ISSN 0002-306X. Proc. of the RA NAS and NPUA Ser. of tech. sc. 2020. V. LXXIII, N2

UDC 621.337

AUTOMATION
AND CONTROL SYSTEMS

K.H. NIKOGHOSYAN, V.T. BEJANYAN

ANALYSIS OF K-NEAREST NEIGHBOR AND ARTIFICIAL NEURAL NETWORK FOR MULTICLASS CLASSIFICATION OF HIGH DIMENSIONAL DATA

In this article, we present two techniques for the multiclass classification of high dimensional data based on the K-Nearest Neighbors algorithm and Artificial Neural Network model. The incremental construction of mathematical models and a careful analysis of the experimental results are presented.

Keywords: classification, neural networks, large amount of data, mathematical modeling, algorithms, high-performance computing.

Introduction. In the last decade, the amount of data stored on the service side has become tremendous. ML algorithms for datasets with such properties should be extremely fast, scaled up easily with volume and dimensionality of input, should be able to learn from streaming data without introducing performance bottlenecks, and which is most important, should be easily deployable on hyper-scale cloud computing systems, which are widely used in high-performance computing problems. Deep Learning (DL), as a subset of ML, is well positioned to address these challenges. ML algorithms have a wide application range and are highly successful in handling high dimensional data classification problems. Classifiers can be split into two main categories: binary classifiers, which are used when there is only one feature of interest under the scope, and multiclass classifiers, which are the generalization of the classifiers described previously. There are two main choices for implementing pattern classifiers: Artificial Neural Networks and classification algorithms.

In this paper, we analyze the algorithm called k-Nearest Neighbors (k-NN) and Artificial Neural Networks (ANN), using the high dimensional dataset called CIFAR-10.

The k-NN is a supervised method which is rather simple to implement and train. But, on the other hand, time overhead during the classification process can be very high, especially with very large datasets. The main aim of this work is to give the comparative analysis of accuracies that can be achieved while applying this and other methods based on ANN. In case of non-linear classification problems, ANNs seem to be a more robust solution. Theoretically, an ANN is capable of learning the

shape of just any function, given enough computational power. ANNs, able to deal with complex relations between variables, non-exhaustive category sets and complex functions relating input to output variables, can show very high accuracies on the problems of high dimensional data classification. But on the other hand, on many datasets, a careful tuning should be done to prevent over- and under-fitting.

K-Nearest Neighbors (kNN). K-NN [1] is one of the widely used algorithm for classification problems. By its non-probabilistic nature, kNN is unaware of any possible distribution with the help of which data can be described. In other words, the model structure is determined by the data. The purpose of kNN is to use a labeled dataset in which data points are already labeled with correct classes and later can use this data to carry out classification. So there is no place for generalization of data, as with Artificial Neural Networks. In other words, there is no explicit training phase or it is very minimal. This also means that the training phase is pretty fast. The lack of generalization means that kNN keeps the training data. To be more exact, all (or most) training data are needed during the testing phase. In Figure 1 let us assume the green-colored point P, for which the label needs to be predicted. First, the algorithm finds the k closest points to P, after which it determines the most common label through all k closest points or 'nearest neighbors' and assigns that label to the data point P.

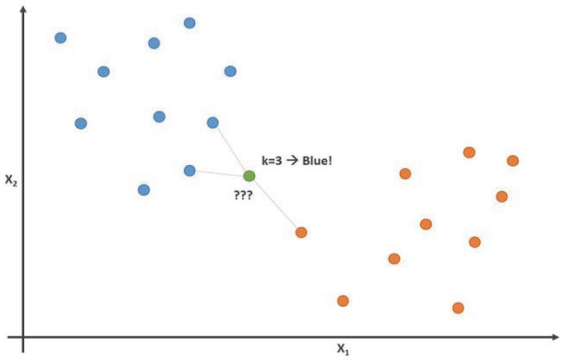


Fig 1. An example of kNN in a two dimensional space

To find the closest points, one can calculate the distance between the points using the distance measures such as Euclidean (1), Manhattan (2) or Minkowski (3) distance:

$$\sum_{i=1}^{n}(x_i - y_i)^2, \tag{1}$$

$$\sum_{i=1}^{n}(|x_i - y_i|), \tag{2}$$

$$(\sum_{i=1}^{n}(|x_i - y_i|)^q)^{1/q}. \tag{3}$$

In this article, we use CIFAR-10 dataset. The dataset was formed by collecting 60.000 images each of which is 32 pixels high and wide. Each image is labeled with one of 10 classes. To perform a train/val/test stages, the dataset was split into two parts with respectively 50.000 and 10.000 images in each.

As we have already said, kNN does not do any generalization of the data. In essence, kNN computes the distances between each pair of vectors, chooses the k nearest neighbors for the current one and assigns to it a most common label.

To compute the distance between two images we represented each image as the vector of pixels and then computed the distance between these two vectors I_1, I_2 using Euclidean (1) distance.

For the kNN, the training phase simply consists of remembering all the training data.

In Figure 2 implementation of cross validation with 5 folds for different choices of k is given. Differences between training and test vectors were calculated using the vectorized approach.

```
num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]
for step_k in k_choices:
    accuracies = []
    for fold_step in range(num_folds):
        test_data = X_train_folds[fold_step]
        test_labels = y_train_folds[fold_step]

        train_data = np.vstack(X_train_folds[0:fold_step] +
                               X_train_folds[fold_step + 1:])
        train_labels = np.hstack(y_train_folds[0:fold_step]
                                 + y_train_folds[fold_step + 1:])

        classifier.train(train_data, train_labels)
        dists = classifier.compute_distances_no_loops(test_data)

        y_test_pred, _ = classifier.predict_labels(dists, k=step_k)
        num_correct = np.sum(y_test_pred == test_labels)

        accuracy = float(num_correct / test_labels.shape[0])
        accuracies.append(accuracy)
    k_to_accuracies[step_k] = accuracies
```

Fig. 2. Implementation of kNN cross validation

Cross-validation. To apply the cross-validation technique, the dataset was split into 5 folds. On each iteration of cross-validation, 1 fold was used for validation and the remaining 4 folds - for training. In Figure 3 an example of 5-fold cross-validation run for the parameter k is shown. For every value of k starting from 0, the model is training on already specified 4 folds. When training is done,

the validation is done on the remaining fold. The trend line is drawn through the average of the results for each k and the error bars indicate the standard deviation. With the help of experimentation for a specific dataset, the best value for k can be derived. So, in this particular case it equals to k =7. If we used more than 5 folds, we might expect to see a smoother (i.e. less noisy) curve.

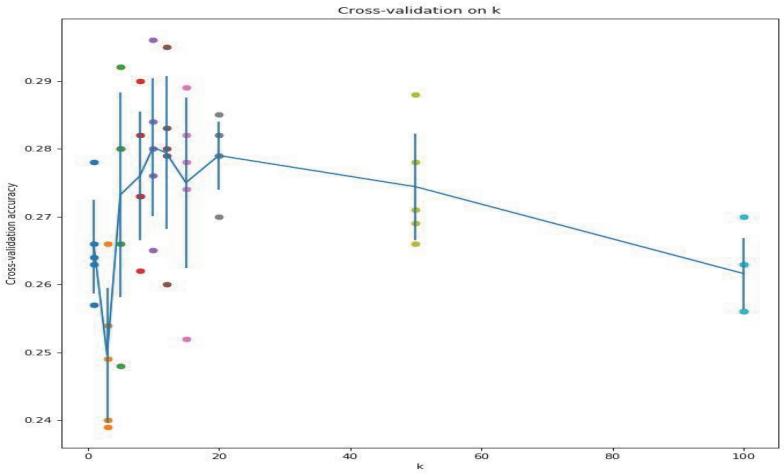


Fig. 3. Accuracies obtained during cross-validation

Artificial Neural Network. The model of an artificial neuron used for the construction of an artificial neural network (ANN) presented in Figure 4 is based on the formalization of the biological neuron, also known as perceptron [2,3]. In a trivial case ANN can be represented as a single-layer single-neuron network where neuron is represented by the model of perceptron. The perceptron itself can be decomposed into the following parts: input connections, vector of weights associated with them, bias, activation function and output connection.

The perceptron has the following mathematical model:

$$y(X, W, b) = sigma(W * X^T + w_b * b), \quad (4)$$

where X is the matrix of input features, W - the matrix vector of weights and finally b - the non-zero bias applied to that sum to avoid zero as a result in the case when one of the input vectors is a zero column vector.

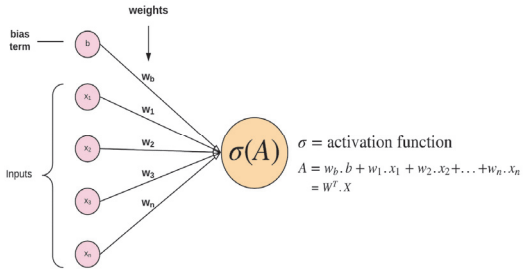


Fig. 4. Model of an artificial neuron

From formula (4) it is clear that perceptron represents the affine transformation to which the sigmoid function is applied to normalize the output of the matrix multiplication. A single artificial neuron can be used to implement a linear binary classifier such as SVM [5] or Softmax [6]. In case of multiple classes the multilayered, fully-connected, non-linear architecture of ANN was selected. The number of neurons in input and output layers are usually fixed. In case of the input layer, the number of neurons depends on the dimensionality of the input data. In case of very high dimensional data, various techniques of dimensionality reduction such as Principal Component Analysis(PCA) or t-Distributed Stochastic Neighbor Embedding(t-SNE)[7] can be applied to perform dimensionality reduction on part of or entire dataset to gain some knowledge of patterns present in the data. Due to its linearity, PCA is not able to present polynomial relationships between the features in the area of interest. In the presence of such correlations between the features, the t-SNE is mostly preferred because of its ability to preserve geometry at all scales. In other words, to present high dimensional data on non-linear manifold, it is important to place similar points close together, which is not possible by the linear algorithm, such as PCA. In Figure 5, an example of a result that can be obtained using the t-SNE technique for dimensionality reduction of high dimensional data is shown.

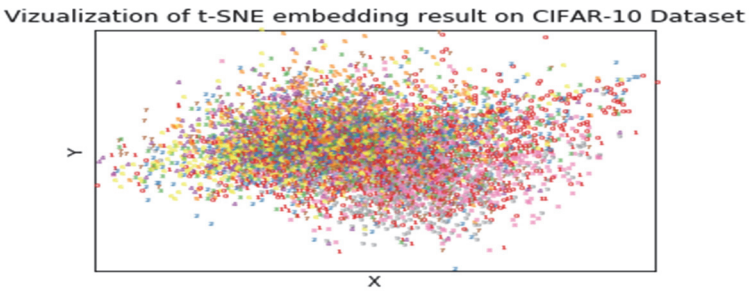
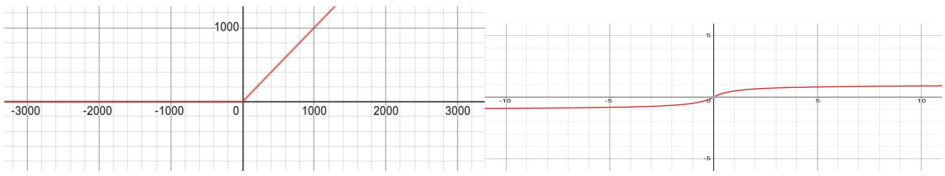


Fig. 5. t-SNE applied to CIFAR-10

ANN also consists of a hidden layer(s) and an output layer. The number of units in the output layer of ANN used for classification is roughly determined by the number of classes. The proposed ANN uses Rectified Linear Unit(ReLU) (5) to embed non-linearity. Formally ReLU can be represented by the following function:

$$ReLU(x) = max(0, x). \tag{5}$$

The network uses ReLU (Figure 6 a) after the first fully-connected layer.



a) *b)*

Fig 6. a - ReLU activation function; b - Normalized logistic sigmoid function

Softmax function was used to make the interpretation of class scores more intuitive. Particularly, Softmax allows to interpret the scores as probabilities assigned to training scores, which are obtained after each iteration of the ANN training. Suppose, the score column vector obtained after one iteration of training is denoted by $\boldsymbol{y}$ and has the size $\boldsymbol{C}$, where $\boldsymbol{C}$ is the number of classes, then Softmax for a particular class $\boldsymbol{c}$ and input vector $\boldsymbol{X}$ can be presented by the following formula:

$$\mathrm{P(C = c)_X} = \frac{\mathrm{e}^{y_\mathrm{C}}}{\sum_\mathrm{i} \mathrm{e}^{y_\mathrm{i}}}. \tag{6}$$

The loss function negative log-likelihood was selected. In case of Softmax function it is presented by the following formula:

$$L = \sum_s ln\left(\frac{e^{(wx_S+b)_S}}{\sum_i e^{(wx_S+b)_i}}\right). \tag{7}$$

The loss function presented in formula (7) is not stable regarding to the changes in b, so any addition to it will affect the solution space. As already noted, term b turns linear transformation into affine transformation which causes a change in the solution space for the linear model. In other words, presented as it is, loss function has multiple solutions, due to the fact that affine transformation can have different origins. To solve this problem, regularization techniques were used. The more robust version of the loss function also includes a regularization term $R(w, b)$. This dependence makes the optimization process more computationally intensive, but the results become more stable. The regulation term can be expressed by the following formula:

$$R(w,b) = \|w\|_2^2 + \|b\|_2^2, \tag{8}$$

where

$$\|x\|_2 = \sqrt{\sum_k^n x_k^2}. \tag{9}$$

The purpose of neural network training is to tune hyperparameters(e.g. weights, biases, learning rate) to minimize the given loss function. The most common way of finding such parameters is called Gradient Descent (GD). In practice, a modification of GD called Stochastic Gradient Descent (SGD) is used. The refinement of weights and biases using SGD is presented by the following formulas:

$$\vec{w} = \vec{w} - \eta\overrightarrow{\nabla_w}L, \tag{10}$$
$$\vec{b} = \vec{b} - \eta\overrightarrow{\nabla_b}L,$$

where η presents the learning rate of the algorithm, nabla operator $\overrightarrow{\nabla_w}$ is the gradient of the given matrix w. It is worth mentioning that a careful choice of learning rate should be made to better the chances in the search of local minima for the loss function. ANN implemented for classification problems is constructed out of 3 layers. It contains one input layer. The number of neurons in the input layer is roughly determined by the dataset. In case of CIFAR-10, each image contains 3 layers and 32 * 32 pixels within each layer, so the size of the input layer is equal to 3072. The input layer is fully-connected to the hidden layer. The number of neurons in the hidden layer, another hyperparameter for ANN, is randomly chosen from a predefined range and initialized once when the network is created. The network is finalized with an output layer with 10 output neurons. The number of the output neurons is equal to the number of the output classes against which classification is done.

In Figure 7, a Python implementation of the training code for a fully connected Artificial Neural Network is given. To optimize the Softmax function, Minibatch Gradient Descent with default batch size equal to 200 is used.

```
def train(self, X, y, X_val, y_val,
            learning_rate=1e-3, learning_rate_decay=0.95,
            reg=5e-6, num_iters=100,
            batch_size=200, verbose=False):
    num_train = X.shape[0]
    iterations_per_epoch = max(num_train / batch_size, 1)

    loss_history = []
    train_acc_history = []
    val_acc_history = []

    for it in range(num_iters):
      X_batch = None
      y_batch = None

      batch_indices = np.random.choice(num_train, batch_size)
      X_batch = X[batch_indices]
      y_batch = y[batch_indices]

      loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
      loss_history.append(loss)

      for key in self.params:
        self.params[key] -= learning_rate * grads[key]

      if verbose and it % 100 == 0:
        print('iteration %d / %d: loss %f' % (it, num_iters, loss))

      if it % iterations_per_epoch == 0:
        train_acc = (self.predict(X_batch) == y_batch).mean()
        val_acc = (self.predict(X_val) == y_val).mean()
        train_acc_history.append(train_acc)
        val_acc_history.append(val_acc)
        learning_rate *= learning_rate_decay

    return { 'loss_history': loss_history, 'train_acc_history': train_acc_history, 'val_acc_hi
story': val_acc_history}
```

Fig. 7. Implementation of the ANN training

Analysis. The aim of this section is to show how techniques presented above are used to obtain experimental results on the multiclass dataset. Specifically, how do the accuracies of kNN and ANN depend on the size of the training data and how the relative accuracy is changed during the increase of the training data. All experiments were carried out inside Jupyter Notebooks environment. For vectorized calculations, libraries such as Numpy were widely used. All plots were created with Matplotlib library using the data obtained during the experiments. The analysis was performed on the dataset called CIFAR-10. This dataset is widely used in ML research, especially for supervised learning problems. It consists of 10 classes, where each class contains 6000 images. Each image is represented by $32 * 32 * 3$ pixels. So, accumulative we have 60000 images. To solve the first analysis problem, the dataset was split into chunks of different sizes and training and validation methods for each technique presented below are used separately on each chunk to find out the best accuracy that can be achieved on the test data for that chunk or for that size of training data. In Figure 8 you can see the dependence which we described above:

Fig. 8. The best accuracies of kNN and ANN on test data depending on the number of the training examples

To solve the second analysis problem, the accuracies obtained above are used to calculate the absolute difference of accuracies at all points (Figure 9).

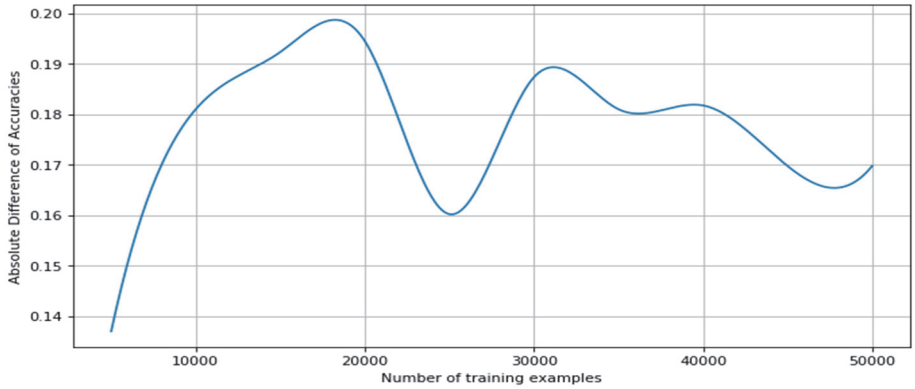


Fig. 9. The absolute differences of accuracies achieved during the train/val/test stages for kNN and ANN

Conclusion. The analysis of completely different methods was done on the CIFAR-10 dataset. From the analysis results, we can see that even on datasets with a small number of training examples, ANN always shows a better test accuracy than kNN. But on the other hand, the accuracies obtained using kNN are more stable in their growth as opposed to ANN, although it uses more computational resources to pass the first part of the analysis than ANN.

REFERENCES

1. **Okfalisa, Ikbal G., Mustakim, Nurul G.I.R.** Comparative analysis of k-nearest neighbor and modified k-nearest neighbor algorithm for data classification // 2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE). - 2017. - P. 294-298.
2. **Lek S., Park Y.S.** Artificial Neural Networks // Reference Module in Earth Systems and Environmental Sciences. - 2008. - P. 237-245.
3. **Leonardo V., Mauro C.** Multilayer Perceptrons // Encyclopedia of Bioinformatics and Computational Biology. - 2019. – Vol. 1. - P. 612-620.
4. **Zhangyang W., Ding L., Thomas S.H.** Deep Learning Through Sparse and Low-Rank Modeling. - Academic Press, 2019. - 296 p.
5. **Byvatov E., Fechner U., Sadowski J., Schneider G.** Comparison of Support Vector Machine and Artificial Neural Network Systems for Drug/Nondrug Classification // Journal of Chemical Information and Computer Sciences. - 2003. – Vol. 43. - P. 1882-1889.
6. **Maida A.S.** Handbook of Statistics. - Elsevier Science Pub Co, 2016. - 314p.
7. **Roman-Rangela E., Marchand-Mailletb S.** Inductive t-SNE via deep learning to visualize multi-label images. - Elsevier Science Pub Co, 2019. - 10 p.

National Polytechnic University of Armenia. The material is received on 21.01.2020.

Կ.Հ. ՆԻԿՈՂՈՍՅԱՆ, Վ.Տ. ԲԵՋԱՆՅԱՆ

Կ- ՄԵՐՁԱԿԱ ՀԱՐԵՎԱՆՆԵՐԻ ԵՎ ԱՐՀԵՍՏԱԿԱՆ ՆԵՅՐՈՆԱՅԻՆ ՑԱՆՑԻ ԲԱԶՄԱՏԵՍԱԿ ԴԱՍԱԿԱՐԳՄԱՆ ՎԵՐԼՈՒԾՈՒԹՅՈՒՆԸ ՄԵԾ ԾԱՎԱԼԱՅԻՆ ՏՎՅԱԼՆԵՐԻ ԴԵՊՔՈՒՄ

Ներկայացվում է մեծ չափայնություն ունեցող տվյալների բազմատեսակ դասակարգման երկու մոտեցում՝ հիմնված K- մերձակա հարևաններ ալգորիթմի և արհեստական նեյրոնային ցանցի մոդելի վրա: Ներկայացված են մաթեմատիկական մոդելների աստիճանական կառուցումը և փորձնական արդյունքների վերլուծությունը:

Առանցքային բառեր. դասակարգում, նեյրոնային ցանցեր, մեծ տվյալներ, մաթեմատիկական մոդելավորում, ալգորիթմներ, բարձր արտադրողականությամբ հաշվարկ:

К.Г. НИКОГОСЯН, В.Т. БЕДЖАНЯН

АНАЛИЗ K-БЛИЖАЙШЕГО СОСЕДА И ИСКУССТВЕННОЙ НЕЙРОННОЙ СЕТИ ДЛЯ МУЛЬТИКЛАССОВОЙ КЛАССИФИКАЦИИ МНОГОМЕРНЫХ ДАННЫХ

Представлены два метода мультиклассовой классификации многомерных данных на основе алгоритма k-ближайших соседей и модели искусственной нейронной сети. Приведено пошаговое построение математических моделей и дан детальный анализ результатов эксперимента.

Ключевые слова: классификация, нейронные сети, большие данные, математическое моделирование, алгоритмы, высокопроизводительные вычисления.