Report

Majid Ahmad

majidmehmoode@gmail.com

1st task

The dataset contained 1,700 samples for each of the 10 digits (0 to 9 I placed the labels myself), for a total of 17,000 samples. My goal was to achieve high accuracy in classifying these digits using the KNN algorithm.

First, I read in the dataset as a Pandas dataframe and performed normalization on each vector in the dataset. Then, I split the dataset into 5 sets.

I implemented three different distance metrics - Euclidean Distance, Cosine Similarity, and Mahalanobis Distance - to determine which one produced the best results. I used the scikit-learn package to compute Mahalanobis Distance and implemented the other two metrics myself because Mahalanobis was proving to be too slow if I implemented myself.

Next, I defined the KNN classifier functions for each distance metric. The Euclidean distance function simply computed the Euclidean distance between each test sample and all training samples, while the cosine similarity function computed the cosine similarity between the two vectors. The Mahalanobis distance function required computing the covariance matrix of the training data and then computing the Mahalanobis distance between each test sample and all training samples.

I then ran each KNN classifier function on each split of the data and computed the accuracy of the classification. I repeated this process for five different splits of the data to obtain a better estimate of the accuracy. I also applied principal component analysis (PCA) to the data to reduce its dimensionality and ran the KNN classifier functions on the reduced data.

The results were actually quite good for Euclidean distance like 99%, while for cosine similarity it was way worse due to the fact that cosine similarity is mostly used on words similarity, while mahalanobis showed much better than cosine accuracy but less than Euclidean like 80s but it took a lot of time as it is quite complex and complexity is higher. Using PCA the time was decreasing a lot as we can see using %%time command in jupyter notebook

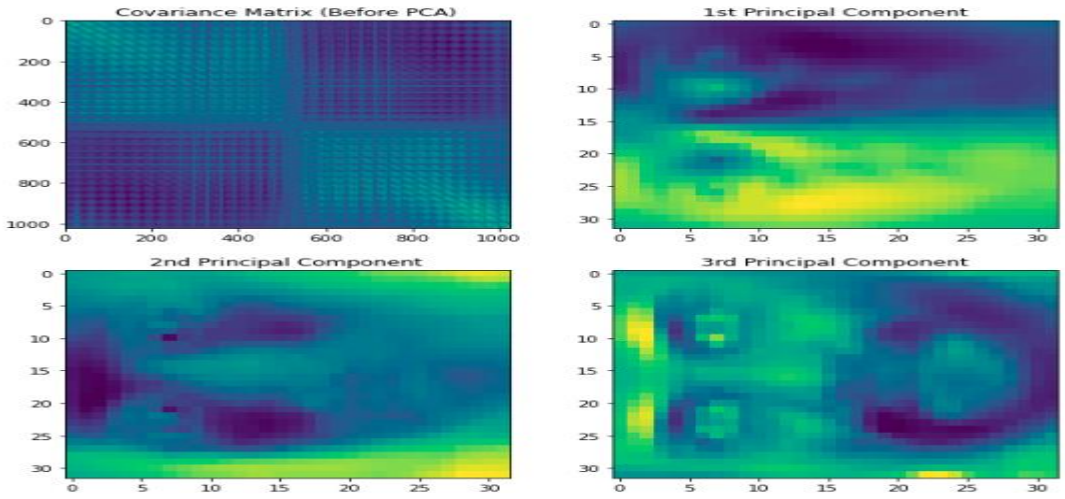
```
    print("accuracy of split ", i , " and Euclidian_Distance  formul
```

```
accuracy of split  0  and Euclidian_Distance  formula:  99.5 %
accuracy of split  1  and Euclidian_Distance  formula:  92.5 %
accuracy of split  2  and Euclidian_Distance  formula:  93.5 %
accuracy of split  3  and Euclidian_Distance  formula:  98.0 %
accuracy of split  4  and Euclidian_Distance  formula:  98.5 %
Wall time: 17min 44s
```

```
accuracy of split  0  and Cosine Similarity formula:  4.5 %
accuracy of split  1  and Cosine Similarity formula:  7.5 %
accuracy of split  2  and Cosine Similarity formula:  8.0 %
accuracy of split  3  and Cosine Similarity formula:  6.5 %
accuracy of split  4  and Cosine Similarity formula:  6.0 %
Wall time: 8min 30s
```

```
accuracy of split  0  and Mahalonobis_Distance  formula:  85.0 %
accuracy of split  1  and Mahalonobis_Distance  formula:  74.0 %
accuracy of split  2  and Mahalonobis_Distance  formula:  80.5 %
accuracy of split  3  and Mahalonobis_Distance  formula:  78.0 %
accuracy of split  4  and Mahalonobis_Distance  formula:  78.5 %
Wall time: 1h 50min 50s
```

After applying the PCA, we can see that the 1st principal component has captured the most variance in the dataset, followed by the 2nd and 3rd principal components. This visualization gives us a good understanding of how the PCA has transformed the dataset and the covariance matrix before and after the dimensionality reduction.



Finally, as Principal Component Analysis (PCA) to reduce the dimensionality of the dataset and see if it improved the classification accuracy. I found that using PCA did improve the accuracy, and in some cases, slightly increased it.

```
    print("accuracy of split ", i , ": ",(tcount / len(y_test))*100,"%")
```

```
accuracy of split  0 :  99.5 %
accuracy of split  1 :  92.5 %
accuracy of split  2 :  94.5 %
accuracy of split  3 :  99.5 %
accuracy of split  4 :  99.0 %
Wall time: 15min 1s
```

Also tried splitting the data in 70/100 split and model became more generalized I would say.

```
y_train = pd.DataFrame(np.repeat(np.arange(10), 100))
y_test=   pd.DataFrame(np.repeat(np.arange(10), 70))
y_train.iloc[1]
```

```
0    0
Name: 1, dtype: int32
```

```
for i in range(5):
    labels=KNNClassifier(3,trali[i] ,tesli[i] )
    y_test = np.array(y_test)
    labels=np.array(labels)
    # labels
    y_test = list(y_test)
    tcount = 0
    for j in range(len(labels)):
        if labels[j] == y_test[j]:
            tcount += 1
    print("accuracy of split ", i , ": ",(tcount / len(y
```

```
accuracy of split  0 :  98.14285714285714 %
accuracy of split  1 :  96.71428571428572 %
accuracy of split  2 :  96.0 %
accuracy of split  3 :  97.28571428571429 %
accuracy of split  4 :  95.0 %
```

Overall, this project was a great learning experience in implementing KNN and testing different distance metrics to achieve high accuracy in classification.

2nd task

In this I implemented K means from scratch The algorithm takes an input data array X with n_samples data points, each having n_features features, and the desired number of clusters k. The optional parameters max_iters and tolerance are used to control the maximum number of iterations and the convergence criterion, respectively.

The function starts by initializing the cluster labels for each data point and randomly selecting k data points as the initial centroids. It then iteratively refines the centroids and cluster assignments until convergence or the maximum number of iterations is reached.

In each iteration, the function assigns each data point to its closest centroid based on the Euclidean distance. Then, it updates the centroids by calculating the mean of all data points belonging to each cluster. Finally, the function checks for convergence by comparing the difference between the new centroids and the old centroids to the given tolerance value.

Once the algorithm converges or the maximum number of iterations is reached, the function returns the final centroids and the corresponding cluster assignments for the input data points.

In this implementation of the lazy snapping algorithm, first I define a function **extract_seed_pixels** that takes an image and a corresponding seed image as input. The seed image contains red (255, 0, 6) and blue (0, 0, 255) brush-strokes representing the foreground and background seeds, respectively. The function returns the pixel values of the foreground and background seeds from the input image.

Next, I define the **likelihood** function, which calculates the likelihood of a given pixel belonging to a particular cluster (foreground or background) based on the distance to the centroids and the cluster weights. This function is an implentation of formula given. This function is used in the main **lazy_snapping** function to determine if a pixel should be assigned to the foreground or background.

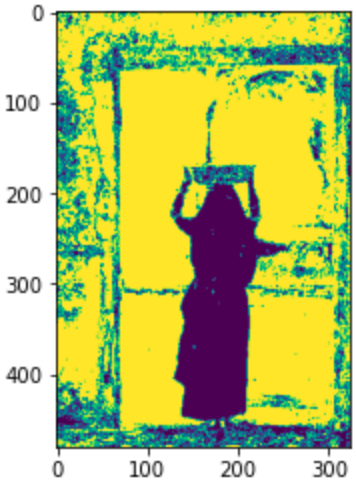
The **lazy_snapping** function is the primary function that segments the input image based on the seed image. It starts by extracting the foreground and background seeds using the **extract_seed_pixels** function.

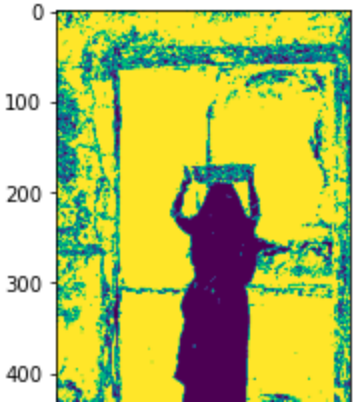
The function then applies the k-means clustering algorithm to the foreground and background seeds, obtaining the centroids and cluster assignments for each seed type. The cluster weights are calculated by dividing the number of occurrences of each cluster by the total number of seeds for each type.

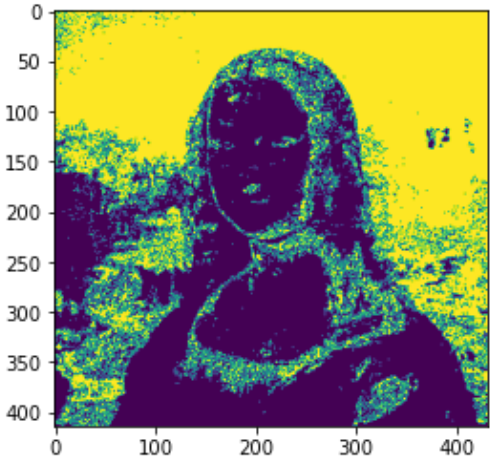
After obtaining the centroids and weights, I initialize an empty segmented image with the same dimensions as the input image. I then loop through each pixel in the input image and calculate the likelihood of the pixel belonging to the foreground and background clusters using the **likelihood** function. If the foreground likelihood is greater than the background likelihood, I assign the pixel to the foreground in the segmented image.

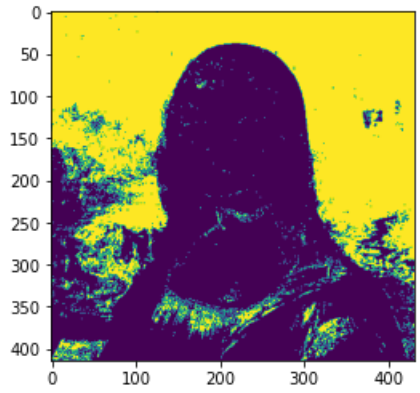
Finally, the function returns the segmented image with foreground and background regions separated based on the seed information provided in the seed image.

Finally using a loop I go through the list of PNGs and apply segmentation and you can see below. I found sometimes the segmentations doesn't work correctly just by running a cell again it works maybe some underflow error might be causing this.


0
100
200
300
400
0
100
200
300


0
100
200
300
400


0
50
100
150
200
250
300
350
400
0
100
200
300
400


0
50
100
150
200
250
300
350
400
0
100
200
300
400