# Introduction to Machine Learning, Fall 2013

## Problem Set 3: Support Vector Machines

**Tuesday, October 8, 2013 at 11am (*before* class begins)**

**Important:** *See problem set policy on the course web site.*

---

**Instructions.** You may use the programming language of your choice. However, you are **not** permitted to use or reference any machine learning code or packages not written by yourself, except for question 2(d)-(g). In addition to your answers to the below questions, e-mail a PDF of your solutions and all code that you write for this assignment to Chen-Chien.

1. In this question, you will implement the Pegasos algorithm [5] to optimize the SVM objective using stochastic sub-gradient descent and revisit the spam data.

```
Initialize: Choose $\boldsymbol{w}_1 = 0, t = 0$.
1. For iter $= 1, 2, \cdots, 20$
2.   For $j = 1, 2, \cdots, |\text{data}|$
3.     $t = t + 1; \quad \eta_t = \frac{1}{t\lambda}$
5.     If $y_j(\boldsymbol{w}_t \cdot \boldsymbol{x}_j) < 1$
6.       $\boldsymbol{u} = (1 - \eta_t)\boldsymbol{w}_t - \eta_t y_j \boldsymbol{x}_j$
7.     Else
8.       $\boldsymbol{u} = (1 - \eta_t)\boldsymbol{w}_t$
9.     $\boldsymbol{w}_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\boldsymbol{u}\|}\right\} \boldsymbol{u}$
8. Output: $\boldsymbol{w}_{t+1}$
```

We use the identical setting as in the problem set 1. Split the data in `spam_train.txt` into a training and validate set, putting the last 1000 emails into the validation set. Transform all of the data into feature vectors. Build a vocabulary list using only the 4000 e-mail training set by finding all words that occur across the training set. Ignore all words that appear in fewer than $X = 30$ e-mails of the 4000 e-mail training set. For each email, transform it into a feature vector $\vec{x}$ where the $i$th entry, $x_i$, is 1 if the $i$th word in the vocabulary occurs in the email, and 0 otherwise.

*Note:* To keep your algorithm simple, we will not use an offset term $b$ when optimizing the SVM primal objective using Pegasos.

(a) Implement the function `pegasos_svm_train(data, lambda)`. The function should return $\boldsymbol{w}$, the final classification vector. For simplicity, the stopping criterion is set so that the total number of passes over the training data is 20. After each pass through the data, evaluate the SVM objective $f(\boldsymbol{w}_t) = \frac{\lambda}{2}\|\boldsymbol{w}_t\|^2 + \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i(\boldsymbol{w}_t \cdot \boldsymbol{x}_i)\}$ and store its value ($m = |\texttt{data}|$). Plot $f(\boldsymbol{w}_t)$ as a function of iteration (i.e. for $t = |\texttt{data}|, \ldots, 20|\texttt{data}|$), and submit the plot for $\lambda = 2^{-5}$.

(b) Implement the function `pegasos_svm_test(data, w)`.

(c) Run your learning algorithm for various values of the regularization constant, $\lambda = 2^{-9}, 2^{-8}, \cdots, 2^{1}$. Plot the average training error, average hinge loss of the training samples (i.e. $\frac{1}{m}\sum_{i=1}^{m}\max\{0, 1-y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i)\}$), and average validation error as a function of $\log_2 \lambda$. You should expect that the hinge loss upper-bounds the training error. What is the minimum of your validation error? For the classifier that has the smallest validate error: What is the test error? How many training samples are support vectors? How did you find them? Compare your test error with your result from problem set 1.

*Note:* Using a smaller value of $X$ such as 0 (i.e., not filtering the vocabulary) would give even better results (the SVM's regularization prevents overfitting).

2. The MNIST dataset is a database of handwritten digits. This problem will apply SVMs to automatically classify digits; the US postal service uses a similar optical character recognition (OCR) of zip codes to automatically route letters to their destination. The original dataset can be downloaded at `http://yann.lecun.com/exdb/mnist/`. For this problem, we randomly chose a subset of the original dataset. We have provided you with two data files, `mnist_train.txt`, `mnist_test.txt`. The training set contains 2000 digits, and the test set contains 1000 digits. Each line represents an image of size $28 \times 28$ by a vector of length 784, with each feature specifying a grayscale pixel value. The first column contains the labels of the digits, 0–9, the next 28 columns respesent the first row of the image, and so on. We also provide two scipts written in MATLAB/Octave and Python, `show_img.m`, `show_img.py` to show a single image; using these will help you have a better understanding of what the data looks like and how it is represented.

   Your linear classifier will obtain less than 15% test error, and using a Gaussian kernel you will obtain less than 7% test error! Had you used more training data, SVM with Gaussian kernel can get down to 1.4% test error (degree 4 polynomial obtains 1.1% test error). With further fine-tuning (e.g., augmenting the training set by adding deformed versions of the existing training images), a SVM-based approach can obtain 0.56% test error [2]. The state-of-the-art, which uses a convolutional neural network, obtains 0.23% test error [1].

   (a) Read in `mnist_train.txt`, `mnist_test.txt` and transform them into feature vectors. Normalize the feature vectors so that each feature is in the range [-1, 1]. Since in this dataset each feature has minimum value 0 and maximum value 255, you can do this normalization by transforming each column $\vec{v}$ to $2\vec{v}/255 - 1$. The normalization step can be crucial when you incorporate higher-order features. It also helps prevent numerical difficulties while solving the SVM optimization problem.

   (b) Implement multi-class prediction using one-versus-all classification. Train 10 binary classifiers using the Pegasos algorithm from the previous question. For each classifier, you relabel one of the labels to 1, and the other 9 labels to -1. Following learning, you will have 10 distinct weight vectors. To predict the label of an example $x$, compute the dot product of $x$ with each weight vector, giving you 10 scores, and predict the label with the maximum score.

   (c) Instead of holding out a specific portion of your training data as a validation set, there is another approach to estimate the test error called $k$-fold cross-validation. Cross-validation is particularly useful when you have a small amount of training data. Cross-validation divides the training data into $k$ parts of equal size. Then, for $i = 1, \ldots, k$, we fit a model using all of the data except for the $k$'th part, and use the remaining part to compute the validation error. Finally, we report the averaged validation error.

   Implement $k$-fold cross-validation with $k = 5$, and find a model having the smallest cross-validation error from $\lambda = 2^{-5}, 2^{-4}, \cdots, 2^{1}$. Plot the cross-validation error vs.

$\lambda$. What is the best $\lambda$? For this value of $\lambda$, re-train the classifier now using *all* of the training data. What is the test error?

(d) We now explore the use of non-linear kernels within Support Vector Machines. We will make use of a widely-used package `libSVM` and explore some of its functions. Please download the `libSVM` software from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/index.html`. You can use any interface for `libSVM` such as `scikit-learn` (Python), `Matlab`, `GNU Octave`, or stay in the command line.[1] The library implements the SMO algorithm described in class, which performs block coordinate descent in the dual SVM [3, 4].

(e) Try the default setting of the `libSVM`, which uses the Gaussian kernel with $\gamma = 1/\text{num\_features}$, and $C = 1$. Make sure that each feature is scaled to $[-1, 1]$ as in problem (b) which could also be done by using `svm-scale`. Note that in the library, the Gaussian kernel is of the form $K(\vec{u}, \vec{v}) = e^{-\gamma\|\vec{u}-\vec{v}\|^2}$ (equivalent to what we showed in class when $\gamma = 1/2\sigma^2$) and the optimization problem is of the form

$$\begin{aligned} \min_{\boldsymbol{w}, b, \xi} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{j=1}^{m} \xi_j \\ \text{subject to} \quad & y_j(\boldsymbol{w} \cdot \boldsymbol{x}_j + b) \geq 1 - \xi_j, \\ & \xi_j \geq 0. \end{aligned}$$

This can be seen to be equivalent to the SVM optimization problem solved by Pegasos when $C = 1/m\lambda$. Train on the full training set. What is the test error?

(f) Rather than using the default settings, we can choose the two parameters to be tuned ($C$ and $\gamma$) using cross-validation. If you prefer, you may use `libSVM`'s option `-v` for this purpose. Report the 10-fold cross-validation error when $\gamma$ and $C$ are at their default settings.

(g) Finally, try different $\gamma$ and $C$ values to find a model with small cross-validation error. What were the best values that you found? What is the cross-validation error? What is the test error for this setting?

## References

[1] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[2] Dennis Decoste and Bernhard Schölkopf. Training invariant support vector machines. *Machine Learning*, 46(1-3):161–190, 2002.

[3] Rong-En Fan, Pai-Hsuen Chen, and Chih-Jen Lin. Working set selection using second order information for training support vector machines. *The Journal of Machine Learning Research*, 6:1889–1918, 2005.

[4] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in kernel methods*, pages 185–208. MIT Press, 1999.

[5] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

[1] If you want to use `scikit-learn`, a convenient way is to download a Python distribution from Anaconda, or Enthought EPD. Both distributions provide free Intel Math Kernel Library (MKL) for academic use.