Final Project: Fashion Mnist Classification

Kaicheng Luo
University of California, Berkeley
Berkeley, CA, 94720
kevinlkc@berkeley.edu

Priscilla Hu
University of California, Berkeley
Berkeley, CA, 94720
priscilla_hu@berkeley.edu

December 14, 2019

1. Introduction

Fashion_Mnist is a canonical dataset for Machine Learning. The original dataset contains $n = 60,000$ grayscale images of clothing, of various kinds. We're using a subset of $n = 18,000$ different 28×28 images, each with a label of either shoes, shirt, or pants (6000 of each). The ultimate goal is to implement high-performance classification algorithms.
The writeup will proceed as follows. In the first section that follows, we'll discuss three different kinds of dimension reduction techniques: Principle Component Analysis, Simple Merging, and Convolutional Neural Network. We'll provide substantial visualization and explore the rationale of the number dimensions we shall keep in our model training.
In the next section, a number of binary classification algorithms will be implemented, including Logistic Regression, Linear Discriminant Analysis (LDA), Classification Tree, k-Nearest Neighbour, Neural Network and Convolutional Neural Network. All the results will be shown in a table for a clear illustration of the accuracy. For those with tuning parameters, we'll use 5-fold validation to pick the best model. Meanwhile, a part of the data will be spared as the validation set. After the best model with the least error rate is chosen, we'll report the final accuracy.
We'll then present the multi-level classification scenarios. Once again, different classification methods and the best will be chosen according to their respective out-of sample error. We'll then bridge the gap between binary and multi-level classifications, and account for the difference in the performance of different models.

2. Dimension Reduction

The original dataset consists of 18000 observations and 784 features. Training models using the untransformed dataset requires a considerable amount of computational power. Though the situation isn't so compelling, thanks to certain optimizations is R, we're still going to discuss some dimension reduction techniques so that we might be more comfortable dealing with higher dimensional data.

2.1 Principal Component Analysis

The first technique that came into our mind is Principal Component Analysis (PCA). In short, that justifies a linear orthogonal projection such that retains that maximum inertia. By sorting all the PCs by ascending eigen values, we can reduce the dimension of the original data without fearing to lose too much information. Note that in PCA, the first k principal components we obtain can be heuristically expressed as $Z = XV_k$ where X is the original feature matrix and V_k is a $p \times p$ matrix with rank k, in which there are $(p-k)$ columns that are zero. If we right multiply the PC matrix by V^{-1}. Then we can recover the "simplified" feature matrix. Plotting them in a graph offers us a straightforward comparison between our original picture and those "reduced" ones. Note that although the graph is 28×28 each. The matrix is actually of rank k, $k = 3, 5, 10$, etc.

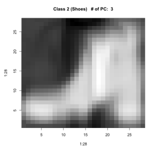

Figure 1: 3 PC

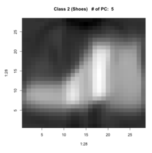

Figure 2: 5 PC

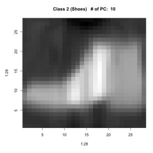

Figure 3: 10 PC

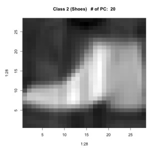

Figure 4: 20 PC

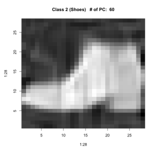

Figure 5: 60 PC

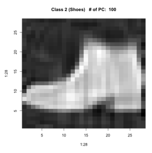

Figure 6: 100 PC

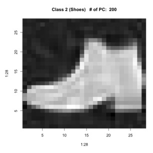

Figure 7: 200 PC

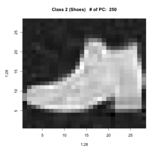

Figure 8: 250 PC

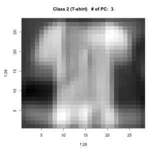

Figure 9: 3 PC

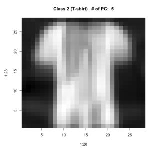

Figure 10: 5 PC

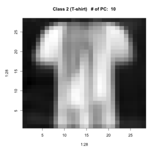

Figure 11: 10 PC

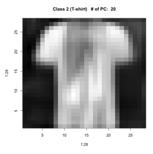

Figure 12: 20 PC

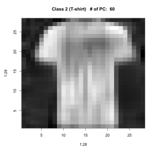

Figure 13: 60 PC

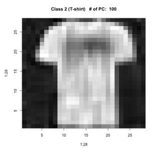

Figure 14: 100 PC

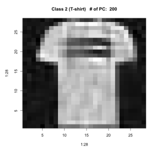

Figure 15: 200 PC

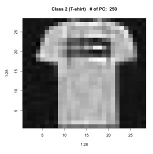

Figure 16: 250 PC

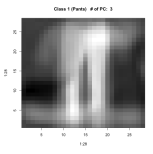

Figure 17: 3 PC

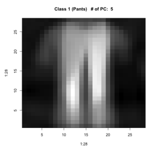

Figure 18: 5 PC

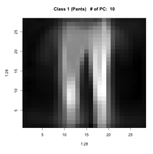

Figure 19: 10 PC

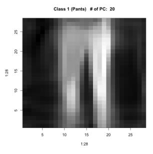

Figure 20: 20 PC

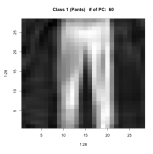

Figure 21: 60 PC

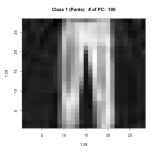

Figure 22: 100 PC

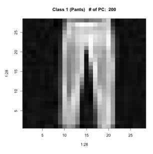

Figure 23: 200 PC

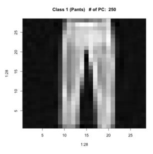

Figure 24: 250 PC

2.2 Local Average

The second method that we used is quite intuitive. Due to the fact that the original data is observed in a 2 dimensional space. We can easy merge the nearest neighbors together to obtain a smaller picture. In a simplified case. Define the multiplier matrix as

$$V = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \dots & 0 & 0 & 0 & 0 \\ \vdots & & & & & & & & \dots & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 1 & 1 & 1 \end{bmatrix} \tag{1}$$

Define the original dataset of features as X, then VXV^T offers us such a projection that we average everything in the neighbourhood and keeps their relative position. A simple illustration is as follows.

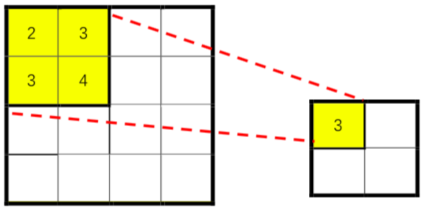

Figure 25: A simple illustration of 4×4 case

We can hence represent our data in a 7×7 matrix

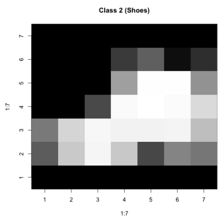

Figure 26: Shoes

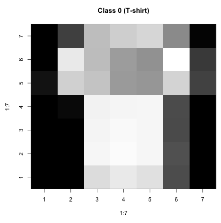

Figure 27: T-shirt

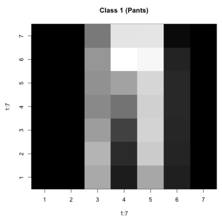

Figure 28: Pants

2.3 How many dimensions shall we keep?

It's reassuring for us to recognize that as long as enough PCs are retained. We could still recognize the shape of a T-shirt, or pants. This offers us some confidence that our final outcome shall be somehow stable, and on top of that, considerable computational power is saved. We chose 49 to be the threshold to train our model, both in order to be consistent between both of our simplification approaches, and to retain clear boundaries that are still distinguishable for our eyes. In reallife machine learning contexts, an ideal model shall be stable with respect to the choice of dimensions. If a slight variation of the k we pick will lead to a tremendous difference in classification outcomes, then we shall certainly cast doubt on the credibility of our model.

2.4 Code

```r
# Idea 1: PCA
pca <- prcomp(X, scale. = TRUE)
V <- pca$rotation[,1:49]
temp <- as.matrix(X) %*% V
rerotation <- solve(pca$rotation)
temp2 <- matrix(c(as.vector(temp), rep(0, 13230000)), nrow = 18000, ncol = 784, byrow = F)
        %*% rerotation
```

```r
# idea 2: Intuitively, merge the neighbouring pixels together.
temp = X
for (i in colnames(X)){
  temp[,i] <- ifelse(X[i]>0, 1, 0)
}

# Create a matrix
zerovector <- rep(0,28)
onevector <- rep(1,4)
multiplier <- matrix(c(onevector, zerovector, onevector, zerovector, onevector, zerovector,
  onevector, zerovector, onevector, zerovector, onevector, zerovector, onevector), nrow= 7, byrow=T)
```

```r
dimred <- matrix(0, nrow = 18000, ncol = 49)
for (i in 1:18000)
{
  x <- matrix(as.numeric(X[i,]), ncol=28, nrow=28, byrow = TRUE)
  x <- apply(x, 2, rev)
  x <- multiplier %*% x %*% t(multiplier) / 16
  x <- as.vector(x)
  dimred[i,] <- x
}
```

3. Binary Classification

3.1 Preparing the data

For binary classification tasks, we split the data into training set, validation set, and test set (with the exception of random forest algorithm which itself can generate an unbiased estimation of the out-of-sample error), each of which is composed of 50%, 33% and 17% of the original observations, respectively. We're using the "reduced" feature matrix with rank 49. Setting the random seed ensures reproducible results.

3.2 Classification Algorithms

In the binary case, we're implementing Logistic regression, Linear discriminant Analysis, Classification Trees, and Random Forest algorithms. Generally, those models do not impose strong assumptions about the prior distribution (with the exception of LDA which assumes Gaussian distribution of features.) In this specific task, we're also not so interested in the inference part of the analysis. Namely, the significance of a certain pixel in the explanatory power is meaningless compared to the overall accuracy. In the subsections, we'll provide some scratch on the fundamental formula for the model.

Logistic Regression

$$Pr(Y|X) = \frac{e^{X\beta}}{1+e^{X\beta}} \tag{2}$$

Linear Discriminant Analysis

$$\delta_k(x) = x^T \mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T \mathbf{\Sigma}^{-1}\mu_k + \log \pi_k \tag{3}$$

Decision Tree

$$\theta = \arg\max \Delta I = \arg\max \left\{ -p\log_2 p + \sum_\theta p_\theta \log_2 p_\theta \right\} \tag{4}$$

3.3 Classification Results

Table 1: Error Rate in Validation Set

	T-shirt vs Pants	T-shirt vs Shoes	Pants vs Shoes
Logistic	0.02175	0.00175	0.00175
LDA	0.03	0.00125	0.001
Tree	0.032	0.00875	0.003
Random Forest	0.012	0.00075	0.00025

3.4 Code

```r
#generate the low dimension dataset
y <- y0
y <- as.factor(y)
data <- data.frame(y, dimred)
#levels(data$y)
```

```r
set.seed(1)

binaryValMCE<-function(data,y1,y2){
  MCE=data.frame(row.names=c("Logistic","LDA","Tree","Random Forest"))

  data=data[(data$y==y1)|(data$y==y2),]
  data$y<-factor(data$y)

  trainSet <- data[1:6000,]
  valSet <- data[6001:10000,]
  testSet <- data[10001:12000,]

  #Logistic Regression
  logreg_default <- glm(y~., family = binomial, data = trainSet)
  pred_log <- ifelse(predict(logreg_default, valSet[,-1], type = "response")>0.5, y1, y2) #2,0???
  MCE[1,1]=1-sum(diag(table(pred_log,valSet[,1])))/sum(table(pred_log,valSet[,1]))
  #CrossTable(pred, valSet[,1], chisq = FALSE)

  #LDA
  lda_default <- lda(y~.,data=trainSet)
  pred_lda<-predict(lda_default,valSet[,-1],type="response")$class
  #CrossTable(pred_lda,valSet[,1],chisq = FALSE)
  MCE[2,1]=1-sum(diag(table(pred_lda,valSet[,1])))/sum(table(pred_lda,valSet[,1])) #(1993+2002)/4000

  #tree
  tree_default<-tree(y~.,trainSet)
  #summary(tree_default)
  #plot the tree
  #plot(tree_default)
  #text(tree_default,pretty=0)
  #summary(tree_default)
  pred_tree<-predict(tree_default,valSet[,-1],type="class")
  #CrossTable(pred_tree,valSet[,1],chisq = FALSE)
  MCE[3,1]=1-sum(diag(table(pred_tree,valSet[,1])))/sum(table(pred_tree,valSet[,1]))

  #Random Forest

  rand_forest<-randomForest(y~.,trainSet)
  rand_forest
  #importance(rand_forest)

  pred_rndForest<-predict(rand_forest,valSet[,-1])
  #CrossTable(pred_rndForest,valSet[,1],chisq = FALSE)
  MCE[4,1]=1-sum(diag(table(pred_rndForest,valSet[,1])))/sum(table(pred_rndForest,valSet[,1]))
  #plot(pred_rndForest,testSet[,1])
  return(MCE)
}

MissClasiError=data.frame("T-shirt vs Pants"=1:4,"T-shirt vs Shoes"=1:4,"Pants vs Shoes"=1:4,"Multiple 
```

4. Multi-level classification

4.1 Classification Algorithms

The preparation of data is identical to the binary case. The implementation of algorithms can also consistent with our discussion in the last section. Sequentially, we split the dataset into training, validation and test and therby tried Logistic regression, Linear discriminant Analysis, Classification Trees, and Random Forest. Something more is implemented here as according to Xiao (2017), the original dataset was constructed to be the entry-level test-set for neural network performance. We also constructed a 2-hidden-layer neural network and a simple convolutional neural network (CNN) to test if there are significant advantage in those methods. More rationale of those algorithms will be presented in section 5. The results are shown as follows:

4.2 Classification Results

Table 2: Error Rate in Validation Set (PCA)

	T-shirt vs Pants	T-shirt vs Shoes	Pants vs Shoes	T-shirt vs Pants vs Shoes
Logistic	0.01325	0.001	0.0005	0.01875
LDA	0.01575	0.0015	0.001	0.01275
Tree	0.03125	0.01225	0.003	0.00425
Random Forest	0.01	0.003	0.0005	0.0085
Neural Network				0.0095
Convolutional NN				0.0075

Table 3: Error Rate in Validation Set (Local Average)

	T-shirt vs Pants	T-shirt vs Shoes	Pants vs Shoes	T-shirt vs Pants vs Shoes
Logistic	0.02175	0.00175	0.00175	0.02275
LDA	0.03	0.00125	0.001	0.02925
Tree	0.032	0.00875	0.003	0.03275
Random Forest	0.012	0.00075	0.00025	0.01125

4.3 Comparison and Remarks

Generally speaking, there is no dominating strategy of dimension reduction techniques, and the difference in accuracy between PCA and the intuitive local average approach is not significant. Heuristically, the influence of dimension reduction on the final performance depends on the classification algorithm that you use.
Take classification tree and random forest as an example, the rationale of those algorithms is too find such a linear split in one of the feature that leads to the greatest homogeneity. Notice that the original data is presented in a vectorized 2-dimensional space. When applying PCA, there's no guarantee that the information about the original relative position is taken into account. The method of local average, though primitive at first sight, turns out to perform better than PCA in such kind of scenarios.
As is presented in the table, we also implemented some neural-network-based algorithms using tensorflow package. We're not going to go through the detail of those algorithms, but instead provide some heuristics. Every image has vertical and horizontal edges which are actually combining to form a image. Convolution operation is used with some filters offering it the very advantage of detecting edges. In general, it can be considered as another dimension reduction technique that represents the data in a lower dimension without losing considerable information.

4.4 Code

```r
# split the dataset
trainSet <- data[1:6000,]
valSet <- data[6001:10000,]
testSet <- data[10001:12000,]
```

```r
#Multiple-class Logistic Regression
library("nnet")
multi_logit<-multinom(y~.,trainSet)
```

```r
pred <- predict(multi_logit,valSet[,-1], "class")
MisClasiError[1,4]=1-sum(diag(table(pred, valSet[,1])))/sum(table(pred, valSet[,1]))
CrossTable(pred, valSet[,1], chisq = FALSE)
```

```
## Total Observations in Table:  4000
##
##
##              | valSet[, 1]
##         pred |          0 |          1 |          2 | Row Total |
## -------------|------------|------------|------------|-----------|
##            0 |       1328 |         18 |          4 |      1350 |
##              |   1601.117 |    413.260 |    428.424 |           |
##              |      0.984 |      0.013 |      0.003 |     0.338 |
##              |      0.964 |      0.014 |      0.003 |           |
##              |      0.332 |      0.004 |      0.001 |           |
## -------------|------------|------------|------------|-----------|
##            1 |         36 |       1310 |          2 |      1348 |
##              |    395.177 |   1659.540 |    431.750 |           |
##              |      0.027 |      0.972 |      0.001 |     0.337 |
##              |      0.026 |      0.986 |      0.002 |           |
##              |      0.009 |      0.328 |      0.001 |           |
## -------------|------------|------------|------------|-----------|
##            2 |         14 |          1 |       1287 |      1302 |
##              |    420.976 |    430.592 |   1782.441 |           |
##              |      0.011 |      0.001 |      0.988 |     0.326 |
##              |      0.010 |      0.001 |      0.995 |           |
##              |      0.004 |      0.000 |      0.322 |           |
## -------------|------------|------------|------------|-----------|
## Column Total |       1378 |       1329 |       1293 |      4000 |
##              |      0.344 |      0.332 |      0.323 |           |
## -------------|------------|------------|------------|-----------|
```

```
#LDA
lda_default <- lda(y~.,data=trainSet)
pred_lda<-predict(lda_default,valSet[,-1],type="response")$class
CrossTable(pred_lda,valSet[,1],chisq = FALSE)
```

```
## Total Observations in Table:  4000
##
##
##              | valSet[, 1]
##     pred_lda |          0 |          1 |          2 | Row Total |
## -------------|------------|------------|------------|-----------|
##            0 |       1374 |         41 |          6 |      1421 |
##              |   1598.006 |    393.688 |    447.417 |           |
##              |      0.967 |      0.029 |      0.004 |     0.355 |
##              |      0.997 |      0.031 |      0.005 |           |
##              |      0.344 |      0.010 |      0.002 |           |
## -------------|------------|------------|------------|-----------|
##            1 |          4 |       1288 |          0 |      1292 |
##              |    437.130 |   1717.864 |    417.639 |           |
##              |      0.003 |      0.997 |      0.000 |     0.323 |
##              |      0.003 |      0.969 |      0.000 |           |
##              |      0.001 |      0.322 |      0.000 |           |
## -------------|------------|------------|------------|-----------|
##            2 |          0 |          0 |       1287 |      1287 |
##              |    443.372 |    427.606 |   1823.461 |           |
##              |      0.000 |      0.000 |      1.000 |     0.322 |
##              |      0.000 |      0.000 |      0.995 |           |
##              |      0.000 |      0.000 |      0.322 |           |
## -------------|------------|------------|------------|-----------|
## Column Total |       1378 |       1329 |       1293 |      4000 |
##              |      0.344 |      0.332 |      0.323 |           |
## -------------|------------|------------|------------|-----------|
MisClasiError[2,4]=1-sum(diag(table(pred_lda,valSet[,1])))/sum(table(pred_lda,valSet[,1]))
```

```
#Tree
tree_default<-tree(y~.,trainSet)
summary(tree_default)
```

```
#plot the tree
plot(tree_default)
text(tree_default,pretty=0)
```

```
pred_tree<-predict(tree_default,valSet[,-1],type="class")
MisClasiError[3,4]=1-sum(diag(table(pred_tree,valSet[,1])))/sum(table(pred_tree,valSet[,1]))
CrossTable(pred_tree,valSet[,1],chisq = FALSE)
```

```
## Total Observations in Table:  4000
##
##
##              | valSet[, 1]
##    pred_tree |         0 |         1 |         2 | Row Total |
## -------------|-----------|-----------|-----------|-----------|
##            0 |      1275 |        55 |        14 |      1344 |
##              |  1424.016 |   343.318 |   406.899 |           |
##              |     0.949 |     0.041 |     0.010 |     0.336 |
##              |     0.925 |     0.041 |     0.011 |           |
##              |     0.319 |     0.014 |     0.004 |           |
## -------------|-----------|-----------|-----------|-----------|
##            1 |        96 |      1273 |         1 |      1370 |
##              |   299.492 |  1469.357 |   440.855 |           |
##              |     0.070 |     0.929 |     0.001 |     0.343 |
##              |     0.070 |     0.958 |     0.001 |           |
##              |     0.024 |     0.318 |     0.000 |           |
## -------------|-----------|-----------|-----------|-----------|
##            2 |         7 |         1 |      1278 |      1286 |
##              |   429.138 |   425.276 |  1788.701 |           |
##              |     0.005 |     0.001 |     0.994 |     0.322 |
##              |     0.005 |     0.001 |     0.988 |           |
##              |     0.002 |     0.000 |     0.320 |           |
## -------------|-----------|-----------|-----------|-----------|
## Column Total |      1378 |      1329 |      1293 |      4000 |
##              |     0.344 |     0.332 |     0.323 |           |
## -------------|-----------|-----------|-----------|-----------|
##
##
```

```
#pruned tree
set.seed(3)
cv.tree=cv.tree(tree_default,FUN=prune.misclass)
cv.tree$size[which.min(cv.tree$dev)]
```

```
## [1] 9
```

```
#Random forest: No need for test set
library(randomForest)
set.seed(1)
rand_forest<-randomForest(y~.,trainSet)
pred_rndForest<-predict(rand_forest,valSet[,-1])
MisClasiError[4,4]=1-sum(diag(table(pred_rndForest,valSet[,1])))/sum(table(pred_rndForest,valSet[,1]))
CrossTable(pred_rndForest,valSet[,1],chisq = FALSE)
```

```
## Total Observations in Table:  4000
##
```

```
##
##                | valSet[, 1]
## pred_rndForest |          0 |          1 |          2 | Row Total |
## ---------------|------------|------------|------------|-----------|
##              0 |       1376 |         29 |          1 |      1406 |
##                |   1641.337 |    410.944 |    452.492 |           |
##                |      0.979 |      0.021 |      0.001 |     0.351 |
##                |      0.999 |      0.022 |      0.001 |           |
##                |      0.344 |      0.007 |      0.000 |           |
## ---------------|------------|------------|------------|-----------|
##              1 |          2 |       1299 |          1 |      1302 |
##                |    444.548 |   1735.288 |    418.874 |           |
##                |      0.002 |      0.998 |      0.001 |     0.326 |
##                |      0.001 |      0.977 |      0.001 |           |
##                |      0.001 |      0.325 |      0.000 |           |
## ---------------|------------|------------|------------|-----------|
##              2 |          0 |          1 |       1291 |      1292 |
##                |    445.094 |    427.269 |   1826.361 |           |
##                |      0.000 |      0.001 |      0.999 |     0.323 |
##                |      0.000 |      0.001 |      0.998 |           |
##                |      0.000 |      0.000 |      0.323 |           |
## ---------------|------------|------------|------------|-----------|
##   Column Total |       1378 |       1329 |       1293 |      4000 |
##                |      0.344 |      0.332 |      0.323 |           |
## ---------------|------------|------------|------------|-----------|
##
##
```

```
#display the misclassification error table
MisClasiError
```

```
# choose random forest as our final model
#Random forest
set.seed(1)

#multiple-class
rand_forest<-randomForest(y~.,data)
rand_forest
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = data)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.77\%
## Confusion matrix:
##      0    1    2 class.error
## 0 5979   13    8  0.00350000
## 1  111 5888    1  0.01866667
## 2    5    1 5994  0.00100000
```

4.5 Additional Code via Convolution Neural Network

```
# Compare our results with NN and CNN (which is the origin of this dataset)
# The basic neural network, 2 layers, loss depicted by cross_entrpoy
model <- keras_model_sequential()
model %>%
  layer_flatten(input_shape = 784) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')

# Compile the model
model %>% compile(
  optimizer = 'adam',
  loss = 'sparse_categorical_crossentropy',
  metrics = c('accuracy')
)
model %>% fit(as.matrix(train_feature), as.numeric(as.matrix(train_label)), epochs = 5, verbose = 2)

score <- model %>% evaluate(as.matrix(test_feature), as.numeric(as.matrix(test_label)), verbose = 0)
cat('Test accuracy:', score$acc, "\n")
```

```
temp <- X
X2 <- array_reshape(as.matrix(temp), dim = c(18000, 28, 28))
train_feature <- X2[1:12000,,]
train_label <- y[1:12000]
test_feature <- X2[12001:18000,,]
test_label <- y[12001:18000]
```

```
model <- keras_model_sequential() %>%
  layer_conv_1d(filters = 32, kernel_size = 2, activation = "relu",
              input_shape = c(28,28)) %>%
  layer_max_pooling_1d(pool_size = 2) %>%
  layer_conv_1d(filters = 64, kernel_size = 2, activation = "relu")

summary(model)
model %>%
  layer_flatten() %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")

model %>% compile(
  optimizer = "adam",
  loss = "sparse_categorical_crossentropy",
  metrics = "accuracy"
)

history <- model %>%
  fit(
    x = train_feature, y = as.numeric(train_label),
    epochs = 10,
    verbose = 2
  )

evaluate(model, test_feature, as.numeric(test_label), verbose = 0)
```

```
## \$accuracy
## [1] 0.993
```

References

[1] Xiao, H., Rasul, K., & Vollgraf, R. 2017, arXiv e-prints, arXiv:1708.07747