CS 189: Introduction to Machine Learning
Homework 1: Support Vector Machines

Due: 11:59 pm on February 10th, 2016

Last updated: January 27, 2016, Version: 1.0

Introduction

Datasets

In the first part of this assignment (Problems 1 through 3), we will do digit recognition using our own variant of the MNIST dataset. The state-of-the-art error rate on this dataset is roughly 0.2%; the methods in this assignment will get you to around 8% (modulo implementation details). We will try to get closer to the state-of-the-art as we move further in the course.

Additionally, in the latter part of the assignment (Problem 4), you will be classifying real spam messages. Your job is to build a classifier that, given the raw text of an email, can classify it as spam or ham (not-spam). We have provided you with the raw text of real emails and extracted some features. You have the freedom and are encouraged to generate new features (and maybe discard or modify some of ours) to improve your accuracy. The features provided should give you around 75% accuracy, although with smart feature engineering, you can achieve better performance.

Guidelines

- Both datasets for this homework are included in the accompanying zip file on bCourses.
- Check Piazza frequently for updates; it will be our official means of communication with anything related to this assignment.
- You may discuss the homework with other students, but everyone must write and submit their own code and write-up.
- You may submit a Jupyter Notebook (e.g. IPython, IJulia, ...); however, please include both the rendered PDF and notebook.

Algorithm

You will be classifying digits and spam messages using the Support Vector Machine (SVM) algorithm. You will be using an external library that provides an implementation of this algorithm. You do not need to know anything about this algorithm yet, except that it is a technique used for classification. Later in the semester, we will revisit SVMs and discuss the theory in more detail.

Implementation

You may choose any programming language and library as long as you only use the linear SVM classifier. Appendix B: Setting Up contains instructions for getting started with Python and MATLAB. We cannot provide support unless you are following those guidelines.

Data Partitioning

To avoid computational expense, we will only train on a subset of the digit data. (Once you have finished this assignment, if you have enough computing power at your disposal, you can try your code on the full dataset.) In the real world, you will most likely receive a huge chunk of (labeled or unlabeled) data. Rarely will you receive "training" data and "testing" data; you will have to partition a validation set yourself. In parts of this homework, you will have to partition this data.

Support Vector Machines

We will use linear support vector machines to classify handwritten digits from 0–9. In this assignment, we will use the simplest of features for classification: raw pixel brightness values.[1] There are several ways to evaluate your classifier. Here, we focus on *classification accuracy* as a measure of the error rate.

[1] In other words, our feature vector for an image will be a row vector with all the pixel values concatenated in a row major (or column major) format.

PROBLEM 1. Train a linear SVM using raw pixels as features. Plot the error rate on a validation set versus the number of training examples that you used to train your classifier. Make sure you set aside 10,000 training images as a validation set. The number of training examples in your experiment should be 100, 200, 500, 1,000, 2,000, 5,000, and 10,000. At this stage, you should expect accuracies between 70% and 90%.

Confusion Matrix

PROBLEM 2. Create confusion matrices[2] for each experiment in Problem 1. Color code and report your results. You may use built-in implementations to generate confusion matrices. What insights can you get about the performance of your algorithm from looking at the confusion matrix?

[2] In a multi-class classification setting, a **confusion matrix** is often used to report the performance of your algorithm. Briefly, a confusion matrix is a matrix where rows correspond to the actual class and columns indicate the predicted class. A more detailed explanation can be found on Wikipedia.

Cross-validation

A common practice while performing machine learning experiments is to perform cross-validation to select model parameters. In this assignment, we will use k-fold cross-validation[3] to determine a good value for the regularization parameter C in the soft-margin SVM algorithm.

While trying to choose a model parameter, cross-validation is repeated for several different parameter values. The parameter value with the highest cross-validation accuracy is used when training the final model on the entire training set.[4]

PROBLEM 3. Explain why cross-validation helps. Implement cross-validation[5] and find the optimal value of the parameter C using 10-fold cross-validation on the training set with 10,000 examples. Train a linear SVM with this value of C. Please report your C value, the validation error rate, and your Kaggle score. If you used additional features, please (briefly) describe what features you added, removed, or modified.

PROBLEM 4. Use your cross-validation implementation from above to train a linear SVM for your spam dataset. Please report your C value, the validation error rate, and your Kaggle score. If you modified the spam features, please (briefly) describe what features you added, removed, or modified.

PROBLEM 5. *Bells and Whistles! (Optional)*

Try running your own data through the classifiers you built! Use this tool: `http://mnist-tester.herokuapp.com/` to make your own MNIST-style digits and see if your classifier can correctly recognize the digit.

For the spam dataset, you can try typing your own spammy/hammy emails and test if your classifier correctly identifies them. You can try building your own custom features to improve upon our default implementation's accuracy!

[3] Briefly, cross-validation is a technique to assess how well your model generalizes to unseen data. In k-fold cross-validation, the training data is *randomly* partitioned into k disjoint sets and the model is trained on $k-1$ sets and validated on the k^{th} set. This process is repeated k times with each set chosen as the validation set once. The **cross-validation accuracy** is reported as the average accuracy of the k iterations.

[4] There are other forms of cross-validation besides k-fold cross-validation. See Wikipedia for more information.

[5] You may not use any built-in cross-validation functionality; you must implement this yourself. Note that the cross-validation error rate is different from the validation set error rate described in Problem 1.

Appendix A: Deliverables

bCourses Submission

Submit a zip file to bCourses that contains the following. Do **NOT** include the data we provided (digits and spam) in your submission.

- **A short README.** It should contain (1) your name and student ID, and (2) instructions on how to use your code to reproduce your results.
- **Your code.** We must be able to run it out of the box. Please take care that your code doesn't take up inordinate amounts of time or memory.

  If your code requires additional dependencies, please include complete instructions in the README for how to download and build the dependencies, as well as the steps to run your code. If your code cannot be executed, your solution cannot be verified. The graders will not try to debug your submission; it is your responsibility to provide clear instructions.
- **A short write-up.** The write-up should be a PDF with your answers to the problems above, including your images and plots. It must include your Kaggle scores for both digits and spam. Include an appendix with all your code at the end of your writeup.

  All plots should be properly labelled. No handwritten homeworks will be accepted.
- **Your best Kaggle predictions.** Submit the two text files (one for digits, one for spam) with your best predictions for the examples in the Kaggle test set.

  The Kaggle invite links and more instructional details will be posted on Piazza. There will be separate Kaggle competitions for each dataset. Although added features are optional, there will be fame, glory, and better letters of recommendation for the winners of the Kaggle competitions.

Gradescope Submission

You should be added to Gradescope soon after this assignment is released. You should upload your write-up to the "Homework 1: Support Vector Machines" assignment on Gradescope.

Appendix B: Setting Up

MATLAB AND LIBLINEAR

Before you begin, you need to make sure everything compiles. You will need MATLAB, and either a Mac or Linux. (If you use Windows, you will need a version of make.)

Most files are just MATLAB, so no compilation is required. However, you need to compile the LIBLINEAR package from `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`. Place the directory you download from LIBLINEAR into your `code` directory, `cd` into it (i.e. `cd code/liblinear-x`), and run `make`.

Then, edit `code/liblinear-x/matlab/Makefile` to point to your MATLAB installation and run `make` from the `matlab` subdirectory. For further information, please read the README file in the package.

PYTHON AND SCIKIT-LEARN

You can save yourself a lot time by installing Python distributions such as Anaconda or Canopy. These are batteries-included Python installations, which should work out of the box.

If you have an existing Python installation (including pip, the Python package manager), you can follow the instructions at `http://scikit-learn.org/stable/install.html` to install `scikit-learn` and its dependencies (namely, `NumPy` and `SciPy`). If an install command doesn't work, you might just need to run `sudo pip install ...` instead of `pip install ...` as suggested in the docs.

The test and training data are stored in `.mat` files, which can be loaded with `scipy.io.loadmat`.

Appendix C: Digits Dataset

This will be used for Problems 1, 2, and 3. We have provided the following files:

- **train.mat** - This file contains 60,000 digit images for training and their respective labels.
- **test.mat** - This file contains 10,000 digit images *without* labels. This will be used for Kaggle.

In addition to the dataset, we have provided two MATLAB functions in the `code` directory. `benchmark.m` will take in the predicted labels and true labels and return the error rate, as well as the indices of the incorrect labels. `montage_images.m` will produce a montage of a set of images. You can use this to visualize digits that you classify

incorrectly. Both files are easily translatable into Python/NumPy if you wish to use that instead.

Appendix D: Spam Dataset

This will be used for Problem 4. We have provided the following files and directories:

- **spam_data.mat** - Data with features extracted. There are three matrices of interest here, labeled `training_data`, `training_labels`, and `test_data`.
- **featurize.py** - Python file that extracts features from the emails. Please read the top of the file for instructions to add more features. If you add more features, be sure re-run this file to update your `.mat` file.
- **ham/** - This directory contains the raw non-spam emails.
- **spam/** - This directory contains the raw spam emails.
- **test/** - This directory contains the raw unlabeled emails you will be using for Kaggle.

You should not be modifying anything in **ham/**, **spam/**, **test/**. The labels are 1 for spam and 0 for ham.

Appendix E: Hints

Things to try if you're stuck:

- Use the digit data in a consistent format within your assignment - either integer values between 0 and 255 or float values between 0 and 1. Training on floats and then testing with integers is bound to cause trouble.
- Cross-validation requires choosing from **random** partitions. This is best implemented by shuffling your training examples and your labels and then partitioning them by their indices.