EE 559 **Homework 7** Posted: Fri., 4/5/2024
Jenkins Due: Fri., 4/12/2024, 11:59 PM

1. **Radial basis function network for regression.**

This problem is to be done by computer. You will train a radial basis function (RBF) network on a regression problem using the provided dataset. You may use any sklearn, NumPy, SciPy and matplotlib functions.

The dataset was synthetically generated. It has 2 features, and feature space extends over:

$$-1 \le x_1 \le 1, \ -1 \le x_2 \le 1.$$

Use Gaussian (RBF) basis functions:

$$\underline{\phi}_m(\underline{x}) = \exp\left\{-\gamma \left\|\underline{x} - \underline{\mu}_m\right\|^2\right\}, \quad \gamma > 0, \quad m = 1, 2, \cdots, M$$

and use as our main error measure the root-mean-squared error:

$$\text{RMSE} = (\text{MSE})^{1/2}$$

so that it can be numerically compared with the predicted and given output values.

The dataset is provided as separate train and test sets, having $N_{Tr} = 1000$, $N_{Test} = 1000$, and number of features D=2. They are posted as:

`HW7_Pr1_S24_training_data.csv` and `HW7_Pr1_S24_test_data.csv.`

Tips:

(1) You will use 2 different techniques (described below in (c) and (d)) for choosing the basis function centers $\underline{\mu}_m$.

(2) You might find the function `sklearn.metrics.pairwise.rbf_kernel` useful, because you will need to compute the RBF of all possible pairs between a data point $\underline{x}_i$ and a basis function center $\underline{\mu}_m$.

(3) Implement the first layer as a nonlinear transformation (given above).

(4) You can implement the second layer as a standard linear regression problem, for example by using `sklearn.linear_model.LinearRegression`.

(5) For the L_2-norm calculation, you may find `sklearn.metrics.pairwise.euclidean_distances` useful.

(a) This part can be done by hand. Calculate an approximate default choice γ_d for the RBF width parameter γ, as follows. If we assume the extent of feature space is approximately the same in each dimension, we can find γ_d as:

(i) Let the average spacing, α, between basis-function centers be:

$$\alpha = \frac{\Delta_x}{M^{1/D}},$$

in which

Δ_x= extent (width) of feature space in any one dimension

D = number of features (dimensionality of unaugmented feature space)

M = number of basis function centers

Comment: this approximation assumes the data, and the RBF centers $\underline{\mu}_m$, are approximately uniformly distributed over the feature space.

Give an expression for α in terms of M.

(ii) For γ_d, choose the half-width of the RBF to cover a few average spacings, thus let $\sigma = 2\alpha$. You can relate σ to γ by comparing:

$$\underline{\phi}_m(\underline{x}) = \exp\left\{-\gamma\left\|\underline{x}-\underline{\mu}_m\right\|^2\right\} = \exp\left\{-\frac{\left\|\underline{x}-\underline{\mu}_m\right\|^2}{2\sigma^2}\right\}.$$

Give an expression for γ_d in terms of M, in simplest form.

Tip: Include γ_d in your range of γ during model selection.

(b) For a baseline comparison to the below systems, run linear regression directly on the dataset (no RBF network and no nonlinear transformation). Use 5-fold cross validation using only the provided training set. Give the average RMSE on the validation sets.

In parts (c) and (d) below, train the RBF network; use the stated method for choosing RBF centers (different method for (c) and (d)). For each part, do the following (i)-(v).

(i) Implement the RBF kernel (nonlinear transformation).

(ii) Use MSE linear regression for the second layer, without regularization.

(iii) Use model selection (with cross validation) for finding a good value for hyperparameters (γ in (c); and γ and K in (d)).

Tip: for γ, choose values that range from $0.01\gamma_d$ to $100\gamma_d$, on a log scale (e.g., $\gamma = \gamma'\gamma_d$, with $\gamma' = 0.01, 0.1, 1, 10, 100$). (You can optionally follow up with a smaller range and smaller increment, near the best value from the first set of γ', to get a more precise optimal value.)

(iv) Report your calculated γ_d (for (c)), and report the best parameter values ($\gamma = \gamma^*$ and $K = K^*$ if applicable). Also, for those best parameter values report the cross-validation average RMSE and standard deviation of RMSE (on the validation sets).

(v) Plot average (over the 5 folds) training and average (over the 5 folds) validation RMSE vs. γ. (For part (d), use your best value of $K = K^*$ for the plot.) (1 plot for (c), 1 plot for (d).)

Tip: for cross validation, you may find `sklearn.model_selection.KFold` useful to directly generate pairs of train-validation sets. See document page for more information.

(c) Choose the basis function centers as the data points: $\underline{\mu}_m = \underline{x}_m, \quad m = 1, 2, \cdots, N$, in which N is the number of training data points during each fold in cross validation. In this part, the only hyperparameter to choose during model selection is γ.

(d) Use K-means clustering to choose basis function centers for a given K; vary K using model selection: use values 10-100 with step size 10. For each value of K, choose your initial cluster centers randomly from the data points (i.e., set `init='random'` in sklearn's K-means).

Tips:

(1) You can use the sklearn K-means clustering function `sklearn.cluster.KMeans`, and use `.cluster_centers_` to get the centers.

(2) Inside the K loop, for each value of K you can run K-means a few times and pick the best result based on the current training set RMSE, then calculate its RMSE on the current validation set for your validation result.

(3) You will need to run K-means before applying the RBF nonlinear transformation; why? (no need to turn in your answer)

In this part you have 2 hyperparameters to find during model selection (γ and K).

In addition to (i)-(v) above, for this part also do:

(vi) In the original 2D feature space, plot the training data points $\underline{x}_i$ and the cluster centers $\underline{\mu}_m$ for your best values of hyperparameters.

Tip: use small enough dots for your data-point symbols so that it is easy to visualize the data points.

(vii) Plot the validation RMSE and its standard deviation vs. K, using the best γ for each value of K. (The value of best γ may depend on K.)

(e) Give the d.o.f. and number of constraints N_c for the second layer (linear regressor) for (c) and for (d) (as a function of K for (d)). For N_c, use the number of data points in the training set within each cross-validation fold.

(f) Get test-set results as follows. Train your (best) model from each of (b), (c), and (d), on the entire original training set (i.e., for (c) and (d) use the optimal values of hyperparameters you already found above). (**Tip:** for (d), you can again run K-means a few times and pick the best result based on its training-set RMSE.) Then run this newly trained model on the test set. (Test-set results for 3 models in total.) Note that this model run on the test set, uses all values of hyperparameters and RBF centroids that were found based on the training set.

(g) Compare and comment on your results from (b)-(f). Specifically, observe and try to explain differences in performance for different values of K and γ during model selection. Also compare and comment on 3 the test-set results of (f) in comparison with each other.

Problem set continues on next page…

2. **PCA and MDA for 3-class classification using wine dataset.**

Comment: you will probably find that solving this problem is shorter than it looks.

You will use a wine cultivar dataset, which gives as features results of a chemical analysis of wines derived from 3 different cultivars (varieties) of grape. The goal is to predict the cultivar from the chemical analysis of the wine. It has 3 classes (cultivars) and 13 features. `Wine_data.csv` is posted with this assignment for you to use; the last column is the class label (1, 2, or 3). This dataset was reformatted from the original dataset of [1].

You may use sklearn, NumPy, and matplotlib as you like for this problem.

For your cross-validation runs below, use the entire provided dataset to divide into training and validation sets for each cross-validation loop. We will not need a separate test set in this problem.

For the parts below, you will use the original (unnormalized) dataset for some runs, and use a standardized version of the dataset for other runs. For this problem, you can standardize the whole dataset using parameters calculated from the whole dataset; then store it for future use.

Normally, it's best practice to standardize the dataset within the cross validation loop, so that the dataset can be standardized using parameters calculated from only the training data. In this problem each cross-validation fold has only 5% of data points in the validation set, so they will on average have only a small effect on the normalization; and we are only using the results to compare across different methods, not to get a best estimate of performance on unknowns. So we are taking the simpler approach of standardizing the dataset once at the beginning.

[1] https://archive.ics.uci.edu/ml/datasets/Wine

(a) Baseline for comparison.

First standardize the dataset.

(i) We will pick 2 pairs of features to look at the data as follows: plot the data projected into x_1, x_2 space, and also plot the data instead projected into x_1, x_6 space. For both plots, use 3 different symbols to denote data points belonging to class 1, 2, and 3.

(ii) Run a multiclass perceptron classifier on the 2D data using only features x_1, x_2. For each run, first shuffle the data, then use 20-fold cross validation, and compute the mean classification error rate over the 20 folds. Also store the weight values from the result of the first fold.

Tip1: `sklearn.linear_model.Perceptron`

Tip2: `KFold` and `cross-val_score` from `sklearn.model_selection.`

Do a total of 5 runs. Report the mean classification error rate from each cross-val run, and also report the average and standard deviation of the mean classification error over the 5 runs.

Also report 2 plots: for the run with the lowest classification error rate, plot the results in a scatter plot (labelled data points of the entire dataset, decision boundaries and regions based on the stored weights resulting from the first fold); then produce another similar plot for the run with the highest classification error rate.

(iii) Repeat part (ii) for x_1, x_6.

(b) PCA based on unnormalized dataset.

Tip: `sklearn.decomposition.PCA`

For this part, use the original dataset with no standardization.

(i) Run PCA, reducing to 2 dimensions, on the entire dataset. Plot the data projected into the 2D space, using the same symbols as in part (a) to denote the class label of each data point. Compared with the baseline of (a), do you expect a better classification result with PCA?

(ii) Repeat (a)(ii) except for the 2 new features resulting from PCA (plot everything in the new $(\tilde{x}_1, \tilde{x}_2)$ space).

(iii) How does it compare with the baselines in (a)(ii)?

(c) PCA based on standardized dataset.

Repeat (b)(i)-(ii) except first standardize the data.

(iii) How does PCA with standardized data compare with PCA using unnormalized data? Why?

(d) Multiple Discriminant Analysis (MDA), using LDA as an approximation to MDA.

For this part, first standardize the data.

(i) Because this is a 3-class problem, MDA (rather than Fisher's Linear Discriminant (FLD)) is appropriate to use. We will implement this using instead Linear Discriminant Analysis (LDA), which is similar to MDA.

Tip: `sklearn.discriminant_analysis.LinearDiscriminantAnalysis`.

Run LDA on the entire dataset to reduce the dimensionality to 2. Plot the data projected into the 2D space, using the same symbols as in part (a) to denote the class label of each data point. How do you expect a linear classifier to do on this data compared with (b) and (a) above?

(ii) Repeat (a)(ii) except for the 2 new features resulting from MDA/LDA (plot everything in the new $(\tilde{x}_1, \tilde{x}_2)$ space).

(iii) How does it compare with the baselines in (a)(ii) and PCA in (b)(ii)?

3. **[Extra credit]** In this problem you will use an RBF network for *function approximation*. Most of the code will be the same as for Problem 1 above.

Use datasets: `HW7_Pr3_S24_training_data.csv`, `HW7_Pr3_S24_test_data.csv`.

These datasets have the same $\underline{x}$ values as Pr. 1 datasets; but the *y* values are now a noise-free target function $y(\underline{x})$ of the input data.

Repeat Pr. 1 parts (b), (c), (d), (f) with this new dataset. (Parts (a) and (e) are unchanged; no need to repeat them. Also no need to repeat (g).) Add new part (h) below.

(h) The target function $\mathrm{y}(\underline{\mathrm{x}})$ that was used to generate the *y*-values of the data is:

$$y(x_1, x_2) = 9\left(\cos^2\left(\frac{\pi}{2}x_1\right) + 0.1\right)\sin\left(\frac{4\pi}{x_1^2+1}\right)\sin(\pi x_2)$$

(i) To visuialize the target function, plot $y(x_1, x_2)$ vs. x_1 , for $x_2 = -0.5,\ 0.5$. Also plot $y(x_1, x_2)$ vs. x_2 for $x_1 = -0.5,\ 0,\ 0.5$. (5 plots total).

(ii) To compare the prediction with the target, plot $\hat{f}(x_1, x_2)$ and $y(x_1, x_2)$ vs. x_1 for $x_2 = 0.5$, for the 3 final models you used in (f). For each of these models, plot the model prediction $\hat{f}(x_1, x_2)$ for input points $\underline{x} = (x_1, 0.5)$ over a defined grid of x_1 values, such as $x_1 \in [-1,1]$ with step size 0.01. Make 3 plots total: one for (f)(b) model with $y(x)$ on the same plot for comparison; one for (f)(c) model with $y(x)$ on the same plot for comparison; and for (f)(d) model with $y(x)$ on the same plot for comparison.

(iii) Comment on the results of (ii). Where is the function well approximated? Where is it not? Try to explain any unusual or unexpected behavior. **Hint:** look at your scatter plot of (d)(vi) of this problem.