EXAM_SL

roi hezkiyahu - 205884018

2023-01-23

imports

```
library(dplyr)
library(ggplot2)
library(purrr)
library(caret)
library(class)
library(MASS)
library(tidyverse)
library(nnet)
library(keras)
library(glmnet)
library(reticulate)
library(ncvreg)
library(e1071)
```

Q1

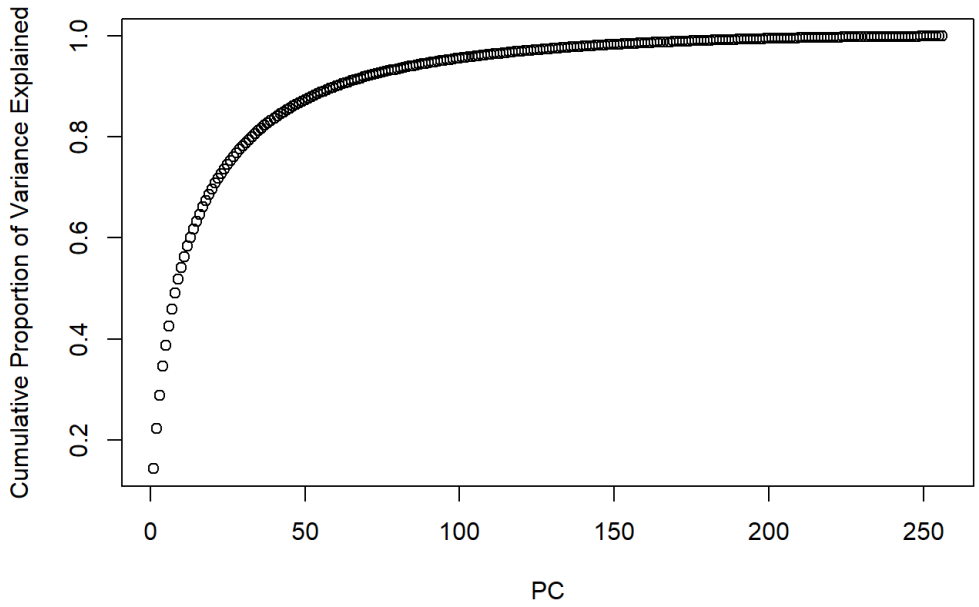
loading the data:

```
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

```
pca <- prcomp(X_train)
plot(cumsum(pca$sdev^2/sum(pca$sdev^2)),
     xlab = "PC",
     ylab = "Cumulative Proportion of Variance Explained")
```



i will take 100 PCS which explain ~ 95% 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])
```

building the models:

```
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)
```

getting results:

```
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 gets a better result on the train set.

lets take a look at the confusion 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 struggles more to distinguish between 5 and 3, after that it has a hard time with 3 and 2, and can separate well enough 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

building the models:

```
multinom_full = multinom(y~., df_train, maxit = 250)
```

```
## # 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
## iter 110 value 0.026716
## iter 120 value 0.011342
## iter 130 value 0.004856
## iter 140 value 0.002520
## iter 150 value 0.001262
## iter 160 value 0.000841
## iter 170 value 0.000667
## iter 180 value 0.000217
## iter 190 value 0.000180
## iter 200 value 0.000159
## final  value 0.000096
## converged
```

```
multinom_pca = multinom(y~., df_pca_train_100, maxit = 250)
```

```
## # 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
## iter 110 value 0.000479
## final  value 0.000068
## converged
```

getting results:

```
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           0.160
## 3 multinom with pca train 0
## 4 multinom with pca test  0.0973
```

clearly we overfit the training 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 that 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 confusion matrix of the PCA multi-class logistic regression the train has perfect score thus the confusion matrix is diagonal and not that interesting, lets see the test confusion matrix:

```
##           Reference
## Prediction   2   3   5
##          2 183  11   4
##          3   6 145  15
##          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 re encode 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 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 = 32, activation = "relu") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 3, activation = "softmax")
```

compile and train the model

```
cnn_model %>% compile(loss = 'sparse_categorical_crossentropy',optimizer = optimizer_adam(learning_rate = 0.0001),metrics =
c('accuracy'))
cnn_model %>% fit(X_train_net, y_train_net, batch_size = 64,epochs = 150,verbose = 1,
                  class_weight = list("0" = 1.5, "1" = 0.7, "2" = 1))
```

getting the results:

```
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  0.0401
```

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) to get even better results

lets take a look at the test confusion matrix

```
##           Reference
## Prediction   2   3   5
##          2 196   1   1
##          3   3 151  12
##          5   1   3 156
```

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

lets improve the model by using some more modern techniques like data augmentation data augmentation "creates" new images by applying several transformations on the image and thus improves the generalization of the model by introducing new observations.

in this case i choose to apply:

rotation - rotates the images by a certain degree

width, height shifts - which applies translation on the image to left/right, up/down

zoom - which zooms in/ out on an image

all of this transformation are rather reasonable by the sense that applying them won't change the image much but they are still valid handwritten numbers. it doesn't really matter if you draw a number in the middle of the box or a bit shifted up/down/left/right, also the scale does not matter, it's the same if you draw a small number or a larger number, and also the same applies for rotation

```
datagen <- image_data_generator(rotation_range = 10,width_shift_range = 0.1,height_shift_range = 0.1,zoom_range = 0.1)

train_generator <- flow_images_from_data(X_train_net,y_train_net,datagen,32)

cnn_model_aug <- 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 = 16, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_batch_normalization() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dropout(rate = 0.1) %>%
  layer_dense(units = 3, activation = "softmax")
```

compile and train

```
cnn_model_aug %>% compile(loss = 'sparse_categorical_crossentropy',optimizer = optimizer_adam(learning_rate = 0.00005),metr
ics = c('accuracy'))

cnn_model_aug %>% fit(train_generator, batch_size = 32,epochs = 200,verbose = 1,
                  class_weight = list("0" = 1.25, "1" = 0.7, "2" = 0.85))
```

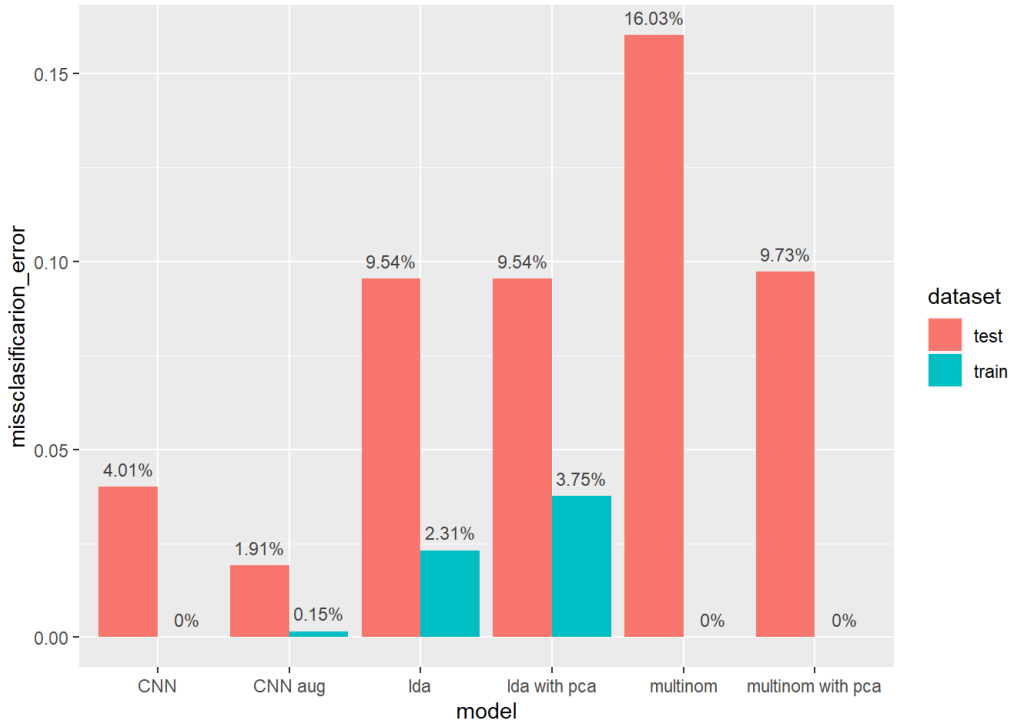
get results

```
cnn_aug_train_res = get_results(cnn_model_aug, X_train_net, y_train, FALSE, TRUE)
cnn_aug_test_res = get_results(cnn_model_aug, X_test_net, y_test, FALSE, TRUE)
tibble(model_dataset = c("CNN aug train", "CNN aug test"),
       err = c(cnn_aug_train_res$classification_err, cnn_aug_test_res$classification_err))
```

```
## # A tibble: 2 × 2
##   model_dataset     err
##   <chr>           <dbl>
## 1 CNN aug train 0.00154
## 2 CNN aug test  0.0191
```

the over all results:

```
## # A tibble: 6 × 3
##   model_dataset     err_train err_test
##   <chr>                 <dbl>    <dbl>
## 1 neural net aug      0.00154   0.0191
## 2 neural net          0         0.0401
## 3 lda                 0.0231    0.0954
## 4 lda with pca        0.0375    0.0954
## 5 multinom with pca   0         0.0973
## 6 multinom            0         0.160
```



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 with augmentation has the best results on the test set, and even better results on the test 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, they should work well for regression and classification.

bad models can be: Linear regression - LR will probably preform not that good as it can't handle missing values well. also K-nn wont work that well for both classification and regression as it cannot handle missing values, also the nature of the problem is in high dimension and we know that the meaning of distance in high dimension is impaired.

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.

*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 must have prediction from all the trees and usually we want a lot of trees to be accurate so the total run time will be rather long. K-nn will also not be that good here as for each new observation we need to calculate the distance to all points in the train set in order to find the K closest ones, and also it will not be good from the reason listed above.

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 increasing the dimension.

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 coefficients. We can use lasso as a model, but also as a feature selection technique and then use all of the other models 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 has no inverse. 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

$$\text{for clearer writing i will use the notation k instead of } i_0$$

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

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

$$\text{lets break down the expression above}$$

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

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

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

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

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

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

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:

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

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

3.d

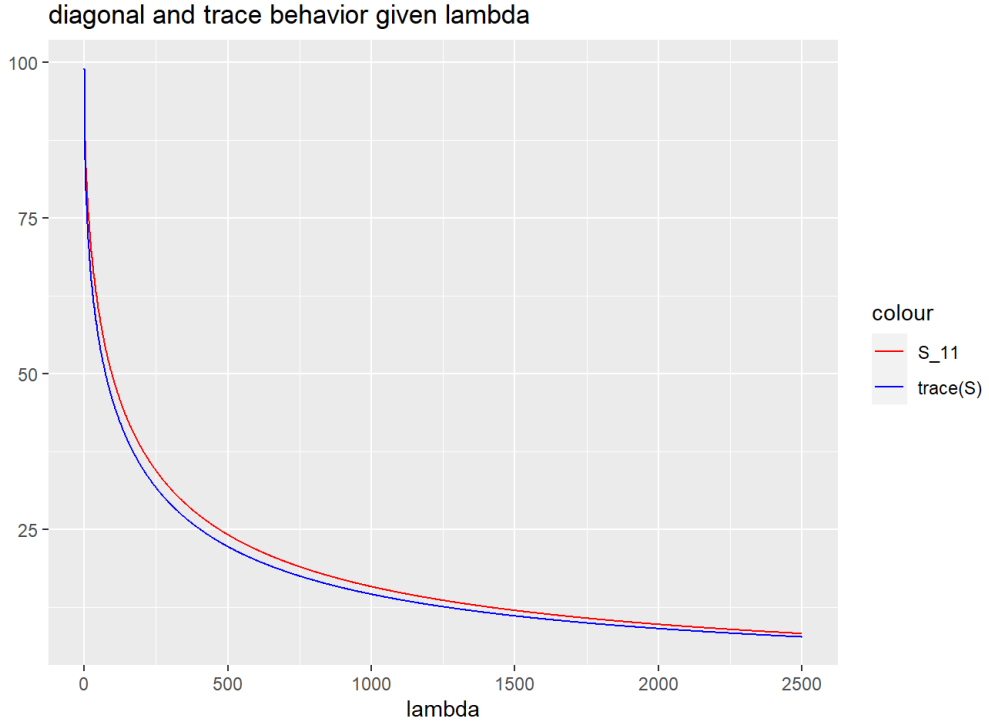
i will show it formally, by differentiating the expression

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

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

and lets also show by simulation:

```
con <- url("http://www.tau.ac.il/~saharon/StatsLearn2022/train_ratings_all.dat")
X <- tibble(read.table(con))
con <- url("http://www.tau.ac.il/~saharon/StatsLearn2022/train_y_rating.dat")
y <- read.table(con)
X = as.matrix(X)[1:100,1:99]
S_lambda = function(X, lambda){
  return(X %*% solve(t(X) %*% X + lambda * diag(1,99,99)) %*% t(X))
}
lambdas = seq(0,2500,1)
s_11 = c()
trace_s = c()
for (lambda in lambdas){
  S_mat = S_lambda(X, lambda)
  s_11 = c(s_11, S_mat[1,1])
  trace_s = c(trace_s, sum(diag(S_mat)))
}
```



S_{11} is scaled *100 so that we will see the relation.

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.

$N_K(x_m) :=$ the K neighbours of x_m

$N'_K(x_m) :=$ the K-1 neighbours of x_m(that is $N_K(x_m)$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 if y has more then one unique value the K-NN model does not supply the 2nd condition

Q4

4.a

X is of full rank, a matrix of p*n matrix is bounded by: $rank(X) \leq min(n,p)$ thus in our setup where $n \leq p, rank(X) = n$

from an algebraic view we know that for a system of linear equations with n equations and p variables where: p>n, we have an infinite number of solutions. thus clearly the solution is not unique. (in our case we have p+1 variables including the intercept)

this conclusion does not change with the loss. if $\sum_i (y_i - X_i^t\beta - \beta_0)^2 = 0$ then $y_i - X_i^t\beta - \beta_0 = 0 \; \forall i$ also if $\sum_i |y_i - X_i^t\beta - \beta_0| = 0$ then $y_i - X_i^t\beta - \beta_0 = 0 \; \forall i \Rightarrow$ we have an infinite number of solutions

for the quantile loss:

$\sum_i \tau(y_i - X_i^t\beta - \beta_0)I(y_i > X_i^t\beta - \beta_0) - (1-\tau)(y_i - X_i^t\beta - \beta_0)I(y_i \leq X_i^t\beta - \beta_0) = 0$ notice that:
$\tau(y_i - X_i^t\beta - \beta_0)I(y_i > X_i^t\beta - \beta_0) \geq 0$, $-(1-\tau)(y_i - X_i^t\beta - \beta_0)I(y_i \leq X_i^t\beta - \beta_0) \geq 0$ thus we sum over non-negative number so in order for the sum to be zero each element need to be zero so we are left with: $y_i - X_i^t\beta - \beta_0 = 0 \; \forall i \Rightarrow$ we have an infinite number of solutions

to conclude all the above losses have an infinite number of solutions in the case were $p \geq n$

4.b

assume in contradiction that we do not have an interpolation solution

$$L = \sum_{i=1}^{n}(y_i - X_i^t\beta - \beta_0)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

$$\forall j \geq 1 : \frac{\partial L}{\partial \beta_j} = 2\sum_i (y_i - X_i^t\beta - \beta_0)X_{ij}^t + 2\beta_j$$

now let $\tilde{\beta}$ be an interpulation solution and plug it in the derevetive above we get:

$$\frac{\partial L}{\partial \beta_j}(\tilde{\beta}) = 0 + \tilde{\beta}_j = 0 \iff \tilde{\beta}_j = 0$$

thus we have that for any interpolation solution $\tilde{\beta}_j = 0 \; \forall j \geq 1, \text{ thus } y_i - X_i^t\tilde{\beta} - \tilde{\beta}_0 = y_i - \tilde{\beta}_0$

but this is an interpolation solution thus $y_i = \tilde{\beta}_0 \forall i \Rightarrow$ y is a vector of constants
thus if y is not a vector of constants we can see that there are no interpolation solutions that minimze L
thus the optimal solution is not an interpolation point

4.c

$$\text{let } \tilde{\beta} \text{ be an optimal solution to } min \sum_{j=1}^{p} \beta_j^2 \; s.t \sum_{i=1}^{n} (y_i - X_i^t \beta - \beta_0)^2 = 0$$

$$\text{the objective of this minimization problem is the L2 norm squared } \sum_{j=1}^{p} \beta_j^2 = ||\beta||_2^2,$$

$$\text{so the } \beta \text{ that minimized it also minimize the L2 norm}$$

$$L(\tilde{\beta}) = \sum_{i=1}^{n} (y_i - X_i^t \tilde{\beta} - \tilde{\beta}_0)^2 + \lambda \sum_{j=1}^{p} \tilde{\beta}_j^2 = 0 + \lambda ||\tilde{\beta}||_2^2 \leq \lambda ||\beta||_2^2 \quad \forall \beta \; s.t \sum_{i=1}^{n} (y_i - X_i^t \beta - \beta_0)^2 = 0$$

$$\text{thus } \tilde{\beta} \text{ is the optimal solution to the ridge problem over all interpolation points}$$

4.d

$$\forall \lambda > 0 \text{ let } \hat{\beta}(\lambda) = argmin_\beta \sum_{i=1}^{n} (y_i - X_i^t \beta - \beta_0)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

$$\text{from the previous results we know:}$$

$$1. \; \hat{\beta}(\lambda) \text{ is unique}$$

$$2. \; \hat{\beta}^{(l2)} \text{ has the minimal norm over all the interpolation points and also has the smalleset ridge penelty}$$

$$\text{as } \lambda \text{ converge to 0: } \hat{\beta}(\lambda) \text{ converges to some } argmin_\beta \sum_{i=1}^{n} (y_i - X_i^t \beta - \beta_0)^2$$

$$\text{meaning that } \hat{\beta}(0) \text{ is an interpolation point}$$

$$\text{so exsists } \varepsilon \text{ close to zero such that for each } \lambda < \varepsilon \text{ exsits an interpolation solution}$$

$$\text{thus for such } \lambda, \hat{\beta}(\lambda) \text{ is an interpolation point, and due to the optimality it also has the minimal norm } \Rightarrow$$

$$\Rightarrow ||\hat{\beta}^{(l2)}||_2 = ||\hat{\beta}(\lambda)||_2 \; \forall \lambda < \varepsilon$$

$$\text{recall that } \hat{\beta}(\lambda) \text{ is a unique solution thus there is only one vector that is an interpolation point and satisfies: } ||\hat{\beta}^{(l2)}||_2 = ||\hat{\beta}(\lambda)||_2$$

$$\text{thus we get } \hat{\beta}(\lambda) \xrightarrow{\lambda \to 0} \hat{\beta}^{(l2)}$$

4.e

$$minimize \beta^t \beta$$

$$s.t: X\beta = y$$

$$\text{using lagrange multiplers we get: } L = \beta^t \beta + \lambda^T (X\beta - y)$$

$$\frac{\partial L}{\partial \beta} = 2\beta + X^T \lambda = 0 \iff \beta = -\frac{1}{2} X^T \lambda$$

$$\frac{\partial L}{\partial \lambda} = X\beta - y = -X\frac{1}{2} X^T \lambda - y = 0 \iff \lambda = -2(XX^t)^{-1} y$$

$$\text{plug it back in to } \beta \text{ and we get: } \hat{\beta} = X^T (XX^t)^{-1} y$$

4.f

$$\text{in this case: } \hat{\beta}(\lambda) = argmin_\beta \sum_{i=1}^{n} (y_i - X_i^t \beta - \beta_0)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

$$\text{and } \hat{\beta}^{(l1)} = argmin \sum_{j=1}^{p} |\beta_j| \; s.t \sum_{i=1}^{n} (y_i - X_i^t \beta - \beta_0)^2 = 0$$

$$\text{in this case the result will be that } \hat{\beta}(\lambda) \xrightarrow{\lambda \to 0} \hat{\beta}^{(l1)}$$

$$\text{and we would expect the results to not chnage according to the loss by the same arguments in section 4.a}$$

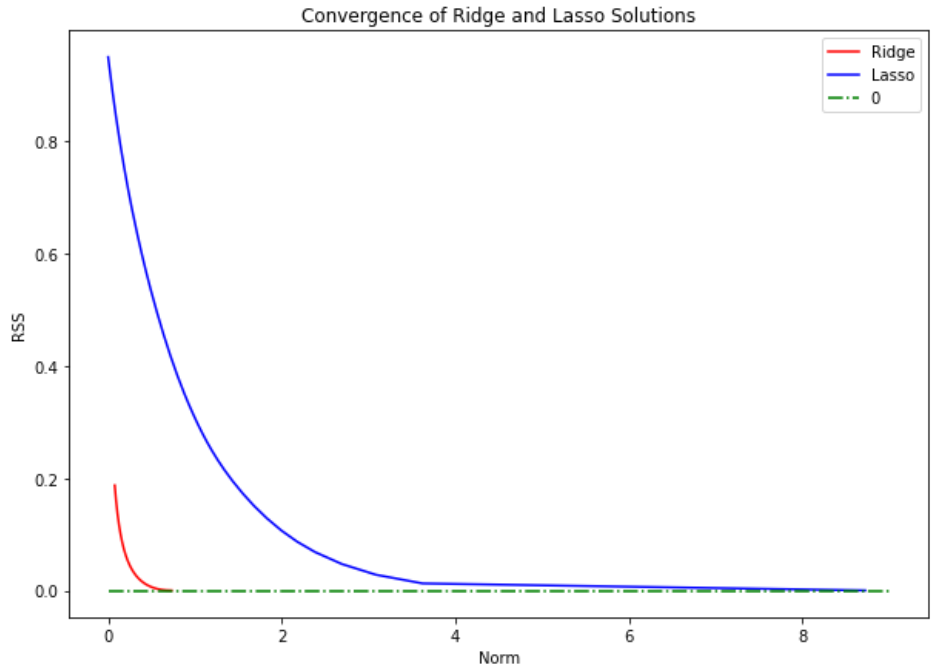
$$\text{in general for L1 loss, L2 loss and qunatile loss the constraints are equivelent to } y_i - X_i^t \beta - \beta_0 = 0 \quad \forall i$$

4.g

4.g.i

for some reason i cant load the l1ce package and becuase of the suggestion to not use glmnet, i will use python instead.

```
import pandas as pd
import requests
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt

con = requests.get("http://www.tau.ac.il/~saharon/StatsLearn2022/train_ratings_all.dat")
X = pd.read_csv(con.url, sep='\t', header=None)
con = requests.get("http://www.tau.ac.il/~saharon/StatsLearn2022/train_y_rating.dat")
y = pd.read_csv(con.url, sep='\t', header=None)

X_small = X.iloc[:50,:]
y_small = y.iloc[:50,:]

alphas = np.linspace(0, 100, num=1000)

ridge_rss = []
ridge_norm = []
for alpha in alphas:
    ridge = Ridge(alpha=alpha)
    ridge.fit(X_small, y_small)
    ridge_rss.append(mean_squared_error(y_small, ridge.predict(X_small)))
    ridge_norm.append(np.sum(np.square(ridge.coef_)))

lasso_rss = []
lasso_norm = []

for alpha in alphas:
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_small, y_small)
    lasso_rss.append(mean_squared_error(y_small, lasso.predict(X_small)))
    lasso_norm.append(np.sum(np.abs(lasso.coef_)))

plt.figure()
plt.plot(ridge_norm, ridge_rss, 'r', label='Ridge')
plt.plot(lasso_norm, lasso_rss, 'b', label='Lasso')
plt.plot([0,9], [0,0],'-.', color = 'g', label='0')
plt.xlabel('Norm')
plt.ylabel('RSS')
plt.title('Convergence of Ridge and Lasso Solutions')
plt.legend(loc='upper right')
plt.show()
```



we can see that as λ increaces the norm increaces and the RSS decreases to 0 (an interpolation solution)

4.g.ii

```
ridge = Ridge(alpha=np.exp(-15))
ridge.fit(X_small, y_small)
ridge_rss = mean_squared_error(y_small, ridge.predict(X_small))
ridge_norm = np.sum(ridge.coef_**2)

lasso = Lasso(alpha=np.exp(-15))
lasso.fit(X_small, y_small)
lasso_rss = mean_squared_error(y_small, lasso.predict(X_small))
lasso_norm = np.sum(abs(lasso.coef_))
```

```
## [1] "lasso norm is:  8.7191 ridge norm is:  0.7302"
```

```
## [1] "lasso rss is:  0 ridge rss is:  0"
```

we can see that indeed both models end up in an interpolation point. and also that the ridge norm is alot lower than the lasso norm. ridge usually shrinks the coefficients while lasso zeros some of them out, also most coefficients are smaller then 1 so squaring them makes them smaller, this is why we can see that the ridge solution has a lower norm (L2) then the lasso norm (L1)

4.h

in regression we said that the model interpolate the data if it can fit each value of y correctly, in the classification scenario it is the same, we mange to classify each of the train observation to the appropriate class.

in the multi class case we classify an observation to class i if $P(y_j = i|x_j, \beta)$ is the maximum over the set $\{p(y_j = k|x_j, \beta)\}_{k=1}^K$, where K is the number of classes

denote $\hat{y}_k(\beta) = argmax_k\{p(y_j = k|x_j, \beta)\}_{k=1}^K$

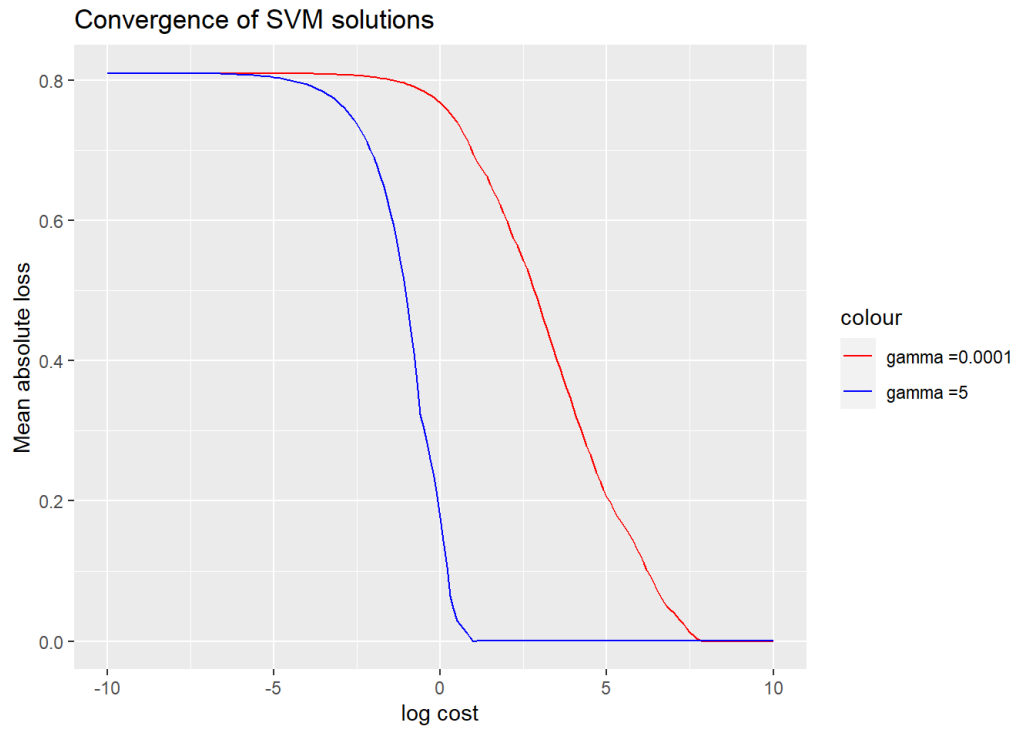
the corresponding minimum norm interpolation would be: $argmin \sum_{j=1}^p \beta_j^2 \; s.t \sum_{i=1}^n (1 - I(y_i = \hat{y}_k(\beta))) = 0$

4.i

4.i.i

we saw that the SVM loss is $L(y, \hat{y}) = (|y - \hat{y}| - \varepsilon)_+$ so in the case that $\varepsilon = 0$ the loss becomes: $(|y - \hat{y}|)_+ = |y - \hat{y}|$ which is the absolute loss

```
X_100 = as.matrix(X[1:100,])
y_100 = y[1:100,]
svm_small_gamma = svm(X_100,y_100, type="eps-regression", epsilon = 0, gamma =0.0001)
svm_large_gamma = svm(X_100,y_100, type="eps-regression", epsilon = 0, gamma =5)
cost_vals  = exp(seq(-10,10,0.1))
err_small = c()
err_large = c()
for (cost in cost_vals){
  svm_small_gamma = svm(X_100,y_100, type="eps-regression", epsilon = 0, gamma =0.0001, cost=cost)
  svm_large_gamma = svm(X_100,y_100, type="eps-regression", epsilon = 0, gamma =5, cost=cost)
  err_small = c(err_small,mean(abs(y_100 - predict(svm_small_gamma,X_100))))
  err_large = c(err_large,mean(abs(y_100 - predict(svm_large_gamma,X_100))))
}

ggplot() +
  geom_line(aes(x = log(cost_vals), y = err_small, color = "gamma =0.0001")) +
  geom_line(aes(x = log(cost_vals), y = err_large, color = "gamma =5")) +
  scale_color_manual(values = c("gamma =0.0001" = "red", "gamma =5" = "blue")) +
  xlab("log cost") +
  ylab("Mean absolute loss") +
  ggtitle("Convergence of SVM solutions")
```



4.i.ii

for a kernel $K(x,y) =< h(x), h(y) >= \sum_{j=1}^{q} h_j(x)h_j(y)$ we defined $f = \sum_j \beta_j h_j$, also we know that: $||f||^2_{HK} = \sum_j \beta_j^2$

the kernel is a function of γ, in our case $K_\gamma(x,y) = exp(-||x-y||^2/(2\gamma^2))$, thus for different values of γ we have different kernels, different h functions and different β_j for each h_j. so if $\gamma \neq \gamma'$ we have $||f_\gamma||^2_{HK} \neq f_{\gamma'}||^2_{HK}$, and this norm is the regularization term

4.i.iii

```
fake_x = matrix(rep(5,99),1,99)
colnames(fake_x) = colnames(X_100)
svm_small_gamma = svm(X_100,y_100, type="eps-regression", epsilon = 0, gamma =0.0001, cost=exp(15))
svm_large_gamma = svm(X_100,y_100, type="eps-regression", epsilon = 0, gamma =5, cost=exp(15))
```

```
## [1] "small gamma prediction: 3.36514 the training mean error is: 2.37040608507377e-05"
```

```
## [1] "large gamma prediction: 3.58998 the training mean error is: 2.46564716309861e-05"
```

```
## [1] "the mean value of y is : 3.59"
```

4.i.iv

we can see that the results are rather close, but a person that rates everything 5 would probably rate the target movie 5 as well, so we can see that both models are interpolating

we see that for large γ we get a prediction very close to the mean of our target vector, this is due to the fact that for large gammas the observations are considered closer together and our K matrix becomes almost $\underline{1}^T\underline{1}$ (a matrix full of ones) so each new prediction will get a constant value of the mean of the target

we can easily demonstrate it using a fake observation with all ones:

```
fake_x = matrix(rep(1,99),1,99)
colnames(fake_x) = colnames(X_100)
svm_large_gamma = svm(X_100,y_100, type="eps-regression", epsilon = 0, gamma =5, cost=exp(15))
```

```
## [1] "large gamma prediction:  3.58998"
```

```
## [1] "the mean value of y is :  3.59"
```

for small values of γ our model acts more or less like 1-NN. for very small values of γ we have
$K_\gamma(x,y) = exp(-||x-y||^2/(2\gamma^2)) \approx exp(-\infty)I(x \neq y) + exp(0)I(x=y) = I(x=y)$ so for each observation on the train set we predict the value of the observation itself because all other observation are given a weight close to zero.

this is in the case that γ is very very small, in our case γ is small but not that close to zero so we have a larger neighborhood

lets demonstrate by taking the 5 closest neighbors of our movie lover fake person

```
dist_mat = (X_100 - matrix(rep(5,99*100),100,99))^2
norms = apply(dist_mat,1,sum)
nn_5 = order(norms)[1:5]
mean(y_100[nn_5])
```

```
## [1] 3.4
```

as we can see the result is very close to the prediction of our small γ

4.i.v

as we saw in the case where $\gamma = 5$ the fit at this point (or in general at any other point) is the mean of y (our target vector).

we have a large cost, thus for minimizing the loss function we need to minimize the errors term because the penalty for large coefficient becomes neglect able.

thus for each interior point we have an interpolation, and each exterior point is a weighted sum over our observation, the weights are given by the kernel, and because of the large γ our distances increase and lose their meaning. so for a new observation we have similar weighted distance to all other observation, thus our new observation is assigned with the value of mean(y)