University of Alberta CMPUT 566 Mini-Project: Evaluation of classification task on hand-written digit & letter recognition

Nikoo Sadat Aghaei — Mohsen Amir Sattarifardi — M Bedir Tapkan
nikooaghaei@ualberta.ca — amir@ualberta.ca — tapkan@ualberta.ca

Introduction

In this report, we will be evaluating the results of several algorithms on the hand-written digit & letter recognition task. The main task is to evaluate the algorithms and practice statistical significance and learn how one should decide on which algorithm is better than the other.

Data Description & Preprocessing

The dataset we will be using belongs to **EMNIST**, which is an extended version of the popular handwritten digits dataset (LeCun and Cortes 2010).

EMNIST (Cohen et al. 2017) has an extension over MNIST in terms of the data size for each digit, and also it includes hand-written capital and lowercase letters in the English alphabet. (Fig.1)

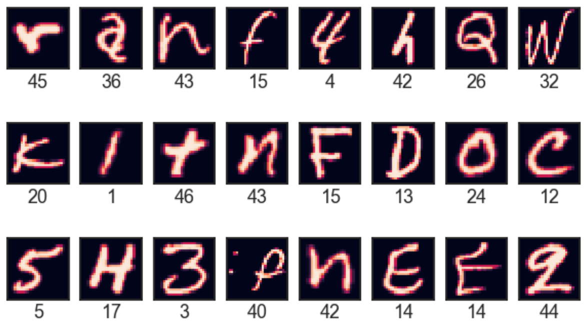

Figure 1: Example images from training set with the corresponding labels representing the letter or the digit (i.e. 17 for capital H)

It is structured in 6 different splits:

- EMNIST ByClass: 814,255 characters. 62 unbalanced classes.
- EMNIST ByMerge: 814,255 characters. 47 unbalanced classes.
- EMNIST Balanced: 131,600 characters. 47 balanced classes.
- EMNIST Letters: 145,600 characters. 26 balanced classes.
- EMNIST Digits: 280,000 characters. 10 balanced classes.
- EMNIST MNIST: 70,000 characters. 10 balanced classes.

Contributions to project are equal, author ordering is by lastname

ByClass and ByMerge have all the data from the NIST Special Database 19. Balanced has the same number of samples for all the classes. Letters have only the uppercase and lowercase letters in a 26-class setting. Digits have only the digits from the ByClass. And lastly, the MNIST is an extended version of the original MNIST dataset.

For our project, we decided to run the experiments on the Balanced split, which will give us a decent amount of data to work with.

Refer to the EMNIST paper for more information about the data.

PCA Principal Component Analysis (PCA) is a preprocessing procedure that uses an orthogonal transformation to convert a dataset that has probably correlated features into a dataset with linearly uncorrelated features that are called principal components. This transformation is defined in a way that the first principal components have the most variance. And the last components have the least variance, so we would be able to omit them to do dimension reduction and omit the noise. We can perform this method in the preprocessing step to reduce the computational overhead and also have better accuracy.

Algorithms

Convolutional Neural Networks Convolutional Neural Networks is one of the most popular and successful image recognition algorithms. They work on a grid-like structure, which also covers images. It's a deep neural network algorithm that uses different layer types with kernels to get the different features of the given sample.

Main building blocks are different types of layers:

1. **Convolutional Layer** Layers we usually use the kernels(filters) that tries to extract specific features. The kernel is slid through all the images (or any other grid-like structure) and computation is made with an activation

function in the end. And, each kernel plus activation function will result in an activation map. (Fig 2.)

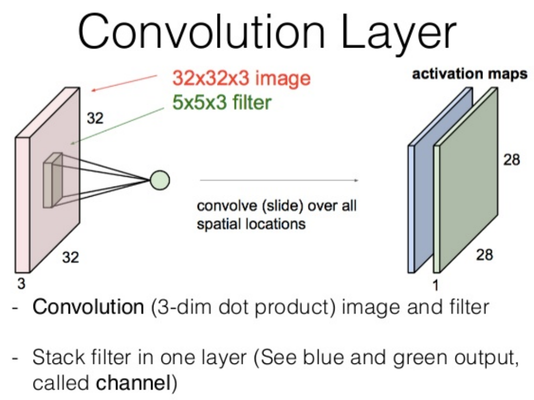
Convolution Layer
32x32x3 image
5x5x3 filter
activation maps
32
32
3
28
28
1
convolve (slide) over all spatial locations
- Convolution (3-dim dot product) image and filter
- Stack filter in one layer (See blue and green output, called channel)

Figure 2: Visual representation of how kernels and convolutional layers are working on the image - figure is representative it will change on final draft

The activation functions are used to decide if a neuron will be active in the given process or not. We have a different type of activation functions that won't be discussed here, for more information about activation functions (Nwankpa et al. 2018)

2. **Pooling Layer** Are placed in between convolutional layers. They are used to keep the feature size in control, like PCA, it's handling the feature extraction. We could have different types of pooling layers, two popular ones are max pooling and average pooling.

   Max pooling would just get the maximum number from the pool whereas average pooling would get the average of the pool. (Fig 3.)

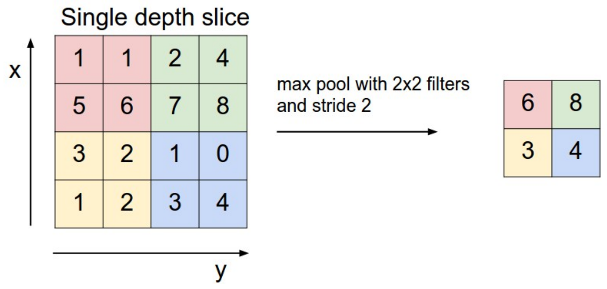

Figure 3: Here is the max pooling for a 4 by 4 convolutional layer, and 2 by 2 pooling kernel size - figure is representative it will change on final draft

3. **Fully Connected Layer** the Last process on the CNN. All the neurons have a fully connected structure with earlier activations from the layers

Add fig 4 *- Fully structured CNN*

Support Vector Machine Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classification and regression challenges. The objective of the algorithm is to find a hyperplane in an N-dimensional space, where N is the number of features, that distinctly classifies the data points. To separate the two classes of data points, many possible hyperplanes could be chosen. Our objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes.(Gandhi 2018) Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends on the number of features. For two input features, the hyperplane is just a line and for 3 input features, the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3. For more about SVMs please check the original paper (Cortes and Vapnik 1995)

Naive Bayes It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods. Bayes theorem provides a way of calculating posterior probability $P(c|x)$ from $P(c)$, $P(x)$ and $P(x|c)$. Look at the equation below:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Above, $P(c|x)$ is the posterior probability of class (c, target) given predictor (x, attributes). $P(c)$ is the prior probability of class. $P(x|c)$ is the likelihood which is the probability of predictor given class. $P(x)$ is the prior probability of predictor. (Ray, Analytics, and Intelligence 2019)

Metrics for multi-class classification

Multi-class Classification algorithms are evaluated using different metrics based on the confusion matrix, which is defined for each class type, such as:

- **Precision:** The proportion of Positive Prediction that are Truly positive, which is

$$\Theta = \frac{TP}{TP + FP}$$

- **Recall:** The proportion of Actual positives that are predicted correctly, which is

$$\Omega = \frac{TP}{TP + FN}$$

- **F1-score:** a metric that can be used to balance between precision and recall, which is

$$f1 = \frac{2 \cdot (\Theta \cdot \Omega)}{(\Theta + \Omega)}$$

- **Accuracy:** Proportion of correct predictions versus total number of predictions

$$\frac{TP + TN}{TP + FP + TN + FN}$$

Moreover, there are some methods to evaluate the performance of an algorithm, such as K-Fold Cross-Validation. It is that the dataset would be split into k folds and in each iteration, we would use 1 fold for the test - calculating the metric - and k-1 folds to train the model. After that we would report the mean metric as the total metric of the algorithm, to obtain a more generalized metric. This algorithm can be performed in three ways:

- **Non-Stratified:** We would just split the dataset into k folds.
- **Stratified:** We would split the dataset into k folds in a way that the proportion of classes in each fold will be the same across different folds.
- **LOOCV (leave-one-out cross-validation):** In this method, each sample is a fold. This method is good for small datasets, but since our dataset is large we are not going to cover that.

Finally, the evaluation between different algorithms is done by Statistical methods, such as (all the samples are created with K-Fold algorithm):

- **Student t-test:** A method of testing hypotheses about the mean of two groups of a small sample, which here each sample is a metric(accuracy or F1-score) for an algorithm, drawn from a normally distributed distribution when the population standard deviation is unknown.
- **Anova test:** The Anova test is a method of testing hypotheses about the mean of groups of samples to see if the mean is different or not.

Grid-Search

A model hyper-parameter is a characteristic of a model that is external to the model and cannot be estimated from data. The value of the hyper-parameter has to be set before the learning process begins. Grid search is the process of performing hyperparameter tuning to determine the optimal values for a given model. This is significant as the performance of the entire model is based on the hyper-parameter values specified. Grid-Search will build a model on each parameter combination possible. It iterates through every parameter combination and stores a model for each combination. As a result, it can be extremely computationally expensive. (Joseph 2018)

Experiments

Preprocessing

We have the dataset with the image size of 28x28 for each input, and each input has the corresponding label. As mentioned in the introduction we are using the 'balanced' dataset which has the same number of samples for each class, therefore we do not do much for cleaning the data or preparing the data for the experiments.

We are using a 6-1 train test split. We have 112.800 training and 18.800 test data.

After having the train-test data, we rank-fold cross-validation to find out the best parameters for the given algorithms. Since we were interested to see the difference between different types of cross-validations, we ran stratified and non-stratified cross-validations for all the results. Table # shows the results.

CNN

We have 3 convolutional layers, and 2 dense layers.

The first convolutional layer has 32 as filter size and 3 by 3 kernel size. Relu activation applied at the end. 2 by 2 max-pooling applied in between layers 1 and 2. The second layer has 64 as filter size and 3 by 3 kernel size. Relu function applied again. Repeating the last process 2 by 2 max-pooling applied in between layers 2 and 3.

The third layer is the same constraints as the second one. After that, the input is flattened and fully connected layers start. The first dense layer is the one we are running the hyper parameters on the filter size and the activation function is used for hyper parameters (explained later in this section). The last layer is the output layer therefore softmax is applied as activation, and the filter size is the unique label size of 62.

As loss function, we used sparse categorical cross entropy and Adam for optimization. Fig # for the full structure.

***Fig #* - Full structure of the CNN model used for the experiments**

Hyper-parameters We choose two hyper-parameters to change and try the algorithm with, the pairs are generated via grid search for all the possibilities. The first one is the number of nodes in the fully connected layer (layer 4), we tried 32, 64 and 128 nodes. The second one is the activation function for the same layer, we tested 'relu' and 'tanh'.

Naive Bayes

For Naive Bayes, we used multinomial Naive Bayes as we have discrete values for labels. There is not much to explain for this part of the experiment, we are using Naive Bayes as our baseline.

We didn't apply any hyper-parameter selection for Naive Bayes.

Support Vector Machine (SVM)

The initial gamma value we used is $1/(N * \sigma_X)$ where N is the number of samples and σ_X is the variance of the data. Since gamma is one of our hyper-parameters, we also used some constant values to see the difference. The experiments are running on different kernels for SVM.

Hyper-parameters We have two hyper-parameters to tweak for SVM too. The first one is the kernel type, 'linear', 'rbf', 'poly'. And the second one is as mentioned before, gamma values. We have two constant values and $1/(N * \sigma_X)$. And again grid search is used on the hyper-parameters.

Results

CNN

Best hyper-parameters picked for CNN are ... Overall we got the accuracy of ... Fig # shows the accuracy vs. epochs. We saw an early converging as can be seen in the figure. The reason is that the data is fairly simple for a big network of this size. Below we also report the test and train accuracy for each fold as well as the f1-scores (Table 1).

include the fig

Folds	Test Accuracy	Train Accuracy	F1-Score
K_1			
K_2			
K_3			
K_4			
K_5			
K_6			
K_7			
K_8			
K_9			
K_{10}			

Table 1: Metric results for all 10 folds for experiment ran on the CNN fed with the best hyper parameters (Best hyper parameters here)

Naive Bayes

The mean accuracy we got is 0.5358 Again we also reported the test and train accuracy and f1-scores for each fold below (Table 2).

SVM

Best hyper-parameters picked for SVM are ... The mean accuracy we got is ... Test and train accuracy and f1-scores for each fold are below (Table 3).

Conclusion

- Results needed

Folds	Test Accuracy	Train Accuracy	F1-Score
K_1	0.5356	0.5424	0.5357
K_2	0.5351	0.5427	0.5365
K_3	0.5196	0.5443	0.5379
K_4	0.5281	0.5430	0.5375
K_5	0.5297	0.5418	0.5369
K_6	0.5404	0.5434	0.5349
K_7	0.5319	0.5434	0.5373
K_8	0.5281	0.5427	0.5353
K_9	0.5196	0.5419	0.5356
K_{10}	0.5505	0.5394	0.5338

Table 2: Same report on Table 1 for Naive Bayes. Best hyper-parameters (hyper params)

Folds	Test Accuracy	Train Accuracy	F1-Score
K_1			
K_2			
K_3			
K_4			
K_5			
K_6			
K_7			
K_8			
K_9			
K_{10}			

Table 3: Same report on Table 1 for SVM. Best hyper-parameters (hyper params)

References

Cohen, G.; Afshar, S.; Tapson, J.; and van Schaik, A. 2017. Emnist: Extending mnist to handwritten letters. In *IJCNN*, 2921–2926. IEEE.

Cortes, C., and Vapnik, V. 1995. Support-vector networks. *Machine learning* 20(3):273–297.

Gandhi, R. 2018. Support vector machine - introduction to machine learning algorithms.

Joseph, R. 2018. Grid search for model tuning.

LeCun, Y., and Cortes, C. 2010. MNIST handwritten digit database.

Nwankpa, C.; Ijomah, W.; Gachagan, A.; and Marshall, S. 2018. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Ray, S.; Analytics, B.; and Intelligence. 2019. 6 easy steps to learn naive bayes algorithm (with code in python).