Fashion-MNIST

University of Illinois Urbana Champaign
STAT432: Basics of Statistical Learning
Professor: Ruoqing Zhu

Xinjin Li(xinjinl2@illinois.edu)
Rita Cui(mcui5@ illinois.edu)
Shuo Wang(shuow6@ illinois.edu)

1. Introduction

The report focuses on the performances of several classifiers on the Fashion-MNIST dataset. Fashion-MNIST is a more complicated version of the classic MNIST benchmark image dataset. We use supervised and unsupervised classification algorithms for both binary and muti-class classification. Our method includes logistic classification, supper vector machines, K-nearest neighbor, and linear discriminative analysis. We show that our model for binary classification achieve 88.24663 accuracy, and multi-class classification achieves 76.62 accuracy by using K-nearest neighbor method.

Our main goal for this project is to evaluate the effectiveness of the results for classification using a variety of machine learning algorithms, while trying to achieve that we also need to apply dimensionality reduce techniques to the Fashion-MNIST dataset. First, we did research on relevant literature. Then analyze and pre-process the Fashion-MNIST data first, there are hundreds of pixels in our training dataset that increase the testing errors when doing predictions. Therefore, we decided to use PCA and marginal screening to help us improves machine learning algorithms performance. The resulting dataset trained faster than the regular one and yielded acceptable classification accuracy. For the Binary Classification, we used Logistic Regression and Support Vector Machine (SVM). For the multi-class classification, we used K-nearest neighbor and linear discriminant analysis.

1.2 Literature Review

The Fashion-MNIST clothing classification problem is a new standard dataset used in computer vision and deep learning. It has been determined that the model with the highest accuracy is the Fine-Tuning DARTS model, with 96.91. The second model with the best accuracy is Shake-Shake with 96.41.

The paper we discussed was centered on classifying garments based on Fashion-MNIST data by using CNN. This paper presents four different convolutional neural networks using the Fashion-MNIST dataset. Fashion-MNIST, a dataset designed to help researchers find models to classify such products (such as clothing, shoes, and pants), describes a paper that compares key classification methods to find ways to better label such data. The main purpose of this research paper is to provide a better comparison of classification methods for future research. This paper proposes a convolutional neural network approach to this problem and compares the classification results with the original results. By using a new CNN model called CNN-dropout-3, this approach can improve SVM accuracy from 89.7% to 99.1%. Throughout all the evidence provided, we can determine classifying fashion products with CNN is more accurate than by using other conventional learning models. Also, it was observed that the dropout technique together with more convulsive layers is effective when it comes to reducing the bias of a model. these new results, we discovered that cnn-dropout-3 is better (using TF2). This is good news because this model is faster to train, since it has an extra max-pooling layer that decreases dense layer inputs by a quarter.

2. Methodology

2.1 Data Description and Manipulation

2.1.1 Raw Data Description and Data Pre-Processing

Fashion-MNIST is a dataset of Zalando's article images. The training dataset contains 785 variables and 60,000 observations, and the testing dataset contains 785 variables and 10,000 observations. Each observation is a 28x28 grayscale image, associated with a label from 10 classes. Column 1 is the class label, which are indicated by numbers varying from 0 to 9, and the remaining column (784 columns) are pixel numbers. We randomly choose 10,000 rows as our actual training dataset `fashion_train`, and 5,000 rows as our actual testing dataset `fashion_test`.

```
set.seed(1)
fashion_train_2000 = sample_n(fashion_train, 10000)
fashion_test_2000 = sample_n(fashion_test, 5000)
```

We also check the how many each label in both training and testing dataset. The code below shows their frequency. Each of the number from 0 to 9 indicates different clothing type: 0 T-shirt/top, 1 Trouser, 2 Pullover, 3 Dress, 4 Coat, 5 Sandal, 6 Shirt, 7 Sneaker, 8 Bag, 9 Ankle boot.

```
train_freq_table = table(fashion_train_2000$label)
train_freq_table
```

```
##
##    0    1    2    3    4    5    6    7    8    9
## 1002  984 1012 1029 1007 1000  977  947  976 1066
```

```
test_freq_table = table(fashion_test_2000$label)
test_freq_table
```

```
##
##   0   1   2   3   4   5   6   7   8   9
## 489 469 491 507 516 489 522 517 510 490
```

2.1.2 Principal Component Analysis:

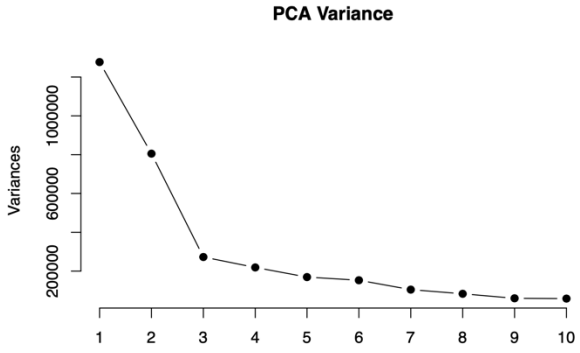
PCA is an unsupervised approach that aims to minimize information loss. A standard visual method is to choose the point which creates the most extreme “elbow” and retain that many PCs (x-axis). There are more than seven hundred predictors (pixels) in our training dataset that increase the testing errors when doing predictions. PCA can perform a type of information compression on the original by subsetting the amount of PCA components we use to describe the original data. We use `prcomp` function and get a list of objects of class `prcomp`. Below are two plots. The first one shows how many variances are explained by each principal component,

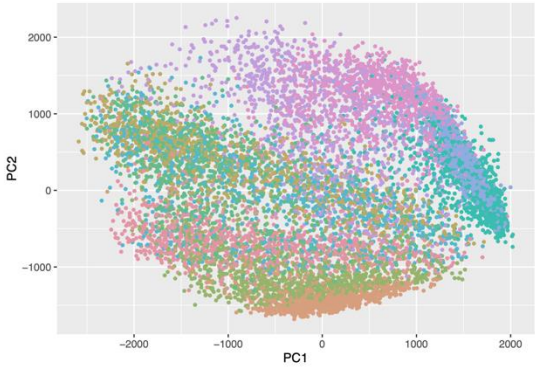
and the second one shows the representations of the train images in the feature space of the first 2 dimensions.

```
fashion_train_pc = prcomp(fashion_train_2000)
train_pc_sum = summary(fashion_train_pc)
```

```
plot(fashion_train_pc, type = "l", pch = 19, main = "PCA Variance")
```

```
ggplot(data = data.frame(fashion_train_pc$x), aes(x=PC1, y=PC2)) +
  geom_point(color = rainbow_hcl(10)[fashion_train_2000[,1]+1], size = 1)
```





For our analysis, we will use a criterion of 95% variance explained. From the 784 components that PCA yields, we will subset the minimum components needed such that the sum of the proportion of explained variance is greater than or equal to 95%.

```
df_pca <- data.frame(t(train_pc_sum$importance))
df_pca$compnum <- 1:(dim(fashion_train_2000))[2]
# How many components account for 95% of the variance in the data?
comp95 <- min(which(df_pca$Cumulative.Proportion>=0.95))
comp95
```

```
## [1] 182
```

From the result given by code above, we obtain that the first 182 components will explain at least 95% variance for our dataset.

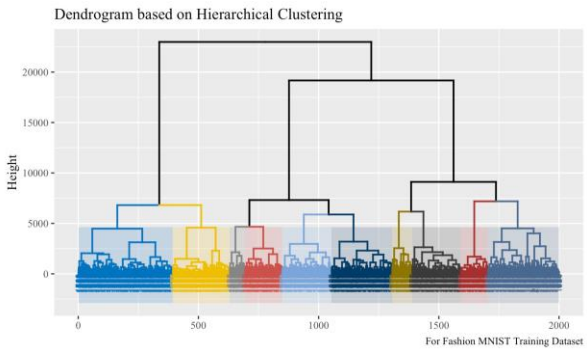
2.2 Clustering[1]

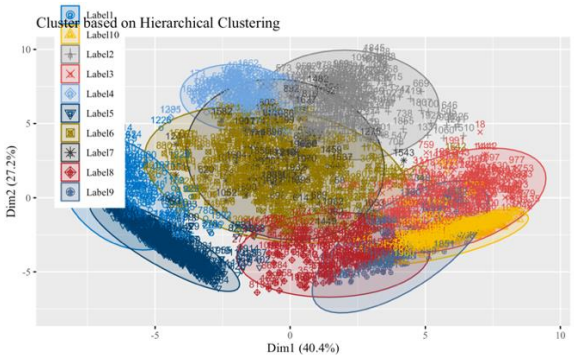
Clustering analysis is an unsupervised machine learning task, which involves automatically discovering natural grouping in data (1). We utilize Hierarchy Clustering for our subset training data. In Hierarchical Clustering, clusters are created such that they have a

[1] We use variables (pixels) with top 50 variances considering the runtime and algorithm complexity, instead of variables with top 100 variances for future analysis like binary and multi-class classification.

predetermined ordering i.e. a hierarchy. In our analysis, we use all subset training data for clustering. We use ` hclust()` function to apply clustering to this subset, and we set the method "ward.D2". Ward's method uses the linkage function specifying the distance between two clusters is computed as the increase in the "error sum of squares" (ESS) after fusing two clusters into a single cluster. In the dendrogram displayed below, each leaf represents one observation.

From the first plot below, we notice that there are approximately three dominate clusters with 10 smaller leaf clusters at the bottom, and we visualize them by the graph on the right, plotting with ten colors. The dominant `y` label in each of these clusters is the first one, as shown in the first graph below, with color blue.





2.3 Binary Classification

We want to apply a binary classification using coat and shirt, which requires us to subset our dataset with rows that label = 4 and label = 6.

```
train_binary = subset(fashion_train_2000, label==4 | label==6)
train_binary$label = as.factor(train_binary$label)

test_binary = subset(fashion_test_2000, label==4 | label==6) # get 4 and 6
test_binary$label = as.factor(test_binary$label)
```

2.3.1 Logistic Regression

Logistic regression is used to predict the class (or category) of individuals based on one or multiple predictor variables (x). It is used to model a binary outcome, that is a variable, which can have only two possible values, label 4 and label 6 for our analysis. By fitting the logistic classification model with `glm` function, we obtain errors for the `glm` algorithm does not converge and the prediction from a rank-deficiency fit may be misleading. The possible reason is there existing two predictors are highly correlated since neighboring range pixels are much more highly correlated than neighboring luminance pixel. Also, some pixels in our fashion_train

dataset have no variation since many pixels are completely blank for all observations. There are two possible solutions, using a subset of current training dataset, and using penalized logistic regression.

2.3.1.1 Regular Logistic Regression

We could perform a marginal screening of all pixels in the train_binary dataset as the code shown below.

```
allvar = apply(train_binary[, -1], 2, var)
cut = sort(allvar, decreasing = TRUE)[100]
binary_train_100 = train_binary[, -(which(allvar < cut)+1)]

binary_test_100 = test_binary[, -(which(allvar < cut)+1)]
```

First, we calculate the variance of all pixels and select pixels that ranked among the 100 largest variances and keep these pixels in our dataset. Based on current columns in fashion_train, we perform subset for fashion_test so the two datasets can have same columns for future calculation. We fit the logistic classification using ` binary_train_100` and use ` binary_test_100` for prediction.

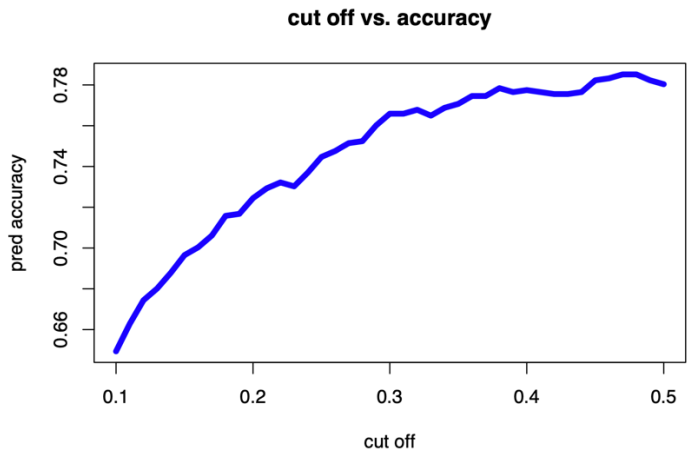
```
max(logistic_accuracy) # best accuracy for regular logistic
```

```
## [1] 0.7851638
```

```
cutoff[which.max(logistic_accuracy)]
```

```
## [1] 0.47
```

Since the prediction of a logistic regression model is a probability, in order to use it as a classifier, we will have to choose a cutoff value (threshold value). Where scores above this value will be classified as positive, those below as negative. Therefore, the choice of the cutoff value is important. We create a sequence of cutoff from 0.1 to 0.5 with increment by 0.01 at each time, and calculate the prediction accuracy, which we could find that when cutoff = 0.47 the regular logistic regression will achieve highest prediction accuracy, 0.7851638.



2.3.1.2 Lasso-Penalized Logistic Regression

Penalized logistic regression imposes a penalty to the logistic model for having too many variables. This results in shrinking the coefficients of the less contributive variables toward zero. The dataset used in penalized logistic classification is the full dataset with label 4 and 6 selection. We use the lasso penalty by `cv.glmnet` function to create the model, which is penalized with the L1-norm of summation of the absolute coefficients. In our lasso penalty, we choose to use `s = lambda.min` as the penalty parameter. Using accuracy as our evaluation criterion, we create the same sequence as the regular logistic classification and find the cutoff value gives the highest testing accuracy.

```
lasso.fit = cv.glmnet(x = data.matrix(train_binary[,- 1]),
                      y=as.numeric(train_binary$label), nfold = 10, family = "binomial")
```

```
logistic_penalize_pred = predict(lasso.fit,
                                 newx = data.matrix(test_binary[,- 1]),
                                 s = "lambda.min",
                                 type = "response")
```

```
cutoff_penalize = seq(0.1, 0.5, 0.01)
num_penalize = length(cutoff_penalize)
logistic_accuracy_penalize <- c()
for (i in 1:num_penalize) {
  logistic_table = table(logistic_penalize_pred > cutoff_penalize[i],
                         test_binary$label)
  logistic_accuracy_penalize[i] = (logistic_table[1,1]+logistic_table[2,2])/
```

```
    sum(logistic_table)
}

max(logistic_accuracy_penalize) # best accuracy for regular logistic
```

```
## [1] 0.8824663
```

```
cutoff_penalize[which.max(logistic_accuracy_penalize)]
```

```
## [1] 0.48
```

The cutoff is 0.48, and the highest test accuracy is 0.8824663. Based on the given result, we could conclude that the generalized regression lasso classifier provides a more accurate predictive model.

2.3.2 Support Vector Machine

Support vector machines (SVM) are supervised learning methods used for both classification and regression models. SVM can classify features in a training set int categories that use either a linear or non-linear model. The linearity of the classifier is determined by the kernel function of the dataset (e.g.: linear, nonlinear, polynomial, radial basis function, and sigmoid). In our analysis, we would use linear kernel and radial kernel for training the models as comparison.

As same as logistic classification, we use the training and testing dataset with largest 100 variances pixel. We consider centering and scaling the covariates since all predictors need to normalize and make their scale comparable and set the method to “svmLinear”. Also, we set ` turn.grid = expand.grid(C = c(0.01, 0.1, 0.5, 1))`, and ` train_control = trainControl(method="cv", number=10)`. The turning parameter in turn.grid, also known as Cost, determined the possible misclassification, which imposes a penalty to the model for making an error: the higher the value C is, the less likely it is that the SVM algorithms will mis-classify a point.

```
library(caret)
set.seed(1)
turn.grid = expand.grid(C = c(0.01, 0.1, 0.5, 1))
train_control = trainControl(method="cv", number=10)

svm.linear <- train(y ~ .,
                    data = data.frame("x" = binary_train_100[,-1],
                                      "y" = as.factor(binary_train_100[,1])),
                    method = "svmLinear",
                    preProcess = c("center", "scale"),
                    tuneGrid = turn.grid,
                    trControl = train_control)
```

```
## Support Vector Machines with Linear Kernel
##
## 1984 samples
##  100 predictor
##    2 classes: '4', '6'
##
## Pre-processing: centered (100), scaled (100)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1787, 1785, 1786, 1786, 1785, 1786, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.01  0.7817434  0.5634548
##   0.10  0.7777360  0.5555629
##   0.50  0.7691653  0.5382933
##   1.00  0.7696729  0.5393242
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

The final value used for the model is C = 0.01, and it gives the highest prediction accuracy 0.8265896.

```
svm.binary_pred <- predict(svm.linear.best, binary_test_100[,-1])
```

```
sum(diag(table(svm.binary_pred,
               as.factor(binary_test_100[,1])))) / nrow(binary_test_100)
```

```
## [1] 0.8265896
```

We also use radial basis function (RBF) as the method for training the mode. Here is the RBF:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\gamma \left|\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right|^2\right), \gamma > 0$$

Here γ is a hyper parameter or a tuning parameter which accounts for the smoothness of the decision boundary and controls the variance of the model. If γ is very large, then we get quiet fluctuating and wiggly decision boundaries which accounts for high variance and overfitting. If γ is small, the decision line or boundary is smoother and has low variance.

```
cost.grid = expand.grid(C = c(0.1, 0.5, 1, 5), sigma = c(0.01, 0.1, 1))
train_control = trainControl(method="repeatedcv", number=10, repeats=3)
# data = data.frame("x" = binary_train_50[,- 1], "y" = as.factor(binary_train_50[,1])
svm.radial <- train(label ~ ., data = binary_train_100,
                    method = "svmRadial",
                    preProcess = c("center", "scale"),
                    tuneGrid = cost.grid,
                    trControl = train_control)
svm.radial
```

We set the cost grid with `cost` and `sigma`, and we obtained sigma = 0.01, and cost = 5, which gives the highest accuracy, 0.871869.

```
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01 and C = 5.
```

```
svm.radial.best <- ksvm(x=as.matrix(binary_train_100[,-1]),
                        y=as.factor(binary_train_100[,1]),
                        kernel=rbfdot(sigma=0.01), C=5)

svm.radial_pred <- predict(svm.radial.best, binary_test_100[,-1])
table(svm.radial_pred, as.factor(binary_test_100[,1]))
```

```
##
## svm.radial_pred   4   6
##               4 466  83
##               6  50 439
```

```
# testing data accuracy
sum(diag(table(svm.radial_pred,
               as.factor(binary_test_100[,1])))) / nrow(binary_test_100)
```

```
## [1] 0.871869
```

2.4 Multiclass Classification

Multiclass classification is a classification task with more than two classes. Multiclass classification assumes that each sample is assigned to only one label. In this part of analysis, we decided to apply similar strategies as mentioned in binary classification (logistic). For the original dataset (fashion_train and fashion_test), we extract pixels with the largest 100 variances[2] and store these pixel columns with the corresponding labels into `multi_train`, and we also select the corresponding pixel columns as the testing dataset, `multi_test`. Specific reasons for subsetting dataset would be discussed in each part. We would apply K Nearest Neighbor and Linear Discriminative Analysis for this multiclass classification problem.

2.4.1 KNN

K-nearest neighbors algorithm is a non-parametric supervised learning method. A supervised machine learning algorithm is one that relies on labeled input data to learn function that produced an appropriate output when given new unlabeled data. We used KNN classifier tested with different parameter sweeps, setting different values of k with a sequence from 1 to 20

[2] The code for how to select largest 100 variance variables is similar to the method to select variables for binary classification. Please refer to 2.3.1.1.

with increment by 1 at each time. Next, we perform our search grid. Since we are working with a large dataset, using resampling (k-fold cross validation) becomes costly.

```
control = trainControl(method = "repeatedcv", number = 5, repeats = 3)
set.seed(651769293)
knn.cvfit <- train(y ~ ., method = "knn",
                   data = data.frame("x" = multi_train[, -1],
                                     "y" = as.factor(multi_train[, 1])),
                   tuneGrid = data.frame(k = seq(1, 20, 1)),
                   trControl = control)
```

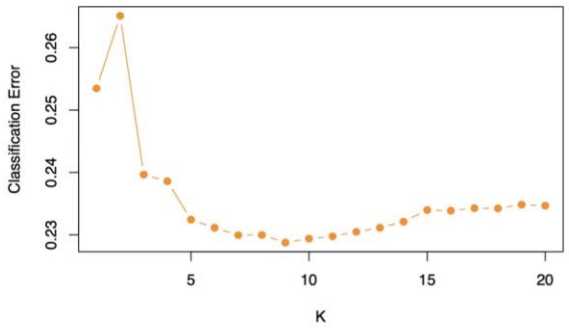
We would like to tune our model with the number of nearest neighbors and use the classification error as our evaluation criterion. By setting `control = trainControl(method = "repeatedcv", number = 5, repeats = 3) `, we use `repeatedcv` instead of `cv` as our train control method since a single run of the k-fold cross-validation procedure may result in a noisy estimate of model performance. Different splits of the data may result in quite different results. Repeated k-fold cross-validation provides a way to improve the estimated performance of a machine learning model. This involves simply repeating the cross-validation procedure multiple times and reporting the mean result across all folds from all runs. This mean result is expected to be a more accurate estimate of the true unknown underlying mean performance of the model on the dataset, as calculated using the standard error. The graph below is the number of nearest neighbors K versus the classification prediction error. The best accuracy we achieve is 0.7662 when k = 9.

```
mean(testpred == as.factor(multi_test[, 1]))
```

```
## [1] 0.7662
```

The prediction table on the right is how much data belongs to X label, and now belong to Y label in the prediction result.

KNN classification error



KNN Test Prediction Table

```
##
## testpred   0   1   2   3   4   5   6   7   8   9
##         0 381   3  12  51   6   2 122   0   1   0
##         1   2 443   0  12   3   0   1   0   0   0
##         2  19   5 373  13 153   0  95   0  14   0
##         3  36  12   3 408  25   2  25   0   6   0
##         4  11   3  66  16 289   0  45   0   9   0
##         5   0   0   0   0   0 351   0  23   3   8
##         6  18   3  23   4  30   1 213   0   3   0
##         7   2   0   0   0   0  80   0 445   4  20
##         8  20   0  13   3  10   8  21   0 466   0
##         9   0   0   1   0   0  45   0  49   4 462
```

Runtime for KNN: (k=9)

The model fitting time by using `knn` needs 7.887216 seconds.

```
start_time_1 <- Sys.time()
testpred = knn(train = multi_train[, -1], test = multi_test[, -1],
               cl = as.factor(multi_train[, 1]), k = 9)
end_time_1 <- Sys.time()
print(end_time_1 - start_time_1)
```

```
Time difference of 7.887216 secs
```

2.4.2 Linear Discriminative Analysis

Discriminant analysis is statistical technique used to classify observations into non-overlapping groups, based on scores on one or more quantitative predictor variables. With multiple discriminant analysis, the goal is to define discriminant functions that maximize differences between groups and minimize differences within groups. We apply linar discriminative analysis for our Fashion MNIST dataset. Linear Discriminant Analysis (LDA) is mainly used to classify multiclass classification problems. The LDA model estimates the mean and variance for each class in a dataset and finds out covariance to discriminate each class.

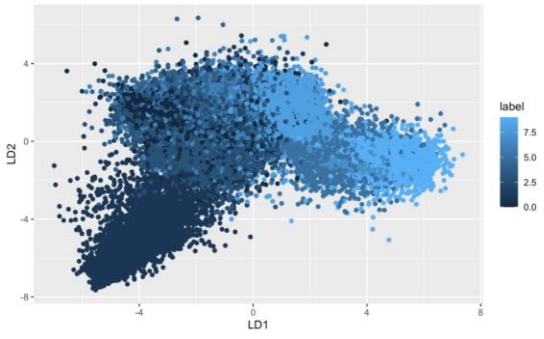
We fit the LDA model using the `lda()` function from the `MASS` package for `multi-train` dataset. The reason why we choose the 100 highest variance pixels as predictors because there would exist rank deficiency when calculating the inverse of covariance matrix. The accuracy finally we achieve is 0.735.

```
multi.lda = lda(label ~ ., data = multi_train)
multi.lda_predict = predict(multi.lda, multi_test)$class
# table(multi.lda_predict, multi_test[, 1])
mean(multi.lda_predict == as.factor(multi_test[, 1]))
```

```
## [1] 0.735
```

LDA Prediction Table

multi.lda_predict	0	1	2	3	4	5	6	7	8	9
0	721	5	26	56	4	0	169	0	4	0
1	1	903	0	9	3	0	2	0	0	0
2	21	13	686	19	217	0	119	0	16	0
3	124	63	9	829	74	1	79	0	16	0
4	9	4	131	19	555	0	132	0	12	0
5	15	6	11	25	2	730	20	142	43	61
6	66	6	105	35	134	2	426	0	30	0
7	2	0	1	0	0	161	5	738	14	33
8	40	0	29	8	11	14	48	2	859	3
9	1	0	2	0	0	92	0	118	6	903



Runtime for LDA: The model fitting is very quick, which is less than 1s (0.7943671 seconds).

```
start_time_2 <- Sys.time()
multi.lda = lda(label ~ ., data = multi_train)
end_time_2 <- Sys.time()
print(end_time_2 - start_time_2)
```

```
Time difference of 0.7943671 secs
```

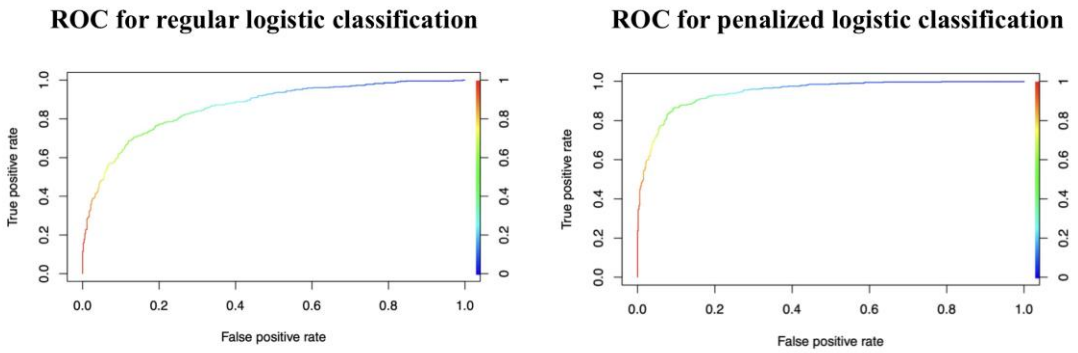
3. Discussion

In this analysis, we mainly use PCA, hierarchy clustering, logistic classification, support vector machine, k-nearest neighbor, and linear discriminant analysis. PCA are helpful because it will reduce the data redundance, the chances of overfitting, and the time necessary to train models. In binary classification part, we would like to use ROC curves to compare different algorithms performances, besides the prediction accuracy we talked before. ROC curves in logistic regression are used for determining the best cutoff value for predicting whether a new observation is a "failure" (0) or a "success" (1). We talked about the cut-off problem in the 2.3.1. So, what an ROC curve does is looks at every possible cutoff value that results in a change of classification of any observation in the data set. Whatever cutoff you choose, a certain number of the rows of data will be correctly classified (you predicted the correct value for that row), and a certain number will be misclassified. Sensitivity and specificity are two metrics for evaluating the proportion of true positives and true negatives. Mathematically these are represented as:

$$Sensitivty = \frac{number\ correctly\ identified\ 1s}{total\ number\ observed\ 1s}$$

$$Specificity = \frac{number\ correctly\ identified\ 0s}{total\ number\ observed\ 0s}$$

The Area Under the ROC curve (AUC) is an aggregated metric that evaluates how well a logistic regression model classifies positive and negative outcomes at all possible cutoffs. It can range from 0.5 to 1, and the higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes. Below are two graphs for ROC-AUC for both regular and penalized logistic classifiers.



The regular logistic gives AUC = 0.8621878, and the penalized logistic gives AUC = 0.9454914. This result stays consistent with our prediction accuracy. We use LASSO that the coefficients of some less contributive variables are forced to be exactly zero. Only the most significant variables are kept in the final model. This could be helpful when dealing with dataset similar to Fashion MNIST that consists of many zero entries in different columns.

4. Conclusion

Through these analysis, we used a marginal screening approach to reduce the dimensionality of Fashion-MNIST from 784 to 100 features. based on our calculation, for the Binary Classification, Lasso-Penalized Logistic Regression achieve higher accuracy which is 88.24663. For multi-class classification, K-nearest neighbor method achieve higher accuracy which is 76.62. Clearly, while doing contrast with the research paper we have read, we still have insufficiency. Hopefully, in the future we can improve the accuracy by learning more algorithm.

5. References[3]

Brownlee, Jason. "10 Clustering Algorithms With Python." Machine Learning Mastery, 20 Aug. 2020, machinelearningmastery.com/clustering-algorithms-with-python/#:%7E:text=Cluster%20analysis%2C%20or%20clustering%2C%20is,or%20clusters%20in%20feature%20space.

"Repeated K-Fold Cross-Validation for Model Evaluation in Python." Machine Learning Mastery, 26 Aug. 2020, machinelearningmastery.com/repeated-k-fold-cross-validation-with-python.

GraphPad Software, LLC. "GraphPad Prism 9 Curve Fitting Guide - Interpreting Logistic ROC Curves." GraphPad, www.graphpad.com/guides/prism/latest/curve-fitting/reg_logistic_roc_curves.htm. Accessed 5 May 2022.

"RPubs - Fashion MNIST PCA." Rpub, 28 Nov. 2021, rpubs.com/SmilodonCub/840625.

"RPubs - Hierarchical Clustering." Rpub, 25 Dec. 2020, rpubs.com/chidungkt/707528.

Toshniwal, Ruchi. "Demystifying ROC Curves - Towards Data Science." Medium, 13 Dec. 2021, towardsdatascience.com/demystifying-roc-curves-df809474529a.

Wilkinson, Prof. Richard. "4.3 An Alternative View of PCA | Multivariate Statistics." Applied Multivariate Statistics, rich-d-wilkinson.github.io/MATH3030/4-3-an-alternative-view-of-pca.html. Accessed 5 May 2022.

"Papers with Code - Fashion-MNIST Benchmark (Image Classification)." *The Latest in Machine Learning,* https://paperswithcode.com/sota/image-classification-on-fashion-mnist.

"Classifying Garments from Fashion-Mnist Dataset Through CNNs." *Research Gate,*

[3] We refer to all the lecture notes and homework in Spring 2022 STAT432 website: https://teazrq.github.io/stat432/.

https://www.researchgate.net/publication/349553873_Classifying_Garments_from_Fashion-MNIST_Dataset_Through_CNNs