DIGIT RECOGNIZER

CS 6375 Final Report

Yang Gao, yxg122530@utdallas.edu

Mingyue Sun, mxs151730@utdallas.edu

Yue Sun, yxs146930@utdallas.edu

CS 6375 Final Project Report

Part 1. Introduction

1-1 Motivation and Application

Application 1: Suppose somebody has a house and sometime he is out of home, for a sense of security consideration, if there are stranger' car stopped in front of our house for more than 1h (more than a proper time), the plate will be sent to our cellphone, that gives the house owner a alert or reminder. Of course the plate number captured by satellite can be sent by format of image. In reality the demand is huge, image transmission can be super costing, if we have algorithm to recognize the plate image into number, the transmission scale be can greatly reduced to bytes.

Application 2: Mobile Payment is more and more popular, expect the app with connect to our bank account, we expect something new, for example, as we know, the mobile camera can scan the QR code, how about the account number on our debt/ credit card?

For many areas, we need the camera to detect the number from video or image frame, then recognize them into numbers, not only for saving time by taking advantage of machine, but also if processing properly, the accuracy is higher then human reading.

1-2 Results and conclusions

We will provide some approach to

1). For both image or numeric input format, provide method with respect to robust of noise, rotation, translation, scale of image.

2). How to train the classifier from both image or their numeric pixels, how to convert between them.

3). Algorithm including SVM, Naïve Bayes in R, PCV with KNN and Artificial Neural Network, and how to validation, test and improve them. Also some plot result and analysis, this is the core part of our project.

4) Algorithm how to remove noise, image compression, and increase robust to illumination and etc. preprocessing with data. This step is necessary, for in our real life, the most part of application related to digit recognition is from and is for image.

Part 2. Problem Definition and Algorithm

2.1 Task Definition

1. Image compression in Java without any third party library, in order to generate 28*28 pixel image

2. Image filer in Java without any third party library, on 28*28 pixels.

3. Convert training/testing image to 285 attribute + 1 class Label with instance for the following R processing.

4. Deep Learning
Build Neuron Network with multiple hidden layers (800) with training dataset and predict by using the test dataset.

5. Gaussian Naïve Bayes
Build Gaussian Naïve Bayes model by calculating relative probabilities of training dataset. And using the test dataset to predict accuracy.

6. KNN
Build K-NN model(K=1) based on given training dataset and predict for the test dataset. Show the result and plots

7 Random Forest
Build Random Forest model based on given training dataset and predict for the test dataset. Show the result and plots.

2.2 Algorithm Definition

2.2.1 Image Preprocessing Algorithms

1. For Image compression, there are many algorithms, here I use a rational operation.

2. For image filter against, I use a 3*3 window with moving average.

3. Convert training/testing image to 284 attribute + 1 class Label with instance for the following R processing.

2.2.2 Learning Algorithms

1. K nearest neighbors (K-NN)

    - Learning Algorithm:
        - Store all the training examples.

    - Prediction Algorithm:
        - For the given test example x, find k training examples $\{(xi,yi)\}_{i=1-k}$ that are the nearest to x
        - If classification problem, return majority class among the k examples
        - If regression problem, return average y value of the k examples.

2. Random Forest
    - Learning Algorithm:
        - Draw n_{tree} bootstrap samples from the original data.
        - For each of the bootstrap samples, grow an un-pruned classification or regression tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample m_{try} of the predictors and choose the best split from among those variables. (Bagging can be thought of as the special case of random forests obtained when m_{try}=p, the number of predictors)
        - Predict new data by aggregating the predictions of the n_{tree} trees (i.e. majority votes for classification, average from regression)
    - Prediction Algorithm:
        - At each bootstrap iteration, predict the data not in the bootstrap sample using the tree grown with the bootstrap sample.
        - Aggregate the OOB predictions. On the average, each data point would be out-of-bag around 36% of the times, so aggregate these predictions. Calculate the error rate, and call it the OOB estimate of error rate.
3. Gaussian Naïve Bayes
    - Learning Algorithm:
        - Importing training dataset for the following probability calculation.
        - For each attribute, calculate the likelihood, label prior, predict prior.
        - Basing on the results above, calculate posterior probabilities for each attribute.
        - Gaussian Naïve Bayes Model has been built.
    - Prediction Algorithm:
        - Importing testing dataset instances and predict the label (0-9) by using the model built.
        - For each instance, calculate posterior probability of each label.
        - Assigning the label with largest posterior probability to the instance.
        - Calculating accuracy, precision, recall and F-score of each label.
4. Deep Learning:
    - Learning Algorithm:
        - Importing training dataset instances as the input layers in the neuron net.
        - Assigning weights and calculating the training instances and generate new nodes and form new layer. Those layers generated between input layer and output layer/node are called hidden layers.
        - Repeat generating new hidden layers and finally generate new output node/layer.
        - Starting back propagation procedure. Iterating until converged or reach the iteration time limitation.
    - Prediction Algorithm:
        - Importing testing dataset instances and predict the label (0-9) by using the model built.
        - Calculating accuracy, precision, recall and F-score of each label.

3. Image, Raw Data Conversion and Pre Processing

3.1 Significance

Our topic is digit recognition, sometimes, the test is presented in image, or frames of video, we need to convert them to matrix in order to apply the following algorithm. This part is talking about how to remove the noise, how to visualize the training, testing picture, and scale, filter them, thus make our data robust to variant format of testing.

3.2 Details about task from 1-3 in 2.1 Task Definition

1) Our matrix for training we use, is a 785* n, where n is the number of training data, 785 attribute includes a a class label and 784=28*28 pixel values. In most of times, we took picture from camera, we get a nicer picture than 28*28 pixels, but that is not so friendly to our training format, we need rescale them. I use a rational operation, find the new pixel location in 28*28 image corresponding to the original one, which we assume is larger than 28*28. By doing this, we loss some information of original image. we use java to implement this, it is in the source code folder/ preprocessing/ ImageCompress.java

running method:

java ImageCompress.java <inputImage> <outputImage>

2) Filter is used to remove noise, I apply a moving a average method, specially, there is a 3*3 windows moving from the left + 1 to right -1 and then up+1 to down – 1, we calculate the 9 numbers' average, assign it the center part of image, as its new pixels, by doing this, noise data is blurred. Obviously, there is a drawback of this method, the edge of the image, can not be processed, for dealing with this issue, the are two approach, the first one is assign its neighbour's value to it. Or just compute the average with limited but less than 9 numbers sum. we implement it by java, the source code is in /preprocessing/Filter.java

running method:

java Fliter.java <inputImage> <outputImage>

3) convert image to matrix, under the situation of predicting a folder of image (all the image are for numbers), after our operation of converting it to 28*28 pixels in step 1, and remove noise in Step 2, we can call ImageMatrixConverter to convert image into matrix, by having matrix in CSV file, we can later apply it in R which we will talk later, there is also a method called converterM2I which you can visual the training or testing data we already have as matrix. The problem we need to realize is, the training and testing data originally given is descripting the white number on a black background, if our image is not in this form, for example, we have black number on a while paper, we need to reverse their color to predict and label them. Fortunately, this is not difficult, in / preprocessing/ ImageMatrixConverter.java, our method has a Boolean parameter which distinguish this situation.

4. Experimental Evaluation

4.1 Methodology

- K-NN
  - Package Required:
    - r-package: class
  - Function Used:
    - knn: k-nearest neighbor classification for test set from training set
  - Parameters:
    - train: matrix or data frame of training set cases
    - test: matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case
    - cl: factor of true classifications of training set
    - k = 1: number of neighbors considered
    - probe = T: If this is true, the proportion of the votes for the winning class are returned as attribute prob.
- Random Forest:
  - Package Required:
    - r-package: randomForest
  - Function Used:
    - RandomForest: randomForest implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points.

- Predict: prediction of test data using random forest.

➢ Parameters:

- data: an optional data frame containing the variables in the model. By default, the variables are taken from the environment which randomForest is called from.
- ntree = 20: Number of trees to grow. This should not be set too small a number, to ensure that every input row gets predicted at least a few times.
- object: an object of class randomForest, as that created by the function randomForest
- newdata: a data frame or matrix containing new data.

▪ Gaussian Naïve Bayes:

➢ Package Required:

- r-package: e1071

➢ Function Used:

- naiveBayes: Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.
- Predict: prediction of test data using naïve bayes.

➢ Parameters:

- formula: A formula of the form class ~ x1 + x2 + ... Interactions are not allowed.
- data: Either a data frame of predictors (categorical and/or numeric) or a contingency table.
- Laplace=5: positive double controlling Laplace smoothing. The default (0) disables Laplace smoothing.

▪ Deep Learning:

➢ Package Required:

- r-package: h2o

➢ Function Used:

- h2o.importFile: Imports files into an H2O cloud. Predict: prediction of test data using naïve bayes.
- h2o.deeplearning: Builds a feed-forward multilayer artificial neural network on an H2OFrame.
- h2o.predict: Predict output label of the test data.

➢ Parameters:

- x: A vector containing the character names of the predictors in the model.
- y: The name of the response variable in the model
- training_frame: An H2OFrame object containing the variables in the model.
- activation: A string indicating the activation function to use. Here we use Rectifier With Dropout method.
- input_dropout_ratio: A fraction of the features for each training row to be omitted from training in order to improve generalization (dimension sampling). We used 0.2.
- fhidden_dropout_ratios = Input layer dropout ratio (can improve generalization) specify one value per hidden layer, defaults to 0.5.
- balance_classes = TRUE: Balance training data class counts via over/under-sampling (for imbalanced data).
- hidden: size of hidden layers. We used 800 hidden layers each of which has 800 nodes.
- epochs = 500 : iteration time.

4.2 Results

- K-NN:
  Accuracy: 96.7%

	0	1	2	3	4	5	6	7	8	9
Precision	0.982 0574	0.968 2875	0.984 9537	0.949 2925	0.977 5561	0.965 2870	0.968 4211	0.954 8533	0.977 6610	0.938 9671
Recall	0.985 5942	0.989 2009	0.963 7599	0.954 9229	0.961 9632	0.957 6159	0.986 8892	0.971 2974	0.935 8491	0.953 5161
F score	0.983 8226	0.978 6325	0.974 2416	0.952 0993	0.969 6970	0.961 4362	0.977 5679	0.963 0051	0.956 2982	0.946 1857

- Random Forest:
  Accuracy: 97.6%

	0	1	2	3	4	5	6	7	8	9
Precision	0.989 2729	0.998 9119	0.966 2162	0.978 1818	0.971 0843	0.955 6714	0.978 5714	0.978 0347	0.969 5044	0.972 5864
Recall	0.996 3986	0.991 3607	0.971 6874	0.957 2954	0.988 9571	0.970 8609	0.979 7378	0.971 2974	0.959 7484	0.972 5864
F score	0.992 8230	0.995 1220	0.968 9441	0.967 6259	0.979 9392	0.963 2063	0.979 1543	0.974 6544	0.964 6018	0.972 5864

- Gaussian Naïve bayes
  Accuracy: 53.47%

	0	1	2	3	4	5	6	7	8	9
Precision	0.748 7127	0.761 7823	0.894 4186	0.655 4810	0.807 2289	0.531 25	0.575 2727	0.951 2195	0.285 3815	0.370 4567
Recall	0.872 7491	0.96	0.212 91053	0.347 56821	0.082 20859	0.022 51656	0.942 78903	0.268 65672	0.616 35220	0.947 55662
F-Score	0.805 98670	0.849 49833	0.342 44080	0.454 26357	0.149 22049	0.043 20203	0.714 54381	0.418 97941	0.390 12739	0.532 66332

- Deep Learning
  Accuracy: 96%

	0	1	2	3	4	5	6	7	8	9
Precision	1	0.958333	0.944	0.955556	0.9603960	0.955556	0.9795918	0.9708738	0.9662921	0.9117647
Recall	0.9906542	0.9583333	0.9516129	0.955556	0.9509804	0.9662921	0.9896907	0.9523810	0.9247312	0.9587629
F-Score	0.9953052	0.9583333	0.9477912	0.955556	0.9556650	0.955665	0.9846154	0.9615385	0.9450549	0.9346734

4.3 Discussion

Given all the experiment result data above, we can conclude that K-NN, Random Forest and Deep Learning algorithms are very powerful in number recognition. Gaussian Naïve Bayes is not suitable for this task.

In the aspect of cost of time, Deep Learning algorithm needs very long time to train the model. The K-NN barely need time to train, but when predicting, it takes a long time. Since it need to compare each instance in test dataset to each instance in training dataset, and for each comparison, 784 calculations are made. Random Forest performs very well, also, its training is very fast. In our project, this is the best training algorithm.

The following is details of our discussion and analysis for the performance of each learning algorithm we chose for the project.

1.

K-NN and Random Forest are good classifiers for this problem, both giving high accuracy ~ 95%.

Since the image is composed of white numbers with gray-scale >0 and large black backgrounds with gray-scale =0, it is very easy for K-NN record the pattern of the digits with low noise even though their orientation or shape is varied. Different digit with different orientation has its own pattern. If we have enough training data, then for every test data, we can find a pattern that is "close" to it (orientation and value is correct one), so that it can get the correct label (to measure the distance, we need to take all the pixels into account, thus "2" and "5" is pretty different). That's why we get a high accuracy.

Similar idea applied to random forest. Each tree in the forest takes only part of attributes of the total available attributes. Thus it focuses more on some specific attributes (local patterns). Combining all of these trees, we get good description of the pattern of the data instances.

2.

Precision refers to how often a case the algorithm predicts as belonging to a class actually belongs to that class.

Recall refers to the proportion of data in a class the algorithm correctly assigns to that class.

F score produce a weighted average of both precision and recall, where the highest level of performance is equal to 1 and the lowest 0.

Considering the above Precision-Recall-F score table. We find:

1) For K-NN: 1 and 7, 3, 6 and 9 has relatively low precision and recall. That means, knn classifier made several possible misclassifications:
   1 is misclassified with 7
   3 is misclassified to 8
   6 is misclassified with 9
   (because the digit may have different orientations)

2) For random forest: 2 ,3 and 5 have relatively low precision and recall. This is probably because all these 3 numbers contain a partial arc which might be selected as split attributes for single trees in forest. This could cause misclassification problem for these 3 numbers.

3.

Deep learning is a very powerful method in digit recognition. Comparing with normal neuron net-work algorithm, deep learning can build a more accurate prediction model. Because the quantity of hidden layers enables this algorithm to dig deeper in the characteristics of different numbers in the pictures. However, the training is really time-consuming. In this experiment, we used partial data (train-9000, test-1000) to build model. The estimated time for training full dataset would be more than 20 hours! But the result still performs well.

4.Gaussian Naïve Bayes:

Gaussian Naïve Bayes performs poorly in this experiment. The reason behind this may lies in the fact that the Naïve Bayes algorithm is based on the probability calculations. It can reveal the relation between factors. However, for the picture analysis, the noise may be largely involved in the model. For example, many numbers in the pictures are placed horizontal. Instead of revealing the topology relationships in the graphic, the model will concentrate more on the probability. Thus, the Naïve Bayes algorithm is not suitable for number recognition tasks.

5. Future Work and Conclusion

5.1 Future Work

The major drawback of our project preprocessing of data and our classify is done by different language, the image/ data conversion including image/data preprocessing is done by java, all classifier are done in R language, so, if we want to predict a video, actually it is widely used in our society, for example on living vehicle plate streaming on the street, we need to manually convert it to frames, then use our java code to convert it to CSV ready for R. this is not convenient.

The possible approach is to apply Python or CPP with some additional library, there are some machine learning library in Python, and it has a nice GUI support, the main idea to convert both part into uniform language, so we can have user friendly GUI, use can choose to input a picture from either hard disk or camera, a video from either a hard disk or living camera, and interactively, we give the labeled result.

Another thing for future work we want to mention here is, image separation, for example, “58601” is a zipcode which comes together on one image, or from a video, in order to input it to our classifier, we need separate it as 5, 8, 6, 0, 1, regarding to method, we can use color based segmentation, or hough detetion for lines.

5.2 Conclusion

We successfully deal with the input training data with high accuracy with different methods.

We give analysis to our machine learning method.

We successfully correlate raw data with image processing

We come out a work flow with make it a great application.