Programming Project 2 – Classification Using Linear Discriminant Functions and Boosting Algorithms

CAP 5638, Pattern Recognition, Fall 2017

Department of Computer Science, Florida State University

Points: 100

Due: Tuesday, December 5, 2017

Maximum Team Size: 2

Submission: Hardcopy (including programs) is required and is due at the beginning of the class on the due date. Only one copy from each team is required.

Background: Bayesian decision theory provides the optimal decision rule for classification when the true probabilities are known. However, for pattern classification applications, the final product we need is a classifier that can be represented by a set of discriminant functions. Therefore, if we learn discriminant functions directly, we can avoid the intermediate step of estimating probability models, which arguably is more difficult than learning discriminant functions with finite training data (note that by doing this we lose the first principle and many techniques are thus ad hoc).

Among the discriminant functions, linear ones are widely used because they can be learned efficiently and can often be analyzed analytically. Yet through kernel methods and boosting algorithms, they can lead to accurate classifiers for complex, real-world applications.

Purpose: To know how to learn two-class/multi-class linear discriminant functions through perceptron-like algorithms and how to use boosting algorithms to create accurate classifiers using linear discriminant functions.

Assignment: Implement programs to do the following.

1) **Basic two-class classification using perceptron algorithms**
   Abstractly, the problem is as follows. Given *n* labeled training samples, $D=\{(x_1,L_1), (x_2, L_2), \ldots, (x_n, L_n)\}$, when Li is +1 / -1, implement Algorithm 4 (Fixed-Increment Single-Sample Perceptron Algorithm) and Algorithm 8 (Batch Relaxation with Margin) of Chapter 5 in the textbook.
2) **Multi-class classification**
   Use the basic two-class perceptron algorithms to solve multi-class classification problems by using the one-against-the-rest and one-against-the-other methods. Note that you need to handle ambiguous cases properly.
3) **Adaboost to create strong classifiers**
   Implement Algorithm 1 (AdaBoost) in Chapter 9 of the textbook to create a strong classifier using the above linear discriminant functions.

After programming is done, apply your programs on the following two datasets.

- UCI wine dataset
  The dataset is divided into a training set (http://www.cs.fsu.edu/~liux/courses/cap5638/assignments/wine_uci_train.txt) and a test set (http://www.cs.fsu.edu/~liux/courses/cap5638/assignments/wine_uci_test.txt). The training set consists of 89 examples in three classes (30 in class 1, 35 in class 2 and 24 in class 3) and the test set consists also of 89 examples (29 in class 1, 36 in class 2, and 24 in class 3). A description of the dataset is available at http://www.cs.fsu.edu/~liux/courses/cap5638/assignments/wine_uci_desc.txt.

- USPS handwritten digit dataset
  Similar to the UCI wine dataset, it is also divided into a training set (http://www.cs.fsu.edu/~liux/courses/cap5638/assignments/zip_train_0_2.txt) and a test set (http://www.cs.fsu.edu/~liux/courses/cap5638/assignments/zip_test_0_2.txt). The training set consists of 2930 training samples (1194 in digit 0, 1005 in digit 1, and 731 in digit 2) and the test set consists of 821 samples (359 in digit 0, 264 in digit 1, and 198 in digit 2). A description of the dataset is available at http://www.cs.fsu.edu/~liux/courses/cap5638/assignments/zip_info.txt. For non-CS students who do not have sufficient computational resources, you can use a subset of the training (http://www.cs.fsu.edu/~liux/courses/cap5638/assignments/zip_train_0_2_small.txt (84 for digit 0, 59 for digit 1, and 57 for digit 2 with a total of 200 training examples)), and a subset of the test set (http://www.cs.fsu.edu/~liux/courses/cap5638/assignments/zip_test_0_2_small.txt (109 for digit 0, 117 for digit 1, and 74 for digit 2 with a total of 300 test samples)).

For each dataset, you need to do the following:

- **Classification using the two two-class algorithms**
  For each dataset, train a classifier to classify class 1 against the rest, class 2 against the rest, and class 3 against the rest by applying the two algorithms on the training set and then use the trained classifier on the test set (with labeling consistent with the training labeling). Document classification accuracy and iterations in training, and compare the two different algorithms. Note that USPS digit dataset may not be linearly separable and you then need to stop the algorithms in some way.

- **Multi-class classification**
  For each dataset, now train a classifier to classify all the classes using the one-against-the-rest and the one-against-the-other methods based on the two two-class algorithms, resulting in four different classifiers on each dataset and then classify the test set. Document classification accuracy, iterations in training, and classification time for test, and compare the one-against-the-rest and the one-against-the-other methods.

- **AdaBoost**
  Boost Algorithm 8 to create a strong classifier for class 1 vs. class 2, class 1 vs. class 3, and class 2 vs. class 3 on the two datasets. Then classify the corresponding test samples from the relevant classes in test sets (in other words, for example, for the class 1 vs. class 2 classifier, you only need to classify test samples from classes 1 and 2); then document classification accuracy and show and analyze the improvement.

Submission:

- **Report** – You need to turn in a report, including description of your algorithms and implementations, and answers to the above questions.
- **Source code** – You should attach all the source programs you developed for this programming assignment.
- **Test results** – Representative running results of your implementation that sufficiently demonstrate the correctness of your programs.

Extra Credit: Please state clearly in your report if you have implemented any of the following extra credit options.

a. **(10 points) Support vector machines**
   By using an available quadratic programming optimizer or an SVM library, implement a training and classification algorithm for support vector machines. Then use your algorithm on the USPS dataset. Document the classification accuracy and compare the results with that from the two basic algorithms.

b. **(10 points) Kernel method for linear discriminant functions**
   Given a kernel function, derive the kernel-version of Algorithm 4 and implement the algorithm, and then apply it on the given wine and USPS datasets. Document the classification accuracy and compare the results with that from the two basic algorithms without kernels. Use the polynomial function of degree three as the kernel function; optionally, you can use other commonly used kernel functions.

c. **(10 points) Multiple-class linear machines and multiple-class boosting**
   Use the Kesler's construction to train a linear machine for multi-class classification and then use the SAMME algorithm to boost its performance on the training set. Apply the algorithm on both datasets and classify the corresponding test samples in the test sets. Document the classification accuracy and compare the results with that from the one-against-the-rest and one-against-the-other algorithms.

Grading

- **Report** – 15 points
- **Correct implementation** – 60 points
  - **Basic two-class algorithms –** 30 points
  - **Multi-class classification algorithms** – 15 points
  - **Adaboost algorithm** – 15 points
- **Results** - 15 points
- **Analysis** – 10 points
- **Support vector machines –** 10 points
- **Kernel methods –** 10 points
- **Multiple-class linear machines and multiple-class boosting** – 10 points

Additional Information

The textbook describes AdaBoost (Section 9.5.2). You can also find papers and references on the web on Adaboost. You can find source code for quadratic optimization at http://www.numerical.rl.ac.uk/qp/qp.html. You can also use a solver written in C (I used this one in the past) that is available here http://www.learning-kernel-classifiers.org/code/linear/pr_loqo.c and http://www.learning-kernel-classifiers.org/code/linear/pr_loqo.h.