CSE 251A - ML: Learning Algorithms
Project 1

Cindy Chen

Abstract

This paper provides a report on using three methods to select training datasets that perform better than randomly selecting training data.

1. Prototype selection

Three prototype selections perform better than randomly selecting M numbers of data from the dataset. We test the performance by doing one-nearest-neighbor classification.

1.1. Using K-Means classification on buckets of data

Since we want a better performance than random selecting data to train. I suppose having the raw data first put in buckets by the same ground truth y. Then the K-Means of the buckets would probably have a better pattern of their x-values. This might result in a better prediction of y using this K-Mean method with buckets. We will test the accuracy of this method with one-nearest neighbor on the testing set.

1.2. Using K-Means classification with majority vote

Given that the same ground truth class 'y' may exhibit a pattern of similar 'x' values, we can employ a majority vote to refine the cluster 'y' values and throw out the x-values that do not have the majority y label. In this approach, each cluster's 'y' class is determined by selecting the one with the maximum count within the cluster. This methodology aims to yield more purified training data. We will test the accuracy of this method with one-nearest neighbor on the testing set.

1.3. Using different random seeds on K-Means classification

Since the algorithm of K-Means will randomly select data for its cluster. If we change the random seed value and do K-Means multiple times. Getting multiple result clusters, and putting them back to do another K-Means may result in a better performance. We will test the accuracy of this method with one-nearest neighbor on the testing set.

2. Pseudocode

Algorithm 1 Kmeans with 10 buckets

```
Input: data x_train, y_train, M
x_selectedDataSet = [], y_selectedDataSet = []
for i = 0 to 9 do
    if y_train_i == i then
        save x_train_i to bucket[i]
    end if
end for
for i = 0 to 9 do
    do kmeans on buckets[i] with M/10 clusters
    x_selectedDataSet += cluster_center
    y_selectedDataSet += i * # of cluster_center
end for
Output: x_selectedDataSet, y_selectedDataSet
```

Algorithm 2 Kmeans with majority vote

```
Input: data x_train, y_train,M
x_selectedDataSet = [], y_selectedDataSet = []
Run kmeans on x_train with M number of clusters
for each cluster in M do
    Find the class of y that has the maximum count being
    y_max_count
    x_cluster = all the x_train that has y value of
    y_max_count
    let y_selectedDataSet = y_max_count
    x_selectedDataSet += the mean value of the
    x_cluster
end for
Output: x_selectedDataSet, y_selectedDataSet
```

3. Experimental results

Table 1,2 and 3 are the experimental results of my three algorithms

4. Critical evaluation

As we see from the table of experimental results. Using one nearest neighbor to test the accuracy. All the algorithms for selecting training datasets have better performance than

Algorithm 3 Kmeans 3 times random seeds with 10 buckets

Input: data M, x_train
$x\_selectedDataSet$ = [], $y\_selectedDataSet$ = []
for random_seed_i where i is 1 to 3 **do**
  for $j = 0$ **to** 9 **do**
    do kmeans on $buckets[j]$ with $M/10$ clusters with random_seed_i
    $cluster\_centers\_buckets$ += kmeans $cluster\_center$.
  end for
  $cluster\_centers\_total$ += $cluster\_centers\_buckets$
end for
Do kmeans on the combined cluster_center of random seed
$x\_selectedDataSet$ += $final\_kmeans\_cluster\_centers$
$y\_selectedDataSet$ += $cluster_{yv}alue$
Output: $x\_selectedDataSet$, $y\_selectedDataSet$

Table 1. Training dataset selection using Algorithm 1 - K-Means with buckets.

Data set	M	Accuracy	Better?
Baseline 1NN	10000	95.14± 0.5	
Baseline 1NN	5000	93.75± 0.5	
Baseline 1NN	1000	88.33± 0.5	
Algorithm 1	10000	97.03	√
Algorithm 1	5000	96.94	√
Algorithm 1	1000	95.79	√

Table 2. Training dataset selection using Algorithm 2 - K-Means with Majority vote.

Data set	M	Accuracy	Better?
Baseline 1NN	10000	95.14± 0.5	
Baseline 1NN	5000	93.75± 0.5	
Baseline 1NN	1000	88.33± 0.5	
Algorithm 2	10000	96.72	√
Algorithm 2	5000	96.22	√
Algorithm 2	1000	95.12	√

Table 3. Training dataset selection using Algorithm 3 - K-Means with different random seed values.

Data set	M	Accuracy	Better?
Baseline 1NN	10000	95.14± 0.5	
Baseline 1NN	5000	93.75± 0.5	
Baseline 1NN	1000	88.33± 0.5	
Algorithm 3	10000	97.19± 0.2	√
Algorithm 3	5000	96.77± 0.4	√
Algorithm 3	1000	95.88± 0.7	√

those using randomly selected datasets(baseline).
When the number of M increases the performance of accuracy tends to increase. The best case of using M=10000 is an accuracy of 97.03 using algorithm 1- the K-mean bucket classification method.

In [5]:

```
import numpy as np
from keras.datasets import mnist
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.cluster import KMeans

(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [6]:

```
M = 10000
```

In [7]:

```
random_indices = np.random.choice(len(X_train), size=M, replace=False)

#selecting M=10000 randomly
selected_M_x = X_train[random_indices]
selected_M_y = y_train[random_indices]

#1nn classifier from sklearn
knn_classifier = KNeighborsClassifier(n_neighbors=1)

reshaped_X_train= X_train.reshape(60000,-1)
reshaped_X_test = X_test.reshape(10000,-1)

reshaped_M_x= selected_M_x.reshape(M,-1)

# Train the classifier on the training data
knn_classifier.fit(reshaped_M_x, selected_M_y)

# Make predictions on the test data
y_pred = knn_classifier.predict(reshaped_X_test)

# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("1NN random Accuracy:", accuracy)
```

```
1NN random Accuracy: 0.9468
```

In [8]:

```
#Kmeans using 10 buckets method

# Create a dictionary to hold the buckets
buckets = {}
cluster_centers = []

kmeans_x_train = []
kmeans_y_train = []

# Iterate through each label and populate the corresponding bucket
for label in range(10):  # Assuming labels range from 0 to 9
    bucket_indices = np.where(y_train == label)[0]
    buckets[label] = reshaped_X_train[bucket_indices]

# Print the shapes of the resulting buckets
for label, bucket in buckets.items():
    print(f"Bucket for label {label}, shape: {bucket.shape}")

# Perform k-means clustering multiple times on different subsets of the dataset
k=M//10

for i in range(10):
    # Create a KMeans instance
```

```
    kmeans = KMeans(n_clusters= k, random_state=42,n_init="auto")

    # Fit the model to the subset of the data
    kmeans.fit(buckets[i])

    # Get cluster labels and cluster centers
    cluster_centers.append(kmeans.cluster_centers_)

kmeans_x_train = np.concatenate(cluster_centers, axis=0)
for i in range(10):
    kmeans_y_train.append([i] * cluster_centers[i].shape[0])
kmeans_y_train = np.concatenate(kmeans_y_train)
print(kmeans_y_train[0])
```

```
Bucket for label 0, shape: (5923, 784)
Bucket for label 1, shape: (6742, 784)
Bucket for label 2, shape: (5958, 784)
Bucket for label 3, shape: (6131, 784)
Bucket for label 4, shape: (5842, 784)
Bucket for label 5, shape: (5421, 784)
Bucket for label 6, shape: (5918, 784)
Bucket for label 7, shape: (6265, 784)
Bucket for label 8, shape: (5851, 784)
Bucket for label 9, shape: (5949, 784)
0
```

In [9]:

```
kmeans_buckets_1nn = KNeighborsClassifier(n_neighbors=1)
kmeans_buckets_1nn.fit(kmeans_x_train, kmeans_y_train)

y_pred = kmeans_buckets_1nn.predict(reshaped_X_test)
kmeans_buckets_accuracy = accuracy_score(y_test, y_pred)
print("Kmeans with buckets accuracy_1NN: ",kmeans_buckets_accuracy)
```

```
Kmeans with buckets accuracy_1NN:  0.9703
```

In [ ]:

In [11]:

```
kmeans = KMeans(n_clusters=M, random_state=42,n_init="auto")

# Fit all data this time
kmeans.fit(reshaped_X_train)

# Get the data in each group
groups = [[] for _ in range(M)]
majority_x_train = []
majority_y_train = []

# data index -> group index
# print(len(kmeans.labels_))

# group index -> data index
for i in range(60000):
    groups[kmeans.labels_[i]].append(i)
```

In [12]:

```
# Input: group contains indices of training data
def majority_vote(group):
    count_label = [0 for _ in range(10)]
    group_label = [[] for _ in range(10)]
    for data_index in group:
        l = y_train[data_index]
        count_label[l] += 1
```

```
        group_label[1].append(data_index)
    majority_label = count_label.index(max(count_label))
    # only find centroid of the majority
    majority_index = group_label[majority_label]
    centroid = np.mean(reshaped_X_train[majority_index], axis=0)
    return centroid, majority_label
```

In [13]:

```
for group in groups:
    centroid, majority_label = majority_vote(group)
    majority_x_train.append(centroid)
    majority_y_train.append(majority_label)

majority_x_train = np.array(majority_x_train)
majority_y_train = np.array(majority_y_train)
```

In [14]:

```
kmeans_majority_1nn = KNeighborsClassifier(n_neighbors=1)
kmeans_majority_1nn.fit(majority_x_train, majority_y_train)

y_pred = kmeans_majority_1nn.predict(reshaped_X_test)
kmeans_majority_accuracy = accuracy_score(y_test, y_pred)
print("Kmeans with majority vote accuracy_1NN: ",kmeans_majority_accuracy)
```

```
Kmeans with majority vote accuracy_1NN:  0.9672
```

In [13…

```
# KMeans is random
# Average over different random seeds
```

In [28]:

```
k = M//10
cluster_centers_0 = []
kmeans_y_train_0 = []
for i in range(10):
    # Create a KMeans instance
    kmeans_random_seeds_0 = KMeans(n_clusters= k, random_state=0,n_init="auto")

    # Fit the model to the subset of the data
    kmeans_random_seeds_0.fit(buckets[i])

    # Get cluster labels and cluster centers
    cluster_centers_0.append(kmeans_random_seeds_0.cluster_centers_)
cluster_centers_0 = np.concatenate(cluster_centers_0, axis=0)

for i in range(10):
    kmeans_y_train_0.append([i] * cluster_centers_0[i].shape[0])
kmeans_y_train_0 = np.concatenate(kmeans_y_train_0)
```

In [31]:

```
len(kmeans_y_train_0)
```

Out[31]: 7840

In [16]:

```
cluster_centers_42 = []
kmeans_y_train_42 = []
for i in range(10):
    # Create a KMeans instance
    kmeans_random_seeds_42 = KMeans(n_clusters= k, random_state=42, n_init="auto")

    # Fit the model to the subset of the data
    kmeans_random_seeds_42.fit(buckets[i])
```

```
    # Get cluster labels and cluster centers
    cluster_centers_42.append(kmeans_random_seeds_42.cluster_centers_)

cluster_centers_42 = np.concatenate(cluster_centers_42, axis=0)

for i in range(10):
    kmeans_y_train_42.append([i] * cluster_centers_42[i].shape[0])

kmeans_y_train_42 = np.concatenate(kmeans_y_train_42)
```

In [30]:

```
len(kmeans_y_train_42)
```

Out[30]: 7840

In [17]:

```
cluster_centers_88 = []
kmeans_y_train_88 = []

for i in range(10):
    # Create a KMeans instance
    kmeans_random_seeds_88 = KMeans(n_clusters= k, random_state=88, n_init="auto")

    # Fit the model to the subset of the data
    kmeans_random_seeds_88.fit(buckets[i])

    # Get cluster labels and cluster centers
    cluster_centers_88.append(kmeans_random_seeds_88.cluster_centers_)

cluster_centers_88 = np.concatenate(cluster_centers_88, axis=0)

for i in range(10):
    kmeans_y_train_88.append([i] * cluster_centers_88[i].shape[0])

kmeans_y_train_88 = np.concatenate(kmeans_y_train_88)
```

In [29]:

```
len(kmeans_y_train_88)
```

Out[29]: 7840

In [18]:

```
x_train_randomseed = []
x_train_randomseed.append(cluster_centers_0)
x_train_randomseed.append(cluster_centers_42)
x_train_randomseed.append(cluster_centers_88)

x_train_randomseed = np.concatenate(x_train_randomseed, axis=0)

y_train_randomseed = []
y_train_randomseed.append(kmeans_y_train_0)
y_train_randomseed.append(kmeans_y_train_42)
y_train_randomseed.append(kmeans_y_train_88)

y_train_randomseed = np.concatenate(y_train_randomseed)

cluster_centers_total = []


# Create a KMeans instance
kmeans_random_seeds = KMeans(n_clusters= M, random_state=30, n_init="auto")
```

```python
# Fit the model to the subset of the data
kmeans_random_seeds.fit(x_train_randomseed)

cluster_centers_total.append(kmeans_random_seeds.cluster_centers_)

cluster_centers_total = np.concatenate(cluster_centers_total, axis=0)
```

In [25]:

```python
x_train_randomseed.shape
```

Out[25]: (30000, 784)

In [19]:

```python
cluster_centers_total.shape
```

Out[19]: (10000, 784)

In [20]:

```python
kmeans_random_seeds.labels_.shape
```

Out[20]: (30000,)

In [32]:

```python
kmeans_y_train_0, kmeans_y_train_42, kmeans_y_train_88 = [], [], []
for i in range(10):
    kmeans_y_train_0.append([i] * 1000)
kmeans_y_train_0 = np.concatenate(kmeans_y_train_0)

for i in range(10):
    kmeans_y_train_42.append([i] * 1000)
kmeans_y_train_42 = np.concatenate(kmeans_y_train_42)

for i in range(10):
    kmeans_y_train_88.append([i] * 1000)
kmeans_y_train_88 = np.concatenate(kmeans_y_train_88)
```

In [33]:

```python
y_train_randomseed = []
y_train_randomseed.append(kmeans_y_train_0)
y_train_randomseed.append(kmeans_y_train_42)
y_train_randomseed.append(kmeans_y_train_88)

y_train_randomseed = np.concatenate(y_train_randomseed)
```

In [34]:

```python
# Do majority vote again

# Get the data in each group
groups = [[] for _ in range(M)]
majority_x_train = []
majority_y_train = []

# data index -> group index
# print(len(kmeans.labels_))

# group index -> data index
for i in range(30000):
    groups[kmeans_random_seeds.labels_[i]].append(i)
```

In [35]:

```python
# Input: group contains indices of training data
def majority_vote2(group):
```

```python
    count_label = [0 for _ in range(10)]
    group_label = [[] for _ in range(10)]
    for data_index in group:
        l = y_train_randomseed[data_index]
        count_label[l] += 1
        group_label[l].append(data_index)
    majority_label = count_label.index(max(count_label))
    # only find centroid of the majority
    majority_index = group_label[majority_label]
    centroid = np.mean(x_train_randomseed[majority_index], axis=0)
    return centroid, majority_label
```

In [36]:

```python
for group in groups:
    centroid, majority_label = majority_vote2(group)
    majority_x_train.append(centroid)
    majority_y_train.append(majority_label)

majority_x_train = np.array(majority_x_train)
majority_y_train = np.array(majority_y_train)
```

In [37]:

```python
kmeans_majority_1nn = KNeighborsClassifier(n_neighbors=1)
kmeans_majority_1nn.fit(majority_x_train, majority_y_train)

y_pred = kmeans_majority_1nn.predict(reshaped_X_test)
kmeans_randomseeds_accuracy = accuracy_score(y_test, y_pred)
print("Kmeans with random seeds combined accuracy_1NN: ",kmeans_randomseeds_accuracy)
```

```
Kmeans with random seeds combined accuracy_1NN:  0.9719
```

In [ ]: