Hand-written Urdu Character Recognition using Machine Learning

Wajahat Riaz  
*Department of Electrical Engineering*University of Engineering and Technology Lahore, Pakistan  
<mailto:wajahatriazmirza@gmail.com>

***Abstract***— **Many localized languages struggle to reap the beneﬁts of recent advancements in character recognition systems due to the lack of a substantial amount of labeled training data. This is due to the difﬁculty in generating large amounts of labeled data for such languages and the inability of machine learning techniques to properly learn from a small number of training samples. This problem is solved by introducing a technique of generating new training samples from the ground up with realistic augmentations that reﬂect actual variations that are present in human handwriting, by adding random controlled noise. The results with a mere 70 handwritten papers and about 500 samples of each Urdu character per class (i.e Alif, Bay, Jeem, and Daal) surpass existing character recognition results in Urdu. This system is useful in character recognition for localized languages that lack much-labeled training data and even in other related more general contexts such as object recognition. A strategy has also been developed to implement a simple pipeline that can be used by any machine learning enthusiast and test the basic classifiers like Decision Trees, Random Forest, Linear SVM, Logistic Regression, and Stochastic Gradient Descent.**

Keywords— Character Recognition, Urdu Character Recognition, Multi-class Classification, Logistic Regression, SVM, Random Forest, Stochastic Gradient Descent, Decision Trees, Scikit-learn

# Introduction

Handwritten character recognition is a nearly solved problem for many of the mainstream languages thanks to the recent advancements in machine learning models [1].

Nonetheless, for many other languages, handwritten digit or character recognition remains a challenging problem due to the lack of sufﬁciently large labeled data sets that are essential to training machine learning models [2]. Conventional models or classifiers such as Logistic Regression (LR), Decision Trees (DTs), Random Forest (RF), Stochastic Gradient Descent (SGDC) and Support Vector Machines (SVM) [3] can be used for this task, they may not be able to achieve the near human level performances provided by deep learning models but formulate the basis for deep learning.

These models have helped to achieve state-of-the-art results due to their ability to encode features and understanding.

One key advantage is that they require a small number of training samples (usually in the scale of hundreds or thousands per class) to train and classify images successfully. As a result, there is a strong interest in training these models with a lesser number of training samples given the lack of data available for digit or character recognition.

In this paper, a simple pipeline is proposed that tackles the problem of the labeled data set being small in size, with

the aid of machine learning models and utilities like Scikit-learn [4].

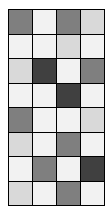
# Related work

MNIST [4] is the widely used benchmark for hand-

written digit recognition task. Multiple works [5, 6, 7, 8,

9] have used machine learning models on MNIST data sets and have achieved results close to 99 percent accuracy. Apart from digit recognition, several attempts in handwritten character recognition with EMNIST data sets. Which are capable of performing image classiﬁcation and especially handwritten character classification.

# Methodology

A generic pipeline is given below which illustrates the steps one has to go through.

Bulk Size Cropping

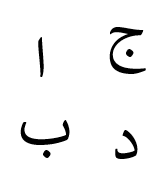
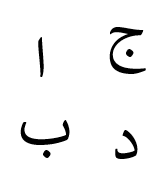
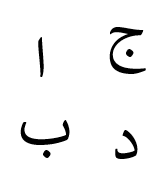
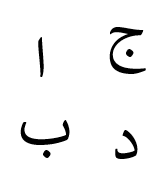
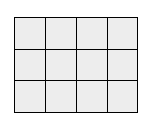
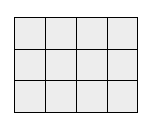
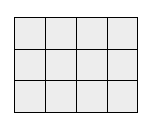
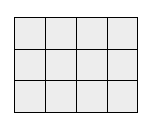


Image Preprocessing and Resizing



Reading every image into an array of order pixel x pixel



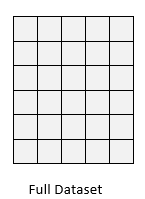
Creating features for machine learning by converting each array of image into one dimensional vector of order

1 x Pixel-squared



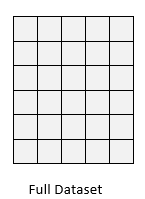


Each vector corresponds to an image. Stacking all the vectors of a class to form a dataset of that particular class. The resultant matrix will be of order Instances x Dimension, where Instances are number of images of that class and Dimension = Pixel-squared



**=** Data Set of Class

Assigning a tag to the classes by making a vector containing the encoded information.



**=** Tag of Class

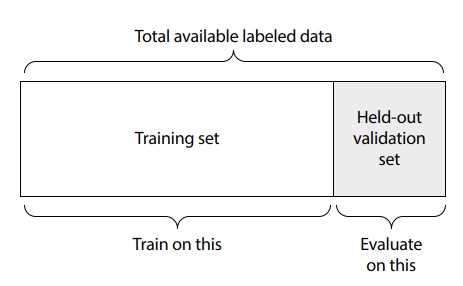
Appending all the datasets to make a matrix **X** and appending all the tags to make a column matrix **Y.**

Y

X

Split the data to for training

and testing



Feed it to the Machine Learning model to generate results

# Experiment

After splitting the data set for training and evaluation. We pass it to our machine learning model. The five most common classifiers are used and they are implemented using Scikit-learn [10].

## Decision Trees (Entropy)

Tree-based learning algorithms are a broad and popular family of related non-parametric, supervised methods for both classification and regression. The basis of tree-based learners is the decision tree wherein a series of decision rules are chained. The result looks vaguely like an upside-down tree, with the first decision rule at the top and subsequent decision rules spreading out below. In a decision tree, every decision rule occurs at a decision node, with the rule creating branches leading to new nodes. A branch without a decision rule at the end is called a leaf. Decision tree learners attempt to find a decision rule that produces the greatest decrease in impurity at a node. While there are several measurements of impurity, entropy is used for impurity for this particular paper. As discussed previously, the reason for the popularity of tree-based models is their interpretability. Decision trees can be drawn out in their complete form to create a highly intuitive model. From this basic tree system comes a wide variety of extensions from random forests to stacking.

## Random Forest

A common problem with decision trees is that they tend to fit the training data too closely (i.e., overfitting). This has motivated the widespread use of an ensemble learning method called random forest. In a random forest, many decision trees are trained, but each tree only receives a bootstrapped sample of observations (i.e., a random sample of observations with a replacement that matches the original number of observations), and each node only considers a subset of features when determining the best split. This forest of randomized decision trees (hence the name) votes to determine the predicted class.

## Support Vector Machine (LinearSVC)

Support vector machines classify data by finding the hyperplane that maximizes the margin between the classes in the training data. In a two-dimensional example with two classes, we can think of a hyperplane as the widest straight “band” (i.e., line with margins) that separates the two classes. LinearSVC implements a simple SVM classifier. SVCs work well in high dimensions i.e. multi-class classification;

## Logistic Regression

Despite having “regression” in its name, logistic regression is a widely used binary classifier (i.e., the target vector can only take two values). In logistic regression, a linear model is included in a sigmoid function. The effect of the logistic function is to constrain the value of the function’s

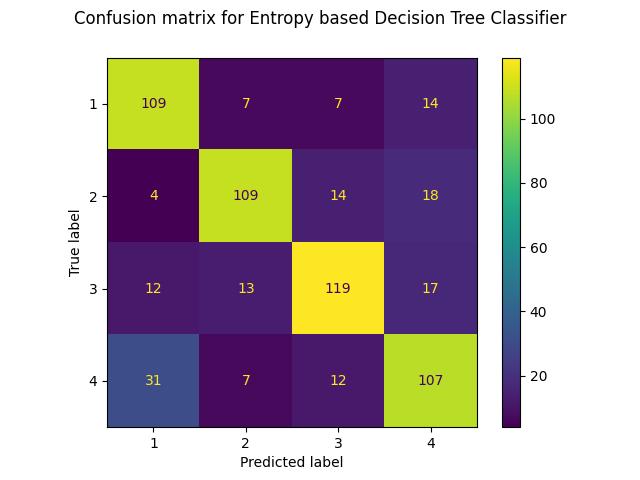
output to between 0 and 1 so that it can be interpreted as a probability. If the obtained probability is greater than the threshold value (0.5), class 1 is predicted; otherwise, class 0 is predicted. On their own, logistic regressions are only binary classifiers, meaning they cannot handle target vectors with more than two classes. However, two clever extensions to logistic regression do just that. First, in one-vs-rest logistic regression (OVR) a separate model is trained for each class predicted whether an observation is that class or not (thus making it a binary classification problem). It assumes that each classification problem (e.g., class 0 or not) is independent.

## Stochastic Gradient Descent (Modified Huber)

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as linear SVM and logistic regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning. Strictly speaking, SGD is merely an optimization technique and does not correspond to a specific family of machine learning models. It is only a way to train a model.

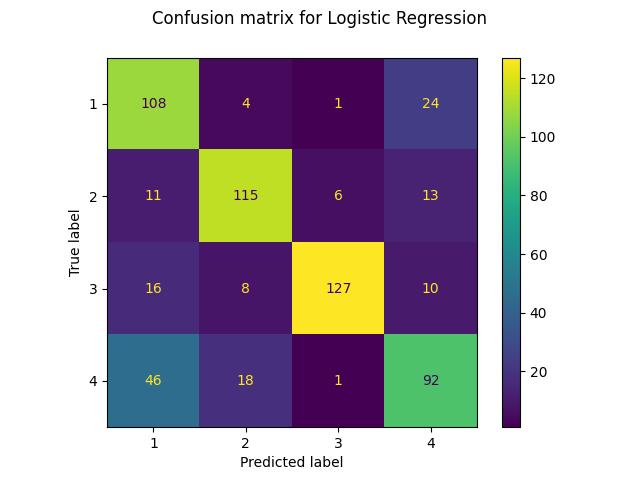
# Results and discussion

We train and evaluate our data five times and recorded classification reports.



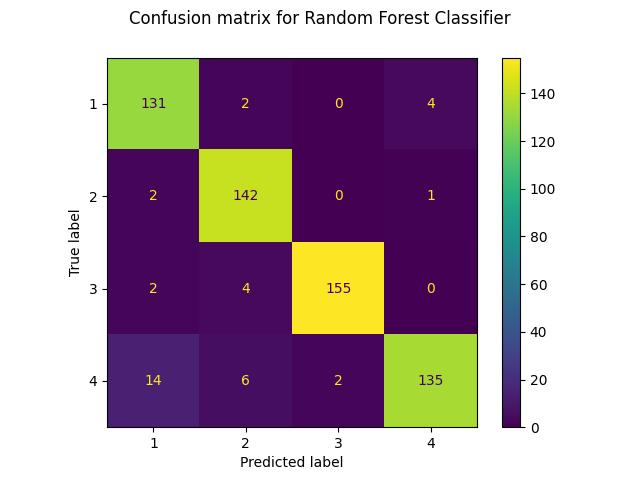
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1-score** | **Support** |
| **1** | 0.73 | 0.74 | 0.73 | 152 |
| **2** | 0.83 | 0.73 | 0.78 | 157 |
| **3** | 0.79 | 0.81 | 0.80 | 144 |
| **4** | 0.71 | 0.78 | 0.75 | 147 |
| **Accuracy** | 0.77 | 0.77 | 0.77 | 0.77 |
| **Macro avg** | 0.77 | 0.77 | 0.77 | 600 |
| **Weighted avg** | 0.77 | 0.77 | 0.77 | 600 |

Fig. 1 Classification Report for Decision Tree Classifier



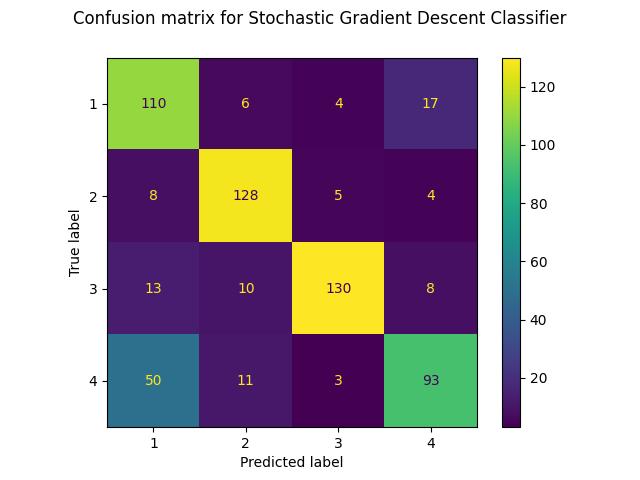
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1-score** | **Support** |
| **1** | 0.60 | 0.79 | 0.68 | 137 |
| **2** | 0.79 | 0.79 | 0.79 | 145 |
| **3** | 0.94 | 0.79 | 0.86 | 161 |
| **4** | 0.66 | 0.59 | 0.62 | 157 |
| **Accuracy** | 0.74 | 0.74 | 0.74 | 0.74 |
| **Macro avg** | 0.75 | 0.74 | 0.74 | 600 |
| **Weighted avg** | 0.75 | 0.74 | 0.74 | 600 |

Fig. 2. Classification Report for Logistic Regression Classifier



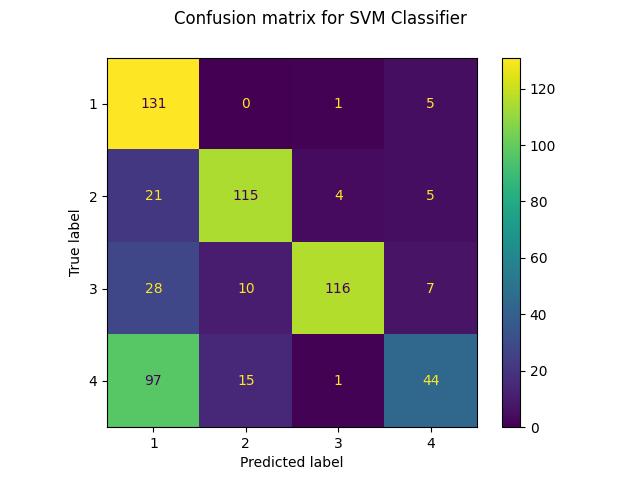
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1-score** | **Support** |
| **1** | 0.88 | 0.96 | 0.92 | 137 |
| **2** | 0.92 | 0.98 | 0.95 | 145 |
| **3** | 0.99 | 0.96 | 0.97 | 161 |
| **4** | 0.96 | 0.86 | 0.91 | 157 |
| **Accuracy** | 0.94 | 0.94 | 0.94 | 0.94 |
| **Macro avg** | 0.94 | 0.94 | 0.94 | 600 |
| **Weighted avg** | 0.94 | 0.94 | 0.94 | 600 |

Fig. 3. Classification Report from Random Forest Classifier



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1-score** | **Support** |
| **1** | 0.61 | 0.80 | 0.69 | 137 |
| **2** | 0.83 | 0.88 | 0.85 | 145 |
| **3** | 0.92 | 0.81 | 0.86 | 161 |
| **4** | 0.76 | 0.59 | 0.67 | 157 |
| **Accuracy** | 0.77 | 0.77 | 0.77 | 0.77 |
| **Macro avg** | 0.78 | 0.77 | 0.77 | 600 |
| **Weighted avg** | 0.78 | 0.77 | 0.77 | 600 |

Fig. 4. Classification Report for Stochastic Gradient Descent Classifier



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Precision** | **Recall** | **F1-score** | **Support** |
| **1** | 0.47 | 0.96 | 0.63 | 137 |
| **2** | 0.82 | 0.79 | 0.81 | 145 |
| **3** | 0.95 | 0.72 | 0.82 | 161 |
| **4** | 0.72 | 0.28 | 0.40 | 157 |
| **Accuracy** | 0.68 | 0.68 | 0.68 | 0.68 |
| **Macro avg** | 0.74 | 0.69 | 0.67 | 600 |
| **Weighted avg** | 0.75 | 0.68 | 0.67 | 600 |

Fig. 5. Classification Report for Linear SVM Classifier

|  |  |  |
| --- | --- | --- |
| **Classifier** | **Accuracy Score** | **AUC\_ROC Score** |
| **Random Forest** | 0.938 | 0.992 |
| **Linear SVM** | 0.678 | 0.891 |
| **Logistic Regression** | 0.7366 | 0.900 |
| **Decision Tree** | 0.74 | 0.828 |
| **Stochastic Gradient Descent** | 0.7683 | 0.500 |

Fig. 6. AUC and Accuracy score of Classifiers

# Conclusion

In summary, we achieved classification reports for our data set by implementing different models. Random Forest happens to give the best performance for the given data set.

##### References

1. Y. LeCun, L. Bottou, Y. Bengio, and P. Haﬀner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol.86, no.11, pp.2278–2324, 1998.
2. W. Jiang and L. Zhang, “Edge-siamnet and edge-triplenet: New deep learning models for handwritten numeral recognition,” IEICE Transactions on Information and Systems, vol.103, no.3, pp.720–723, 2020.
3. G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, “Emnist: Extending mnist to handwritten letters,” 2017 International Joint Conference on Neural Networks (IJCNN), pp.2921–2926, IEEE, 2017.
4. Hinton, G.E., Krizhevsky, A., Wang, S.D.: Transforming auto-encoders. In: ICANN, Berlin, Heidelberg (2011) 44–51
5. Y. Song, G. Xiao, Y. Zhang, L. Yang, and L. Zhao, “A handwrittencharacter extraction algorithm for multi-language document image,” 2011 International Conference on Document Analysis and Recognition, pp.93–98, IEEE, 2011.
6. H. Kusetogullari, A. Yavariabdi, A. Cheddad, H. Grahn, and J. Hall, “Ardis: a Swedish historical handwritten digit dataset,” Neural Computing and Applications, pp.1–14, 2019.
7. M. Biswas, R. Islam, G.K. Shom, M. Shopon, N. Mohammed,S. Momen, and M.A. Abedin, “Banglalekha-isolated: A comprehensive Bangla handwritten character dataset,” arXiv preprintarXiv:1703.10661, 2017.
8. H. Khosravi and E. Kabir, “Introducing a very large dataset of handwritten Farsi digits and a study on their varieties,” Pattern recognition letters, vol.28, no.10, pp.1133–1141, 2007.
9. U. Bhattacharya and B.B. Chaudhuri, “Handwritten numeraldatabases of Indian scripts and multistage recognition of mixed numerals,” IEEE transactions on pattern analysis and machine intelligence, vol.31, no.3, pp.444–457, 2008.
10. [Multiclass and Multioutput Algorithms — scikit-learn 1.2.0 documentation](https://scikit-learn.org/stable/modules/multiclass.html)