# CS 189: Introduction to Machine Learning
# Spring 2015
# Homework 1: Support Vector Machines

Due: 11:59 pm on February 3rd, 2015

## Introduction

In this assignment, 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 0.5%; 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 latter parts of the assignment, you will be classifying real spam messages. We have provided you with raw text of real emails and extracted some features. Your job is to build a classifier that, given the raw text of an email, can classify it as spam or ham (not-spam). You have the freedom and are encouraged to generate new features (and maybe discard or modify some of ours) to improve your accuracy. Your baseline accuracy (with the features provided) should give you around 75% accuracy, although with smart feature engineering, you can achieve better performance. Please refer to Appendix D for details on the spam dataset and instructions on how to use it.

You will be classifying digits and spam messages using an algorithm called Support Vector Machine (SVM). Currently, you probably know very little about this algorithm except that it is a technique used for classification, which is fine since you will be using an external software library that already has an implementation of this algorithm. Later in the semester, we will revisit the topic of SVMs and dive into the theory in more detail. Before starting, please read Appendix B to setup your choice of software package.

You may work with other students, but everyone must write and submit their own code and writeup.

## Data Partitioning

To avoid computational expense, we will only train on a subset of the digits 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: pixel brightness values. Simply put, our feature vector for an image would 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. The choices of the number of training examples should be 100, 200, 500, 1,000, 2,000, 5,000 and 10,000. Make sure you set aside 10,000 other training points as a validation set. You should expect accuracies between 70% and 90% at this stage.

## Confusion Matrix

In a multiclass classification setting, a confusion matrix is normally used to report the performance of your algorithm. Briefly, a confusion matrix is a matrix whose rows correspond to the actual class and columns indicate the predicted class. A more detailed explanation can be found at **http://en.wikipedia.org/wiki/Confusion_matrix**

**Problem 2**
Create confusion matrices for your experiments in Problem 1. Color code your results and report them. You can 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?

## 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 to determine a good value for the SVM regularization parameter **'C'**.

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.

While trying to fix a model parameter, the above process is repeated for a number of different values of the parameter, and the parameter value with the highest cross-validation accuracy is selected. The final trained model is obtained by training on the entire training set with the obtained parameter value. There are other forms of cross-validation too (**http://en.wikipedia.org/wiki/Cross-validation_(statistics)**).

**Problem 3**
Explain why cross-validation helps. Find the optimal value of the parameter **'C'** using 10-fold cross-validation on the training set with 10,000 examples. Train an SVM with this value of **'C'** and report the validation error rate.

You should not any built-in cross-validation functionality; you should implement this yourself. Note that the cross-validation error rate is different from a validation set error rate as in Problem 1.

**Problem 4**
Train a linear SVM for your spam dataset. Please state your **'C'** value, your accuracy using cross validation, and (briefly) what features you added, removed, or modified, if any. Adding features is optional.

**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 recognise the digit.

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

# Appendix

## A - Deliverables

You should submit a zip file to bcourses that contains the following three things:

- Your code. We should be able to run it out of the box.
- A short write-up containing the answers to the above. The write-up should be a PDF with your images and plots inside.
- Your write-up should include your Kaggle score for both digits and spam.
- A short README that contains (1) your name and student ID, and (2) instructions on how to use your code to reproduce your results.

You are also required to submit a text file with your best predictions for the examples in the test set (for both MNIST and spam) to Kaggle. There will be separate Kaggle competitions for each dataset. The Kaggle invite links and more instructional details will be posted on the course website.

## B - Setting up your SVM

### MATLAB and LIBLINEAR

Before you begin, you need to make sure everything compiles well. You will need MATLAB, and either a Mac or Linux. If you are in 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 that you will download from `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`. Place the directory you download from LIBLINEAR into the code subdirectory and cd into it, e.g cd tocode/liblinear-x, then type make. Then cd into code/liblinear-x/matlab. Edit the Makefile to point to your local copy of MATLAB, and then type make. For further information, please read the README file in the package.

### Python and scikit-learn

You will need a normal Python install, which should include the Python package manager, pip.

Follow the instructions here: `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 blah" instead of "pip install

blah" as suggested in the docs. The test and training data is stored in .mat files, which can be loaded with the scipy.io.loadmat function (`http://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html`).

## C - Digits Dataset

We have provided the following files:

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

## D - Spam Dataset

We have provided the following files and directories:

- **ham/** - directory contains the raw non-spam emails
- **spam/** - directory contains the raw spam emails
- **test/** - directory contains the emails you were be testing on and submitting to Kaggle.
- **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.
- **spam_data.mat** - Data with features extracted. There are three matrices of interest here, labeled 'training_data', 'training_labels', 'test_data'.

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

## E - Hints

Things to try if you're stuck:

- Use the digit data in a consistent format within your assignment - either in 0-255 integer values or in 0-1 double values. Training on doubles and then testing with integers is bound to cause trouble.

- Be careful of integer concatenation when decimal places are important. This is especially relevant when trying non-integer values of C in problem 3.
- 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.

## F - Final Notes

The MNIST dataset is provided in the code folder. We provide two functions for you. `benchmark.m` will take in the predicted labels and true labels and return the error rate, and also the indices of the labels that are wrong. `montage_images.m` will produce a montage of a set of images: use this, for instance, to visualize the images on which you make errors. Both files are easily translatable into Python/numpy if you wish to use that instead.

Also, just to clarify, problems 1–3 are to be executed on the digits dataset only. Problem 4 is to be executed on the spam dataset only.