Handwritten Digit Recognition using Decision Trees, KNN and SVM

Arshiful Islam Shadman, Jerome Doe

George Washington University

Department of Data Science

1. Introduction

Handwritten recognition is not a new concept as we have already seen how a computer can receive inputs of handwritten letters and numbers from various sources and convert them into actual texts. This project is to understand how classification techniques such as Decision Trees, KNN and SVM can be used to recognize digits and compare all three to find the best technique.

2. Method

For this project we used python and leveraged sklearn to run the three above mentioned classification techniques. The dataset being used is the famous Modified National Institute of Standards and Technology (MNIST) database constructed from 42,000 scanned images of handwritten digits (i.e. 0 to 9). Each image in the dataset is 28 by 28 pixel square giving a total of 784 pixels. The dataset was split into 1:1 train and test ratio before applying the classification techniques to the training dataset, build respective models and then apply the models over the test dataset. The accuracy and the runtime of the models to classify and determine the labels of the test dataset was determined for Decision tree, KNN and SVM.

2.1 A Glimpse into the dataset

Table 1. Head of the dataset

label	pixel0	pixel1	pixel2	……...	pixel781	pixel782	pixel783
1	0	0	0		0	0	0
0	0	0	0		0	0	0
1	0	0	0		0	0	0
4	0	0	0		0	0	0
0	0	0	0		0	0	0

Note. The most common pixel format is the byte image, where this number is stored as an 8-bit integer giving a range of possible values from 0 to 255. Typically zero is taken to be black, and 255 is taken to be white. Values in between make up the different shades of gray.

2. Decision Tree (DT)

The Decision Tree classifier is widely used in classification. The aim is to have a model that can classify the label of test handwritten digits data by discovering list of rules from the training dataset. DT classifier tools have been used in different fields, such as digit and text predicative, extract information, machine learning, and recognition.

2.1 Python Script (hand_written-decisionTree.py)

```
from datetime import datetime
startTime = datetime.now()
import numpy as np
import matplotlib.pyplot as pt
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import tree
data=pd.read_csv("imageData.csv").as_matrix()
clf=DecisionTreeClassifier()
#training_dataset
xtrain=data[0:21000,1:]
train_label=data[0:21000,0]
clf.fit(xtrain,train_label)
#testing data
xtest=data[21000:,1:]
actual_label=data[21000:,0]
#d=xtest[15]# this is a 1
#d=xtest[16]# this is a 9
#d=xtest[17]# this is a 5
d=xtest[18]# this is a 2
d.shape=(28,28)
pt.imshow(255-d,cmap='gray')
#print(clf.predict([xtest[15]])) # this is a 1
#print(clf.predict([xtest[16]])) # this is a 9
#print(clf.predict([xtest[17]])) # this is a 5
print(clf.predict([xtest[18]])) # this is a 2
pt.show()
p=clf.predict(xtest)
count=0
for i in range (0,21000):
    count+=1 if p[i]==actual_label[i] else 0
print("Decision Tree Accuracy=", (count/21000)*100)
print(confusion_matrix(actual_label,p))
print(classification_report(actual_label,p))
tree.plot_tree(clf)
print(datetime.now() - startTime)
```

2.2 Confusion Matrix for Decision Tree

		Actual label										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
Predicted label	**0**	1904	4	27	32	14	28	33	11	26	9	2088
	1	1	2170	29	24	16	10	13	17	23	9	2312
	2	42	43	1661	72	27	23	43	56	77	25	2069
	3	26	22	98	1676	24	110	17	44	85	72	2174
	4	10	22	29	17	1686	23	28	37	38	107	1997
	5	30	20	41	127	35	1427	53	26	92	46	1897
	6	46	15	29	18	41	79	1770	5	45	14	2062
	7	8	13	50	18	26	14	2	2000	21	82	2234
	8	35	59	56	82	55	77	35	22	1580	57	2058
	9	14	16	37	43	102	59	9	79	57	1693	2109
		2116	2384	2057	2109	2026	1850	2003	2297	2044	2114	21000

2.3 Classification Report for Decision Tree

	precision	**recall**	**f1-score**	**support**
0	0.9	0.91	0.91	2088
1	0.91	0.94	0.92	2312
2	0.81	0.8	0.81	2069
3	0.79	0.77	0.78	2174
4	0.83	0.84	0.84	1997
5	0.77	0.75	0.76	1897
6	0.88	0.86	0.87	2062
7	0.87	0.9	0.88	2234
8	0.77	0.77	0.77	2058
9	0.8	0.8	0.8	2109
accuracy			**0.84**	21000
macro avg	0.83	0.83	0.83	21000
weighted avg	0.84	0.84	0.84	21000

3. K-nearest neighbor (KNN)

This technique relies on the distance between feature vectors and classifies unknown data points by finding the most common label among the k closest examples. Each instance in the k closest cast a vote and the highest category number of votes wins. The KNN model made using the training dataset was applied over the 21000 test data points of images. A k-value of 10 was chosen. We didn’t chose a range of k-value as calculating k numbered distances for all 21,000 is computationally expensive.

3.1 Python Script (hand_written-knn.py)

```
from datetime import datetime
startTime = datetime.now()
import numpy as np
import matplotlib.pyplot as pt
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import os
#import pixel data
data=pd.read_csv("imageData.csv").as_matrix()
neighbors = 10
knn = KNeighborsClassifier(n_neighbors=neighbors) # instantiate with n value given
#training_dataset
xtrain=data[0:21000,1:]
train_label=data[0:21000,0]
knn.fit(xtrain,train_label)
#testing data
xtest=data[21000:,1:]
actual_label=data[21000:,0]
#d=xtest[15]# this is a 1
#d=xtest[16]# this is a 9
#d=xtest[17]# this is a 5
d=xtest[18]# this is a 2
d.shape=(28,28)
pt.imshow(255-d,cmap='gray')
#print(knn.predict([xtest[15]])) # this is a 1
#print(knn.predict([xtest[16]])) # this is a 9
#print(knn.predict([xtest[17]])) # this is a 5
print(knn.predict([xtest[18]])) # this is a 2
pt.show()
p=knn.predict(xtest)
count=0
for i in range (0,21000):
    count+=1 if p[i]==actual_label[i] else 0
print("KNN Accuracy=", (count/21000)*100)
print(confusion_matrix(actual_label,p))
print(classification_report(actual_label,p))
print(datetime.now() - startTime)
```

3.2 Confusion Matrix for KNN

		Actual label										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
Predicted label	**0**	2073	0	2	0	0	3	10	0	0	0	**2088**
	1	0	2299	5	0	1	1	0	6	0	0	**2312**
	2	20	28	1955	7	3	6	4	41	4	1	**2069**
	3	3	13	13	2064	0	29	1	19	18	14	**2174**
	4	3	24	0	0	1906	0	9	4	0	51	**1997**
	5	7	6	0	33	2	1806	27	1	2	13	**1897**
	6	14	3	1	0	2	11	2031	0	0	0	**2062**
	7	1	22	8	1	10	0	0	2162	0	30	**2234**
	8	8	39	6	38	13	44	9	5	1869	27	**2058**
	9	12	9	4	20	26	5	0	41	7	1985	**2109**
		2141	**2443**	**1994**	**2163**	**1963**	**1905**	**2091**	**2279**	**1900**	**2121**	**21000**

3.3 Classification Report for KNN

	precision	**recall**	**f1-score**	**support**
0	0.97	0.99	0.98	2088
1	0.94	0.99	0.97	2312
2	0.98	0.94	0.96	2069
3	0.95	0.95	0.95	2174
4	0.97	0.95	0.96	1997
5	0.95	0.95	0.95	1897
6	0.97	0.98	0.98	2062
7	0.95	0.97	0.96	2234
8	0.98	0.91	0.94	2058
9	0.94	0.94	0.94	2109
accuracy			**0.96**	21000
macro avg	0.96	0.96	0.96	21000
weighted avg	0.96	0.96	0.96	21000

4. Support Vector Machine (SVM)

SVM is a set of supervised learning methods used for classification, regression and outlier detection. In SVM, each data item will be plot as a point in n-dimensional space (n = number of feature) with the value of each feature being value of a particular coordinate. The classification is done by finding the hyper-plane that differentiate the two (2) classes. The SVM in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input. In scikitlearn have three (3) classes that capable of performing multiclass classification on a dataset which is SVC, NuSVC and LinearSVC. In this system we will use SVC class to perform the classification of MNIST dataset and observe the accuracy.

4.1 Python Script (hand_written-svc.py)

```
from datetime import datetime
startTime = datetime.now()
import numpy as np
import matplotlib.pyplot as pt
import pandas as pd
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import os
#import pixel data
data=pd.read_csv("imageData.csv").as_matrix()
svclassifier = SVC(kernel='linear')
#training_dataset
xtrain=data[0:21000,1:]
train_label=data[0:21000,0]
svclassifier.fit(xtrain,train_label)
#testing data
xtest=data[21000:,1:]
actual_label=data[21000:,0]
#d=xtest[15]# this is a 1
#d=xtest[16]# this is a 9
#d=xtest[17]# this is a 5
d=xtest[18]# this is a 2
d.shape=(28,28)
pt.imshow(255-d,cmap='gray')
#print(knn.predict([xtest[15]])) # this is a 1
#print(knn.predict([xtest[16]])) # this is a 9
#print(knn.predict([xtest[17]])) # this is a 5
print(svclassifier.predict([xtest[18]])) # this is a 2
pt.show()
p=svclassifier.predict(xtest)
count=0
for i in range (0,21000):
    count+=1 if p[i]==actual_label[i] else 0
print("SVM Accuracy=", (count/21000)*100)
print(confusion_matrix(actual_label,p))
print(classification_report(actual_label,p))
print(datetime.now() - startTime)
```

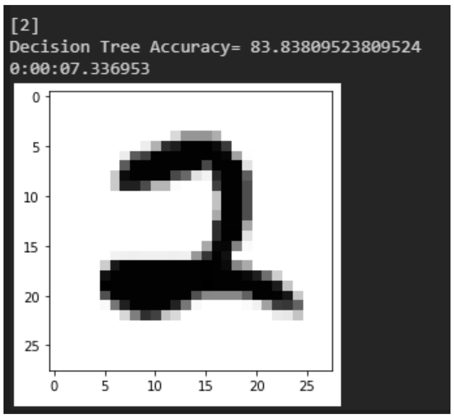
4.2 Confusion Matrix for SVC

		Actual label										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
Predicted label	**0**	2034	0	12	2	3	13	18	0	6	0	**2088**
	1	0	2271	9	5	1	4	2	5	14	1	**2312**
	2	14	23	1889	34	19	10	20	27	31	2	**2069**
	3	7	14	45	1921	4	87	8	15	58	15	**2174**
	4	14	9	33	0	1858	0	16	7	3	57	**1997**
	5	26	7	16	110	20	1635	27	1	40	15	**1897**
	6	21	2	32	0	7	28	1964	0	7	1	**2062**
	7	7	5	32	13	22	4	0	2056	4	91	**2234**
	8	15	49	43	117	9	68	9	11	1722	15	**2058**
	9	14	9	17	30	127	13	0	100	20	1779	**2109**
		2152	**2389**	**2128**	**2232**	**2070**	**1862**	**2064**	**2222**	**1905**	**1976**	**21000**

4.3 Classification Report for SVC

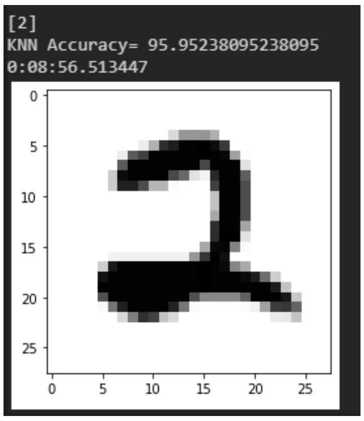
	precision	**recall**	**f1-score**	**support**
0	0.95	0.97	0.96	2088
1	0.95	0.98	0.97	2312
2	0.89	0.91	0.9	2069
3	0.86	0.88	0.87	2174
4	0.9	0.93	0.91	1997
5	0.88	0.86	0.87	1897
6	0.95	0.95	0.95	2062
7	0.93	0.92	0.92	2234
8	0.9	0.84	0.87	2058
9	0.9	0.84	0.87	2109
accuracy			**0.91**	21000
macro avg	0.91	0.91	0.91	21000
weighted avg	0.91	0.91	0.91	21000

5. Comparing Results



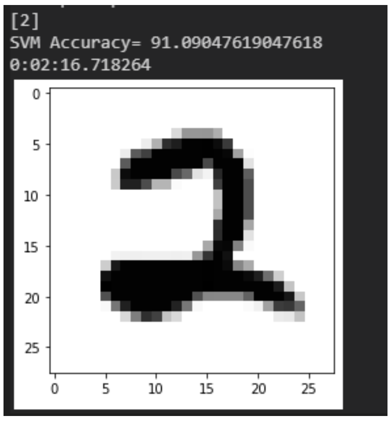
```
[2]
Decision Tree Accuracy= 83.83809523809524
0:00:07.336953
```

Figure 1. DT Output



```
[2]
KNN Accuracy= 95.95238095238095
0:08:56.513447
```

Figure 2. KNN Output



```
[2]
SVM Accuracy= 91.09047619047618
0:02:16.718264
```

Figure 3. SVM Output

	Decision Tree	**SVM**	**KNN**
Accuracy	83.83%	91.09%	95.95%
Classification Runtime	7 seconds	2 min 16 seconds	8 min 56 seconds

6. Conclusion

In this project, we saw 3 different approaches of classification. The K-nearest neighbor’s (KNN) algorithm is fast to train the data but is slow to compute the results. However KNN had the highest accuracy of 96%. On the other hand, the Decision Tree is faster to classify the data but the accuracy of 84% is significantly low compared to that of KNN. The results obtained with SVC were found to be somewhere inbetween with an accuracy of 91%. Clearly KNN outperformed the other two classification techniques but at the expense of computation.

7. References

[1] MNIST ("Modified National Institute of Standards and Technology") Dataset.

Retrieved from https://www.kaggle.com/ngbolin/mnist-dataset-digit-recognizer

[2] Leveraged Scikit Learn Classification Library

https://scikit-learn.org/stable/supervised_learning.html#supervised-learning