IST-718: Lab #3

fashion mnist classification

Jie Wang

2023

**Introduction**

Image vision is a very significant technology in our life. It allows visual information to be converted into data that can be processed and evaluated by patterns. MNIST ("Modified National Institute of Standards and Technology") is a classic dataset of handwritten images that has been severed as basis of computer visualization and classification algorithms. More recently, Zalando research published a new dataset, with 10 different fashion products -- fashion MNIST.

The fashion MNIST dataset was meant to be replacement for the original MNIST which turned out to be too simple for machine learning study. The fashion MNIST promises to be more challenging, so that machine learning algorithms have to learn more advanced features to accurately classify the images.

Each image in the dataset features has a resolution of 28 x 28 pixels. The dataset includes 60,000 images for training and 10,000 for testing. This study aims to classify fashion images with multiple methods.

**Data preparation**

The dataset was loaded straight from the Keras into a Training set (60,000 images) and Testing set (10,000 images). The images are 28X28 arrays with pixel values range from 0 to 255, and the label are an array of integers 0 to 9. No additional data were used for this study. Given that the date is clean, no additional clean procedure in this step.

t10k-images-idx3-ubyte.gz

t10k-images-idx1-ubyte.gz

train-images-idx3-ubyte.gz

train-labels-idx3-ubyte.gz

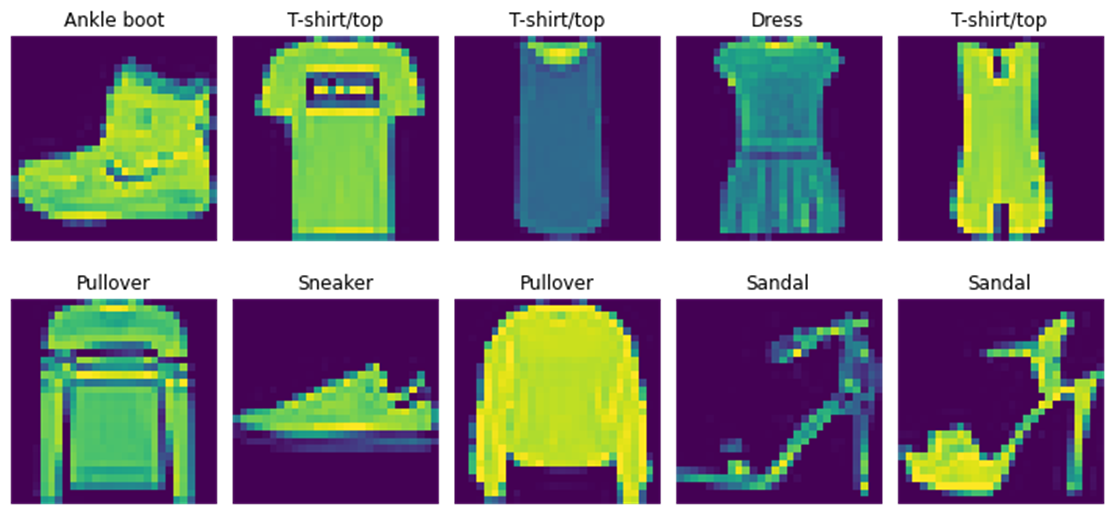
**Methods**

There are three approaches were implemented for classifying the fashion images. (1) Neural Networks (NN, with 3- layers, 6-layers, and 12- layers); (2) Naïve Bayes (NB), and (3) Linear Classification.

**Results**

**A sample of fashion MNIST image from train data set**

A brief exploratory data shows the 10 fashion MNIST image from the train data set. The code used to generate these images were shown in Appendix A.



**Figure 1**. Visual of first 10 fashion MNIST images in Train data set.

**Neural Networks with 3 layers, 6 layers, and 12 layers**

NN with 3 layers (**Figure 2-4**) was performed first. codes were shown inAppendix B

Model: "sequential" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Layer (type) Output Shape Param # ================================================================= flatten (Flatten) (None, 784) 0 dense (Dense) (None, 128) 100480 dense\_1 (Dense) (None, 10) 1290 ================================================================= Total params: 101,770 Trainable params: 101,770 Non-trainable params: 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Figure 2**. 3-layers Neural Network Model summary

Epoch 1/5 1500/1500 [==============================] - 4s 2ms/step - loss: 0.5173 - accuracy: 0.8195 - val\_loss: 0.4217 - val\_accuracy: 0.8468

Epoch 2/5 1500/1500 [==============================] - 3s 2ms/step - loss: 0.3890 - accuracy: 0.8597 - val\_loss: 0.3823 - val\_accuracy: 0.8622

Epoch 3/5 1500/1500 [==============================] - 3s 2ms/step - loss: 0.3481 - accuracy: 0.8737 - val\_loss: 0.3574 - val\_accuracy: 0.8665

Epoch 4/5 1500/1500 [==============================] - 3s 2ms/step - loss: 0.3203 - accuracy: 0.8830 - val\_loss: 0.3338 - val\_accuracy: 0.8788

Epoch 5/5 1500/1500 [==============================] - 3s 2ms/step - loss: 0.3043 - accuracy: 0.8874 - val\_loss: 0.3408 - val\_accuracy: 0.8782

**Figure 3**: 3-layers Neural Network training result

In this case, the model’s loss (the error between its predictions and the true label) decreases and the accuracy increases as the training progresses over the epochs. After five epochs, the value of the loss function for the training set and validation set is 0.3043 and 0.3408, respectively. The training accuracy is 0.8874 and the validation accuracy is 0.8782. This suggests that the model is performing well on the training set and generalizing well to new, unseen data.

313/313 [==============================] - 1s 1ms/step –

loss: 0.3688 - accuracy: 0.8682

Model - 3 layers - test loss: 36.884307861328125

Model - 3 layers - test accuracy: 86.82000041007996

**Figure 4**: 3-layers Neural Network Test result

The test dataset consists of 313 samples, and it took approximately 1 millisecond to evaluate each samples; the average value of the loss function on the test set is 0.3688. The accuracy of the model on the test dataset is 0.868.

15 data points was visualized, with the labelled images, and the probability graph next to them. Red bar means the prediction did not match the true label; otherwise, it is match. It looks like NN with 3-layers model got 1 image wrong out of 15 images (**Figure 5**).

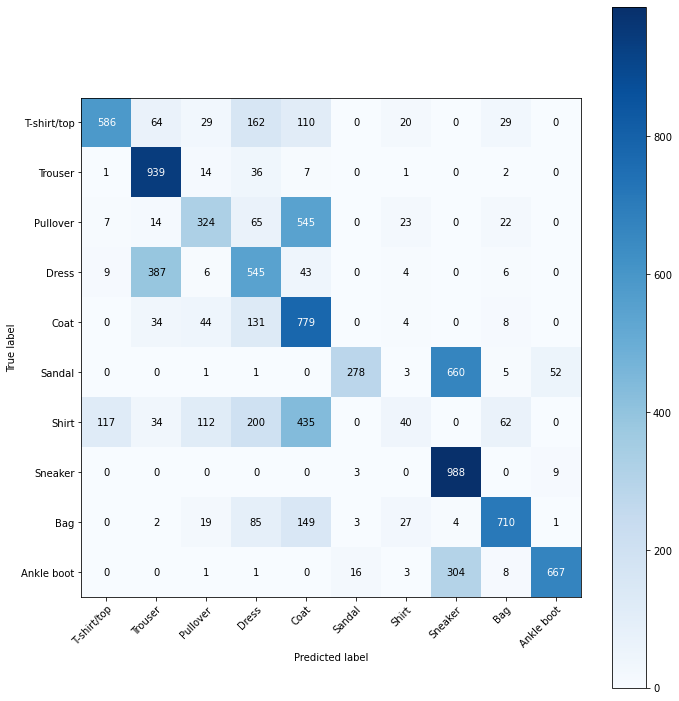
A picture containing window

Description automatically generated

**Figure 5**: Visualizing the predictions.

In addition, NN model with 6 layers and 12 layers were performed as well (code were shown in Appendix C-D). For the NN with 6 layers, the average value of the loss function on the test set is 0.3622; the accuracy of the model on the test dataset is 0.865. For the NN with 12 layers, the average value of the loss function on the test set is 0.4148; the accuracy of the model on the test dataset is 0.856.

**Implement Naïve bayes classifier on fashion MNIST data**

Next, Naïve bayes classifier was applied to classify the fashion MNIST (**Figure 6**), the prediction accuracy is 0.5856. The code for this part were shown Appendix E.

Text

Description automatically generated

**Figure 6**: Naïve bayes prediction on fashion MNIST data

**Implement Linear classification classifier on fashion MNIST data**

Text

Description automatically generatedFinally, the linear classification was used for fashion MNIST data classification. As shown in **Figure 7**, the accuracy is 0.8231, which is much better than NB model. The code for this part were shown Appendix F.

**Figure 7**: Linear classification on fashion Linear classification

**The summary of accuracy of Neural networks, Naïve Bayes, and linear classification model in prediction of Fashion MNIST data**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **NN-3 layer** | | | **NB** | **Linear** |
| **-3 layer** | **-6 layer** | **-12 layer** |
| Accuracy | 0.878(train set) | 0.864(train set) | 0.856(train set) | 0.586 | 0.823 |
|  | 0.868(test set) | 0.865 (test set) | 0.856(test set) |

**Table 1:** The accuracy of each model

**The trade-off of Neural networks, Naïve Bayes, and linear classification model in prediction of Fashion MNIST data**

Neural network, Naïve bayes, and linear classification models each have their own strengths and weakness, which can affect their performance on Fashion MNIST data. Here are some of the trade-offs to consider: 1) Neural networks models, especially using deep learning architectures, are highly flexible and can learn complex non-linear relationships in the data. This can make them very effective at achieving high accuracy on Fashion MNIST. However, neural networks can be computationally expensive to train and require large amounts of data to avoid overfitting. 2) Naïve bayes models are simple and fast to train, making them are a good choice when working with limited computational resources. They assume the features are conditionally independent given the class label, which can be a strong assumption in same case. In this case, NB did not perform well and could not accurately classify sneaker and sandal. 3) Linear classification models are also simple and fast to train, and can perform well on Fashion MNIST data. They make the assumption that the decision boundary between class is linear, which may not true in some cases. However, linear models can be effective when the features are well separated and there is little noise in the data.

**The compute performance of Neural networks, Naïve Bayes, and linear classification model in prediction of Fashion MNIST data**

The compute performance is dependent on many factors. 1) Neural networks: training deep neural networks with more layers and parameter can be computationally expensive. In this study, TensorFlow implementation /library was used, and which can speed up training times. 2) Naïve Bayes: Even Fashion MNIST data is large, Naïve bayes model are very fast to train and perform inference. Since they rely on simple probability distributions, the computations involved are relatively straightforward and require little memory. In this case, the training time is around half minutes. 3) Linear models: Linear models are also generally fast to train and perform inference, especially when used with efficient optimization algorithms. The computational complexity of linear models is linear in the number of features, making them scalable to high – dimensional datasets, in this case, Fashion MNIST. In this case. Linear model in training Fashion MNIST took a minute and achieve a high accuracy prediction.

**Conclusions**

This study has demonstrated that a three- layers of neural network can outperform the traditional machine learning approaches, such as Naïve Bayes and linear classification. However, as indicated by the results, further investigation is needed to optimize the parameters of all three approaches. It is recommended that future studies explore the impact of parameter tuning on the performance of these models, in order to identify the most effective setting for the given task.

Overall this study provides valuable insights into the comparative performance of different machine learning approaches, and highlights the potential of neural networks in particular for achieving high accuracy in complex classification tasks.

**Appendix A**

import tensorflow as tf

from tensorflow import keras

from keras.models import Sequential

from keras.layers import Dense

import numpy as np

import matplotlib.pyplot as plt

print (tf.\_\_version\_\_)

# loading the fashion MNIST data

fashion\_mnist = keras.datasets.fashion\_mnist

(train\_images, train\_labels), (test\_images, test\_labels) = fashion\_mnist.load\_data()

# returns 4 numpy arrays: 2 training sets and 2 test sets

# images: 28x28 arrays, pixel values: 0 to 255

# labels: array of integers: 0 to 9 => class of clothings

# Training set: 60,000 images, Testing set: 10,000 images

# class names are not included, need to create them to plot the images

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# exploring and visualiztion the data

print("train\_images:", train\_images.shape)

print("test\_images:", test\_images.shape)

# Visualize the first 10 images from the training dataset

fig, ax = plt.subplots(2, 5, figsize=(10, 5), gridspec\_kw={'height\_ratios': [2, 2]}) # to create a 2X5 grid of subplot

# loop through the first 10 images in the training dataset

for i in range(10):

row, col = i // 5, i % 5

ax[row, col].imshow(train\_images[i])

ax[row, col].set\_title(class\_names[train\_labels[i]])

ax[row, col].axis('off')

plt.tight\_layout()

plt.show()

**Appendix B**

# Normalizing the data

# scale the values to a range of 0 to 1 of both data sets.

train\_images = train\_images / 255.0

test\_images = test\_images / 255.0

###

# Training the first NN model

# step1: build the architecuture

# step2: Compile the model

# Step3: Train the model

# Step4: Evaluate the model

# Step 1- build the architecture

# model a simple 3 layer neural network using the Keras API of TensorFlow.

model\_3 = keras.Sequential([

keras.layers.Flatten(input\_shape = (28,28)),

# this layer is used to flatten the input image of shape (28,28) into a 1D array of size '784'

# this layer does not have any trainable parameters

keras.layers.Dense(128, activation = tf.nn.relu),

# this layer is a fully connected layer with 128 units and the Relu activation function

# this layer has '784\*128 + 128 = 100490' trainable parameters, where 784 is the input size and 128 is the output size

keras.layers.Dense(10, activation = tf.nn.softmax)

# this layer is another fully connected layer with 10 units and softmax activation function

# the softmax function is used to convert the output of network to a probability distribution over the 10 class.

# This layer has 128\*10+10=1290 trainable parameters, where 128 is the input size, and 10 is the output size.

])

model\_3.summary()

# step2 - compile the model

# compiling the model configures its learning process, including the optimizer, loss function, and metrics

# once the model is compiled, we can train it on the MNIST dataset using the model\_3.fit() method

model\_3.compile(optimizer = 'adam', # an adaptive learning rate optimization algorithm

loss = 'sparse\_categorical\_crossentropy', # common loss function used for multi-class classification problems

metrics = ['accuracy']) # trach the accuracy of the model during training and evaluation

# step3- train the model, by fitting it to the training data

# 5 epochs, and split the train set into 80/20 for validation

#epochs=5: the number of times the entire training dataset will be processed during training

# Setting epochs = 5 means that the model will perform five complete iterations over the entire training dataset, updating its parameters after each iteration.

# The number of epochs to use during training is a hyperparameter that can be tuned to optimize the performance of the model on the task at hand.

model\_3.fit(train\_images, train\_labels, epochs = 5, validation\_split =0.2)

[{"metadata":{"trusted":true},"cell\_type":"code","source":"#Step 4 - Evaluate the model\ntest\_loss, test\_acc = model\_3.evaluate(test\_images, test\_labels)\nprint(\"Model - 3 layers - test loss:\", test\_loss \* 100)\nprint(\"Model - 3 layers - test accuracy:\", test\_acc \* 100)","execution\_count":23,"outputs":[{"output\_type":"stream","text":"313/313 [==============================] - 1s 1ms/step - loss: 0.3688 - accuracy: 0.8682\nModel - 3 layers - test loss: 36.884307861328125\nModel - 3 layers - test accuracy: 86.82000041007996\n","name":"stdout"}]}]

# plot image in a grid

def plot\_image(i, predictions\_array, true\_label, img):

predictions\_array, true\_label, img = predictions\_array[i], true\_label[i], img[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted\_label = np.argmax(predictions\_array)

if predicted\_label == true\_label:

color = 'blue'

else:

color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(class\_names[predicted\_label],

100\*np.max(predictions\_array),

class\_names[true\_label]),

color=color)

# plot the value array

def plot\_value\_array(i, predictions\_array, true\_label):

predictions\_array, true\_label = predictions\_array[i], true\_label[i]

plt.grid(False)

plt.xticks([])

plt.yticks([])

thisplot= plt.bar(range(10), predictions\_array, color="#777777")

plt.ylim([0,1])

predicted\_label = np.argmax(predictions\_array)

thisplot[predicted\_label].set\_color('red')

thisplot[true\_label].set\_color('blue')

# Plot the first 15 test images, their predicted label, and the true label

# Color correct predictions in blue, incorrect predictions in red

num\_rows = 5

num\_cols = 3

num\_images = num\_rows\*num\_cols

plt.figure(figsize=(2\*2\*num\_cols, 2\*num\_rows))

# plt.title("Predictions of the first 15 images, with NN-3")

for i in range(num\_images):

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+1)

plot\_image(i, predictions, test\_labels, test\_images)

plt.subplot(num\_rows, 2\*num\_cols, 2\*i+2)

plot\_value\_array(i, predictions, test\_labels)

**Appendix C**

# Model a simple 6-layer neural network

model\_6 = keras.Sequential([

keras.layers.Flatten(input\_shape=(28,28)),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(10, activation=tf.nn.softmax)

])

#model\_6.summary()

model\_6.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

#Train the NN-6 with 5 epochs

model\_6.fit(train\_images, train\_labels, epochs=5, validation\_split=0.2)

#Evaluate the model with test datasets

test\_loss, test\_acc = model\_6.evaluate(test\_images, test\_labels)

print("Model - 6 layers - test loss:", test\_loss \* 100)

print("Model - 6 layers - test accuracy:", test\_acc \* 100)

**Appendix D**

# Model a simple 12-layer neural network

model\_12 = keras.Sequential([

keras.layers.Flatten(input\_shape=(28,28)),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(128, activation=tf.nn.relu),

keras.layers.Dense(10, activation=tf.nn.softmax)

])

#model\_12.summary()

model\_12.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

#Train the NN-12 with 5 epochs

model\_12.fit(train\_images, train\_labels, epochs=5, validation\_split=0.2)

#Evaluate the model

test\_loss, test\_acc = model\_12.evaluate(test\_images, test\_labels)

print("Model - 12 layers - test loss:", test\_loss \* 100)

print("Model - 12 layers - test accuracy:", test\_acc \* 100)

**Appendix E**

## Nb model

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_openml

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Define class names

class\_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',

'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

# Load Fashion MNIST dataset

X, y = fetch\_openml('Fashion-MNIST', version=1, return\_X\_y=True)

# Convert pixel values to integers between 0 and 255

X = X.astype('int')

# Split dataset into training and test sets

train\_size = 60000

X\_train, X\_test = X[:train\_size], X[train\_size:]

y\_train, y\_test = y[:train\_size], y[train\_size:]

# Train the Naive Bayes model

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

# Predict labels for test set

y\_pred = gnb.predict(X\_test)

# Calculate accuracy and confusion matrix

accuracy = accuracy\_score(y\_test, y\_pred)

cm = confusion\_matrix(y\_test, y\_pred, labels=np.unique(y\_test))

# Plot confusion matrix with class names

fig, ax = plt.subplots(figsize=(10,10))

im = ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)

ax.figure.colorbar(im, ax=ax)

ax.set(xticks=np.arange(cm.shape[1]),

yticks=np.arange(cm.shape[0]),

xticklabels=class\_names,

yticklabels=class\_names,

xlabel='Predicted label',

ylabel='True label')

plt.setp(ax.get\_xticklabels(), rotation=45, ha="right",

rotation\_mode="anchor")

# fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i in range(cm.shape[0]):

for j in range(cm.shape[1]):

ax.text(j, i, format(cm[i, j], 'd'),

ha="center", va="center",

color="white" if cm[i, j] > thresh else "black")

fig.tight\_layout()

plt.show()

# Print accuracy

print("Accuracy:", accuracy)

**Appendix F**

# Try linear classification models

from sklearn.linear\_model import SGDClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix

# Train Linear Classification classifier

sgd = SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, max\_iter=1000, tol=1e-3, random\_state=42)

sgd.fit(x\_train, y\_train)

# Predict labels for test set

y\_pred = sgd.predict(x\_test)

# Calculate loss, accuracy, and confusion matrix

loss = np.mean(y\_pred != y\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

cm = confusion\_matrix(y\_test, y\_pred)

# Print results

print("Loss:", loss)

print("Accuracy:", accuracy)

print("Confusion matrix:\n", cm)