CSE 546 Machine Learning, Autumn 2015
Homework 3

Due: Tuesday, November 24, beginning of class

1 Fitting an SVM classifier by hand [50 Points]

(Source: Murphy text, Exercise 14.1) Consider a dataset with 2 points in 1d:

$$(x_1 = 0, y_1 = -1) \text{ and } (x_2 = \sqrt{2}, y_2 = 1).$$

Consider mapping each point to 3d using the feature vector $\phi(x) = [1, \sqrt{2}x, x^2]^T$ (This is equivalent to using a second order polynomial kernel).

When we examined the perceptron algorithm in class, we related the number of mistakes to the margin of the data. If our data are linearly separable, then a natural method is to try to find a classifier which maximizes the margin. Here, the max margin classifier has the form

$$\hat{w}, \hat{w_0} = \arg\min ||w||^2 \quad s.t. \tag{1}$$
$$y_1(w^T\phi(x_1) + w_0) \geq 1 \tag{2}$$
$$y_2(w^T\phi(x_2) + w_0) \geq 1 \tag{3}$$

1. *(10 Points)* Write down a vector that is parallel to the optimal vector $\hat{w}$. Hint: Recall from Figure 14.12 (page 500 in the Murphy text) that $\hat{w}$ is perpendicular to the decision boundary between the two points in the 3d feature space.

2. *(10 Points)* What is the value of the margin that is achieved by this $\hat{w}$? Hint: Recall that the margin is the distance from each support vector to the decision boundary. Hint 2: Think about the geometry of the points in feature space, and the vector between them.

3. *(10 Points)* Solve for $\hat{w}$, using the fact the margin is equal to $1/||\hat{w}||$.

4. *(10 Points)* Solve for $\hat{w_0}$ using your value for $\hat{w}$ and Equations 1 to 3. Hint: The points will be on the decision boundary, so the inequalities will be tight. A "tight inequality" is an inequality that is as strict as possible. For this problem, this means that plugging in these points will push the left-hand side of Equations 2 and 3 as close to 1 as possible.

5. *(10 Points)* Write down the form of the discriminant function $f(x) = \hat{w_0} + \hat{w}^T\phi(x)$ as an explicit function of x. Plot the 2 points in the dataset, along with $f(x)$ in a 2d plot. You may generate this plot by hand, or using a computational tool like Python.

Show your work.

2 SVMs: Hinge loss and mistake bounds [25 Points]

Suppose we classify with the hypothesis $h_w(x) = \text{sgn}(w^\top x)$ where $\text{sgn}(z) = 1$ if z is positive and -1 otherwise. Here we are dealing with binary prediction where each label is in $\{-1, 1\}$. Let us say we make a mistake on x, y with w if $y \neq \text{sgn}(w^\top x)$. The number of mistakes with w on a dataset is the classification loss.

The SVM loss function can be viewed as a relaxation to the classification loss. The *hinge* loss on a pair (x, y) is defined as:

$$\ell((x, y), w) = \max\{0, 1 - yw^\top x\}$$

where $x \in \mathrm{R}^d$ and $y \in \{-1, 1\}$. The SVM attempts to minimize:

$$\frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i w^\top x_i\} + \lambda\|w\|^2$$

where λ is a regularizer (there are different conventions as to how we write the regularizer, e.g. sometimes we write $\frac{\lambda}{n}$ instead of just λ).

The hinge loss provides a way of dealing with datasets that are not separable. Let us go through the argument as to why we view this as a relaxation of finding the max margin classifier.

1. *(5 Points)* Argue that the function $\ell((x, y), w) = \max\{0, 1 - yw^\top x\}$ is convex (as a function of w).

2. *(10 Points)* (Margin mistakes) Suppose that for some w we have a correct prediction of y_i with x_i, i.e. $y_i = \text{sgn}(w^\top x_i)$. What range of values can the hinge loss, $\ell((x_i, y_i), w)$, take on this correctly classified example? Points which are classified correctly and which have non-zero hinge loss are referred to as margin mistakes.

3. *(10 Points)* (Mistake bound) Let $M(w)$ be the number of mistakes made by w on our dataset (in terms of the classification loss). Show that:

$$\frac{1}{n}M(w) \leq \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i w^\top x_i\}$$

   In other words, the average hinge loss on our dataset is an upper bound on the average number of mistakes we make on our dataset.

3 Programming Question [50 Points]

In this problem, we seek to perform a digit recognition task, where we are given an image of a handwritten digit and wish to predict what number it represents. This is a special case of an important area of language processing known as Optical Character Recognition (OCR). We will be simplifying our goal to that of a binary classification between two relatively hard-to-distinguish numbers (specifically, predicting a 3 versus a 5). To do this, you will implement a linear support vector machine (SVM).

3.1 Dataset

The digit images have been taken from a Kaggle competition, `http://www.kaggle.com/c/digit-recognizer/data`, and the data is on here `https://courses.cs.washington.edu/courses/cse546/15au/hw/hw3-data.zip`. This data was originally from the MNIST database of handwritten digits, but was converted into a easier-to-use file format.

The data have also undergone some preprocessing. They have been filtered to just those datapoints whose labels are 3 or 5, which have then been relabeled to 1 and -1 respectively. Then, we subsampled to create two datasets `validation.csv` and `test.csv` ($N = 1000$ for both). Each row in these files represents an image. The first column is the label of the image, and the remaining columns give the grayscale values for each pixel. We will use `validation.csv` as training data, and `test.csv` as test data.

Preprocessing The data provided in the csv file is in the range $[0, 256)$. Normally when you work with a linear model, you want to normalize the data so that the range is closer to $[0, 1]$. There are many ways to perform data normalization, and in this homework we use the following L2 normalization

$$\mathbf{x} \leftarrow \mathbf{x} / \|\mathbf{x}\| \tag{4}$$

Here $\|\mathbf{x}\|$ is the L2 norm of $\mathbf{x}$. This normalization will make each input vector have unit length, and it usually works well in practice. You can use the following code to load and normalize the data:

```
def loaddata(fname):
    data = np.loadtxt(fname,skiprows=1, delimiter=',')
    y = data[:,0]
    X = data[:,1:]
    nm = np.sqrt(np.sum(X * X, axis=1))
    X = X / nm[:,None]
    return y, X
trainY, trainX = loaddata('validation.csv')
testY, testX = loaddata('test.csv')
```

3.2 Implement Linear SVM

The objective of a linear SVM can be written as

$$\arg\min_{\mathbf{w}, w_0} \|\mathbf{w}\|_2^2 + C \sum_{j=1}^{N} l(\mathbf{w}, w_0, \mathbf{x}^{(j)}, y^{(j)}) \tag{5}$$

Here $l(\mathbf{w}, w_0, \mathbf{x}^{(j)}, y^{(j)})$ is the *hinge loss* of the j-th instance:

$$l(\mathbf{w}, w_0, \mathbf{x}^{(j)}, y^{(j)}) = \max\left(1 - y^{(j)}(\mathbf{w}^T\mathbf{x}^{(j)} + w_0), 0\right)$$

Note that this objective value grows with N and C. In this homework, we will use an equivalent formalization, by multiplying the objective with $\frac{1}{NC}$, so that the objective and its gradient have the same scale as N and C changes.

$$\arg\min_{\mathbf{w}, w_0} \frac{1}{NC} \|\mathbf{w}\|_2^2 + \frac{1}{N} \sum_{j=1}^{N} l(\mathbf{w}, w_0, \mathbf{x}^{(j)}, y^{(j)}) \tag{6}$$

We can use stochastic gradient descent to optimize this objective. The update rule for $\mathbf{w}_i$ with the j-th instance will be

$$\mathbf{w}_i^{(t+1)} \leftarrow \mathbf{w}_i^{(t)} - \eta \, \partial_{\mathbf{w}_i} \tilde{L}^{(j)}(\mathbf{w}^{(t)}, w_0^{(t)}) \tag{7}$$

where $\tilde{L}^{(j)}$ is approximation of the loss function using only the j-th sample

$$\tilde{L}^{(j)}(\mathbf{w}, w_0) = \frac{1}{NC} \|\mathbf{w}\|_2^2 + l(\mathbf{w}, w_0, \mathbf{x}^{(j)}, y^{(j)}) \tag{8}$$

and $\partial_{\mathbf{w}_i} \tilde{L}^{(j)}(\mathbf{w}^{(t)}, w_0^{(t)})$ is the subgradient of the approximate loss over $\mathbf{w}_i$. A similar update rule can be applied to w_0.

1. *(4 points)* Write the stochastic gradient descent update rules for both $\mathbf{w}$ and w_0 in linear SVMs.

2. Implement SGD for linear SVMs. Initially start all the weights at 0.

3. *(9 points)* Using `validation.csv` as your training set, run 50 passes over the dataset (i.e. 50×1000 SGD updates), using $\eta = 0.1$ and $C = 1$. Plot the loss in Eq. (6) after each pass.

4. *(9 points)* Using the $\hat{\mathbf{w}}$, $\hat{w_0}$ learned after 50 passes, report:

   (a) The prediction error on `test.csv` (test error)

   (b) The prediction error on `validation.csv` (training error)

   (c) $\|\mathbf{w}\|$

5. *(18 points)* Change C to 100 and repeat what you did in the previous two questions, using the same values for η and the number of passes.

6. *(10 points)* Compare the results you get using $C = 100$ vs. $C = 1$ on the test/training error and $\|\mathbf{w}\|$. Explain the difference using the bias-variance tradeoff. (Hint: Which setting has higher variance?)

4 Extra Credit: Fourier Features and getting state of the art on MNIST [75 Points]

Let us now try to get state of the art on MNIST (and replicate the experiments done in class).

1. *(50 points)* Obtain less than 1.2% test error on the binary classification problem (and remember to submit your code). Hint: here is that I did in class:
   (a) I first PCAed the images down to 30 dimensions, using the projection obtained from the top 30 directions of the covariance matrix.
   (b) Then I made around 100K Fourier features of the form, where the i-th features is $\phi_i(x) = \cos(w_i^\top x)$ where w_i sampled from a Gaussian $N(0, \sigma^2 I)$ where $\sigma^2 = E[\|x\|^2]/4$. The choice of $1/4$ is a reasonable heuristic in practice (provided the data are near to mean 0, which is the case after our PCA). Here i is the i-th feature.
   (c) I then used block coordinate ascent.
2. *(25 points)* Try using the same features using logistic regression or an SVM and report what you get. Did you find an improvement?

5 Extra Credit: Boosting [30 Points]

The boosting theory claims that one can boost the performance (on the training set) of any weak learner arbitrarily high, provided the weak learned could always perform slightly better than chance. We will now verify this in the AdaBoost framework. (Note that this assumption is rather strong and is specifically only about the training set).

1. (*7 points*) Given a set of N observations (x^j, y^j) where y^j is the label $y^j \in \{-1, 1\}$, let $h_t(x)$ be the weak classifier at step t and let α_t be its weight. First we note that the final classifier after T steps is defined as:

$$H(x) = sgn\left\{\sum_{t=1}^{T} \alpha_t h_t(x)\right\} = sgn\{f(x)\}$$

   Where

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$

   Show that:

$$\epsilon_{\text{Training}} = \frac{1}{N}\sum_{j=1}^{N} 1_{\{H(x^j)\neq y^j\}} \leq \frac{1}{N}\sum_{j=1}^{N} \exp(-f(x^j)y^j)$$

   Where $1_{\{H(x^j)\neq y^j\}}$ is 1 if $H(x^j) \neq y^j$ and 0 otherwise.

2. (*8 points*) The weight for each data point j at step $t+1$ can be defined recursively by:

$$w_j^{(t+1)} = \frac{w_i^{(t)} \exp(-\alpha_t y^j h_t(x^j))}{Z_t}$$

Where Z_t is a normalizing constant ensuring the weights sum to 1:

$$Z_t = \sum_{j=1}^{N} w_j^t \exp(-\alpha_t y^j h_t(x^j))$$

Show that:

$$\frac{1}{N} \sum_{j=1}^{N} \exp(-f(x^j) y^j) = \prod_{t=1}^{T} Z_t$$

3. (*15 points*) We showed above that training error is bounded above by $\prod_{t=1}^{T} Z_t$. At step t the values $Z_1, Z_2, \ldots, Z_{t-1}$ are already fixed therefore at step t we can choose α_t to minimize Z_t. Let

$$\epsilon_t = \sum_{j=1}^{m} w_j^t 1_{\{h_t(x^j) \neq y^j\}}$$

be the weighted training error for weak classifier $h_t(x)$ then we can re-write the formula for Z_t as:

$$Z_t = (1 - \epsilon_t) \exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)$$

(a) First find the value of α_t that minimizes Z_t then show that

$$Z_t^{opt} = 2\sqrt{\epsilon_t(1 - \epsilon_t)}$$

(b) Assume we choose Z_t this way. Then re-write $\epsilon_t = \frac{1}{2} - \gamma_t$ where $\gamma_t > 0$ implies better than random and $\gamma_t < 0$ implies worse than random. Then show that:

$$Z_t \leq \exp(-2\gamma_t^2)$$

You may want to use the fact that $\log(1 - x) \leq -x$ for $0 \leq x < 1$

Thus we have:

$$\epsilon_{\text{training}} \leq \prod_{t=1}^{T} Z_t \leq \exp(-2 \sum_{t=1}^{T} \gamma_t^2)$$

(c) Finally, show that if each classifier is better than random (e.g. $\gamma_t \geq \gamma$ for all t and $\gamma > 0$) that:

$$\epsilon_{\text{training}} \leq \exp(-2T\gamma^2)$$

Which shows that the training error can be made arbitrarily small with enough steps.