```
#!/usr/bin/env python
# coding: utf-8

# In[20]:


import numpy as np
import matplotlib.pyplot as plt
import struct
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier, NearestNeighbors
from sklearn.metrics import accuracy_score
import time
import matplotlib.pyplot as plt
from scipy.stats import norm


# In[2]:


def read_file(filename):
    with open(filename, 'rb') as f:
        zero, data_type, dims = struct.unpack('>HBB', f.read(4))
        shape = tuple(struct.unpack('>I', f.read(4))[0] for d in range(dims))
        return np.frombuffer(f.read(), dtype=np.uint8).reshape(shape)


# In[107]:


# Load Dataset
train_images_file = 'archive/train-images.idx3-ubyte'
train_labels_file = 'archive/train-labels.idx1-ubyte'
test_images_file = 'archive/t10k-images.idx3-ubyte'
test_labels_file = 'archive/t10k-labels.idx1-ubyte'

train_images = read_file(train_images_file)
train_labels = read_file(train_labels_file)
test_images = read_file(test_images_file)
test_labels = read_file(test_labels_file)


# In[44]:


# Random Prototype Selection
def Random_prototypes(X, y, M):
    indices = np.random.choice(len(X), size=M, replace=False)
    return X[indices], y[indices]


# In[85]:


# K means Prototype Selection
def Kmeans_prototypes(X, n_clusters):
    kmeans = KMeans(n_clusters=n_clusters, random_state=None, n_init='auto').fit(X)
    prototypes = kmeans.cluster_centers_
    return prototypes

def Assign_centroids(X, y, centroids):
    kmeans = KMeans(n_clusters=len(centroids), init=centroids, n_init=1, max_iter=1)
    kmeans.fit(X)
    labels = np.zeros(len(centroids))

    for i in range(len(centroids)):
        labels[i] = np.argmax(np.bincount(y[kmeans.labels_ == i]))

    return labels
```

```
# In[91]:


# CNN Prototype Selection
def CNN_prototypes(X, y, M):
    indices = np.random.permutation(len(X))
    X_shuffled = X[indices]
    y_shuffled = y[indices]
    prototypes = [X_shuffled[0]]
    labels = [y_shuffled[0]]
    nn = NearestNeighbors(n_neighbors=1)

    for i in range(1, M):
        nn.fit(prototypes, labels)
        nearest = nn.kneighbors([X_shuffled[i]], 1, return_distance=False)
        nearest_label = labels[nearest[0][0]]
        if y_shuffled[i] != nearest_label:
            prototypes.append(X_shuffled[i])
            labels.append(y_shuffled[i])

    return np.array(prototypes), np.array(labels)


# In[92]:


# Evaluate Test Error
def Evaluate_test_error(prototypes, prototype_labels, X_test, y_test):
    knn = KNeighborsClassifier()
    knn.fit(prototypes, prototype_labels)
    y_pred = knn.predict(X_test)
    test_error = 1 - accuracy_score(y_test, y_pred)
    return test_error


# In[96]:


prototype_sizes = [100, 500, 1000, 5000, 10000]
methods = ['Random', 'K-Means', 'CNN']
results = {}
runtime_results = {}
num_experiments = 5
error_results = {method: {size: [] for size in prototype_sizes} for method in methods}


# In[97]:


for size in prototype_sizes:
    for method in methods:
        for e in range(num_experiments):
            start_time = time.time()
            if method == 'Random':
                prototypes, labels = Random_prototypes(train_images.reshape(-1, 784),
train_labels, size)
            elif method == 'K-Means':
                prototypes = Kmeans_prototypes(train_images.reshape(-1, 784), size)
                labels = Assign_centroids(train_images.reshape(-1, 784), train_labels,
prototypes)
            elif method == 'CNN':
                prototypes, labels = CNN_prototypes(train_images.reshape(-1, 784),
train_labels, size)

            test_error = Evaluate_test_error(prototypes, labels, test_images.reshape(-1,
784), test_labels)
            error_results[method][size].append(test_error)
```

```
            # results[(method, size)] = test_error
            end_time = time.time()
            runtime = end_time - start_time
            runtime_results[(method, size)] = runtime
            print(e,method,size,test_error)

avg_errors = {method: [] for method in methods}
std_errors = {method: [] for method in methods}
for method in methods:
    for size in prototype_sizes:
        m = np.mean(error_results[method][size])
        s = np.std(error_results[method][size])
        avg_errors[method].append(m)
        std_errors[method].append(s)
        results[method, size] = m
results


# In[108]:


std_errors


# In[105]:


# Plot Test Error with Confidence Intervals
confidence = 0.95
z_score = norm.ppf(1 - (1 - confidence) / 2)  # Z-score for 95% confidence

ci_errors = {method: [] for method in methods}

for method in methods:
    for size in prototype_sizes:
        sample_mean = np.mean(error_results[method][size])
        sample_std = np.std(error_results[method][size])
        margin_error = z_score * (sample_std / np.sqrt(num_experiments))
        ci_errors[method].append(margin_error)
for method in methods:
    avg_error = [np.mean(error_results[method][size]) for size in prototype_sizes]
    ci_error = ci_errors[method]
    plt.errorbar(prototype_sizes, avg_error, yerr=ci_error, label=method, capsize=5)

plt.xlabel('Prototype Size')
plt.ylabel('Average Test Error')
plt.title('Average Test Error with 95% Confidence Intervals vs Prototype Size')
plt.legend()
plt.grid(True)

plt.savefig('AverageTestError95ConfidenceIntervals.png', format='png', dpi=300)
plt.show()


# In[106]:


# Plot Test Error
plt.figure(figsize=(10, 6))

for method in methods:
    sizes = []
    errors = []
    for size in prototype_sizes:
        sizes.append(size)
        errors.append(results[(method, size)])
    plt.plot(sizes, errors, marker='o', label=method)

plt.xlabel('Prototype Size')
```

```
plt.ylabel('Test Error')
plt.title('Test Error vs Prototype Size')
plt.legend()
plt.grid(True)
plt.savefig('TestError.png', format='png', dpi=300)
plt.show()

# Plot Runtime
plt.figure(figsize=(10, 6))

for method in methods:
    sizes = []
    runtimes = []
    for size in prototype_sizes:
        sizes.append(size)
        runtimes.append(runtime_results[(method, size)])
    plt.plot(sizes, runtimes, marker='o', label=method)

plt.xlabel('Prototype Size')
plt.ylabel('Runtime (seconds)')
plt.title('Runtime vs Prototype Size')
plt.yscale('log')
plt.ylim(ymin=0.1)
plt.legend()
plt.grid(True)
plt.savefig('Runtime.png', format='png', dpi=300)
plt.show()
```