Darrell Nelson II

Data Analytics (IST - 707)

05/28/2019

HW07: Supervised Learning for handwriting recognition

Introduction

Machine learning and deep learning play an important role in computer technology and artificial intelligence. With the use of deep learning and machine learning, human effort can be reduced in recognizing, learning, and predicting patterns that may take decades or centuries to uncover. Machine learning also allows humans to take a look under the hood of how our own brains may generate patterns. What makes a 6 a 6? Do we recognize the top curl and then the bottom loop? What if the bottom loop was a square?

From experience, it is known that when humans read, we typically focus on the first and last letter in the sentence. But, for objects like digits what part of the shape do our brains recognize? How much cropping, reshaping, twisting, and contorting can you do to a digit before it is no longer readable? These are the types of questions that allow us to look deeper into ourselves and understand how we learn. These are also the same questions that must be asked when building a computer model to handle digit recognition.

Digit recognition is the working of a machine to train itself to recognize digits from a myriad of different sources like emails, bank checks, papers, and images. It is extremely useful in day to day life for online handwriting recognition when e-signing documents, recognize license plates, depositing checks, or even filling in numeric entries on electronic forms based on paper forms like when doing your taxes.

Analysis and Models

About the Data

The MNIST (Modified National Institute of Standards and Technology database) database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST (National Institute of Standards and Technology database). The digits have been size-normalized and centered in a fixed-size image. To ensure reproducibility for beginners in data science a small 1,400 row sample was taken for training purposes, and a 1,000-row sample of unlabeled data was used to test the model.

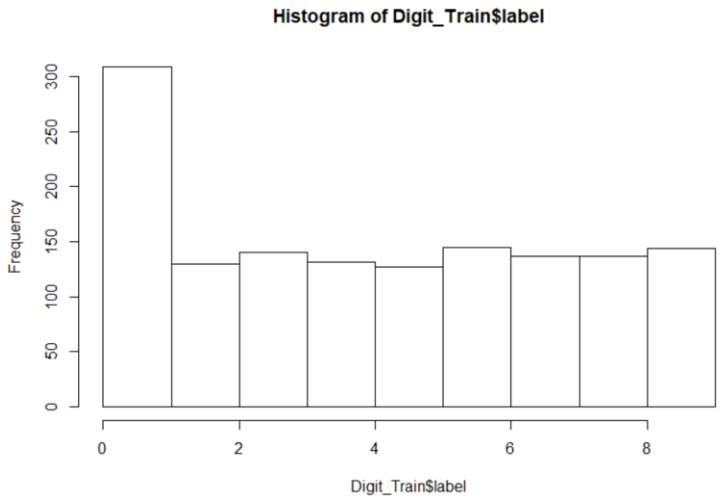
Both the testing and training dataset are comma separated values (CSV) text documents that were formatted into a matrix. Each column represented a different pixel in the picture and its numeric value equated to how bright (turned on) that pixel was. The photographs were in black and white, with 0 = black/turned off and 255 = white/turned all the way on. There are a total of 784 pixels that produce the

handwritten digit photograph. Based off of the position and brightness of each pixel there could be a pattern the computer could recognize between digits written with different scripts.

In the entire training set the total amount of bright white pixels is ~80,000 and the total amount of off pixels (black registers as 0) is ~900,000. With such a small minority of pixels being highly activated it appears that there is a large boundary of darkness that surrounds each digit. This means that there are a lot of pixels that never get turned on.

```
> length(which(Digit_Train > 250))
[1] 79786
> length(which(Digit_Train == 0))
[1] 888028
```

The labeled number frequency of the training set is below:



Al numbers appear to have roughly equal amounts of sample data except for 0 which appears to have ~3 times as many samples. The labels on the training set were converted to factors in order to build classification models. This is doubly important because there is much less class imbalance.

Classification Models

Support Vector Machine

A support vector machine (SVM) is a classification technique that can be used to run supervised machine learning. This model has its roots in statistical learning theory and has shown promising empirical results in many practical applications, including handwritten digit recognition. SVMs also work well with high-dimensional data and avoid the curse of [the] dimensionality problem. Another unique aspect of this approach is that it represents the decision boundary using a subset of the training examples, known as

the support vectors.[1] The decision boundary is chosen by maximizing the distance between the datapoints within a cluster and the boundary itself without sacrificing accuracy. This increases the chance of properly labeling unknown test data they may reside near the boundaries. Two different types of test boundaries were chosen for this dataset: 1) polynomial and 2) linear. The cost parameter (cost of constraints violation) was held constant.

k-nearest neighbor

The k-nearest neighbor algorithm (*k*NN) is a non-parametric method that can be used in supervised machine learning applications. It uses instance-based learning where the model is not computed until after the test data is given. Once the test data is given, it compares the similarity between the test example and the training examples and assigns the label to the test data based on majority vote (which *k*-neighbors are closest). It works well on extremely complex boundary functions as it creates the function by piece-meal (bit by bit). If the number of neighbors (*k*) to be compared to the test data is too small, the model is more susceptible to noise, and if *k* is too large the neighborhood may include too many points from other classes and lead to misclassification.

Random Forest

Random forest, is an ensemble method that manipulates its input features and uses decision trees as its base classifiers. [2] It creates multiple random subsets of features and builds decision trees on each subset which are then combined at the end to make a forest that bases its classification on majority vote.

Results

All models were tested and trained on the same datasets as the models worked on during the previous week to create a fair comparison. The first SVM model used a polynomial kernel with a cost feature of 100. This model was able to accurately predict which hand-written digits belong to which number ~91% of the time!

```
> SVM_Poly
[1] 0.9055794
```

This is astonishing! Recall that the decision tree and Naïve-Bayes models produced last week could only reach ~69% total accuracy. Paying closer attention to the decision boundary by using a subset of the training data has drastically improved the classifier. Breaking the performance out by accuracy per number:

	0	1	2	3	4	5	6	7	8	9
1	0.9803922	0.8627451	0.975	0.9069767	0.8888889	0.7959184	0.8571429	0.9333333	0.9545455	0.9183673

[1] (Tan, Steinbach, & Kumar, 2006)
[2] (Tan, Steinbach, & Kumar, 2006)

The accuracy has increased across the board. The best predictions were for digits 0, 2, and 8. The hardest number to classify is #5, which was the troubling class in all the previous models as well. Despite the performance of this SVM classifier being 11% greater than the top performing decision tree model they both struggled with class 5. Perhaps, it's because the top vertical bar of a 5 is closely related to a 7 or maybe because the bottom half-circle could be mis-identified as any number of bottom circled digits (i.e. 6, 8,3).

Changing from a polynomial kernel to a linear kernel shifts the boundary between classes from a polynomial function to a linear one. Holding all other parameters constant the overall accuracy was calculated as:

```
> SVM_linear
[1] 0.8862661
```

This ~89% overall accuracy is less than the 91% with the polynomial kernel. Inspecting the accuracy by class:

	0	1	2	3	4	5	6	7	8	9
1	0.9814815	0.9347826	0.925	0.8409091	0.8695652	0.8085106	0.875	0.86	0.9230769	0.8461538

The best predictions were for digits 0, 1, and 2. The worst predictor again was #5.

A *k*NN (nearest neighbor) model was also tested on this dataset with an overall accuracy of:

```
> kNN
[1] 0.8261803
```

Using a lazy classifier significantly reduced the accuracy of the model. This loss could be attributed to the intermingling of data points. A nearest neighbor model is sensitive to outliers if the number of nearest neighbors *k* is too large. The rule of thumb for *k* in this model was to use the square root of the number of data points in the training set. Changing the *k* to a lower value may prove meaningful in further studies. Accuracy by class:

	0	1	2	3	4	5	6	7	8	9
1	0.9607843	0.6027397	1	0.755102	0.8809524	0.9375	0.86	0.8510638	0.9705882	0.7288136

The nearest neighbor model was able to predict class 2 with 100% accuracy. It also fared well predicting classes 0, 8, and 5. This is of note because all the previous models struggled predicting class 5. The class five data points appear to be in the same *neighborhood,* however, are generally classified in different groups when a cost function is used to create a boundary. Perhaps an ensemble model that implements both lazy classifiers and standard supervised learners can improve on this accuracy.

Random Forest is an ensemble classifier that runs multiple decision trees on the dataset and chooses which classifiers are the best based on majority vote. A single decision tree (pruned or unpruned) run on

this dataset achieved an accuracy of ~69%; using a random forest to combine 500 decision trees yielded a result of:

```
> RF
[1] 0.8862661
```

Accuracy by class:

	0	1	2	3	4	5	6	7	8	9
1	1	0.8148148	0.9268293	0.8043478	0.8333333	0.8666667	0.8723404	1	0.9047619	0.86

Random Forest classification with 500 trees was able to correctly predict all the 0 and 7 classes in the test data set. The lowest performing classes were 1 and 3.

Evaluating the performance of all 8 models that were trained and tested on this dataset based on their overall test accuracy and accuracy by class yields the following conclusions.

```
             overall_acc
DT             0.6857143
DT_pruned      0.5828571
NBe1071        0.4585714
NaiveB         0.4721030
SVM_Poly       0.9055794
SVM_linear     0.8862661
kNN            0.8261803
RF             0.8862661
```

Overall, decision trees and Naïve-Bayes classifiers performed 20% worse than the SVMs, kNN, and random forest models. Using joint probabilities (NB) to determine how likely a data point is within a singular group appears to be the worst methodology to predict handwritten digits. Using a support vector to create the boundaries between classes seems to be the highest performing approach. SVMs, kNN, and random forests all have comparable results therefore slight parameter tweaking may bring any of these three techniques to the forefront. Compartmentalizing the models' performances based on individual classes yields:

	DT	DT_pruned	NBe1071	NaiveB	SVM_Poly	SVM_linear	kNN	RF
0	0.7375	0.7375	0.825757575757576	0.878048780487805	0.980392156862745	0.981481481481482	0.96078431372549	1
1	0.806818181818182	0.669767441860465	0.423076923076923	0.451612903225806	0.862745098039216	0.934782608695652	0.602739726027397	0.814814814814815
2	0.654411764705882	0.398009950248756	0.84375	0.888888888888889	0.975	0.925	1	0.926829268292683
3	0.819444444444444	0.53030303030303	0.853658536585366	0.769230769230769	0.906976744186046	0.840909090909091	0.755102040816326	0.804347826086957
4	0.769911504424779	0.559006211180124	0.459459459459459	0.666666666666667	0.888888888888889	0.869565217391304	0.880952380952381	0.833333333333333
5	0.491428571428571	0.371794871794872	0.5	0.5	0.795918367346939	0.808510638297872	0.9375	0.866666666666667
6	0.572368421052632	NaN	0.728813559322034	0.75	0.857142857142857	0.875	0.86	0.872340425531915
7	0.846153846153846	0.68	0.702702702702703	0.695652173913043	0.933333333333333	0.86	0.851063829787234	1
8	0.62992125984252	0.674418604651163	0.251231527093596	0.194690265486726	0.954545454545455	0.923076923076923	0.970588235294118	0.904761904761905
9	0.656976744186046	0.690647482014389	0.293193717277487	0.385321100917431	0.918367346938776	0.846153846153846	0.728813559322034	0.86

	Highest Accuracy by Model and Class							
Class	DT	DT_pruned	NBe1071	NaiveB	SVM_Poly	SVM_linear	kNN	RF
0	0.7375	0.7375	0.825758	0.878049	0.9803922	0.9814815	0.960784	1
1	0.806818	0.6697674	0.423077	0.451613	0.8627451	0.9347826	0.60274	0.814815
2	0.654412	0.39801	0.84375	0.888889	0.975	0.925	1	0.926829
3	0.819444	0.530303	0.853659	0.769231	0.9069767	0.8409091	0.755102	0.804348
4	0.769912	0.5590062	0.459459	0.666667	0.8888889	0.8695652	0.880952	0.833333
5	0.491429	0.3717949	0.5	0.5	0.7959184	0.8085106	0.9375	0.866667
6	0.572368	NaN	0.728814	0.75	0.8571429	0.875	0.86	0.87234
7	0.846154	0.68	0.702703	0.695652	0.9333333	0.86	0.851064	1
8	0.629921	0.6744186	0.251232	0.19469	0.9545455	0.9230769	0.970588	0.904762
9	0.656977	0.6906475	0.293194	0.385321	0.9183673	0.8461538	0.728814	0.86

Formatting the above R output into Excel highlights the performance of each class in each model compared to all the other models and classes. There are three key takeaways: 1) There is a clear distinction between the performance of the first 4 models versus the last 4. 2) “SVM_Poly” outperforms the rest of the field in terms of accuracy. 3) The handwritten digits that top 4 models (4 on far right) had the easiest time classifying were 0, 2, and 8.

The last takeaway is corroborated when the data is filtered out by ranking the accuracy of each class in their specific model:

	Highest Performing Accuracy by Class within model							
	DT	DT_pruned	NBe1071	NaiveB	SVM_Poly	SVM_linear	kNN	RF
0	0.7375	0.7375	0.825758	0.878049	0.9803922	0.9814815	0.960784	1
1	0.806818	0.6697674	0.423077	0.451613	0.8627451	0.9347826	0.60274	0.814815
2	0.654412	0.39801	0.84375	0.888889	0.975	0.925	1	0.926829
3	0.819444	0.530303	0.853659	0.769231	0.9069767	0.8409091	0.755102	0.804348
4	0.769912	0.5590062	0.459459	0.666667	0.8888889	0.8695652	0.880952	0.833333
5	0.491429	0.3717949	0.5	0.5	0.7959184	0.8085106	0.9375	0.866667
6	0.572368	NaN	0.728814	0.75	0.8571429	0.875	0.86	0.87234
7	0.846154	0.68	0.702703	0.695652	0.9333333	0.86	0.851064	1

8	0.629921	0.6744186	0.251232	0.19469	0.9545455	0.9230769	0.970588	0.904762
9	0.656977	0.6906475	0.293194	0.385321	0.9183673	0.8461538	0.728814	0.86

The table above also points out which models did the best on which classification which is extremely useful when looking to manually develop an ensemble method. Different classifiers can be chosen based on their relative performance on each class. For example, although kNN and RF had identical overall accuracies they varied in which classes they got correct. Therefore, an ensemble model that utilizes a RF for digits 0, 7, and 9 and a kNN for digits 5 and 8 would have improved overall accuracy than either of the two alone.

To best picture which ensemble would maximize the overall accuracy of the tested models; the accuracies for each class were compared across all models:

	Highest Performing Accuracy by class							
	DT	**DT_pruned**	**NBe1071**	**NaiveB**	**SVM_Poly**	**SVM_linear**	**kNN**	**RF**
0	0.7375	0.7375	0.825758	0.878049	0.9803922	0.9814815	0.960784	1
1	0.806818	0.6697674	0.423077	0.451613	0.8627451	0.9347826	0.60274	0.814815
2	0.654412	0.39801	0.84375	0.888889	0.975	0.925	1	0.926829
3	0.819444	0.530303	0.853659	0.769231	0.9069767	0.8409091	0.755102	0.804348
4	0.769912	0.5590062	0.459459	0.666667	0.8888889	0.8695652	0.880952	0.833333
5	0.491429	0.3717949	0.5	0.5	0.7959184	0.8085106	0.9375	0.866667
6	0.572368	NaN	0.728814	0.75	0.8571429	0.875	0.86	0.87234
7	0.846154	0.68	0.702703	0.695652	0.9333333	0.86	0.851064	1
8	0.629921	0.6744186	0.251232	0.19469	0.9545455	0.9230769	0.970588	0.904762
9	0.656977	0.6906475	0.293194	0.385321	0.9183673	0.8461538	0.728814	0.86

Based on the above rankings the first four models can be disregarded altogether, and an ensemble model can be created utilizing the last four. Picking out the highest performing models for each class yields the table below:

Model	**RF**	**kNN**	**SVM_linear**	**SVM_poly**
Class	0,7	2,5,8	1,6	3,4,9

In conclusion, although the “SVM_poly” is the best stand-alone model, it only predicts 3 classes better than the rest of the field. To further improve the testing accuracy multiple models should be incorporated to make up for the others’ deficiencies.

Conclusion

Various models have been used to accurately predict the correct class for handwritten digits. Based off the above analysis SVMs, *k*NN, and RFs are the best models to use “out of the box” to get a high

accuracy with a support vector machine with a polynomial kernel edging out the other classifiers with an accuracy of ~91%. Although, this model still has a long way to go to be worthy of the Kaggle top 30, model accuracy of 100%, the groundwork has been laid to achieve even further heights.

Using ensemble techniques that bring different models together is the next iteration to achieving higher accuracy. In the Results section it is detailed how although the SVM with a polynomial kernel was the best model, there were weaknesses in that classifier that other techniques were strong at. By combining forces, models can cover for each others' weaknesses.

In conclusion, SVMs appear to be the best model to use for handwritten digit recognition based off this dataset. However, there is no need to just stop there. With so many different parameters that can be optimized on so many different models the different combinations and possibilities feel almost endless. This is why data science is perceived more as an art form versus an exact science, there are so many right answers to choose from. It's all about how you get there.