TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Datenverarbeitung
PD Dr. Martin Kleinsteuber

Information Retrieval in High Dimensional Data

Lab #4, 3.5.2018

Logistic Regression

Task 1. Recall the data processing routines from the last lab course. The following excercises build on top of the extracted feature representations, but instead of the prebuilt classifier, we want to implement logistic regression by hand, i.e. by minimizing $F(\mathbf{w})$ from Chapter 3 in the lecture notes. To this end, make sure, that the variables `train`, `test`, `train_data_features` and `test_data_features` are loaded to your IPython shell.

a) Write a PYTHON function `logistic_gradient` that expects a training set matrix `X_train`, a ground truth label vector `y_train` and a current weight vector `w` as its input and returns the gradient `g` of the negative log-likelihood function of the logistic regression. Refer to the lecture notes for the mathematical definition.

b) Write a PYTHON function `find_w` that expects a training set matrix `X_train`, a ground truth label vector `y_train`, a step size `alpha` and a maximum iteration number `max_it` that determines the optimal logistic regression weight vector `w_star` by performing gradient descent, i.e. calling `logistic_gradient` in each iteration. Make sure to include the affine offset $w_0 = b$ into your model.

c) The dataset at hand is quite large. Applying standard gradient descent could likely lead to Python throwing a `MemoryError` exception. To avoid this, we will employ a variation of *stochastic gradient descent* that has been proven successful in training deep neural networks. In *minibatch learning*, each iteration of the algorithm is replaced by a so-called *epoch*. In each epoch the training set is divided randomly into subsets of equal size, the minibatches. For each minibatch, the gradient is computed and applied only with respect to the samples in the minibatch. An epoch is finished when the gradient step was performed for each minibatch. Modify `find_w` such that it is capable of mini-batch learning. You will need to replace `max_it` by `n_epochs` and add the parameter `n_minibatch` to your function definition. Note: take care of normalization of the gradient and the loss function.

d) Write a function `classify_log` that expects a weight vector `w` and a test set matrix `X_test` and classifies the samples in `X_test` via logistic regression, returning a label vector `y_test`. Test your implementation of `find_w` and `classify_log` on `train_data_features` and `test_data_features` with 10 epochs, minibatches of size 100 and step size `alpha=1`.

e) Logistic regression is prone to *overfitting*. To prevent this, *regularizers* can be used. Adjust your implementation in such a way that instead of minimizing $F(\mathbf{w})$, it minimizes the term

$$F(\mathbf{w}) + \lambda \|\mathbf{w}\|^2, \tag{1}$$

where λ is a non-negative regularization parameter. Test your implementation with $\lambda = 10^{-3}$. Did the results improve? Why/why not?

Helpful Python/Numpy functions

`np.diag(x)`	creates diagonal matrix with x on diagonal
`np.exp(X)`	exponential function