

assignment2_copy

March 8, 2024

Assignment 2

This assignment requires you to implement image recognition methods. Please understand and use relevant libraries. You are expected to solve both questions.

Data preparation and rules

Please use the images of the MNIST hand-written digits recognition dataset. You may use torchvision.datasets library to obtain the images and splits. You should have 60,000 training images and 10,000 test images. Use test images only to evaluate your model performance.

```
[ ]: import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
```

```
[ ]: from tensorflow.keras.datasets import mnist
import cv2
import numpy as np

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

def preprocess_images(images):
    images = images.astype('float32') / 255.0
    return images

train_images = preprocess_images(train_images)
test_images = preprocess_images(test_images)
print(train_images.shape)
print(test_images.shape)
```

```
2024-03-08 21:36:17.593891: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.1 SSE4.2 AVX AVX2 AVX_VNNI FMA
```

To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2024-03-08 21:36:17.695047: I tensorflow/core/util/port.cc:104] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.

(60000, 28, 28)

(10000, 28, 28)

Q1: SIFT-BoVW-SVM [4 points]

1. [2 points] Implement the SIFT detector and descriptor. Compute cluster centers for the Bag-of-Visual-Words approach. Represent the images as histograms (of visual words) and train a linear SVM model for 10-way classification. Note 1: You may want to use libraries such as cv2 (OpenCV) and sklearn (Sci-kit learn) for doing this question. <https://scikit-learn.org/stable/modules/svm.html#multi-class-classification> may be useful for the SVM. Note 2: Seed random numbers for reproducibility (running the notebook again should give you the same results!).

```
[ ]: from tensorflow.keras.datasets import mnist
import cv2
import numpy as np
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

def preprocess_images(images):
    images = images.astype('float32') / 255.0
    return images

train_images = preprocess_images(train_images)
test_images = preprocess_images(test_images)

def calc_features(images, thresh):
    sift = cv2.SIFT_create(thresh)
    features = []
    for img in images:
        img = np.uint8(img * 255) # Convert back to OpenCV usable format
        _, des = sift.detectAndCompute(img, None)
        if des is not None:
            features.append(des)
    return np.vstack(features) if features else np.empty((0, 128)) # Assuming
↪ SIFT descriptors have a length of 128
```

```

def perform_kmeans(features, k):
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
    _, _, centers = cv2.kmeans(features, k, None, criteria, 10, cv2.
↳KMEANS_RANDOM_CENTERS)
    return centers

def bag_of_features(features, centers, k):
    vec = np.zeros((1, k), dtype=np.float32)
    for i in range(features.shape[0]):
        diff = np.linalg.norm(np.tile(features[i], (k, 1)) - centers, axis=1)
        idx = np.argmin(diff)
        vec[0, idx] += 1
    return vec

def train_and_evaluate(train_images, train_labels, test_images, test_labels,
↳thresh, k):
    features = calc_features(train_images, thresh)
    centers = perform_kmeans(features, k)

    def create_feature_vec(img):
        des = calc_features([img], thresh)
        if des.size > 0: # Changed from None check to size check
            return bag_of_features(des, centers, k).flatten()
        else:
            return np.zeros((k,)) # Return a zero vector if no features are
↳detected

    # Convert training and testing images to feature vectors
    train_vec = np.array([create_feature_vec(img).flatten() for img in
↳train_images])
    test_vec = np.array([create_feature_vec(img).flatten() for img in
↳test_images])

    # Check the shape of the feature vectors
    assert train_vec.ndim == 2, "train_vec is not a 2D array"
    assert test_vec.ndim == 2, "test_vec is not a 2D array"

    # Train SVM
    clf = SVC(kernel='linear', probability=True)
    clf.fit(train_vec, train_labels)

    # Evaluate
    preds = clf.predict(test_vec)
    return accuracy_score(test_labels, preds), confusion_matrix(test_labels,
↳preds)

```

```

thresh = 10 # SIFT feature threshold
k = 150 # Number of clusters for KMeans
accuracy, conf_mat = train_and_evaluate(train_images, train_labels,
    ↪test_images, test_labels, thresh, k)

print(f'Accuracy: {accuracy*100:.2f}%')
print('Confusion Matrix:')
print(conf_mat)

```

Accuracy: 74.66%

Confusion Matrix:

```

[[ 858   6  16   2   1  21  51   8  12   5]
 [   1 1108   2   1   3   0   3  15   1   1]
 [  53  26 654  34  15  31  30 154  28   7]
 [   7   6  76 769  14  66  23  25  19   5]
 [   3  17  25  14 774  17  25  29  20  58]
 [  48  12  27  57  19 575  73  37  11  33]
 [  59  21  33  11   7  49 572  59  20 127]
 [  13  72  95  18  24  18  30 743   5  10]
 [  20   1  40  38  32  22  44  10 725  42]
 [  23  11  14   9  38  44 137  20  25 688]]

```

2. [1 point] Keeping everything else constant, plot how classification accuracy changes as you sweep across 6 different values for the number of clusters. Please decide what numbers are meaningful for this question. Explain the trends in classification accuracy that you observe. Note 1: It is recommended to try hyperparameters in logarithmic steps such as 2x or 3x multiples. An example of 2x multiples is: 1, 2, 5, 10, 20, ... An example of 3x multiples is: 1, 3, 10, 30, 100, ...

```

[ ]: cluster_counts = [10, 30, 100, 300]
    # cluster_counts = [10, 30, 100, 300, 1000, 3000]

    accuracies = []

    for k in cluster_counts:
        print(f"Evaluating for k = {k}")
        accuracy, _ = train_and_evaluate(train_images, train_labels, test_images,
    ↪test_labels, thresh, k)
        accuracies.append(accuracy)
        print(f"Accuracy for k = {k}: {accuracy*100:.2f}%")

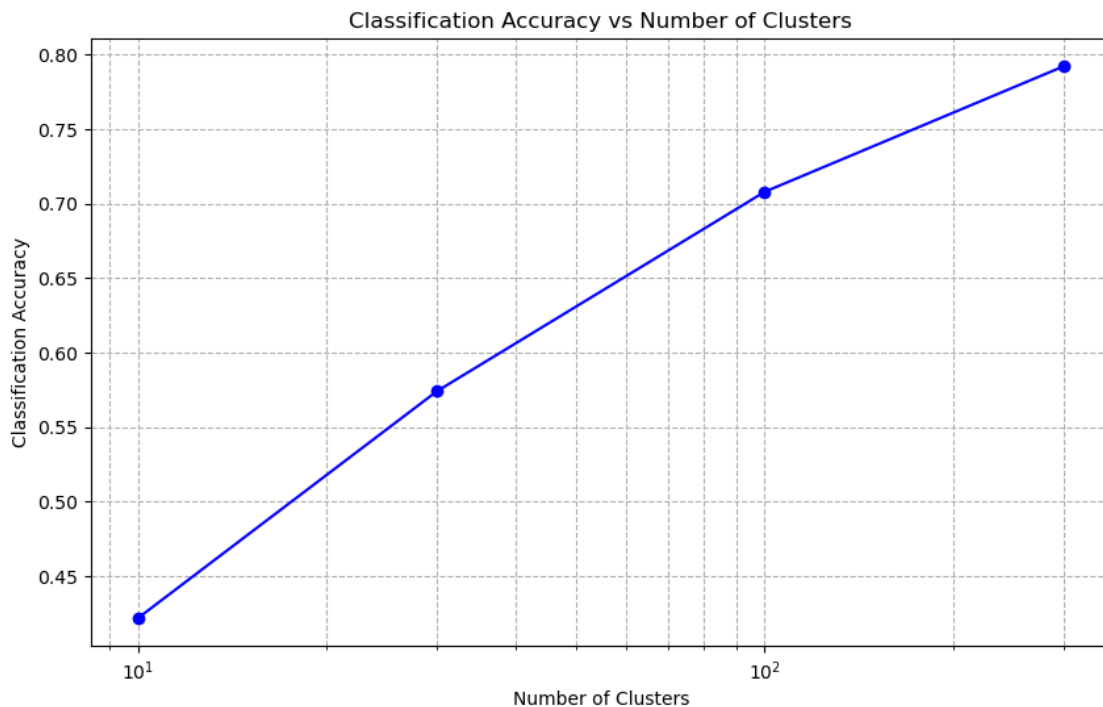
    import matplotlib.pyplot as plt

    plt.figure(figsize=(10, 6))
    plt.plot(cluster_counts, accuracies, marker='o', linestyle='--', color='b')
    plt.title("Classification Accuracy vs Number of Clusters")
    plt.xlabel("Number of Clusters")
    plt.ylabel("Classification Accuracy")

```

```
plt.xscale("log") # Because we're using a logarithmic scale for cluster counts
plt.grid(True, which="both", ls="--")
plt.show()
```

Evaluating for k = 10
 Accuracy for k = 10: 42.17%
 Evaluating for k = 30
 Accuracy for k = 30: 57.41%
 Evaluating for k = 100
 Accuracy for k = 100: 70.80%
 Evaluating for k = 300
 Accuracy for k = 300: 79.23%



- [1 point] Show the results for 6 different hyperparameter settings. You may play with the SIFT detector or descriptor and the linear SVM. Keep the number of clusters constant based on the answer to the previous question. Explain the trends in classification accuracy that you observe.

```
[ ]: # Define hyperparameters settings for SIFT and SVM
hyperparameter_settings = [
    {'sift_thresh': 10, 'svm_c': 0.1},
    {'sift_thresh': 15, 'svm_c': 1},
    {'sift_thresh': 20, 'svm_c': 10},
    {'sift_thresh': 25, 'svm_c': 1, 'svm_gamma': 'scale'},
    {'sift_thresh': 30, 'svm_c': 0.01},
```

```

        {'sift_thresh': 35, 'svm_c': 100},
    ]

    # Store the results
    results_table = []

    # Iterate over the hyperparameter settings
    for setting in hyperparameter_settings:
        # Extract SIFT features using the specified threshold
        train_features = calc_features(train_images, setting['sift_thresh'])
        test_features = calc_features(test_images, setting['sift_thresh'])

        # Perform KMeans clustering and create bag of features representations
        k = 150 # Number of clusters kept constant
        centers = perform_kmeans(train_features, k)
        train_bow = np.array([bag_of_features(feats, centers, k) for feats in
↪train_features])
        test_bow = np.array([bag_of_features(feats, centers, k) for feats in
↪test_features])

        # Train the SVM
        svm_c = setting['svm_c']
        svm_gamma = setting.get('svm_gamma', 'auto')
        clf = SVC(C=svm_c, kernel='linear', gamma=svm_gamma)
        clf.fit(train_bow, train_labels)

        # Predict and evaluate
        test_preds = clf.predict(test_bow)
        accuracy = accuracy_score(test_labels, test_preds)

        # Store the results
        results_table.append({
            'sift_thresh': setting['sift_thresh'],
            'svm_c': svm_c,
            'svm_gamma': svm_gamma,
            'accuracy': accuracy
        })

    # Markdown table header
    print("| SIFT Threshold | SVM C | SVM Gamma | Accuracy |")
    print("|-----|-----|-----|-----|")

    # Fill the table with results
    for result in results_table:
        print(f"| {result['sift_thresh']} | {result['svm_c']} |
↪{result['svm_gamma']} | {result['accuracy']:.2f} |")

```

```

-----
ValueError                                Traceback (most recent call last)
Cell In[10], line 30
    28 svm_gamma = setting.get('svm_gamma', 'auto')
    29 clf = SVC(C=svm_c, kernel='linear', gamma=svm_gamma)
--> 30 clf.fit(train_bow, train_labels)
    32 # Predict and evaluate
    33 test_preds = clf.predict(test_bow)

File ~/miniforge3/envs/a2q1/lib/python3.8/site-packages/sklearn/base.py:1152, in
↳ _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
    1145     estimator._validate_params()
    1147 with config_context(
    1148     skip_parameter_validation=(
    1149         prefer_skip_nested_validation or global_skip_validation
    1150     )
    1151 ):
-> 1152     return fit_method(estimator, *args, **kwargs)

File ~/miniforge3/envs/a2q1/lib/python3.8/site-packages/sklearn/svm/_base.py:
↳ 190, in BaseLibSVM.fit(self, X, y, sample_weight)
    188     check_consistent_length(X, y)
    189 else:
--> 190     X, y = self._validate_data(
    191         X,
    192         y,
    193         dtype=np.float64,
    194         order="C",
    195         accept_sparse="csr",
    196         accept_large_sparse=False,
    197     )
    199 y = self._validate_targets(y)
    201 sample_weight = np.asarray(
    202     [] if sample_weight is None else sample_weight, dtype=np.float64
    203 )

File ~/miniforge3/envs/a2q1/lib/python3.8/site-packages/sklearn/base.py:622, in
↳ BaseEstimator._validate_data(self, X, y, reset, validate_separately,
↳ cast_to_ndarray, **check_params)
    620     y = check_array(y, input_name="y", **check_y_params)
    621 else:
--> 622     X, y = check_X_y(X, y, **check_params)
    623     out = X, y
    625 if not no_val_X and check_params.get("ensure_2d", True):

```

```

File ~/miniforge3/envs/a2q1/lib/python3.8/site-packages/sklearn/utils/validation.py:1146, in check_X_y(X, y, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y_numeric, estimator)
    1141     estimator_name = _check_estimator_name(estimator)
    1142     raise ValueError(
    1143         f"{estimator_name} requires y to be passed, but the target y is,
    1144         None
    1145     )
-> 1146 X = check_array(
    1147     X,
    1148     accept_sparse=accept_sparse,
    1149     accept_large_sparse=accept_large_sparse,
    1150     dtype=dtype,
    1151     order=order,
    1152     copy=copy,
    1153     force_all_finite=force_all_finite,
    1154     ensure_2d=ensure_2d,
    1155     allow_nd=allow_nd,
    1156     ensure_min_samples=ensure_min_samples,
    1157     ensure_min_features=ensure_min_features,
    1158     estimator=estimator,
    1159     input_name="X",
    1160 )
    1162 y = _check_y(y, multi_output=multi_output, y_numeric=y_numeric,
    1163 estimator=estimator)
    1164 check_consistent_length(X, y)

```

```

File ~/miniforge3/envs/a2q1/lib/python3.8/site-packages/sklearn/utils/validation.py:951, in check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator, input_name)
    946     raise ValueError(
    947         "dtype='numeric' is not compatible with arrays of bytes/strings "
    948         "Convert your data to numeric values explicitly instead."
    949     )
    950 if not allow_nd and array.ndim >= 3:
--> 951     raise ValueError(
    952         "Found array with dim %d. %s expected <= 2."
    953         % (array.ndim, estimator_name)
    954     )
    956 if force_all_finite:
    957     _assert_all_finite(
    958         array,
    959         input_name=input_name,
    960         estimator_name=estimator_name,
    961         allow_nan=force_all_finite == "allow-nan",
    962     )

```



```
ValueError: Found array with dim 3. SVC expected <= 2.
```

Challenges:

1. Tuning and modularizing the pipeline for SIFT detectors, kmeans and SVM
2. Vectorizing the code to ensure optimal run time

Learning Outcomes: Challenges of Manually performing image classification such as: 1. Slow run time 2. Low accuracy 3. Accuracy v/s robustness trade off is high