Assignment Report 4 - Machine Learning COL774

Piyush Kaul - 2015EEY7544

May 3, 2016

1 K-Means Clustering

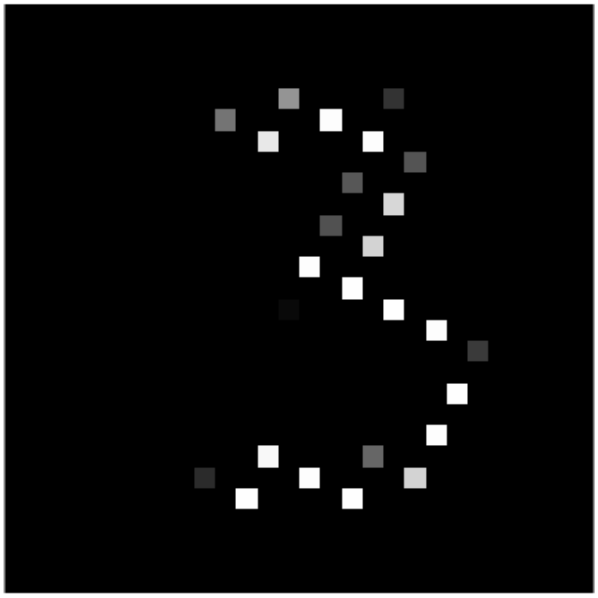
In this problem, you will be working with a subset of the OCR (optical character recognition dataset) from the Kaggle website. This data was originally taken from the MNIST digit recognition database, but was converted into an easier to use fule format. Each of the images in the dataset is represented by a set of 784 (28 by 28) grayscale pixel values varying in the range [0; 255]. The data was further processed 1 so that it could be easily used to experiment with K-means clustering. The processed dataset contains 1000 images for four different (1; 3; 5; 7) handwritten digits. Each image is represented by a sequence of 157 grayscale pixel values (a subset of the original 784 pixel values). A separate file is provided which contains the actual digit value for each of the images. Note that dataset above is similar to the one used in Neural Network learning problem in Assignment 2 but presented differently. Here, we will model the problem of digit recognition as a clustering problem. We will implement the k-means clustering algorithm to categorize each of the images into one of the k = 4 digit clusters given the pixel values. Follow the instructions below. Be aware that some of these operations can take a while (several minutes even on a fast computer)!

1.1 PART A

Q. Write a program to visualize the digits in the data. Your script should take an example index and display the image corresponding to the gray scale pixel values. You can assume a grayscale value of 0 for the pixel values not present in the file.
Ans. The following the command used to print digit number 360

```
kmeans_show_digit(360)
```

Figure 1.1: Digit 360 is 3



1.2 PART B

Q. Implement the k-means (k = 4) clustering algorithm (as discussed in class) until convergence to discover the clusters in the data. Start from an initial random assignment of the cluster means. You can stop at 30 iterations if you find that the algorithm has still not converged. Report your findings..

Ans. Final Results are as follows

Run	Iterations	Detection%
Random Init Run 1	14	73.51%
Random Init Run 2	8	70.10%
Random Init Run 3	13	69.30%
Random Init Run 4	12	74.45%
Random Init Run 5	25	75.50%
Deterministic Init	25	80.12%

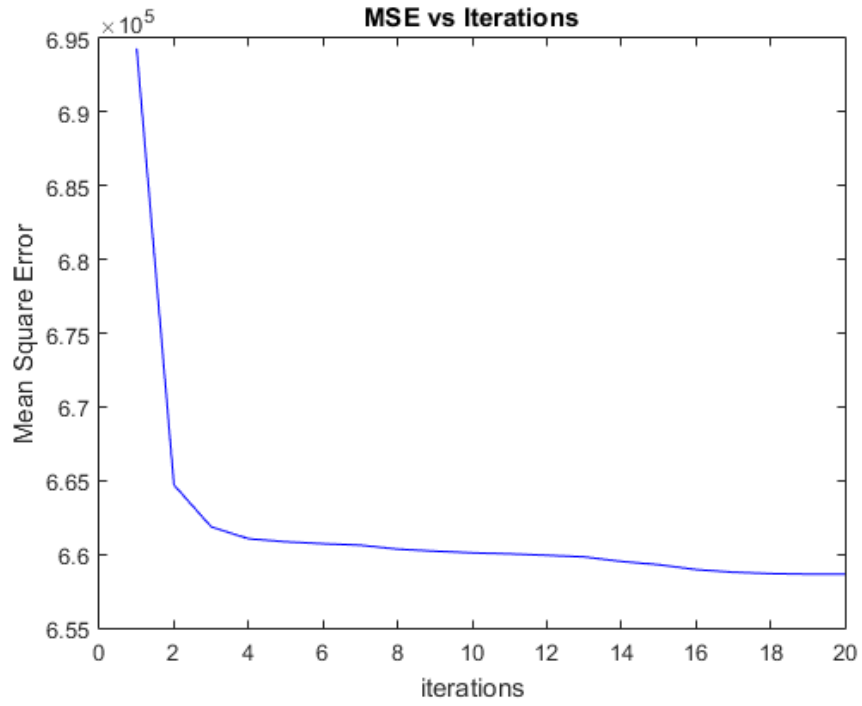
As is visible from the multiple run, there is substantial variance of final correct clustering based on random initialization. Deterministically allocating initial centroids from each of the 4 difference classes performs best.

1.3 PART C

Q. One of the ways to characterize the goodness of the clusters obtained is to calculate the sum of squares of distance of each of the data points x(i) from the mean of the cluster it is assigned to i.e. the quantity $S = \sum_i (x(i) - \mu_{k_i})$ where x(i) denotes the ith data point, ki is the index of the cluster that x(i) belongs to $1 \leq k_i \leq k$, k is the total number of clusters and k_i denotes the mean of the cluster with index k_i. S essentially captures how close the points within each cluster are (or equivalently, how far points across different clusters are). Presumably, smaller the value of S, better the clustering is. Plot the quantity S as we vary the number of iterations from 1 to 20. Comment on your
ndings.

Ans. The final results are as follows

Figure 1.2: Mean Square Error Decrease with Iterations

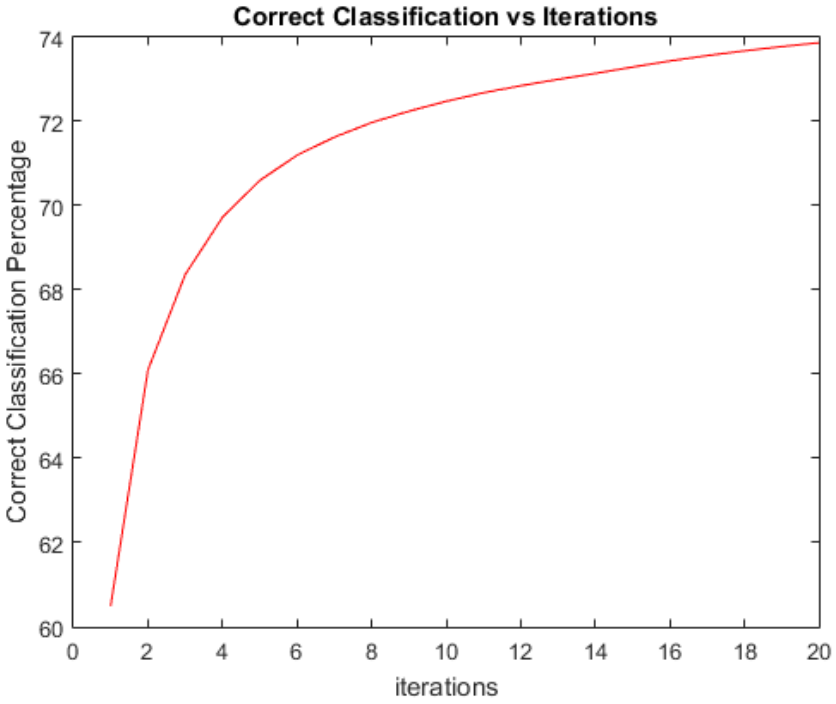


The mean square error decrease with iterations since as the clusters start to correspond to the actual classes, the mean distance between centroids of classes and data points belonging to those classes decrease.
Final Tree

1.4 PART D

Q.Since we have the true labels of the data in this example, we can plot the error of clustering as we run the K-means algorithm. For each cluster obtained at a given step of k-means, assign to the cluster the label which occurs most frequently in the examples in the cluster. The remaining examples are treated as misclassified. Plot the error (ratio of the number of misclassified examples to the total number of examples) as we vary the number of iterations from 1 to 20. Comment on your observations.
Ans. The following results were obtained

Figure 1.3: Correct Classification Percentage with Iterations



The correct classification is successively improved with each iterations as the clusters start getting correctly assigned to data.

2 ANS-2 EXPECTATION MAXIMIZATION

Q. Consider the Bayesian network given in Figure 1. Assume each of the variables in the network is Boolean valued. Recall that the table associated with each variable node in the network represents its conditional distribution given the values of its parent nodes. In this problem, we will implement the EM algorithm over the Bayesian network given above. You are provided with a training set with 10; 000 examples giving values for each of five variables in the network in the order H;B; L;X; F. Each row represents one example. Some of the values in the data could be missing (denoted by ' ?'). We have generated 3 different versions of the data corresponding to different amounts of missing values. File train.data corresponds to the case when there are no missing values. File train-m1.data has exactly one missing value in each example. File train-m2.data has one or two missing values in each example. You are also given a test file test.data with 2000 examples and no missing values.

2.1 PART A

Q. Calculate the ML (Maximum-Likelihood) estimate of the parameters using the fully observed data (train.data). Output the learned parameters in a tabular format for each of the variables in the network. Calculate the log-likelihood of the test data using the ML estimate of the parameters obtained in the previous step

Ans.

Log Likelihood Full obtained is **-2535.6293**

History of Smoking

H=0	H=1
0.804	0.196%

Bronchitis

	B=0	B=1
H=0	0.950	0.0497
H=1	0.584	0.416

Lung Cancer

	L=0	L=1
H=0	0.996	0.042
H=1	0.709	0.290

X-Ray

	X=0	X=1
L=0	0.977	0.022
L=1	0.392	0.607

Fatigue

	F=0	F=1
B=0,L=0	0.951	0.049
B=0,L=1	0.923	0.076
B=1,L=0	0.483	0.516
B=1,L=1	0.300	0.699

2.2 PART B

Q. Implement the EM algorithm over the above network. Initialize the parameters by ignoring missing values (as discussed in class). Carefully define what the E and M steps should be. Note that your program should be able to handle the cases when there is a varying number (one or two) of missing values in each example. Note that different examples might have different variables with missing values. E-step should fill-in the missing values accordingly. Carefully define your convergence criteria.?

Ans. The criteria for convergence is the norm of all probability estimates doesn't change more than $10e-6$

2.3 PART C

Q. Run the EM algorithm implemented above for each of the training data versions with different amounts of missing values. For each case, report the initial set of parameters as well as the parameters obtained after running EM until convergence. In each case, also report the log-likelihood of the test data using the parameters obtained at convergence. Compare these likelihoods with the likelihood obtained in part(a) of the problem (i.e. with no missing data). Did you expect the likelihoods to be very different from each other? Why or why not? Also, comment on your observations.n

Ans.

	Log Likelihood Initial	Log Likelihood Final
Single Value Missing	-2526.2576	-2457.7419
Double Value Missing	-2583.5788	-2478.6975

As visible above. There is not much difference in final Log Likelihood between single missing and double missing case at the end of Expectation Maximization Algorithm run. The reason is that Expectation Maximization uses missing values quite effectively.

3 ANS-3 KAGGLE

3.1 PART A - NON-COMPETITIVE

Q. 1. [15 points] Train and evaluate using the following algorithms.

1. Linear, Gaussian
2. Decision Trees and Random Forests
3. Naive Bayes

You should use only the training data for training, and evaluate on all 3 validation sets. Depending on the model, try to tune the various parameters, rather than just running a default out-of-the-box version of the algorithm. You are free to use standard implementations of learners in Python & MATLAB (you don't need to implement them on your own though you are free to do so). Report various parameter settings that you try for each of the algorithms. Report your training accuracies as well as accuracies on each of the validation sets using the (single) best setting of the parameters for each of the algorithms. If there are different parameter settings giving best results for different validation sets, you can choose one of them based on some criteria (e.g., best average accuracy). Carefully comment on your observations.
Please make sure that (a) the libraries are installed on your laptop, so that we can ask you train them during the demo if needed; (b) you have separate scripts for training the model and evaluating the model on the validation sets that we can run directly from the terminal.
Ans. Python based scikit-learn library was used for all experiments done in this part of Assignment
The following accuracy were obtained.

Performance for different algorithms

	Training	Val 1	Val 2	Val 3
Decision Tree	0.7435	0.7262	0.7141	0.7003
Random Forest	0.9999	0.8051	0.7674	0.7323
SVM RBF	0.8142	0.8054	0.7718	0.731
SVM Linear	0.59646	0.5899	0.5763	0.5291
Naive Bayes	0.7500	0.7445	0.7314	0.7149

For Decision Tree and Random Forests, PCA was used to reduce dimension for training, validation and test vectors.

The parameter settings for different algorithms was as follows.

1. Decision Trees
    - criterion="entropy", splitter='best', max_depth=8, min_samples_split=5, min_samples_leaf=5, min_weight_fraction_leaf=0.0, max_features=None, random_state=0, max_leaf_nodes=None, class_weight=None, presort=False

2. Random Forests
    - n_estimators=1000, criterion='gini', max_depth=None, min_samples_split=5, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, bootstrap=True, oob_score=False, n_jobs=6, random_state=0, verbose=0, warm_start=False, class_weight=None
3. SVM with RBF Kernel
    - C=2.0, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovo', degree=3, gamma=0.001, kernel='rbf', max_iter=-1, probability=False, random_state=0, shrinking=True, tol=0.001, verbose=False
4. SVM with Linear Kernel
    - penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=1.0, multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=1, max_iter=10000
5. Naive Bayes
    - Default

3.2 Part B - Competitive Leaderboard

The following algorithms were tried using scikit.learn library in python.

1. Random Forests of trees upto 10,000 were tried. Beyond 10,000 trees, not much gain was noticed. The Random Forests were evaluated with and without shrinking feature dimensions with PCA. With PCA better performance was observed.
2. Extremely Randomized Trees were used with upto 5,000 trees. The randomization is taken a step further in this technique by even randomizing the splitting feature used. This performed better than Random Forests.
3. Adaboost technique was used which relies on boosting by combining many weak learners to give a good estimate.
4. Bagging classifier was run on top of Extremely Randomized forests by selecting only 50% randomly selected features for any model but it didn't lead to any gain in performance.
5. Perceptron was tried out with default parameters but failed to perform well.
6. VotingClassifier was tried out combining the results of all the above mentioned techniques using majority votes over these. This performed slightly better than individual techniques.
7. SVM was tried with exponential and RBF kernel and the parameters C and Gamma were selected using Grid Search. This performed best and was the final submission for Leaderboard.

Performance for different algorithms

Algorithm	Performance	Private Leaderboard Rank
Decision Tree	70.03*	
Random Forest	75.75	
Extremely Randomized Forest	75.86	
Bagged Trees	70.24	
Naive Bayes	71.49*	
Perceptron	75.86*	
Adaboost	72.60	
Voting Classifier	75.81*	
RBF Support Vector Machine	**78.26**	21

the performance marked * are on a subset of training data.
Final best performing algorithm was RBF support Vector Machine with **78.2**% performance and obtaining **Rank 21** on the Leaderboard