582744 Advanced Course in Machine Learning

Project work **Version 1.1**

Instructions:

1. To complete the course using the *separate exam* you need to have either solved the exercise problems during the lecture period OR you need to solve this set of small projects. There is no need to solve the project if you already solved the weekly exercises.
2. You need to solve all four parts of the project, or at least seriously attempt solving them. In case you cannot solve some part satisfactorily, please explain your attempt as clearly as you can so that your effort can be properly evaluated. The solutions will be graded and you need to obtain at least 50% of the available points to pass the course. Each of the four parts is worth equally many points.
3. You should return the solutions within 7 weeks of the exam.
4. Your submission is composed of two parts: (a) A single PDF file containing your answers to all questions, including all plots and discussion. (b) a single compressed file (zip or tar.gz) containing your code (and nothing else).
5. Submit the result by sending an email for the lecturer (`arto.klami@cs.helsinki.fi`). It is also recommended to send him an email when you *start* working on the project, so that in case there are any updates for the problems he can inform you.
6. Please typeset your work using appropriate software such as LaTeX. However, there is no need to typeset the pen and paper answers – you can also include a scanned hand-written version.
7. You can solve the programming tasks in any reasonable language. The recommended alternatives are Python and R.
8. Pay attention to how you report the results. The project problems are fairly open-ended, which means you sometimes have to make decisions regarding the details. Always explain the decisions you made and give a brief justification for the choice. Remember to also include necessary plots and tables as part of your report, and always discuss the findings. Try to make conclusions.
9. In case you have any questions regarding the problems or think there is a mistake of some sort, please contact Arto by email or visit in room A311 during regular working hours.

1 Gradient-based optimization

Considerable number of machine learning algorithms use some form of gradient-based optimization. In this problem you will implement a small library for studying various gradient-based optimization algorithms. You will experiment with the *logistic regression* model (Section 8), but should write the code so that any other (simple) model could be plugged in as well. That is, you should write the code for performing the actual optimization so that the loss function and the gradient are provided via separate functions.

1. Create functions that compute the negative logarithmic likelihood (Equation 8.2) and its gradient (Equation 8.5). Your functions should support also l_2 regularization for the weights, with user-specific regularization parameter.

2. Write a function for performing standard gradient descent (Section 8.3.2) using fixed step size.

3. Write function for performing stochastic gradient descent (Section 8.5). Your code should support mini-batches for evaluating the gradient.

4. Implement the Adam algorithm for automatic adaptation of the step-length. See https://arxiv.org/pdf/1412.6980.pdf for description of the algorithm. This algorithm was not covered during the lecture course, but is included here as an exercise that requires reading and understanding a scientific paper.

Now use the software written above to study the different optimization algorithms and regularization techniques on the MNIST data, a collection of images of hand-written digits. The helper package available at http://www.cs.helsinki.fi/u/sorkhei/mnist.tar.gz contains scripts for reading and visualizing the data. Since logistic regression is a binary classifier you can solve the following problems for a simplified task where only digits '5' and '6' are used.

To study overfitting you need to split the data into three separate parts. One part is used for training the model ("training data"), another for validating the results and for selecting regularization parameters ("validation data"), and the final one for evaluating the final results ("test data"). You should not access the test data until the very last stage when you test the final model.

1. Run standard gradient descent with some suitable step-size and plot the training and validation losses as a function of the iteration. How do they behave?

2. Run standard gradient descent until convergence for a range of step-sizes. Plot the final training and validation errors as a function of the step-size. Describe your findings.

3. Repeat steps 1 and 2 for stochastic gradient descent, using some suitable mini-batch size. Note that the step-size should decrease during optimization (see Section 8.5.2.1).

4. Run SGD with Adam for controlling the step size adaptively and compare the results to the SGD results obtained above. Does it work? In case you were not able to implemente Adam, you can use AdaGrad instead – see the model solutions of Exercise 2 of the lecture course.

5. Now study regularization, using one of the above algorithms. Use a small subset of samples for training and plot the final training and validation losses for a collection of regularization parameter values. Do the same for two other sizes of training data, to see how the importance of regularization depends on the available data. Describe your findings.

6. Try early-stopping. That is, stop the optimization process when the validation error starts to increase.

7. Finally, evaluate the following models on the test data and report the results: (1) Unregularized model trained until convergence, (2) The regularized model with the regularization parameter that provided the best validation loss, trained until convergence, (3) Unregularized model trained with early-stopping. Which one is the best?

8. Plot some examples that were classified incorrectly (the helper package has functions for this). Do they look unusual? Plot also some examples for which the classifier gave very high confidence – do they look like prototypical '5's and '6's?

2 Multi-layer perceptron

Multi-layer perceptron (MLP) is a feedforward neural network, described in detail in Section 16.5 of the course book. In this exercise you will implement a simple MLP and evaluate it in solving a regression task.

1. Implement multi-layer perceptron that supports at least one layer of hidden units. The hidden layer(s) should use the sigmoid activation function and the output layer should be linear. Your network should include also bias terms for each of the nodes.

2. Verify that your code computes the output correctly if initialized with some suitable values for the weights. That is, compare the outputs of some (very) small network against ones computed by hand.

3. Implement the backpropagation algorithm described in Section 16.5.4, assuming the loss function is the squared error. That is, write code that computes the derivatives of the loss with respect to each of the weights (and the bias terms) in the network. To verify your gradient computation is correct, you can compare it against the finite differences; see http://ufldl.stanford.edu/wiki/index.php/Gradient_checking_and_advanced_optimization for instructions.

4. Train the network to minimize the squared error for the PROBLEM2_TRAIN.CSV data. It has 10 real-valued input features and one real-valued output variable (the last column)[1]. Use some gradient-based optimization algorithm to minimize the training loss, plotting both the training and validation losses (PROBLEM2_VALID.CSV) as function of the iteration. Try out a few alternative choices for the number of hidden units (and layers if your implementation supports more than one layer) and describe the main findings.

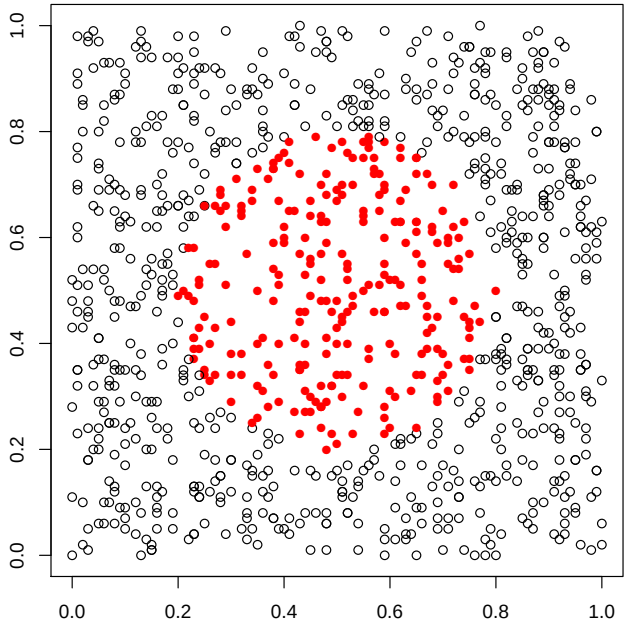
5. Implement the same network using the TensorFlow library. Download and install the library from https://www.tensorflow.org/install (Hint: To avoid messing up with your other installed packages, you should use Virtualenv, Anaconda or Docker) and follow the tutorials at https://www.tensorflow.org/get_started/get_started to understand how a simple model like MLP can be implemented. Run the model on the same data as above and compare the results against the model you implemented from scratch. Do they match? Which one is faster? Which one required less development effort?

[1] The data is a modified version of the BlogFeedback data available at https://archive.ics.uci.edu/ml/datasets/BlogFeedback. Our version of the data contains only features 51, 52, 53, 56, 57 58, 61, 62, 277, and 280. Furthermore, both inputs and outputs were transformed with $\log(x+1)$ to compress the scales.

3 AdaBoost

AdaBoost is an algorithm for learning an ensemble of weak learners. A weak learner is a classifier that performs better than random guessing but not necessarily by much, and boosting is a technique for constructing a strong classifier by combining multiple weak learners.

In this problem you will use AdaBoost (Section 16.4.3) to classify the simple two-dimensional dataset depicted below (and available in the file PROBLEM3_DATA.CSV, using linear classifiers. Use the first 500 of these for training and the rest for validation. The first two columns are the inputs and the last column is the class label.



1. Implement the AdaBoost algorithm. Your implementation should support any choice of weak learner (that supports weighted inputs), and hence the basic implementation only needs to be able to compute errors, re-weight the samples based on the error, compute the weights for the base learners, and perform the final classification.

2. Use AdaBoost with arbitrary linear classifiers to solve the problem. You can either train each classifier until convergence, of you can just run a couple of gradient-based iterations – this should be enough to make them better than random. Here a linear classifier can be the logistic regression implemented in Problem 1 (but note that you need to add support for weighted data points), or you can use the perceptron algorithm (Section 8.5.4).

3. Plot the training and validation errors as a function of the number of base learners added for the ensemble. Describe the findings. Try also drawing the individual decision boundaries on top of the data scatter plot. Do they look reasonable?

4. Try solving the same problem with linear classifier restricted to pass though the origin (the bottom left corner in the figure above) by assuming the bias term is zero, again plotting the errors and the decision surfaces. Can you still reach small training error? Do you need more base learners than in the previous subtask?

4 Theoretical exercises

Solve the following exercise problems using pen and paper; no coding is required or expected. The problems are mathematically not very challenging and can often be solved with a few simple steps once you understand the key idea – if you find yourself performing complicated mathematical operations that require considerable amount of time then you are probably doing something wrong.

Most of the problems are taken from the course book. Please read the exact description in the book; the brief summaries below are only to make sure you look at the right problem.

1. Exercise 3.1: MLE for Bernoulli model.
2. Exercise 5.3: Reject option in classifiers.
3. Exercise 11.7: Manual calculation of the M step for a GMM.
4. Exercise 12.5: Deriving the residual error for PCA.
5. Exercise 13.12: Subderivative of the hinge loss function.
6. In Problem 1 we considered l_2-regularized logistic regression. Section 13.3.2 of the coursebook describes a coordinate desccent algorithm for l_1-regularized linear regression (called lasso). Explain how the algorithm presented for lasso could be applied for creating l_1-regularized logistic regression. Compute the necessary gradients and simplify them as far as you can. Would this be a good algorithm?
7. The polynomial kernel is usually defined with $k_{ij} = (\gamma + \mathbf{x}_i^T \mathbf{x}_j)^p$ for some positive *gamma* (often $\gamma = 1$). Another alternative would be to use the kernel $k_{ij} = (\mathbf{x}_i^T \mathbf{x}_j)^p$, corresponding to $\gamma = 0$. Write down the feature maps $\psi(\mathbf{x})$ for both alternatives for $p = 2$ and explain the main difference. What kind of an effect the additive constant has for the kernel values?
8. Consider the regularized loss function $L(\theta_1, \theta_2) = (\theta_1 - 2)^2/4 + (\theta_2 - 2)^2 + \lambda|\boldsymbol{\theta}|$ (l_1-regularization), where $\boldsymbol{\theta} = [\theta_1, \theta_2]$. Characterize the sparsity of the solution as a function of the regularization parameter λ. That is, find out for which values of λ the optimal solution has two non-zero elements, for which values it has one non-zero element, and for which values the result is the zero vector.