误差边界对1-最近邻分类器 [30分]

一个很好的结果由Cover和Hart（1967）显示，随着训练数据量的无限增长，1-最近邻分类器的误差率最多为Bayes最优误差率的两倍。在这个问题中，你将证明这个结果，用于二分类问题，且输入为实数值。

设$x_1, x_2, ...$为训练样本，$y^j$为相应的二进制类标签，$y^j \in \{0, 1\}$。
你可以将$x$视为固定维度$d$的欧式空间中的点。

1. (5分) 给定这些表达式计算真实的概率$q(x) = p(Y = 1|X = x)$，该数据点$x$属于类1的概率。用$p_0(x)$，$p_1(x)$和$\theta$来表达$q(x)$。（提示：使用贝叶斯法则。）

2. (5分) Bayes最优分类器是一个总是将数据点$x$分配给最可能的类$\arg\max_y P(Y = y|X = x)$的分类器。这意味着Bayes最优分类器在最大化正确分类的概率。给定一些测试数据点$x$，什么才是样本$x$将被误分类的条件？使用Bayes最优分类器，误分类的概率$q(x)$是$q(x)$的函数吗？（提示：开始通过将两个类分开来考虑这个问题。）

3. (5分) 现在1-最近邻分类器将一个测试数据点$x$分配给最近的训练点$x'$的类标签。给定测试数据点$x$和其最近的邻$x'$，$x'$的误差$q(x')$是什么？

4. (5分) 在渐近情况下，每个类的训练样本数量趋于无穷，训练数据填充空间的方式是密集的。则$x'$的最近邻$x'$已$q(x')$收敛到$q(x)$。通过在前一个表达式中进行此替换，给定点$x$的1-最近邻分类器的误差$q(x)$的渐近误差。

5. (5分) 证明在第4部分得到的渐近误差不大于2倍的Bayes最优误差。

6. (5分) 为什么这个渐近误差边界不会在非渐近情况下成立，即训练样本数量有限？你可能需要画一个图来说明你的观点。
2 Learning Theory [5 Points]

If the Hypothesis space \( H \) is finite, you have \( m \) i.i.d. samples, and \( 0 < \epsilon < 1 \) then for any learned hypothesis \( h \) we have:

\[
P(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon) \leq |H|e^{-2N\epsilon^2}
\]

If we fix some desired \( \epsilon, \delta > 0 \) such that:

\[
P(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon) \leq |H|e^{-2N\epsilon^2} \leq \delta
\]

Then with probability at least \( 1 - \delta \) we have:

\[
\text{error}_{true}(h) - \text{error}_{train}(h) \leq \epsilon
\]

But from the first bound:

\[
|H|e^{-2N\epsilon^2} \leq \delta
\]

1. **(2 Points)** Calculate the minimum required number of samples \( N \) in terms of \( \epsilon \) and \( \delta \) such that (1) holds.

2. **(3 Points)** Consider using decision trees with fixed depth \( k \) then the number of decision trees is bounded by:

\[
2 \cdot (2n)^{2k-1}
\]

Calculate the minimum required number of samples \( N \) if you wish to use decision trees of depth \( k = 4 \) and \( \delta = 0.05 \) and \( \epsilon = 0.05 \) you have \( n = 10 \) features.

3 Fitting an SVM classifier by hand [30 Points]

(Source: Murphy text, Exercise 14.1) Consider a dataset with 2 points in 1d: \((x_1 = 0, y_1 = -1)\) 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.) The max margin classifier has the form

\[
\min ||w||^2 \quad \text{s.t.}
\]

\[
y_1(w^T \phi(x_1) + w_0) \geq 1
\]

\[
y_2(w^T \phi(x_2) + w_0) \geq 1
\]

1. **(6 Points)** Write down a vector that is parallel to the optimal vector \( w \). Hint: recall from Figure 14.12 (page 500 in the Murphy text) that \( w \) is perpendicular to the decision boundary between the two points in the 3d feature space.

2. **(6 Points)** What is the value of the margin that is achieved by this \( w \)? Hint: recall that the margin is the distance from each support vector to the decision boundary. Hint 2: think about the geometry of 2 points in space, with a line separating one from the other.

3. **(6 Points)** Solve for \( w \), using the fact the margin is equal to \( 1/||w|| \).

4. **(6 Points)** Solve for \( w_0 \) using your value for \( w \) and Equations 2 to 4. 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 3 and 4 as close to 1 as possible.
5. (6 Points) Write down the form of the discriminant function \( f(x) = w_0 + w^T \phi(x) \) 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 R or Matlab.

4 Programming Question [35 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 kernelized version of the Perceptron algorithm.

4.1 Dataset

The digit images have been taken from the Kaggle competition linked to on the projects page, [http://www.kaggle.com/c/digit-recognizer/data](http://www.kaggle.com/c/digit-recognizer/data). This data was originally from the MNIST database of handwritten digits, but was converted into an easier-to-use file format.

The data has also undergone some preprocessing. It has been filtered to just those datapoints whose labels are 3 or 5, which have then been relabeled to 1 and -1 respectively. Then, 1000-point samples have been created, named validation.csv and test.csv. The first column of these files is the label of each point, followed by the grayscale value of each pixel.

4.2 Perceptron

In the basic Perceptron algorithm, we keep track of a weight vector \( w \), and define our prediction to be \( \hat{y}(t) = \text{sign}(w \cdot x(t)) \). If we predict a point correctly, we make no update and continue running. Any time we make a mistake, our update step is

\[
    w(t+1) \leftarrow w(t) + y(t)x(t),
\]

so at time \( t \), \( w(t) = \sum_{i \in M(t)} y^i x^i \)

where \( M(t) \) is the set of mistakes made up to time \( t \).

4.3 Kernels

To apply the kernel trick, we would like to replace \( x \) and \( w \) with \( \Phi(x) \) and \( \Phi(w) \), where \( \Phi : X \to F \) is a mapping into some high- or infinite-dimensional space. For example, \( \Phi \) could map to the set of all polynomials of degree exactly \( d \). To do this, we try to find a function \( k \) for this particular \( \Phi \) that has the property \( k(u, v) = \Phi(u) \cdot \Phi(v) \) for every \( u \) and \( v \). The trick, however, is that although this function lets us compute dot products easily, we must not actually deal with any \( \Phi(x) \) directly. Because of this, the natural extension of storing our weight vector doesn’t work:

\[
    \Phi(w(t)) = \sum_{i \in M(t)} y^i \Phi(x^i)
\]

would require both computing the sum of \( \Phi(x^i) \) explicitly and storing it as \( \Phi(w) \). As stated above, \( \Phi(w) \) could have millions (or in fact an infinite number) of terms, so this can quickly become impractical. Instead, we can rely on the fact that our prediction becomes

\[
    \hat{y}(t) = \text{sign}(\Phi(w) \cdot \Phi(x)) = \text{sign} \left( \sum_{j \in M(t)} y^j \Phi(x^j) \cdot \Phi(x(t)) \right) = \text{sign} \left( \sum_{j \in M(t)} y^j k(x^j, x(t)) \right)
\]
This means that it’s possible for us to store the \((x^j, y^j)\) pairs of our mistakes \(M^{(t)}\), and use these to compute our dot product \(\Phi(w) \cdot \Phi(x)\).

The drawback of this is that we are now storing a list of all mistakes we ever make, which is quite a bit more overhead than simply \(w\) in the case without kernels. This also means that if we make too many mistakes, performing the dot product can become quite slow. However, we are now able to build much more complex models, and changing between models is as easy as switching kernel functions.

### 4.4 Task

_Hint: It is probably a good idea to write your Perceptron implementation so that it can be passed a kernel function as an argument. If you wish to apply a function to each row of a matrix in R, the built-in "apply" function is much more efficient than using a loop._

1. **(15 Points)** First, get Perceptron working with the kernel \(k^1_p(u, v) = u \cdot v + 1\).
   
   \((k^1_p\) is what the standard dot product would give us, if we had added a constant term \(x_0 = 1\).\)

   Run Perceptron for a single pass over the validation set with this kernel, and plot the average loss \(\bar{L}\) as a function of the number of steps \(T\), where

   \[
   \bar{L}(T) = \frac{1}{T} \sum_{j=1}^{T} \mathbb{1}(\hat{y}^{(t)} \neq y^{(t)})
   \]  
   
   where \(\hat{y}^j\) is the label that Perceptron predicts for datapoint \(t\) as it runs, and \(\mathbb{1}\) is an indicator function, which is 1 if its condition is true and 0 otherwise. Record the average loss every 100 steps, e.g. [100, 200, 300, ...].

2. **(10 Points)** For a positive integer \(d\), the polynomial kernel \(k^d_p(u, v) = (u \cdot v + 1)^d\) maps \(X\) into a feature space of all polynomials of degree up to \(d\).

   For the set \(d = [1, 3, 5, 7, 10, 15, 20]\), run Perceptron for a single pass over the validation set with \(k^d_p\), and plot the average loss over the validation set \(\bar{L}(1000)\) as a function of \(d\). What value of \(d\) produces the lowest loss?

3. **(10 Points)** For \(\sigma > 0\), the Exponential kernel \(k^\sigma_E(u, v) = e^{-\frac{\|u - v\|}{2\sigma^2}}\) is a map to all polynomials of \(x\), where \(\sigma\) is a tuning constant that roughly corresponds to the "window size" of the exponential. Tuning on the validation set has produced a value of \(\sigma = 10\).

   For the \(d\) you chose in the previous step, run Perceptron with both \(k^d_p\) and \(k^{10}_E\) for a single pass over the testing set. For each of these two kernels, plot the average loss \(\bar{L}(T)\) as a function of the number of steps. As above, report the average loss for every 100 steps.