Daniel González Muela

Comments:
Data is always scaled
CV is always 10 folds
Use of train-validation sets and test set
More Figures Available in code

Q2: MINST Dataset

a) Filtering and Dimensionality Reduction.

In this section we are dealing with a dataset of 50K images (28x28 features) that represent numbers from 0 to. As in the previous question, the first thing I did was to divide the data between an 80% training set and a 20% test set that won't be accessible until the end of the training process. The training dataset has around 10% samples of each class, so I will be using accuracy as the main evaluation metric. Also, it is large in p and very large in n, so some filtering and dimensionality reduction techniques can be very useful.

- Filtering: one of the first ideas was to randomly subsample a percentage of the dataset expect a probability convergence to the real distribution. However, I wanted to explore some new methods and I decided to use fast diversified sampling. This algorithm aims to sample for each class in a way that the values are uniformly distributed and represent the class range. For training I don't think it would be good to use this type of sampling as the distributions are transformed into a uniform one, but for visualization purposes it can help a lot. In order to test my hypothesis, I plotted the 2PC of Linear PCA as well as the scree plot for a random subsample and a diversified subsample of size 5000 [Figure 1]: one can see that both approaches produce very similar results but diversified subsampling is more reliable as there is no room for random oversampling of a certain type of features, therefore I proceed to try other reduction techniques using the diversified subsample.

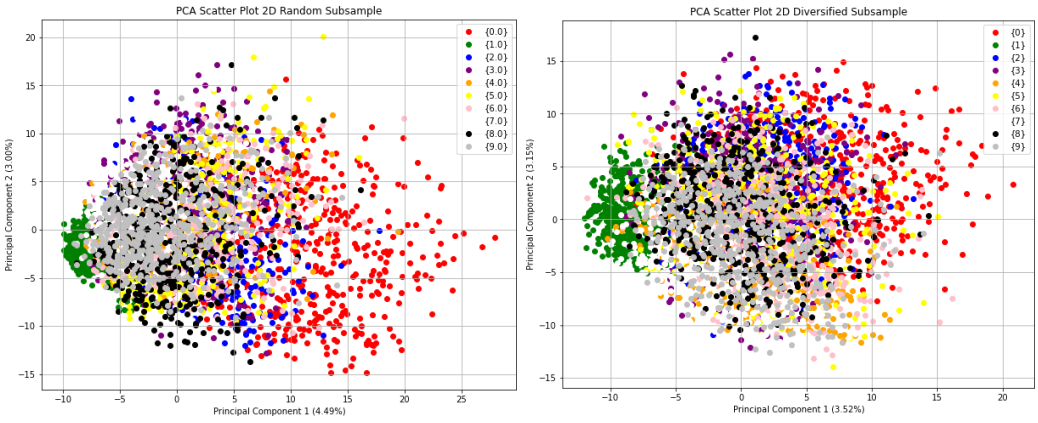

Figure 1: PCA for Random/Diversified Subsampling

- Dimensionality Reduction: In addition to the linear PCA, I also applied KPCA (rbf) and Truncated SVD, plotting the 3 PCs, the 2PCs and the scree plot. The PC plots were quite similar so I based my assessment on the scree plot decay [Figure 2]: the 3 methods indicate around 60 components, but KPCA and Truncated SVD are the ones with a fastest decay, being KPCA slightly better.

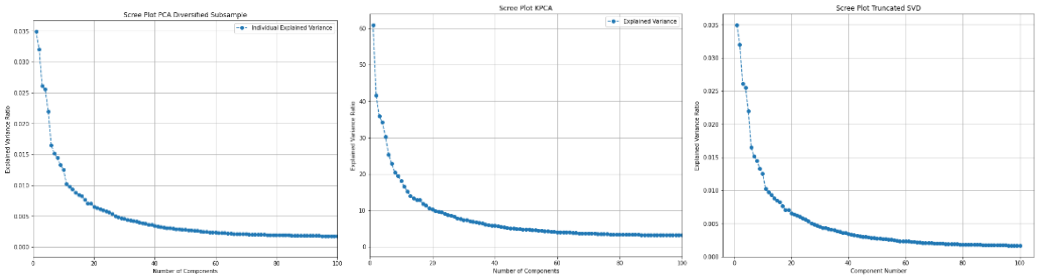

Figure 2: Scree Plot PCA/KPCA/Truncated SVD

I also tried Sparse PCA, which was computationally expensive, and NMF, that focuses on returning only 2 latent factors, but these methods were not as informative as the previous ones. Finally, I applied TSNE (for 3 different perplexities) and UMAP which were the best techniques in terms of visualization, being UMAP slightly better than TSNE. In conclusion, KPCA and UMAP are the dimensionality reduction techniques that seemed to work better [Figure 3]

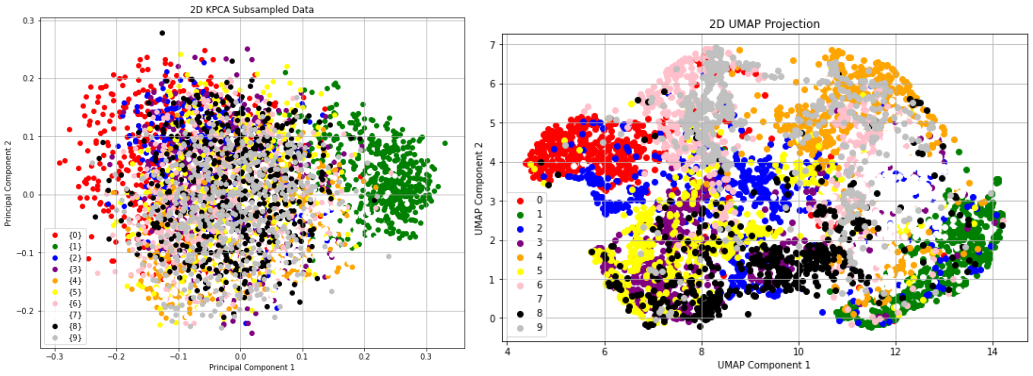

Figure 3: 2PCS of KPCA/UMAP Diversified Subsampling

It seems that classes 0 and 1 will be very easy to classify. However, the other classes are not that clear. There are some visible patters, especially in UMAP, but from the 2D plot we can expect to use non-linear methods and still not find a perfect classification among all the classes. For example, 6 and 9 seem to be very close to each other, which suggest very common features of even some mislabeling. Also some of the other numbers (like 2,8 or 3) seem to be all over the place, meaning that 2PCs are not able to capture most of the features.

b) Classifiers as a function of training size

For the next section we had to train different classifiers, and since diversified subsampling won't capture the original distributions and make everything uniform, I chose to implement m-out-of-n bootstrap as the subsampling method for this section. With a training data of 40,000, ideally one would sample around $\sqrt{40.000} = 200$ bootstrap samples with a size of 10.000 each, but due to computational resources I decided to work with 10 bootstrap samples of 1000 data points each. The first important decision was whether to use KPCA or UMAP (or Truncated SVD). For assessing that I created 3 Pipelines that consisted on scaling, dimensionality reduction to 60 components (as the scree plots suggested), and KNN with default parameters. I used cross validation for each of the 10 bootstrap samples and took the mean accuracy score. KPCA was the best performing method with and average cross validation score of 0.83, followed by

SVD with 0.81, and finally UMAP with 0.73. From these results I decided to choose KPCA with 60 components as the main reduction technique used to train the classifiers. Optimally, one would try with more methods instead of only KNN, but it is a good proxy of what one could start with.

I trained the same classifiers than in question 1 (as they implement a good variety of approaches) which in total are KNN, QDA, Linear + Kernel SVM, Logistic + Kernel Regression, RF and NN. The approach was the following: use GridSearchCV to tune some of the hyperparameters across the 10 bootstrap samples and use mean and standard deviation as the main indicators, which turned out to be very stable (except the depth parameter of RF). Finally, use the tunned models to iterate through the 10 bootstrap samples, but this time doing a train-validation split for different validation sizes [0.2…0.9]: fit the pipeline (scale + KPCA + Classifier) to the training set and get the accuracy score for both test and training. The final metric is the mean of the 10 accuracies as a function of the validation size [Figure 4].

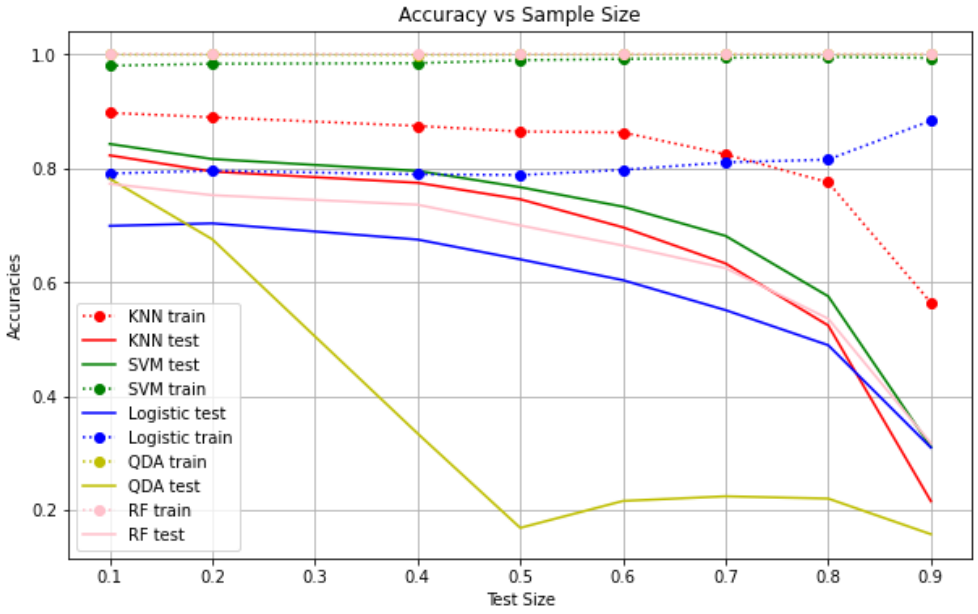

Figure 4: Mean Accuracy for Different Classifiers (10 bootstrap samples)

One can see that as the size of the validation set increases the train accuracies of all the models remain constant, except from KNN which makes sense as it is a data-based model (same problem in Q1). Regarding the validation accuracies they all decrease at the same rate except from QDA that completely breaks down as the training size might not be enough to properly define the gaussian distribution of the classes. For the other methods it is an expected result because they have to make more predictions with less data to train.

c) Clustering Methods

For the final section we want to explore clustering methods and compare them with the actual labels. I applied 3 different clustering methods (Kmeans, DBSCAN and Aglomerative Clustering) to the 60 components of KernelPCA. My objective wass to compare the optimal number of classes predicted by the algorithms vs the actual number of classes (9 numbers) and to see if the unsupervised classification has any correlation with the actual feature classification.

Kmeans

The elbow method didn't seem to have any significant jump as it had a continuous decrease, however, the silhouette method plot suggested somewhere around 5 to 8 would being 8 the optimal in this case [Figure 5]

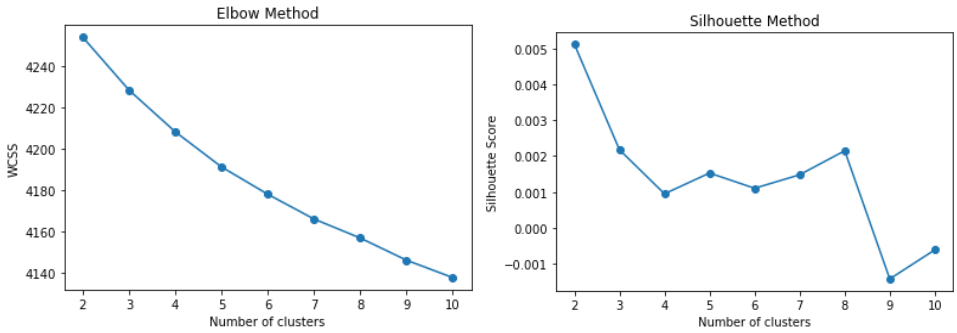

Figure 5: Elbow and Silhouette Method Kmeans

To study the correlation between the cluster and the actual labels I took the clusters produced with 8 labels and printed the correlation matrix with respect to the actual labels. Note that the predicted labels do not correspond to any number, so when the predicted label is "0" it could mean that the actual label predicted is any other number. Therefore, after seeing the confusion matrix I decided to change the predicted labels in a way that each predicted label corresponds to the majority of its true label: predicted 0 would become 7, 1 would remain 1, 2 also would predict 1, and so on. I called it Trasformed Confusion Matrix [Figure 7]

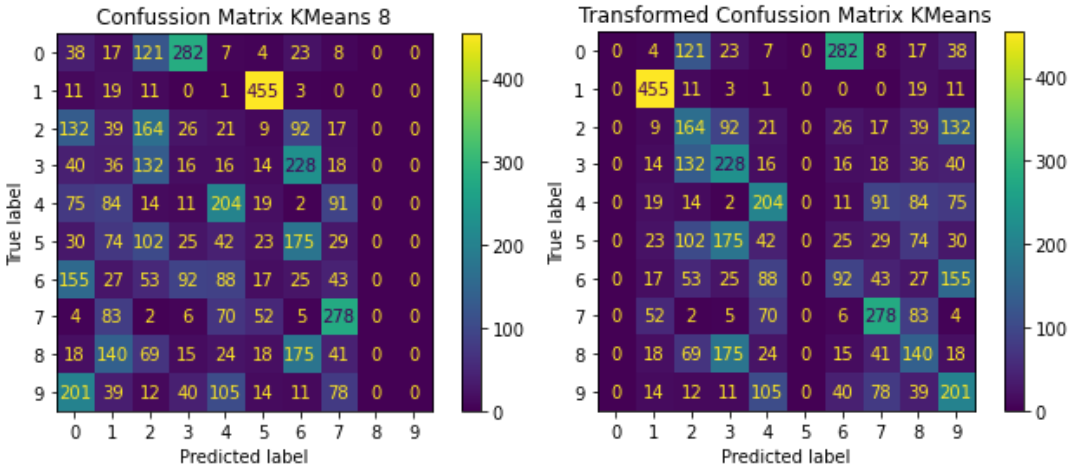

Figure 7: Confusion Matrix and Transformed Confusion Matrix KMeans

After applying this transformation, the comparison becomes much easier and one can see how there is a clear relation between the clusters and some of the labels. For example, the clusters belonging to 0 and 1 seem to capture most of the 0s and 1s. However, for other numbers like 2 and 3 the clusters don't work that well and it seems that the inner features are not related to the number itself. Finally, 4 and 7 seem to be clustered quite closed to each other, as well as 5/8 and 6/9, which were all classified as one cluster.

DBSCAN

I had problems with this algorithm which suggest that the varying density of the clusters could make the algorithm fail. I tried a wide combination of parameters: eps values from 0.1 to 10 and min samples from 1 to 10, as well as bigger cases like min samples in the range of 10 to 120, and higher esp values. However, the algorithm kept returning everything in the same class, except in the case where min samples was 1, as in that case the algorithm is allowed to create one class per data point. I also tried with the original dataset and different random samples (I originally applied to the KPCA 60 components reduced dataset), but the results were the same. [Figure 9]

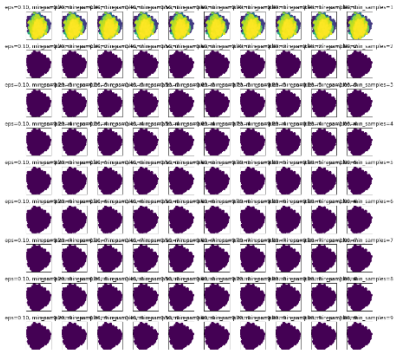

Figure 9: DBSCAN clustering result for different parameters

Agglomerative Clustering

In this hierarchical sampling approach the data points start in its own cluster and are merged into clusters at each step, meaning that dendrograms can be very useful. I applied the algorithm using different linkage methods (single, complete, average, ward), which are used to define a distance measure between the clusters. [Figure 10]

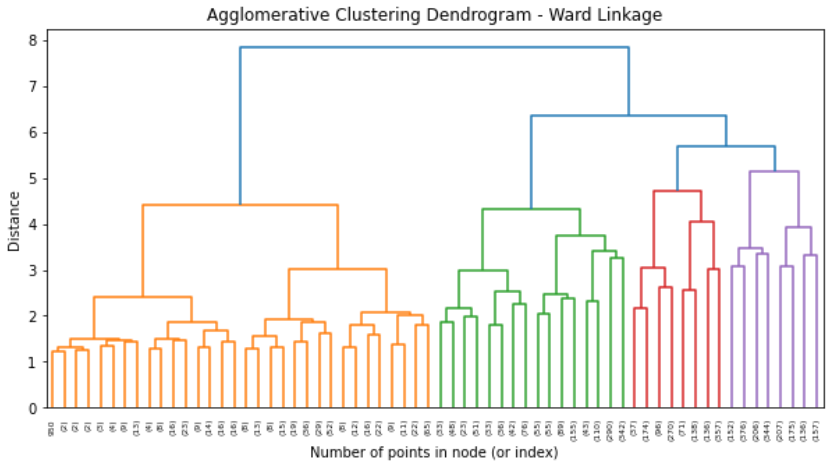

Figure 10: Dendogram of Ward Linkage

Single and average linkage had a very poor performance (for our purposes) as it clustered most the points in one class and a lot of single points as an individual class. On the other hand, complete linkage was providing to many clusters (over 40). But Ward Linkage had some promising results: at a distance of 4 it was predicting 8 clusters, therefore I decided to reapply the algorithm focusing on 8 clusters and plot the confusion matrix and adjust in the same way I did for Kmeans. [Figure 11]

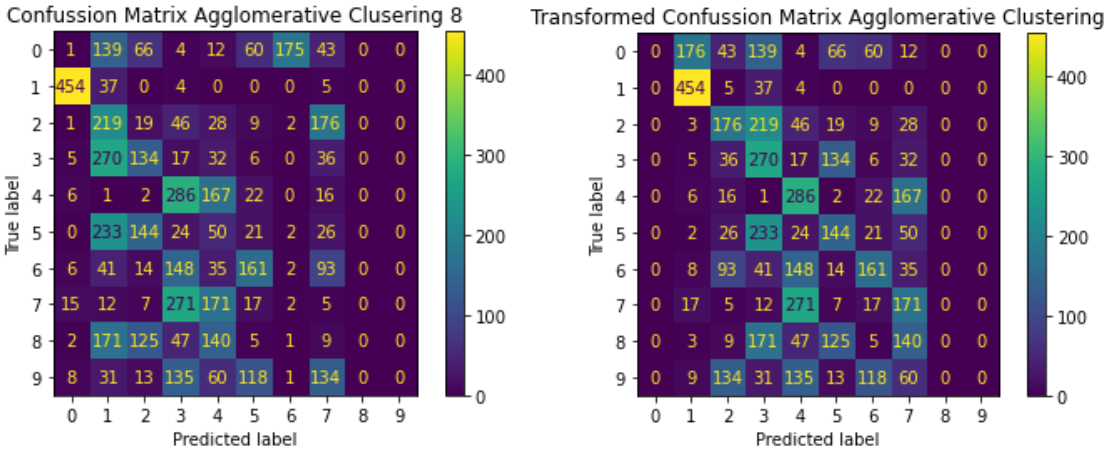

Figure 11: Confusion Matrix and Transformed Confusion Matrix Agglomerative Clust

To conclude, after transforming the confusion matrix, the results were not as good as for KMeans. It seemed to confuse a lot of the main features for classifying the samples as a certain number, meaning that this algorithm is taking into account other fundamental features of the dataset, rather than the actual number.