Machine and Deep Learning Course Project Report

Image Classification Task on Fashion-MNIST Dataset

Can Abdullah Camuz - 2041519
canabdullah.camuz@studenti.unipd.it

1. Introduction

In this project, Machine Learning approach was used on a dataset of images, which were processed to make accurate predictions. The problem was the classification of cloth images; therefore the purpose was correctly predicting the classes of the images outside of the training set. Since there were 10 classes for the items to label each, it was a multiclass classification problem. The study was important to see and compare how different models (parametric and non-parametric) worked on the same dataset. Different algorithms (models) were employed to achieve this goal, learned in the scope of the course. Neural Network, Decision Tree, Logistic Regression and Random Forest models were run on the dataset, accuracy was measured, the predictions were evaluated and finally some visualization tools were used to understand how the parameters changed for each model.

The best results were obtained by the Neural Network approach (88.1 %) even though all the algorithms had close accuracy values (between 80% and 90%).

1.1. Dataset

The dataset is named fashion MNIST and contains 70000 greyscale images in 10 categories such as Sneaker, Pullover, Coat, Shirt, etc. The images show individual articles of clothing at low resolution (28×28 pixels). Each pixel is assigned to a value between 0 and 255, with higher numbers meaning darker. Therefore, each image consists of 784 pixels and in total 784×70000 = 54.880.000 values were processed.

1.2. Data Preparation

In order to be able to construct a model, the dataset was split into 3 subsets. First, a split was made as 60000 for train set and 10000 for the test set; then 12000 data was extracted from the train set, remaining 48000.

The train set was used as input values to our models to make the classification. Validation and test sets were used to evaluate the performance of the models based on the regarding measures.

Since the data came in 28x28 matrices, they needed to be flattened into a vector in order to reduce the workload on the algorithms and to work efficiently. So, for each image, 784 pixel values were flattened into a vector and matrices of (60000, 784) and (10000, 784) were created in the first step. (60000 for train and validation, 10000 for test). Here, the rows were the images and the columns were the corresponding pixel values.

Because the Machine Learning algorithms start searching for the optimal by using random numbers, this randomness needs to be controlled to understand the model behavior with different parameters. Different random starting points would place the weights nearer to different local minimum in the cost function, meaning that they would lead to different results at the end of training.[1] So, a random seed was fixed to prevent this and train several networks with different hyperparameters from the same random start point.

Since the pixel values were stored as integers in the range 0 to 255, there was a need to scale them down to [0,1]. This process is called Normalization and it aims to facilitate the optimization processes and maximize the probability of obtaining good results.[2] Normalizing a vector is to divide the data by the norm of the vector. Depending on the data, the norm here was 255, the values in the sets were normalized by dividing each by 255.

Finally, validation set was extracted from the training data which yielded 48000 training points and 12000 validation points (80% - 20%).

2. Models And Methods Used For Solution

2.1. Building A Neural Network Model

2.1.1 The General Structure

For the solution of the problem, a Neural Network was built that was able to classify the items in the dataset. The Network consisted of 1 input layer, 2 hidden layers, and 1 output layer. Hidden layers had 128 nodes (neurons) and since there were 10 classes, output layer contained 10 nodes.

Hidden layers have the neurons where the “learning” of the model occurs. They take in a set of weighted inputs and produce an output through an activation function. In this model, Sequential class was used to group a linear stack of

layers and to provide training and inference features on this model. For the hidden layers, activation function ReLU was used, which is a piecewise linear function that outputs the input directly if it is positive, otherwise, outputs zero. ReLU was chosen because of its speed and efficiency over other sigmoid functions.[3] However, in the Output Layer, Softmax function was used, which converts a vector of numbers into a vector of probabilities. This implies that it works well for multi-class classification problems where class membership is required on more than two class labels, such as in the fashion MNIST case.

2.1.2 Model Compilation and Training

After the completion of layer construction, the model was compiled by inserting the following 3 parameters to specify the methods for a proper classification.

As an optimizer, "Adam" was selected for its efficient method that requires less memory compared to classic SGD. Since the parameters of Adam controls the decay rates of the learning rate, learning rate changed during the training, which - called Learning Rate Decay – helped to converge the optimum faster. [4]

Second parameter was the loss function of the model and for this "sparse_categorical_crossentropy" was selected which computes the crossentropy loss between the labels and predictions. This loss function was preferred due to multiple number of classes, and labels provided as integers. So, one-hot encoding was not explicitly performed, which would require using "categorical_crossentropy" function.

Finally, accuracy was chosen to evaluate the performance of the model, which basically calculates the rate of correct predictions over all predictions. This was applied to train, validation and test sets to have an idea of performance. Also, to check for overfitting or underfitting.

The model was then trained by the data in the training set, with a batch size of 1200, for 28 epochs. The epochs consisted of 40 steps due to the batch size. These parameters were decided heuristically, and mostly by trial. After fitting the model, 91.59% training and 89.05% validation accuracy were reached.

Below, the change in the accuracy and the loss can be seen graphically during the trainings (epochs).

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (1200, 128)               100480

 dense_1 (Dense)             (1200, 128)               16512

 dense_2 (Dense)             (1200, 10)                1290

=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
```

Figure 2: Neural Network Model Summary

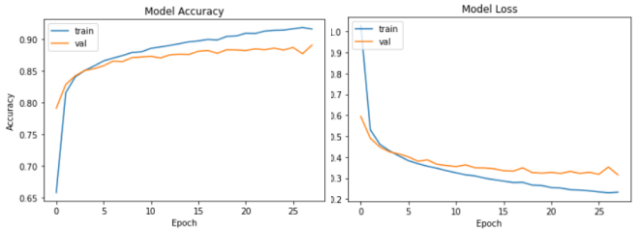

Figure 1: Accuracy & Loss of Datasets

2.1.3 Evaluation and Results

It can be concluded that the model was trained well, considering that accuracy increases as the loss decreases over time. Validation and Training accuracies are close values and this can be interpreted as overfitting to be less likely. After using the `evaluate()` function, which is the last phase of the model to have more realistic idea on the performance, 0.881 of test accuracy and 0.34 of loss were achieved. Since accuracy and loss are not the only outputs that should be considered, a confusion matrix was built, which was a robust and visual tool to evaluate the performance of the multiple-class classification. Dimension of the matrix was 10×10 due to the number of classes. In the matrix, distribution of correctly and incorrectly predicted items can be seen. The numbers in the cells represent the number of predictions made for the related class.

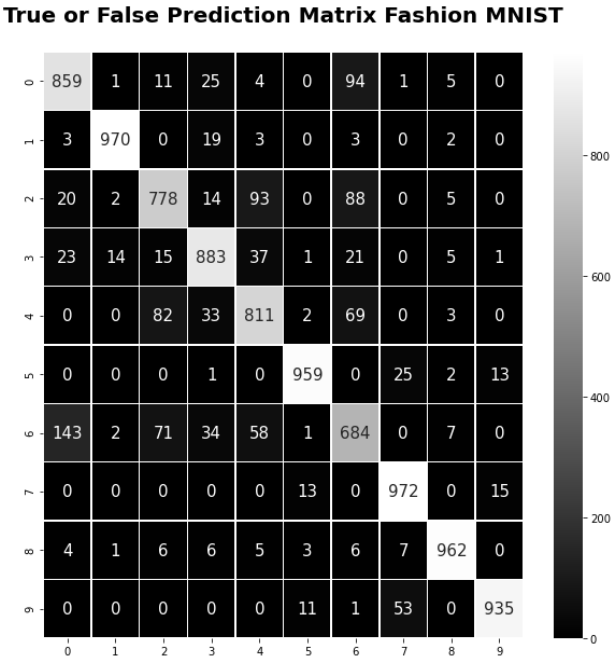

Figure 3: Confusion Matrix of The Model

The matrix is grey-scaled as the correctly predicted values have lighter background. It is obvious that the diagonal represents the True Positive values.

Finally, to see other important parametrics, a score table was built which consisted of Precision, Recall, F1 and Support values of each class. According to this, Precision, Recall and F1 scores share an average of 0.88, and Support value is 1000 for each class. This indicates that the predictions were balanced and the overall performance was reasonable.

2.2. Building A Decision Tree Model

2.2.1 General Structure and Model Compilation

To follow a different approach and to compare with the Neural Network, a Decision Tree Model was built by using `DecisionTreeClassifier()` from the Scikit-learn library. Depth of the Tree was decided to be 7 in order to prevent taking too much time considering the complexity of the problem. By default, this parameter was None, and the Tree would have expanded until all leaves contain less than `min_samples_split` samples.[5] On the contrary, minimum leaf number was decided to be increased to 5, which was 1 by default, to smooth the model better.

Since the data was prepared in the beginning, the model was directly run and after the training, 0.779 training accuracy and 0.766 validation accuracy were obtained. Then a prediction was made on the test values to compare with the actual values on the test dataset. After the comparison 0.759 test accuracy was obtained. Obviously, the depth needed to be increased for a better convergence.

2.2.2 Grid Search Approach

Grid Search passes all combinations of hyperparameters one by one into the model and check the result. Finally it gives the set of hyperparameters for the best result after passing in the model.[6] For maximum tree depth [10,11,12] and for classification parameter (known as 'criterion') ['gini', 'entropy'] values were tried. After applying this procedure, the best parameters were achieved as: `{'criterion': 'entropy', 'max_depth': 12}`.
Next, the model was built with the optimal parameters this time and accuracy scores were 0.883, 0.814 and 0.805 for training, validation and test sets respectively.

2.3. Building A Logistic Regression Model

A Logistic Regression Model was created with 'sag' solver, 0.1 tolerance and 100 of maximum iterations. 'Sag' solver was preferred because of its speed for large datasets. [7] Tolerance was increased 1000 times since the default one took excessive running time. Accuracy scores were 0.868, 0.851 and 0.842 for training, validation and test sets respectively. Precision, Recall and F1 scores converged 0.84 on average.

Once again, a Grid Search was conducted to look for better parameters. For regularization weakness (C) [0.1, 0.5] and for intercept (known as bias) [True, False] values were tried. Grid Search yielded the parameters as: `{'C': 0.1, 'fit_intercept': True}`. After fitting the model with the optimal parameters, accuracy scores and parametrics were barely changed. Hence, it can be concluded that the original parameter choice was satisfactory.

2.4. Building A Random Forest Model

Finally, a Random Forest Model was created with 25 trees and 12 of maximum tree depth to compare with the other algorithms. Without any special arrangement, 0.921 0.861 and 0.851 accuracy values were obtained for training, validation and test sets respectively. Precision, Recall and F1 scores converged 0.87 on average.

3. Conclusion

With 0.88 of accuracy and evaluation parametrics, Neural Network was the best model among the ones implemented. This was reasonable considering the robustness of Artificial Neural Networks in learning non-linear and complex relationships. Simple Random Forest Model followed that with 0.85. Logistic Regression and Decision Tree models could not succeed as much in spite of the Grid Search application. The performance criteria can be seen on the table below. Since Precision and Recall values were included in the F1 Score, they were not represented.

Model	Test Accuracy	F1 Score
`Neural Network`	0.881	0.88
`Decision Tree`	0.805	0.80
`Logistic Regression`	0.842	0.84
`Random Forest`	0.851	0.87

Table 1: Experimental Results of Different Models

References

[1] https://machinelearningmastery.com/why-initialize-a-neural-network-with-random-weights/
[2] https://www.baeldung.com/cs/normalizing-inputs-artificial-neural-network
[3] https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
[4] https://keras.io/api/optimizers/
[5] https://scikitlearn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[6] https://www.projectpro.io/recipes/optimize-hyper-parameters-of-decisiontree-model-using-grid-search-in-python
[7] https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html