6.867 Machine Learning

Problem Set 3

Due Monday, October 27

Problem 1: Support Vector Machines

References: Lectures 7 & 8.

This problem introduces you to the support vector machine (SVM) and asks you to construct and test SVMs for a given data set. We have provided most of the code with some gaps for you to fill.

Suppose now that we have a labeled training set, $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, where each $\mathbf{x}_i$ is an m-dimensional vector. The possible labels are $y = -1$ and $y = +1$. We consider classifiers of the following form:

$$\hat{y}(\mathbf{x}; w_0, \mathbf{w}) = \text{sign}(w_0 + \mathbf{w}'\phi(\mathbf{x})) \tag{1}$$

where $\phi : \mathcal{R}^m \rightarrow \mathcal{R}^M$ maps each m-dimensional example $\mathbf{x}$ to an M-dimensional "feature vector" $\phi(\mathbf{x})$.

To design an SVM classifier, we will require here that the given examples are seperable w.r.t. the chosen features, i.e. there exists atleast one setting of $(w_0, \mathbf{w})$ such that this decision rule will correctly classify all of the given examples. The paramaters $(w_0, \mathbf{w}) \in \mathcal{R}^{M+1}$ are optimized to "best" separate the two classes of examples, i.e., optimized so as to maximize the "margin" (the distance from the decision boundary to the closest training). We can formally solve this problem as a contrained optimization problem where we minimize the regularization penalty function $\frac{1}{2}\|\mathbf{w}\|^2$ subject to the constraints that each example is classified correctly.

$$\text{minimize} \qquad \tfrac{1}{2}\textstyle\sum_{k>0} w_k^2 \tag{2}$$

$$\text{s.t.} \quad y_i(w_0 + \mathbf{w}'\phi(\mathbf{x}_i)) - 1 \geq 0 \quad \forall i \tag{3}$$

Essentially, we require that all the +1 labeled examples (their feature vectors) live in the half space $H = \{\phi : (w_0 + \mathbf{w}'\phi) \geq 1\}$, while -1 labeled examples fall into the opposite, gap separated, half space $H = \{\phi : (w_0 + \mathbf{w}'\phi) \leq -1\}$. The hyperplane in the middle of the gap, $H_0 = \{\phi : w_0 + \mathbf{w}'\phi = 0\}$, then determines the maximum margin seperating hyperplane in R^M.

The problem with the above "primal" formulation is that we have to explicitly represent the feature vectors $\phi(\mathbf{x})$ that may be quite high dimensional vectors. As shown in the lecture, we can derive the following "dual" optimization problem that only involves inner products between the feature vectors; inner products are real numbers that can be typically evaluated without ever explicitly constructing the feature vectors. Specifically, the dual optimization problem takes the following form

$$\begin{aligned} \text{maximize} \quad & J(\alpha) = \textstyle\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) && (4) \\ \text{s.t.} \quad & \alpha_i \geq 0 \quad \forall i && (5) \\ & \textstyle\sum_i \alpha_i y_i = 0 && (6) \end{aligned}$$

where $\{\alpha_i\}$ are the Lagrange multipliers associated to the inequality constraints and $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)'\phi(\mathbf{x}_j)$ is the *kernel function*.

The solution to the original problem is then recovered as

$$\begin{aligned} \hat{\mathbf{w}} &= \arg \min_{\mathbf{w} \in \mathcal{R}^M} \left\{ f(\mathbf{w}) - \sum_i \hat{\alpha}_i y_i \mathbf{w}' \phi(\mathbf{x}_i) \right\} && (7) \\ &= \sum_i \hat{\alpha}_i y_i \phi(\mathbf{x}_i) && (8) \end{aligned}$$

where $\hat{\alpha}_i$ are the optimal multipliers which solve the dual problem. The value of the offset (bias) parameter w_0 needs to be recovered separately as discussed below.

The dual problem exposes an important feature of SVM's. Typically, only a few of the constraints are active at the optimum so that the solution is characterized by just those critical examples corresponding to active contraints. Let $S = \{i | \hat{\alpha}_i > 0\}$ denote the set of active constraints. The optimal parameters $\hat{w}$ are a linear combination (in feature space) of these critical examples (called *support vectors*):

$$\hat{\mathbf{w}} = \sum_{i \in S} \hat{\beta}_i \phi(\mathbf{x}_i) \tag{9}$$

where $\hat{\beta}_i = y_i \hat{\alpha}_i$.

We can recover $\hat{w}_0$ based on the active constraints by observing that for each active constraint $i \in S$, we must have

$$y_i(\hat{w}_0 + \hat{\mathbf{w}}'\phi(\mathbf{x}_i)) = y_i \left(\hat{w}_0 + \sum_{j \in S} \hat{\beta}_j \phi(\mathbf{x}_j)'\phi(\mathbf{x}_i) \right) = y_i \left(\hat{w}_0 + \sum_{j \in S} \hat{\beta}_j K(\mathbf{x}_j, \mathbf{x}_i) \right) = 1 \tag{10}$$

or (by multiplying both sides by y_i), $\hat{w}_0 + \sum_{j \in S} \hat{\beta}_j K(\mathbf{x}_j, \mathbf{x}_i) = y_i$. In other words, the margin constraint is satisfied with equality. This is because any active constraint wouldn't be satisfied without a non-zero Lagrange multiplier; the Lagrange multiplier only changes the solution to the extent that the constraint is just barely satisfied, i.e., the inequality constraint is satisfied with equality.

Finally, in the resulting decision rule

$$\hat{y}(\mathbf{x}; \hat{w}_0, \hat{\beta}) = \text{sign}\left(\hat{w}_0 + \sum_{i \in S} \hat{\beta}_i K(\mathbf{x}_i, \mathbf{x})\right) \tag{11}$$

we compare the new example $\mathbf{x}$ to each of the support vectors (but not other training examples).

The problem:

Let's start by figuring out how to construct a valid kernel function. What is a valid kernel function? For us to be able to interpret the kernel as an inner product between some feature vectors, the matrix $K = (K_{ij})$ defined by $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, has to be positive semi-definite for any finite set of examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.[1] For example, this is the case for the simple linear kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i' \mathbf{x}_j$ since

$$K = \mathbf{X}'\mathbf{X} \tag{12}$$

where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]'$.

(1-1) (5pts) Show that the sum of any two kernels, $K(\mathbf{x}_i, \mathbf{x}_j) = K_1(\mathbf{x}_i, \mathbf{x}_j) + K_2(\mathbf{x}_i, \mathbf{x}_j)$, is a valid kernel in the sense discussed above. In other words, show that for any finite training set, K_{ij} defines a positive semi-definite matrix.

(1-2) (5pts) Show that $f(\mathbf{x}_i)K(\mathbf{x}_i, \mathbf{x}_j)f(\mathbf{x}_j)$ is a valid kernel for any real valued function $f(\mathbf{x})$ and kernel $K(\mathbf{x}_i, \mathbf{x}_j)$.

(1-3) (5pts) Show that the elementwise product of any two kernels K_1 and K_2,

$$K(\mathbf{x}_i, \mathbf{x}_j) = K_1(\mathbf{x}_i, \mathbf{x}_j)K_2(\mathbf{x}_i, \mathbf{x}_j), \tag{13}$$

is a valid kernel. (*Hints.* Use the fact that $\xi_i K(\mathbf{x}_i, \mathbf{x}_j)\xi_j$, for any $\xi_i, i = 1, \ldots, n$ can be interpreted as a kernel of the previous type; the fact that a matrix product AB is semi-definite for any semi-definite A and B; and the fact that $\text{trace}(AB) = \sum_{ij} A_{ij}B_{ij}$ (for symmetric matrices) is also a sum of eigenvalues of the square matrix product AB. *Alternate Approach.* Use the fact the any positive semi-definite matrix K may be factored as $K = R'R$ with R a real-valued $n \times n$ matrix.[2])

As an example use of these rules, let's figure out why a radial basis kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\{-\frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\} \tag{14}$$

[1] By definition, the matrix K is *positive semi-definite* if and only if $\mathbf{f}'K\mathbf{f} \geq 0$ for all $\mathbf{f} \in \mathcal{R}^n$.

[2] For instance, let $R' = SD^{1/2}$ where $K = SDS'$ is the eigendecomposition of K. Because K is positive semi-definite, the eigenvalues $d_{ii} = \lambda_i$ are non-negative and the eigenvectors (columns of S) are real-valued so that R is a real-valued matrix.

is a valid kernel.

$$\exp\{-\frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|^2\} = \exp\{-\frac{1}{2}\mathbf{x}_i'\mathbf{x}_i + \mathbf{x}_i'\mathbf{x}_j - \frac{1}{2}\mathbf{x}_j'\mathbf{x}_j\} \quad (15)$$

$$= \overbrace{\exp\{-\frac{1}{2}\mathbf{x}_i'\mathbf{x}_i\}}^{f(\mathbf{x}_i)} \cdot \exp\{\mathbf{x}_i'\mathbf{x}_j\} \cdot \overbrace{\exp\{-\frac{1}{2}\mathbf{x}_j'\mathbf{x}_j\}}^{f(\mathbf{x}_j)} \quad (16)$$

Here $\exp\{\mathbf{x}_i'\mathbf{x}_j\}$ is a sum of simple products $\mathbf{x}_i'\mathbf{x}_j$ and therefore a kernel based on the first and the third rules; the second rule allows us to incorporate $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$.

We have provided you with four matlab functions `K1` (linear kernel) ,`K2` (2nd order polynomial kernel), `K3` (3rd order polynomial kernel), and `Kr` (radial basis kernel) which use the above rules to generate the kernel matrices. Each of these functions takes as inputs two sets of examples (stored as row vectors) and returns the corresponding kernel matrix. So, for example, `K1(X1,X2)` returns a matrix of simple inner products between the rows of matrices `X1` and `X2`. The number of rows in the returned matrix corresponds to the number of rows in `X1`, and the number of columns is the number of rows in `X2`.

We are now ready to generate MATLAB code to build and execute SVMs. You will need to write bits and pieces of the code that we have already mostly written. You can, of course, write your own svm code if you like. You will test your SVM code using data provided in `svm_data.mat` which contains two structures `svm_train_data` and `svm_test_data`. Load this data into MATLAB with the `load` command. Each structure contains an $n \times m$ matrix `X` containing all examples ($m = 2$) (as row vectors) and the corresponding labels stored in an $n \times 1$ column vector `y`. You may display either data set using the provided subroutine `svm_plot_data`.

(1-4) (20pts) You can solve the SVM optimization problem (obtain α's) by using MATLAB's standard quadratic programming routine `x=quadprog(H,f,A,b)`. This routine solves the following generic quadratic programming problem:

$$\text{minimize} \quad \tfrac{1}{2}x'Hx + f'x \quad (17)$$

$$\text{s.t.} \quad Ax \leq b \quad (18)$$

(you are welcome to use other variations of `quadprog(...)` if you like; some versions permit explicit equality constraints, upper and lower bounds, etc.). For this routine to be useful you need to set up the matrices `H`, `A`, and vectors `f` and `b` so that the solution `x` will be the α's that we are after. Complete these steps in subroutine `SVM = svm_build(data,kernel)`, the skeleton of which we have provided. Your function takes a MATLAB function handle to indicate which kernel function to use (e.g., `svm_build(data,@K1)`). You then call the kernel function using `feval` (i.e., `K = feval(kernel,X,X)`) to get the kernel matrix. Your procedure should call `quadprog` to solve the dual problem, which requires you to set up the appropriate matrices described above. The vector β is defined in the skeleton code as before, $\hat{\beta}_i = y_i\hat{\alpha}_i$,

but here only for support vectors. You also need to compute the offset parameter `w0` from the solution. We have already provided means for storing the necessary values. Run and debug your routine (using the training data) before you proceed.

(1-5) (5pts) Write a MATLAB subroutine `function f = svm_discrim_func(X,SVM)` to evaluate the discriminant function $f(\mathbf{x}; \hat{w}_0, \hat{\beta}) = \hat{w}_0 + \sum_{i \in S} \hat{\beta}_i K(\mathbf{x}_i, \mathbf{x})$ for each input in `X` given the specifications in `SVM`. (we have provided you with a skeleton). You will need this function to classify the test data and also to run our display function `svm_plot` which will plot the decision boundary of your SVM.

(1-6) (5pts) For each of the kernels you implemented in part 1, build a SVM based upon the training data we have provided. First, check to verify that your SVM does indeed correctly classify all of these training examples (otherwise you have a bug somewhere). Once everything checks out, record the specifications for each of your SVMs. Use the procedure `svm_plot` to display the training data and overlay a plot of the decision boundary and support vectors for each of your SVMs. Submit one plot for each of your kernel functions.

(1-7) (5pts) Apply each of the SVMs you have constructed to the test data. Report the number of misclassified examples for each SVM. Which one did the best?

Problem 2: Regularized Parameter Estimation

Thus far, we have focused on the maximum likelihood approach to parameter estimation. Given a parameterized family of probability models $P(x|\theta)$ and a data set $x^n = (x_1, \ldots, x_n)$ comprised of independent samples $x_i \sim P(\cdot|\theta)$, we *fit* the model to the data so as to maximize the likelihood (or log-likelihood) of all samples. This gives the *maximum-likelihood* (ML) estimate of the parameters:

$$\hat{\theta}_{ML} = \arg\max_{\theta} \log P(x^n|\theta) \tag{19}$$

This approach does not express any prior bias as to which values of θ we should prefer when data is limited.

In the sequel, we consider a regularized approach to parameter estimation. Here, we specify a *prior model* $p(\theta)$ over the set of allowed parameter settings Θ. Given a prior model, we may then employ *Bayes rule* to compute the posterior probability of θ given the observations:

$$p(\theta|x^n) = \frac{P(x^n|\theta)p(\theta)}{P(x^n)} \tag{20}$$

where

$$P(x^n) = \int_{\Theta} P(x^n|\theta)p(\theta)d\theta \tag{21}$$

Then, we fit the model to the data by maximizing the (log-) probability of θ conditioned on the data,

$$\begin{aligned}
\hat{\theta}_{MAP} &= \arg\max_{\theta} \log p(\theta|x^n) && (22)\\
&= \arg\max_{\theta} \{\log P(x^n|\theta) + \log p(\theta) - \log P(x^n)\} && (23)\\
&= \arg\max_{\theta} \{\log P(x^n|\theta) + \log p(\theta)\} && (24)
\end{aligned}$$

Note that we have dropped the $-\log P(x^n)$ term as this does not depend upon θ and does not affect the parameter estimate. Hence, we do not need to explicitly evaluate the integral in (21). This may be viewed as a penalized log-likelihood criterion, i.e. we maximize $J(\theta) = \log p(x^n; \theta) - f(\theta)$ subject to the regularization penalty $f(\theta) = -\log p(\theta)$. The parameter estimate $\hat{\theta}_{MAP}$ is known as the *maximum a posterior* (MAP) estimate.

In this problem you will construct MAP estimates for the probabilities of a (potentially biased) M-sided die, i.e. $x \in \{1, \ldots, M\}$. We consider the fully-parameterized representation $P(x = k) = \theta_k$, where $0 \leq \theta_k \leq 1$ for $k = 1, \ldots, M$ and $\sum_k \theta_k = 1$. This simple model has many relevant applications.

Consider a document classification task, where we need class-conditional distributions over words in the documents. Suppose we only consider words $1, \ldots, M$ (for relatively large M). Each word in the document is assumed to have been drawn at random from the distribution $P(x = k|y, \theta) = \theta_{k|y}$, where $\sum_{k=1}^{M} \theta_{k|y} = 1$ for each class y. Thus the selection of words according to the distribution $\theta_{\cdot|y}$ can be interpreted as a (biased) M-sided die.

Now, the probability of generating all words $x_1, \ldots, x_n$ in a document of length n would be

$$P(x^n|y,\theta) = \prod_{i=1}^{n} P(x_i|y,\theta) = \prod_{i=1}^{n} \theta_{x_i|y} \tag{25}$$

assuming the document belongs to class y. Note that this model cares about how many times each word occurs in the document. It is a valid probability model over the set of words in the document.

Since we typically have very few documents per class, it is important to requlargize the parameters, i.e., provide a meaningful prior answer to the class conditional distributions.

Let's start by briefly revisiting ML estimation of the (biased) M-sided die. Similarly to calculations you have already performed, the ML estimate of the parameter θ from n samples is given by the *empirical distribution*:

$$\hat{\theta}_x = \frac{n(x)}{n} = \hat{P}(x) \;\; \text{(empirical distribution)} \tag{26}$$

where $n(x)$ is the number of times value x occured in n samples. $n(x)$ is also a *sufficient statistic* for θ_x as it is all we need to know from the available n samples in order to estimate θ_x.

(2-1) (5pts) Derive the above ML estimate by maximizing the log-likelihood directly with respect to $\theta_1, \ldots, \theta_M$, using a Lagrange multiplier to enforce the normalization constraint $\sum_k \theta_k = 1$.

Next, we consider MAP estimation. To do so, we must introduce a prior distribution over the θ's. A natural choice for this problem is the Dirichlet distribution

$$p(\theta;\beta) = \frac{1}{Z(\beta)} \prod_{k=1}^{M} \theta_k^{\beta_k} \tag{27}$$

with non-negative hyperparameters $\beta = (\beta_k > 0, k = 1, \ldots, M)$ and where $Z(\beta)$ is just a normalization constant (which you do not need to evaluate in this problem).

(2-2) (10pts) First, consider this prior model (ignoring the data for the moment). What value of θ is most likely under this prior model? That is, compute

$$\hat{\theta}(\beta) = \arg\max_{\theta} \log p(\theta;\beta) \tag{28}$$

This is the *a priori* estimate of θ before observing any data.

(2-3) (10pts) Next, given the data x^n, compute the MAP estimate of θ as a function of the hyperparameters β and the data x^n (use the sufficient statistics $n(x)$):

$$\hat{\theta}_{MAP}(x^n;\beta) = \arg\max_{\theta} \log p(\theta|x^n,\beta) \tag{29}$$

Note that you do not need to calculate $Z(\beta)$ in order to perform this optimization; you can optimize the penalized log-likelihood $J(\theta) = \log P(x^n|\theta) - f(\theta;\beta)$ with a simple penalty function $f(\theta;\beta)$, as discussed above. Thus we do not have to evaluate the full posterior distribution $p(\theta|x^n,\beta)$ in order to perform the regularization.

(2-4) (10pts) Show that your MAP estimate may be expressed as a convex combination of the a priori estimate $\hat{\theta}(\beta)$ and the ML estimate $\hat{\theta}_{ML}(x^n)$. The means that we may write

$$\hat{\theta}_{MAP}(x^n;\beta) = (1-\lambda)\hat{\theta}_{ML}(x^n) + \lambda\hat{\theta}(\beta) \tag{30}$$

for some $\lambda \in [0,1]$. Note that the same convex combination holds for each component θ_x. Determine λ as a function of the number of samples n and the hyperparameters β.

As this shows, one way of thinking of a prior distribution is that is it is a proxy for any data we have observed in the past but no longer have available. The normalized parameters $\hat{\beta}_i = \beta_i/N$, where $N = \sum_i \beta_i$, express our prior estimate of the parameters θ while the normalization parameter N expresses how strongly we believe in that prior estimate.

Let's briefly explore the use of this prior in a document classification task. Our data is slightly inappropriate given the model but we will use it regardless. The data `docdata.mat` provides matrices `xtrain`, `xtest`, and label vectors `ytrain` and `ytest`. Each document is represented as a binary (row) vector, where each component indicates whether a specific word (out of 600 words listed in `words.txt`) appears in the document or not. To use our model appropriately we would need to know how many times each word occured in the document, not just whether they did or not. This information is not available in the data, however. We will use the data regardless, and stubbornly interpret the binary numbers as word counts. The two classes of documents, $y = 1$ and $y = 2$, correspond to email messages posted to online discussion boards, one for users of MS Windows and another for users of X Windows. We want a classifier that would label a new message as belonging to one of the groups.

(2-5) (10pts) Generate the prior parameter estimate from both training and test documents using `theta_prior = prior_theta([xtrain;xtest])`. `prior_theta` corrsponds to $\hat{\theta}(\beta)$ discussed earlier, and is the same for both classes. For this estimation to be relevant in practice we would need to have access to the test documents. Note that for classification it is fine for us to use the words in the test documents so long as we do not see the labels.

The ML estimate $\hat{\theta}$ can be found with `theta_hat = ml_theta(xtrain,ytrain)`. Note that `theta_hat` has two rows of parameters, one corresponding to each class.

Plot the test errors as a function of different convex combinations of estimates, specified by parameter λ discussed earlier. Use `lambda = [0:0.1:0.9]`. You can evaluate the discriminant function corresponding to the simple two class model using

`log_likel_ratio(theta,xtest)`. For each row x^n in `xtest`, the function returns $\log P(x^n|\theta_1, y = 1) - \log P(x^n|\theta_2, y = 2)$.

Can you explain the resulting curve?

(2-6) (5pts) How would you set λ in practice? We would need to fix the value of this parameter before seeing any test labels.