Prototype Selection for Nearest Neighbor Classifier

Zongze Liu

Abstract

In this report we experiment with K-means clustering prototype selection and condensed nearest neighbor algorithm for KNN classification on the MNIST dataset. We compare the results with KNN on random sub-samples of the full MNIST dataset.

1 Description

Given a fixed subsample size M, we create a balanced sub-training set: for each label i, we compute the ratio r_i of the label in the total training set and randomly sample $r_i * M$ many data points with label i and add them to the subsample training set, X_{sub}. Then we apply K-means clustering to the subsample set X_{sub} and use the centroids of K-means clustering as the prototypes for training the 1-nearest-neighbor algorithm. We also use condensed nearest neighbor algorithm to produce an alternative prototype subset U by scanning through X_{sub} and adding points x whose nearest neighbor in X_{sub} has a different label from its nearest neighbor in U. We use U as the prototype set for training the 1NN algorithm.

2 Pseudo Code for K-means Clustering Prototype Selection

Input: MNIST training set, subsample size M
output: K-means clustering prototypes

1. For each label i, compute the ratio r_i of training data with label i in the entire training set.

2. For each label i, generate randomly $r_i * M$ number of training data with label i and combine them to form a subsample training set X_{sub} with size M.

3. Apply K-means clustering to the subsample training set X_{sub} with number of centroids $= M/4$.

4. Use the centroids from K-means clustering as the prototype set and train 1-NN with the prototype set.

3 Pseudo Code for Condensed Nearest Neighbor Prototype Selection

We use the imblearn library for the implementation of condensed nearest neighbor prototype set.

Input: MNIST training set, subsample size M
output: condensed nearest neighbor prototypes

1. For each label i, compute the ratio r_i of training data with label i in the entire training set.

2. For each label i, generate randomly $r_i * M$ number of training data with label i and combine them to form a subsample training set X_{sub} with size M.

3. Build a prototype set U by scanning through X_{sub} and add data points x whose nearest neighbor in X_{sub} has different label from its nearest neighbor in U to U. Repeat until no more points can be added to U

4.Train 1-NN with the prototype set U.

4 Experiment results for K-means Clustering Prototype Selection

We compare the result of K-means clustering prototype with uniform random selection prototypes. We experiment with subsample sizes $M = 120, 200, 500, 1000, 5000, 10000$ and we run 15 iterations for each subsample size. We record the mean accuracy and standard deviation of each M in the table below.

M subsample size	**Random selection accuracy**	**Random selection std dev**	**K-means clustering prototype accuracy**	**K-means clustering prototype std dev**
120	67.81	2.2819	71.71	1.2301
200	73.96	1.8228	78.81	1.2538
500	81.03	0.5468	85.08	0.3205
1000	85.38	0.6350	89.65	0.1633
5000	93.34	0.2406	94.93	0.1693
10000	94.83	0.1702	94.898	0.067

5 Experiment results for Condensed Nearest Neighbor (CNN) Prototype Selection

We compare the result of CNN prototype with uniform random selection prototypes. We experiment with subsample sizes $M = 120, 200, 500, 1000, 5000, 10000$ and we run 15 iterations for each subsample size. In the experiments, the size of uniform random selections will be matched with the size of CNN prototype sets. We record the mean accuracy and standard deviation of each M in the table below.

M subsample size	**Random selection accuracy**	**Random selection std dev**	**CNN prototype accuracy**	**CNN prototype std dev**
120	58.91	2.9166	61.75	2.3898
200	66.08	2.897	68.15	1.6126
500	73.05	1.5434	74.01	0.3205
1000	79.11	0.8176	78.35	0.7109
5000	87.94	0.4985	84.6	1.0192
10000	90.576	0.4355	86.74	0.9543

6 Critical evaluation

1. We see that the K-means clustering prototype method outperforms uniform random selection prototype for smaller values of sample sizes $M = 120, 200, 500, 1000$ with significantly smaller standard deviations within the iterarions. The two methods perform comparably for large values of $M = 5000, 10000$. For the larger sample sizes, it could be that because MNIST is a balanced data set with a small number of outliers, large random sample sizes are a good approximation of the entire dataset and thus works well for 1-NN classification. For K-means clustering prototpes, we set the number of centroids to be a number porportional to the subsampled data set X_{sub}. For improvement, we could perform cross-validation on X_{sub} and search for the optimal number of centroids via the elbow method. We could also try to use binary search to find the optimal number of centroids.

2. The condensed nearest neighbor prototype method outperforms the random selection methods for small values of $M = 100, 200$. This could be because condensed neareast neighbor works best when under-sampling imbalanced data sets whereas MNIST is balanced and has fewer outliers. Therefore the performance condensed nearest neighbor prototypes is not ideal compared to uniform random subsampling.