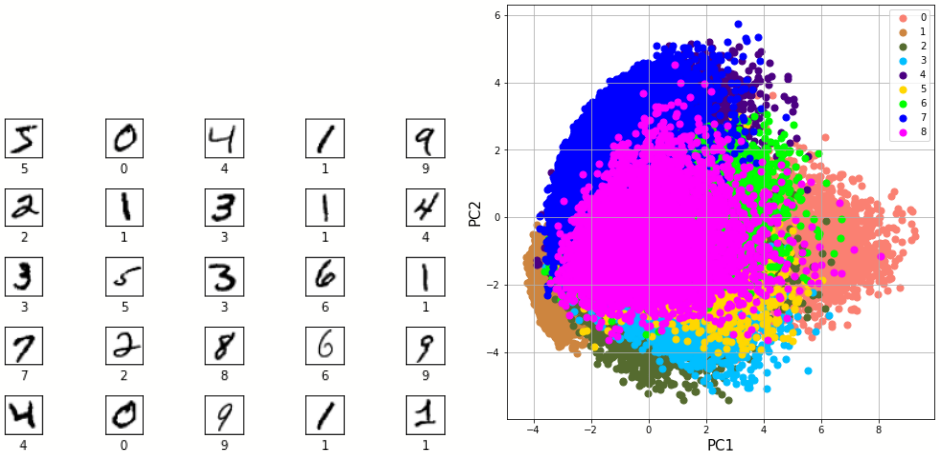
Exploring Machine Learning Methods for Handwritten Digit Classification

Eliot McGinnis and Trent Avent

Introduction and Methodology

We implemented a variety of both supervised and unsupervised machine learning methods in order to classify handwriten digits. Our data is taken from the MNIST library and includes 70000 images of handwriten numbers in the form of 28 x 28 arrays of RGB values. This data has been studied by many machine learning experts and is the subject of papers testing various algorithms. These papers have examined various linear classifiers, k-nearest neighbors with various choices of k, neural networks an other models. We seek to apply methods which we have learned in this class to the data, to find an optimal method of the classical approaches. We fit both unsupervised and supervised methods to the data, and gain insight into the nature of the models in the process. Because the data is pixelated, there is some variation in model performance- some models handle this form of data better than others.

For preprocessing, we split the data into 60000 training instances and 10000 test cases. In order to proceed, we chose to scale the data points between zero and one by dividing all of the predictor values by 255, the previous scale. We use the sklearn and tensorflow libraries for our analysis. Output is taken directly from the Jupyter notebook, which is available online at https://github.com/sureframe846/comp562-final-project



Data Exploration and Principal Components

Above, we use the unsupervised technique principal component analysis to reduce the dimension of the data for our exploratory analysis. The first two principal components contain a large part of the variation of the data, and by plotting these components and coloring by digit value, we see some relationships in the data. Principal component analysis is an excellent way to visualize our data.
Using only the first two principal components, there is no clear decision boundary. There is overlap among all of the classes, and our models will need to overcome this. This is natural- perhaps messier handwriting can lead to digits which are harder to recognize for both machine learning models and humans.

Supervised Approaches

SVM

We first implemented SVM using the sklearn library. We fit both linear and RBF kernels. We see that the RBF kernel performs better than the linear kernel, with a very low test error rate.

```
RBF kernel SVM test error:                Linear SVM test error
0.0208                                    0.059
RBF- error by class:
0    0.992857
1    0.992070
2    0.974806
3    0.985149
4    0.978615
5    0.976457
6    0.985386
7    0.968872
8    0.975359
9    0.961348
```

Multiclass Logistic Regression

We use sklearn to fit multiclass logistic regression on the data. The performance is worse than either of the SVM models. It performs worst on classes 5 and 8.

```
Multiclass Logistic Regression Test error:
0.0802
0    0.978571
1    0.979736
2    0.891473
3    0.908911
4    0.931772
5    0.862108
6    0.946764
7    0.925097
8    0.872690
9    0.888999
dtype: float64
```

K-Nearest Neighbors

We fit a KNN model to our data. We found the best model to be using K=5. Note that we chose an odd K to ensure that there were no ties for classification in the data along class boundaries. KNN performed excellently. The table below shows the correct rate for each class.

```
KNN- test error:
0.0312
0    0.993878
1    0.998238
2    0.960271
3    0.966337
4    0.961303
5    0.966368
6    0.986430
7    0.961089
8    0.937372
9    0.953419
```

Unsupervised Approaches: Other Methods

K-Means Clustering

We fitted a KMeans model with 10 clusters to the training data. We received poor results as seen in the table. There were no distinct trends apparent, so there was issues properly labelling the clusters. This is similar to the uncertain results that we found by plotting the data on the principal components. We must consider that k-means uses a distance function that may not be appropriate for working with pixel data.

actual	0	1	2	3	4	5	6	7	8	9
cluster										
0	48	2	65	696	0	287	2	0	209	7
1	2	1	709	40	5	4	18	13	6	3
2	5	469	84	7	36	105	30	57	43	11
3	0	660	59	73	30	23	28	59	34	29
4	447	0	3	1	1	9	20	1	6	4
5	423	0	19	18	1	38	24	1	11	10
6	3	0	30	15	559	56	22	292	34	541
7	19	1	27	146	5	281	19	1	586	13
8	30	2	25	7	35	19	794	1	10	4
9	3	0	11	7	310	70	1	603	35	387

Neural Network

Finally, we tried a neural network with 2 hidden layers with 128 and 10 nodes each. The neural net is fit over 10 epochs in order to further increase the accuracy.The final test accuracy of the model is quite good, though not as good as the SVM model and only.

```
0    0.979592
1    0.976211
2    0.898256
3    0.862376
4    0.960285
5    0.904709
6    0.931106
7    0.912451
8    0.880903
9    0.882061
```

```
Test accuracy: 0.9193000197410583
```

Results and Conclusion

Overall, each method seemed to perform reasonably well. The best methods were the radial basis kernel SVM and k-nearest-neighbors. We see that even pixel data can be interpreted by our models. The neural net was the best of the unsupervised methods, but did not perform as well as either SVM or KNN. Logistic regression performed the worst of any method. It seems that there is no digit that is universally harder to classify than any others, though logistic regression, KNN and the Neural net perform poorly on the digit 8. Overall, we see that with appropriate preprocessing of the data and the right models, we can perform digit classification with good accuracy.

References and Data Sourcing

We used Jupyter notebooks for the code, which can be found online at https://github.com/sureframe846/comp562-final-project

The data is credited to Yann Lecun, Corinna Cortes and Christopher Burges.

The data was obtained from the TensorFlow library.

We used Tensorflow, sklearn, numpy, pandas and matplotlib for this project.