Homework 1 Solutions

1. *The different types of machine learning problems.* Determine whether the tasks described below involve supervised learning or unsupervised learning. For supervised learning problems, identify them as regression, classification, or probabilistic classification.

   (a) Predict the risk of an accident at an intersection, given features such as the time of day and weather.
   (b) Identify cars, bicyclists, and pedestrians in video taken by an autonomous vehicle’s cameras.
   (c) Determine the probability that there is a stop sign in an image.
   (d) Generate new road scenarios (generate streets, place stop signs and intersections) for testing autonomous vehicles in a simulation.

**Solution.**

   (a) Regression
   (b) Classification
   (c) Probabilistic classification
   (d) Unsupervised learning

2. *Train vs test datasets.* Suppose you are building a classifier that identifies cats and dogs. You have a dataset of 3,000 images containing cats, dogs, or other objects (neither cat nor dog). You randomly split the data into a 2,500 image training set and a 500 image test set.

   (a) Why is it important to “reserve” some images for the test dataset? (Why shouldn’t we use all 3,000 images to train the classifier?)
   (b) After training your classifier for a while, you observe it performs well on the training images, but poorly on the test images. What is one possible explanation?
   (c) Suppose we have a classifier with good performance on the training set but high error on the test data. For each of the following methods, briefly explain if they can solve this issue.

      i. Increasing the size of the test set from 500 to 1,000, while the total number of images remains the same.
      ii. Adding 2,000 more images to the dataset, training over a set of 4,000 images and testing with the other 1,000 images.
      iii. Using a more complex feature mapping to generate features from raw data.

**Solution.**
(a) The test dataset should be separate from the training dataset because we want to make sure the classifier performs well on images it has never seen before.

(b) The classifier could be overtraining. Conceptually, it has “memorized” the training dataset instead of learning features that generalize to the test dataset.

(c) • Decreasing the size of the training set cannot improve the performance of the classifier (and will probably make it worse).
• A larger dataset may result in the classifier learning the patterns better, so this can solve the problem.
• A more complex feature map can capture more complicated patterns, so if the issue is the result of the classifier being too simple, then this method can solve it.

3. Some true or false questions. For each of the statements below, state whether it is true or false. If true, give a brief explanation why. If false, give a counterexample.

Note that the nearest neighbor predictors in this problem are meant for regression (not classification).

(a) Adding a single data point to our dataset can change the $k$-NN predictor output for every possible input.

(b) A soft nearest neighbor predictor $g$ with the parameter $\rho \to \infty$ becomes a constant function, i.e., $g(x)$ will be independent of $x$.

(c) Let $g$ be a $k$-NN predictor trained over a data set. If $(x, y)$ is a data point from the training set, then $y = g(x)$.

(d) The computation time needed to predict the output of the $k$-NN predictor for a new sample depends on the size of data set, making this algorithm computationally expensive for large data sets.

Solution.

(a) True. If we have two data points $x_1 = -1$ and $x_2 = 1$ with $y_1 = y_2 = 0$, with $k = 2$, the predicted value is always 0. Now, if we add a third data point $x_3 = 0$ with $y_3 = 1$, then the 2-NN prediction for all points will be 0.5.

(b) True. This is correct since all the weights will converge to $1/n$.

(c) False. The example of part 3a works as a counterexample here.

(d) True. That is correct since finding the $k$ nearest neighbors requires investigating all data points.

4. Fitting a known function using samples. In this problem you will use various nearest neighbor methods to predict $y \in \mathbb{R}$ given $x \in \mathbb{R}$, for a simple case in which we know the exact relation between $x$ and $y$. (This is never the case in practical prediction problems.)

Consider the function $f(x) = \sin(10x)$ over $x \in [0, 1]$. 


(a) Randomly sample 30 points \(x^i\) from \([0, 1]\) using a uniform distribution, and let \(y^i = f(x^i)\). Plot these data points as dots, along with \(f\) as a curve. (To plot \(f\), evaluate it for 500 points uniformly spaced in \([0, 1]\), \(i.e., x = (k - 1)/499, k = 1, \ldots, 500\).)

(b) On eight separate plots, plot the \(k\)-nearest neighbor predictors for \(k = 1, 2, 3\) and the soft nearest neighbor predictors for \(\rho = \sqrt{0.0001}, \sqrt{0.0003}, \sqrt{0.001}, \sqrt{0.003}, \sqrt{0.01}\). Include the 30 data points, shown as dots, in these plots.

(c) RMS error. For each of the eight predictor functions in part (b), evaluate the RMS error on the 500 uniformly spaced points used to plot the functions, given by

\[
\left(\frac{1}{500} \sum_{k=1}^{500} (\hat{y}_k - y_k)^2\right)^{1/2},
\]

with \(y_k = f((k - 1)/499)\) and \(\hat{y}_k = g((k - 1)/499)\), where \(g\) is your predictor.

Julia hints. \texttt{rand(N)} generates \(N\) points from a uniform distribution on \([0, 1]\). To generate a uniformly spaced set of \(N\) values between \(a\) and \(b\) (with \(a < b\)), use \texttt{range(a, stop=b, length=N)}. To apply a function \(f : \mathbb{R} \rightarrow \mathbb{R}\) elementwise to a vector \(x\), use \texttt{f.(x)}.

Solution. The precise plots and values for this problem will vary (slightly) for each student, as each student generated their own samples of \(f(x)\).

An example of a possible solution in Julia is shown below.

```julia
# plotting
import PyCall
import PyPlot; const plt = PyPlot
plt.plt.style.use("seaborn")

using Random

Random.seed!(0)

#=================================================================================================

#normal knn predictor
function knn(X, Y, x, k)
    n = size(X)[1]
    #find distances of examples to x
    dists = [sum((X[i, :] .- x).^2) for i=1:n]
    #find k-nearest neighbors
    nearest_neighbor_idxs = sortperm(dists)[1:k]
    #average k-nearest neighbors
    y_hat = sum(Y[nearest_neighbor_idxs, :], dims=1)/k
    return y_hat

#end
```

3
# soft nn predictor

```plaintext
function softnn(X, Y, x, rho)
    n = size(X)[1]
    exp_dists = [exp(-sum((X[i,] .- x).^2)/rho) for i=1:n]
    w = exp_dists / sum(exp_dists) #find weights
    y_hat = sum(w .* Y, dims=1) #weighted combination of Y
    return y_hat
end
```

```plaintext
f(x) = sin(10*x)
```

# part a

```plaintext
# part a
samples = sort(rand(30))
x = [(k-1)/499 for k=1:500]
plt.close("all")
plt.scatter(samples, f.(samples), color="blue", label="samples")
plt.plot(x, f.(x), color="black", label="f(x)")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.tight_layout()
plt.savefig("q2a.pdf")
```

# part b, c

```plaintext
function RMSE(y, yhat)
    mse = sum((y .- yhat).^2)/length(y)
    rmse = sqrt(mse)
    return rmse
end
```

```plaintext
plt.close("all")
y_samples = f.(samples)
```

# knn

```plaintext
# knn
fig, ax = plt.subplots(3,1, figsize=(10,10), sharex=true)
for (j,k) in enumerate([1, 2, 3])
    Yhat = [float(knn(samples,y_samples,x[i],k))[1] for i=1:length(x)]
    ax[j].plot(x, f.(x), color="black", label="f(x)")
    ax[j].plot(x, Yhat, color="green", label=string("k-NN"))
    ax[j].scatter(samples, f.(samples), color="blue", label="samples")
```
for (j,rho) in enumerate([0.0001, 0.0003, 0.001, 0.003, 0.01])
    Yhat = [float(softnn(samples,y_samples,x[i],rho))[1] for i=1:length(x)]
    ax[j].plot(x, f.(x), color="black", label="f(x)"
    ax[j].scatter(samples, f.(samples), color="blue", label="samples"
    ax[j].plot(x, Yhat, color="green", label=string("soft NN"))
    ax[j].set_title(string("rho=" rho," RMS error=" round(RMSE(f.(x), Yhat), digits=3)))
    ax[j].legend()
    ax[j].set_xlabel("x"
end
plt.tight_layout()
plt.savefig("q2b_softnn.pdf")

(a) The plot is shown in figure 1.
Figure 2 Plots of $k$-nearest neighbor predictors, along with $f(x)$.

Figure 3 Plots of soft nearest neighbor predictors, along with $f(x)$. 
(c) The RMS errors are reported in the title of each subplot in figures 2 and 3. The model that performed the best was the soft nearest neighbor predictor with $\rho = 10^{-3}$, as this model yielded the smallest RMS error. (The best model could be different for each student; we gave full credit if you justified your answer properly, \textit{i.e.}, you chose the model with the smallest RMS error.)