Lab #2 – Dimension Reduction

- Open the file *lab2_packeges.R* and run it. **DO NOT change this file!**
- Open a **new** file and continue lab on the new file.

Penalized Regression

Useful commands and packages:

Command	Explanation
glmnet (The package)	Runs different types of shrinkage methods, also has a cross validation option
glmnet(x, y, alpha, lambda)	Runs ridge (alpha = 0), lasso (alpha = 1), given input matrix (x), response variable (y), and **optional** lambda sequence.
cv.glmnet(x, y, alpha, nfolds, lambda)	Like glmnet, includes k-fold cross validation.
model.matrix(formula = y ~ x, data)	Produces matrix of data, turns qualitative variables into dummies. Comfortable for use with glmnet.
sample(data, size)	Generates random samples of data. Use for creating index when splitting data into train and test.
predict(object, newx, s)	Calculates model predictions on a new dataset (newx) for lambda (s).

Example code for penalized regression:

```
library(glmnet)
load("/.../dat.Rdata")
#create train and test sets
y_ind <- dim(dat)[2]
inds <- sample(x=1:nrow(dat), size=.3*nrow(dat))
test <- dat[inds,]
train <- dat[-inds,]
train_y <- train[,y_ind]
train_x <- train[,-y_ind]
test_y <- test[,y_ind]
test_x <- test[,-y_ind]
#run regression
ridge.mod <- glmnet(x = train_x, y = train_y, alpha=0, standardize=TRUE)
#view regression output and graphs
plot(ridge.mod, xvar = "lambda")
# run cross-validation
cv.ridge.mod <- cv.glmnet(x=train_x, y= train_y, alpha=0, standardize=TRUE, nfolds=5)
# predict using chosen lambda
pred <- predict(ridge.mod, newx= test_x, s=cv.ridge.mod$lambda.min)
# calculate MSE
mean((pred-test_y)^2)
```

Exercise

1. Import library *glmnet* (run *library(package_name)*)
2. Load CA_samp.Rdata , this will be our data for the lab.
   load("path...../CA_samp.Rdata")
   * Hint: you do not need to insert it into a variable, just load the data and it will appear in your environment
   Explanation of the dataset (Not important for the lab!): This data is a sample of the California Housing dataset we worked on in the previous lab. Besides being only a sample of the original dataset, it also has many more variables. The variables were obtained by taking a polynomial (up to the degree of 4) of each of the original variables and by taking all two-way interactions between the variables.
   (i.e. for each variables x we added x^2, x^3, x^4 as variables, and then we took interactions)
3. The data has already been preprocessed, so you can continue on without worries. However, it is good to know that using *glmnet* requires some preprocessing, as described below.
   ** Prepare data: glmnet( ) accepts data in matrix form, with quantitative variables only, and no missing values. Use model.matrix( ) to prepare the data for modelling. Since will be predicting median house value using all available predictors, therefore our formula will be medianHouseValue~. . notice model.matrix( ) creates an intercept column. Get rid of this column before modelling (we do not shrink the intercept!).*
4. Recall we want to predict median house value.
5. What is the size of your data? What will happen if we perform ordinary linear regression on this dataset?
6. Subset your data into train and test. Remember we do not use the test set while choosing our model!
   * Hint: sample a number indexes and select the appropriate rows for the training and test sets using *dat[inds,]* and *dat[-inds,]*
7. Divide the response from the variables in the training and test sets.
   * Hint: the response *medainHouseValue* is the last column in *CA_samp* (you can confirm that by printing *colnames(CA_samp)* and looking at the last one). Use *dim(CA_samp)[2]* to get the number of columns in the dataset (*p*) and select by *train[,p]* and *train[,-p]*
8. Since we have many variables, we want to use penalized regression. In the following section we will compare Ridge and LASSO:
   a. Run Ridge and LASSO models. Define each model by a name that specifies the model type. Notice that the resulting models are *glmnet* objects (run *class(model)*). Make sure that the function standardizes the variables before fitting the model.
      * Set *alpha=0* for Ridge and *alpha=1* for LASSO in *glmnet* function. For standardization set *standardize = TRUE*.
   b. Recall that the Ridge and LASSO are defined by a parameter lambda. Find which lambdas were used in the evaluation of each model.
      (Since we did not supply the function with a chosen lambda, the function "chooses" a sequence of lambdas to use.)

* Hint: the lambdas are an attribute of a *glmnet* object, therefore can be accessed by *model$lambda*.

c. Check how many non-zero coefficients were computed for each lambda. Notice the difference between the Ridge and the LASSO!

   * Hint: this also is an attribute of a *glmnet* object, and therefore can be accessed by *model$df*.

d. Use *plot(model, xvar="lambda")* to view a plot of the coefficients as a function of lambda.

9. The lambda is our "tuning parameter" and therefore should be chosen using cross-validation.
   a. Run *cv.glmnet()* for cross-validation and choose the number of folds for the CV.

      * Remember to choose the appropriate alpha for Ridge and for LASSO

   b. Extract the lambda that minimizes the cross-validation error

      * Hint: it is an attribute of the cv model; try *model$lambda.min* .....

   c. Plot both models (*plot(model)*) – what do these plots show?
   d. Predict the median house value of the test set using the lambda that minimizes the cross-validation error
   e. Calculate test MSE. Which method predicts better?

For Advanced

1. Run Ridge and LASSO without using cross-validation and find the lambda that minimizes the error on the training set. Which lambda was chosen? Was it the biggest or smallest? Why?

   * Hint: Using *predict* on the Ridge/LASSO model evaluates the model for every lambda in the model (*model$lambda*). Iterate over each column (either by a *for* loop or by *apply*), find its train MSE and find which lambda minimizes the MSE.

2. Repeat 1 but now give the *glmnet* function a grid of lambdas that includes 0 (for example: *seq(0,10,length=100)^3*) – which lambda is chosen now? Why? Does this model coincide with a previous model we learned?

PCR Analysis

Useful commands

Command	Explanation
pcr(formula, data, scale, ncomp, validation)	Runs pca regression on an lm() object (formula). Use ncomp to choose specific number of components wanted, or validation = "cv" for cross validation.
validationplot(object)	Plots cross validation MSE as function of number of components.

Example code for PCR:

```
library(pls)
# prepare dataset
train_dat <- data.frame(train_x)
train_dat$y <- train_y
# run regression
pcr.mod <- pcr(y~., data=train_dat, scale=TRUE, ncomp=10, validation="CV")
summary(pcr.mod)
# plot RMSE as a function of the number of components
validationplot(pcr.mod)
# predict
pred <- predict(pcr.mod, newdata = data.frame(test_x), ncomp=6)
# calculate MSE
mean((pred-test_y)^2)
```

Exercise:

1. Import library *pls*
2. Convert the training and test sets to data frames (*data.frame(dat)*)
   * *pcr* function accepts data frames (as opposed to *glmnet* which accepts matrices)
3. Fit a PCR model
   a. Run *pcr* with cross validation. Set the number of principal components it should calculate.
      * Use *validation="CV"* to perform cross-validation, and use *ncomp = #number* to set the number of components to calculate
   b. Print summary of the *pcr* object. Find the number of components with the lowest cross-validation error. How much variance can be explained by that number of components? Notice that there is a tradeoff between the two!
   c. Plot the cross-validation error as a function of the number of components (use *validationplot(model)*)
   d. Use the number of components found in b. to predict the median house value on the test set and find the MSE.
4. Repeat 3 with a much bigger value for *ncomp*.

Elastic Net

In the previous class, we discussed the concept of penalized regression. Penalized regression helps us when the number of observations is small relative to the number of predictors.[1] Another motivation for penalized our regression is to reduce

[1] The concepts of "small" and "large" depend on the context of the problem. There is no exact n_0 that bellow it we have a small n.

overfitting. Increasing the number of predictors, we improve our prediction on the training set (via decreased bias) and increase our variance. In other words, our model fits the training data too well.

In penalized regression we add a regularization term to the loss function, that pushes our covariates toward zero. In Ridge regression, we optimize $\beta \in \mathcal{R}^p$ over the following loss function

$$\min_{\beta} \left|\left|y - X\beta\right|\right|^2 + \lambda_2 ||\beta||^2$$

where $||\beta||^2 = \Sigma_{j=1}^{p} \beta_j^2$ and in Lasso we optimize $\beta \in \mathcal{R}^p$ using the loss function

$$\min_{\beta} ||y - X\beta||^2 + \lambda_1 ||\beta||_1$$

Where $||\beta||_1 = \Sigma_{j=1}^{p} \left|\beta_j\right|$.

Lasso suffers from two significant limitations. If $p > n$, then Lasso selects at most n variables before it saturates. This limitation set bound on our prediction performance. Also, if there is a group of highly correlated variables, then the LASSO tends to select one variable from a group and ignore the others.

In this section, we introduce the Elastic Net regression, which is also a way to penalize our parameters, using a linear combination of Lasso and Ridge regularization terms. In Elastic net, β would be equal to

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}}(\left|\left|y - X\beta\right|\right|^2 + \alpha||\beta||^2 + (1-\alpha)||\beta||_1)$$

where $\alpha = \frac{\lambda_2}{\lambda_1 + \lambda_2}$. Pay attention that if $\alpha = 0$ we are using a Ridge regression and if $\alpha = 1$ we are using a Lasso regression.

1. We are going to use CA_samp.Rdata again. Split is to train (80%) and test (20%) sets.
2. Are the datasets similar? For each variable in our dataset, present the mean in a table as follows

	Train	Test

Variable 1	mean(Variable 1)	mean(Variable 1)
Variable 2	…	…
Variable 3		
…		

Usually, we put our test set aside, train the model, tune hyper-parameters, and only at the very end we check our performance on the test set.

3. Train an Elastic Net model on our data, recall that we want to forecast the median house value. As before, use the k folds method using cv.glmnet.
   - Set k=5
   - cv.glmnet gets α as an input. For a given α, it produces a vector of λs. For each λ it computes the mean cross validation error. We would look for the minimal mean cross validation error.
   - The mean cross validation vector is saved under "cvm". Say we fitted a cv.glmnet model and kept it in an object called A, then, the minimal mse would be $mse = \min(A\$cvm)$.
   - Run over 10 different values of $\alpha \in [0,1]$. For each α:
     - Fit an Elastic Net with cross validation
     - Keep the best cross validation error
     - Predict on the test set using s = A$lambda.min. specifically, use `predict(A, s = A$lambda.min, newx = Xtest)`
   - Present your results in a table as the following

α	Cross Validation MSE	Test MSE
0		
0.01		
…		