Saurabh Mylavaram
ID# 5593072
mylav008@umn.edu

Homework 4
CSCI 5525-Machine Learning

2020-12-10

Preprocessing

Missing values

The original 'breast-cancer-wisconsin' dataset has missing values in some columns, which are marked by a '?' symbol. To deal with this, we replaced missing values with the **mode** of the corresponding column(feature).

ID column in Cancer data

Since the IDs of patients seemed irrevelant to our classification task, we excluded the first column of data which has patient IDs for problem 1 and the (i) part of problem 2. However, this column is included in part(ii) of problem 2, as we need 10 columns for setting m=10.

Image data

We normalized the image data in problem 3 from the range $[0, 255]$ to the range $[0, 1]$ to avoid large numbers while calculating distances squares.

1 Problem 1: Adaboost

Summary of Methods

In this problem, we use Ada-boosting with 1-level decision trees for classification. We use the following steps to implement this classifier:

1. Initialize weight for all datapoints to $1/N$ where N is the number of training samples.
2. Then we train a binary decision stump on this data using the information gain as criteria in the following way:
   (a) For each attribute i, we get a list of all unique values $\{v_{i,1}, v_{i,2}, \dots\}$.
   (b) We try each of these values $v_{i,j}$ as a threshold to split the data into 2 groups.
   (c) Calculate the entropy for these splits using the formula
   $$H(Y|x_i) = p(x_i < v_{i,j})H(Y|x_i < v_{i,j}) + p(x_i \geq v_{i,j})H(Y|x_i \geq v_{i,j})$$
   where,
   $$H(Y|x_i < v_{i,j}) = -p(Y = 0)\log p(Y = 0) - p(Y = 1)\log p(Y = 1)$$
   (These probabilities are for data with $x_i < v_{i,j}$)
   (d) Select the splitting attribute i with the highest information gain. Gain can be calculated as
   $$\text{Gain}_i = H(Y) - H(Y|x_i)$$
   For comparing different attributes we can simply use:
   $$\text{Gain}_i = 1 - H(Y|x_i)$$
3. With this decision stump, we calculate the error rate on training data ϵ. This can be used to compute the estimator weight α_t:
   $$\alpha_t = \frac{1}{2}\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$
4. This classifier (decision stump) and corresponding α_t are stored in a list $\mathcal{L}$.

5. α_t is also used to update the weights for the training data according to equation:

$$w_{t+1} = w_t \exp(-\alpha_t \; y \; y_{pred})$$

   Weights are normalized by diving with the sum of all weights:

$$w_{t+1} = \frac{w_{t+1}}{\text{sum}(w_{t+1})}$$

6. For the next iteration, a new dataset is generated by using these new weights w_{t+1} as a probability distribution to sample from the original dataset.

7. Repeat steps 2-6 with the new dataset until the required number of weak learners (100) are reached.

To predict a new data-point using the learned model $\mathcal{L}$, we use each stump G_t in $\mathcal{L}$ to make predictions $G_t(x)$. The final output labels are obtained by a weighted sum of all classifiers' predictions:

$$y_{pred} = \text{sign}\left[\sum_{t=1}^{T} \alpha_t G_t(x)\right]$$

Results

For this problem we used 50% training-testing split. Using **100** weak decision stumps as classifiers, we got a minimum **train error-rate of 0.3%** and **test error-rate of 4.6%**. The plot of 'Error rate' against the 'Number of weak classifiers' used is shown below:

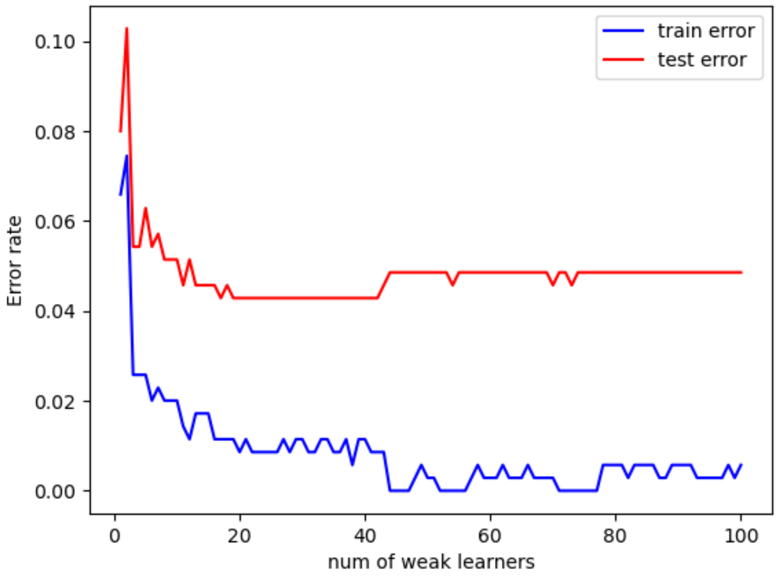


Figure 1: Adaboost: Error rate vs Number of weak learners

2 Problem 2: Random Forest Classification

In this problem we use a Random forest of 1-level decision stumps for classification. We use the following method for classification:

1. First we generate an equally sized random sample D_i from the given dataset D. (There is a chance some points will be repeated and some points will be omitted). We will use this sampled dataset D_i for the rest of the operations.

2. Out of all features in the data, we choose $m = 3$ features, and train a decision stump by selecting a feature with highest information gain from just those m features.

3. Equations for information gain and entropy while selecting the splitting feature, are exactly the same as in the previous problem.
4. We store this decision stump classifier in a list $\mathcal{L}$.
5. Repeat the steps 1-4 for the desired number of trees (stumps) are learned.

To classify a new data-point x_i, we loop through all the decision stumps G_t stored in $\mathcal{L}$, and make predictions: $G_t(x_i)$. The final class label is given by the majority vote among all $G_t(x_i)$.
Since the class labels are +1 and -1, we can just add all the predictions to see which prediction is more frequent:

$$y_{i,pred} = \text{sign}\left[\sum_{t=1}^{T} G_t(x_i)\right]$$

Results

First we run the random forest classifier with $m = 3$ and increasing the number of trees from 1 to 100. Plot of error rate vs the number of trees is shown below:

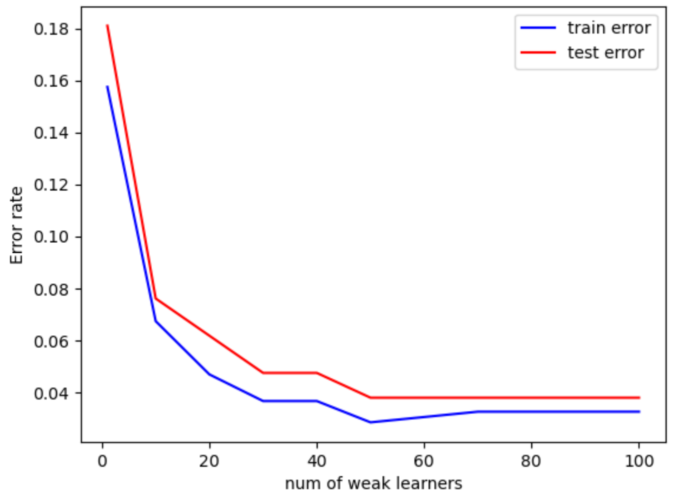


Figure 2: Random Forest: Error rate vs Number of trees

Next, we fix the number of trees as 100 and vary the number of attributes m selected for decision. Plot of error rate vs m is shown below:

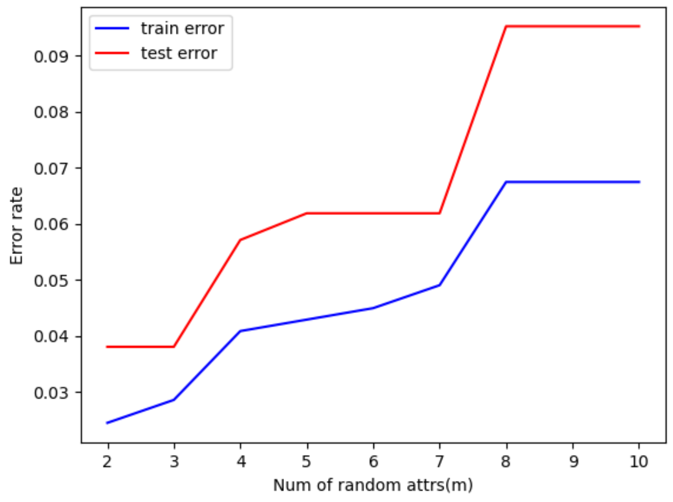


Figure 3: Random Forest: Error rate vs Number of randomly selected attributes

3 Problem 3: Image Segmentation using K-Means

In this problem, we use K-means clustering to reduce the number of colors used in a given image.

Summary of methods

1. Read image in as a $(W \times H \times 3)$ numpy array. Reshape this into a $(W.H \times 3)$ array to get a dataset in the shape of NxD.
2. Initialize K random points $\{C_1, C_2, C_3 \ldots\}$ as cluster centers.
3. **Maximization step:** For each data point (pixel) x_n, calculate the distances from all K cluster centers. Assign the point to the cluster with the smallest distance.

$$r_{n,k} = \begin{cases} 1 & \text{if } k = \text{argmin}_k(||x_n - C_k||^2) \\ 0 & \text{otherwise} \end{cases}$$

4. **Expectation step**: With the new cluster assignments, re-calculate the mean for each cluster.

$$C_k = \frac{\sum_{n=1}^{N} r_{n,k}||x_n - C_k||^2}{\sum_{n=1}^{N} r_{n,k}}$$

   If we find any cluster which does not have any points assigned to it, we re-initialize this cluster mean to a random (3D) point.

5. Calculate the overall distortion measure for all points J. It is the same as sum of squared distances of all points from their cluster centers.

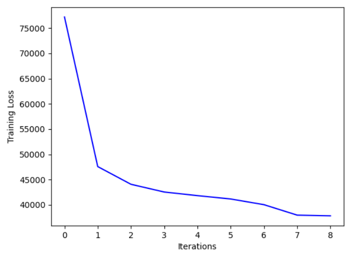
$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{n,k}||x_n - C_k||^2$$

6. We say the convergence criteria is reached when the difference in this loss value between two iterations changes by less than **1%**. Until this convergence criteria is reached, repeat steps 3-4.

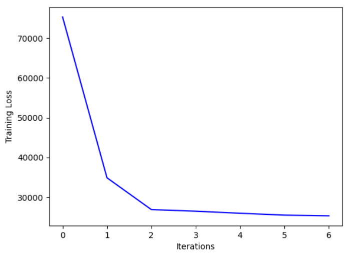
After convergence, we apply the final cluster centers' colors to all the points in that respective cluster. This will give us our final images.

Results

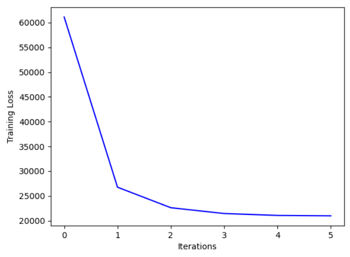
We run our K-means algorithm thrice using K= 3,5 and 7. We plot the training loss vs number of iterations for each run as shown below:



(a) $K = 3$



(b) $K = 5$

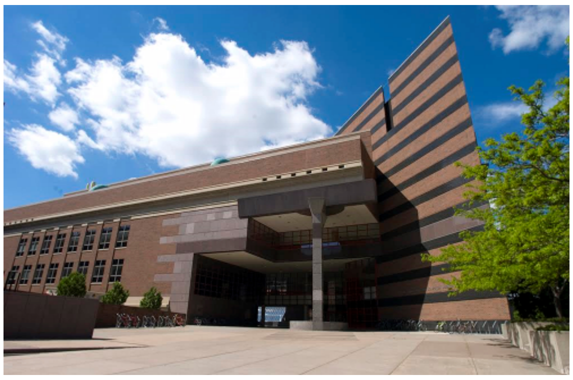


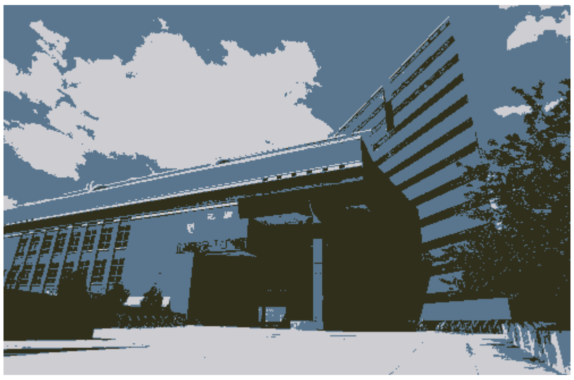
(c) $K = 7$

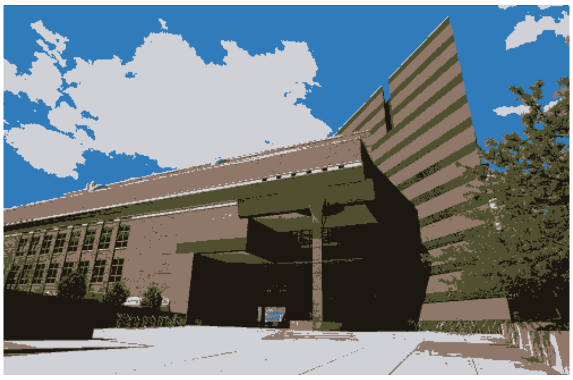
Figure 4: K-Means: Distortion Loss vs Number of Iterations

The resulting images after applying the respective cluster colors are shown in the next page:

(Please see next page)


(a) Original Image


(b) $K = 3$


(c) $K = 5$

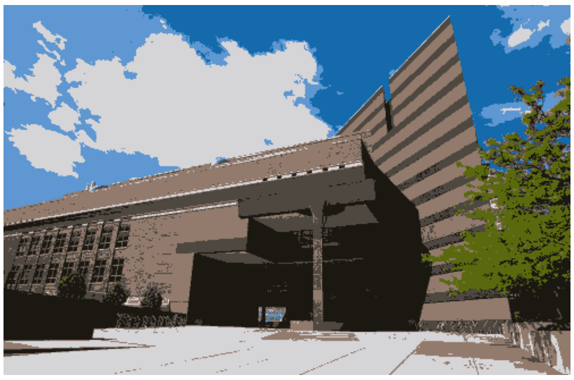

(d) $K = 7$

Figure 5: K-Means: Segmented/Compressed images