Assignment 1

Tutors: (list all your tutors)

Group members: Jielong ZHONG (430548048), Zhizhi CAI (470259267), Ruiqiang LI (460241999)

Abstract

This article introduces three algorithms to classify images into 10-category fashionable objects from given Fashion MNIST data set. We firstly have a detailed discussion on each algorithm to show how they work for classification task. Then we show how we set up experiments to evaluate the performance of each algorithm. On this part, we know that neural network has the best performance over the other two classifiers base on the given dataset, although training process of neural network is the most time-consuming. Nevertheless, neural network can be trained end-to-end without any feature engineering. After that, pros and cons of each classifier are discussed.

1 Introduction

Machine learning has been widely used in data science across various industries with the help of statistic techniques. Data scientists, researchers and engineers take advantages of statistic methods to make machine learning algorithms learn from data and then use trained model to predict results from unseen data. In this article, we aim to introduce and develop three different algorithms to recognize the fashionable objects (e.g. T-shirt, shoes and bags) in the grey scale images and classify them as corresponding labels. These three algorithms are K-Nearest Neighbors, Support Vector Machines and Neural Network. Besides, we also aim to compare the performance of these three algorithms by evaluating them greatly on the basis of accuracy. The main dataset we use is part of the Fashion MNIST, which are grey scale images of 10 distinct categories relating to fashionable clothes. It is known to us that mobile E-commerce has been a great part of our lifestyle. Developing programs to correctly recognize fashionable objects can be a great tool to help customers quickly access and select the product they want. Therefore, it is important to develop such a program embedded in mobile phone E-Commerce application. Finally, through this article, we can know how each algorithm work and their performance on the given dataset.

2 Methods

2.1 Pre-processing

We tried two preprocess methods for the dataset to run the experiments. They are normalization and principal component analysis. The distribution of the dataset including train set and test set can be represent as: $D = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_n, y_n)\}, x \in \mathbb{R}^d, y \in \mathbb{R}$.

2.1.1 Normalization for Neural Network

Due to the large difference in the magnitude of the image data, the feature of sample data will be affected by magnitude of the data [1]. But before that, we reshape the training and test data into two dimensions with shape (number_of_samples, 784) by multiplying width with height of the

images. To avoid such problem in neural network model, we normalize the sample by standard deviation. At first, we calculated the standard deviation of each sample:

$$\sigma = \sqrt{\frac{\sum(x-\bar{x})^2}{n}}$$

Then we convert the sample data values to the range between 0 and 1 using following calculation:

$$x^* = \frac{x-\bar{x}}{\sigma}$$

2.1.2 Principle Components Analysis

We also tried principal component analysis (PCA) to reduce the dimensions of the data when implementing the K-Nearest Neighbor because the dimensions of the sample is very high. Since the data of the sample is sparse with a high dimension, it will take a long time to calculate their Euclidean Distance. PCA can be divided into five steps. First, we execute the centralization for each sample: $x_i = x_i - \frac{1}{m}\sum_{i=1}^{m} x_i$. Then, we calculate the covariance matrix: $X * X^T$ using matrix of each sample. In the third step, we do eigenvalue decomposition for covariance matrices $X * X^T$ and sort the eigenvalues and eigenvectors according to the eigenvalues in descending orders. After that, we take the eigen vectors corresponding to the first d^* eigenvalues to form the projection matrix $W^* = (w_1, w_2, \dots, w_{d^*})$. Finally, we obtain the result from equation: $X * W^*$.

2.2 Classifiers

2.2.1 Neural Network Classifier

We initialize the weights of neural network with Gaussian Distribution: $\mathcal{N}(0, \sqrt{2/n})$, since such initialization method have a good performance for Rectified Linear Unit activation function [4]. Samples are combined with weights and bias in each layer. Then, Rectified Linear Unit (ReLU) are used as activation function before output to gain the effect of non-linearity. The relation of output and input for each node can be represented as equation: $z_i = \max(0, w_i^T x + b), i = 1,2,3 \dots c$. The Softmax function is used in the final layer to generate classification results. Suppose $z_i, i = 1,2,3 \dots K$ is the result of each node in the last layer. Then, the value of each node will be described as:

$$\sigma_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

The Softmax value represent the proportion of the corresponding category.

We use the cross entropy as the loss function. The loss function of the result can be calculated by equation: $L_i = \sum_{i=1}^{k} y_i * \log \sigma_i$ with y_i referring the ground truth. Then the gradient of each weight can be calculated by using chain rule from equation:

$$\delta_i = \frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial \sigma_i}\frac{\partial \sigma_i}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}$$

Combing with the feedforward equation, Softmax equation and loss function equation, then we can update the parameters weights and bias through training iterations to get optimal parameters.

In the process of training the neural network, the model can obtain excellent performance on the train dataset, but the performance such as accuracy rate on the test dataset is not good. To avoid this type of overfitting problem, we use L2 regularization to increase the generalization ability of the neural network model. Our objective function in neural network is:

$$\underset{w}{arg\min} \sum_{i=1}^{m} y_i * \log \frac{e^{f(w_i^T x+b)}}{\sum_k e^{f(w_k^T x+b)}}$$

Thus, we introduce the L2 regularization to the objective function:

$$\underset{w}{\operatorname{argmin}}\, y_i * \log \frac{e^{z_i}}{\sum_k e^{z_k}} + \lambda \|\omega\|^2$$

The L2 regularization term added to gradient computation and backward propagation can help reduce the generalization error. With the help of regularization term, it can effectively avoid overfitting problems during model training.

2.2.2 Support Vector Machines

The second classifier we tried is support vector machine (SVM). Basically, SVMs are used to find the hyperplane among different classes of samples. A general hyperplane can be described as: $w^T x + b = 0$ where $w(w_1, w_2, \dots, w_n)$ refers to the normal vector and b refers to the bias. Suppose we get a hyperplane that already can divide the sample data, then those sample points on the hyperplane boundary $w^T x + b = 1$ are the support vectors. The margin refers to the sum of the distances between two support vectors at different boundaries to the hyperplane. The margin can be represented as: $\gamma = \frac{2}{\|w\|}$. As a result, finding such hyperplane can be described as mathematic equation:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad , s.t. \, y_i(w^T x_i + b) \geq 1, i = 1,2, \dots m.$$

To solve this problem, the equation uses the Lagrange multiplier method to convert the previous problem into its dual problem:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_i {x_i}^T x_j \quad , s.t. \sum_{i=1}^{m} \alpha_i y_i = 0, \alpha_i \geq 0, i = 1,2, \dots, m.$$

After the calculation, the optimal result α should satisfy the Karush-Kuhn-Tucker (KKT) condition:

$$\begin{cases} \alpha_i \geq 0 \\ y_i f(x_i) - 1 \geq 0 \\ \alpha_i (y_i f(x_i) - 1) = 0 \end{cases}$$

We tried the Sequential Minimal Optimization (SMO) algorithm to speed up the training process of SVM classifier. The SMO algorithm first selects two variables α_1, α_2 and fixes other variables. Then the sub problem of the previous optimization problem can be described as:

$$\arg\min_{\alpha_1,\alpha_2} W(\alpha_1,\alpha_2) = \frac{1}{2}\kappa(x_1,x_1){\alpha_1}^2 + \frac{1}{2}\kappa(x_2,x_2){\alpha_2}^2 + \kappa(x_1,x_2)y_1y_2\alpha_1\alpha_2 - (\alpha_1+\alpha_2)$$

$$+y_1\alpha_1\sum_{i=3}^{m}\alpha_i y_i\kappa(x_i,x_1) + y_2\alpha_2\sum_{i=3}^{m}\alpha_i y_i\kappa(x_i,x_2)$$

$$s.t. \quad \alpha_1y_1 + \alpha_1y_2 = -\sum_{i=3}^{m}\alpha_i y_i = \varsigma \ , \ 0 \le \alpha_i \le C \text{ [8].}$$

The $\kappa(x_i,x_j)$ is the kernel function which is used to is used to map the system of equations into a high-dimensional hyperplane to find a parameter that satisfies the condition. Both ς and C are constant term.

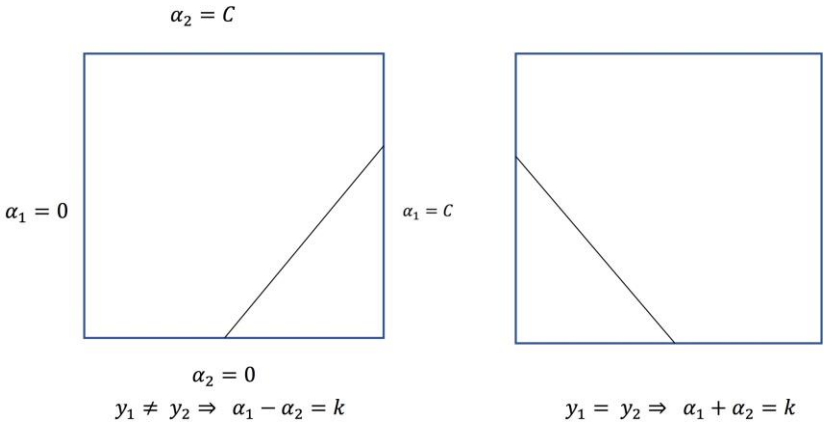

Figure 2. Restriction Description

The restrictions can be described by **Figure 2** and this means that the result of such two variables quadratic programming problem can be derived from single variable optimization problem. Assuming that the initial feasible solution is: ${\alpha_1}^{old}, {\alpha_2}^{old}$ and the optimal solution is: ${\alpha_1}^{new}, {\alpha_2}^{new}$. We first calculate the initial predictive value: $f(x_i) = \sum_{i=1}^{m}\alpha_i y_i\kappa(x_i,x) + b$ and its difference from label: $E_i = f(x_i) - y_i, i = 1,2$. If we take the α_2 as the optimization objective. Then the unclipped result of optimal α_2 can be derived from equation: ${\alpha_2}^{new,unclipped} = {\alpha_2}^{old} + \frac{y_2(E_1-E_2)}{\eta}$ with $\eta = \kappa(x_1,x_1) + \kappa(x_2,x_2) - 2\kappa(x_1,x_2)$. According to figure2.1 and conditions above, the optimal result ${\alpha_2}^{new}$ should satisfy the constraining direction: $L \le {\alpha_2}^{new} \le H$ where:

$$\begin{cases} L = \max(0, {\alpha_2}^{old} - {\alpha_1}^{old}) \quad H = \min(C, C + {\alpha_2}^{old} - {\alpha_1}^{old}) & if \ \ y_1 \neq y_2 \\ L = \max(0, {\alpha_2}^{old} + {\alpha_1}^{old} - C) \quad H = \min(C, {\alpha_2}^{old} + {\alpha_1}^{old}) & if \ \ y_1 = y_2 \end{cases}$$

After getting the ${\alpha_2}^{new}$, the optimal result of α_1 can be calculated by:

$${\alpha_1}^{new} = {\alpha_1}^{old} + y_1y_2({\alpha_2}^{old} - {\alpha_2}^{new})$$

and then we can use the ${\alpha_2}^{new}$ and ${\alpha_1}^{new}$ to predict the result of test sample [7].

2.2.3 K-Nearest Neighbors

Another classifier we implemented was K-Nearest Neighbors (KNN). The KNN model has no training process. Instead, they just keep the train sample in the training sample and calculate the distance between each training sample and the given test sample [2]. The distance can be derived from equation: $distance = \sqrt{\sum_{i=1}^{m}(t_i - x_i)^2}$. The parameters t_i and x_i refer to the features of each testing sample and training respectively. Then we sort the labels of the samples based on the calculated distance between given test data and each train data while at the same time take out the first K labels because we consider them as the most similar sample images for the given test image. Finally, we count the number for each labelled category and output the biggest ones, which are nearest data points to each test data.

3 Experiments and Results

In order to evaluate the performance of different classifiers with regards to the given dataset, the experiments consist of comparing two main types of classification algorithms: traditional machine learning algorithms and deep learning ones. Specifically, we performed following overall experiments: (a) KNN classifier, which is a non-parametric classifier and set up as baseline experiment; (b) Self-designed SVM with sequential minimal optimization algorithm for only single one label (the target is 'label 1'); (c) Use SVM with RBF kernel from Scikit-learn to get full categorical classification results (the target is all 10 categories); (d) We finally implemented a neural network to do classification task. For measurement, we evaluate the performance of classifiers by calculating the number of correctly labelled data over the total number of data used for all designed experiments. The formula used is shown below:

$$\text{Accuracy} = \frac{Number\ of\ correct\ classification}{Total\ number\ of\ test\ examples\ used} * 100\%$$

However, in terms of neural network, in addition to the application of accuracy measurement, we also applied loss/cost function to monitor that the training process worked smoothly. The detailed description of cost function applied is mentioned in the Section 2.2.1.

For KNN algorithm, we just calculate the Euclidean Distance between every single data point in test data set and the training data. Then assigning predicted label to each test sample according to the k-nearest distance. In our experiment, we found that the best option for hyper-parameter k is 6 through testing, which made the classifier achieve an accuracy of 83.95% based on the given 2,000 test images. The result can be seen from Table 1.

Table 1. Performance Comparison based on Accuracy and Running Time

	Training Accuracy	Test Accuracy (2000)	Running Time
KNN (Baseline)		83.95	330s
SVM (Target on only one class)	87.00	86.50	> 8h
SVM (Target on All Classes) <Scikit-learn Version>	92.01	87.15	516s
Neural Network	89.13	87.15	2295s

On the other hand, although the self-designed SVM-SMO algorithm has a relatively high accuracy for only single one class with training accuracy for 87.00% and test accuracy for 86.50%

respectively, we found it needed a huge amount of time to train the classifier. To that end, we chose to test the performance and run an extra experiment on well-designed, effective and optimized classifier via using Scikit-learn (Sklearn) library due to the limited time we had to train and tune our own SVM-SMO classifier. There are many kernel options we can use from Sklearn SVM. Nevertheless, we chose kernel with radial basis function (RBF) to training our classifier instead of using polynomial or linear kernel. Choosing RBF kernel is because it can efftectively deal with non-linearity and more time-effective. As we can see from Table 2 and Table 3 that SVM with RBF kernel has higher performance than linear kernel and the time needed for training is less. With the help of Sklearn SVM, we gain a good result on both the training data (92.01%) and test data (87.15%) which can be seen from Table 1.

Table 2. Performance on Settings of Different C

	RBF Kernel Results							
Parameter C	1	5	10	20	30	40	50	60
Accuracy	0.8740	0.8860	0.8890	0.8910	0.8875	0.8855	0.8855	0.8865
Time	564 s	544 s	511 s	514 s	533 s	534 s	530 s	530 s

Table 3. Performance on Settings of Different C

	Linear Kernel Results		
Parameter C	1	5	10
Accuracy	0.8235	0.8165	0.8165
Trainning Time	498 s	2389 s	4611 s

It is notable that we might need to do some feature engineering for traditional machine learning algorithms. For example, in our experiments we apply PCA in reducing data dimensions to get more useful features for KNN algorithm. However, neural network can take as input training data and train the network end-to-end without doing any feature engineering. But for the neural network, the number of hidden layers and number of neurons in each hidden layer are two main aspects that must be considered while designing the network structure. The number of hidden layers we used was 4 since the training images are all grey scale and not that complex. If we choose more hidden layers with more neurons, the result could be at a risk of overfitting. With regard to neuron selection, we took into account following thumb-of-rules mentioned by [6]:

i. $\text{size of output layer} \leq \text{hidden neurons} \leq \text{size of input layer}$

ii. $\text{hidden neurons} = 2/3 * size\ of\ input\ layer + size\ of\ output\ layer$

iii. $\text{hidden neurons} < 2 * \text{size of input layer}$

We tried various structures and learning rate with slight modifications, the accuracy interval is (86.10, 87.5). The overall experiment setup for neural network with testing best performance is: learning rate=0.05, number of running epochs=3000, regularization term $\gamma = 0.7$, layer_dimensions=[784, 256, 128, 64, 10]. From Table 1, we can see the neural network has competitive performance with accuracy of 87.15% compared to SVM classifier using Scikit-learn and has a higher accuracy score than KNN and self-designed SVM. In addition, the losses over 3000 epochs with two different learning rates reduced smoothly as shown in Figure 2, which means that the network works relatively stable based on the proposed structure. Although neural network

achieved almost the best performance among these three classifier algorithms, it is the most time-consuming algorithm due to its complex neural network structure.

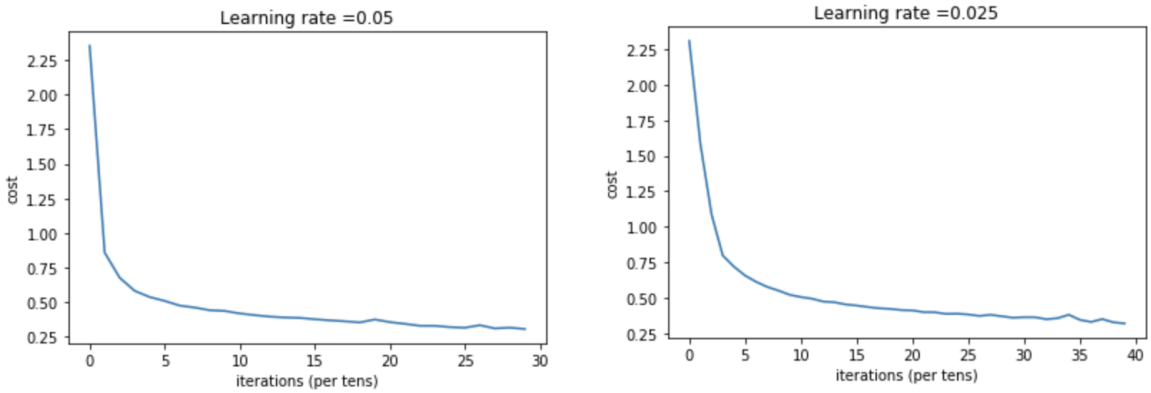

Figure 2. Cross Entropy Losses with Different Learning Rates (0.05 and 0.025 respectively)

It should be noticed that all above experiments are ran on a computer with specifications shown in Table 2. Using higher specifications of computer to run the experiments might gain more advanced efficiency or slight differences on accuracies.

Table 2. Hardware and Software Specifications

	Description
Computer Model	Macbook Pro (13-inch 2017)
Processor	2.3 GHz Intel Core i5
Memory	8 GB 2133 MHz LPDDR3
Graphics	Intel Iris Plus Graphics 640 1536 MB
System	macOS Mojave version 10.14
IDE	Jupyter Notebook
Software Used	Google Chrome

4 Discussion

It should be noticed that KNN running time calculation is different from other two algorithms. Since KNN algorithm is a non-parametric classifier, the running time for KNN is the time for computing the Euclidean Distance between test data and training data. However, SVM and neural network are parametric classifier, they have model training process. The training time depends on the volume of training data and the parameters needed for the model. Besides, we have found that the self-designed SVM algorithm has a big problem of training the classifier efficiently. This could be due to the inefficient coding. Although we have tuned our self-designed SVM-SMO algorithm with different hyper-parameters, it is still hard for us to reduce the training time. However, the running time decreased dramatically while we used Sklearn SVM. Moreover, although neural network achieved a competitive performance, it is the most time-consuming algorithm due to its complex neural network.

5 Conclusion

In brief, end-to-end neural network does offer an accurate classification for the given data set over other two classifier. However, it is time consuming to train a competent model. In addition, we can know that SVM is also good at classification but hard to implement from scratch. Although

KNN is easy to implement and has a reasonable performance, the training time for it will increase tremendously once the amount of data set becomes huge or the data set becomes complicated. So, for those datasets that are small and easy to separate, KNN is a sound option to implement. Otherwise, it is better to use SVM or neural network to train a robust classifier to get higher accuracy results.

References

1. Aksoy, S., & Haralick, R. M. (2001). Feature normalization and likelihood-based similarity measures for image retrieval. Pattern recognition letters, 22(5), 563-582.
2. Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. IEEE transactions on information theory, 13(1), 21-27.
3. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. Han Xiao, Kashif Rasul, Roland Vollgraf. arXiv:1708.07747
4. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE international conference on computer vision, pp. 1026-1034.
5. Jolliffe, I. (2011), Principal component analysis, In International encyclopedia of statistical science, Springer, pp. 1094-1096
6. Heaton, J., (2008), Introduction to Neural Networks for Java, Heaton Research, pp. 158-159
7. Platt, J. C. (1999). 12 fast training of support vector machines using sequential minimal optimization. Advances in kernel methods, 185-208.
8. Platt, J. (1998), Sequential minimal optimization: A fast algorithm for training support vector machines.