Open Lab 4

Unsupervised Learning

CSCI 4350 - Introduction to Artificial Intelligence

Due: Dec. 4 @ 11:00pm

Overview

Develop a software agent in Python to perform K-means clustering on labeled classification data.

Procedure

1. Create a Python program (`kmeans.py`) which calculates a K-means clustering of a provided set of input training data, assigns classification labels to each cluster using a majority vote, and then reports the classification performance on a separate set of input validation data.
   - The program should take 3 command-line arguments: (**integer**: the number of clusters, **string**: input training data filename, **string**: input validation data filename).
   - The program should read in the **training data** file (one training example per line, see below).
   - The program should read in the **validation data** file (one validation example per line, see below).
   - Each line will contain any number of **real-valued features** and a single **integer value** (the class label) at the end.
   - The program should perform a K-means clustering by first **initializing** K centroid vectors (no class labels included) using the first K examples from the training data (the training data file is assumed to be randomly shuffled so that this is effectively random selection).
   - The program should then determine the **closest** centroid vector to each training example (using Euclidean distance), and create a new set of centroid vectors by **averaging** the feature vectors assigned to each centroid.
   - The program should **repeat** the previous step **until** the centroid vectors no longer change (i.e. until all training examples are assigned to the same centroid on two consequtive iterations).
   - Once the final centroid vectors have been calculated, a class label will be assigned to each cluster (represented by each centroid) by taking a majority vote amongst it's assigned examples from the training set (ties will be broken by preferring the smallest integer class label).
   - Finally, the program will calculate the closest centroid vector to each **validation** example and determine if the cluster label and the testing example label match (i.e. a correct classification).
   - The program should only output the total number of validation examples classified **correctly** by the K-means clustering.
2. Use your program to calculate the **mean** performance and **standard error** of the K-means classifier using leave-10-out repeated random subsampling cross-validation.
   - Use your program to determine the performance of K-means across all numbers of clusters, $K = [1, 2, 3, \cdots, n - 10]$
   - For each value of K, run 100 random shuffles of the data with a training set size of $n - 10$ and a validation set size of 10 where n is the total number of examples in the data set (use `split.bash` and/or `parallelize.bash` from prior labs to help as needed).
   - Make a plot of the mean performance (as a percentage) vs. number of clusters (K) which includes error bars of +/- 1.96 standard errors away from the mean.
3. Write a report (at least 2 pages, single spaced, 12 point font, 1 inch margins, no more than four pages) describing:
   - the K-means method,
   - the code you developled to implement it,
   - the performance of the code under cross-validation (using the statistics above for justification),
   - any limitations of the overall approach,
   - and describe any additional implementation details that improved the performance of your code.

Requirements

- You should utilize the Iris data set to build and validate your K-means agent: iris-data-txt
  - A link to the original data set, with additional information can be found here: Iris@UCI
  - **DO NOT** use the original data set from the UCI link as input; I have re-formatted it to match the specifications above.
- You should also utilize the Breast Cancer data set to analyze the performance of the K-means agent: cancer-data.txt
  - A link to the original data set, with additional information can be found here: BreastTissue@UCI
  - **DO NOT** use the original data set from the UCI link as input; I have re-formatted it to match the specifications above.
- Include a header in the source code with relevant information for assignments (your name, course number/name, etc).
- Your code should **only** print the number of correctly classified testing examples **followed by a newline** character.
- Example Training Data (training.txt):

```
4.9 2.5 4.5 1.7 2
5.6 2.8 4.9 2.0 2
7.7 3.0 6.1 2.3 2
4.6 3.2 1.4 0.2 0
6.0 2.9 4.5 1.5 1
```

- Example Validation Data (validation.txt):

```
6.1 2.8 4.0 1.3 1
5.5 4.2 1.4 0.2 0
6.3 3.3 4.7 1.6 1
```

- Example Run Command: `python kmeans.py 2 training.txt validation.txt`
- Example Output:

```
1
```

- Write your report such that a peer NOT taking this course would understand the problem, your approach to solving it, justification of various choices, and your final comments.
- Include at least one figure to illustrate the K-means method.
- Include plots of **all** of the statistics compiled for your report.
- An example plot for this assignment may be found here: OLA4-example.html
- All sources must be properly cited; failure to do so may result in accusations of plagiarism.
- Your report should be submitted in PDF format.

Submission

- A zipped file (.zip) containing (with **exact** filenames):
  - `kmeans.py`
  - `report.pdf`
- Typical command to zip your lab: `zip OLA4.zip kmeans.py report.pdf`
- Download your zip file and then use your PipelineMT credentials to log in and submit your zip file to the Open_Lab_4 dropbox: https://jupyterhub.cs.mtsu.edu/azuread/services/csci4350-assignments/