Handwritten Alphabets Recognition using KNN Classifier

Smrati Pandey
smratipandey@ufl.edu
CISE, University of Florida

***Abstract*— Recognizing offline handwritten characters still pose a significant challenge in the field of Optical Character Recognition (OCR) because every person writes the same set of characters differently which the application should be able to identify. Though different efficient and reliable methodologies for this task have been proposed, but in this paper, we only evaluate the performance of KNN algorithm to correctly classify a data of alphabets. We vary different parameters associated with the model implementation and check which combination yield the best performance.**

***Index Terms*— Bipartite Hungarian Graph matching, Distance Metrics, Handwritten Character Recognition, K-Nearest Neighbors (KNN), Linear Discriminant Analysis (LDA), Principal Component Analysis (PCA), Shape-context Descriptors**

I. Introduction

HANDWRITTEN characters recognition is a popular and interesting machine learning research problem because shape and size of characters vary with largely differentiated human writing styles. And, it is very crucial in OCR applications to speedily recognize characters (digits, alphabets, etc.) with high accuracy. Such desired performance of recognizing the characters depends on two major factors: (a) applying a relevant feature extraction technique, (b) selecting the best classifier model. So, we look at some of the those approaches that have been proposed in the past and analyze the results for OCR in this paper.

We understand that Deep Neural Networks (DNN), here the Convolutional Neural Networks (CNN) for character images, are predicted to perform the best for being the current state-of-the-art. However, these models generally tend to overfit the input data because data required to train the model may not be sufficient to accommodate the expensive computation of large number of model parameters with respect to a high dimensional input feature space. In this paper, we implement the KNN classifier [9][10] to recognize these alphabets which is more transparent, computationally less expensive and doesn't need lots of data. It is also appropriate for this problem because KNN generally works wells for complex classifications having non-linear decision boundaries. Here, we mainly refer to the work done by Yann LeCun on OCR problems [4].

Our main focus is building a highly accurate model using KNN classifier to identify the characters 'a' and 'b' correctly from a set of handwritten characters. We go one step further to build a more generalized model to classify all the eight English alphabets: 'a', 'b', 'c', 'd', 'h', 'i', 'j' and 'k' as per the requirements of the project. For this, we are provided with around 6400 images dataset by the course instructor in which each character appears almost 800 times. Because the data is pre-cleaned and labeled, we directly start by applying different data preprocessing techniques and varying different associated hyperparameter to compare and analyze the best accuracy in less training time. Among different experimental setups, we use different feature extraction and reduction techniques along with varying the input distance metric or the number of nearest neighbors as parameters to the model.

Lastly, the different sections of this paper are organized as follows. In Section II, it discusses different approaches taken to implement the KNN model in order to improve the algorithm's time complexity. We characterize the training data we are given and apply different preprocessing techniques before we feed the data to the model. We also compare those results with the results we get when we directly input the raw data to the model. In Section III, we describe our factorial design of experiments and analyze their results with each other so as to find the best combination of parameters identifying the characters correctly. We discuss that these parameters may not be independent of each other and also depend on the order in which the design steps are performed. Some parameters which are expected to perform better give bad result due to limitation of computational resources at hand while we make those parameters more flexible to make the preprocessing computationally less expensive. Section IV concludes by providing the final results and a discussion on scope of improvement in the implementation.

II. Model Implementation

KNN is a relatively simple yet highly effective classification tool and we use the one written in scikit-learn's python package which uses Minkowski's distance to find the K-nearest data points by default. In this the majority voting decides what class shall the test point belong to. We follow a

simple pipeline to implement our model, that is: (a) We first characterize the given training data, divide the data to first classify 'a' and 'b' only (Easy Dataset) and then to classify all desired labels (Hard Daraset), (b) the sorted data is split into training and validation sets using k-fold validation criterion for both cases, (c) the data is then preprocessed before it is fed into the KNN classifier, (d) processed training data is given to the KNN model while the validation data is tested for predicting it's class label by the algorithm, (e) the results are analyzed and evaluation metric is used to assess the model, (f) the model parameters / hyperparameters are further tuned to improve the results and accuracy.

Characters' Data

The provided training data consists of 6400 labeled binary images of different alphabets stated in the introduction. Each character appears around 800 times in the dataset. Because the data is already labeled, it doesn't require further cleaning for the model input. Each binary image has a maximum dimension of 54 in either row or column, but the images differ in dimensions. By binary we mean that each image pixel has a value 'true' for the character's outline and 'false' elsewhere. *Fig1.* shows a plot of some of these images.

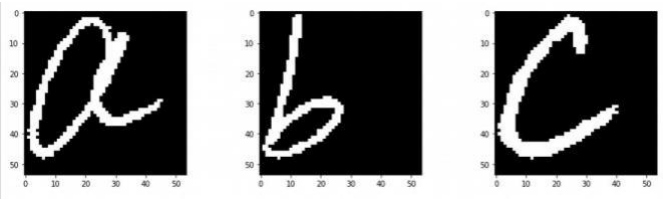

Fig1. Some samples from the Training dataset (padded)

Data Preprocessing

A. K-fold validation

We split the data into k equally sized subsets.We perform the test on random one subset while all other subsets are taken for training. We vary the value of k till we get the most consistent classification accuracy results, i.e. until the error rate curve of both the validation set and the training set is at the minimum. Once the curve of validation set starts rising again from the minimum, that would indicate the overfitting the data.

B. Padding and down-scaling the image

Because the binary imageset has images / matrices of different sizes, we pad the matrices at both row and column ends with 'false' pixel values which doesn't change the context of the image but helps in making the size of row and column of each image equal, i.e. 54x54 pixels (total 2916).

Because a large set of the image is empty and doesn't contain any relevant information, we downscale the image further to 28x28 pixels (total 784) so as to decrease the computations non-informative redundant features in the large pixel feature space. We also try to reduce any noisy features through this step. Thus, scaling is the important sensitive factor in determining the performance of our model.

Normalization and Dimensionality Reduction

Even if we downscale the image to 784 pixels, the images still carry a lot of unnecessary information, and are not required for the classification. KNN performs better with a lower number of features than when we have a larger number of those. By the principle of curse of dimensionality, the data required grows exponentially with the number of features or dimensions. So, having a large number of features but a small dataset would result in a model overfitting that data. To reduce the dimensionality or the feature space, we use the first [2] of the two classical techniques: (a) PCA, (b) LDA. Before applying PCA, we normalize the imageset by subtracting each from the mean of the corresponding class and dividing with the standard- deviation of that class. This helps making the data invariant to different scaling and size. PCA helps avoiding carrying over redundant non-relevant information by finding a rotation vector in the feature space along the direction of maximum variance of data for the first principal component. The next principal components are selected such that the maximum variance decreases after the preceding ones. The data is projected onto those selected principle components, which results in a non-redundant independent new feature space.

Distance Metrics in KNN

Being a non-parametric lazy learning algorithm, KNN doesn't make any prior assumptions about the underlying data, and requires only the parameter K (decided based on data) and a distance metric that defines the nearest proximity between any two data points in the *k* dimensional feature space. We use the following metrics as input distance parameter to the KNN classifier. Here, 'mu' is the class mean and 'S' is the class covariance.

Minkowski's Distance	$\left\{\sum_{i=1}^{k}(\lvert x_i - y_i\rvert)^q\right\}^{1/q}$
Euclidean Distance	$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$
Hamming Distance	$D_H = \sum_{i=1}^{k}\lvert x_i - y_i\rvert$ $x = y \Longrightarrow D = 0$ $x \neq y \Longrightarrow D = 1$
Mahalanobi's Distance	$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1}(\vec{x} - \vec{\mu})}$

Shape Context Descriptors (for feature extraction)

We are interested in extracting further a few more relevant features in a new transformed feature space. A 28x28 pixel image is represented in a feature space of 784 dimensions, which is quite a high number. With PCA we are able to significantly reduce the features which in turn helps making the algorithm compute faster. But we hope that we can improve further by adopting the concept of 'shape context descriptors' [1] [5], which are invariant to any rotation or scaling or other image transformations while the shape/context of the image is maintained. This is because these take advantage of relative positional information of respective shapes.

A shape context of a point describes the arrangement of rest of the points in the shape with respect to this point. In

this representation, each shape is defined by a discrete set of points sampled from inner or outer contours of the image. The shape context of corresponding points on similar shapes S and S' will be similar. So we use shape context to find the correspondence between two shapes. The idea is to build a histogram based on the sampled points in Polar coordinates system. So for each point we get the Euclidean distance and angles against all other points in the sampled set and normalize them. Then, based on the logspace map, we count number of points that occur in each map region. Each such logspace map define the shape context descriptor for that point in one image.

III. Design and Experimentation

We compare our model on the basis of performance, accuracy, time complexity, sensitivity and specificity by varying different parameters with the classifier. In the above section, we discussed different techniques and selection of parameter values for the implementation of KNN classifier. Here, we will discuss different experimental setups designed using those parameters and techniques. We mention any challenges faced while running these experiments that become the deciding factor in either going ahead with that experiments or compromising the setup. We also print the corresponding results and analyze them using (a) Accuracy tables, and (b) Confusion matrix, (on Easy and Hard dataset). We split both the Easy (for 'a', 'b') and Hard datasets (for all 8 labels) into training and validation sets for these experiements. Also, multiple parameters are changed during the code implementation to get the optimal outcomes while below we explain their impact individually for a clear understanding.

Simply KNN (varying K value)

We vary the odd values of K (first hyperparameter) from 1 to 100. We don't take even values to avoid getting any ties between class labels. In this setup, we take non-processed Easy and Hard dataset one-by-one which has padded images of size 54x54 and apply the scikit-learn's KNN method that by default uses the 'Minkowski distance'.

We check that with K taken as very large value, the precision of the decision boundary worsens. Also, if K is same as the number of inputs, the model predicts same value irrespective of the input.

KNN for downscaled images

In this setup, we downscale the images in both Easy and Hard datasets to 28x28 from 54x54 and input those downscaled images to 1-NN algorithm. The accuracy improves from 89.75% (for 54x54 input size) to 93.50% (for input 28x28 input size) for Easy dataset, classifying 'a' and 'b', which is expected from the above discussion in Section II. *Fig2.* shows the accuracy plots for simple KNN applied to both 54x54 and 28x28 type datasets.

KNN with dimensionality reduction

This experiments involves applying PCA, the popular statistical analysis method of dimensionality reduction based on variance of the dataset, on both Easy and Hard datasets. We first take non-processed images of 54x54 size and normalize them using the corresponding class mean and co-variances. We do the similar step for images of size 28x28. This way we now have 4 types of datasets (Easy and Hard for 54X54 and 28x28 types). We apply PCA on these different sets and vary the percentage of variance we take for the newly reduced feature space. *Fig3.* shows a 54x54 and 28x28 images before and after PCA is applied. Here, we compare the model accuracy for 4 different datasets with 6 different %variance in PCA, i.e. 95%, 90%, 80%, 70%, 60%, 50%. The number of features comprising the new feature-space depends on these amount of variance. *Fig4.* shows the corresponding accuracy tables and plots. We check that accuracies for both 60% and 50% are comparable. For 60% variance, we get maximum accuracy for 1-NN while for 50%, model exhibits less variance in accuracy for varying k.

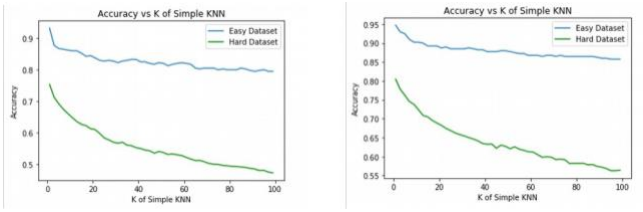

Fig2. (a) For 54x54 dataset, (b) For 28x28 dataset

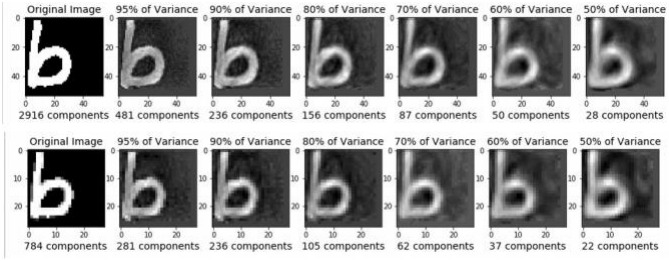

Fig3. (a) For 54x54 image, (b) For 28x28 image

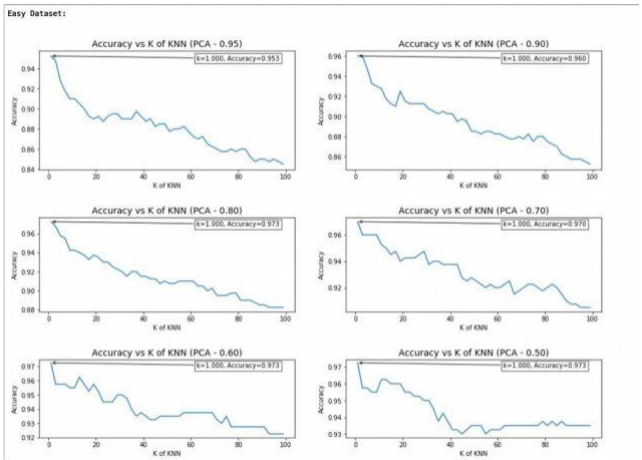

Fig4(a). For 28x28 Easy dataset – Accuracy vs K (kNN) plots.

Experiment results for Easy Dataset i.e. (a and b):

	Variance %	Feature Vector Dimension	Performance
0	100	784	0.9350
1	95	281	0.9525
2	90	190	0.9600
3	80	105	0.9725
4	70	62	0.9700
5	60	37	0.9725
6	50	22	0.9725

Experiment results for Hard Dataset i.e. (a, b, c, d, h, i, j, and k):

	Variance %	Feature Vector Dimension	Performance
0	100	784	0.775000
1	95	367	0.801875
2	90	251	0.828125
3	80	136	0.863125
4	70	79	0.890625
5	60	47	0.892500
6	50	28	0.877500

Fig4(b). For 28x28, accuracy tables.

Experiment results for Easy Dataset i.e. (a and b):

	Variance %	Feature Vector Dimension	Performance
0	100	2916	0.8975
1	95	481	0.9250
2	90	309	0.9425
3	80	156	0.9625
4	70	87	0.9700
5	60	50	0.9775
6	50	28	0.9650

Experiment results for Hard Dataset i.e. (a, b, c, d, h, i, j, and k):

	Variance %	Feature Vector Dimension	Performance
0	100	2916	0.735625
1	95	830	0.773750
2	90	497	0.882500
3	80	227	0.846250
4	70	118	0.884375
5	60	64	0.896250
6	50	36	0.901250

Fig4(c). For 54x54, accuracy tables.

KNN with different Distance Metrics

By default, scikit-learn's KNN classifier method uses Minkowski distance for multi-dimensional data points, which can be derived to Euclidean Distance for *q=2* and to Manhattan Distance for *q=1* in the table in Section II. We apply the KNN algorithm passing different distance metric after each completion and compare the results. *Fig5*. shows the corresponding accuracy table

	Distace Metric	For Easy Dataset	For Hard Dataset
0	minkowski	0.9725	0.892500
1	euclidean	0.9725	0.892500
2	hamming	0.5550	0.171875
3	manhattan	0.9800	0.878750

Fig5. Accuracy of KNN with different Distance Metrics

KNN with Shape Context Descriptors

We try to implement the procedure and python code as explained in [1],[5]. After extracting the shape contours of different images, we divide the points in each contour into equal parts which is equal to number of samples we require for the descriptors, thus ensuring that the points selected randomly will never be from only a small zone of the image. Taking sufficient samples from the contour relatively matches the approach explained in [1] and the distance between sampled points will also be minimum producing a meaningful shape context. But, then this increases the time complexity for computation of different steps involved in getting the matched descriptors by the order of $O(n_5)$, n being the input size. For example, in order to even compute for 'a' and 'b', we have an input of size 400x1200x100x100x60 for matching two images. Here, 400 is the validation set size, 1200 is the training set size, 100 is the sample points size, 60 is the number of bins in the logspace map descriptor of the shape. In order to avoid getting into such slow expensive computation, we try to tune some of these values. Number of bins is taken as standard value where there 5 bins for logarithmic radial distance and 12 bins for the angle between any two points. So, we try with taking 50 sample points from the contours by dividing the contour space into 50 equal parts and selected at random one point from each. We also try taking 10 samples from the contour space and run the algorithm for only 50 validation image points at a time thus reducing the complexity to 50x1200x10x10x60. This generates the outcomes significantly very fast but highly compromising on the expected accuracy for recognizing 'a'
& 'b', which comes out be only ~80%. The referred paper and respective the blog on its implementation both support the slowness of computing the bipartite Hungarian graph matching step and then of computing the affine transformations to avoid which they recommend using cosine similarity as the criteria to matches these shape descriptors. Corresponding boundary extracted and sample selection images for this experiment can be found on this paper's implementation GitHub repository.

IV. Conclusion

The experimental set-ups are not independent of the order in which we vary different parameters. It also should be noted that KNN doesn't learn the model but simply store the distance metric for each training data point implying that it can only approximate 'local' results. This also makes KNN computationally highly intensive for large datasets which can be improved by using special KD-tree or ball-tree data structures [8]. We observe that we get best accuracies for 1-NN here implying that the input data has a lot of similarity among each other for 'a' & 'b', thus k > 1 computes mostly similar structural distances in both types of images. However, we expect to perform even better by improving the computation of shape descriptors. Lastly, from all the experiments performed, we see that we get best accuracy of identifying 'a' & 'b' in case of applying KNN with PCA (60% variance). *Fig6.* we show a corresponding confusion matrix for 1-NN and 28x28 Easy (a) & Hard (b) imageset.

Confustion matrix for Easy Dataset

```
[[178   8]
 [  3 211]]
```

Confustion matrix for Hard Dataset

```
[[190   0   1   5   2   2   2   1]
 [  2 183   0   6   9   4   0   2]
 [  4   0 184   3   0   2   1   2]
 [  3   1   0 172   2   2  13   1]
 [  0  13   2   0 155   3   0  14]
 [  1   1   0   3   2 179   7   0]
 [  0   1   1   5   3   7 186   1]
 [  5   3   1   3  21   4   1 179]]
```

Fig6. (a) & (b)

References

[1] Belongie, S., Malik, J., & Puzicha, J. (2001). *"Shape context: A new descriptor for shape matching and object recognition."* In *Advances in neural information processing systems* (pp. 831-837).

[2] Liu, C.L., Nakashima, K., Sako, H. and Fujisawa, H., 2003. Handwritten digit recognition: benchmarking of state-of-the-art techniques. *Pattern recognition*, *36*(10), pp.2271-2285.

[3] LeCun, Y., Jackel, L. D., Bottou, L., Brunot, A., Cortes, C., Denker, J. S., ... & Simard, P. (1995, October). Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks* (Vol. 60, pp. 53-60).

[4] Yann.lecun.com, '*THE MNIST DATABASE of handwritten digits*' [Online]. Available: *http://yann.lecun.com/exdb/mnist/* . [Accessed: 01-Dec-2019]

[5] Medium.com, *'Shape Context descriptor and fast characters recognition'*, Available: *https://medium.com/machine-learning-world/ shape-context-descriptor - and-fast-characters-recognition-c031eac726f9* . [Accessed: 01-Dec-2019]

[6] Makkar, T., Kumar, Y., Dubey, A. K., Rocha, Á. and Goyal, A., 2017, December. "*Analogizing time complexity of KNN and CNN in recognizing handwritten digits*." In *2017 Fourth International Conference on Image Information Processing (ICIIP)* (pp. 1-6). IEEE.

[7] Otair, D., 2013. "*Approximate k-nearest neighbour based spatial clustering using kd-tree.*" *arXiv preprint arXiv:1303.1951*.

[8] Mikolajczyk, K. and Matas, J., 2007, January. "*Improving descriptors for fast tree matching by optimal linear projection*." In *2007 IEEE 11TH INTERNATIONAL CONFERENCE ON COMPUTER VISION, VOLS 1-6* (pp. 337-344).

[9] Jiang, S., Pang, G., Wu, M. and Kuang, L., 2012. "*An improved K-nearest-neighbor algorithm for text categorization*." *Expert Systems with Applications*, *39*(1), pp.1503-1509.

[10] Hwang, W.J. and Wen, K.W., 1998. "*Fast kNN classification algorithm based on partial distance search.*" *Electronics letters*, *34*(21), pp.2062-2063.

[11] Dutt, A. and Dutt, A., 2017. "*Handwritten digit recognition using deep learning.*" *Intl. J. of Advanced Research in Computer Engineering & Technology*, *6*(7), pp.990-997.