Handwritten Letter Identification: Fundamentals of Machine Learning Project01

Kiana Alikhademi
Computer & Information
Sciences & Engineering
University of Florida
kalikhademi@ufl.edu

Diandra Prioleau
Computer & Information
Sciences & Engineering
University of Florida
dprioleau@ufl.edu

Armisha L. Roberts
Computer & Information
Sciences & Engineering
University of Florida
ar23@ufl.edu

Abstract— **This paper illustrates the work conducted by researchers to identify English handwritten characters. A non-parametric and instance-based classifier, k-Nearest Neighbor (kNN), was chosen to classify the detection of the characters a, b, c, d, h, i, j, and k. Through data processing, each image of the individual characters were normalized into the same dimensions and the contours of the images were created. Each character was classified by extracting its features using the diagonal feature extraction, which was applied to both the training and testing data. The researchers conducted multiple experiments to determine the hyperparameters of the algorithm, including "k" value, the train-test split size, and the distance metric used with the classifier to train and test. Using k=3 with a 70/30 train/test split and k=9 with an 80/20 train/test split, we achieved 97.5% and 75% average accuracy in the base (A and B) and extra credit cases respectively.**

I. INTRODUCTION

A. Motivation

Developing technologies that can perform handwritten character recognition is a heavily researched topic within the field of image processing; even prior to the development of the computer, this task has fascinated researchers [1]. There has been a variety of techniques created to accomplish this task that utilize the capabilities of machine learning such as Neural Networks, k-Nearest Neighbor, Support Vector Machines, and other various methods [2], [3], [4], [5]. The problem of identifying handwritten characters has spanned across multiple languages and includes numerical identification as well. Handwritten character recognition can take place in two forms, either online or off-line. Online signifies that the system receives input in real time of the character to be identified. For off-line, the handwritten character is provided as input to the system as an image or some other digital format after it has already been written. The type of handwriting recognition that will be discussed throughout this paper is an experiment with off-line recognition. There are a variety of ways where this advancement can be applied to improve the human-computer interaction by providing additional means of engagement with a device through multimodal interactions by permitting handwriting.

B. Related Works

As previously mentioned, this research problem has drawn significant attention, which has led to various methodologies being developed and applied to resolve this issue. In [2], the authors detail the importance of feature selection and the multiple features that can be used to train the system to acquire desired outcomes. These authors selected a diagonal feature extraction scheme which was also utilized by the researchers of this work. However, Pradeep, Srinivasan, and Himavathi also utilized an additional 15 features based on the average of the row and column values acquired from each image after determining their zone values from the 10 x 10 zones partitioned from each image. They applied these features to a Neural Network and achieved an accuracy score of 97.8% when only using the diagonal features and 98.5% when using all of their acquired features. [6] applies the Freeman Chain Code of English characters to a Support Vector Machine for lowercase letters, uppercase letters, and a blend of both uppercase and lowercase letters; they obtained an accuracy of 86%, 88%, and 73%, respectively. Patel and Shimpi applied a Neural Network to a handwritten English character data set of both uppercase and lowercase letters. Through the use of back propagation, they were able to obtain an accuracy over 70% [7]. This implementation discussed in this paper was modeled after the worked conducted by Kumar, Jindal, and Sharma, in which the k-Nearest Neighbor (kNN) classifier and feature extraction methodology were used. The researchers of this paper applied Kumar, Jindal, and Sharma's techniques, which were used to recognize Gurmukhi handwritten characters, to English handwritten characters to assess their methods effectiveness and transferability to English characters. The kNN classifier provided researchers with a vast level of flexibility to manipulate the neighbors being used for classification, weights of the neighbors, the distance measure being used within the algorithm, and the features being used for classification. Due to these reasons, the researchers selected this classifier to address the character recognition problem. The researchers conducted multiple experiments by 1) altering the "k" value used to determine the number of neighbors, ranging between 5 and 21; 2) testing the effects of the different distance measures - Euclidean, Manhattan, and Chebyshev; and 3) the impact of varying train-test splitting ratios from 50/50 to 90/10 in ten percent increments, with the larger percentage being used for training except for the 50/50 case where there is an even division between training and test data. The work conducted by the researchers will be detailed further within Section III of this paper.

II. Implementation

Our implementation was based on the proposed recognition system created by Kumar et. al [4]. It consisted of 4 phases - digitization, pre-processing, feature extraction, and classification.

A. Digitization

During the digitization process, the paper-based handwritten characters converted to a binary vector have been provided by Dr.Zare. To train for classifying ”unknown” images, the A-Z Handwritten Alphabets data set obtained through Kaggle [8], which consisted of all uppercase A-Z letters, was used. In addition, the researchers also wrote lowercase letters e-g, l-z and 0-9 to use in the training process for ”unknown” classification. These data sets can be accessed through the repository. Both data sets were converted to binary images using the code provided by Dr. Zare in homework07.

B. Pre-processing

During the pre-processing phase, the character image is normalized to a window size of 100 x 100. After which, the image is converted to a bitmap image, which is then converted into a contour image.

C. Feature Extraction

After finding contours the images, the features need to be extracted from each contour of the image to classify the character. In [4], two feature extraction methods were used: diagonal and transition. The diagonal feature extraction was chosen to identify the handwritten English characters instead of the transition feature extraction since the transition feature extraction resulted in lower performance when identifying handwritten Gurmukhi characters [4]. Using same approach as Kumar et al. [4], the following steps are taken to extract the features: :

1) Divide the image into 100 zones, each zone with a size of 10 x 10 pixels.
2) The features are extracted from the pixels of each zone by moving along its diagonal.
3) Each zone consists of 19 diagonals, where each diagonal is considered a sub-feature. As you move along the diagonal, sum up the foreground pixels to form a single sub-feature.
4) The 19 sub-feature are averaged to one value, which corresponds to the feature for the zone. Therefore, for an image, there are 100 features.
5) Zones that do not have foreground pixels, the feature value will be zero.

D. Classification

The k-Nearest Neighbor (kNN) classifier is used to identify the characters. kNN is a non-parametric algorithm which compares each test point to the “k” nearest training data points. The label of the test point is chosen based on the majority vote of the labels of the ‘k’ nearest neighbors. One of the advantages of using kNN is that it does not require any “training” other than storing the training points and their corresponding labels. One of the disadvantages with kNN is that the training process can be time-consuming; however, since the model with the best performance is being saved for testing purposes, kNN was still chosen as the best option. The features extracted from the previous phase are used in the kNN classifier to classify the images. The Manhattan distance was chosen as the metric for computing the distance between points after conducting cross-validation, which will be discussed in the next section since it resulted in the best performance. The equation for Manhattan distance is defined in the Experiments section.

III. Experiments

In this section, we first review all the parameters which are used in the experiments along with the goal of each experiment. The latter part focuses on the results of each experiment and outlines the major findings in each case.

A. Cross Validation

During this phase, “k” value, train-test split size, and distance metric are considered as the hyperparameters to tune.
“k”: kNN does not have an extensive training phase and only computes the distance between test instances and training data to predict the labels based on the majority vote. To obtain better results for kNN, we needed to tune the “k” parameter based on the train-test split size, distance metric, and the number of target classes. As Zakka [9] suggested, the value of “k” provides some control over the decision boundary in the process of prediction. Small values of k may lead to a higher influence of noise on the classification result if not enough information is obtained from the whole distribution. In contrast, large values are computationally expensive due to kNN’s non-parametric and instance-based nature, which may result in outliers in the labeling process. Therefore, the parameter k should be chosen through cross-validation and diagnostic checks. The goal in this experiment was to investigate the impact of “k”, ranging from 1 to 21, on the classification accuracy.
Train-Test Data Split: The provided data is split into a training and testing data set to assess the model’s performance. Regarding the data splitting approaches, there are two main concerns. First, the estimates of target parameters have higher variance if a small number of training data is selected. In contrast, with less testing data your performance statistic will have higher variance. Literature suggests picking the data splitting approach such that none of these variances are significantly higher than the other. Due to the instance-based nature of kNN, each test instance would be compared with the whole training data to do the classification. Therefore, a smaller test size could benefit the model in reducing the time complexity. However, it is essential to be cautious about overfitting and underfitting. As it is evident, there are multiple factors to consider while choosing the proper train-test data split. Therefore, investigating the impact of different training and testing data sizes with varying values of “k”

is the second goal of the experiment. The data partitioning approaches were the same as the work in [4] which consisted of 50/50, 60/40, 70/30, 80/20, 90/10 train/test splits.

Distance Metric: [10] emphasized that using the proper distance metric in the kNN method is vital. Their research has shown that there is not one distance metric which performs best for all data sets [10]. Therefore, finding the distance metric which fits best for recognizing handwritten English characters was another goal. Using the KNeighborsClassifier method from the Sklearn library, the model performance was assessed by varying distance metrics, values of k, and data partitioning approaches which were changed in the cross-validation step simultaneously. [10] detailed the distance metric formulas for computing the distance between x_s and y_s row vectors as follow:

- Euclidean Distance

$$d^2_{ss} = (x_s - y_s)(x_s - y_s)' \tag{1}$$

- Manhattan Distance

$$d_{ss} = \Sigma^n_{j=1} |x_{sj} - y_{sj}| \tag{2}$$

- Chebyshev Distance

$$d_{ss} = Max_j(|x_{sj} - y_{sj}|) \tag{3}$$

B. Results

1) A and B case: All three parameters mentioned in the previous section were coupled together to see how they each impacted the performance of the models. First, it was investigated how the averaged accuracy would change if different distance metrics and data splitting approach were used while changing "k" values. To visualize the results, the accuracies across different values of "k" for each train-test split were computed. As shown in Fig. 1 the Manhattan distance outperformed the other distance metrics across all the data splitting approaches, which is why this metric was selected for distance. The main reason for this could be related to the fact that Manhattan distance (L_1) is less sensitive to outliers, the various ways one character can be written. Moreover, the 70/30 and 90/10 training/test splits showed the best results. To decide which data partitioning approach should be chosen, the accuracies using the Manhattan distance, as shown in Fig. 2, for the top three training/testing splits (70/30, 80/20, 90/10) and different values of "k" were compared and assessed. Increasing the values of "k" across all splits led to a decrease in accuracy. This decrease in performance could have been related to the fact that higher values of "k" consisted of outliers and unnecessary complexity. k=1 and 90/10 resulted in the highest accuracy, but there was a concern of overfitting the training data. Therefore, it seemed more reasonable to pick k=3 and the 70/30 split, which also had a high performance.

After fitting the kNN model using the parameters obtained in cross-validation, the model performance concerning both classes were assessed and summarized in Fig. 3. The misclassification for each class was less than 8%, other performance metrics including precision, recall, and F1_ score were 0.96 for all metrics. The average precision value for both classes shows that around 96% percent of the cases that were reported positive were positive. Moreover, the recall emphasizes that 96% of the actual instances were marked correctly. The F1_score, a harmonic mean of precision and recall, shows that our method is performing better than the average in distinguishing the two existing classes. These results are promising as our model was able to recognize "a" and "b" well even though there were different rotations and writing styles for the characters. The higher classification result for "b" in comparison to "a" could be based on the few writing styles for the letter "b".

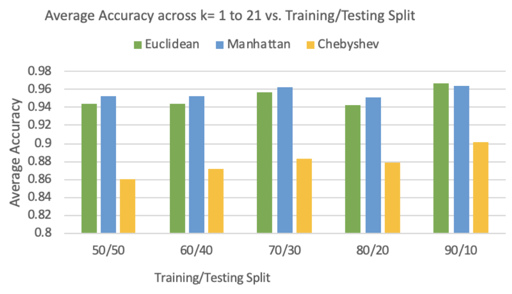

Fig. 1. Average Accuracy across k= 1 to 21 vs. Training/Testing Split (A and B case)

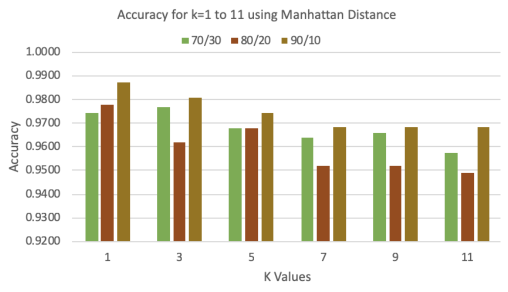

Fig. 2. Accuracy for k=1 to 11 using Manhattan Distance(A,B case), As the other values are leading to less accuracy it is not showed for visualization purposes.

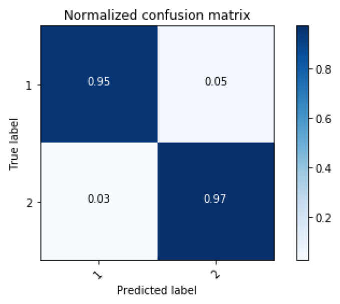

Fig. 3. Confusion Matrix for A and B case

2) Extra Credit: Similar to the A and B case, different distance metrics, data partitioning approaches, and "k" values were explored to assess their contribution to the performance for classifying other characters and "unknowns". Optimum test data size and "k" values have been chosen by considering the optimum distance metric. The average accuracy of different data splits, across k=1 to 13, is shown in Fig. 4. Although the difference between Euclidean and Manhattan distance was small, the Manhattan distance was chosen because it was less sensitive to the noise, which is an important factor in this task as it is focusing on recognizing 8 different characters from a diverse data set. Moreover, Manhattan captures the small changes in the features better than Euclidean due to its function shape and properties. Therefore, the Manhattan distance was considered as the optimum distance metric in this case. After selecting the Manhattan distance, the next steps were to investigate how accuracy changes while varying values of "k" and training/testing data size. The result of this experiment is shown in Fig. 5. It is shown that varying the values of "k" leads to increase in accuracy until we reach k=9 and the accuracy dropped afterwards. This behavior shows us that k=9 could be the optimum value of "k". As the highest results were obtained using 80/20 training/test splits, k=9 and 80/20 train/test split were chosen to build the final model.

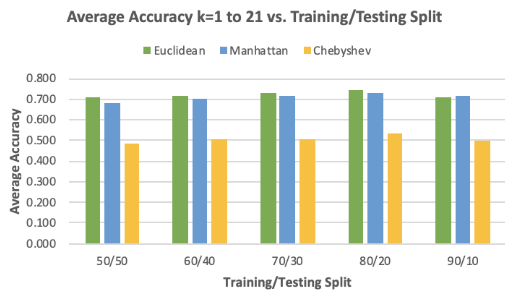

Fig. 4. Average Accuracy of different distance metrics vs. different data splitting

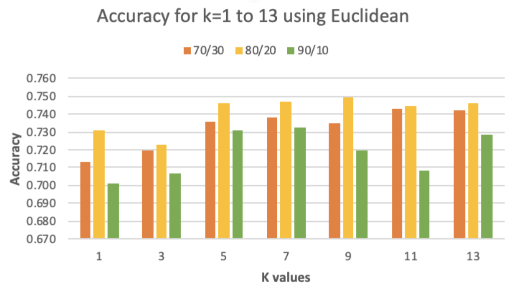

Fig. 5. Accuracy of different k values vs. different data splitting using Manhattan distance

Computing the overall performance of the model, in this case, is extremely important as it offers insight into which classes were misclassified the most. The two highest true positive rates were for characters "c" and "a" while the lowest were for characters "k", "j" and unknowns. There are not many different writing styles or rotations for character "a" and "c" which could mislead our defined model. In contrast, character "k" was recorded in many different ways in the training images which could impact the performance of the classifier. All the differences and unknown data could make it challenging to distinguish characters using a simple method like kNN. Aside from character "k", character "j" was also misclassified at times. The binary images provided for characters "i" and "j", in some cases, looked very similar and may have led to misclassification of those letters. Other performance metrics including precision, recall, and F1_score were calculated as 0.74, 0.75, and 0.74 respectively. The average precision values show that around 74% percent of the cases reported for each class were classified correctly. Although there were different writing styles for characters "k", "i", "j" and similarity between characters "i" and "j", 75% of the actual instances were marked correctly as shown by the recall metric. Moreover, the unknown data has been recognized correctly more than 25% of the time although this data has a diverse nature of including handwritten uppercase and lowercase characters along with numerical digits.

IV. Conclusion

This work utilized the versatility of the kNN algorithm to experiment and to determine the optimal "k" value, distance metric, and train-test split that would result in the highest accuracy in recognizing the English handwritten letters "a" and "b", which resulted in a classification accuracy of 97.5%, as well as letters "a", "b", "c", "d", "h", "i", "j","k" and unknowns, consisting of uppercase and various lowercase letters and digits, which resulted in a classification accuracy of 75%. The conclusions reached only support the characters trained and tested on; the researchers cannot generalize these results for the entire English alphabet.

References

[1] S. Mori, C. Y. Suen, and K. Yamamoto, "Historical review of ocr research and development," *Proceedings of the IEEE*, vol. 80, no. 7, pp. 1029–1058, 1992.

[2] J. Pradeep, E. Srinivasan, and S. Himavathi, "Diagonal based feature extraction for handwritten character recognition system using neural network," in *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, vol. 4. IEEE, 2011, pp. 364–368.

[3] C. Wu, W. Fan, Y. He, J. Sun, and S. Naoi, "Handwritten character recognition by alternately trained relaxation convolutional neural network," in *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*. IEEE, 2014, pp. 291–296.

[4] M. Kumar, M. Jindal, and R. Sharma, "k-nearest neighbor based offline handwritten gurmukhi character recognition," in *Image Information Processing (ICIIP), 2011 International Conference on*. IEEE, 2011, pp. 1–4.

[5] P. Dhande and R. Kharat, "Recognition of cursive english handwritten characters," in *Trends in Electronics and Informatics (ICEI), 2017 International Conference on*. IEEE, 2017, pp. 199–203.

[6] D. Nasien, H. Haron, and S. S. Yuhaniz, "Support vector machine (svm) for english handwritten character recognition," in *Computer Engineering and Applications (ICCEA), 2010 Second International Conference on*, vol. 1. IEEE, 2010, pp. 249–252.

[7] V. Patil and S. Shimpi, "Handwritten english character recognition using neural network," *Elixir Comput Sci Eng*, vol. 41, pp. 5587–5591, 2011.

[8] S. Patel. Csv_to_images. [Online]. Available: https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

[9] K. Zakka. A complete guide to k-nearest-neighbors with applications in python and r. [Online]. Available: https://www.kaggle.com/sachinpatel21/csv-to-images/data

[10] K. Chomboon, P. Chujai, P. Teerarassamee, K. Kerdprasop, and N. Kerdprasop, "An empirical study of distance metrics for k-nearest neighbor algorithm," in *Proceedings of the 3rd International Conference on Industrial Application Engineering*, 2015.

[11] V. Prasath, H. A. A. Alfeilat, O. Lasassmeh, and A. Hassanat, "Distance and similarity measures effect on the performance of k-nearest neighbor classifier-a review," *arXiv preprint arXiv:1708.04321*, 2017.