Handwritten Letters: Comparison of Classification Techniques

By,
Rajiv Iyengar
Deep Joshi

Background

A common problem in the field of Machine Learning is the classification of handwritten letters. The process of classifying characters is known as Optical Character Recognition also known as OCR. This technology is vital for many industries and can be used for reading postal addresses on packages, bank check amounts, and automating the processing of forms. In an academic setting, students usually prefer writing on paper to typing notes as it can be distracting. To jot down every word that the professor says is very difficult. So when students take copious notes, the handwriting tends to get messier which leads to unclear notes. The studying time is hampered trying to understand what's written. If a software could translate their handwritten notes to a printed alphabet it would be a time saver for most students and would help them understand their notes better. To achieve this, the first step is to understand and classify the alphabet that is written which is the problem we are trying to address.

From Kaggle, we obtained a data set of handwritten Russian letters that had already been labeled and digitized into PNG images. The data contained 3 sets of handwritten letters with each set using different backgrounds. For our purposes, we used the second set which contained handwritten examples of all Russian letters on a plain white background. For each letter, several handwritten examples were present with varying positions within the frame of the image and in varying colors. The images did not have consistent sizes and some had different dimensions from the rest.

We sought to compare several different Machine Learning techniques and evaluate them based on accuracy, ease of use, and computational efficiency (time). We compared Gaussian Naive Bayes, K-nearest neighbors, Multi-Layer Perceptron, as well

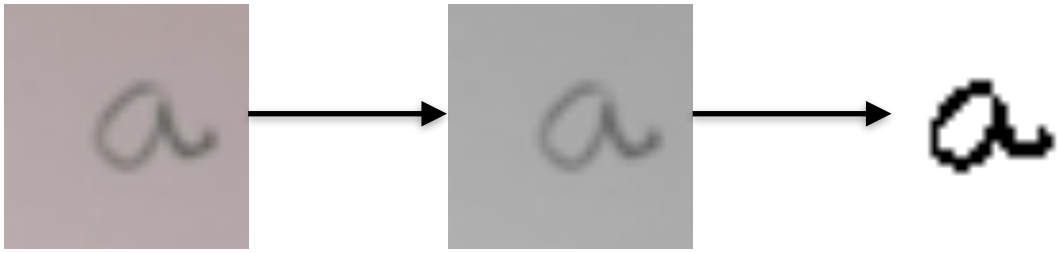
as a Linear Discriminant Classifier. We tried permutations and combinations of these as well to see which determined better test results. All of our methods were implemented using Python and the NumPy and Sci-Kit Learn libraries.

Methods

To prepare our data we processed the images using Python Image Library (PIL). First, the images were imported and converted to grayscale, then resized to a 32x32 image. Then the grayscale images were converted to 1-dimensional NumPy arrays of pixel intensity values. Next, we applied Otsu thresholding to binarize the images. This allowed us to remove a lot of noise from background pixels and amplify the intensity of the foreground pixels. We then split our data, yielding 80% for a training set and reserving 20% for testing our fitted models.

After processing our data, we began by performing a combination of Principal Component Analysis and Linear Discriminant Analysis. We reduced the dimensionality of our data using PCA and fed the transformed data into a Linear Discriminant Classifier. Using an iterative process, we adjusted the proportion of scatter preserved after PCA while monitoring the accuracy of our Linear Discriminant Classifier. Additionally, we attempted to use a K-nearest neighbors classifier. We used an iterative approach to determine the ideal number of neighbors to use for estimation.

Additionally, we attempted Gaussian Naive Bayes classification. Gaussian Naive Bayes (GNB) assumes a multinomial gaussian distribution for each observation in the data and that each class in the data follows a unique multinomial gaussian distribution. For each class, GNB estimates the parameters of the multinomial distribution from the



Raw Image Grayscale Conversion After Otsu Threshold

training data. To classify a new observation, its probability is calculated using the estimated multinomial gaussian distribution for each class. The observation is assigned to the class who's distribution yields the highest probability of that observation.

Lastly, we implemented a Multi Layer Perceptron to classify our data. We did not use any dimensionality reduction techniques for this classifier and opted to use our full data. Again, we used an iterative approach to tune our models hyper-parameters. We began by finding an optimal number of hidden layers that yielded high accuracy on the test set, while avoiding a very high accuracy score on the training set to avoid overfitting. Next, we adjusted the number of neurons in the hidden layers to further optimize accuracy. Our workflow can be seen in Figure 1.

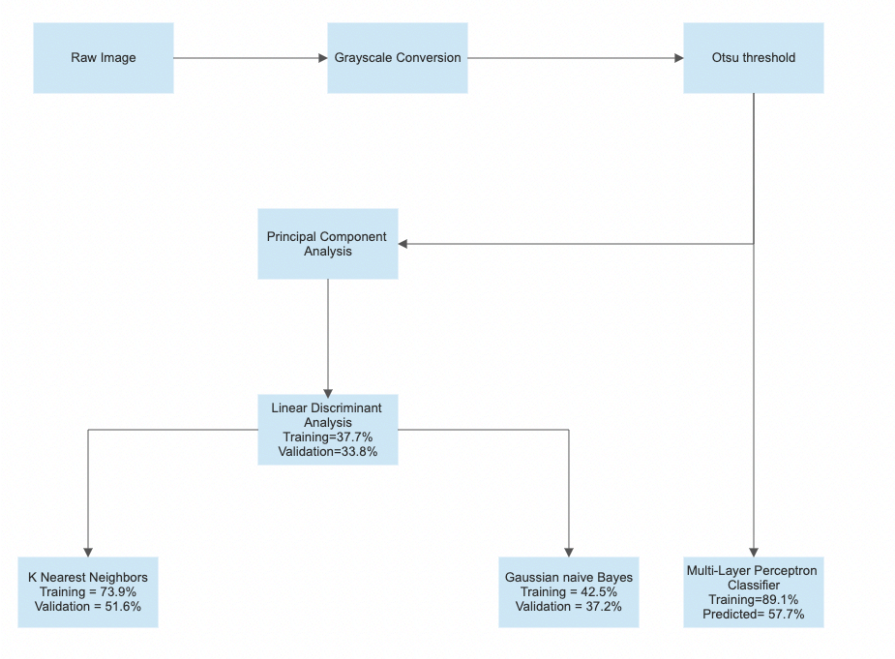


Fig. 1: Flow chart of processing workflow

Results

In order to measure the accuracy of our classifiers, we used a balanced accuracy score. This calculates the proportion of observations that were classified correctly weighted by the number of observations in each class. This gives us a better idea of how well the classifier performs overall across all classes. Additionally, we calculated confusion matrices and heat maps for all classifiers.

In general, a combination of PCA & LDA performed poorly on our data set. 5 A combination of Principal Component Analysis and Linear Discriminant Analysis had relatively low accuracy on our data yielding only 33.8% balanced accuracy. Gaussian Naive Bayes was only marginally better with 37.2% accuracy. K-Nearest Neighbors performed very well with 51.6% accuracy. However, the best performing classifier overall was the Multi Layer Perceptron with 57.7% accuracy.

In terms of computational efficiency, MLP was by far the most time consuming classifier. Training the model takes a significant amount of time in the order of minutes as opposed to seconds with GNB, LDA, and K-nearest neighbors. Additionally, the model will have to be trained multiple times in order to test different hyper-parameters and optimize the number of hidden layers and neurons. On our data, finding an optimal hidden layer size took about 2 days. Without such optimization, MLP failed to outperform GNB. For this dataset, we settled on 5 hidden layers with 715 neurons in the first hidden layers, and linearly decreasing to 250 neurons for the final hidden layer.

K-nearest neighbors worked exceedingly well considering it's speed, efficiency, and ease of use. Without any data transformations or tuning it yielded relatively high accuracy. Within just 3 iterations of fitting the model and testing, we had landed on the optimal number of nearest neighbors to use to classify. For our data, 3 nearest neighbors yielded optimal results.

	Training Set Accuracy	Test Set Accuracy
PCA & LDA	37.7%	33.8%
GNB	42.5%	37.2%
KNN	73.9%	51.6%
MLP	89.1%	57.7%

Fig. 2: Training and test set accuracy by classifier

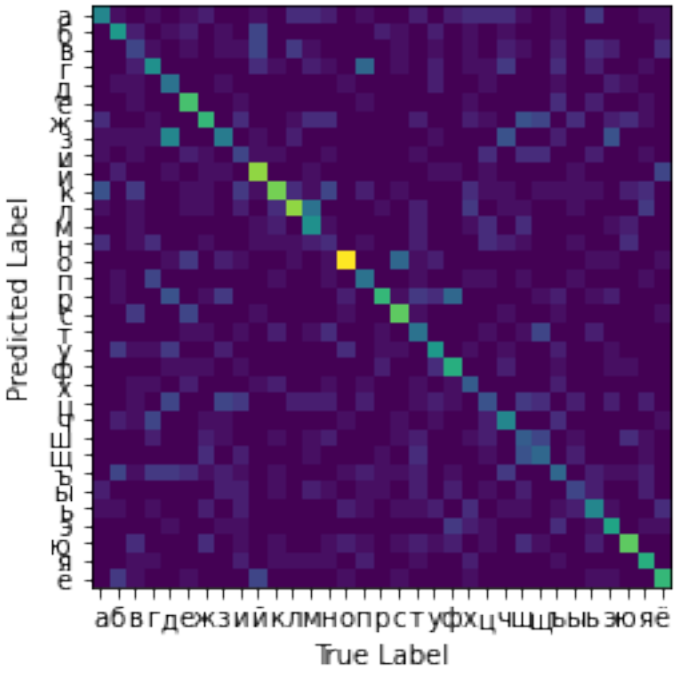


Fig. 3: Heat Map for PCA & LDA

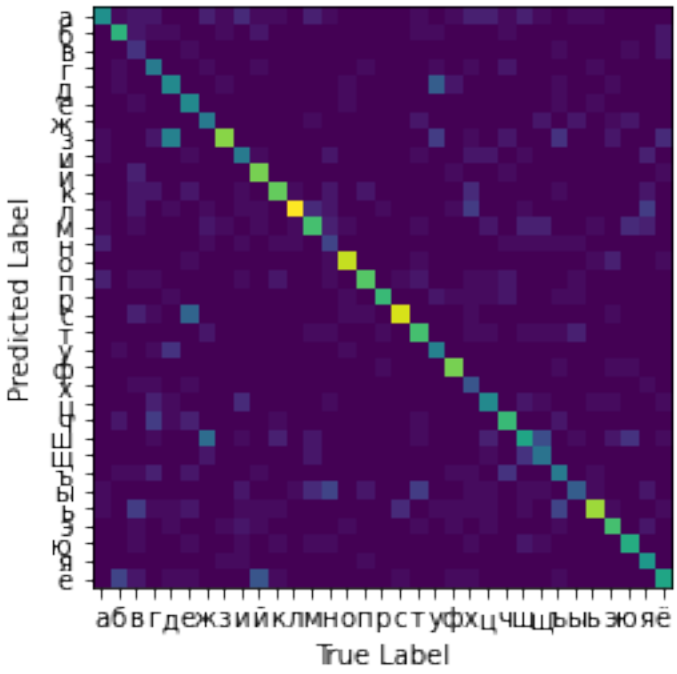


Fig. 4: Heat Map for MLP

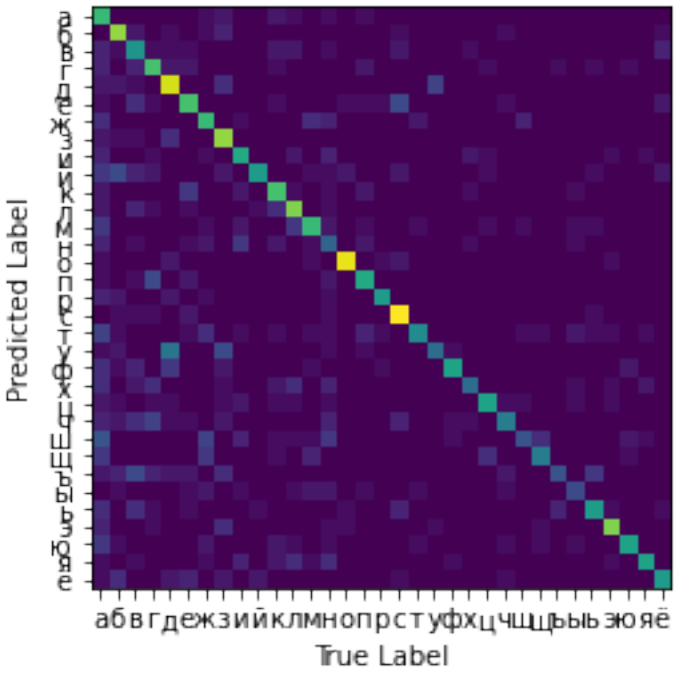


Fig. 5: Heat Map for KNN

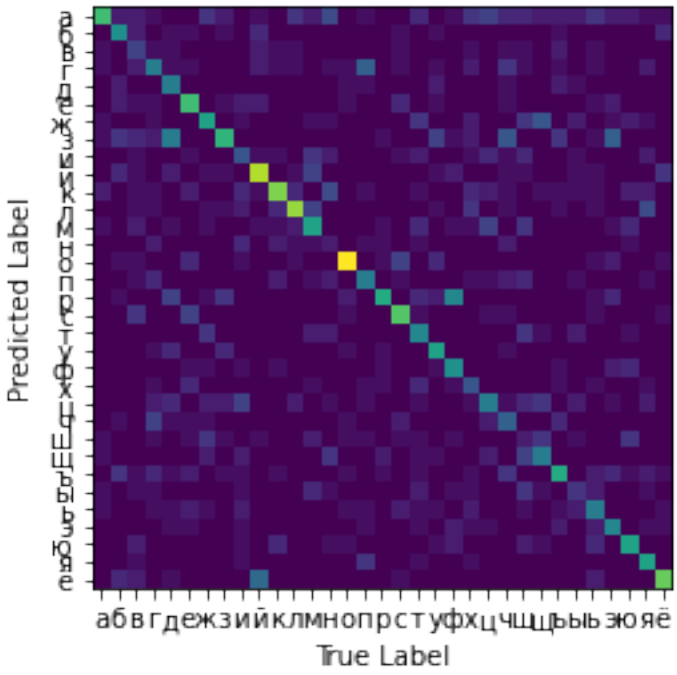


Fig. 6: Heat Map for GNB

Discussion

We sought to try multiple machine learning and classification techniques in order to classify a set of handwritten Russian Letters. Our workflow for the classifiers was as outlines in figure 1. Overall, two techniques stood out for accuracy which were Machine Learning Perceptron and K-nearest neighbors. A combination of PCA and LDA were not effective as a classifier, but were helpful in reducing the dimensionality of the data and increasing the accuracy of the KNN classifier.

We believe two factors contributed to the inaccuracy of the MLP specifically. These were the characteristics of the data set as well as our image processing steps. Our dataset contained several images of handwritten letters, however the

letters were not necessarily centered within their frames. This can be challenging for an MLP to interpret. One approach might be to use a Convolutional Neural Network as they can better understand translations of images. An alternative and potentially more efficient approach would be a more sophisticated image processing step in which the images are centered before fitting models.

Another approach to improve robustness of our model would be to sue image augmentation. This process involves taking a data set of images, and then performing several rotations and translations on those images and adding them back to the data set.