K-Nearest Neighbor classifier

Assignment-1 Report

-Sridevi Divya Krishna Devisetty(4436572)

Summary of code implementation:

- The program reads files from the directory ./mnist/dataset/ , parses these files and extracts train images, test images, train labels and test labels as numpy arrays.
- It then randomly shuffles train data and equally distributes them into 10 folds, to perform 10 fold cross validation on train data.
- For each of k value ranging from 1 to 10, it executes 10-fold cross validation and finds mean of all the fold accuracies and assigns it as accuracy for that k-value.
- The maximum accuracy of all these k- accuracies is considered as the optimal-k accuracy.
- The k value with maximum accuracy is considered as optimal-k.
- We then execute standard knn classifier with train and test images and obtain classification accuracy with confidence interval. We report confusion matrix and confidence interval.
- After that we execute sliding window knn classifier. We report confusion matrix and confidence interval with classification accuracy.

Execute Program:

1. Place the mnist files under a directory and change the constant(DATA_DIR) value in program file to that directory. *DATA_DIR = "./mnist/dataset/"*
2. Run train.py file.

Packages Used:

Python inbuilt:

- struct : to extract data
- Counter : to find neighbor with maximum counts
- os : to get files from directory
- copy : to copy arrays
- itertools : to flatten list
- math: to find sqrt of standard deviation

Other:

- numpy : matrix operations.

Dataset Details:

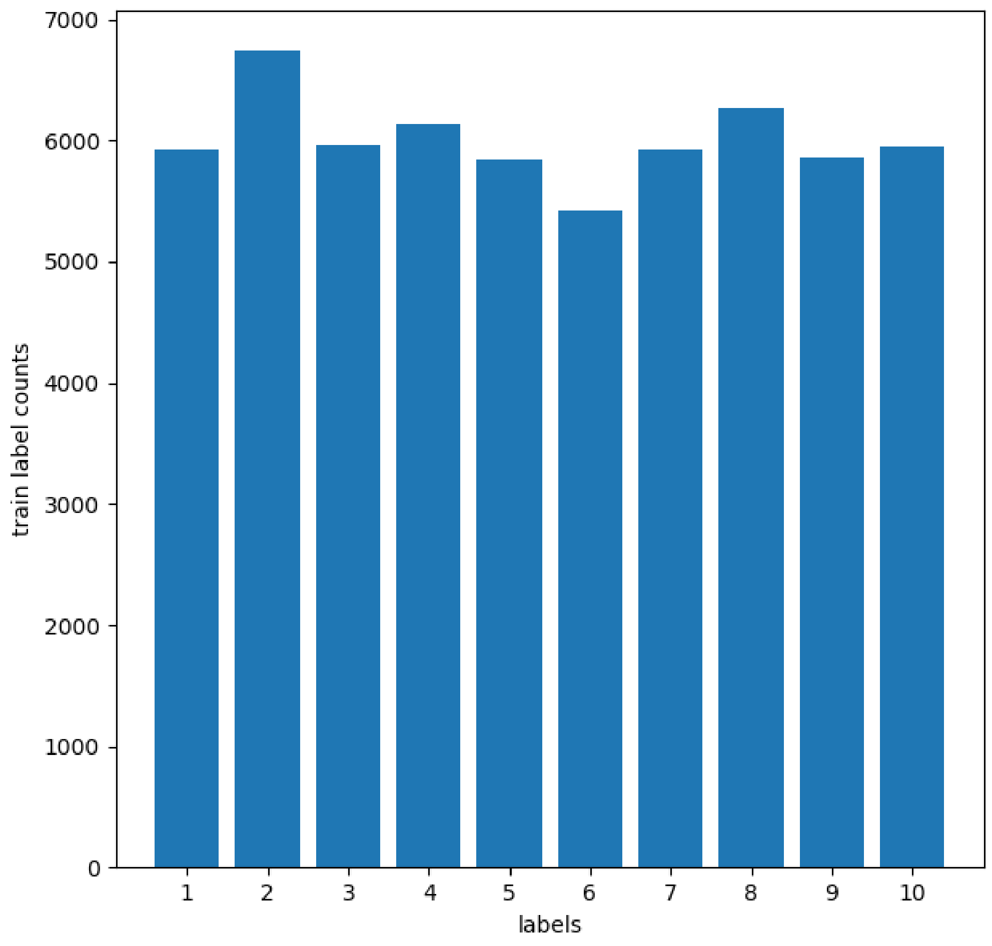
Train images: 60000, each image of shape 28*28 **Test images:** 10000, each image of shape 28*28

Train Labels: 60000 **Test Labels:** 10000

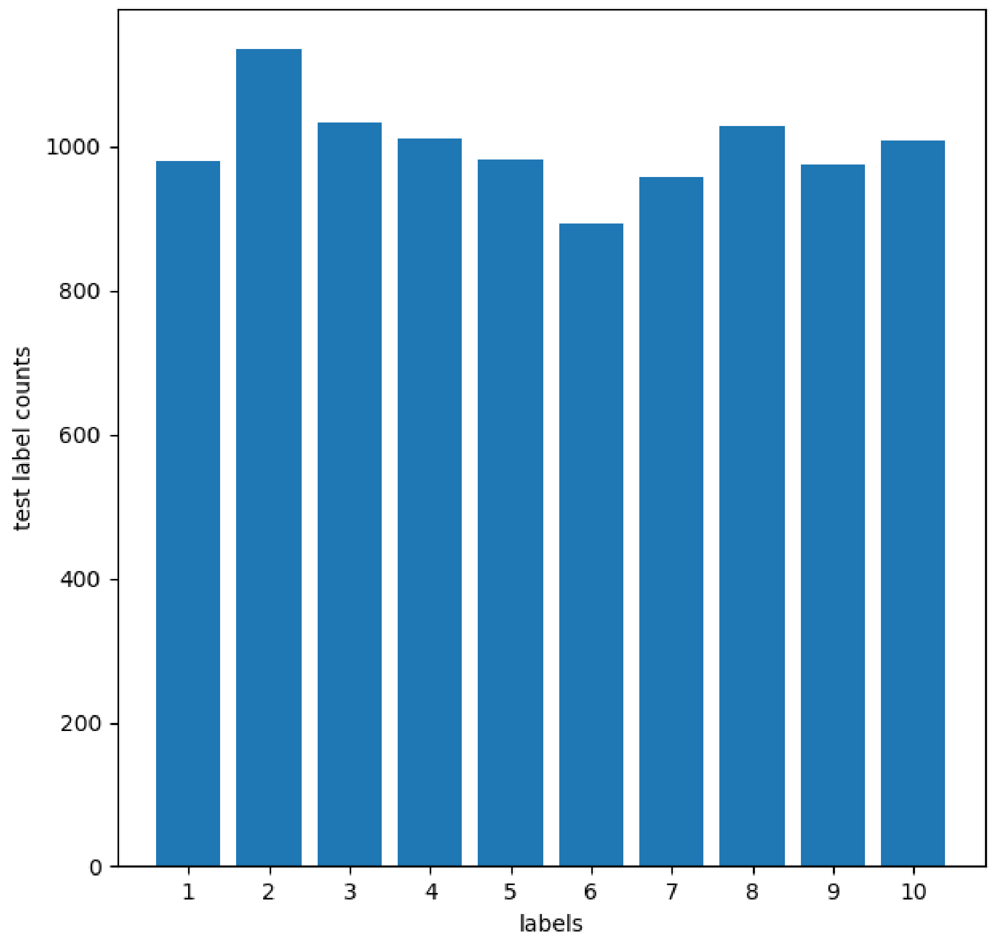
Train and Test data size for each label:

Label	Train size	Test size
0	5923	980
1	6742	1135
2	5958	1032
3	6131	1010
4	5842	982
5	5421	892
6	5918	958
7	6265	1028
8	5851	974
9	5949	1009

Train label count for each label:



Test Label count for each label:



Pseudocode and Results:

1. Data Parsing and dividing into 10-folds:

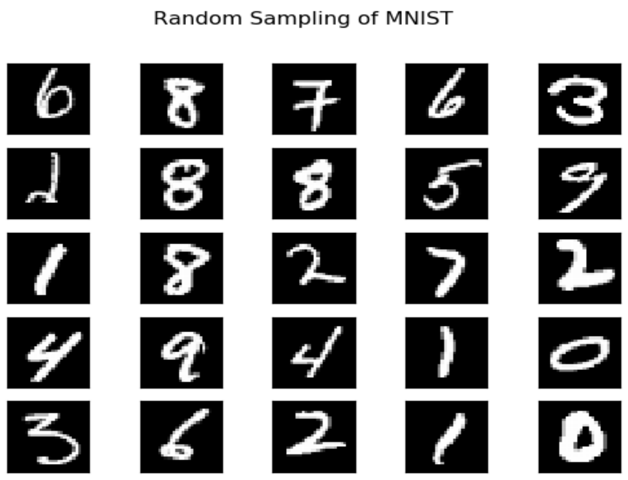
Train and Test data files are parsed into numpy arrays of train and test images. Train images are then randomly shuffled and divided into 10-folds.

Pseudocode for data preprocessing:

```
train_data, train_labels, test_data, test_labels = get_mnist_data()
        dataset = (train_data, train_labels)
        random. shuffle(dataset)
        folds=[]
        fold_size=len(dataset)//10
        for i in range(0,len(dataset),fold_size):
            folds.append(i,i+fold_size)
```

Results of data parsing and shuffling:

Figure represents First 25 images of shuffles data.



2. 10 Fold cross Validation for k-values ranging from 1 to 10:

Out of 60k train images, we randomly shuffle and distribute them into 10 equal folds, we carry out 10 fold cross validation by using one of the folds as validation set in each of 10 fold iterations, with 54k images as train data and 6k images as validation data out of 60k train image set. We obtain accuracies of all iterations for each k, and mean of those accuracies is assigned as accuracy for that k-value. Max accuracy k-value is optimal k.

Pseudocode for cross-validation:

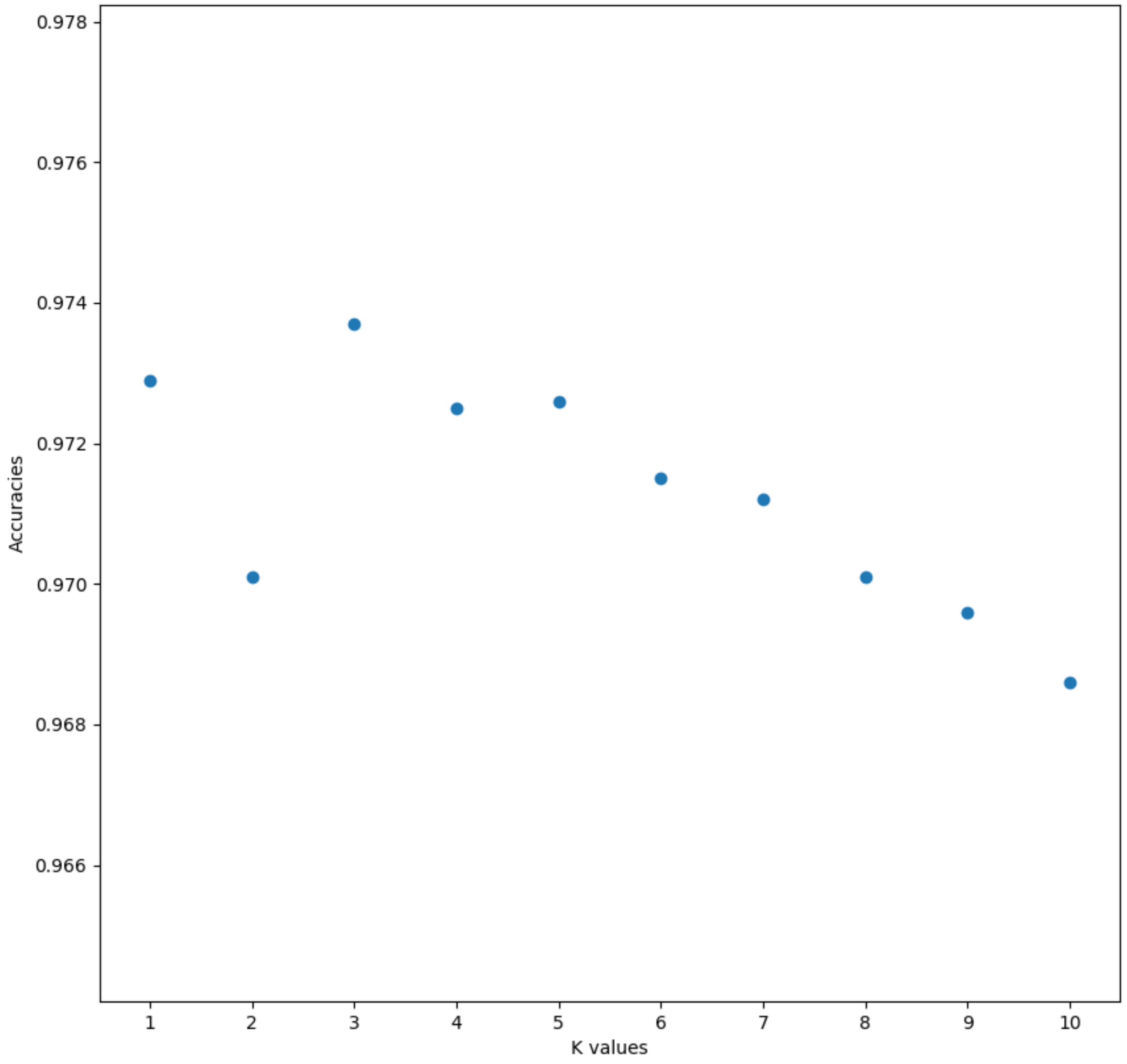
```
k_accuracy=[]
for k in range(1,11): #11 is exclusive
    fold_accuracy=[]
    for j in range(len(folds)):
        validation_fold = folds[j]
        train_fold = folds[:j]+folds[j+1:]
        model = KNN(train_fold,k)
        valid_predictions = [model.predict(img) for img in validation_fold]
        fold_accuracy.append((valid_predictions==test_labels)/no.of.test_labels)
    k_accuracy.append(sum(fold_accuracy)/len(folds))
optimal_k = k_accuracy.index(max(k_accuracy))+1
```

Results for cross validation:

10-fold cross-validation:

Folds\k	K=1	K=2	K=3	K=4	K=5	K=6	K=7	K=8	K=9	K=10
Fold-1	0.9722	0.97	0.972	0.9755	0.9752	0.9738	0.9709	0.9698	0.9685	0.9689
Fold-2	0.9747	0.9715	0.9753	0.971	0.9698	0.9685	0.9667	0.9643	0.9638	0.963
Fold-3	0.978	0.976	0.9778	0.9728	0.9747	0.9741	0.9734	0.9722	0.973	0.972
Fold-4	0.974	0.9672	0.9723	0.9706	0.9717	0.9693	0.9723	0.972	0.9702	0.9695
Fold-5	0.9704	0.9702	0.9715	0.9746	0.9761	0.9743	0.9741	0.9732	0.9718	0.9713
Fold-6	0.9714	0.9675	0.9743	0.9731	0.973	0.972	0.9699	0.9687	0.9675	0.967
Fold-7	0.97384	0.9704	0.9738	0.9718	0.9693	0.9682	0.9715	0.9713	0.9697	0.9697
Fold-8	0.9714	0.9702	0.9732	0.9746	0.9741	0.974	0.9739	0.9717	0.973	0.9702
Fold-9	0.97184	0.9675	0.972	0.9696	0.9715	0.971	0.9698	0.9703	0.9696	0.9687
Fold-10	0.973	0.9704	0.9748	0.9715	0.9703	0.9703	0.9687	0.9673	0.969	0.9673
Final Accuracy	**0.9729**	**0.9701**	**0.9737**	**0.9725**	**0.9726**	**0.9715**	**0.9712**	**0.9701**	**0.9696**	**0.9686**

Optimal k-value = 3 with accuracy 0.9737 (97.37%)



3. Train knn classifier with optimal-k=3 and test it on test image set:

For each image in test set of 10000 we find distances with all train set images of 60000, sort the distance and consider the first k values and return the maximum frequent class label as prediction.

Pseudocode for Standard knn classifier:

```
class KNN:
    def __init__(self,train_img,k):
        self.train_images=train_img
        self.k=k
    def predict(test_img):
        dist = [(euclid_dist(test_img, img),i) for i, img in enumerate(train_images)]
        dist.sort()
        for i in range(k):
            index=dist[i][1]
            labels.append(train_labels[index])
        return Counter(labels).most_common(1)[0][0]
```

Pseudocode for calling standard KNN and printing result:

```
train_model = KNN(train,optimal_k)
test_predictions=[ train_model.predict(img) for img in test_images]
test_accuracy = get_accuracy(test_predictions,test_labels)
std_dev= get_std_dev(test_accuracy,len(test_lables))
confidence_interval=get_conf_intvl(test_accuracy,variance)
print_confusion_matrix(test_predictions, test_labels)
```

Results for standard KNN:

Accuracy: **Accuracy: 0.9716 →97.16% Error: 0.0284 → 2.84%**

Accuracy: Number of correct predictions/Total Predictions
Error: Number of incorrect predictions/Total Predictions

Confusion Matrix:

Actual|predicted	0	1	2	3	4	5	6	7	8	9
0	**974**	1	1	0	0	1	2	1	0	0
1	0	**1133**	2	0	0	0	0	0	0	0
2	10	9	**995**	2	0	0	0	13	2	1
3	0	1	4	**974**	1	11	1	7	4	7
4	1	6	0	0	**948**	0	4	2	1	20
5	6	1	0	11	2	**859**	5	1	3	4
6	5	3	0	0	3	3	**944**	0	0	0
7	0	21	4	0	1	0	0	**991**	0	11
8	7	0	3	11	4	10	3	4	**927**	5
9	4	4	1	5	9	2	1	8	4	**971**

Confidence Interval:

Confidence interval=(accuracy + z*variance, accuracy-z*variance)
Z=1.96 ; n=10000 ; variance = sqrt((accuracy*(1-accuracy))/n);

Confidence interval for classification accuracy: (0.9683441904370188, 0.9748558095629812)

4. Train knn sliding window classifier with k=3 and test it on test image set:

In sliding window approach, we find Euclidean distance between a test image and a train image which have been cropped to 9 images of size 28*28 after border padding with 0s.

Pseudocode for sliding KNN classifier:

```
def get_slide_image(train_image):
            n = pad_train_img(train_image)
            r, c = len(train_image), len(train_image[0]) #window size
            for x in range(3):
              for y in range(3):
                yield n[x:x + r, y:y + c]
  class KNN:
       def __init__(self,train_img,k):
            self.train_images=train_img
            self.k=k
       def predict_slide(test_img):
           dist=[]
           for i,img in enumerate(train_images):
               slide_dist=[euclid_dist(test_img,slide_img) for slide_img in get_slide_image(train_image)]
                dist.append((min(slide_dist),i))
           dist.sort()
          for i in range(k):
              index=dist[i][1]
              labels.append(train_labels[index])
          return Counter(labels).most_common(1)[0][0]
```

Pseudocode for calling KNN for slide model and printing results:

```
slide_model = KNN(train,optimal_k)

test_predictions_slide=[ train_model.predict_slide(img) for img in test_images]

test_accuracy_slide = get_slide_accuracy(test_predictions_slide,test_labels)

variance_stddev= get_slide_stddev(test_accuracy_slide, len(test_labels))

confidence_interval_slide=get_slide_confidence(variance_slide,test_accuracy_slide)

print_confusion_matrix(test_predictions_slide, test_labels)
```

Results for Sliding window KNN:

Accuracy for sliding window: **Accuracy: 0.977 → 97.7%** **Error: 0.023 →2.3%**

Accuracy: Number of correct predictions/Total Predictions
Error: Number of incorrect predictions/Total Predictions

Confusion Matrix for sliding window:

Actual|predicted	0	1	2	3	4	5	6	7	8	9
0	**976**	0	1	0	0	1	1	1	0	0
1	0	**1131**	2	0	0	0	1	0	0	1
2	6	4	**1006**	2	0	0	0	12	1	1
3	0	1	4	**988**	0	5	0	3	5	4
4	0	8	0	0	**952**	0	3	1	0	18
5	2	2	0	7	1	**869**	6	1	1	3
6	7	4	0	0	2	1	**944**	0	0	0
7	0	20	3	1	2	0	0	**994**	0	8
8	5	0	1	9	5	6	2	3	**936**	7
9	3	8	1	3	5	4	0	8	3	**974**

Confidence Interval for sliding window:

```
Confidence interval=(accuracy + z*variance, accuracy-z*variance)
Z=1.96 ; n=10000 ; variance = sqrt((accuracy*(1-accuracy))/n);
```

Confidence interval for classification accuracy: (0.9740618952775641, 0.9799381047224358)

5.Discussion on classifier Performance:

Classification Model	Accuracy in percentage
Standard KNN	**97.16**
Sliding window KNN	**97.70**

The table above shows the comparison between the performance of the two models. It can be seen that there is slight improvement in the sliding window performance. It can be concluded that model's ability to recognize labels have improved. This indicates that the sliding window improved the accuracy and that one can conclude with 95% confidence or more that obtained values were not just due to chance.

About p-value:

p-value is the probability of finding any result under the test conditions for a given set of data presuming that the null hypothesis were true. The p-value has to be smaller than the significance level because that gives us an indication that the probability of that result happening is very low (in our case, less than 5%) by chance only.

About significance threshold alpha:

0.05 value represents how extreme we want our evidence to be before we reject the null hypothesis. 0.05 means that if the null were true, we would only get evidence that is extreme enough to incorrectly reject the null about 5% of the time.

Significance Testing:

Standard KNN accuracy (a1)= 0.9716 (97.16%)

Sliding KNN accuracy(a2) = 0.977 (97.7%)

Difference of accuracies(a2-a1)= 0.005399999999999996 (0.5%)

Mean of accuracies on 20k samples((a1+a2)/20000) = 9.743e-05=0.00009743

p=(1.96*sqrt((mean*(1-mean))/20000)) = 0.00013679375373333149

p is less than difference of accuracies.

p < alpha(0.05), so we reject null-hypothesis and it is evident that there is significance difference.

Difference Rule:

We can see that difference between accuracies of both knn classifiers is more than 0.2%, this could be due to that in sliding window we calculate distances on 9 cropped images of each of train images, by sliding over the entire train image and we consider the best matched image among 9 image shifts, by taking best distance among 9 images, hence improving our distance metric, thus by making the model more robust to minor translations and noise .