CS 189 Introduction to Machine Learning
Spring 2017 Jonathan Shewchuk

HW 01

1 Getting Started

Start this homework early! You can only submit to Kaggle twice a day.

Also: a reminder that while discussion of homework problems is encouraged, all homeworks, including programming, must be written individually. We will actively check for plagiarism. The typical penalty is a large NEGATIVE score, but we reserve the right to give an instant F for even one violation, and will always give an F for two.

1.1 Datasets

In this assignment, we will be running out-of-the-box implementations to classify three datasets.

1. 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.

Figure 1: Examples from the MNIST dataset.

2. Additionally, 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.

3. Finally, you will be categorizing objects in images, using the CIFAR-10 dataset. You should see around 30% accuracy.

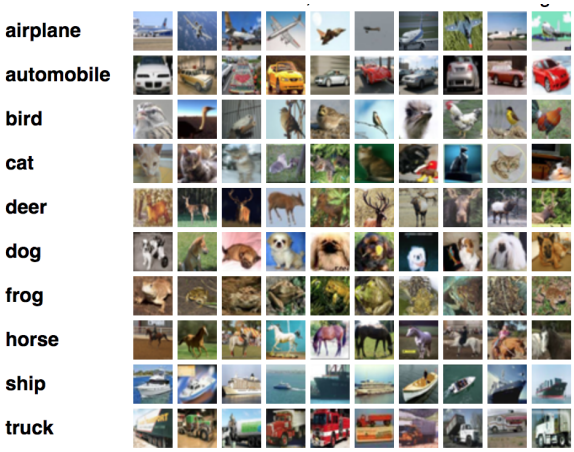

Figure 2: Examples from the CIFAR-10 dataset.

1.2 Algorithm

You will be classifying digits, spam messages, and objects using the Support Vector Machine (SVM) algorithm. You will be using the scikit-learn library. 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.

1.3 Implementation

No skeleton code is provided.

You can save yourself a lot time by installing Python distributions such as Anaconda. 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.

To get started, take a look at `scipy.io.loadmat` and `sklearn.svm`. You'll find everything else you need in the scikit-learn, NumPy, and SciPy documentation.

2 Data Partitioning

PROBLEM 1. Rarely will you receive "training" data and "validation" data; you will have to partition a validation set yourself. For the MNIST dataset, set aside 10,000 training images as a validation set. For the spam dataset, set aside 20% training samples as a validation set. For the CIFAR-10 dataset, set aside 5,000 training images as a validation set. Be sure to shuffle your data before splitting it to make sure all classes are represented in your partitions.

3 Support Vector Machines

We will use linear support vector machines to classify our datasets. For images, we will use the simplest of features for classification: raw pixel brightness values. 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.

There are several ways to evaluate models. We will use *classification accuracy* as a measure of the error rate.

PROBLEM 2. Train a linear SVM on all three datasets. Plot the error rate on the training and validation sets versus the number of training examples that you used to train your classifier.

1. For the **MNIST** dataset, use raw pixels as features. At this stage, you should expect accuracies between 70% and 90%. The number of training examples in your experiment should be 100, 200, 500, 1,000, 2,000, 5,000, and 10,000.

2. For the **spam** dataset, use the provided word frequencies as features. In other words, each document is represented by a vector, where the *i*th entry denotes the number of times word *i* (as specified in `featurize.py`) is found in that document. At this stage, you should expect accuracies between 70% and 90%. The number of training examples in your experiment should be 100, 200, 500, 1,000, 2,000, and finally all the data in your training split.

3. For the **CIFAR-10** dataset, use raw pixels as features. At this stage, you should expect accuracies between 25% and 35%. A warning that training SVMs for CIFAR-10 takes a couple minutes to run as the number of training examples increases. As such, you only need to train SVMs for 100, 200, 500, 1,000, 2,000, and 5,000 examples.
   We found that `SVC(kernel='linear')` was faster than `LinearSVC`.

4 Hyperparameter Tuning

In the previous problem, you found the parameters for a model that classifies the given data. Many classifiers also have *hyperparameters* that you can tune that influence the parameters. In this problem, we'll determine good value for the regularization parameter C in the soft-margin SVM algorithm.

When trying to choose a hyperparameter value, the model is trained repeatedly with different hyperparameters and the hyperparameter corresponding to the model with the highest accuracy on the validation dataset is selected. Before generating predictions for the test set, the model can be retrained on all labelled data using the previously-determined hyperparameter.

PROBLEM 3. For the MNIST dataset, find the best C value. In your report, list the best C value, the C values you tried, and the corresponding accuracies. As in the previous problem, you need only train on up to 10,000 examples for performance reasons.

5 K-Fold Cross-Validation

For smaller datasets (e.g. the spam dataset), the validation set will contain fewer examples, and our estimation of our error will have high variance. A way to combat this is to use *k-fold cross-validation*.

In k-fold cross-validation, the shuffled training data is 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.

PROBLEM 4. For the spam dataset, use k-fold cross validation to find and report the best C value. Use $k = 5$. In your report, list the best C value, the C values you tried, and the corresponding accuracies.

6 Kaggle

PROBLEM 5. Using the best model you trained for each dataset, generate predictions for the test sets we provide and save those predictions to .csv files. Upload your predictions to the Kaggle leaderboards (details on Piazza). In your report, include your Kaggle name as it displays on the leaderboard and your Kaggle score for each of the three datasets.

For your Kaggle submission, you may optionally add more features or use a non-linear SVM kernel to get a higher position on the leaderboard. If you do, please explain what you did your report and cite your external sources. Examples of things you might investigate: SIFT and HOG features for images, and bag of words for spam/ham. Almost everything is fair game as long as your underlying model is an SVM (i.e. do not use a neural network, decision tree, etc.). You are also not allowed to search for the labelled test data and submit that to Kaggle. If you have any questions about whether something is allowed or not, ask on Piazza.

Start early! Kaggle only permits two submissions per leaderboard per day.

Appendix A: Deliverables

GRADESCOPE ZIP SUBMISSION

You should upload your *zipped* files to the "Homework 1 Code: Support Vector Machines" assignment on Gradescope. Do **NOT** include the data we provided (digits, spam, and objects) 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. You may submit a Jupyter Notebook (e.g. IPython, IJulia, ...)

  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.

GRADESCOPE PDF SUBMISSION

You should upload your write-up to the "Homework 1: Support Vector Machines" assignment on Gradescope.

- **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. Finally, include a list of people you discussed the homework with at the top of your write-up.

  If you submitted a Jupyter Notebook, please include the rendered PDF here.

  All plots should be properly labelled. No handwritten homeworks will be accepted.

KAGGLE

As mentioned in the previous section, you'll need to submit your predictions for the test sets to Kaggle and include your Kaggle scores in your writeup.

Appendix B: Digits Dataset

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.

The images are flattened 28×28 (grayscale).

Appendix C: Spam Dataset

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 D: CIFAR-10 Dataset

We have provided the following files and directories.

- **train.mat** - This file contains 50,000 object images for training and their respective labels.
- **test.mat** - This file contains 10,000 object images *without* labels. This will be used for Kaggle.

The images are flattened $3 \times 32 \times 32$ (3 color channels). The labels 0-9 correspond alphabetically to to the categories. For example, 0 is airplane, 1 is automobile, 2 is bird, and so on.

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.