MLSL2 Hands-on Assignment

*Batch 14*

**Instructions:**

* There are five questions
* Each question carries 20 points.

1. **Fisher Discriminant**

* Dataset: <https://www.kaggle.com/c/digit-recognizer>
* For every pair of classes (0, 1), (0, 9), …, (8, 9)
* Compute the Fisher Discriminant of each pixel (feature)
  + Note that some of the pixels might have a zero denominator
  + Treat that as 0 Fisher
* Normalize the remaining Fisher discriminant values from 0 to 1
* Draw the Fisher images of each of the pairs of classes
* See how they make sense (e.g. Class (0, 1), (3, 5), (4, 6)).

1. **kNN and Parzen Window Classifier**

* Dataset: <https://www.kaggle.com/nishan192/letterrecognition-using-svm>
* Sample 70% training and 30% test data from each class.
* Plot the training and test accuracy of k-NN classifier as a function of
  1. k = 1 to 15
* Plot the training and test accuracy of Parzen window classifier as a function of sigma
  1. sigma = 1 to 10 in increments of 1
* Comment on the “Sweet Spot” where the complexity is “just right”

1. **Support Vector Machines – 1 vs Rest**

* Dataset: <https://www.kaggle.com/nishan192/letterrecognition-using-svm>
* Sample 70% training and 30% test data from each class.
* Write an SVM classifier to build a 1-vs-rest classifier for each class.
* Populate the following table (Test Accuracy)
* Comment on what’s going on as C increases for Linear models
* Comment on what’s going on as Kernel goes from
  + Linear, Polynomial, and RBF for C = 5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Target Class** | **Linear (C = 1)** | **Linear (C = 5)** | **Linear (C = 10)** | **Polynomial**  **(d = 2, C = 5)** | **Polynomial**  **(d = 3, C = 5)** | **RBF**  **(C = 5)** |
| A |  |  |  |  |  |  |
| B |  |  |  |  |  |  |
| … |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Z |  |  |  |  |  |  |

1. **Support Vector Machines – Pair-wise Classifier**

* Dataset: <https://www.kaggle.com/nishan192/letterrecognition-using-svm>
* Sample 70% training and 30% test data from each class.
* Write an SVM classifier to build a pair-wise classifier for each class.
* Populate the following table (Test Accuracy) – Linear (C = 5)
* Comment on which pair of classes are most similar and which are most different.

|  |  |  |
| --- | --- | --- |
| **Positive Class** | **Negative Class** | Linear (C = 5) |
| A | B |  |
| A | C |  |
| … |  |  |
|  |  |  |
| Y | Z |  |

1. **Local Linear Regression**

* **Binary Search**: In Computer science when we have to search for an element in a “sorted” list we use binary search (<https://en.wikipedia.org/wiki/Binary_search_tree>)
* We will use a similar approach to develop a local-linear embedding in 1-D data
  + We will first take the full dataset and learn a “linear regression” model on it
  + Then we will divide the data into two parts – left and right and learn a linear regression on each part and continue till each “leaf node” of the tree gives us an RMSE below a threshold (specified by the user).
  + The trick is in finding the right place to divide the range into two parts.
* Example:
  + Let’s say the x axis is between -10 to 10.
  + Let’s say we have generated 20000 data points in this range at equal intervals
  + We can now try to divide this from [-10, a] and [a, 10]
  + We can run a for loop over all possible a values from say -10 to +10
  + For each value of a, we learn a linear regression
    - for the left part [-10, a] and right part [a, 10]
  + Let n(left|a) and n(right|a) be the number of data points in the two parts
  + Let rmse(left|a) and rmse(right|a) be the rmse of the two parts
  + Then rmse(a) = n(left|a) \* rmse(left|a) + n(right|a) \* rmse(right|a)
  + We find the value of a that gives the best partition of [-10, 10] (min rmse(a)) say A.
  + We store the two linear regression models for [-10, A] and [A, 10]
  + Now we repeat the same exercise for each of the partitions to grow this further
  + If rmse([-10, A]) is < rmse\_threshold then
    - we stop growing otherwise we call LLE(…, -10, A) again with this range.
  + If rmse(A, 10) < rmse\_threshold then
    - We stop growing otherwise we call LLE(… A, 10) again with this range.
* **Dataset**: We will learn the local linear embedding for the following function:
  + <https://en.wikipedia.org/wiki/Ricker_wavelet>
* The code to generate this data is given here:
  + <http://www.southampton.ac.uk/~fangohr/training/python/snippets/lecture09/mexhat-numpy.py>
* Generate this data once and plot it y against x and see if you get the same curve.
* **[9 points]** Write a function to train such a local linear regression model recursively
  + [best\_model, best\_partition] = **LLE**(xy\_dataset, min\_x, max\_x, rmse\_threshold)
  + Store this model so you now where it is in the tree structure
  + rmse\_threshold determines how deep you go – it’s a hyper-parameter
* Note: Ultimately you don’t need all intermediate models, you just need final models:
  + Min\_x and Max\_x value for each model
  + Linear coefficients for each model
* When we get a new data point, we first check which model it should go to based on its x value (and the Min\_x, Max\_x of each model) and then estimate the y value based on the selected linear model.
* **[5 points]** Write a function to predict the y value for a given input using the above structure.
* **[6 points]** Play with it to see how rmse\_threshold affects the “number of linear models” you get.