AI CW1 – Report

Grigoris Zinonos

34715967

**Abstract:**

The purpose of this report is to describe the content of the EMNIST Letters database, investigate the training, evaluation and results of 3 Machine Learning classification algorithms, discuss the selected model, and describe the performance of the selected model from an error-analysis perspective.

**EMNIST letters database:**

The EMNIST letters database is a set of character digits derived from the NIST Special Database. The database is split into 2 smaller databases, the training data and the test data. This split makes it easier to train and test a model since you do not have to deal with the process of separating data to training and testing sets. Both the train and test datasets consist of 2 other tables. One table is called images and it is a 124800x784 matrix. This table hold 124800 representation of handwritten letters and therefore is has 1248000 rows. Each row represents an image. Each image size is 28x28 and therefore a row with 784 columns is used to represent the value of each pixel an image. The other table is called labels. The purpose of the content of this table is to map each of the images with the actual letter that is represents. For example; if the first row of the images table represents an ‘b’, the first row of the labels table will hold the value 2. Both train and test datasets have their label mapping table which makes the evaluation and training of classifications models easier. The EMNIST letters dataset has 26 classes with 4800 observations for each class (26 letters of the alphabet). Each observation has 784 features since each pixel value of a 28x28 image is a feature

Overview of EMNIST letters:

images

TRAIN

labels

EMNIST ltrs DB

images

TEST

labels

**Models Training and Evaluation:**

For the purpose of this task I trained and evaluate 3 classification models using the K-nearest neighbours’ algorithm, Naïve Bayes algorithm, and the Decision Tree algorithm. Since this classification task involves 26 classes, I decided to go with the above-mentioned algorithms because they support multi-class classification. Some of the most well-known algorithms work better for binary classification and it would take much longer to implement 26 binary classifications, train them and evaluate them. An example of such algorithm is the Support Vector Machine (SVM). Additionally, to train an SVM classification algorithm I would need to convert the pixel arrays to grayscale for them to be used as training input for the model. I tried this approach, but I decided to not use the SVM algorithm because the convention process was extremely time consuming.

I did not use cross validation for the training of the models because the dataset is already split into train and test, which means that the data used for training and testing are non-overlapping and the results are not biased.

In order to make the training of the model easier, I isolated the train set from the EMNIST letters dataset.

//extract features from train data

images=dataset.train.images

//extract labels from train data

labels=dataset.train.labels

All 3 algorithms were quite straight forward to train because the fitcknn function, the fitcnb function, and the fitctree function only need the images data as the first argument and the labels as the second argument.

The only difference is that the Naïve Bayes and the Decision Tree train function needed the images to be converted to double to be used as training input.

knnmodel= fitcknn(images,labels)

NBmodel = fitcnb(double(images), labels, "DistributionNames","mn")

DTmodel = fitctree(double(images), labels)

In order to evaluate each model, I used time to train metric, time to predict metric and accuracy. To calculate the time needed for training and prediction I used the tic – toc method on MatLab. This method returns the time elapsed between the tic and the toc;

Tic

TrainModel(Arg1, Arg2)

Toc

Output: Elapsed time is x.xx seconds.

To calculate the accuracy of each model I compared the predicted label with the actual lebel;

predicted=predict(model, images)

evaluate = [predicted, actual\_labels]

evaluate=[evaluate (evaluate(:,1)==evaluate(:,2))]

//Column 1 is predicted labels, column 2 is the actual label

//After this commant, evaluate will have a 3rd column with 0 and 1 depending on if the prediction was correct.

accuracy= sum(evaluate(:,3))/length(evaluate(:,3))

**Evaluation Results and Model selection:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **Train Time** | **Predict Time** | **Accuracy** |
| K-NN | 0.9 s | 1457 s | 86% |
| Naïve Bayes | 2 s | 45 s | 58% |
| Decision Tree | 94 s | 2 s | 72 % |

Given that all models were equally easy to implement I decided to select the Decision Tree classification algorithm because it gave relatively high accuracy percentage with quite short training and prediction time.

Moreover, the accuracy of the Decision Tree classifier can improve if we provide a bigger training set.

**Error Analysis on Decision Tree classification:**

By analysing the Confusion Matrix of the selected model, we conclude to the fact that the model demonstrates issues on classifying class 7 **(G)** from class 17 **(L)**, and class 9 **(I)** from class 12 **(L)**.

This observation is reasonable because those letters look similar to each other. Additionally, if we consider that every one of us has its own handwriting character, it is inevitable that classification between characters that look alike is challenging.

