P556 Homework 3

Fall 2021

Due on Nov 3rd, 11:59pm

Question 1: Bias Variance Decomposition (40 points)

Recall that the squared error can be decomposed into bias, variance and noise:

$$\underbrace{\mathbb{E}[(h_D(x)-y)^2]}_{\text{Error}} = \underbrace{\mathbb{E}[(h_D(x)-\bar{h}(x))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[(\bar{h}(x)-\bar{y}(x))^2]}_{\text{Bias}} + \underbrace{\mathbb{E}[(\bar{y}(x)-y(x))^2]}_{\text{Noise}}$$

We will now create a data set for which we can approximately compute this decomposition. The function toydata.py generates a binary data set with class 1 and 2. Both are sampled from Gaussian distributions:

$$p(\vec{x}|y=1) \sim \mathcal{N}(0, I) \text{ and } p(\vec{x}|y=2) \sim \mathcal{N}(\mu_2, I), \tag{1}$$

where $\mu_2 = [2; 2]^\top$ (the global variable OFFSET = 2 regulates these values: $\mu_2 = [\text{OFFSET} ; \text{OFFSET}]^\top$).

You will need to implement four functions: computeybar, computehbar, computevariance and biasvariancedemo.

(a) Noise (computeybar): First we focus on the noise. For this, you need to compute $\bar{y}(\vec{x})$ in computeybar. With the equations, $p(\vec{x}|y=1) \sim \mathcal{N}(0, I)$ and $p(\vec{x}|y=2) \sim \mathcal{N}(\mu_2, I)$, you can compute the probability $p(\vec{x}|y)$. Then use Bayes rule to compute $p(y|\vec{x})$.

Hint: You may want to use the norm probability density function, which you can directly use some package to call this function if you find some. With the help of computeybar you can now compute the "noise" variable within biasvariancedemo. Here is a plot that what your data is supposed to be like in Figure 1:

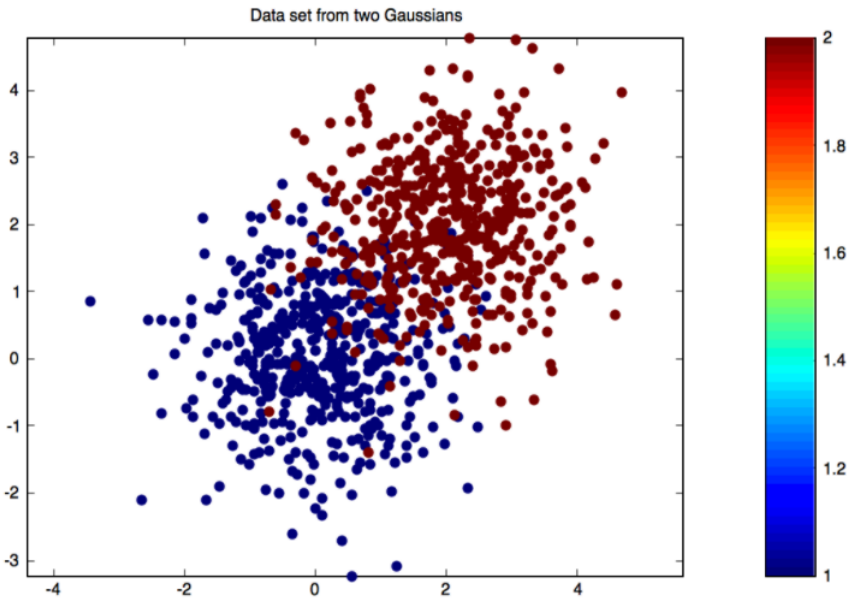

Figure 1: Question 1

(b) Bias (computehbar): For the bias, you will need $\bar{h}$. Although we cannot compute the expected value $\bar{h} = \mathbb{E}[h]$, we can approximate it by training many h_D and averaging their predictions. Edit the function computehbar: average over *n_models* different h_D, each trained on a different data set of n inputs drawn from the same distribution. Feel free to call toydata.py to obtain more data sets.

Hint: You can use ridge regression for h_D. With the help of computehbar you can now compute the "bias" variable within biasvariancedemo.

(c) Variance (computevariance.py): Finally, to compute the variance, we need to compute the term $\mathbb{E}[(h_D - \bar{h})^2]$. Once again, we can approximate this term by averaging over *n_models* models. Edit the file computevariance.

With the help of computevariance you can now compute the "variance" variable within biasvariancedemo.

(d) Demo (biasvariancedemo): In this function, you need to implement a plotting function that how the error decomposes (roughly) into bias, variance and noise when regularization constant λ increases. You can see the trend if you did everything correctly.

Hint: You can set a training dataset with a size of 500, for a really big dataset you can set the size as 100000. You can try average over 25 models. But all these parameters you can change freely.

The bigger number for the number of models and/or the training dataset, the better your approximation will be for $\mathbb{E}[h]$ and $\mathbb{E}[(h_D - \bar{h})^2]$).

If you get everything correct, you should get some plot like this:

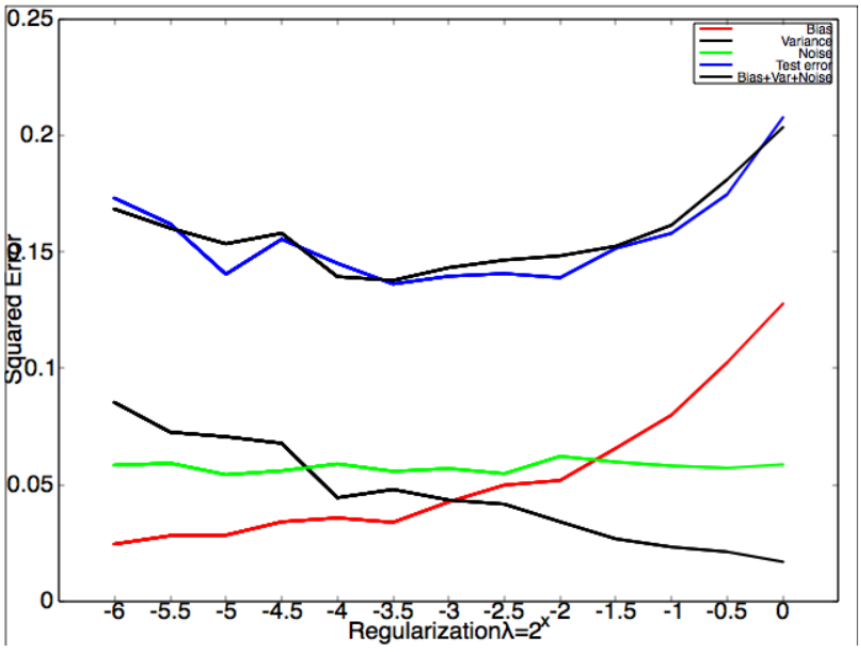

Figure 2: Question 1

Question 2: SVM (30 points)

Here is an visualization of a set of 3600 points in 2D space (*hw3_data2.txt*).

- Which classifier would be able to achieve better performance on this distribution? Justify your choice.
- Implement your chosen classifier and report your accuracy.
- Produce a plot that shows your final classifier as a dotted line, along with the original data points. You can make the plot either in the original space or feature space.

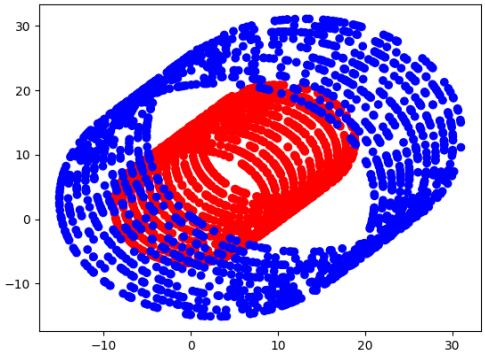

Figure 3: Question 2

Question 3: Gaussian Processes and Hyper Parameter Tuning (40 points)

- Show that the posterior distribution is $P(y^\star|X^\star, X, y) = N_{y^\star}(\mu, \Sigma)$, where $\mu = k(X^\star, X)k(X, X)^{-1}y$ and $\Sigma = k(X^\star, X^\star) - k(X^\star, X)k(X, X)^{-1}k(X, X^\star)$. Note: for this question, you may assume that the conditional distribution is of a Normal form, however you must derive the mean and variance. Note: It is not recommended to calculate the pdf of the posterior, which is a long and painful process. **Hint:** Useful reading material: Chapter 2 of Gaussian Processes for Machine Learning, Rasmussen and Williams (Available online at: http://www.gaussianprocess.org/gpml/chapters/)

- Now for this problem, you will implement a basic Gaussian Progress Regression. We will be using the standard radial basis kernel:

$$K(x_i, x_j) = \sigma exp\left(\frac{-||x_i - x_j||_2^2}{2h^2}\right)$$

where σ, h are known as the scale and bandwith parameters.

So your implementation will be tested on the Concrete Compressive Strength dataset from the UCI repository (which is the data repository I announced before, but we will attach the dataset). The strength of concrete is predicted from 8 features consisting of the ingredients that make up the concrete composition and its age.

Hint:

  - Implement **[K]** = RBF Kernel($X_1, X_2, sigma, h$) which takes as input two matrices of examples X_1 ($n_1\times$ D),X_2 ($n_2\times$ D) with hyperparameters σ, h, and outputs the kernel matrix where $K_{i,j} = k({X_1}^i, {X_2}^j)$, where k is the RBF function described above.
  - Implement **[GPMean, GPVariance]** = GPRegression(XTrain, yTrain, XTest, sigma, h) which carries out the Gaussian Process regression and returns the estimated mean and variances for the variables in XTest. See page 19 of chapter 2 in Rasmussen and Williams for help on making this computationally efficient and numerically stable.
  - In this step, you need to find hyperparameters for the Gaussian Process. One reasonable method for Gaussian processes is to choose parameters that maximizes the log marginal likelihood. First implement **[logml]** = LogMarinalLikelihood(XTrain, yTrain, sigma, h) which computes the log marginal likelihood of the training data given the parameters.
  - Implement **[h, sigma]** = HyperParameters(XTrain, yTrain, hs, sigmas), which does a grid search across the parameters in hs, sigmas and returns the combination that minimizes the log marginal likelihood (here you can just call the grid search function provided by Python).

– Run your Gaussian process regression method on the dataset provided. Compare and report your results with a naive mean prediction. Get your hyperparameters by using your implemented HyperParameters functions and searching over the space of hs = logspace(-1,1,10)'⋆ norm(std(XTrain)) and sigmas = logspace(-1,1,10)'⋆ std(yTrain).