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

Nikoo Sadat Aghaei — Amirmohsen Sattarifard — M Bedir Tapkan
nikooaghaei@ualberta.ca — sattarif@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 with a part of the EMNIST dataset. 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. There are two hypothesis we are testing; How does stratified and non-stratified cross validation effects the performance of the classification, and which algorithm among CNN (Convolutional Neural Network), SVM and Naive Bayes performs the best in both settings (stratified and non-stratified CV). We at the end realize that SVM is not feasible at given task an changed it with a more convenient method, KNN (K-Nearest Neighbors)(Reasoning is on experiments section).

The results are showing that since we used a balanced data set, there is not much difference between stratified and non-stratified setting, but due to time constraints we couldn't try with the unbalanced dataset. About the algorithms, in overall CNN performs better than the others.

Data Description & Preprocessing

The dataset we will be using belongs to **EMNIST**, which is an extended version (more samples) 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)

It is structured in 6 different splits:

- EMNIST ByClass: 814,255 samples. 62 unbalanced classes.
- EMNIST ByMerge: 814,255 samples. 47 unbalanced classes.
- EMNIST Balanced: 131,600 samples. 47 balanced classes.
- EMNIST Letters: 145,600 samples. 26 balanced classes.

Contributions to project are equal, author ordering is by lastname

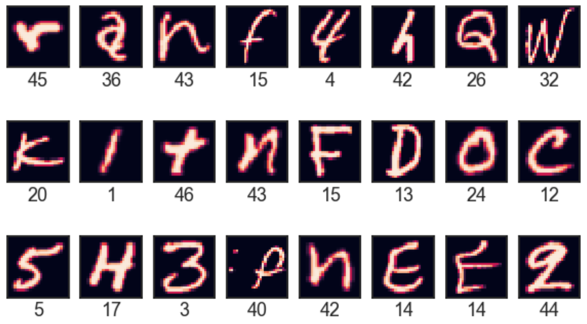

Figure 1: Example images from training set with the corresponding labels representing the letter or the digit (i.e. 17 for capital H)

- EMNIST Digits: 280,000 samples. 10 balanced classes.
- EMNIST MNIST: 70,000 samples. 10 balanced classes.

ByClass and ByMerge have all the data from the NIST Special Database 19. The difference in between is that ByClass is basically all the letters are separated as their lower case and uppercase being different classes. On the other hand ByMerge class has merged classes if the letters are not too different. For example uppercase 'U' and lowercase 'u' are not carrying too much difference so they are merged to one class, but lowercase 'a' and uppercase 'A' is too different to be distinguished so they are in different classes. Balanced is the same version of ByMerge but has the same number of samples for each of 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 sample wise 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.)

   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.)

3. **Fully Connected Layer** the Last process on the CNN. All the neurons have a fully connected structure with earlier activations from the layers. Check Fig 4 for an example of a full 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

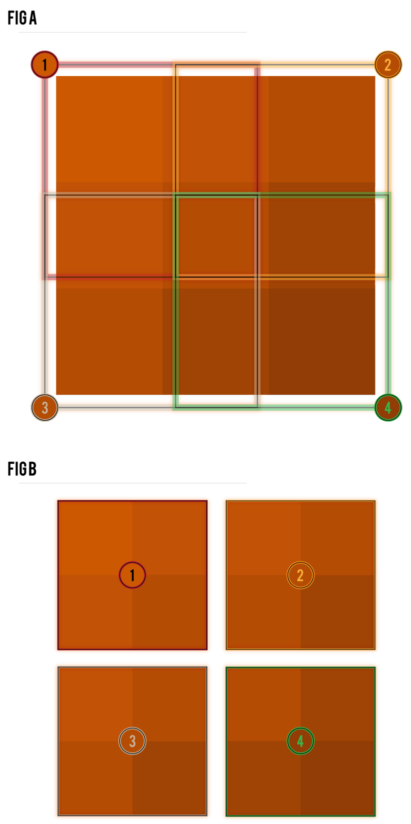

Figure 2: Visual representation of how kernels on convolutional layers are working. **Fig A:** Kernels are laid out in all the space, here we have 3 by 3 image and kernel size is 2 by 2, so in total we will need to run the kernel 4 times to cover all of the picture. **Fig B:** Shows the 4 image parts that are taken as input to change in the kernel

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)

KNN(K Nearest Neighbor) K nearest neighbors is a simple algorithm used for both classification and regression problems. It is also a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset. Lazy algorithm means it does not need any training data points for model generation. All training data used in the testing phase. This makes training faster and testing phase slower and costlier. KNN stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). A case is classified by a majority vote of its neighbors, with

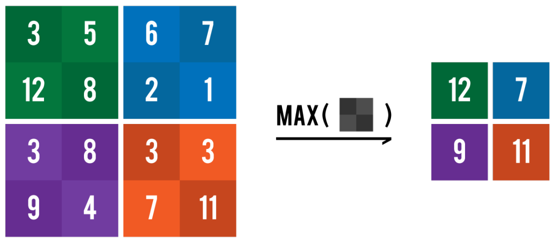

Figure 3: Here is the max pooling for a 4 by 4 convolutional layer, and 2 by 2 pooling kernel size.

the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function. If $K = 1$, then the case is simply assigned to the class of its nearest neighbor.

K in KNN is a hyper-parameter that it is chosen at the time of model building. According to researches, in the case of a small number of neighbors, the noise will have a higher influence on the result, and a large number of neighbors makes it computationally expensive. Also a small amount of neighbors are most flexible fit which will have low bias but high variance and a large number of neighbors will have a smoother decision boundary which means lower variance but higher bias.

Generally, data scientists choose K as an odd number if the number of classes is even, but it can be checked by generating the model on different values of K and check their performances.

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. If the dataset is large, we have an exponentially increased number of operations therefore this method is not used for big data. For our dataset, since we have a mid-to-large size data we decided not to use this method.

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.

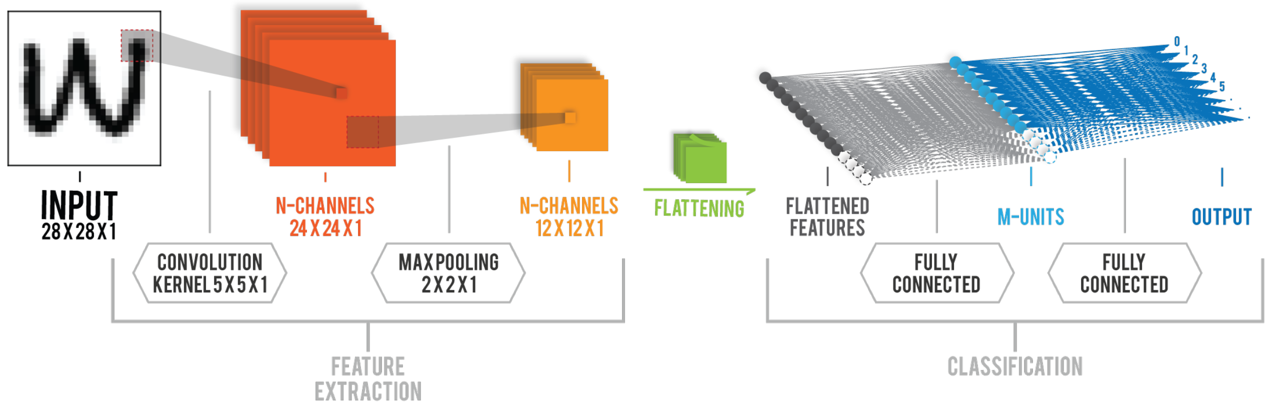

Figure 4: Fully structured CNN. As can be seen, the input is fed to convolutional layer passing by a kernel of size 5x5. N-Channels are created, N being the number of kernels used on input. Max-pooling than applied to the convolved input, and the obtained representation is flattened and being used as the new feature set. After that two fully connected layers are applying simple neural network with activations.

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 balancing the data or preparing the data for the experiments. We only perform PCA as pre-processing before naive-bayes and SVM, which will be changed into KNN as explained ahead. The number of components to be kept in PCA was chosen by running the algorithm for with different number of component. As a result, we chose component = 256.

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. Tables 1 and 4 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.

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)

While working on the data we realized that SVM is not a good choice of algorithm to run this size of data since the complexity is $O(n^3)$. Therefor we changed the algorithm to test, to KNN.

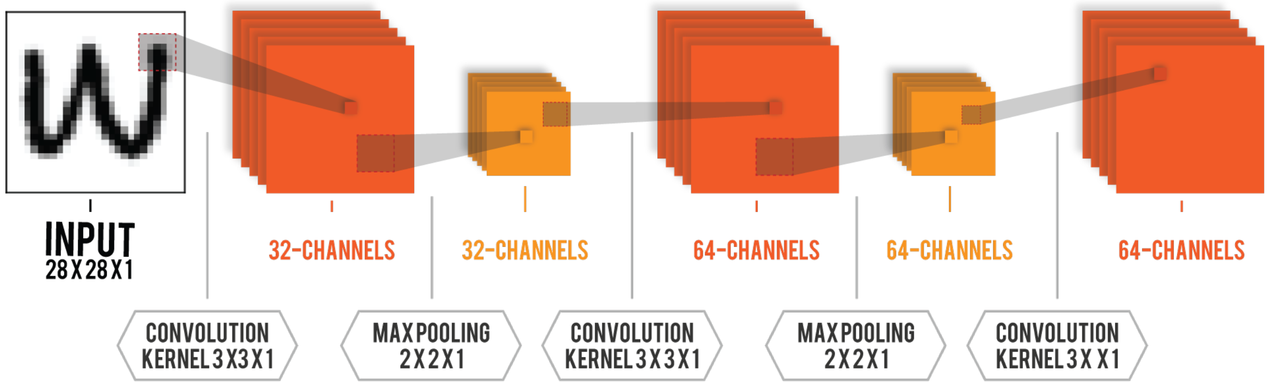

Figure 5: Structure of feature extraction part of the CNN model used for the experiments.

Results

We decided to report F1-Score aside with accuracy, because accuracy by itself is not a good measure even though our dataset is balanced. More information on this can be found on chapter 10 of Regression Modeling Strategies by Frank Harrell (Harrell Jr 2015).

	Activation	Num of Nodes	Accuracy
Stratified	Relu	32	0.8728724
		64	0.8757092
		128	0.8710107
	Tanh	32	0.8762411
		64	0.8791667
		128	0.8765071
Non-Stratified	Relu	32	0.8740248
		64	0.8703014
		128	0.8796986
	Tanh	32	0.8743795
		64	0.8763298
		128	0.8737589

Table 1: Accuracy across different hyper parameters for CNN with stratified and non-stratified CV

CNN

Best hyper-parameters picked for CNN are activation function = 'relu' and dense layer node = 64 for non-stratified and activation function = 'relu' and dense layer node = 128 for stratified. Overall we got the accuracy of 87.57% for stratified and 87.96% for non-stratified CV. Fig 1 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 2 and 3).

Folds	Test Accuracy	Train Accuracy	F1-Score
K_1	0.8716312	0.9050926	1.4954
K_2	0.87765956	0.90527976	1.5137
K_3	0.87810284	0.9034673	1.5233
K_4	0.86648935	0.9069346	1.6756
K_5	0.87092197	0.9053487	1.5433
K_6	0.87916666	0.9028566	1.5124
K_7	0.88005316	0.9039401	1.5457
K_8	0.8760638	0.9046099	1.5093
K_9	0.87437946	0.9057427	1.5594
K_{10}	0.87659574	0.9055851	1.5220

Table 2: Metric results for all **stratified** 10 folds for experiment ran on the CNN fed with the best hyper parameters (activation func = 'relu' and dense layer node = 64)

Folds	Test Accuracy	Train Accuracy	F1-Score
K_1	0.87659574	0.91181046	1.4695
K_2	0.88014185	0.9113081	1.5048
K_3	0.8817376	0.91131794	1.4769
K_4	0.88005316	0.9121257	1.4924
K_5	0.87526596	0.91136724	1.5050
K_6	0.8746454	0.9130024	1.4885
K_7	0.88147163	0.91088456	1.4827
K_8	0.87225175	0.9070528	1.5026
K_9	0.8784574	0.9095646	1.4781
K_{10}	0.8828014	0.91066784	1.4739

Table 3: Metric results for all **non-stratified** 10 folds for experiment ran on the CNN fed with the best hyper parameters (activation func = 'relu' and dense layer node = 128)

	Num of Neighbour centers	Accuracy
Stratified	20	0.77489
	40	0.75414
	60	0.74196
Non-Stratified	20	0.77489
	40	0.75414
	60	0.74196

Table 4: Accuracy across different hyper parameters for KNN with stratified and non-stratified CV

	Accuracy
Stratified	0.54941
Non-Stratified	0.54941

Table 5: Accuracy for Naive Bayes with stratified and non-stratified CV

Naive Bayes

The mean accuracy we got is 0.5494 Again we also reported the test and train accuracy and f1-scores for each fold (stratified and non-stratified below (Table 6 and 7).

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 6: Same report on Table 2 for Naive Bayes with stratified CV.

KNN

The mean accuracy we got is 0.7748 Again we also reported the test and train accuracy and f1-scores for each fold below (Table 8 and 9).

Folds	Test Accuracy	Train Accuracy	F1-Score
K_1	0.5333	0.5488	0.5425
K_2	0.53561	0.5471	0.5397
K_3	0.5243	0.5498	0.5421
K_4	0.5212	0.5387	0.5424
K_5	0.5321	0.5367	0.5398
K_6	0.5323	0.5399	0.5376
K_7	0.5344	0.5488	0.5398
K_8	0.5212	0.5412	0.5401
K_9	0.5287	0.5454	0.5388
K_{10}	0.5457	0.5431	0.5365

Table 7: Same report on Table 2 for Naive Bayes with non-stratified CV.

Folds	Test Accuracy	Train Accuracy	F1-Score
K_1	0.6920212765	0.7155437352	0.69304241325
K_2	0.675	0.715780141	0.672852028
K_3	0.67606382	0.7145390	0.67716463
K_4	0.67659574	0.718498817	0.671235615
K_5	0.67765957	0.714834515	0.675047316
K_6	0.6856382978	0.7151300	0.681948424
K_7	0.6558510	0.7178486	0.6565165
K_8	0.6659574	0.7157801	0.67509464
K_9	0.69202127	0.7131205	0.68479377
K_{10}	0.6925531	0.71371158	0.6893671

Table 8: Same report on Table 2 for KNN with stratified CV. Best hyper-parameters (number of neighbors = 20)

Folds	Test Accuracy	Train Accuracy	F1-Score
K_1	0.6776595	0.717553	0.67781564
K_2	0.6856382	0.713947	0.681729
K_3	0.6840425	0.716489	0.683384
K_4	0.672340	0.7174940	0.6714060
K_5	0.68510638	0.7156619	0.684446
K_6	0.66702127	0.7184988	0.666163
K_7	0.6718085	0.7174940	0.67114049
K_8	0.6941489	0.7152482	0.696478
K_9	0.67234042	0.718321	0.672725
K_{10}	0.686170	0.7145981	0.6885618

Table 9: Same report on Table 2 for KNN with non-stratified CV. Best hyper-parameters (number of neighbors = 20)

paired t-test samples	p value	test result
CNN vs. KNN	0	$\mu_{CNN} > \mu_{KNN}$
KNN vs. Naive	0	$\mu_{KNN} > \mu_{Naive}$

Table 10: t-test results, p-value, and results of test for both non-stratified and stratified CV

paired t-test samples	p value	test result
CNN$^{s} vs. CNN^{ns}$	0.849159	$\mu^{s}_{CNN} == \mu^{ns}_{CNN}$
KNN$^{s} vs. KNN^{ns}$	0.411409	$\mu^{s}_{KNN} == \mu^{ns}_{KNN}$
Naive$^{s} vs. Naive^{ns}$	0.624313	$\mu^{s}_{Naive} == \mu^{ns}_{Naive}$

Table 11: t-test results, p-value, and results between the same algorithm on two different CV methods (s:stratifeid, ns: non-stratified)

Conclusion

In conclusion, we compared KNN, Naive Bayes, and CNN on EMNIST dataset using K-fold cross validation and stratified cross validation. The results show that CNN overruns the other two algorithms based on accuracy and f1 score. Then to make sure that our results are not due to randomness, we did the statistical significant test on pairs of algorithms and between stratified and k-fold cross validation as well. As we used balanced dataset the p-value of same algorithm on k-fold and cross validation is very high (at least 0.4) which is really high. Thus, there is no significant difference between the results on k-fold and stratified cross validation. On the other hand, the p-value for different algorithms are zero which means there is a significant difference between them and CNN achieves the best result. (Table 10 and 11)

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.

Harrell Jr, F. E. 2015. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer.

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).