CS 189 Introduction to Machine Learning
Spring 2019 Jonathan Shewchuk

HW1

Due: Wednesday, January 30th at 11:59 pm

This homework is comprised of two parts. The first part consists of a set of coding exercises. The second part consists of math problems.

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

Deliverables:

1. Submit your predictions for the test sets to Kaggle as early as possible. Include your Kaggle scores in your write-up (see below).
2. Submit a PDF of your homework, **with an appendix listing all your code**, to the Gradescope assignment entitled "HW1 Write-Up". You may typeset your homework in LaTeX or Word (submit PDF format, **not** .doc/.docx format) or submit neatly handwritten and scanned solutions. **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.
   - In your write-up, please state with whom you worked on the homework.
   - In your write-up, please copy the following statement and sign your signature next to it. (Mac Preview and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make it *extra* clear so that no one inadverdently cheats.

     "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."
3. Submit all the code needed to reproduce your results to the Gradescope assignment entitled "HW1 Code". Yes, you must submit your code twice: once in your PDF write-up (above) so the readers can easily read it, and once in compilable/interpretable form so the readers can easily run it. Do **NOT** include any data files we provided. Please include a short file named README listing your name, student ID, and instructions on how to reproduce your results. Please take care that your code doesn't take up inordinate amounts of time or memory. If your code cannot be executed, your solution cannot be verified.

1 Python Configuration and Data Loading

Please follow the instructions below to ensure your Python environment is configured properly, and you are able to successfully load the data provided with this homework. No solution needs to be submitted for this question. For all coding questions, we recommend using Anaconda for Python 3.

(a) Either install Anaconda for Python 3, or ensure you're using Python 3. To ensure you're running Python 3, open a terminal in your operating system and execute the following command:

```
python --version
```

Do not proceed until you're running Python 3.

(b) Install the following dependencies required for this homework by executing the following command in your operating system's terminal:

```
pip install scikit-learn scipy numpy matplotlib
```

Please use Python 3 with the modules specified above to complete this homework.

(c) You will be running out-of-the-box implementations of support vector machines to classify three datasets. You will find a set of `.mat` files in the `data` folder for this homework. Each `.mat` file will load as a Python dictionary. Each dictionary contains three fields:

- **training_data**, the training set features. Rows are samples and columns are features.
- **training_labels**, the training set labels. Rows are samples. There is one column: The label for each sample.
- **test_data**, the test set features. Rows are samples and columns are features. You will fit a model to predict the labels for this test set, and submit those predictions to Kaggle.

The three datasets for the coding portion of this assignment are described below.

- **mnist_data.mat** contains data from the MNIST dataset. There are 60,000 labeled digit images for training, and 10,000 digit images for testing. The images are grayscale, 28×28 pixels flattened. There are 10 possible labels for each image, namely, the digits 0–9.

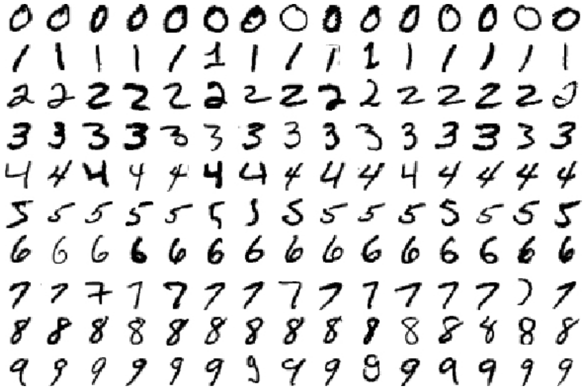

Figure 1: Examples from the MNIST dataset.

- **spam_data.mat** contains featurized spam data. The labels are 1 for spam and 0 for ham. The `data` folder includes the script **featurize.py** and the folders **spam**, **ham** (not spam), and **test** (unlabeled test data); you may modify **featurize.py** to generate new features for the spam data.
- **cifar10_data.mat** contains data from the CIFAR10 dataset. There are 50,000 labeled object images for training, and 10,000 object images for testing. The images are flattened $3 \times 32 \times 32$ (3 color channels). The labels 0–9 correspond alphabetically to the categories. For example, 0 means airplane, 1 means automobile, 2 means bird, and so on.

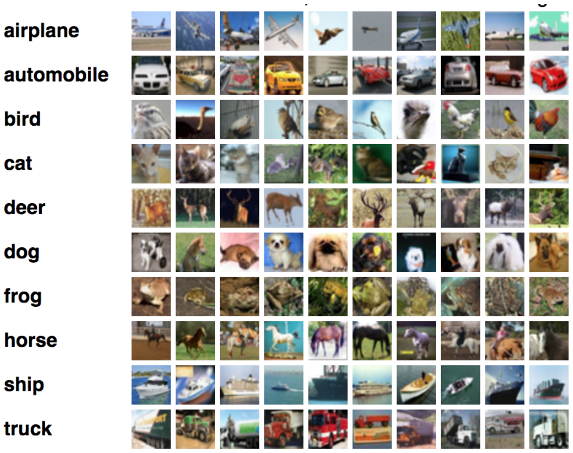

Figure 2: Examples from the CIFAR-10 dataset.

To check whether your Python environment is configured properly for this homework, ensure the following Python script executes without error. Pay attention to errors raised when attempting to import any dependencies. Resolve such errors by manually installing the required dependency (e.g. execute `pip install numpy` for import errors relating to the numpy package).

```
import sys
if sys.version_info[0] < 3:
    raise Exception("Python 3 not detected.")

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io

for data_name in ["mnist", "spam", "cifar10"]:
    data = io.loadmat("data/%s_data.mat" % data_name)
    print("\nloaded %s data!" % data_name)
    fields = "test_data", "training_data", "training_labels"
    for field in fields:
        print(field, data[field].shape)
```

2 Data Partitioning

Rarely will you receive “training” data and “validation” data; usually you will have to partition available labeled data yourself. The datasets for this assignment are described below. Write code to partition the datasets as follows.

(a) For the MNIST dataset, write code that sets aside 10,000 training images as a validation set.

(b) For the spam dataset, write code that sets aside 20% of the training data as a validation set.

(c) For the CIFAR-10 dataset, write code that sets aside 5,000 training images as a validation set. Be sure to shuffle your data before splitting it to make sure all the classes are represented in your partitions.

3 Support Vector Machines: Coding

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) order.

There are several ways to evaluate models. We will use *classification accuracy* as a measure of the error rate (see here: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html`).

Train a linear support vector machine (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. The number of training examples in your experiment will vary per dataset.

You may only use sklearn for the SVM model and the accuracy metric function. Everything else (train vs. val plots) must be done without the use of sklearn.

(a) For the **MNIST** dataset, use raw pixels as features. Train your model with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, 5,000, 10,000. At this stage, you should expect accuracies between 70% and 90%.

Hint: Be consistent with any preprocessing you do. Use either integer values between 0 and 255 or floating-point values between 0 and 1. Training on floats and then testing with integers is bound to cause trouble.

(b) 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. Train your model with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, **ALL**.

Note that this dataset does not have 10,000 examples; use all of your examples instead of 10,000. At this stage, you should expect accuracies between 70% and 90%.

(c) For the **CIFAR-10** dataset, use raw pixels as features. At this stage, you should expect accuracies between 25% and 35%. Be forwarned that training SVMs for CIFAR-10 takes a couple minutes to run for a large training set. Train your model with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, 5,000.
Note: We find that `SVC(kernel='linear')` is faster than `LinearSVC`.

4 Hyperparameter Tuning

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

When we are trying to choose a hyperparameter value, we train the model repeatedly with different hyperparameters. We select the hyperparameter that gives the model with the highest accuracy on the validation dataset. Before generating predictions for the test set, the model should be retrained using all the labeled data (including the validation data) and the previously-determined hyperparameter.

The use of automatic hyperparameter optimization libraries is prohibited for this part of the homework.

(a) For the MNIST dataset, find the best C value. In your report, list the C values you tried, the corresponding accuracies, and the best C value. As in the previous problem, for performance reasons, you are required to train with up to 10,000 training examples but not required to train with more than that.

5 K-Fold Cross-Validation

For smaller datasets (e.g., the spam dataset), the validation set contains fewer examples, and our estimate of our error might not be accurate—the estimate has high variance. A way to combat this is to use *k-fold cross-validation*.

In k-fold cross-validation, the training data is shuffled and partitioned into k disjoint sets. Then 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 we report is the accuracy averaged over the k iterations.

Use of automatic cross-validation libraries is prohibited for this part of the homework.

(a) For the spam dataset, use 5-fold cross-validation to find and report the best C value. In your report, list the C values you tried, the corresponding accuracies, and the best C value.

Hint: Effective cross-validation requires choosing from **random** partitions. This is best implemented by randomly shuffling your training examples and labels, then partitioning them by their indices.

6 Kaggle

- MNIST Competition: https://www.kaggle.com/c/cs189-hw1-mnist
- SPAM Competition: https://www.kaggle.com/c/cs189-hw1-spam
- CIFAR-10 Competition: https://www.kaggle.com/c/cs189-hw1-cifar10

Using the best model you trained for each dataset, generate predictions for the test sets we provide and save those predictions to .csv files. Be sure to use integer labels (not floating-point!) and no spaces (not even after the commas). Upload your predictions to the Kaggle leaderboards (submission instructions are provided within each Kaggle competition). 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 submissions, 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 in your report and cite your external sources. Examples of things you might investigate include 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 labeled test data and submit that to Kaggle. If you have any questions about whether something is allowed or not, ask on Piazza.

Remember to start early! Kaggle only permits two submissions per leaderboard per day. To help you format the submission, please use check.py to run a basic sanity check on your submission and save_csv.py to help save your results.

To check your submission csv,

```
python check.py <competition name, eg. mnist> <submission csv file>
```

7 Theory of Hard-Margin Support Vector Machines

A *decision rule* (or *classifier*) is a function $r : \mathbb{R}^d \to \pm 1$ that maps a feature vector (test point) to $+1$ ("in class") or -1 ("not in class"). The decision rule for linear SVMs is

$$r(x) = \begin{cases} +1 & \text{if } w \cdot x + \alpha \geq 0, \\ -1 & \text{otherwise,} \end{cases} \tag{1}$$

where $w \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ are the weights (parameters) of the SVM. The hard-margin SVM optimization problem (which chooses the weights) is

$$\min_{w,\alpha} \; |w|^2 \; \text{ subject to } \; y_i(X_i \cdot w + \alpha) \geq 1, \;\; \forall i \in \{1, \ldots, m\}, \tag{2}$$

where $|w| = \|w\|_2 = \sqrt{w \cdot w}$.

We can rewrite this optimization problem by using Lagrange multipliers to eliminate the constraints. (If you're curious to know what Lagrange multipliers are, the Wikipedia page is recommended, but you don't need to understand them to do this problem.) We thereby obtain the equivalent optimization problem

$$\max_{\lambda_i \geq 0} \min_{w,\alpha} \; |w|^2 - \sum_{i=1}^{m} \lambda_i (y_i (X_i \cdot w + \alpha) - 1). \tag{3}$$

(a) Show that Equation (3) can be rewritten as the *dual optimization problem*

$$\max_{\lambda_i \geq 0} \; \sum_{i=1}^{m} \lambda_i - \frac{1}{4} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j X_i \cdot X_j \; \text{ subject to } \; \sum_{i=1}^{m} \lambda_i y_i = 0. \tag{4}$$

Hint: Use calculus to determine what values of w and α optimize Equation (3). Explain where the new constraint comes from.

We note that SVM software usually solves this dual quadratic program, not the primal quadratic program.

(b) Suppose we know the values λ_i^* and α^* that optimize Equation (3). Show that the decision rule specified by Equation (1) can be written

$$r(x) = \begin{cases} +1 & \text{if } \alpha^* + \frac{1}{2}\sum_{i=1}^{m} \lambda_i^* y_i X_i \cdot x \geq 0, \\ -1 & \text{otherwise.} \end{cases} \tag{5}$$

(c) The training points X_i for which $\lambda_i^* > 0$ are called the support vectors. In practice, we frequently encounter training data sets for which the support vectors are a small minority of the training points, especially when the number of training points is much larger than the number of features (i.e., the dimension of the feature space). Explain why the support vectors are the only training points needed to evaluate the decision rule. Then explain why the non-support vectors nonetheless still have some influence on the decision rule . . . what is the nature of that influence?

(a) Show that Equation (3) can be rewritten as the *dual optimization problem*

$$\max_{\lambda_i \geq 0} \sum_{i=1}^{m} \lambda_i - \frac{1}{4} \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y_i y_j X_i \cdot X_j \text{ subject to } \sum_{i=1}^{m} \lambda_i y_i = 0. \tag{4}$$

$$\max_{\lambda_i \geq 0} \min_{w,\alpha} |w|^2 - \sum_{i=1}^{m} \lambda_i (y_i (X_i \cdot w + \alpha) - 1).$$

$\nabla_w = 2w - \sum_i \lambda_i y_i X_i = 0$

$w = \frac{1}{2} \sum_i \lambda_i y_i X_i$

$\nabla_a = -\sum_i \lambda_i y_i = 0$

$\sum_i \lambda_i y_i = 0$

$\rightarrow -w \cdot w - \sum_i -\lambda_i$

$= \sum_i \lambda_i - \frac{1}{4} \sum_i \sum_j \lambda_i \lambda_j y_i y_j X_i X_j$

$s.t: \sum_i \lambda_i y_i = 0$

constraint $\sum_i \lambda y_i = 0$ is needed for convex optimization.
If there is no such constraint, we can put any values for λ_i to maximize the whole function.
So for eq.(4) w/o constraint, we can put a value that makes $\sum_i \lambda_i y_i$ ∞ that the f(x) will be ∞.

(b) Suppose we know the values λ_i^* and α^* that optimize Equation (3). Show that the decision rule specified by Equation (1) can be written

$$r(x) = \begin{cases} +1 & \text{if } \alpha^* + \frac{1}{2} \sum_{i=1}^{m} \lambda_i^* y_i X_i \cdot x \geq 0, \\ -1 & \text{otherwise.} \end{cases} \tag{5}$$

$r(x) \begin{cases} +1 \\ -1 \end{cases}$ $wx + a = 0$. $w = \frac{1}{2} \Sigma \lambda y X$

$\hookrightarrow r(x) \begin{cases} +1 & (\frac{1}{2} \Sigma \lambda y X) x + a^* \\ -1 \end{cases}$

$= r(x) \begin{cases} +1 & \frac{1}{2} \Sigma \lambda^* y X \cdot x + a^* \\ -1 \end{cases}$

(c) The training points X_i for which $\lambda_i^* > 0$ are called the support vectors. In practice, we frequently encounter training data sets for which the support vectors are a small minority of the training points, especially when the number of training points is much larger than the number of features (i.e., the dimension of the feature space). Explain why the support vectors are the only training points needed to evaluate the decision rule. Then explain why the non-support vectors nonetheless still have some influence on the decision rule . . . what is the nature of that influence?

The boundary is in $wx + b = 0$ and two classes are in either $wx + b = \pm 1$.
The goal of SVM is to find the boundary in the middle of two classes while maximizing margins, which can be solved via quadratic programming.
If point that are $wx + b > 1$, then it belongs to class 1 and same for class -1.
Support vectors are those points reside on lines ($wx + b = \pm 1$) and there are no other points between those and the boundary. Since it is to find the midpoint of $wx + b = \pm 1$, points that are $|wx + b| > 1$ dont affect determining the boundary.
However, before training a model, we dont know which points are support vectors that disregarding non-SV w/o knowing may influence the model.
Once we know support vectors, non-support vectors are not considered for boundary.