_ones.m`: add a row of ones as a first row to the data matrix X in order to get rid of the bias term

`matrix2vector.m`: convert data from 2-dimensional matrix form into a 1-dimensional feature vector

`vector2matrix.m`: convert back from 1-dimensional feature vector to 2-dimensional matrix

`compute_loss_logreg.m`: computes the cross-entropy loss function for the logistic regression task

`compute_gradient_logreg.m`: computes the derivative of the loss function for logistic regression, returns the value of the derivative and the value of the loss function

`gradient_descent_logreg.m`: performs iterative gradient descent optimization for logistic regression

`compute_accuracy.m`: compute the classification accuracy, i.e., number of correctly classified images divided by total number of images

List provided helper functions

`mnistRead.m`: Function from the handwritten postal digits dataset from `http://yann.lecun.com/exdb/mnist/`. It loads images and labels for training and testing. Data are stored in the files `train-images-idx3-ubyte.gz`, `train-labels-idx1-ubyte.gz`, `t10k-images-idx3-ubyte.gz` und `t10k-labels-idx1-ubyte.gz`.

`mnistShow.m`: Function to display digits from the dataset

1. Logistic regression gradient

Logistic regression is based on the cross-entropy loss function L,

$$L(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N} y^{(n)} \cdot \log(\hat{y}^{(n)}) + (1 - y^{(n)}) \cdot \log(1 - \hat{y}^{(n)}),$$

where, as usual, the true class labels are given as $y^{(n)}$, chosen as binary labels $y^{(n)} \in \{0, 1\}$. The estimated class-label is given by $\hat{y}^{(n)}$; more specifically, $\hat{y}^{(n)}$ denotes the estimated probability of class 1 being present. Hence, $y^{(n)} = 1$ ist estimated to be true with probability $\hat{y}^{(n)}$, and $y^{(n)} = 0$ is estimated to be true with probability $(1 - \hat{y}^{(n)})$.

Here, $\hat{y}^{(n)}$ is estimated by a linear model with subsequent logistic non-linearity:

$$\hat{y}^{(n)} = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x}^{(n)})}.$$

The gradient of $L(\mathbf{w})$ with repect to $\mathbf{w}$ is given by

$$\nabla_{\mathbf{w}} L(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} (\hat{y}^{(n)} - y^{(n)})\, \mathbf{x}^{(n)}.$$

Check out how gradient descent optimization works and implement gradient descent for logistic regression in this exercise.

2. Main script for this exercise session

Edit the script `ex07_script.m`, it contains template code for the main steps of this exercise.

3. Loading image data

The mnist dataset is read by the function call `[train_images, train_labels, test_images, test_labels] = mnistRead()` which returns matrices and vectors for training and test images and labels.

4. Converting data matrix to feature vector

Convert each image in the multi-dimensional data array into a feature vector that is used for optimization. Simple reshaping of the data does the job, and should be performed for each image matrix stored in the multi-dimensional data array. The corresponding inverse transformation is needed to visualize the obtained weight vector.

```
function [mat_features, NCol, NRow, NFrame] = matrix2vector(mda_spectrogram)
function mat_out = vector2matrix(vec_in, NCol, NRow)
```

5. Adding a row of dummy ones to the data to avoid offset term

Prepend a row of ones to the matrix containing the feature vectors in order to get rid of the bias term.

Function definition:

```
function mat_X = add_dummy_ones(mat_X)
```

6. Perform gradient descent optimization with logistic regression and visualize the results

Find the optimum weight vector for the given data. Finally, visualize your results. Plot the evolution of the prediction error across gradient descent iterations and the evolution of some (interesting) example weights. Reshape the optimum weight vector to matrix and visualize it to facilitate interpretation in terms of spectro-temporal neuronal sensitivity.

Function definition:

```
function [vec_L, mat_W, vec_w_opt] = gradient_descent_logreg(max_iter,
step_size, vec_w_init, mat_X, vec_y)
```

7. Compute the classification accuracy of the trained classifier on the training and test data

To evaluate the learning success of the classifier, we use the accuracy ("percent correct") of images that have been assigned to the correct class. Do this both on the train and the test set in order to see whether the classifier successfully generalizes from training data to new, previously unseen data. Compare both results.

Function definition:

```
function accuracy_train_logreg = compute_accuracy(vec_y, vec_y_hat);
```

8. Classify previously unseen data from the test set portion of the dataset, but use the model you trained on the training data

To compare classification accuracy on the train and test set, you first need to take the model (i.e., weight vector) obtained during training and apply it to the test portion of the data. Look also at example plots of digits from the test data and at the estimated class labels.

Function definition:

```
function vec_y_hat_test = classify_logreg(vec_w_opt_ls, mat_X_test);
```

9. Add a regularization term to the logistic regression gradient and perform gradient descent optimization for different parameter values of the regularization parameter λ

Regularization based on the l2-norm of the weight vector $\mathbf{w}$essentially adds the l2-norm of the weight vector to loss function $L(\mathbf{w})$ which results in the partial derivative (i.e., the gradient's components) of

$$\frac{\partial}{\partial w_i} L(\mathbf{w}) = \sum_{n=1}^{N} (\hat{y}^{(n)} - y^{(n)})\, x_i^{(n)} + \frac{\lambda}{N} w_i.$$

Note that this expression should be used only for $i = 1, \ldots, D$ but not for the bias-term w_0. For hints, refer to, e.g., `https://towardsdatascience.com/implement-logistic-regression-with-l2-regularization-from-`

Rerun your logistic regression algorithm from above with regularization for several values of the regularization parameter λ.

Plot the "digit"-image-representation corresponding to the weight vector and observe how it changes across the different λ-values.