Comparing different classification algorithms for uppercase, lowercase and digit recognition

By: Shantanu Dixit, Daniel Ehrmann and Mara Hiemenz

Abstract

Character recognition is a famous Machine Learning application and is used in a variety of ways today. There are many machine learning algorithms that can make character recognition software, each with differences in their accuracy and efficiency. In this research paper, we discuss the advantages and disadvantages of using three different classification algorithms for digit, uppercase letter and/or lowercase letter recognition. The classification algorithms used include Convolutional Neural Network (CNN), K-Nearest Neighbors (KNN) and Support Vector Machine (SVM). Implementations of these algorithms were done with the help of Scikit-learn packages, Tensorflow and Keras. Each implementation used the EMNIST dataset for training and testing. A review of previous research literature indicates that CNN gives the most accurate results while having a large computation time for training but a small computation time for testing/ predicting. We will describe the accuracy and runtime measurements for each of the three algorithms and make a comparison between the algorithms to determine which is the best in terms of accuracy, and which is the best in terms of efficiency. We have tested each algorithm on two categories: digit recognition and digit, uppercase, and lowercase letter recognition. In each category we specifically compared accuracy (which is how correct the predictions made by the algorithm are), time taken to train, and time taken to make a prediction for each algorithm. Our comparative analysis between the three classification algorithms suggests that CNN algorithm yields the most accurate results, while KNN algorithm trains the most efficiently and CNN predicts the most efficiently.

1. Introduction + Algorithms Explained

Before diving into our findings, let's discuss the algorithms we are investigating. Every algorithm we have tested is a supervised classification learning algorithm (SL), which means the algorithms are trained using labelled data available to us. Given a dataset containing (x,y) i.e data, represented by x, and the corresponding label, represented by y, the algorithms aim to learn a function that predicts y given x. We train the algorithm using (x,y) and test it using data not already applied on the algorithm, which can be represented by (X,Y) where X is unseen data and Y is unseen label [1]. For SL algorithms, data is divided into two categories: the training set, used for training the model, and the testing set, used to validate the model. This technique avoids overfitting, or the memorization of examples used to train the model.

The k-Nearest Neighbors algorithm, or KNN algorithm, was introduced in 1951 by Evelyn Fix and Joseph Hodges [11]. It works by selecting k number of examples that are in the training set closest to a query (input). In the case of classification, after k examples are selected, the most frequent label is chosen as the label for the given query. The time and space complexity for training a KNN model is O(n*d), where n represents the number of examples, or data points, and d represents the number of features that determine each datapoint. For testing, on the other hand, the time complexity is O(n*k*d) where k is the number of nearest neighbors to consider [2].

In order for character recognition with KNN to work the images must be relatively consistently cropped such that features appear in similar locations across the dataset. Each cell of the image where a

histogram is created will have its own one-dimensional surface to be placed on and compared to other samples against in distance, making consistency in feature placement on the image integral to accurate classification. Additionally, this means that the barriers between similar characters such as 0 and O may become somewhat blurred, as they will be placed in the same cluster in N-dimensional space since they're locational gradients are nearly identical. It's up to chance whether the k samples taken happen to be of the correct category.

Since the algorithm relies on selecting the closest examples, it calculates the distance between the query and every existing point, which makes it slow for large datasets. As suggested by the time complexity of KNN, a higher number of dimensions would also slow down the algorithm significantly. The algorithm's dependency on "good value" for k is also one of its limitations. Outlier sample points in the dataset, called anomalies, can cause incorrect classification to occur if k (in KNN) is too low. A solution to this is to get rid of the outliers using anomaly detection techniques, which adds to the runtime of the algorithm. Since when a query is performed, distance is computed for every point in the dataset, the algorithm can be very inefficient for large datasets. To rectify this, data structures such as k-d trees and ball trees can be used to reduce the number of distance computations by pruning portions of the dataset so some examples aren't searched [6]. This reduces runtime of queries, but requires extra memory. We decided not to implement these as we have a limited understanding of the data structures. The KNN algorithm has proved to be highly accurate for smaller datasets. A 97.67% accuracy for MNIST numeral digits has been reported by S. Roy in [16].

The Support Vector Machine algorithm, or SVM algorithm, was first introduced by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963 [12]. It classifies linearly separable data by finding a hyperplane that separates data by considering support vectors, or the closest points to the hyperplane, and maximizing the distance to them. If the data is not linearly separable, then transformations are applied to it, mapping the sample points to a higher dimensional space. Because of this, the decision boundary, or the hyperplane that separates the data, is also of that higher dimensional space. A technique known as the kernel trick is used to project the non-linear data to higher dimensions. So called Kernels are functions used to accomplish this task while avoiding complicated calculations. Popular kernels include the Polynomial kernel, Gaussian kernel, linear kernel and the Gaussian radial basis function (RBF) [3]. For the purposes of our experiment, we have used the linear kernel and the RBF kernel. The time complexity for training an SVM model is O(n^2) where n is the number of data points. The time complexity for testing an SVM model, on the other hand, is O(kn) where n is the number of data points and k is the number of support vectors. The space complexity for SVM is O(k) where k is the number of support vectors as k vectors are stored in memory [4].

SVM is better at dealing with outliers than KNN as it only considers support vectors to find a linear classifier. It is also more memory efficient as only the support vectors are needed, which is a subset of the training points. There are, however, limitations of SVM to consider. Because of the O(n^2) time complexity of training an SVM model, SVM does not perform well with large datasets. Additionally, kernel functions are difficult to understand, making SVM less interpretable and outputs less predictable. A popular alternative to SVM is the perceptron algorithm, which aims to find a linear classifier without an optimized/ maximum separation distance. The perceptron algorithm is easy to understand, which makes it interpretable, unlike SVM. The cost, however, is that the hyperplane the perceptron algorithm generates varies with the same sample points and solely relies on the initial weights of the perceptron. The SVM algorithm has achieved a higher accuracy than KNN for character recognition. An accuracy of 98% has been achieved for bi-class and multiclass character recognition [17].

We used the popular feature extraction method called Histogram of Oriented Gradients to extract features. With the help of horizontal and vertical Gradients, it is able to detect intensity (magnitude and direction) at portions in an image to get the useful features of that image. Each character is represented as a point in an n-dimensional space and the axes represent the features. Points closer to one another have similar features (or rather, their features resemble each other's). This gives rise to clusters in the n-dimensional space that contain points representing just one character. To reiterate, SVM achieves classification of points by dividing the space using an optimal hyperplane (which has a maximal margin), which is a line in a 2D space and a plane in 3D space, such that each side has the same type of points. To find the hyperplane, we used our training set which has characters each labelled with the character they represent to train our model. The hyperplane starts out as an arbitrary hyperplane at first, and as the model is trained, the SVM comes closer to achieving optionality. Once the model is trained and an optimal hyperplane is found, new points can be classified based on where they are in the space relative to the hyperplane. Letters that look similar, such as the letter I and the digit 1, have very similar features, causing their clusters/ points to be very close together in the space. This results in small margins (or closer support vectors to the hyperplane) which causes the model to confuse one character for the other. If the margin is large, however, then the hyperplane can classify new points with greater accuracy.

A Neural Network, or NN, was first proposed by Warren McCullough and Walter Pitts in 1944 [13]. A neural network is a collection of algorithms that tries to mimic the way neurons of the brain work. Neurons in the brain transmit signals when they receive input, causing other connected neurons to subsequently fire. Similarly, a neural network is made up of layers of neurons, which represents a mathematical function. Each layer is connected to every neuron in the layer before. Neural networks typically have the input layer, which takes in inputs, multiple hidden layers, that perform calculations on the inputs, and the output layer, which delivers the output. The edges (or the synapses in biology) connecting the neurons have weights (as numbers). These weights are used to get a weighted sum, which is determined by multiplying each neuron activation (or neuron output) in a layer with the weight connecting the neuron to the next layer. A function is used to transform the weighted sum into an output. This function is called the activation function. Examples of activation functions include RelU, Sigmoid and Tanh. We have used the leaky RelU activation function, which is based on the RelU function and, when graphed, has a subtle negative slope for negative x-values (instead of the flat slope that RelU has for negative x-value).

There are three important neural networks: Artificial Neural Networks (ANN), Recurrent Neural Networks (RNN), and Convolutional Neural Network (CNN). CNNs are most commonly used when dealing with image data, and are used by us for character recognition. Instead of layers being connected to every neuron in the previous layer, neurons are connected to neurons close to it and have the same weight. CNNs have 2 special layers called the convolution and the pooling layer. The convolution layer places a filter over an array (of pixels) to get a feature map. The pooling layer reduces the dimensions of the feature map to reduce computations in the network. [5]

The convolution layers are integral to how the neural network can “see” an image. The filters placed overtop of and slid around the image are scanning for edges that resemble the edge the filter depects. The resulting samples at each position of the filter act as a mapping for where that edge occurs throughout the image. When the image reaches the fully connected layers each neuron receives values that represent the presence of features at particular locations on the image. The flattening retains the positional nature of the feature detection and the weights assign importance to the presence of features in relation to one another’s position for the application of categorization.

The time and space complexity of a neural network varies from type to type and number of hidden layers, among other things. The memory requirement is much less compared to the other algorithms as only weights, activations and input data are stored. Furthermore, CNN takes very little time in the training phase compared to the testing phase. While it is true that CNN is one of the most useful ML algorithms today, it still has its limitations. The most notable limitation is that it requires a large training set for it to work effectively. Another limitation is that, if the number of hidden layers is too high, training may take too long. There aren't many famous alterations to CNN, but there are other algorithms that can be used for character recognition such as decision tree and Naive Bayes, both of which are SL algorithms. CNN has widely been reported as the best algorithm for character recognition based on accuracy. CNN has achieved an accuracy of 99.77% when tested with MNIST handwritten digits dataset [18].

The three ML algorithms have been extensively compared implementation-wise. [15] applies SVM, KNN and Deep Neural Networks for car ladle digit recognition to compare their accuracy. It was found that CNN produces the most accurate results of 97.6%. Furthermore, accuracies of the algorithms were in the range of 98-100%.

2. Methodology

As these algorithms have been implemented and explored extensively by others before, we utilized pieces of other publicly shared implementations of each algorithm. This ensured the results we found were consistent with how these algorithms are typically implemented. All of our implementations share the EMNIST dataset [7]. Making the dataset accessible to each algorithm implementation involved opening a CSV representation of the dataset as a numpy array. The CSV representation and code snippet to convert the CSV data to workable images arrays was borrowed from Kaggle user Ashwani Jindal [8]. The code snippet first uses the pandas library to read the CSV and the iloc function to separate out each image into its own row in an array. From there the 1D array of image pixel data is converted into a 28x28 array and rotated 90 degrees, as each image is rotated 90 in the opposite direction in the original EMNIST dataset.

The KNN algorithm was guided by Hussein Moghnieh's article on the website Towards Data Science [9]. The article presents a solution using the skimage library's Histogram of Oriented Gradients function to extract features from each image in the imported dataset. The feature extractions are added to an array and then split into testing and validation groups using the sklearn's train_test_split function. Then, using sklearn's KNeighborsClassifier a KNN algorithm is created, trained, and scored for accuracy.

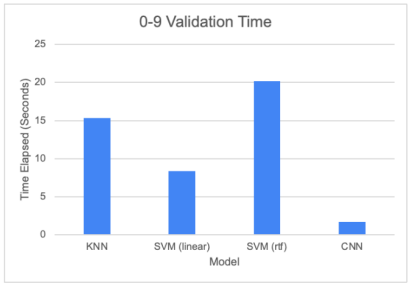
For the SVM algorithm, code from the KNN implementation that extracted features from each image and made a split into testing and validation sets was reused. Instead of using sklearn's KNeighborsClassifier, sklearn's SVM package was used to train the model, make predictions, and evaluate the model. Sklearn's SVM package provides a method called SVC to create an SVM algorithm using 3 parameters called the kernel, degree and gamma [14]. Code for the SVM algorithm was heavily influenced by the channel Cloud and ML Online's video on YouTube titled "Support Vector Machine - SVM - Classification Implementation for Beginners (using python) - detailed" [19]. Our first algorithm used the linear kernel, while the second used the RBF kernel. After creating the algorithms, they were trained and evaluated for accuracy.

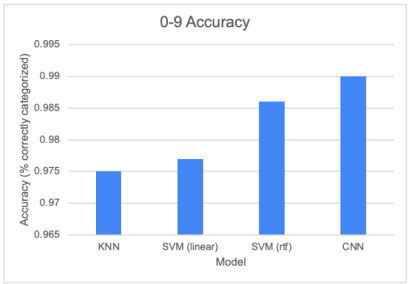
The CNN algorithm code used was provided by user Aditya Sharma's tutorial at the website Data Camp [10]. It takes advantage of the Keras library, which acts as a layer on top of the popular tensorflow

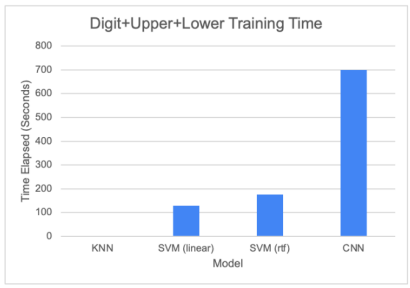
library. The initial section of the coe snippet reshapes the image array and alters the data such that it is compatible with what Keras functions will expect, ensuring the pixel values are floats between 0 and 1. Then, the category labels for the dataset are converted from integer numeric labels to one-hot encoding, as the Keras functions expect. The data is split similarly to how it was in the KNN implementation. The structure of the CNN is now established by adding sets of Keras functions to a sequential neural network model. The structure involved a convolution later activated using the LeakyReLU function and fed into a max pooling layer. This sequence is repeated 3 times before being flattened into a 1D list of vectors and fed to a deeply connected dense layer that breaks the output into one category of the number that was specified prior.

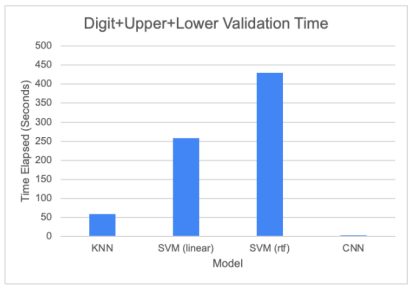
3. Experimental Studies

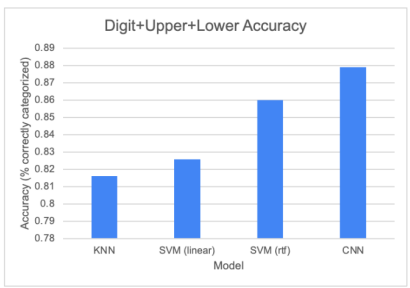
As all these algorithms can be used to classify input data we wanted to see how they compared to one another in the specific problem of classifying handwritten characters. Our goal was to identify how each algorithm compares in training efficiency, validation/prediction efficiency, and accuracy in this problem set. We implemented the three algorithms: KNN, SVM and CNN, for digit, uppercase letter and lowercase letter recognition. We have used two different datasets to test the algorithms. The first, a 9 character dataset containing every digit (0-9). The second, a 47 character dataset consisting of every digit (0-9), every lowercase letter (a-z) and every uppercase letter (A-Z), with letters (i-m), o, p, s, and (u-z) being shared between uppercase and lowercase classifications due to how the EMNIST dataset is arranged. We have measured the time it took to train and test each model and each model's accuracy for each class. Our findings are represented visually below.











In training time the KNN far outperformed the other algorithms. The KNN algorithm only needs to load the dataset into a plot where it can be compared against other incoming data, meaning the upfront cost of training the set is nearly imperceptible. The SVM algorithm is similar, however it has the added cost of needing to calculate and update the position of the hyperplanes dividing the data. CNN algorithm's complex sequence of processing layers make it very costly to train on large sets of data compared to the other algorithms.

In validation time the CNN algorithm performs the best. The process of classifying an image through the network without updating any weights is fairly inexpensive. Similarly, the KNN scales nicely to the larger dataset with more classifications, as it only needs to do K distance calculations for each image it classifies, regardless of the number of classification groups present. The SVM algorithm's training and testing runtime and accuracy was heavily dependent on the kernel used. SVM with the RBF kernel increased the training and validation time while increasing the accuracy for both classes compared to the linear kernel. The SVM algorithm with a linear kernel performed well in the 0-9 dataset, however when the number of possible classifications and size of the dataset increased is scaled less favorably compared to KNN.

Regardless of the number of possible classifications, each algorithm compared consistently with one another in accuracy performance. All algorithms' accuracy decreased considerably when increasing the number of possible classifications, making each algorithm less viable as solutions to handwritten character classification in applications where a reasonable level of accuracy is expected. This tells us that simply adding more possible classifications is not the best solution to the hand-written character classification problem. These algorithms do not scale nicely to larger sets of possible classifications. Instead it may be better to break the dataset into smaller subsets and have separate models for classification of each subset.

4. Conclusion

Our work could have been improved by spending more time optimizing each algorithm for each particular problem set. For example, the k-value chosen for the KNN was based on data collected in testing different k-values against the 0-9 set. We carried this k-value over for the larger 0-9, lower, and upper dataset. It may have been improved by choosing a k-value specifically for that set. Similarly, the number of epochs and batch-size for the CNN were chosen somewhat arbitrarily, driven by informal observations comparing when the accuracy began to stagnate from epoch to epoch while training on the 0-9 dataset. We also did very little to properly adjust for overfitting in the convolutional neural network. While this would have improved the accuracy of our work, it likely would not have affected the conclusions we came to specifically with regards to how each algorithm compares to one another. Additionally, it would have been interesting to explore further optimizations to solve the problem beyond improving each algorithm, and instead taking an approach that combines different layers can more accurately classify.

5. References

[1]: https://towardsdatascience.com/laymans-introduction-to-knn-c793ed392bc2#:~:text=Two%20offices%20of%20USAF%20School,nearest%20neighbor%20(kNN)%20algorithm.

[2]: https://medium.com/analytics-vidhya/knn-k-nearest-neighbors-1add1b5d6eb2

[3]: https://data-flair.training/blogs/svm-kernel-functions/

[4]: https://alekhyo.medium.com/computational-complexity-of-svm-4d3cacf2f952

[5]: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[6]: https://www.quora.com/Which-other-algorithms-methods-can-be-used-as-alternative-to-the-nearest-neighbours-algorithm-for-distance-measurement-and-which-performs-better-than-NN

[7]: https://www.nist.gov/itl/products-and-services/emnist-dataset

[8]: https://www.kaggle.com/ashwani07/emnist-using-keras-cnn

[9]: https://towardsdatascience.com/scanned-digits-recognition-using-k-nearest-neighbor-k-nn-d1a1528f0dea

[10]: https://www.datacamp.com/community/tutorials/convolutional-neural-networks-python

[11]: https://holypython.com/knn/k-nearest-neighbor-knn-history/

[12]: https://towardsdatascience.com/support-vector-machines-a-brief-overview-37e018ae310f

[13]: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414

[14]:
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[15]:
http://ceur-ws.org/Vol-2864/paper44.pdf

[16]:
http://ijariie.com/AdminUploadPdf/Handwriting_Recognition_using_KNN_classificatiob_algorithm_ijariie6729.pdf

[17]:
https://www.semanticscholar.org/paper/A-SVM-based-character-recognition-system-Sharma-Sasi/85da44e9c3c8381c1818ca3f5b9c16a5f4a40397

[18]:
https://ieeexplore.ieee.org/document/6248110

[19]: https://youtu.be/7sz4WpkUIIs