February, 2020, Cleveland, Ohio USA

Data Mining Project 1

Project 1 Report

Chen Guo
Computer Science
Case Western Reserve University
Cleveland Ohio U.S.A
cxg451@case.edu

Yuzhou Cao
Computer Science
Case Western Reserve University
Cleveland Ohio U.S.A
yxc1426@case.edu

McKenzie Hawkins
Computer Science
Case Western Reserve Univ.
Cleveland Ohio U.S.A
mjh244@case.edu

ABSTRACT

A common issue in the fields Computer Science and Data Mining is the classification of data, including objects and images. In order to explore classification methods, we used a spreadsheet which contained brightness values for each pixel on a 28x28 image. Each image was of a hand drawn number, and the spreadsheet included labels that indicated which number the image displayed. Our team programmed six classification models using the existing scikit-learn library, and trained these models on the spreadsheet. This report will detail which methods we used, how we used them, and the accuracy achieved through such methods.

1.0 **Methods**

In order to perform our classification experiment, we used 6 different classification models. These models, listed below, were programmed in Python, using Google Colab, and the scikit-learn libraries. Some of these models were chosen due to their expected performance, while others were chosen due to their expected development time.

1. Support Vector Machine
2. K-Nearest Neighbors
3. Neural Network
4. Random Forest
5. Decision Tree
6. Multinomial Naive Bayes

Each of these models were trained and tested on the image data available.

1.1 **Training**

The training of each model was performed through the use of the scikit-learn libraries. These libraries are commonly used for machine learning and data mining purposes such as classification, regression, clustering, preprocessing, etc.

1.1.1 Support Vector Machine

To build the SVM model, we used the sklearn.svm library. For the svm.SVC( ) function which builds the model, we take all the default values. Also, training data is divided into three parts, two for training and one for testing. Function MinMaxScaler is imported from sklearn.preprocessing to do scaling,to be specific, to make the feature value ranging from -1 to 1 by setting the parameters. Finally, function accuracy_score is imported from sklearn.metrics to get the final accuracy.

1.1.2 K-Nearest Neighbors

We noticed that the library sklearn.mdel_selection would help a lot in splitting training and testing data and doing cross validation.So we used the function train_test_split( ) and set the parameter test_size to 0.25(i.e., 25% to test and the rest to train the model). What's more, the function RandomizedSearchCV( ) from the very library is used to do a 3-fold cross validation, by setting parameter cv to 3 and taking a 100-iteration(n_iter=100). Similarly, the KNN model is built with the function KNeighborsClassifier( ) with its default value parameters. Finally, accuracy is acquired with function accuracy_score( ) is imported from sklearn.metrics.

1.1.3 Neural Network

We use sklearn.neural_network library to build a 4-layer neural network. The network has the following

schema: 784-200-50-10. We split the data into train set(70%) and testing set(30%), the optimizer is adam(by default) with learning rate 1e-5. The maximum iteration is 6000. Then use neural_network.MLPClassifier() to build the model.Once the model was built, we tested it on the appropriate third of the training data, holding the handwritten number labels as the key and comparing the results. After the model is completed testing, the accuracy was displayed by using the build-in method score to Return the mean accuracy on the given test data and labels.

1.1.4 Random Forest

We use sklearn.ensemble library to build a random forest classifier . The max depth of the forest is set to 10, in this model We split the data into train set(70%) and testing set(30%), The number of decision trees in this model is set to be 100. Then use sklearn.ensemble.RandomForestClassifier to build the model. Once the model was built, we tested it on the appropriate third of the training data, holding the handwritten number labels as the key and comparing the results through predict() function.After the model is completed testing, the accuracy was displayed by calling metrics.accuracy_score() function.

1.1.5 Decision Tree

To build the Decision Tree model we used the sklearn.tree library. Before creating the model we split the training data into 2 parts using the train_test_split() function, with two thirds of the data being used for training, while the other third was used for testing. Once the data was split, we used the tree.DecisionTreeClassifier() function to build the model. Once the model was built, we tested it on the appropriate third of the training data, holding the handwritten number labels as the key and comparing the results. After the model is completed testing, the accuracy was displayed.

1.1.6 Multinomial Naive Bayes

To build the Multinomial Naive Bayes model we used the sklearn.naive_bayes library. Before creating the model we split the training data into 2 parts using the train_test_split() function, with two thirds of the data being used for training, while the other third was used for testing. Once the data was split, we used the MultinomialNB() function to build the model. Once the model was built, we tested it on the appropriate third of the training data, holding the handwritten number labels as the key and comparing the results. After the model is completed testing, the accuracy was displayed.

1.2 **Results**

Listed below are the accuracy ratings of our models.

Classification Model	Accuracy (%)
Support-Vector Machine	96.65
K-Nearest Neighbors	95.5
Neural Network	93.9
Random Forest	93.7
Decision Tree	79.4
Multinomial Naive Bayes	82.37

Included below is a figure of bar chart that indicate the performance of each model when tested on the spreadsheet.

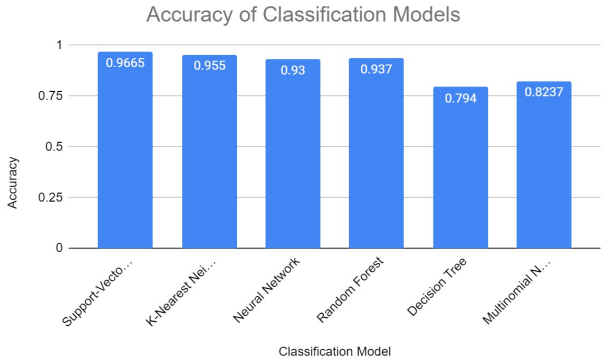

Figure 1: **Accuracy of the Six Selected Models**

Due to its accuracy rating, the Support-Vector Machine appears to be the best model out of the six we programmed for differentiating handwritten numbers. The Support-Vector Machine was likely the most accurate because the SVM model is able to use a boundary to differentiate between various handwritten numbers.

1.3 **Participation**

Listed below is the participation of each member.

Chen Guo - 33.3%
Yuzhou Cao - 33.3%
McKenzie Hawkins - 33.3%

Each team member programmed two of the classification models, and contributed to the report. All team members agree with the specified effort.

1.4 **Conclusion**

To conclude, our team successfully programmed six classification models that were used to identify the images of handwritten numbers. Each of these classification models had an accuracy of over 75%. The Support-Vector Machine was the most accurate at identifying images of handwritten numbers with an accuracy of about 96%. The Support-Vector Machine was likely the most accurate because of its ability to use a boundary to differentiate between various handwritten numbers.

REFERENCES

[1] 2020. *Scikit-Learn*. Accessed February 14. https://scikit-learn.org/stable/index.html.
[2] "1.4. Support Vector Machines." 2020. *Scikit-Learn*. Accessed February 14. https://scikit-learn.org/stable/modules/svm.html.
[3] "1.6. Nearest Neighbors." 2020. *Scikit-Learn*. Accessed February 14. https://scikit-learn.org/stable/modules/neighbors.html
[4] "1.17. Neural network models (supervised)." 2020. *Scikit-Learn*. Accessed February 14. https://scikit-learn.org/stable/modules/neural_networks_supervised.html.
[5] "1.11. Ensemble methods." 2020. *Scikit-Learn*. Accessed February 14. https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees.
[6] "1.10. Decision Trees." 2020. *Scikit-Learn*. Accessed February 14. https://scikit-learn.org/stable/modules/tree.html.
[7] "1.9. Naive Bayes." 2020. *Scikit-Learn*. Accessed February 14. https://scikit-learn.org/stable/modules/naive_bayes.html.