Homework 3, due Monday March 23 COMS 4771 Spring 2015

Problem 1 (Evaluating soft-margin SVMs). Download the spam data set from Courseworks (`spam.mat`). This is the data set described in the *ESL* text for a binary classification problem of predicting whether an e-mail is spam or not. The training data and test data are in the usual format. You can read about the features in *ESL* (Chapter 9.1, pages 300–301).

There is also MATLAB code on Courseworks for training and predicting with a (homogeneous) soft-margin kernel SVM classifier (`hw3_train_ksvm.m` and `hw3_test_ksvm.m`). See the files themselves for information on how to use them. They rely on the MATLAB Optimization Toolbox.

Write a MATLAB script (call it `hw3.m`) that carries out a thorough evaluation of the (homogeneous) soft-margin SVM with the Gaussian kernel on the spam data set. The Gaussian kernel

$$K(\boldsymbol{x}, \boldsymbol{x}') := \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2h}\right)$$

has a parameter $h > 0$ which is called the *bandwidth*. It is well-known that the bandwidth needs to be appropriately tuned in order to get good predictive performance. The soft-margin SVM optimization itself also has a regularization parameter $\lambda > 0$ that needs to be tuned. For a proper evaluation, you need to tune these parameters (say, using hold-out or cross-validation error) *using only the training data.* To make this exercise as realistic, *do not touch the test data until the end.*

Some suggestions:

- If there is any randomness in your script (e.g., to choose a random hold-out set), fix a seed for the pseudorandom number generate. (In MATLAB, use the `rng()` function to do this.) This will help with reproducibility and debugging.
- It is highly advisable to *standardize* the features using the feature means and standard deviations from the training data.

  If you want to try other feature pre-processing, you should make sure you evaluate it only using the training data (again, using something like hold-out error or cross-validation error).
- You can use the MATLAB function `pdist2()` on this assignment to compute pairwise Euclidean distances. To get squared distances, use something like `pdist2(data,data).^2`.
- Some sensible values of h are various quantiles of all pairwise squared Euclidean distances between points in the training set. I suggest the 0.1, 0.25, 0.5 (the median), 0.75 and 0.9 quantiles. The MATLAB function `quantile()` is useful for this.
- The parameter λ is more difficult to determine. I suggest trying several values in a geometrically decreasing sequence starting from 1 (e.g., $1, e^{-1}, e^{-2}, \ldots, e^{-10}$).
- The optimization routines in `hw3_train_ksvm.m` might not actually set dual variables to zero, so if you simply use `nnz()` to count the number of non-zero dual variables, it may simply be all of them. Instead, I suggest only counting a dual variable as non-zero if it is larger than $1/(10^6 n\lambda)$, where n is the number of data points (i.e., total number of dual variables), and λ is the regularization parameter used.

Write-up: Your write-up should contain a **complete and precise description of your evaluation process**—enough so that anyone can replicate your results. The write-up should not reference your MATLAB script; provide all necessary pseudocode in the write-up (e.g., precisely how you determine a hold-out set, how you choose bandwidth and regularization parameters). Provide details of hold-out/cross-validation errors for different parameters you try (perhaps organized in a plot or table), and the values of the tuning parameters you ultimately choose. You should ultimately end up with one final classifier: conclude the report with the training and test errors of this classifier (*this is the only time the test data should be used*), the breakdown of training and test errors into false positives and false negatives, as well as the number of support vectors. I suggest making a *confusion matrix*: a 2×2 matrix as follows:

	Predict -1	Predict $+1$
Label -1	% true positives	% false positives
Label $+1$	% false negatives	% true negatives

Note: You may consider using other soft-margin kernel SVM solvers (that are either included in MATLAB or are freely available and usable in MATLAB), but (1) please first check with the instructors, and (2) please make sure it is indeed solving the correct optimization problem. (Some packages solve a different form of SVM where the hinge loss is squared, and yet others solve something called ν-SVM.) I haven't had much luck with the MATLAB solvers in the Statistics Toolbox (`svmtrain` and `fitcsvm`), so I don't recommend them. **You should not use any existing parameter tuning procedures or scripts**—you should write these yourself for this exercise.

Problem 2 (Convex optimization)**.** In this problem, we'll consider an optimization formulation for *linear regression*—this is the supervised learning problem where the output space is $\mathcal{Y} = \mathbb{R}$ (rather than a categorical value like $\mathcal{Y} = \{0, 1\}$). Instead of classification error, the goal here is to learn a function $f : \mathbb{R}^d \to \mathbb{R}$ so as to minimize the expected *squared error* with respect to a distribution P over $\mathbb{R}^d \times \mathbb{R}$: $\mathbb{E}[(f(\boldsymbol{X}) - Y)^2]$ (the expectation is taken with respect to the random couple $(\boldsymbol{X}, Y)$ which has distribution P). We consider the case where f is a *linear function* represented by a weight vector $\boldsymbol{w} \in \mathbb{R}^d$. Let S be an i.i.d. sample from P; consider the following optimization problem:

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \quad \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2 + \frac{1}{|S|} \sum_{(\boldsymbol{x}, y) \in S} (\langle \boldsymbol{w}, \boldsymbol{x} \rangle - y)^2 .$$

(Above, $\lambda > 0$ is assumed to be a positive scalar.)

(a) Show that this optimization problem is a convex optimization problem. Do this from the definition(s) of convexity (possibly the ones for differentiable functions) and the composition rules from lecture. Do not assume that any particular function (even $z \mapsto z^2$) is convex.

(b) Derive a gradient descent algorithm for solving the optimization problem (following the recipe from lecture). No need to specify the initial solution $\boldsymbol{w}^{(1)}$ or the step sizes η_t.

(c) Suppose we add the following constraints to the optimization problem:

$$w_i^2 \leq 1 \quad \text{for all } i \in \{1, 2, \ldots, d\}.$$

Is the optimization problem still convex? Briefly explain why or why not.

(d) Same as (c), but with the following constraints instead (assuming d is even):

$$w_i = w_{i+1} \quad \text{for all } i \in \{1, 3, 5, \ldots, d-1\}.$$

(Hint: can you write equality constraints as a pair of inequality constraints?)

(e) Same as (c), but with the following constraints instead:

$$w_i^2 = 1 \quad \text{for all } i \in \{1, 2, \ldots, d\}.$$

Problem 3 (Consistent Classifier Algorithm)**.** In this problem, we'll direct analyze the Consistent Classifier Algorithm for a finite function class $\mathcal{F}$. We shall make the realizability assumption here: there exists some $f^\star \in \mathcal{F}$ such that $\operatorname{err}(f) = 0$, where the true error is defined w.r.t. a fixed distribution P. The Consistent Classifier Algorithm takes as input training data S, and returns a classifier $\hat{f} \in \mathcal{F}$ with $\operatorname{err}(\hat{f}, S) = 0$ (and if more than one such function exists, ties are broken arbitrarily unless otherwise stated).

The goal is to show the following: for any $\varepsilon \in (0,1)$ and $\delta \in (0,1)$, if the number of training data S (assumed to be an i.i.d. sample from P) satisfies

$$|S| \geq \frac{\ln(|\mathcal{F}|/\delta)}{\varepsilon},$$

then with probability at least $1-\delta$ (over the random draw of S), the classifier $\hat{f}$ returned by the Consistent Classifier Algorithm has true error $\operatorname{err}(\hat{f}) \leq \varepsilon$.

(a) Consider any fixed classifier $f \in \mathcal{F}$, and suppose $\operatorname{err}(f) > \varepsilon$. Let A denote an i.i.d. sample from P. Derive a bound on the probability that $\operatorname{err}(f, A) = 0$. The bound should decrease exponentially with $|A|$. You may want to use the fact that for any $z \in \mathbb{R}$, $1 - z \leq e^{-z}$.

(b) Explain why your answer to (a) implies the desired claim. (You should use the union bound.)

(c) Now suppose instead that the training examples in $S = ((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})$ are independent but not necessarily identically distributed. For each $i \in \{1, 2, \ldots, n\}$, let P_i denote the distribution of $(x^{(i)}, y^{(i)})$.

We now *re-define* P to be the uniform mixture of $P_1, P_2, \ldots, P_n$. That is, a random example (x, y) from P is generated as follows: first pick some $I \in \{1, 2, \ldots, n\}$ uniformly at random, then randomly draw (x, y) from P_I.

Explain why $|S| \geq \ln(|\mathcal{F}|/\delta)/\varepsilon$ still suffices to guarantee that $\Pr(\operatorname{err}(\hat{f}) \leq \varepsilon) \geq 1 - \delta$.

(d) We return to the original setting where S is an i.i.d. sample from P. Suppose the classifiers in $\mathcal{F}$ are ordered in a list as $f_1, f_2, f_3, \ldots$; the ordering is determined *before any data is ever seen.* For instance, for a collection of "union of rectangles" functions, you may order the functions so that for any $k \in \mathbb{N}$, functions involving k rectangles come before those involving $k+1$ rectangles.

Consider the following version of the Consistent Classifier Algorithm: given training data S, pick the *first* classifier $f_t \in \mathcal{F}$ in the list such that $\operatorname{err}(f_t, S) = 0$, and return $\hat{f} := f_t$.

Using the argument from lecture, show that with probability at least $1 - \delta$, if the classifier returned by the algorithm is $\hat{f} = f_t$ (for some $t \in \{1, 2, \ldots, |\mathcal{F}|\}$), then

$$\operatorname{err}(\hat{f}) \leq \frac{2\ln(t(t+1)/\delta)}{|S|}.$$

This implies, for instance, that with high probability, if a classifier within the first 100 classifiers on the list is found to have zero training error, then it has true error $O\Big(\frac{\log(1/\delta)}{|S|}\Big)$, which does not depend on the total number of classifiers $|\mathcal{F}|$ at all! In fact, $\mathcal{F}$ need not even be finite—all that is required is that $\mathcal{F}$ be countable.

Hint: You can mimic the argument from lecture very closely, except the "failure events" $\mathcal{E}_f$ should be defined differently. You may use the (easily verified) fact that $\sum_{t=1}^{\infty} \frac{1}{t(t+1)} \leq 1$.

Problem 4 (Additional practice for midterm)**.** This problem will not be graded; it is simply additional practice for the midterm.

(a) Which of the following classifiers are *linear classifiers* in $\mathbb{R}^d$?

- $f(\boldsymbol{x}) = \operatorname{sign}\left(\frac{1}{1+\exp(-\langle \boldsymbol{w}, \boldsymbol{x}\rangle)}\right)$ for some vector $\boldsymbol{w} \in \mathbb{R}^d$.
- The 1-NN classifier based on the following training data ($d = 2$):

	x_1	x_2	y
Example 1	-1	-1	-1
Example 2	$+1$	$+1$	$+1$
Example 3	-1	$+3$	-1
Example 4	$+1$	$+5$	$+1$

- A classifier trained using Kernelized Online Perceptron with the kernel $K(\boldsymbol{x}, \tilde{\boldsymbol{x}}) := \langle \boldsymbol{x}, \boldsymbol{A}\tilde{\boldsymbol{x}}\rangle$, where $\boldsymbol{A}$ is a $d \times d$ symmetric positive definite matrix.

(b) Consider the model $\mathcal{P}$ of distributions over the positive integers $\mathbb{N}$ with probability mass functions given by

$$P_\theta(k) = (1-\theta)^{k-1}\theta, \quad k \in \mathbb{N}$$

for parameter $\theta \in (0,1)$ called the *success parameter*.

- Derive a formula for the maximum likelihood estimator for the success parameter given data $x_1, x_2, \ldots, x_n \in \mathbb{N}$.
- Let $X \sim P_\theta$ for some $P_\theta \in \mathcal{P}$; here θ is an unknown success parameter. Let $\hat{\theta}$ denote the maximum likelihood estimate of the success parameter given the training data $S = \{X\}$ (well, *datum*). What is the expectation of $\hat{\theta}$? (You may leave your answer in terms of θ and a summation over $\mathbb{N}$ if you like.)

(c) Consider the model $\mathcal{P}$ of probability distributions over the non-negative reals $\mathbb{R}_+$ with probability densities on the given by

$$p_\lambda(x) = \lambda e^{-\lambda x}, \quad x \in \mathbb{R}_+$$

for parameter $\lambda > 0$ called the *rate parameter*.

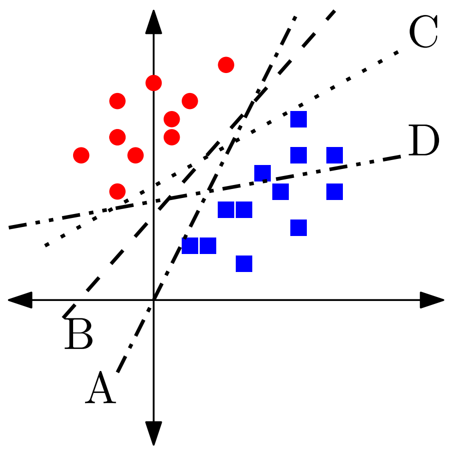
- Derive a formula for the maximum likelihood estimator for the rate parameter given data $x_1, x_2, \ldots, x_n \in \mathbb{R}_+$.
- Let $X \sim p_\lambda$ for some $p_\lambda \in \mathcal{P}$; here λ is an unknown rate parameter. Let $\hat{\lambda}$ denote the maximum likelihood estimate of the rate parameter given the training data $S = \{X\}$ (well, *datum*). What is the expectation of $\hat{\lambda}$? (You may leave your answer in terms of λ and an integral over $\mathbb{R}_+$ if you like.)

(d) Explain how to "kernelize" *one step* of the gradient descent algorithm from Problem 2(b). That is, assume $K\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a positive definite kernel with feature map $\boldsymbol{\phi}\colon \mathbb{R}^d \to \mathbb{R}^D$, and assume you have a black box subroutine for computing $K(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ for any $\boldsymbol{x}, \tilde{\boldsymbol{x}} \in \mathbb{R}^d$. Give pseudocode for evaluating $\langle \boldsymbol{w}^{(2)}, \boldsymbol{\phi}(\boldsymbol{z})\rangle$ for any $\boldsymbol{z} \in \mathbb{R}^d$, where $\boldsymbol{w}^{(2)} \in \mathbb{R}^D$ is the result of one step of the gradient descent algorithm for solving the optimization problem

$$\min_{\boldsymbol{w} \in \mathbb{R}^D} \quad \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \frac{1}{|S|}\sum_{(\boldsymbol{x},y)\in S}\left(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x})\rangle - y\right)^2.$$

You can assume that $\boldsymbol{w}^{(1)} = \mathbf{0}$ and $\eta_t := \eta$ for some fixed $\eta \in (0, 1/\lambda)$.

(e) The figure below depicts the decision boundaries of four linear classifiers (labeled A, B, C, D), and the locations of some labeled training data (positive points are squares, negative points are circles).



Consider the following algorithms for learning linear classifiers which could have produced the depicted classifiers given the depicted training data:

- (Batch) Perceptron
- Online Perceptron (making one pass over the training data)
- ERM for homogeneous linear classifiers
- An algorithm that exactly solves the SVM problem

What is the most likely correspondence between these algorithms and the depicted linear classifiers? Explain your matching.