EXAM_SL

roi hezkiyahu - 205884018

2023-01-23

imports

```r
library(dplyr)
library(ggplot2)
library(purrr)
library(caret)
library(class)
library(MASS)
library(tidyverse)
library(nnet)
library(keras)
```

Q1

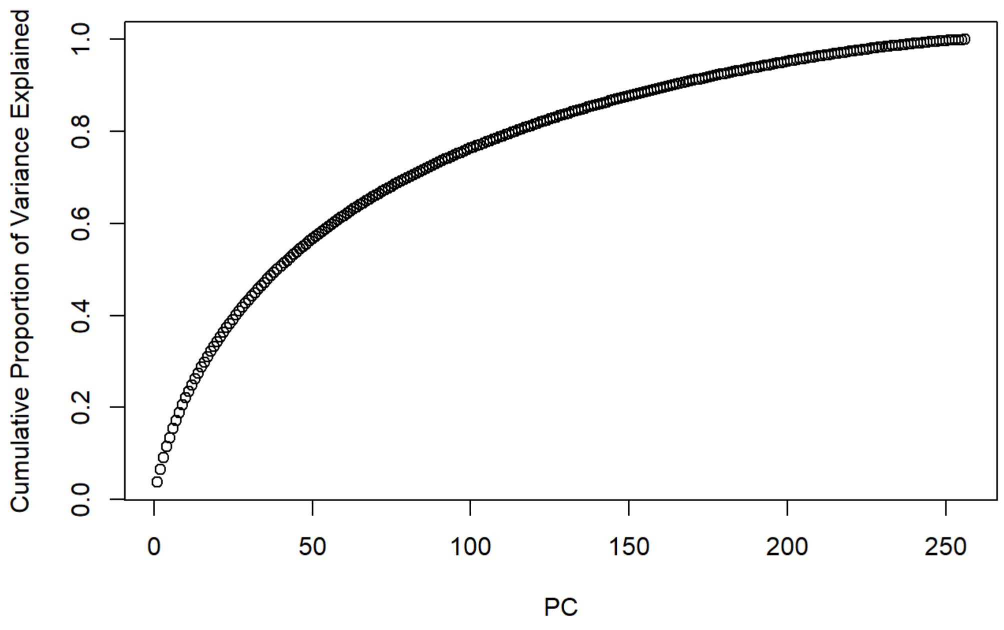
loading the data:

```r
df_train <- read.table("zip.train") %>% rename("y" = "V1") %>%
  filter(y %in% c(2,3,5))
df_test <- read.table("zip.test") %>% rename("y" = "V1") %>%
  filter(y %in% c(2,3,5))
X_train = df_train[,2:257]
y_train = df_train[,1]
X_test = df_test[,2:257]
y_test = df_test[,1]
```

1.a

- using PCA

```r
pca <- prcomp(X_train)
plot(cumsum(pca$sdev/sum(pca$sdev)),
     xlab = "PC",
     ylab = "Cumulative Proportion of Variance Explained")
```



i will take 100 PCS which explaines ~ 75% of the variance in the data.

```
X_pca_train_100 = as_tibble(pca$x[,1:100])
X_pca_test_100 = as_tibble(predict(pca,X_test)[,1:100])
```

```
lda_full = lda(y~., df_train)
df_pca_train_100 = X_pca_train_100 %>% mutate(y=y_train)
lda_pca = lda(y~., df_pca_train_100)
```

```
get_results = function(model, X, y_true, lda=TRUE, neural_model=FALSE){
  # this function returns the confusion matrix and miss classification error of a given model
  y_pred = predict(model, X)
  if (lda) {
     y_pred = y_pred$class
  }
  if (neural_model){
    y_pred = apply(y_pred,1,which.max)
    y_pred[y_pred==1] = 5
  }
  classification_err = mean(y_pred != y_true)
  conf_matrix = confusionMatrix(as.factor(y_true), as.factor(y_pred))
  return( list("classification_err" = classification_err,
               "conf_matrix" =   conf_matrix$table))
}
lda_train_results = get_results(lda_full, X_train, y_train, TRUE)
lda_test_results = get_results(lda_full, X_test, y_test, TRUE)
lda_pca_train_results = get_results(lda_pca, X_pca_train_100, y_train, TRUE)
lda_pca_test_results = get_results(lda_pca, X_pca_test_100, y_test, TRUE)
```

```
## # A tibble: 4 × 2
##   model_dataset         err
##   <chr>               <dbl>
## 1 lda train          0.0231
## 2 lda test           0.0954
## 3 lda with pca train 0.0375
## 4 lda with pca test  0.0954
```

we can see that both the model with the PCA and the model without the PCA preform rather similar on the test set (9.5% error) but the model on the full data tends to overfits more then the PCA. meaning the PCA here helps reduce the overfit.

lets take a look at the confution matrix of the full LDA

train:

```
##           Reference
## Prediction   2   3   5
##          2 719  10   2
##          3   7 644   7
##          5   3  16 537
```

we can see that the model struggels more to distinguish between 5 and 3, after that it has a hard time with 3 and 2, and can seperate well enougth 5 and 2

test:

```
##           Reference
## Prediction   2   3   5
##          2 184  10   4
##          3   6 148  12
##          5   0  18 142
```

we can see in general the same behavior as in the train set, but with larger percentage of mistake, notice that the test set is smaller then the train set and that we have higher values outside the diagonal (which indicate false prediction)

1.b

```
multinom_full = multinom(y~., df_train)
```

```
## # weights:  774 (514 variable)
## initial  value 2136.800901
## iter  10 value 400.032251
## iter  20 value 139.626891
## iter  30 value 56.619769
## iter  40 value 24.791592
## iter  50 value 15.917622
## iter  60 value 8.683908
## iter  70 value 3.882552
## iter  80 value 0.925665
## iter  90 value 0.245970
## iter 100 value 0.066039
## final  value 0.066039
## stopped after 100 iterations
```

```
multinom_pca = multinom(y~., df_pca_train_100)
```

```
## # weights:  306 (202 variable)
## initial  value 2136.800901
## iter  10 value 192.871780
## iter  20 value 97.706870
## iter  30 value 54.966766
## iter  40 value 43.944227
## iter  50 value 41.985445
## iter  60 value 40.308383
## iter  70 value 37.790193
## iter  80 value 27.292217
## iter  90 value 0.983953
## iter 100 value 0.013846
## final  value 0.013846
## stopped after 100 iterations
```

```
multinom_train_results = get_results(multinom_full, X_train, y_train, FALSE)
multinom_test_results = get_results(multinom_full, X_test, y_test, FALSE)
multinom_pca_train_results = get_results(multinom_pca, X_pca_train_100, y_train, FALSE)
multinom_pca_test_results = get_results(multinom_pca, X_pca_test_100, y_test, FALSE)
```

```
## # A tibble: 4 × 2
##   model_dataset              err
##   <chr>                    <dbl>
## 1 multinom train          0     
## 2 multinom test           0.145 
## 3 multinom with pca train 0     
## 4 multinom with pca test  0.0935
```

clearly we overfit the data, we can see a train error of 0, which means that in the train set we are able to classify each observation to the correct y value, but for the test set we have much larger errors 14.5% for the full multi-class logistic regression, and 9.35% for the multi-class logistic regression with PCA. we see that in this case the PCA helps improve the results of the model, also the multi-class logistic regression with PCA has similar results to the LDA

lets take a look at the confution matrix of the PCA multi-class logistic regression the train has perfect score thus the confution matrix is diagonal and not that interesting, lets see the test confution matrix:

```
##           Reference
## Prediction   2   3   5
##          2 184  10   4
##          3   6 146  14
##          5   4  11 145
```

we can see a rather similar performance, the model struggles the most with 5 and 3, after that with 3 and 2, and is rather ok with 5 and 2.

1.c

we are using images, and as we know the SOTA for image predictive modeling are CNNS so lets make a CNN to predict the class

first lets prepare the data: - the loss expects values to be between 0-2, so i will reencode 5 => 0, 2=> 1, 3 => 2

```
X_train_net = array_reshape(as.matrix(X_train), c(nrow(X_train),16,16,1))
y_train_net = y_train-1
y_train_net[y_train_net == 4] = 0

X_test_net = array_reshape(as.matrix(X_test), c(nrow(X_test),16,16,1))
y_test_net = y_test-1
y_test_net[y_test_net == 4] = 0
```

now create and train the model

```
cnn_model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 16, kernel_size = c(3, 3), activation = "relu", input_shape = c(16, 16, 1)) %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_batch_normalization() %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 3, activation = "softmax")


summary(cnn_model)
```

```
## Model: "sequential"
## ____________________________________________________________________
## Layer (type)                   Output Shape               Param #
## ====================================================================
## conv2d_1 (Conv2D)              (None, 14, 14, 16)         160
## ____________________________________________________________________
## conv2d (Conv2D)                (None, 12, 12, 32)         4640
## ____________________________________________________________________
## max_pooling2d (MaxPooling2D)   (None, 6, 6, 32)           0
## ____________________________________________________________________
## flatten (Flatten)              (None, 1152)               0
## ____________________________________________________________________
## batch_normalization (BatchNorm (None, 1152)               4608
## ____________________________________________________________________
## dense_2 (Dense)                (None, 64)                 73792
## ____________________________________________________________________
## dropout_1 (Dropout)            (None, 64)                 0
## ____________________________________________________________________
## dense_1 (Dense)                (None, 32)                 2080
## ____________________________________________________________________
## dropout (Dropout)              (None, 32)                 0
## ____________________________________________________________________
## dense (Dense)                  (None, 3)                  99
## ====================================================================
## Total params: 85,379
## Trainable params: 83,075
## Non-trainable params: 2,304
## ____________________________________________________________________
```

```
cnn_model %>% compile(loss = 'sparse_categorical_crossentropy',optimizer = optimizer_adam(learning_rate = 0.0001),metric
s = c('accuracy'))
cnn_model %>% fit(X_train_net, y_train_net, batch_size = 64,epochs = 100,verbose = 1,
                  class_weight = list("0" = 1.5, "1" = 0.7, "2" = 1))
```

```
cnn_train_res = get_results(cnn_model, X_train_net, y_train, FALSE, TRUE)
cnn_test_res = get_results(cnn_model, X_test_net, y_test, FALSE, TRUE)
tibble(model_dataset = c("neural net train", "neural net test"),
       err = c(cnn_train_res$classification_err, cnn_test_res$classification_err))
```

```
## # A tibble: 2 × 2
##   model_dataset       err
##   <chr>             <dbl>
## 1 neural net train 0
## 2 neural net test  0.0324
```

we can see that this model is far better then the ones before, also it still has a much lower error in the train set, we can keep playing with the paramaters of the model (add some more regularization or use data augmentation -[which for some reason doesnt work for me in r]) to get even better results

lets take a look at the test confution matrix

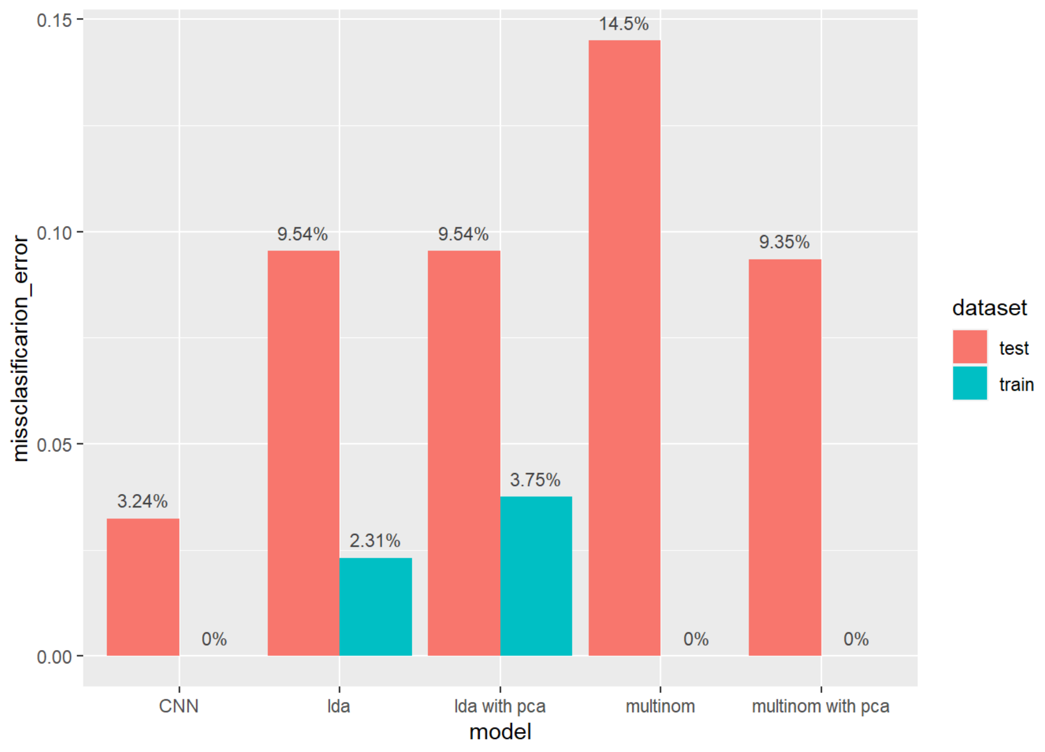
```
##           Reference
## Prediction   2   3   5
##          2 194   1   3
##          3   2 156   8
##          5   0   3 157
```

we can see that the results are much better, we still some dificulties with the 5's which is most chakkenging to classify.

the over all results:

```
## # A tibble: 5 × 3
##   model_dataset     err_train err_test
##   <chr>                 <dbl>    <dbl>
## 1 neural net           0        0.0324
## 2 multinom with pca    0        0.0935
## 3 lda                  0.0231   0.0954
## 4 lda with pca         0.0375   0.0954
## 5 multinom             0        0.145
```

```
## Warning in geom_col(stat = "identity", position = "dodge"): Ignoring unknown
## parameters: `stat`
```



we can see from the full results that the multinom and neural net both have perfect score on the train set, also the neural net has the best results on the test set, and even better results on the rest set then other models get on the train set!

Q2

2a.i

x has many categorical variables and missing values, accuracy is critical good models can be - random forest and gradient boosting, both models have lower variance due to the fact that they average many predictors which will result in better accuracy, also both of them can handle missing values

bad models can be: Linear regression - LR will probably preform not that good as it can’t handle missing values well.

2a.ii

x has many categorical variables and missing values, speed of model building is critical

good: Two models that can be good given the proper processing of missing values can be Logistic regression for classification and Linear regression for regression problems. they are both fast to train and run inference, but both will have a hard time with the high dimension so we can use lasso as a feature selection technique and thus reduce the dimension or ridge to shrink the coefficients. *a proper way for processing the missing values could be to set a new category “missing” for each categorical feature with missing values, after that we can use one hot encoding. it might even surprise us with very good predictions if the data is not missing at random and might have a connection to our target.

bad models: Random forest, Boosting, both these models can achieve high accuracy and tree based models can handle missing values as well, but they will take a larger time to train and also run inference, each observation mush have prediction from all the trees and usually we want a lot of trees to be accurate, also each tree inference time is rather costly. K-nn will also not be that good here as for each new observation we need to calculate thee distance to all points in the train set in order to find the K closest ones

2a.iii

Large n, small p, you believe that there are complex dependencies between x and y (not only main effects / additive / low order)

good - gradient boosting, random forest, deep learning and kernel machines can model complex dependencies between x and y thus all of them will be good in this case, also because we have a large number of samples and not many variables they will likely perform well without any regularization. the tree based models can capture complex dependencies as they split the sub space, neural nets can capture complex dependencies by using nonlinear activation functions, and kernel machines can capture complex dependencies by the nature of the dimention increacment

bad - Linear regression for regression, Logistic regression for classification, LDA for classification, all of these models assume some form of linearity thus will not be cable to capture complex dependencies between x and y and will probably preform poorly

2a.iv

Small n, large p, you believe only a small number of variables are important

Good - Lasso: will help reduce the dimension of the problem by setting some of the coefficients to zero and keeping the important coefiecents. We can use lasso as a model, but also as a feature selection technique and then use all of the other models as with the selected features. We can also use PCA to reduce p and afterwards we can use all the models (if we can find a d dimensional subspace where d<<n)

Bad - linear regression will be very bad due to the fact that X^tX is not invertible. also without any regularization we can’t expect any of the other models to perform well when p>>n

2b

LDA assumption: we assume that $X|Y_i \sim N(\mu_i, \Sigma)$

LR assumption: the model that we have for the linear regression is $y = X\beta + \varepsilon$ we assume that $\varepsilon \sim N(0, \sigma^2)$

Ridge assumption: Ridge assumes a normal prior for β meaning: $\beta \sim N(0, \tau^2 I)$

Q3

3.a

the ridge regression model yields $\hat{y} = X\hat{\beta}_{ridge} = X(X^TX + \lambda I)^{-1}X^TY$

setting $S(X, \lambda) = X(X^X + \lambda I)^-1X^T$ we get that ridge regression follows the first condition

3.b

for clearer writing i will use the notation k instead of i_0

$$\text{denote } \hat{\beta} = argmin_\beta \sum_i (\tilde{y}_i^{(-k)} - ((X^{(-k)})^t\beta)_i)^2 + \lambda \sum_j \beta_j^2 = argmin_\beta L_k$$

$$\hat{\tilde{\beta}} = argmin_\beta \sum_i (\tilde{y}_i - X_i^t\beta)^2 + \lambda \sum_j \beta_j^2$$

lets break doen the above expression

$$\sum_i (\tilde{y}_i - X_i^t\beta)^2 + \lambda \sum_j \beta_j^2 = \sum_{i\neq k} (y_i - X_i^t\beta)^2 + (\hat{y}_k^{(-k)} - X_k^t\beta)^2 + \lambda \sum_j \beta_j^2 = \sum_{i\neq k} (y_i - X_i^t\beta)^2 + (X_k^T\hat{\beta} - X_k^t\beta)^2 + \lambda \sum_j \beta_j^2 =$$

$$= \sum_{i\neq k} (y_i - X_i^t\beta)^2 + (X_k^T(\hat{\beta} - \beta))^2 + \lambda \sum_j \beta_j^2 = L_k + (X_k^T(\hat{\beta} - \beta))^2$$

$$\text{thus } \hat{\tilde{\beta}} = argmin_\beta L_k + (X_k^T(\hat{\beta} - \beta))^2$$

the argmin of L_k is $\hat{\beta}$ and if we plug it in the formulaa above we get $L_k(\hat{\beta})$ which achives the minimum of the equation above

$$\text{thus we can conclude that } \hat{\tilde{\beta}} = \hat{\beta}$$

$$\hat{\tilde{y}}_k = S(\tilde{y})_k = X_k\tilde{\beta} = X_k\hat{\beta} = \hat{y}_k^{(-k)}$$

3.c

we saw in calss that for a linear model (like ridge regression as we proved above) the optimism is: $\frac{2\sigma^2 tr(S)}{n}$

thus the smaller the diagonal elements are the smaller tr(S) becomes and the optimisim is smaller

$$S(X,\lambda)_{ii} = (X(X^TX + \lambda I)^{-1}X^T)_{ii}$$

using the SVD decomposition we can derive:

$$X = UDV^t$$
$$X^tX = VD^2V^t$$
$$(X^tX)^{-1} = VD^{-2}V^t$$
$$(X^tX + \lambda I_p)^{-1} = V(D^2 + \lambda I_p)^{-1}V^t$$
$$X(X^tX + \lambda I_p)^{-1}X^t = UD(D^2 + \lambda I_p)^{-1}DU^t = D^2(D^2 + \lambda I_p)^{-1}$$
$$S(X,\lambda)_{ii} = D_{ii}^2(D_{ii}^2 + \lambda I_p)^{-1}$$

thus $S(X,\lambda)_{ii}$ is a decreaceing funtion of λ

3.d

i will show it formally, by differentiating the expression

$$\frac{\partial D_{ii}^2(D_{ii}^2 + \lambda I_p)^{-1}}{\lambda} = -D_{ii}^2(D_{ii}^2 + \lambda I_p)^{-2} < 0 \quad \forall\lambda$$

thus we can conclude that $S(X,\lambda)_{ii}$ is indeed a decreaceing funtion of λ

3.e

we would not expect the lemma to hold for the lasso penelty

the solution for the lasso problem does not hold for the first condition

this is because we cannot derive a closed form for lasso regression, and thus cannot conclude it is a linear model

3.f

the lemma does not hold for the K-NN regression, the K-NN is a linear model but the second condition does not hold.

$$N_K(x_m) := \text{the K neighbours of } x_m$$

$$N'_K(x_m) := \text{the K-1 neighbours of } x_m \text{(that is } N_K(x_m) \text{whitout } x_m)$$

$$\hat{\tilde{y}}_m = \sum_{x_j \in N_K(x_m)} \frac{1}{K}\tilde{y}_j = \frac{1}{K}[(\sum_{x_j \in N'_K(x_m)} y_j) + \hat{y}_m^{(-m)}]$$

$$\hat{y}_m^{(-m)} = \sum_{x_j \in N'_{K+1}(x_m)} \frac{1}{K}y_j = \frac{\hat{y}_m^{(-m)} - \hat{y}_m^{(-m)}}{K} + \frac{1}{K}\sum_{x_j \in N'_{K+1}(x_m)} y_j = \frac{\hat{y}_m^{(-m)} - \hat{y}_m^{(-m)}}{K} + \frac{y_l}{K} + \frac{1}{K}\sum_{x_j \in N'_K(x_m)} y_j =$$

$$= [(\sum_{x_j \in N'_K(x_m)} y_j) + \hat{y}_m^{(-m)}] + \frac{y_l - \hat{y}_m^{(-m)}}{K} = \hat{\tilde{y}}_m + \frac{y_l - \hat{y}_m^{(-m)}}{K}$$

$y_l$is the K+1 neighbour

thus we get that for all m: $\hat{y}_m^{(-m)} = \hat{\tilde{y}}_m \iff y_l = \hat{y}_m^{(-m)} \iff y$ is a vector of the same values

thus we can conclude that