# CS 229, Public Course
# Problem Set #2: Kernels, SVMs, and Theory

~~1.~~ **Kernel ridge regression**

In contrast to ordinary least squares which has a cost function

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2,$$

we can also add a term that penalizes large weights in $\theta$. In *ridge regression*, our least squares cost is regularized by adding a term $\lambda\|\theta\|^2$, where $\lambda > 0$ is a fixed (known) constant (regularization will be discussed at greater length in an upcoming course lecutre). The ridge regression cost function is then

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2 + \frac{\lambda}{2}\|\theta\|^2.$$

(a) Use the vector notation described in class to find a closed-form expreesion for the value of $\theta$ which minimizes the ridge regression cost function.

(b) Suppose that we want to use kernels to implicitly represent our feature vectors in a high-dimensional (possibly infinite dimensional) space. Using a feature mapping $\phi$, the ridge regression cost function becomes

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(\theta^T \phi(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2}\|\theta\|^2.$$

Making a prediction on a new input $x_{\text{new}}$ would now be done by computing $\theta^T\phi(x_{\text{new}})$. Show how we can use the "kernel trick" to obtain a closed form for the prediction on the new input without ever explicitly computing $\phi(x_{\text{new}})$. You may assume that the parameter vector $\theta$ can be expressed as a linear combination of the input feature vectors; i.e., $\theta = \sum_{i=1}^{m}\alpha_i\phi(x^{(i)})$ for some set of parameters $\alpha_i$.
[Hint: You may find the following identity useful:

$$(\lambda I + BA)^{-1}B = B(\lambda I + AB)^{-1}.$$

If you want, you can try to prove this as well, though this is not required for the problem.]

~~2~~. **$\ell_2$ norm soft margin SVMs**

In class, we saw that if our data is not linearly separable, then we need to modify our support vector machine algorithm by introducing an error margin that must be minimized. Specifically, the formulation we have looked at is known as the $\ell_1$ norm soft margin SVM. In this problem we will consider an alternative method, known as the $\ell_2$ norm soft margin SVM. This new algorithm is given by the following optimization problem (notice that the slack penalties are now squared):

$$\begin{array}{rl} \min_{w,b,\xi} & \frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2 \\ \text{s.t.} & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i,\ i = 1,\ldots,m \end{array}.$$

(a) Notice that we have dropped the $\xi_i \geq 0$ constraint in the $\ell_2$ problem. Show that these non-negativity constraints can be removed. That is, show that the optimal value of the objective will be the same whether or not these constraints are present.

(b) What is the Lagrangian of the $\ell_2$ soft margin SVM optimization problem?

(c) Minimize the Lagrangian with respect to $w$, $b$, and $\xi$ by taking the following gradients: $\nabla_w \mathcal{L}$, $\frac{\partial \mathcal{L}}{\partial b}$, and $\nabla_\xi \mathcal{L}$, and then setting them equal to 0. Here, $\xi = [\xi_1, \xi_2, \ldots, \xi_m]^T$.

(d) What is the dual of the $\ell_2$ soft margin SVM optimization problem?

??3. **SVM with Gaussian kernel**

Consider the task of training a support vector machine using the Gaussian kernel $K(x,z) = \exp(-\|x-z\|^2/\tau^2)$. We will show that as long as there are no two identical points in the training set, we can always find a value for the bandwidth parameter $\tau$ such that the SVM achieves zero training error.

(a) Recall from class that the decision function learned by the support vector machine can be written as

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x) + b.$$

Assume that the training data $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ consists of points which are separated by at least a distance of $\epsilon$; that is, $||x^{(j)} - x^{(i)}|| \geq \epsilon$ for any $i \neq j$. Find values for the set of parameters $\{\alpha_1, \ldots, \alpha_m, b\}$ and Gaussian kernel width $\tau$ such that $x^{(i)}$ is correctly classified, for all $i = 1, \ldots, m$. [Hint: Let $\alpha_i = 1$ for all $i$ and $b = 0$. Now notice that for $y \in \{-1, +1\}$ the prediction on $x^{(i)}$ will be correct if $|f(x^{(i)}) - y^{(i)}| < 1$, so find a value of $\tau$ that satisfies this inequality for all $i$.]

(b) Suppose we run a SVM with slack variables using the parameter $\tau$ you found in part (a). Will the resulting classifier necessarily obtain zero training error? Why or why not? A short explanation (without proof) will suffice.

(c) Suppose we run the SMO algorithm to train an SVM with slack variables, under the conditions stated above, using the value of $\tau$ you picked in the previous part, and using some arbitrary value of $C$ (which you do not know beforehand). Will this necessarily result in a classifier that achieve zero training error? Why or why not? Again, a short explanation is sufficient.

4. **Naive Bayes and SVMs for Spam Classification**

In this question you'll look into the Naive Bayes and Support Vector Machine algorithms for a spam classification problem. However, instead of implementing the algorithms yourself, you'll use a freely available machine learning library. There are many such libraries available, with different strengths and weaknesses, but for this problem you'll use the WEKA machine learning package, available at `http://www.cs.waikato.ac.nz/ml/weka/`. WEKA implements many standard machine learning algorithms, is written in Java, and has both a GUI and a command line interface. It is not the best library for very large-scale data sets, but it is very nice for playing around with many different algorithms on medium size problems.

You can download and install WEKA by following the instructions given on the website above. To use it from the command line, you first need to install a java runtime environment, then add the `weka.jar` file to your `CLASSPATH` environment variable. Finally, you

I dont like to add jar to classpath :)
java -cp weka.jar weka.classifiers.bayes.NaiveBayesMultinomial -…

can call WEKA using the command:

```
java <classifier> -t <training file> -T <test file>
```

For example, to run the Naive Bayes classifier (using the multinomial event model) on our provided spam data set by running the command:

```
java weka.classifiers.bayes.NaiveBayesMultinomial -t spam_train_1000.arff -T spam_test.arff
```

The spam classification dataset in the `q4/` directory was provided courtesy of Christian Shelton (cshelton@cs.ucr.edu). Each example corresponds to a particular email, and each feature correspondes to a particular word. For privacy reasons we have removed the actual words themselves from the data set, and instead label the features generically as `f1`, `f2`, etc. However, the data set is from a real spam classification task, so the results demonstrate the performance of these algorithms on a real-world problem. The `q4/` directory actually contains several different training files, named `spam_train_50.arff`, `spam_train_100.arff`, etc (the ".arff" format is the default format by WEKA), each containing the corresponding number of training examples. There is also a single test set `spam_test.arff`, which is a hold out set used for evaluating the classifier's performance.

(a) Run the `weka.classifiers.bayes.NaiveBayesMultinomial` classifier on the dataset and report the resulting error rates. Evaluate the performance of the classifier using each of the different training files (but each time using the same test file, `spam_test.arff`). Plot the error rate of the classifier versus the number of training examples.

(b) Repeat the previous part, but using the `weka.classifiers.functions.SMO` classifier, which implements the SMO algorithm to train an SVM. How does the performance of the SVM compare to that of Naive Bayes?

5. **Uniform convergence**

In class we proved that for any finite set of hypotheses $\mathcal{H} = \{h_1, \ldots, h_k\}$, if we pick the hypothesis $\hat{h}$ that minimizes the training error on a set of $m$ examples, then with probability at least $(1-\delta)$,

$$\varepsilon(\hat{h}) \leq \left(\min_i \varepsilon(h_i)\right) + 2\sqrt{\frac{1}{2m}\log\frac{2k}{\delta}},$$

where $\varepsilon(h_i)$ is the generalization error of hypothesis $h_i$. Now consider a special case (often called the *realizable* case) where we know, a priori, that there is some hypothesis in our class $\mathcal{H}$ that achieves zero error on the distribution from which the data is drawn. Then we could obviously just use the above bound with $\min_i \varepsilon(h_i) = 0$; however, we can prove a better bound than this.

(a) Consider a learning algorithm which, after looking at $m$ training examples, chooses some hypothesis $\hat{h} \in \mathcal{H}$ that makes zero mistakes on this training data. (By our assumption, there is at least one such hypothesis, possibly more.) Show that with probability $1-\delta$

$$\varepsilon(\hat{h}) \leq \frac{1}{m}\log\frac{k}{\delta}.$$

Notice that since we do not have a square root here, this bound is much tighter. [Hint: Consider the probability that a hypothesis with generalization error greater than $\gamma$ makes no mistakes on the training data. Instead of the Hoeffding bound, you might also find the following inequality useful: $(1-\gamma)^m \leq e^{-\gamma m}$.]

(b) Rewrite the above bound as a sample complexity bound, i.e., in the form: for fixed $\delta$ and $\gamma$, for $\varepsilon(\hat{h}) \leq \gamma$ to hold with probability at least $(1 - \delta)$, it suffices that $m \geq f(k, \gamma, \delta)$ (i.e., $f(\cdot)$ is some function of $k$, $\gamma$, and $\delta$).