Don't Just Divide; Polarize and Conquer!

Shivin Srivastava[1] **Siddharth Bhatia**[1] **Lingxiao Huang**[2] **Jun Heng Lim**[3] **Kenji Kawaguchi**[1] **Vibhav Rajan**[1]

[1]School of Computing, National University of Singapore, Singapore
[2]Huawei TCS Labs, Shanghai, China
[3]Nanyang Technological University, Singapore

Abstract

In data containing heterogeneous subpopulations, classification performance benefits from incorporating the knowledge of cluster structure in the classifier. Previous methods for such combined clustering and classification are either 1) classifier-specific and not generic, or 2) independently perform clustering and classifier training, which may not form clusters that can potentially benefit classifier performance. The question of how to perform clustering to improve the performance of classifiers trained on the clusters has received scant attention in previous literature, despite its importance in several real-world applications. In this paper, we design a simple and efficient classification algorithm called **C**lustering **A**ware **C**lassification (CAC) to find clusters that are well suited for being used as training datasets by classifiers for each underlying subpopulation. We also develop a deep learning method for simultaneous clustering and classification, DEEPCAC. Our experiments on synthetic and real benchmark datasets demonstrate the efficacy of CAC and DEEPCAC over previous methods for combined clustering and classification.

1 INTRODUCTION

Many real datasets have complex underlying structures such as clusters and intrinsic manifolds. For classification tasks on such data, linear classifiers fail to learn well because of the inherent non-linear data distribution. Non-linear classifiers such as (deep) neural networks perform well in such cases, but often require large training data sets to achieve good generalization. When clusters are found or suspected in the data, it is well known that classification performance benefits from incorporating knowledge of such cluster structure in the model, for both linear and non-linear classifiers [1, 2, 3, 4].

As an example, consider the problem of predicting the risk of a disease using patients' clinical data. Patient populations, even for a single disease, show significant clinical heterogeneity. As a result, subpopulations (called subtypes) having relatively homogeneous clinical characteristics can often be found in the data. Models that do not consider underlying subtypes may be consistently underestimating or overestimating the risks in specific subtypes [5], and risk models that account for subtypes have been found to be more effective [6, 7, 4, 8]. Similar examples illustrate the benefit of clustering to build classifiers in, e.g., image processing [9, 10] and natural language processing [11].

Previous works on using clustering for classification broadly fall into two categories. The first group of techniques modify specific classification techniques to account for the underlying clusters, e.g., Clustered SVMs [2]. These techniques are closely tied to the modified classifier and inherit both its modeling strengths and weaknesses. The second and more common approach is to first cluster the data and then independently train classifiers on each cluster. Although simple and intuitive, such a 'cluster-then-predict' approach may not form clusters in a manner that benefits the performance of classifiers trained on those clusters. This is also a limitation of techniques in the second group where clustering and classifier training occur independently. To the best of our knowledge, the question of performing clustering to improve the performance of classifiers trained on the clusters has not been addressed by previous literature.

Thus, in this paper, we first investigate the fundamental question of when and how clustering can help in obtaining accurate classifiers. Our analysis yields novel insights into the benefits of using simple classifiers trained on clusters compared to simple or complex (in terms of Rademacher complexity [12]) classifiers trained without clustering. It provides clues on when such a clustering-based approach may not improve the subsequent classification. Finally, it also motivates the design of our clustering-based framework for

Accepted for the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022).

classification, called Clustering Aware Classification (CAC).

The CAC framework can be viewed as a classification algorithm that, by design, finds clusters that are intended to be used as training datasets by the (base) classifiers of choice e.g., Logistic Regression (LR), SVM or Neural Networks. The key idea of our approach is to induce class separability (polarisation) within each cluster in addition to dividing the data for better classification. This is effected by designing a cost function that can be used within the optimization framework of k-means. However, finding clusters with this cost function using the popular Lloyd's algorithm may not guarantee convergence. Therefore, we design an approach based on Hartigan's method [13] and prove that it converges. Our design is generic and not classifier-specific, and leads to an efficient iterative method to find the clusters. We also theoretically investigate the benefits offered by class separation to classifiers employing the standard log-loss function. We prove that the upper and lower bounds for the standard training log-loss of a dataset are linearly dependent on class separation within the dataset. We analyze the sensitivity of CAC to various data characteristics such as class separation, intercluster distance, and the number of clusters using synthetic data. Motivated by the idea of exploiting class separation in data, we formulate DEEPCAC, a deep variant of CAC that not only encourages clusters with high class separation but also induces class separability in clusters by training a neural network to find such representations. Finally, we empirically study the performance of CAC on benchmark datasets, including a large clinical dataset. In summary, our contributions are:

1. Our theoretical analysis provides insights into the fundamental question of how clustering can aid in potentially improving the performance of classifiers trained on the clusters.
2. We design a novel clustering algorithm, Clustering Aware Classification (CAC) to find clusters in data with the aim of improving the performance of classifiers trained on the clusters. CAC is simple, efficient, and provably convergent. It can be used with any classifier.
3. We develop a model for simultaneous clustering and classification through a Deep Learning based variant of CAC, DEEPCAC that supports multi class data also.
4. Our experimental results on benchmark datasets including a large clinical dataset, show that CAC and DEEPCAC yield higher improvement in classification performance compared to previous approaches for combined clustering and classification.

REPRODUCIBILITY: Our code and datasets are publicly available at this anonymous link.

2 RELATED WORK

The idea of training multiple local classifiers on portions of the dataset has been extensively studied for specific classifiers [14, 15, 16, 17, 18, 19, 20, 21]. For instance, [22] presents a novel locality-sensitive support vector machine (LSSVM) for the image retrieval problem. Locally Linear SVM [23] has a smooth decision boundary and bounded curvature. Its authors show how functions defining the classifier can be approximated using any local coding scheme. [24] proposes Mixed SVMs which partition the feature space into subregions of linearly separable data points and learn a Linear SVM for each of these regions. [2] proposes Clustered Support Vector Machines (CSVMs) which tackles the problem in a divide-and-conquer manner, thus avoiding the need for computationally expensive kernel SVMs. CSVM groups the data into several clusters, after which it trains a linear SVM in each cluster to separate the data locally. Additionally, it imposes a global regularization, that aligns the weight vectors of each locally linear SVM with a global weight vector.

There are other related works, which aim to find suitable representations of raw data such that the performance in downstream prediction tasks is better when they use the learned embeddings. Large Margin Nearest Neighbour (LMNN) [25] introduces the much-celebrated triplet loss. In this work, the authors find a data-specific Mahalanobis distance embedding for the data, subject to the constraint that k similarly labeled points are mapped in the neighborhood of every point. The transformed data is more amenable for a k Nearest Neighbour classifier to work with. [26] extends LMNN to make it parameter-free. Recently, there has been a surge of interest in contrastive learning [27, 28] as well where the core idea is to group similar/dissimilar representations of data into similar/dissimilar groups. The latent representations thus generated are beneficial to supervised classifiers in classification tasks.

Some Bayesian approaches have been developed for combined classification and clustering, e.g., [1, 29] but they are computationally expensive. To the best of our knowledge, closest and most recent work in spirit and purpose to our work is Deep Mixture Neural Network (DMNN) [4]. DMNN consists of an embedding network with gating and a user-defined number of local predictive networks. It proposes simultaneous patient subtyping and risk prediction. The gating network computes high-level feature representations of raw input features, which are then clustered to find patient cohorts. Local predictor networks are then trained to predict the outcomes for their respective cohorts. The subgroup-specific sets of features learned by the local features provide valuable information from which we can reason about the cluster phenotype. Thus, DMNN represents 'cluster-then-predict' type of approach as it uses local predictor networks to classify subpopulations in the

data.

3 PROBLEM STATEMENT

Let $X = (x_1, x_2, \cdots, x_N)$ be a training dataset with n records $x_i \in \mathbb{R}^d$ together with binary labels $y_i \in \{0, 1\}$. Given an integer $k \geq 1$, our goal is to:

1. Divide X into k non-empty, distinct clusters $\mathcal{C} = \{C_1, C_2, \cdots, C_k\}$, and
2. Obtain a collection of k classifiers $\mathcal{F} = \{f_1, f_2, \cdots, f_k\}$ where each $f_j : C_j \rightarrow \{0, 1\}^{|C_j|}$ is trained on all $x_i \in C_j$ with the objective $\text{argmin}_{\mathcal{F},\mathcal{C}} \sum_{j=1}^{k} \sum_{i \in C_j} \ell(f_j(x_i), y_i)$.

There is no restriction on the choice of the classifier, which is determined apriori by the user.

4 CAN CLUSTERING AID CLASSIFICATION?

We provide a principled motivation behind our clustering approach by analysing when clustering can potentially help in obtaining accurate classifiers. Our analysis yields crucial insights into developing the CAC framework. Fix $j \in \{1, \ldots, k\}$ and let $S_j = \{(x_i, y_i)\}_{i=1}^{m_j}$ be the training dataset from the j-th cluster (having m_j points) with $x \in \mathcal{X}_j$ and $y \in \mathcal{Y}_j$ s.t. $\mathcal{X}_j = \{x : x \in C_j\}$ and $\mathcal{Y}_j = \{y : y \in C_j\}$. Then, the following theorem — a direct application of a previous result [30, 12] to our problem — suggests a potential benefit of clustering in terms of the expected error $\mathbb{E}_{x,y}[\ell(f(x), y)]$ for unseen data:

Theorem 1. *(Proof in Appendix A.1). Let $\mathcal{F}_j$ be a set of maps $x \mapsto f(x)$, and the function $q \mapsto \ell(q, y)$ be a λ-uniformly bounded function for any $q \in \{f(x) : f \in \mathcal{F}_j, x \in \mathcal{X}_j\}$ and $y \in \mathcal{Y}_j$. Then, for any $\delta > 0$, with probability at least $1 - \delta$ over an i.i.d. draw of m i.i.d. samples $((x_i, y_i))_{i=1}^{m}$, the following holds: for all maps $f \in \mathcal{F}_j$,*

$$\mathbb{E}_{x,y}[\ell(f(x), y)] \leq \frac{1}{m_j} \sum_{i=1}^{m_j} \ell(f(x_i), y_i) + 2\mathcal{R}_{m_j}(\ell \circ \mathcal{F}_j) + \lambda \sqrt{\frac{\ln(1/\delta)}{2m_j}}, \qquad (1)$$

where

$$\mathcal{R}_{m_j}(\ell \circ \mathcal{F}_j) := \mathbb{E}_{S,\sigma}[\sup_{f \in \mathcal{F}_j} \frac{1}{m_j} \sum_{i=1}^{m_j} \sigma_i \ell(f(x_i), y_i)]$$

and where $\sigma_1, \ldots, \sigma_{m_j}$ are independent uniform random variables taking values in $\{-1, 1\}$.

Theorem 1 shows that the expected error $\mathbb{E}_{x,y}[\ell(f(x), y)]$ for unseen data is bounded by three terms: the training error $\frac{1}{m_j} \sum_{i=1}^{m_j} \ell(f(x_i), y_i)$, the Rademacher complexity $\mathcal{R}_{m_j}(\ell \circ \mathcal{F}_j)$ of the set of classifiers $\mathcal{F}_j$ on the j-th cluster, and the last term $O(\ln(1/\delta)/2m_j)$ in equation (1). To apply Theorem 1 to models without clustering, we can set the j-th cluster to contain all the data points with $m_j = N$.

The upper bound can be used to analyze the benefit of using a simple model (e.g., a linear model) *with* clustering compared to two cases of (1) a simple model and (2) a complex model (e.g., a deep neural network), both *without* clustering. For a simple model *without* clustering, the training error term $\frac{1}{m_j} \sum_{i=1}^{m_j} \ell(f(x_i), y_i)$ in equation (1) can be large, because a simple model may not be able to sufficiently separate training data points (underfitting) to minimize the training error. In the case of a complex model *without* clustering, the training error can be small but the Rademacher complexity $\mathcal{R}_{m_j}(\ell \circ \mathcal{F}_j)$ tends to be large. A simple model *with* clustering can potentially trade-off these by minimizing the Rademacher complexity $\mathcal{R}_{m_j}(\ell \circ \mathcal{F}_j)$ while making the training data in the cluster to be linearly separable to minimize the training error $\frac{1}{m_j} \sum_{i=1}^{m_j} \ell(f(x_i), y_i)$. This observation leads us to explore ways to induce class separability in data clusters.

Theorem 1 also shows that such a clustering-based approach may also not help in some cases in improving classification. Using the clustering approach decreases the value of m and thus increases the last term $O\left(\ln(1/\delta)/2m\right)$ in Equation (1), when compared with models without clustering. Therefore, using a simple model with clustering can be potentially advantageous when the first and second terms in Equation (1) are dominant, but may not be beneficial if the last term is dominant.

Since large m reduces both the second and third terms, unsurprisingly, larger sample sizes may lead to improved performance. The theorem shows that simple models within each cluster may lead to good generalization due to smaller values in the first two terms. It thus motivates the formation of clusters, each of which can be linearly separable internally with respect to the classes. We observe the implications of Theorem 1 in Section 7 in our empirical results.

5 CAC

In this section, we develop our framework Clustering Aware Classification (CAC). It can be viewed as a classification algorithm that by design finds clusters that are intended to be used as training datasets by classifiers. The key idea of our approach is to induce class separability within each cluster. If the points of different classes are well separated within a cluster, then it naturally aids any classifier in determining the class boundaries. We now describe the CAC algorithm.

5.1 CLASS SEPARABILITY

We use a simple heuristic to measure class separability within each cluster. For each cluster C_j ($j \in [k]$), we define: *cluster centroid*: $\mu(C_j) := |C_j|^{-1} \sum_{x \in C_j} x_i$, *positive centroid*: $\mu^+(C_j) := (\sum_{x_i \in C_j} y_i)^{-1} \sum_{x_i \in C_j} y_i x_i$, and *negative centroid*: $\mu^-(C_j) := (\sum_{x_i \in C_j} (1 - y_i))^{-1} \sum_{x_i \in C_j} (1 - y_i) x_i$.

Class separability is defined by considering the distance between positive and negative centroids within each cluster. Figure 1 shows two clusters C_1 and C_2. C_1 has greater class separation while in C_2, the data points are more intermingled and thus their centroids are closer. Thus, $\|\mu_1^+ - \mu_1^-\| > \|\mu_2^+ - \mu_2^-\|$.

5.2 CLUSTERS WITH CLASS SEPARABILITY

To induce our notion of class separability within clustering we consider the k-means clustering formulation that is widely used for its simplicity, effectiveness, and speed. Clusters using k-means are obtained by minimizing the size-weighted sample variance of each cluster: $\mathcal{D}(X) = \sum_{p=1}^{k} \inf_{\mu(C_p) \in X} \sum_{i \in C_p} \|x_i - \mu(C_p)\|^2$.

Thus, the cost of each cluster, given by $\sum_{i \in C_j} \|x_i - \mu(C_j)\|^2$, is minimized. To induce class separability, a natural formulation is to add a cost with respect to class separability for each cluster. With our measure for class separability, and $\alpha > 0$ as a class separation hyperparameter, this becomes:

$$\phi(C_j) = \sum_{i \in C_j} \left(\|x_i - \mu(C_j)\|^2 - \alpha \cdot \|\mu^+(C_j) - \mu^-(C_j)\|^2 \right) \tag{2}$$

Defining the cost function in this manner encourages the decrease of cluster variance term and increase of the class separability term, weighted by α. Consequently, the overall cost function is defined to be $\phi(\{C_j\}_1^k) := \sum_{j=1}^{k} \phi(C_j)$ as the *unsupervised* loss of CAC that we aim to minimize.

5.3 FINDING THE CLUSTERS

The most common heuristic to find k-means clusters is Lloyd's Algorithm [31]. It begins with an initial clustering and iteratively computes cluster centroids and assigns points to clusters corresponding to their closest centroids. However, using this heuristic with our proposed cost may not guarantee convergence. (See our discussion after Theorem 2). Instead, we adopt an approach based on Hartigan's method [13]. For k-means clustering, Hartigan's method proceeds point by point in a greedy manner. Each point is reassigned to a cluster such that the overall cost is reduced.

In our setting, denote the cost of merging a point x to some cluster C_q as $\Gamma^+(C_q, x) = \phi(C_q \cup \{x\}) - \phi(C_q)$. Symmetrically, denote the cost of removing a point x from some cluster C_p as $\Gamma^-(C_p, x) = \phi(C_p \setminus \{x\}) - \phi(C_p)$. Consequently, define the improvement in cost by moving a point $x \in C_p$ from cluster C_p to cluster C_q to be $\Phi(x; C_p, C_q) := \Gamma^+(C_q, x) + \Gamma^-(C_p, x)$. The exact expressions for $\Gamma^+(C_q, x)$ and $\Gamma^-(C_p, x)$ w.r.t $x, \mu, \mu^{\pm}$ are derived in Appendix A.3.

Our approach is summarized in Algorithm 1. It begins with an initial clustering (k-means found clusters) and then proceeds iteratively in rounds. In each round, all points are checked to determine if a point has to be re-assigned to another cluster. For a point in the current iteration, x_i, if moving $x_i \in C_p$ would result in C_p / x_i having points of only one class, then it is not moved. If not, the cluster C_q that results in the maximum decrease in cost $\Phi(x_i; C_p, C_q)$ is found and x_i is moved from its current cluster C_p to C_q.

In each round, all three centroids for each cluster are updated and once the loss function has become stagnant, the algorithm is considered to have converged. Base classifiers are then trained on the clusters found and are used for predicting labels for the unseen test data.

Algorithm 1: CAC

Input: Training Data: $X \in \mathbb{R}^{n \times d}$, binary labels $y^{n \times 1} \in \{0, 1\}^n$, $\{C_j\}_1^k$, $\{f_j\}_1^k$ and α

▷ **Initialization**

Compute $\mu(C_j), \mu^+(C_j), \mu^-(C_j) \; \forall \; C_j \; s.t. \; j \in \{1, \cdots, k\}$.

▷ **Algorithm**

while *not converged* **do**

  for $i \in \{1 \dots n\}$ **do**

    if *Removing* $x_i \in C_p$ *from its cluster does not lead to a* 1*-class cluster* **then**

      ▷ **Assign new cluster to** x_i

      Let $q := \text{argmin}_j \, \Phi(x_i; C_p, C_j)$ (breaking ties arbitrarily).

      if $\Phi(x_i, C_p, C_q) < 0$ **then**

        Assign x_i to C_q from C_p and update $\mu(C_p)$, $\mu(C_q)$, $\mu^{\pm}(C_p)$ and $\mu^{\pm}(C_q)$.

      Otherwise, let x_i remain in C_p.

▷ **Train Classifiers**

For every cluster C_j, train a classifier f_j on $(\mathcal{X}_j, \mathcal{Y}_j)$.

Output Classifiers $\mathcal{F} = \{f_j\}_{j=1}^{j=k}$ and cluster centroids $\mu = \{\mu_j\}_{j=1}^{j=k}$

We now show that CAC is guaranteed to converge and prove that its per-round time complexity is linear in the number of data points, dimensions, and number of clusters.

Theorem 2. *(Proof in Appendix A.2) Algorithm 1 converges to a local minimum.*

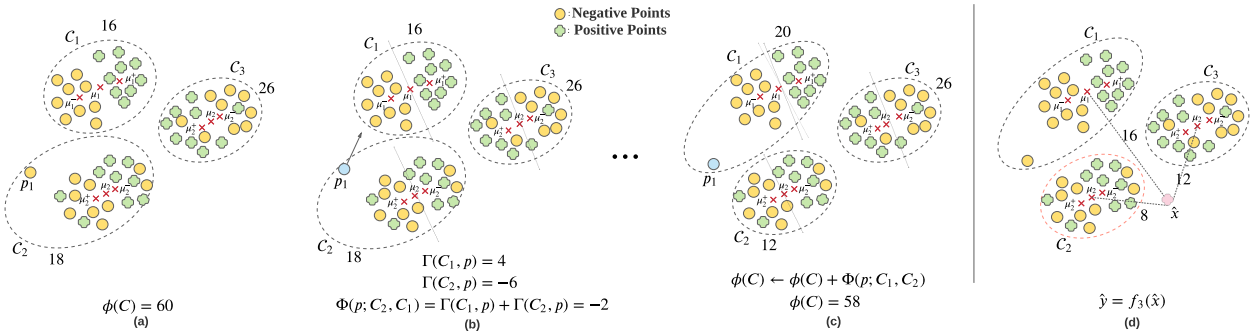


Figure 1: CAC problem setting (with dummy values for illustrative purposes): (a) The total CAC cost of this clustering is 60; (b) Point p_1 is selected to be reassigned to cluster C_1 based on the cluster update equations; (c) p_1 is assigned to C_1 and the cost functions of C_1 and C_2 are updated; (d) At testing time, $\hat{x}$ is assigned to cluster C_2 as it lies nearest to $\hat{x}$.

Theorem 3. *(Proof in Appendix A.3) The time complexity for each round of CAC is* $O(ndk)$.

Remark (Hartigan vs. Lloyd). *For* k*-means clustering, both Lloyd's method and Hartigan's method converge since the cost function always decreases monotonically with each update. Note that Lloyd's method does not consider the change in centroids while updating the assignment of a point, and instead, directly assign points to their current closest centroids. In our setting, Lloyd's method can similarly fix all centroids, compute a new assignment, and update the centroids accordingly. For instance, replacing the assignment rule of* x_i *in Algorithm 1 by* $q := \operatorname{argmin}_j \|x_i - \mu(C_j)\|^2 - \alpha \cdot \|\mu^+(C_j) - \mu^-(C_j)\|^2$*. However, our cost function contains an additional squared distance term between positive and negative centroids, i.e.,* $-\alpha \cdot \|\mu^+(C_j) - \mu^-(C_j)\|^2$*. There is no guarantee that this term monotonically decreases by Lloyd's method during the update of* $\mu^+(C_j)$ *and* $\mu^-(C_j)$ *after an assignment. Hence, it is non-trivial to ensure and prove the convergence of Lloyd's method with our cost function. However, with the use of Hartigan's method, convergence is ensured by design. As for the rate of convergence, empirical experiments demonstrate that the rate of convergence is positively related to* α *(See Figure 2b).*

CAC prediction CAC outputs a set of cluster representatives denoted by their centroids and the trained classifiers associated with those clusters. To make predictions for a test point $\hat{x}$, the first step is to find the cluster to which the test point is most likely to belong. This can be done by assigning the test point to a cluster represented by the 'closest' centroid C_j in the data space i.e. $j = \operatorname{argmin}_l \|\hat{x} - \mu_l\|^2$. The corresponding classifier f_j can then be used to predict $\hat{y} = f_j(\hat{x})$.

Bounding Log-Loss of Classifiers in terms of Class Separation We now prove how class separability within clusters has a direct effect on the training error of classifiers employing log-loss. The following theorem also validates our choice of using $\|\mu^+ - \mu^-\|$ as a heuristic to measure class separability.

Lemma 1. *(Proof in Appendix A.4) For* d *dimentional vectors* $\mathbf{u}, \mathbf{v}$ *and* $\boldsymbol{\beta}$*,* $\frac{\partial \Delta}{\partial \|\mathbf{u}\|} < 0$ *for* $\Delta = A - (B\boldsymbol{\beta}\mathbf{u} + C\boldsymbol{\beta}\mathbf{v})$ *if* $\beta_i u_i > 0, \forall i \in \{1, 2, \cdots, d\}$ *and* $A, B > 0$.

Theorem 4. *(Proof in Appendix A.5) Let* $X = (x_1, x_2, \cdots, x_N)$ *be a training dataset with* n *records* $x_i \in \mathbb{R}^d$ *together with binary labels* $y_i \in \{0, 1\}$*. Define the log-loss over the entire dataset as :* $\ell(X) = -\Sigma_i y_i \ln(p(x_i)) + (1 - y_i)\ln(1 - p(x_i))$*, where* $p(x_i) = [1 + \exp(-\beta x_i)]^{-1}$ *and* $\beta = \operatorname{argmin}_\beta \ell(X)$*. Then,*

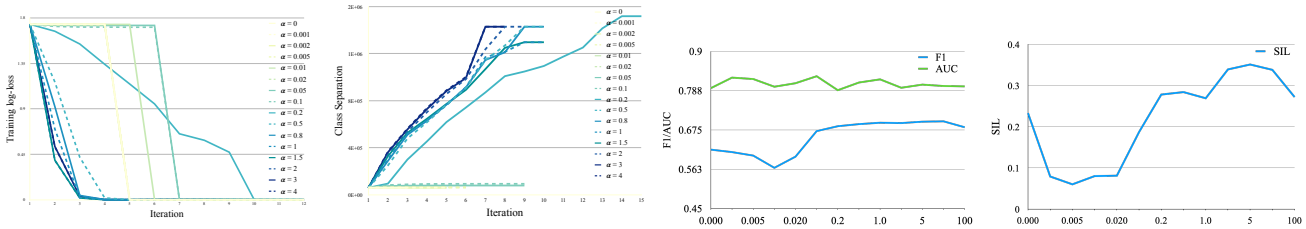
$$C_1 - (C_3\beta\mu^+ - C_4\beta\mu^-) \leq \ell(X) \leq C_2 - (C_3\beta\mu^+ - C_4\beta\mu^-)$$

where $C_1 = N\ln(2)$*,* $C_2 = N\ln(1+\exp(c)) - \frac{Nc}{2}$*,* $C_3 = \frac{N^+}{2}$*,* $C_4 = \frac{N^-}{2}$*,* $N^+ = \sum_{x_i \in X} y_i$*,* $N^- = \sum_{x_i \in X}(1 - y_i)$ *and* $c = \operatorname{argmax}_i \|\beta x_i\|$.

Note that Theorem 4 can be easily extended to multi-class scenario whereby we will get the bounds in a similar form i.e. $C_0 - (\sum_k C_k \beta\mu^k)$. C_0 would be the same for the upper and lower bound as in the above theorem. Additional constants denoting the counts of various class points ($C_k = \frac{N^k}{2}$) and $\beta\mu^k$, where N^k representing the number of points belonging to class k and μ^k representing the class centroid of class k can easily be anticipated.

6 DEEPCAC

CAC assumes linear class separability. To model non-linear boundaries in complex real-world data, we develop a deep learning variant of CAC. DEEPCAC is an autoencoder based deep clustering algorithm that induces clusters in an embedded space and can thus handle non-linear separation boundaries in the data space. Similar to CAC, DEEPCAC then trains individual local networks on the clusters thus found to



(a) Training Log-Loss vs iteration.

(b) Total Class separation v/s iterations of CAC for different α.

(c) F1/AUC (CIC)

(d) SIL (CIC)

Figure 2: CAC Training and Testing curves on Credit dataset with varying α ($k = 3$) and LR base classifier (a–b). Sensitivity Analysis with F1/AUC and Silhouette scores on the CIC dataset w.r.t. α for DEEPCAC (c–d)

Algorithm 2: DEEPCAC

Input: Training Data: $X \in \mathbb{R}^{n \times d}$, class labels $y^{n \times 1} \in [\mathcal{B}]^n$, $\{C_j\}_1^k$, $\{f_j\}_1^k$ and α

▷ **Initialization**

Pretrain E and D to initialize $\mathcal{U}, \mathcal{V}$

Compute $\mu(C_j), \mu^c(C_j)\ \forall\ C_j\ s.t.\ j \in \{1, \cdots, k\}, c \in \{1, \cdots, \mathcal{B}\}$.

▷ **Algorithm**

while *not converged* **do**

  for *every mini-batch* $\mathcal{X}_b$ **do**

    Update Network parameters by backpropagating L

    Update cluster assignments

    Update cluster centroids

▷ **Train Classifiers**

For every cluster C_j, train a classifier f_j on $(\mathcal{X}_j, \mathcal{Y}_j)$.

Output Classifiers $\mathcal{F} = \{f_j\}_{j=1}^{j=k}$ and cluster centroids $\mu = \{\mu_j\}_{j=1}^{j=k}$

build predictive risk models. DEEPCAC also modifies the class separability component in the overall loss function to accommodate multiple ($\mathcal{B}$) classes. The various components of DEEPCAC algorithm are described below.

6.1 NETWORK ARCHITECTURE

The DEEPCAC architecture is based on a stacked autoencoder consisting of an encoder and decoder parameterized by $(\mathcal{U}, \mathcal{V})$ as $E(\mathcal{U}) : X \rightarrow Z$ and $D(\mathcal{V}) : Z \rightarrow X$. Z denotes the space of lower dimensional embeddings of the original data points. There also exist k local networks $(\{LN_j(\mathcal{W}_j)\}_{j=1}^k)$ that are trained separately from the autoencoder. Similar to CAC, the data points are clustered but now in the embedded data space Z. Each cluster has its corresponding cluster and class centroids $(\mu_Z, \mu_Z^b$ s.t. $b \in [\mathcal{B}]$ as described previously. For notational simplicity, let M denote the centroid matrix and $s_{i,j}$ be the assignment vector of data point x_i which has only one non-zero element. $s_{j,i}$ denotes the j^{th} element of s_i, and the k^{th} column of M , i.e. μ_k, denotes the centroid of the k^{th} cluster.

6.2 LOSS FUNCTION

The CAC loss function encourages cluster structure formation and intra-cluster class separation by directly optimizing for the distance between class cluster centroids. But we observed that DEEPCAC was not able to directly optimize such a formulation of class separability through gradient descent. We thus use the Additive Margin Softmax Loss (AM Softmax) [32] to induce class seperablity within clusters. Apart from this, we also include the reconstruction loss from the decoder network to regularize the encoder as suggested in [33]. We also observed that normalizing the cluster SSE stabilizes training and prevents degenerate solutions. The final loss function of DEEPCAC thus becomes:

$$L = \sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - D(E(x_i))\|^2 + \frac{\beta * \|E(x_i) - M s_i\|^2}{|C_j|} - \frac{\alpha}{|C_j|} \sum_i \log \frac{e^{s \cdot (W_{y_i}^T z_i - m)}}{e^{s \cdot (W_{y_i}^T z_i - m)} + \sum_{j=1, j \neq y_i}^{c} e^{s W_j^T z_i}} \quad (3)$$

$$\text{s.t. } s_{j,i} \in \{0, 1\}, \mathbf{1}^T s_i = 1\ \forall i, j$$

where W_j^T is the j-th column of the last fully connected layer of the AMSoftmax loss.

6.3 TRAINING

Before actual training begins, the autoencoder parameters $(\mathcal{U}, \mathcal{V})$ are initialized by training the autoencoder to minimize the reconstruction loss without activating the CAC loss functions. Once the autoencoder is initialized, the data embeddings are clustered using k-means algorithm and the cluster centroids are initialized. The overall DEEPCAC loss function is non-convex and difficult to optimize. Stochastic gradient descent (SGD) cannot be directly applied to jointly optimize $\mathcal{U}, \mathcal{V}, M$ and $\{s_i\}$ because the block variable $\{s_i\}$ is constrained on a discrete set. We thus use an alternating

scheme, similar to [34], to optimize the subproblems w.r.t. one of $M, \{s_i\}$ and $(\mathcal{U}, \mathcal{V})$ while keeping the other two sets of variables fixed. Further details regarding the updating of network parameters are given in the Appendix C.

7 EXPERIMENTS

7.1 SYNTHETIC DATA

We perform two sets of experiments. The first set of experiments, on synthetic data, study the sensitivity of CAC on various characteristics of the data like cluster separation and class separation within clusters. We find that cluster-then-predict method does enhance overall classifier performance and exploiting class separation can lead to further improvements. Appendix B.1 presents the details of these experiments and the results. In the second set of experiments, described below, we compare the performance of CAC with that of other methods on real benchmark datasets. All experiments are conducted on a 2.6GHz Intel $i7$ CPU with 16 GB RAM running MacOS 11.1.

7.2 REAL DATA

Datasets. We use 6 benchmark binary-labelled datasets summarized in Table 1. In addition, we use a large, clinical dataset, that we call CIC, from the 2012 Physionet challenge [35] of predicting in-hospital mortality of intensive care units (ICU) patients at the end of their hospital stay. The data has time-series records, comprising various physiological parameters, of 12,000 patient ICU stays. We follow the data processing scheme of [36] (the top ranked team in the competition) to obtain static 117-dimensional features for each patient. We also derive a multiclass dataset (CIC-LOS) from the CIC dataset where we predict Length of Stay (LOS), discretized into 3 classes (based on 3 quartiles).

Experimental Setup. For each dataset we randomly choose 25% of the data as held-out data on which classifier performance is evaluated. The remaining data is used for training the classifiers and performing GridCV parameter search. Standard binary classification metric, the F1 score, is used to evaluate classifier performance.

Baselines. The state-of-the-art baseline method for combined clustering and classification is DMNN [4]. DMNN originally employs the k-means algorithm for clustering and a neural network based classifier. In addition, we use 7 different base classifiers, implemented by *sklearn* library (BSD licenced): LR, Linear SVM, Linear Discriminant Analysis (LDA), single Perceptron, Random Forest (#trees = 10), k Nearest Neighbors ($k = 5$), SGD classifier and Ridge classifier (Ridge regressor predicting over the range $[-1, 1]$) as a baseline directly and with k-means (KM+X) in a 'cluster-then-predict' approach. FNN has the base network of DEEPCAC but without any clustering. We implement DMNN ourselves in PyTorch.

Hyperparameters. For all KM+X and CAC+X experiments in Table 1, we choose $k = 2$ ($k = 3$ for Credit dataset) (after GridCV hyperparameter search). For CAC, we find the optimal value of α by performing a GridCV search over the range $[0.01, 3]$. We evaluate the algorithm on a separate held-out test dataset. For DMNN and DEEPCAC, the autoencoder has 3 layer of sizes $64 - 32 - 64$ and a two-layered local predictor network of size $30 - 1$. The autoencoder has L1-L2 regularization (L1: 1e-5, L2: 1e-4) and no activation. The local prediction networks use softmax and relu activation functions and are not regularized. More details are in Appendix B.2.

Setting α parameter. The performance of CAC heavily depends on the value of α. We illustrate this in Figure 2a for the Credit dataset. If α is too small, then the training will be very slow, and the clusters found will not have enough class separability. If α is too large, CAC will show overfitting behavior (Figure 4 in Appendix). Although the clusters formed will have large class separability and the training error of base classifiers will approach 0, because the clusters are too spread out, at test time, the performance will be poor. Figure 2c shows that the classification and clustering performance of DEEPCAC first deteriorates but then improves as α increases. We thus choose $\alpha = 5, \beta = 2$ for all DEEPCAC experiments.

Results. Table 1 presents the F1 scores obtained by all algorithms.

- We observe that the average % increase in F1 score of CAC (KM + classifier) over the base classifier ranges from 3.26% (1.85%) for Titanic dataset to 91.5% for the Diabetes dataset (15.07% for Credit dataset) (values not in Table 1). This is in line with theorem 1 that predicts better performance of multiple simple classifiers as compared to a single simple classifier.
- CAC performs better than the simple k-means + classification approach on all the datasets. This also serves as an ablative evidence of the contribution of linear separability criteria since removing the latter reduces CAC to applying k-means and then predicting.
- DEEPCAC beats the baseline FNN and DMNN consistently on all datasets (except Adult) by a large margin. Note that as hypothesized in Section 4, on relatively smaller datasets such as Titanic and Heart datasets, simpler models with clustering work better than complex models (FNN, DMNN and DeepCAC).
- On the Heart dataset, having just 303 samples, simple base classifiers outperform any clustering based or complex neural network model. The moderately

Table 1: F1 of each method on different datasets evaluated on a separate held out testing dataset. $*/*/*$ indicates the number of times the base classifier, KM+Base and CAC+Base attains the best performance. Highlighted values signify the best result in that category. Size of datasets are mentioned below in parentheses.

Dataset/ Classifier	Titanic (2200)	Heart (303)	Credit (30000)	Adult (45000)	Diabetes (100000)	CIC (12000)	CIC-LOS (12000)	Total
LR	**0.839**	**0.871**	0.336	0.556	0.12	0.45	-	
KM + LR	0.674	0.718	0.505	0.619	**0.503**	**0.468**	-	$2/2/2$
CAC + LR	0.838	0.623	**0.506**	**0.642**	0.497	0.467	-	
SVM	0.84	**0.884**	0.249	0.532	0.112	0.412	-	
KM + SVM	0.862	0.881	0.398	0.601	0.125	0.44	-	$\mathbf{1/0/5}$
CAC + SVM	**0.862**	0.857	**0.447**	**0.62**	**0.497**	**0.444**	-	
LDA	0.828	**0.897**	0.358	0.531	0.12	0.446	-	
KM + LDA	0.857	0.892	**0.423**	0.592	0.105	**0.487**	-	$\mathbf{1/2/3}$
CAC + LDA	**0.857**	0.881	0.421	**0.62**	**0.497**	0.48	-	
Perceptron	0.729	0.765	0.367	0.482	0.469	0.378	-	
KM + Perceptron	**0.837**	**0.857**	0.329	0.579	0.407	0.421	-	$\mathbf{0/2/4}$
CAC + Perceptron	0.831	0.692	**0.457**	**0.62**	**0.497**	**0.446**	-	
RF	0.836	0.800	0.409	0.659	0.413	0.348	-	
KM + RF	0.835	**0.829**	0.396	0.655	0.409	0.343	-	$\mathbf{0/1/5}$
CAC + RF	**0.839**	0.825	**0.446**	**0.662**	**0.497**	**0.444**	-	
KNN	0.809	0.843	0.425	**0.654**	0.415	0.258	-	
KM + KNN	0.833	0.841	0.421	0.653	0.411	0.225	-	$\mathbf{1/0/5}$
CAC + KNN	**0.833**	**0.854**	**0.446**	0.653	**0.497**	**0.444**	-	
SGD	0.831	**0.810**	0.272	0.531	0.248	0.438	-	
KM + SGD	**0.857**	0.729	0.250	0.605	0.15	0.376	-	$\mathbf{1/1/4}$
CAC + SGD	0.835	0.759	**0.375**	**0.62**	**0.497**	**0.472**	-	
Ridge	0.828	**0.897**	0.223	0.495	0.112	0.329	-	
KM + Ridge	**0.858**	0.884	0.372	0.581	0.124	0.356	-	$\mathbf{1/1/4}$
CAC + Ridge	0.858	0.871	**0.446**	**0.62**	**0.497**	**0.444**	-	
FNN	0.591	0.797	0.690	**0.782**	0.443	0.652	0.431	
DMNN	0.523	0.811	0.631	0.740	0.262	0.595	0.412	$\mathbf{1/0/6}$
DEEPCAC($k = 3$)	**0.655**	**0.849**	**0.690**	0.777	**0.486**	**0.697**	**0.468**	
Total	$1/5/3$	$5/2/1$	$\mathbf{0/1/8}$	$\mathbf{1/1/7}$	$\mathbf{0/0/9}$	$\mathbf{0/2/7}$	$\mathbf{0/0/1}$	

sized Titanic dataset demonstrates how simple, cluster-then-predict algorithms outperforms more complex neural network methods.

8 CONCLUSION

Clustering has been effectively used with classification models in many previous models and for various applications. In this paper, we theoretically analyze potential reasons for the good performance of classifiers trained on underlying clusters in data. Our analysis yields insights into the benefits of using simple classifiers trained on clusters compared to simple or complex classifiers trained without clustering. Deeper analysis along these lines would be an interesting direction for future work.

Our analysis motivates the development of a new clustering-based framework for classification, CAC, that enforces class separability within each cluster during cluster discovery. We design an efficient iterative algorithm to find such clusters. Unlike most previous approaches, CAC is simple, efficient, and can be used with any classifier. We also propose DEEPCAC a deep variant of CAC that can handle multi-class data and non-linear separation boundaries in data. In our experiments, we find that CAC and DEEPCAC outperforms state-of-the-art methods of combined clustering and classification.

It would be interesting to explore the benefits offered by unsupervised techniques in supervised applications like adversarial robustness, efficient regularization etc. Recent success of self supervised and contrastive learning approaches strongly suggests untapped potential that may be exploited.

References

[1] Qiang Qian, Songcan Chen, and Weiling Cai. Simultaneous clustering and classification over cluster structure representation. *Pattern Recognition*, 2012.

[2] Quanquan Gu and Jiawei Han. Clustered support vector machines. In *Artificial Intelligence and Statistics*, 2013.

[3] Tanmoy Chakraborty. Ec3: Combining clustering and classification for ensemble learning. In *ICDM*, 2017.

[4] Xiangrui Li, Dongxiao Zhu, and Phillip Levy. Predicting clinical outcomes with patient stratification via deep mixture neural networks. *AMIA Summits on Translational Science*, 2020.

[5] Ralph Snyderman. Personalized health care: from theory to practice. *Biotechnology journal*, 2012.

[6] Ahmed M Alaa, Jinsung Yoon, Scott Hu, and Mihaela Van der Schaar. Personalized risk scoring for critical care patients using mixtures of gaussian process experts. *arXiv preprint arXiv:1605.00959*, 2016.

[7] Harini Suresh, Jen J Gong, and John V Guttag. Learning tasks for multitask learning: Heterogenous patient populations in the icu. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[8] Zina M Ibrahim, Honghan Wu, Ahmed Hamoud, Lukas Stappen, Richard JB Dobson, and Andrea Agarossi. On classifying sepsis heterogeneity in the icu: insight using machine learning. *Journal of the American Medical Informatics Association*, 2020.

[9] Tae-Kyun Kim and Roberto Cipolla. Mcboost: Multiple classifier boosting for perceptual co-clustering of images and visual features. *Advances in Neural Information Processing Systems*, 21:841–848, 2008.

[10] Mohammad Peikari, Sherine Salama, Sharon Nofech-Mozes, and Anne L Martel. A cluster-then-label semi-supervised learning approach for pathology image classification. *Scientific reports*, 2018.

[11] Izzat Alsmadi and Ikdam Alhami. Clustering and classification of email contents. *Journal of King Saud University-Computer and Information Sciences*, 2015.

[12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT Press, 2012.

[13] Manchek A Wong and JA Hartigan. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 1979.

[14] Sibylle Hess, Wouter Duivesteijn, and Decebal Mocanu. Softmax-based classification is k-means clustering: Formal proof, consequences for adversarial attacks, and improvement through centroid based tailoring. *arXiv:2001.01987*, 2020.

[15] Hind Elouedi, Walid Meliani, Zied Elouedi, and Nahla Ben Amor. A hybrid approach based on decision trees and clustering for breast cancer classification. In *SoCPaR*, 2014.

[16] Martin Vincent and Niels Richard Hansen. Sparse group lasso and high dimensional multinomial classification. *Computational Statistics & Data Analysis*, 2014.

[17] Andrea Visentin, Alessia Nardotto, and Barry O'Sullivan. Predicting judicial decisions: A statistically rigorous approach and a new ensemble classifier. In *ICTAI*, 2019.

[18] Ahmed Mahfouz, Abdullah Abuhussein, Deepak Venugopal, and Sajjan Shiva. Ensemble classifiers for network intrusion detection using a novel network attack dataset. *Future Internet*, 2020.

[19] William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. The power of ensembles for active learning in image classification. In *CVPR*, 2018.

[20] Ivan Titov, Alexandre Klementiev, Kevin Small, and Dan Roth. Unsupervised aggregation for classification problems with large numbers of categories. In *AISTATS*, 2010.

[21] Botao Hao, Will Wei Sun, Yufeng Liu, and Guang Cheng. Simultaneous clustering and estimation of heterogeneous graphical models. *JMLR*, 2018.

[22] G. Qi, Q. Tian, and T. Huang. Locality-sensitive support vector machine by exploring local correlation and global regularization. In *CVPR*, 2011.

[23] Philip HS Torr. Locally linear support vector machines. In *ICML*, 2011.

[24] Zhouyu Fu, Antonio Robles-Kelly, and Jun Zhou. Mixing linear svms for nonlinear classification. *IEEE Transactions on Neural Networks*, 2010.

[25] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *JMLR*, 2009.

[26] Kun Song, Feiping Nie, Junwei Han, and Xuelong Li. Parameter free large margin nearest neighbor for distance metric learning. In *AAAI*, 2017.

[27] Hiroaki Sasaki, Takashi Takenouchi, Ricardo Monti, and Aapo Hyvarinen. Robust contrastive learning and nonlinear ica in the presence of outliers. In *UAI*, 2020.

[28] Conor Durkan, Iain Murray, and George Papamakarios. On contrastive learning for likelihood-free inference. In *ICML*, 2020.

[29] Weiling Cai, Songcan Chen, and Daoqiang Zhang. A simultaneous learning framework for clustering and classification. *Pattern Recognition*, 2009.

[30] Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *JMLR*, 2002.

[31] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 1982.

[32] Feng Wang, Jian Cheng, Weiyang Liu, and Haijun Liu. Additive margin softmax for face verification. *IEEE Signal Processing Letters*, 25(7):926–930, 2018.

[33] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *IJCAI*, 2017.

[34] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *international conference on machine learning*, pages 3861–3870. PMLR, 2017.

[35] Ikaro Silva, George Moody, Daniel J Scott, Leo A Celi, and Roger G Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. *Computing in cardiology*, 2012.

[36] Alistair EW Johnson, Nic Dunkley, Louis Mayaud, Athanasios Tsanas, Andrew A Kramer, and Gari D Clifford. Patient specific predictions in the intensive care unit using a bayesian ensemble. In *CinC*, 2012.

[37] Sanjoy Dasgupta. *The hardness of k-means clustering*. University of California, 2008.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, et al. Scikit-learn: Machine learning in Python. *JMLR*, 2011.

A PROOF OF THEOREMS

A.1 PROOF OF THEOREM 1

Theorem 5. *Let $\mathcal{F}_j$ be a set of maps $x \mapsto f(x)$, and the function $q \mapsto \ell(q, y)$ be a λ-uniformly bounded function for any $q \in \{f(x) : f \in \mathcal{F}_j, x \in \mathcal{X}_j\}$ and $y \in \mathcal{Y}_j$. Then, for any $\delta > 0$, with probability at least $1 - \delta$ over an i.i.d. draw of m i.i.d. samples $((x_i, y_i))_{i=1}^m$, the following holds: for all maps $f \in \mathcal{F}_j$,*

$$\mathbb{E}_{x,y}[\ell(f(x), y)] \leq \frac{1}{m_j} \sum_{i=1}^{m_j} \ell(f(x_i), y_i) + 2\mathcal{R}_{m_j}(\ell \circ \mathcal{F}_j) + \lambda \sqrt{\frac{\ln(1/\delta)}{2m_j}}$$

where $\mathcal{R}_{m_j}(\ell \circ \mathcal{F}_j) := \mathbb{E}_{S,\sigma}[\sup_{f \in \mathcal{F}_j} \frac{1}{m_j} \sum_{i=1}^{m_j} \sigma_i \ell(f(x_i), y_i)]$ where $\sigma_1, \ldots, \sigma_{m_j}$ are independent uniform random variables taking values in $\{-1, 1\}$.

Proof of Theorem 1. Let $S = ((x_i, y_i))_{i=1}^m$ and $S' = ((x'_i, y'_i))_{i=1}^m$. Define

$$\varphi(S) = \sup_{f \in \mathcal{F}_j} \mathbb{E}_{x,y}[\ell(f(x), y)] - \frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i), y_i). \quad (4)$$

To apply McDiarmid's inequality to $\varphi(S)$, we compute an upper bound on $|\varphi(S) - \varphi(S')|$ where S and S' be two test datasets differing by exactly one point of an arbitrary index i_0; i.e., $S_i = S'_i$ for all $i \neq i_0$ and $S_{i_0} \neq S'_{i_0}$. Then,

$$\varphi(S') - \varphi(S) \leq \sup_{f \in \mathcal{F}_j} \frac{\ell(f(x_{i_0}), y_{i_0}) - \ell(f(x'_{i_0}), y'_{i_0})}{m} \leq \frac{\lambda}{m}. \quad (5)$$

Thus, by McDiarmid's inequality, for any $\delta > 0$, with probability at least $1 - \delta$,

$$\varphi(S) \leq \mathbb{E}_S[\varphi(S)] + \lambda \sqrt{\frac{\ln(1/\delta)}{2m}}. \quad (6)$$

Moreover,

$$\mathbb{E}_S[\varphi(S)] \quad (7)$$

$$= \mathbb{E}_S \left[\sup_{f \in \mathcal{F}_j} \mathbb{E}_{S'} \left[\frac{1}{m} \sum_{i=1}^{m} \ell(f(x'_i), y'_i) \right] - \frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i), y_i) \right] \quad (8)$$

$$\leq \mathbb{E}_{S,S'} \left[\sup_{f \in \mathcal{F}_j} \frac{1}{m} \sum_{i=1}^{m} (\ell(f(x'_i), y'_i) - \ell(f(x_i), y_i)) \right] \quad (9)$$

$$\leq \mathbb{E}_{\xi,S,S'} \left[\sup_{f \in \mathcal{F}_j} \frac{1}{m} \sum_{i=1}^{m} \xi_i (\ell(f(x'_i), y'_i) - \ell(f(x_i), y_i)) \right] \quad (10)$$

$$\leq 2\mathbb{E}_{\xi,S} \left[\sup_{f \in \mathcal{F}_j} \frac{1}{m} \sum_{i=1}^{m} \xi_i \ell(f(x_i), y_i)) \right] = 2\mathcal{R}_m(\ell \circ \mathcal{F}_j) \quad (11)$$

where the first line follows the definitions of each term, the second line uses the Jensen's inequality and the convexity of the supremum, and the third line follows that for each $\xi_i \in \{-1, +1\}$, the distribution of each term $\xi_i(\ell(f(x'_i), y'_i) - \ell(f(x_i), y_i))$ is the distribution of $(\ell(f(x'_i), y'_i) - \ell(f(x_i), y_i))$ since S and S' are drawn iid with the same distribution. The forth line uses the subadditivity of supremum. □

A.2 PROOF OF THEOREM 2

Theorem 2. *Algorithm 1 converges to a local minimum.*

Proof. To prove convergence, it suffices to verify that the cost function, that re-assigns point x from cluster C_p to cluster C_q, monotonically decreases at each iteration. This is equivalent to showing $\phi(C_p \setminus \{x_i\}) + \phi(C_q \cup \{x_i\}) < \phi(C_p) + \phi(C_q) \implies (\phi(C_p \setminus \{x_i\}) - \phi(C_p)) + (\phi(C_q \cup \{x_i\}) - \phi(C_q)) < 0 \implies \Gamma^+(C_q, x) + \Gamma^-(C_p, x) < 0 \implies \Phi(x_i, C_p, C_q) < 0$.

$\Phi(x_i, C_p, C_q) < 0$ is exactly the update condition for the algorithm that ensures that the total cost function is always decreasing with every point update and in every iteration. □

A.3 PROOF OF THEOREM 3

Theorem 3. *The time complexity for each round of CAC is $O(ndk)$.*

Proof. For the time complexity of each round, it suffices to prove the running time for each point x_i is $O(dk)$. For each $j \in [k]$, we denote the centroid, positive centroid and negative centroid of $C_j \cup \{x_i\}$ as $\tilde{\mu}(C_j)$, $\tilde{\mu}^+(C_j)$ and

$\tilde{\mu}^-(C_j)$ respectively. Given a cluster C_j and a point $z \in \mathbb{R}^d$, we denote the clustering cost of C w.r.t. z to be

$$g(C_j, z) := \sum_{x \in C_j} \|x - z\|^2.$$

A standard result from the k-means literature [37] is the following bias-variance decomposition of the k-means cost function:

$$g(C_j, z) = \phi(C_j, \mu(C_j)) + |C_j| \cdot \|\mu(C_j) - z\|^2.$$

Note that the above result holds for CAC cluster cost function also as can be shown easily by a little algebra. Based on this property, we have

$$\begin{aligned}
&\Gamma^+(C_j, x_i) \\
=&\|\tilde{\mu} - x_i\|^2 + |C_j| \cdot \|\tilde{\mu}(C_j) - \mu(C_j)\|^2 \\
&+ \alpha|C_j| \cdot \|\mu^+(C_j) - \mu^-(C_j)\|^2 \\
&- \alpha(|C_j| + 1) \cdot \|\tilde{\mu}^+(C_j) - \tilde{\mu}^-(C_j)\|^2 \\
&\Gamma^-(C_j, x_i) \\
=&- \|\mu(C_j) - x_i\|^2 - (|C_j| - 1)\|\tilde{\mu}(C_j) - \mu(C_j)\|^2 \\
&+ \alpha|C_j| \cdot \|\mu^+(C_j) - \mu^-(C_j)\|^2 \\
&- \alpha(|C_j| - 1) \cdot \|\tilde{\mu}^+(C_j) - \tilde{\mu}^-(C_j)\|^2
\end{aligned}$$

Then it suffices to prove that both $\Gamma^+(C_j, x_i)$ and $\Gamma^-(C_j, x_i)$ can be computed in $O(d)$ time, which implies $O(dk)$ time for computing $q := \text{argmin}_j \Phi(x_i; C_p, C_j)$. Moreover, by the above formulations, it suffices to prove that $\tilde{\mu}(C_j)$, $\tilde{\mu}^+(C_j)$ and $\tilde{\mu}^-(C_j)$ can be computed in $O(d)$ time. Note that $\tilde{\mu}(C_p) = \frac{|C_p| \cdot \mu(C_p) - x_i}{|C_p| - 1}$ and $\tilde{\mu}(C_j) = \frac{|C_j| \cdot \mu(C_j) + x_i}{|C_j| + 1}$ for $j \neq p$. Hence, each $\tilde{\mu}(C_j)$ can be computed in $O(d)$ time. We discuss the following two cases for the computation of $\tilde{\mu}^+(C_j)$ and $\tilde{\mu}^-(C_j)$.

- Case $y_i = 1$. We have that $\tilde{\mu}^+(C_p) = \frac{\left(\sum_{x_l \in C_p} y_l\right) \cdot \mu_p^+ - x_i}{\sum_{x_l \in C_p} y_l - 1}$ and $\tilde{\mu}^-(C_p) = \mu^-(C_p)$. For $j \neq p$, we have that $\tilde{\mu}^+(C_j) \leftarrow \frac{\left(\sum_{x_l \in C_j} y_l\right) \cdot \mu_j^+ + x_i}{\sum_{x_l \in C_j} y_l + 1}$ and $\tilde{\mu}^-(C_j) = \mu^-(C_j)$.
- Case $y_i = 0$. We have that $\tilde{\mu}_p^- \leftarrow \frac{\left(\sum_{x_l \in C_p} (1 - y_l)\right) \cdot \mu_p^- - x_i}{\sum_{x_l \in C_p} (1 - y_l) - 1}$ and $\tilde{\mu}^+(C_p) = \mu^+(C_p)$. For $j \neq p$, we have that $\tilde{\mu}_j^- \leftarrow \frac{\left(\sum_{x_l \in C_j} (1 - y_l)\right) \cdot \mu_j^- + x_i}{\sum_{x_l \in C_j} (1 - y_l) + 1}$ and $\tilde{\mu}^+(C_j) = \mu^+(C_j)$.

By the above update rules, both $\tilde{\mu}^+(C_j)$ and $\tilde{\mu}^-(C_j)$ can be computed in $O(d)$ time, which completes the proof. □

A.4 PROOF OF LEMMA 1

Lemma 2. *For d dimentional vectors $\mathbf{u}, \mathbf{v}$ and $\boldsymbol{\beta}$, $\frac{\partial \Delta}{\partial \|\mathbf{u}\|} < 0$ for $\Delta = A - (B\boldsymbol{\beta}\mathbf{u} + C\boldsymbol{\beta}\mathbf{v})$ if $\beta_i u_i > 0, \forall i \in \{1, 2, \cdots, d\}$ and $A, B > 0$.*

Proof. Let u_i be the i^{th} element of $\mathbf{u}$. Expanding $\frac{\partial \Delta}{\partial \|\mathbf{u}\|}$ using chain rule, we get

$$\frac{\partial \Delta}{\partial \|\mathbf{u}\|} = \frac{\partial \Delta}{\partial \mathbf{u}} \cdot \frac{\partial \mathbf{u}}{\partial \|\mathbf{u}\|} = -B\boldsymbol{\beta} \cdot \frac{\partial \mathbf{u}}{\partial \|\mathbf{u}\|} \text{ (Since } \frac{\partial \Delta}{\partial \mathbf{u}} = -B\boldsymbol{\beta}\mathbf{u})$$

Let u_i denote the i^{th} element of $\mathbf{u}$. Then, from the definition of vector derivatives we have,

$$\begin{aligned}
\frac{\partial \mathbf{u}}{\partial \|\mathbf{u}\|} &= \left[\frac{\partial u_1}{\partial \|\mathbf{u}\|}, \frac{\partial u_2}{\partial \|\mathbf{u}\|}, \cdots, \frac{\partial u_d}{\partial \|\mathbf{u}\|}\right] \\
&= \left[\left(\frac{\partial \|\mathbf{u}\|}{\partial u_1}\right)^{-1}, \left(\frac{\partial \|\mathbf{u}\|}{\partial u_2}\right)^{-1}, \cdots, \left(\frac{\partial \|\mathbf{u}\|}{\partial u_d}\right)^{-1}\right] \\
&= \|\mathbf{u}\| \left[\frac{1}{u_1}, \frac{1}{u_2}, \cdots, \frac{1}{u_d}\right]
\end{aligned}$$

Thus, $\frac{\partial \Delta}{\partial \|\mathbf{u}\|} = -B \cdot \|\mathbf{u}\| \cdot \sum_{i=1}^{d} \frac{\beta_i}{u_i}$, where $u_i = \mu_i^+ - \mu_i^-$. Since $\beta \cdot \mu^+ > 0$ and $\beta \cdot \mu^- < 0$, it implies that $\beta \cdot \mathbf{u} > 0$. Assuming that every dimension of $\mathbf{u}$ lies on the same side of regression plane β as the original point, i.e. $\beta_i u_i = \beta_i > 0, \forall i$, $\frac{\beta_i}{u_i} > 0 \ \forall i$, then $\frac{\partial \Delta}{\partial \|\mathbf{u}\|} < 0$ as B and $\|\mathbf{u}\|$. Thus under above assumption, we prove that Δ decreases as $\|\mathbf{u}\|$ increases.

□

A.5 PROOF OF THEOREM 4

Theorem 4. *Let $X = (x_1, x_2, \cdots, x_N)$ be a training dataset with n records $x_i \in \mathbb{R}^d$ together with binary labels $y_i \in \{0, 1\}$. Define the log-loss over the entire dataset as :*

$$\ell(X) = -\Sigma_i y_i \ln(p(x_i)) + (1 - y_i) \ln(1 - p(x_i))$$

where $p(x_i) = [1 + \exp(-\beta x_i)]^{-1}$ and $\beta = \text{argmin}_\beta \, \ell(X)$. Then,

$$C_1 - (C_3\beta\mu^+ - C_4\beta\mu^-) \leq \ell(X) \leq C_2 - (C_3\beta\mu^+ - C_4\beta\mu^-)$$

where $C_1 = N \ln(2)$, $C_2 = N \ln(1 + \exp(c)) - \frac{Nc}{2}$, $C_3 = \frac{N^+}{2}$, $C_4 = \frac{N^-}{2}$, $N^+ = \sum_{x_i \in X} y_i$, $N^- = \sum_{x_i \in X} (1 - y_i)$ and $c = \text{argmax}_i \, \|\beta x_i\|$.

Proof. **Lower Bound** Let $\ell(X, y) = \Sigma_{y_i = 1} \ell^+(x_i) + \Sigma_{y_i = 0} \ell^-(x_i)$ s.t.

$$\ell^+(x_i) = -\ln(p_i), \ \ell^-(x_i) = -\ln(1 - p_i) \tag{12}$$

Applying Jensen's inequality on ℓ^+ and ℓ^-,

$$\begin{aligned}\ell(X) =&\Sigma_{x_i\in X^+}\ell^+(x_i) + \Sigma_{x_i\in X^-}\ell^-(x_i)\\ =&-\Sigma_{y_i=1}\ln[1+\exp(-\beta x_i)]^{-1}-\\ &\Sigma_{y_i=0}\ln[1-(1+\exp(-\beta x_i))^{-1}] \quad \text{(By eq. 12)}\end{aligned}$$

$$\begin{aligned}\geq& -N^+\ln\left[1+\exp\left(-\beta\cdot\left(\frac{\Sigma_{x_i\in X^+}x_i}{N^+}\right)\right)\right]^{-1}\\ &-N^-\ln\left[1-\left(1+\exp\left(-\beta\cdot\left(\frac{\Sigma_{x_i\in X^-}x_i}{N^-}\right)\right)\right)^{-1}\right]\end{aligned}$$

(Jensen's inequality for convex functions)

$$\begin{aligned}\geq& N^+\ln[1+\exp(-\beta\mu^+)] - N^-\ln\left[\frac{\exp(-\beta\mu^-)}{1+\exp(-\beta\mu^-)}\right]\end{aligned}$$

(Substituting μ^+ and μ^-)

$$\geq N^+\ln[1+\exp(-\beta\mu^+)] + N^-\ln[1+\exp(\beta\mu^-)]$$

We observe that $\ln(1+\exp(x)) \geq \ln(2)+\frac{x}{2}\ \forall x\in\mathbb{R}$ by considering the Taylor series expansion of $\ln(1+\exp(x))$ at $x=0$. $\ln(1+\exp(x)) = \ln(2)+\frac{x}{2}+\frac{x^2}{8}+\mathcal{O}(x^4)$. Then,

$$\begin{aligned}\ell(X) &\geq N^+\left(\ln(2)-\frac{\beta\mu^+}{2}\right)+N^-\left(\ln(2)+\frac{\beta\mu^-}{2}\right)\\ &\geq N\ln(2)-\frac{1}{2}\cdot(N^+\beta\mu^+ - N^-\beta\mu^-)\end{aligned}$$

where $C_1 = N\ln(2), C_3 = \frac{N^+}{2}, C_4 = \frac{N^-}{2}$.

Upper Bound

Since all points x_i are constrained within their respective clusters, we note that the maximum log-loss for a point x_i labelled as y_i is not unbounded. Let $c = \text{argmax}_i \|\beta x_i\|$. Since ℓ^+ is a monotonically decreasing function and ℓ^- is monotonically increasing, we can bound them using linear functions s^+ and s^- respectively. $s^+ \geq \ell^+$ and $s^- \geq \ell^-$ for all points X^+ and X^- respectively. $s^+(x_i)$ passes through $(-c, \ln(p(-c)))$ and $(c, \ln(p(c)))$. Hence $s^+(x) = K_1 - K_2\cdot\beta x$ where $K_1, K_2 > 0$. Similarly, $s^-(x) = K_3 + K_4\cdot\beta x$ where $K_3, K_4 > 0$.

Solving for $K_1\cdots K_4$, we get

$$K_1 = K_3 = \ln(1+\exp(c)) - \frac{c}{2},\ K_2 = K_4 = \frac{1}{2}$$

Then,

$$\begin{aligned}\ell(X) &= \Sigma_{x_i\in X^+}\ell^+(x_i) + \Sigma_{x_i\in X^-}\ell^-(x_i)\\ &< \Sigma_{x_i\in X^+}s^+(x_i) + \Sigma_{x_i\in X^-}s^-(x_i)\\ &\leq N^+s^+\left(\frac{\Sigma_{x_i\in X^+}x_i}{N^+}\right) + N^-s^-\left(\frac{\Sigma_{x_i\in X^-}x_i}{N^-}\right)\end{aligned}$$

(Jensen's inequality for a linear function)

$$\begin{aligned}&\leq N^+s^+(\mu^+) + N^-s^-(\mu^-)\\ &\leq N^+(K_1 - K_2\cdot\beta\mu^+) + N^-(K_3 + K_4\cdot\beta\mu^-)\\ &\leq C_2 - (C_3\beta\mu^+ - C_4\beta\mu^-) \text{ (Rearranging terms)}\end{aligned}$$

where $C_2 = N\ln(1+\exp(c)) - \frac{Nc}{2}$ and $C_3 = \frac{N^+}{2}, C_4 = \frac{N^-}{2}$. □

B SUPPLEMENTARY EXPERIMENTAL RESULTS AND DISCUSSIONS

B.1 RESULTS ON SYNTHETIC DATASET

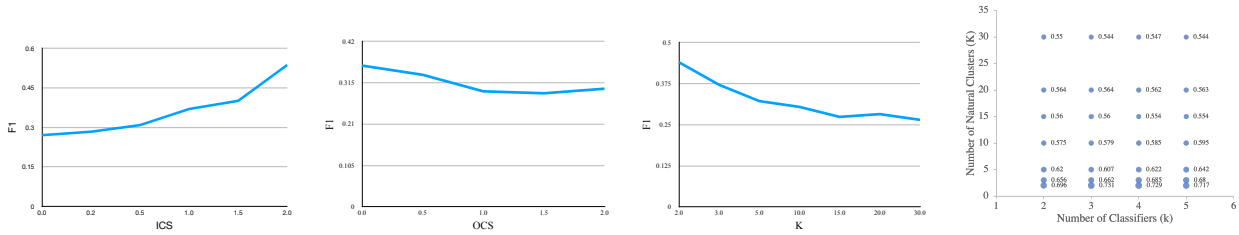
We analyze how the performance of CAC depends on the following data/algorithm parameters:

1. Inner Class Separation (ICS): Distance between class centroids in an individual cluster.
2. Outer Cluster Separation (OCS): Distance between cluster centroids.
3. Number of natural clusters in the dataset (K).
4. Number of classifiers used in CAC (k), i.e., the number of clusters used as input.

To study the performance of CAC as a function of these 4 parameters, we modify the `make classification` function of *sklearn* library [38], to generate custom datasets with varying ICS, OCS, and K. We find that the performance of CAC improves with increasing ICS. This is in line with our hypothesis that if the classes are well separated within the clusters, then it will improve the performance of the classifiers. The performance of CAC does not depend much on the OCS. As the number of natural clusters (K) in the dataset increases, the performance of CAC decreases. On the other hand, the performance is better when $k > K$.

We choose Logistic Regression as the base classifier for CAC while testing its performance on the synthetic dataset as described in Appendix B.1. The range of these parameters in our simulations are as follows: $ICS \in \{0, 0.2, 0.5, 1, 1.5, 2\}$, $OCS \in \{1, 1.5, 2\}$, $K \in \{2, 3, 5, 10, 15, 20, 30\}$ and $k \in \{2, 3, 4, 5\}$. We run CAC+LR for all combination of parameters.

Figure 3 presents the results of the experiments. In each subfigure, the results are averaged over all the parameters not shown in the axes. In Figure 3a, note that the performance of CAC improves as Inner Class Separation within



(a) Effect of Inter Class Separation (ICS) on F1 and AUC.

(b) Effect of Outer Cluster Separation (OCS) on F1 and AUC.

(c) Effect of number of natural clusters (K) on F1 and AUC.

(d) AUC with varying #classifiers (k) and Natural #Clusters (K).

Figure 3: Evaluating the effect of different parameters on the performance of CAC

the clusters increases. This is in line with our hypothesis that if the classes are well separated within the clusters, then it will improve the performance of the classifiers. The performance of CAC does not depend much on the Outer Cluster Separation (Figure 3b). This is also expected as a classifier will be relatively unaffected by the presence of another cluster, irrespective of its distance as the data in that cluster is not a part of its training set.

As the number of natural clusters in the dataset increases, the performance of CAC decreases (Figure 3c). This phenomenon is expected from linear classifiers as they cannot handle nonlinearities in data. Hence, dividing the data into multiple groups will decrease the piece-wise nonlinearity and the overall algorithm will work better [2]. In Figure 3d, we observe that the results are better for the cases when $k \geq K$ (i.e., closer to the points near the x-axis). This shows that the performance is better when the number of clusters given as input is not less than the natural clusters in the data, compared to the case when the input number of clusters is less than the natural number of clusters. In the latter case, each (linear) classifier has to learn a non-linear boundary due to the intrinsic cluster structure that leads to a decrease in the performance.

B.2 HYPERPARAMETER DETAILS

Table 2 shows the tuned values of α used by different variants of CAC on different datasets. We find these values by performing a 5-fold Grid search cross validation. Figure 4 shows how the CAC cost function decreases monotonically (as proved in Theorem 2) over iterations for different values of α.

Table 2: Tuned α values used for CAC in experiments found after 5-fold GridCV search

Classifier	Titanic	Credit	Adult	Diabetes	CIC
LR	0.8	0.02	0.05	2.5	0.05
SVM	0.01	0.15	0.15	2.5	0.5
LDA	0.08	0.02	0.15	2.5	0.05
Perceptron	0.5	0.2	0.15	2.5	0.5
RF	0.05	0.15	0.02	2.5	0.5
KNN	0.01	0.15	0.02	2.5	0.5
SGD	0.3	0.08	0.15	2.5	0.01
Ridge	0.01	0.15	0.15	2.5	0.5
DMNN	0.2	0.3	0.8	0.15	0.1

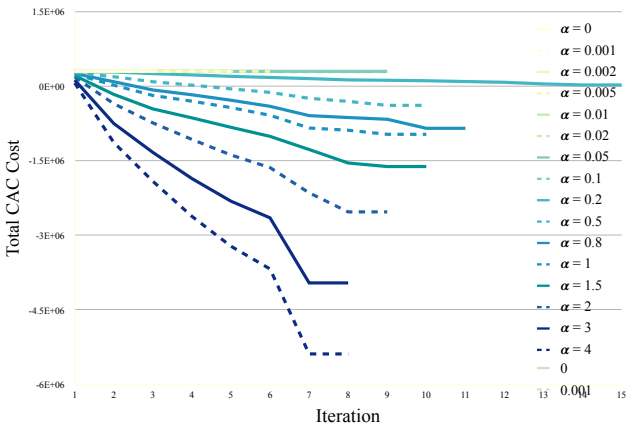


Figure 4: Total CAC cost v/s # of iterations for different α.

C DEEPCAC TRAINING DETAILS

Updating Clustering Parameters For optimizing the clustering loss, we follow the procedure defined in [34]. For fixed network parameters and cluster assignment matrix M, the assignment vectors of the current sample, i.e., s_i, are updated in an online fashion. Specifically, s_i is updated as follows:

$$s_{i,j} = \begin{cases} 1, & \text{if } j = \text{argmin}_l \, \|f(x_i) - \mu_l\|^2 \\ 0, & \text{otherwise} \end{cases}$$

In [34], the authors update the cluster centroids in an online manner instead of simply taking the average of every minibatch as the current minibatch might not be representative of the global cluster structure. So the cluster centroids are

updated as follows by a simple gradient step:

$$\mu_j \leftarrow \mu_j - \frac{1}{c_k^i}(\mu_j - f(x_i))s_{k,i}$$

where c_k^i is the count of the number of times algorithm assigned a sample to cluster k before handling the incoming sample x_i. The class cluster centroids μ_k^b are also updated in a similar, online manner due to the above mentioned reasons.

Updating Autoencoder's weights. For fixed (M, s_i), the subproblem of optimizing $(\mathcal{W},\mathcal{U})$ is similar to training an SAE – but with an additional loss term on clustering performance. Modern deep learning libraries like PyTorch allow us to easily backpropagate both the losses simultaneously. To implement SGD for updating the network parameters, we look at the problem w.r.t. the incoming data x_i:

$$\min_{\mathcal{U},\mathcal{W}} L^i = \ell(g(f(x_i)), x_i) + \beta\|f(x_i) - Ms_i\| + \alpha\sum_{j=1}^{k} L_{AM}$$

The gradient of the above function over the network parameters is easily computable, i.e., $\nabla X_{\mathcal{J}} L_i = \frac{\partial \ell(g(f(x_i;\mathcal{W});\mathcal{U})}{\partial \mathcal{J}} + \alpha\frac{\partial f(x_i)}{\partial \mathcal{J}}(f(x_i) - Ms_i)$, where $\mathcal{J} = (\mathcal{W},\mathcal{U},\mathcal{V})$ is a collection of network parameters and the gradients $\frac{\partial l}{\partial \mathcal{J}}$ can be calculated by back-propagation. Then, the network parameters are updated by

$$\mathcal{J} \leftarrow \mathcal{J} - \tau\nabla_{\mathcal{J}} L^i$$

where τ is the diminishing learning rate.

D EXPLAINING CAC ON REAL DATASETS

In Figure 5, we note that the Diabetes dataset has a low ICS(KM) value but ICS after clustering using CAC is very high. This gives a hint as to why CAC performs the best on Diabetes dataset.

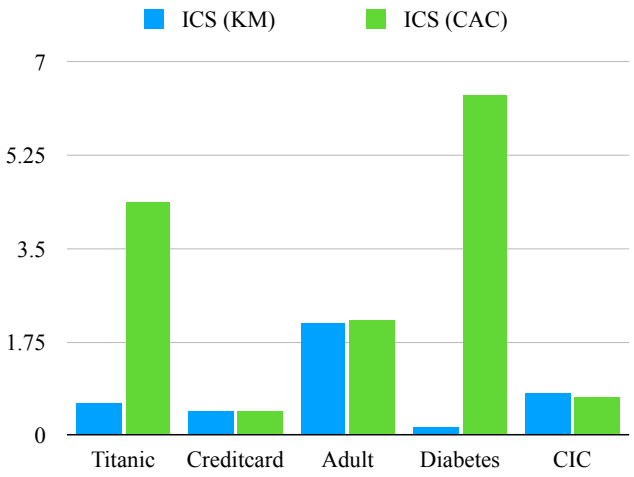


Figure 5: The total Inter Cluster Separation (ICS) for clusters found by k-means and CAC algorithms. α for CAC is set to the optimum values used by the CAC + LR variant.