Machine Learning Project Report

Rodrigo Castiel
Center for Informatics
Federal University of Pernambuco
Recife, Brazil
rcrs2@cin.ufpe.br

***Abstract*—In the first part of this report, we compare k-means to KCM-F-GH, its hard clustering variation in feature space proposed by Carvalho et al [1]. Both methods are evaluated on the segmentation dataset available in [2] by computing the adjusted rand index to analyze the similarity between the predicted cluster solutions and the ground-truth labels. In the second part, we perform a comparative analysis between three classifiers: a maximum likelihood gaussian estimator, a k-nearest neighbors classifier and a hybrid-model committee classifier. Those classifiers are also tested on the segmentation dataset. We run cross-validation multiple times to estimate the accuracy and the error margin of each classifier. At the end, we perform Friedman test to determine the best algorithm. Our results show that KCM-F-GH outperforms the standard k-means in most cases, with the tradeoff of being considerably more time-consuming. In the second part, our experiments show that both KNN and the hybrid classifier are equally accurate in the dataset, and better than the maximum-likelihood estimator.**

***Index Terms*—clustering, supervised learning, classification, pattern recognition**

I. Introduction

This research report contains two main sections. The first one briefly describes how k-means and KCM-F-GH were implemented, and how they performed in the experiments. In the second one, we point some implementation strategies for the maximum likelihood gaussian estimator (MLE), the k-nearest neighbors classifier (KNN) and the hybrid-model committee classifier. Then, we compare their accuracy in a series of experiments.

The *Python* code, the development history and the dataset are all available on Rodrigo Castiel's personal github (click here). Basic repository structure:

- *part_1.py*: main script for running and comparing the clustering algorithms.
- *part_2.py*: main script for running and comparing the supervised classifiers.
- *classifiers*: contains a list of classes, each implementing a classifier or a clustering algorithm.
- *core*: contains *data_loader.py*, a utility module for managing the segmentation data.
- *data*: training and test datasets.

Additionally, we use *Numpy* and *Scikit*, *Python* libraries for linear algebra, statistics and learning utilities.

II. Part I - Clustering

A. Implementation

The test code start point is located in *part_1.py*. The script arguments, to be passed in from the terminal, are a list of views which will be tested. For example the list *RGB* and *SHAPE* tells the program to run both k-means and KCM-F-GH on the RGB and the shape view, separately. For each view, the script then builds each classifier and calls the method fit on it. The parameter num_times controls how many times KCM-F-GH will be executed before the best fit run is taken. The implementation of k-means and KCM-F-GH are located in *k_means_clustering.py* and *kcm_f_gh_clustering.py*, respectively. They are both thoroughly documented.

Note. During the initial tests of KCM-F-GH, we noticed that the update of the hyper-width parameters is not robust to constant-valued features. That is, if at a given moment a specific feature becomes constant within each cluster, the denominator of equation (24) becomes 0 [1], which breaks the execution. It means that before actually running this algorithm in feature space, we must remove redudant dimensions in the dataset. For this reason, we removed the features "REGION-PIXEL-COUNT" and "SHORT-LINE-DENSITY-2" from the shape view.

B. Experiments and Results

In the first experiments, we noticed that KCM-F-GH takes an average of approximately one hour to converge on the test dataset containing 2100 points. Since our personal computer's hardware is not powerful, executing KCM-F-GH 100 times per view is impossible in practice. Instead, we run it only 50 times on the combined views, 10 times on the separate views.

Figure II-B shows the output log of Part I on the full view (RGB + shape views). In this case, KCM-F-GH achieves $adjusted\_rand\_index = 0.4906$, outperforming k-means with $adjusted\_rand\_index = 0.2826$. However, each run of k-means converges in less than a minute, whereas KCM-F-GH takes nearly one hour to converge.

In Figure II-B, we can see that k-means performs better on the pure RGB view with $adjusted\_rand\_index = 0.3990$, while KCM-F-GH's accuracy drops to 0.4692. Notice that KCM-F-GH converges in less than 15 iterations on the RGB in all runs. On the full view, it may take up to 30 iterations, but in the average it converges in 15 iterations.

The shape view experiments show that KCM-F-GH struggles to converge in most runs. That is, it takes over 30

```
+--------------------+
|      FULL VIEW     |
+--------------------+
KCM_F_GH (c = 7, #points = 2100)
Run 0. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16. Finish.
Run 1. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11. Finish.
Run 2. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
Run 3. Start. Iteration: 0 1 2 3 4 5 6 7 8 9. Finish.
Run 4. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10. Finish.
Run 5. Start. Iteration: 0 1 2 3 4 5 6 7 8. Finish.
Run 6. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23. Finish.
Run 7. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10. Finish.
Run 8. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
Run 9. Start. Iteration: 0 1 2 3 4 5 6 7 8 9. Finish.
Run 10. Start. Iteration: 0 1 2 3 4 5 6. Finish.
Run 11. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11. Finish.
Run 12. Start. Iteration: 0 1 2 3 4 5 6 7. Finish.
Run 13. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16. Finish.
Run 14. Start. Iteration: 0 1 2 3 4 5 6 7 8. Finish.
Run 15. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25.
Run 16. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24. Fi
Run 17. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21. Finish.
Run 18. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10. Finish.
Run 19. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17. Finish.
Run 20. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15. Finish.
Run 21. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11. Finish.
Run 22. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23. Finis
Run 23. Start. Iteration: 0 1 2 3 4 5 6. Finish.
Run 24. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10. Finish.
Run 25. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11. Finish.
Run 26. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25.
Run 27. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Run 28. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12. Finish.
Run 29. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15. Finish.
Run 30. Start. Iteration: 0 1 2 3 4 5 6 7 8. Finish.
Run 31. Start. Iteration: 0 1 2 3 4 5 6 7 8. Finish.
Run 32. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19. Finish.
Run 33. Start. Iteration: 0 1 2 3 4 5 6 7 8. Finish.
Run 34. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14. Finish.
Run 35. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12. Finish.
Run 36. Start. Iteration: 0 1 2 3 4 5 6 7 8. Finish.
Run 37. Start. Iteration: 0 1 2 3 4 5 6 7. Finish.
Run 38. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10. Finish.
Run 39. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12. Finish.
Run 40. Start. Iteration: 0 1 2 3 4 5 6 7 8. Finish.
Run 41. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11. Finish.
Run 42. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13. Finish.
Run 43. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Run 44. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
Run 45. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12. Finish.
Run 46. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18. Finish.
Run 47. Start. Iteration: 0 1 2 3 4 5 6 7 8. Finish.
Run 48. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12. Finish.
Run 49. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14. Finish.

> Adjusted rand score:  0.4906009283628602
> Best fit error:  14787.738476291504

+ K-means (k = 7, #points = 2100)
Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
> Adjusted rand score:  0.2828537895013041
```

Fig. 1. Output log of Part I on full view (RGB + shape). We run KCM-F-GH $num\_times = 50$ with random initial representatives in each cluster. Then, we pick the best fitting KCM-F-GH and compute its adjusted rand index (see bottom). KCM-F-GH achieved $ari_{KCM\text{-}F\text{-}GH} = 0.4906$ while k-means achieved $ari_{k\text{-}means} = 0.2826$.

iterations and does not minize the fitting error properly (the distance in feature-space). In many cases, it gets to a point where the clusters gather points sharing feature with the same value, which causes the undefined behavior mentioned in the note above. We ran KCM-F-GH on the smaller training dataset, but we noticed that it does not converge on the shape view either. We limited it to 30 iterations and we computed its adjusted rand index - it was close to 0, almost like a random classification. Therefore, we do not display its log here (though you can run it on your computer).

III. Part II - Supervised Learning

A. Implementation

The test code start point is located in *part_2.py*. In the first moment, we perform a grid search in the training dataset to find the best hyper-parameters for the classifiers. Then, we run cross-validation multiple times in the training dataset to estimate the accuracy (and its standard deviation) for each classifier. After that, we run the classifiers on the test dataset to evaluate the overall accuracy. Last, we perform the Friedman test to compare all pairs of classifiers. The program writes the output log into *part_2_log.txt*.

Each classifier contains two main methods: *fit* and *predict*.

```
+--------------------+
|      RGB VIEW      |
+--------------------+
KCM_F_GH (c = 7, #points = 2100)
Run 0. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14. Finish.
Run 1. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Run 2. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12. Finish.
Run 3. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12. Finish.
Run 4. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13. Finish.
Run 5. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15. Finis
Run 6. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Run 7. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12. Finish.
Run 8. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10. Finish.
Run 9. Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14. Finish.

> Adjusted rand score:  0.46922750560439264
> Best fit error:  15182.755937129377

+ K-means (k = 7, #points = 2100)
Start. Iteration: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 2
> Adjusted rand score:  0.39899769844274374
```

Fig. 2. Output log of Part I on full view (RGB). We run KCM-F-GH $num\_times = 10$ with random initial representatives in each cluster. Then, we pick the best fitting KCM-F-GH and compute its adjusted rand index (see bottom). KCM-F-GH achieved $ari_{KCM\text{-}F\text{-}GH} = 0.4692$ while k-means achieved $ari_{k\text{-}means} = 0.3990$.

- *gaussian_mle.py*: defines class *GaussianMLE*. In its method *fit*, the training dataset is split up into subsets sharing the same label. Then, the gaussian distribution parameters (*i.e.*, the mean μ and the covariance matrix Σ) are estimated for each class. Once trained, the MLE classifier is also able to compute the a posteriori probabilities for a given point. This is needed for the hybrid classifier.
- *knn_classifier.py*: defines class *KNNClassifier*. It performs no computation in the method *fit*. Instead, it only stores the labeled points. In the method *predict*, we then compute the distances from each input point to all examples and use a heap to keep track of the nearest k points. The predicted label is computed by picking the class of the majority.
- *combinex_max_classifier.py*: defines class *CombinedMaxClassifier*, the hybrid classifier. It takes a list of views to be tested on the dataset, and the hyper-parameter k for its KNN classifiers. Internally, it performs a cartesian product between the list of classifiers (MLE and KNN) and all views. It forwards *fit* and *predict* to all its classifier-view pairs, and returns the combined answer.

B. Experiments and Results

Grid-search finds that the optimal hyper-parameters are $k = 1$ for KNN and $k = 7$ for the hybrid method. Figure III-B was taken from the output log of Part II. As we can see in the cross-validation table, Gaussian MLE has the lowest average accuracy among all classifiers ($\approx 74\%$), while both hybrid and KNN achieve similarly higher accuracies ($\approx 83\%$). On the complete test set containing 2100 points, KNN achieves the highest accuracy, followed by hybrid and then Gaussian MLE. In fact according to the Friedman test results, KNN and hyrid are statistically equivalent.

When it comes to the comparative computational costs, KNN is certainly cheaper than hybrid, since hybrid needs to run multiple classifiers internally before combining their an-

```
------------------- 30x 10-fold Cross-Validation ---------------------
Classifier                                  Accuracy
+ Gaussian MLE ............................ 73.825397% (+/-15.188378%)
+ KNN Classifier (K = 1) .................. 83.730159% (+/-14.284832%)
+ Combined-Max Classifier (K = 7) ......... 82.682540% (+/-14.205028%)
----------------------------------------------------------------------


------------------- Accuracy Evaluation on Test Set --------------------
Classifier                                   Accuracy
+ Gaussian MLE ............................... 74.619048% (1567/2100)
+ KNN Classifier (K = 1) ..................... 87.666667% (1841/2100)
+ Combined-Max Classifier (K = 7) ............. 81.285714% (1707/2100)
----------------------------------------------------------------------


----------------------------- Friedman Test  -------------------------------
Reject H0. The classifiers are not equivalent.
> Gaussian MLE is different from KNN Classifier (K = 1).
> Gaussian MLE is different from Combined-Max Classifier (K = 7).
> KNN Classifier (K = 1) is equivalent to Combined-Max Classifier (K = 7).
----------------------------------------------------------------------------
```

Fig. 3. Output log of Part II. The first section shows the average accuracy and its error margin measured by running 10-fold cross-validation 30 times on the training dataset. The second section shows the overall accuracy measured on the test dataset. The last section shows the Friedman test result (on the training dataset).

swers. Gaussian MLE is indeed the cheapest method, because its prediction consists only of simple likelihood computations.

IV. Conclusion

Part I. KCM-F-GH outperforms the standard k-means, but it is dozens of times slower than it. It may be achieve better results in different datasets. However, on the segmentation dataset it has more drawbacks than advantages. Perhaps, the instrinsic nature of this data does not fit well with gaussian kernels (and therefore, its related algorithms).

Part II. KNN achieved the best tradeoff between accuracy and computational cost. Though Gaussian MLE is linear in complexity, it tends to be less accurate than KNN (on this particular dataset). Furthermore, additional optimizations in KNN, like the usage of spatial datastructures to store points, may improve considerably its performance on large datasets. Another advantage is its hyper-parameter k, which makes it more robust to noise in data.

References

[1] Francisco de A.T. de Carvalho, Eduardo C. Simes, Lucas V.C. Santana, Marcelo R.P. Ferreira, Gaussian kernel c-means hard clustering algorithms with automated computation of the width hyper-parameters, Pattern Recognition, Volume 79, 2018, Pages 370-386, ISSN 0031-3203, https://doi.org/10.1016/j.patcog.2018.02.018.

[2] Vision Group, University of Massachusetts, Image Segmentation Data, http://archive.ics.uci.edu/ml/machine-learning-databases/image/.