Lab 3: Cross Validation

First Name:Bill **Last Name**:Zhou **NetID**:wz325

Lab 3 is due Feb 25th (Friday) by 11:59pm on Gradescope. Submit your answers as one .pdf file, including all R code that you had to write to obtain your answers. Regrade requests should be submitted via Gradescope.

In this lab, we will learn to apply Cross-Validation techniques to select the degrees of polynomials in regression. We will consider the validation set approach, LOOCV and k-fold CV, and apply them to the ToyotaCorolla data set. Some of the commands in this lab may take a while to run in your computer.

Download the `ToyotaCorolla.csv` data set from Canvas. Our goal is to use a linear regression model with high order terms of different degrees to predict the price of a vehicle for resell in the market using the variables Age and KM, and then find the appropriate order of the polynomial of these variables for our model.

Before analyzing the data, do the same procedures to clean it as we did previously, i.e.

1. Delete all other columns except for `Age_08_04` and `KM`, which we will be using.
2. Check if there is any missing data. If so, deal with it.
3. Check the classes of the variables.

The Validation Set Approach

When we think we have enough samples in our dataset, we can split the data into training / validation / test set. Recall that the validation set is "revealed" to our evaluation pipeline, while the test set is only used at the very end (often as a sanity check for our evaluation procedure). Here, we pick half of the observations to be our training data. Moreover, we set the random seed to 1 for reproducibility.

```
> set.seed(1)
> train_id = sample(nrow(corollas), nrow(corollas)/2)
```

Then we fit a linear regression model using only the data points in the training set.

```
> lm.fit = lm(Price~Age_08_04+KM, data=corollas, subset=train_id)
```

Now we can calculate the MSE over the validation set. Write the code below to do that.

```
corollas_val=corollas[-train_id,]
MSE1=mean((test-corollas_val$Price)^2)
```

Our next step is to fit models with high-order terms of the predictors, making use of the `poly()` function. Write a function `eval_lm()`, accepting an argument `deg`, which fits a model with terms of degree `deg` of the predictors, and returns the MSE on the complement of the training set.

```
eval_lm=function(data,deg)
{
  lm0.fit=lm(Price~poly(Age_08_04,deg)+poly(KM,deg), data=data, subset=train_id)
  MSE0=mean((Price-predict(lm0.fit,corollas))[-train_id]^2)
  return(MSE0)
}
```

Calculate the MSE for regressions models with degree up to 5. Note, this may take a while for R to run. What do you observe?

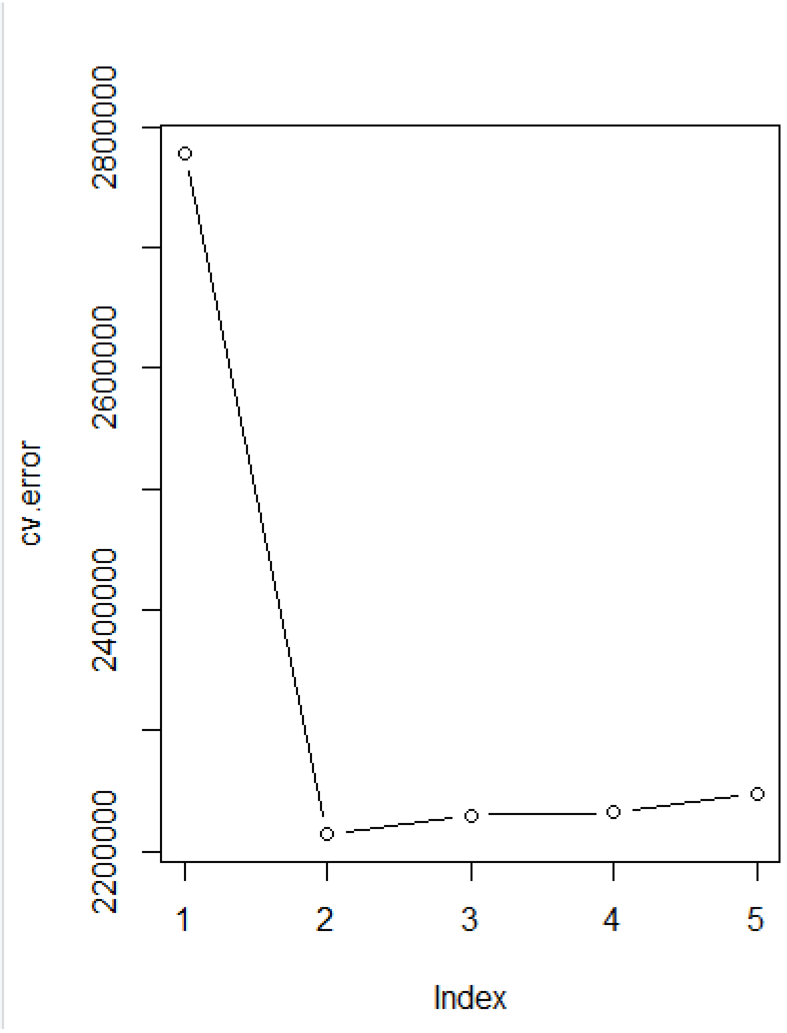
```
MSE1=2436088
MSE2=1965966
MSE3=1994139
MSE4=2003634
MSE5=2001002
As we can see, as the deg increase, the MSE are not monotonically decreasing
Actually quadratic model has the best performance for corollas dataset
```

Leave-One-Out Cross-Validation

We then consider Leave-One-Out Cross-Validation (LOOCV). The LOOCV estimate can be automatically computed for any generalized linear model using the `glm()` and `cv.glm()` functions. As previously, we fit and validate models of degree up to 5.

```
> cv.error=rep(0,5)
> for (i in 1:5) {
+   glm.fit = glm(Price~poly(Age_08_04, i)+poly(KM,i), data = corollas)
+   cv.error[i] = cv.glm(corollas, glm.fit)$delta[1]
+ }
> plot(cv.error, type="b")
```

According to the LOOCV estimates, which model do you recommend? Why?



```
According to the plot above, we can see that deg=2 has the best performance for LOOCV validatic
```

Check p-values in the summary for the cubic model. Describe your observations and compare them with your previous findings.

```
poly(Age_08_04, 3)1       NA
poly(Age_08_04, 3)2  < 2e-16 ***
poly(Age_08_04, 3)3   0.0591 .
poly(KM, 3)1              NA
poly(KM, 3)2          0.1185
poly(KM, 3)3          0.8044
overall P:2.2e-16
The result above shows that for terms which has p>0.05(Age_08_04^3,KM^2,KM^3)
they should berule out in our model
Since all the cubic terms has p>0.05, we can just take i=2 for our lm and
```

```
ignore cubic terms
```

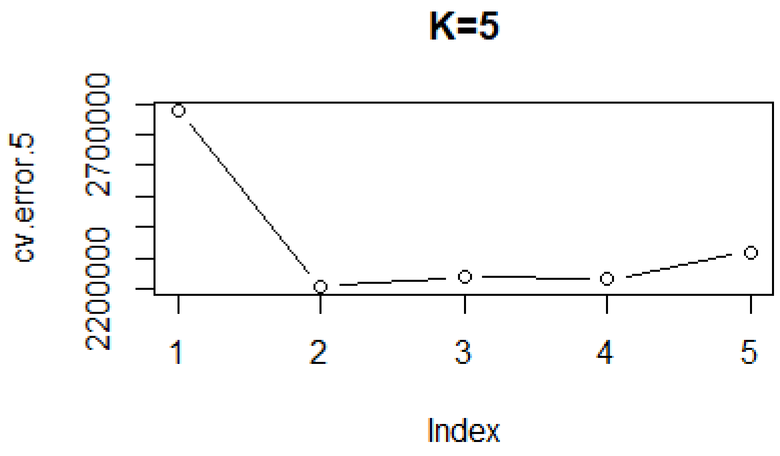
k-fold Cross-Validation

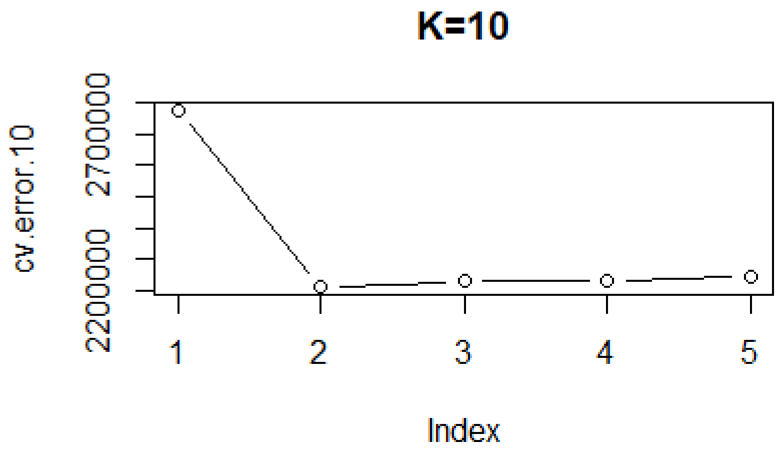
The validation set approach and LOOCV can be thought of a two "extremes" of cross-validation. The method of k-fold cross-validation aims to interpolate between the two. The `cv.glm()` function can also perform k-fold CV (type `?cv.glm` for help with the syntax). We will consider polynomials of degrees one to five, and then plot the corresponding cross-validation errors for $k \in \{5, 10\}$. Write a script that computes the cross-validation error for all configurations of parameters, and produces an appropriate plot, and attach the generated plot.

```
cv.error.5 <- rep (0, 5)
for (i in 1:5) {
  glm.fit <- glm (Price ~ +poly(KM,i)+poly (Age_08_04 , i), data = corollas)
  cv.error.5[i] <- cv.glm (corollas , glm.fit , K = 5)$delta[1]
}

cv.error.10 <- rep (0, 5)

for (i in 1:5) {
  glm.fit <- glm (Price ~ +poly(KM,i)+poly (Age_08_04 , i), data = corollas)
    cv.error.10[i] <- cv.glm (corollas , glm.fit , K = 10)$delta[1]
    }
```





Report your findings. Do the cases $K = 5$ and $K = 10$ show different behavior? How do they compare with LOOCV?

```
Generally plot for K=5& K=10 looks the same, the only difference
we can observe is that when K=5,case deg=5 has a relative larger MSE
AS for comparison to LOOCV,CV=10 has almost identical result as LOOCV
```

Report and compare the running time of the three CV methods. You can use the following command to record running time. Which is / should be the fastest, and why?

```
> ptm = proc.time()
> # INSERT YOUR CODE HERE
> proc.time() - ptm
> # CHECK elapsed
```

```
LOOCV:
   user  system elapsed
  27.20    1.14   27.93
CV K=5:
   user  system elapsed
    0.1     0.0     0.1
CV K=10:
   user  system elapsed
   0.18    0.02    0.19
K=5 is absolutely the quickest,because we only need to fit 5 different
models and cross validate them
K=10 then we need 10 models
LOOCV we need number of data points models
```

Without running any `R` code: do different runs of the validation test approach produce different estimates? How about LOOCV and k-fold CV? Why?

```
Maybe,
The answer is no (estimation will always be the same)
if we have already sliced the whole data set to K chunks ,then
applied to CV, the answer is no, every run through is just the repeat
for the previous ones
The extreme case is K=num_of_datapoints(LOOCV) which well always return us
the same result
However, if we just give a value K, and every time our
CV algorithm random sample the Train and Val sets, then the answer is Yes
(estimation will change)
because different way of slicing the train/val data set will result
in different fitting result, correspondingly influence the estimates
```

Take Home Questions

Part 1. The bias-variance tradeoff

If we use higher degree polynomials, how does it affect the bias, variance and test error of the model? Explain your answer.

```
Higer degree polynomials means more complexity, which will
decrease bias, increase variance
As for testing error, it will go down first, then go up if our
model is in the regime of over fitting
```

Which unknown quantity are the aforementioned approaches trying to estimate?

```
Variance
```

Consider a case where the "true model" describing some phenomenon is: $y = f(x) + \epsilon$, where $f(x) = x$ and ϵ is the irreducible noise with distribution $\epsilon \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1)$. Our training set contains two data points, $x = -1, x = 1$ and corresponding y's generated from the true model above.
We want to predict y when $x = 2$. What would the expected MSE of our prediction be if we chose $\hat{f}$ to be the constant function 0?
Hint: $\mathbb{E}\left[\left(\hat{f}(x) - y\right)^2\right] = \text{Var}\left(\hat{f}(x)\right) + \text{Var}(\epsilon) + \left(\mathbb{E}\left[\hat{f}(x) - f(x)\right]\right)^2$

```
Var(f(x))=0
Var(ε)=1
E[f'(x)−f(x)]^2=1
Thus total MSE=2 in this case
```

Let TE* be your desired test error threshold. After developing your model and calculating the training and test errors, you notice that your training error is much lower than your test error. You also notice that your training error is lower than TE* and your test error is higher than TE*. Explain why you are seeing these errors and what steps you would take to remedy/improve your model.

```
Training error is relatively low, however testing error is relatively high
This indicates that our model has high variance thus we need to prevent
overfiting, by either reducing model complexity
Or use' averaged' model to reduce model's variance
```

In this scenario, you notice that your test error and training error are very close to each other but both are still significantly above TE^*. Explain why you might see this situation and what steps you would take to remedy/improve your model.

```
Both Training error and test error are too high
This indicates that our model has high bias thus we need to prevent
underfiting,by increasing the model complexity
```

What would happen if the number of features are significantly greater than the number of observations (i.e. $p \gg n$)? How would the training error and test error behave?

```
If num of feature is even more than number of data points, then obviously
we are expecting to have overfiting as result
we will have ideally 0 as training error, and unpredictable testing error
(could be low or high)
```

We will now examine the bias-variance tradeoff in practice using R's `cars` dataset. This can be accessed by typing '`cars`'. If you see an error, try importing the `datasets` package by `library(datasets)`.
We will develop 3 different models : no - intercept model, intercept model, and non-linear model. Consider the following procedure:

1. Set the seed to 2022 with set.seed(2022) so we can create reproducible results for each model
2. Split the cars data into 2/3 of training data and the rest to test data
3. Create your model (response variable = `dist`, features = `speed`)
4. Use the test set to predict using your model
5. Calculate the training MSE
6. Calculate the test MSE

You will need to write your own R code and submit.
What is the training MSE and test MSE of the no intercept linear regression model?

```
Training MSE:270
Testing MSE:236
```

What is the training MSE and test MSE of the intercept linear regression model?

```
Training MSE:221
Testing MSE:243
```

What is the training MSE and test MSE of the quadratic (non-linear) model?
Hint: use poly() as you did in first part of this lab

```
Training MSE:188
Testing MSE:312
```

```
Code for this part

#importing /sampling data
library(datasets)
```

```
cars=cars
set.seed(2022)
train_id=sample(nrow(cars),nrow(cars)*2/3)
#model 1: no intercept
lm1.fit=lm(dist~speed+0,data=cars,subset=train_id)
attach(cars)
MSE1_test=mean((dist-predict(lm1.fit,cars))[-train_id]^2)
MSE1_train=mean((dist-predict(lm1.fit,cars))[train_id]^2)
#model 2:lm with intercept
lm2.fit=lm(dist~speed,data=cars,subset=train_id)
attach(cars)
MSE2_test=mean((dist-predict(lm2.fit,cars))[-train_id]^2)
MSE2_train=mean((dist-predict(lm2.fit,cars))[train_id]^2)
#model 3: non linear model
lm3.fit=lm(dist~poly(speed,2),data=cars,subset=train_id)
attach(cars)
MSE3_test=mean((dist-predict(lm3.fit,cars))[-train_id]^2)
MSE3_train=mean((dist-predict(lm3.fit,cars))[train_id]^2)
```

Repeat the above procedure a few times using a different random seed. What do you notice about the training errors and test errors of the different models? Can you explain your observations?

```
I set seed as 8964/97376 and both return results that training MSEs are smaller
than testing MSEs for all three models
Besides, usually if we rank the performance of all three models( based on how
low the MSE could be) non-linear one usually gives us a best result
As for the comparison between different seed, we can notice that different
way of split train/test dataset will lead to minor difference, but not significantly
large different between different seed, usually like 10 or 20 percentage difference
```

Part 2. Logistic regression

In this section, we will apply logistic regression to predict the class of synthetic data with noise, and evaluate our models with the cross validation approaches, as well as another technique called the bootstrap.
Before anything else, let us run the following code to create a synthetic data set `df` and make a plot. Again, let us set the random seed to make sure our results are reproducible.

```
> set.seed(1)
> n = 1000
> x1 <- runif(n)
> x2 <- runif(n, -2, 1)
> z <- (x1-0.2)*(x1-0.5)*(x1-0.9) * 25 - x2*(x2+1.2)*(x2-0.8) + rnorm(n)/3
> y <- as.integer(z>0)
> plot(x1, x2, col=c("red", "blue")[y+1])
> df <- data.frame(x1,x2,y)
```

The bootstrap. *Questions 4 concerning the bootstrap is optional for this lab.* The bootstrap is another method for model evaluation, and aims to approximate the out-of-sample (test) error when applied to classification. In this section, we compare the bootstrap to k-fold cross validation.

Bootstrap works by sampling **with replacement** from the original training dataset (with n samples) to generate more datasets of size n (this means that a data point may appear more than once in the generated data). For each of the new datasets (which we shall refer to as the *bootstrap datasets*), the statistic of interest is recomputed, and its average over all runs is returned. For estimating the (expected) prediction error, the simplest pipeline is described in Algorithm 1.

Algorithm 1 Estimating prediction error via the bootstrap

```
1: Input: dataset $\{(x_i, y_i)\}_{i=1}^n$, loss function $\ell$.
2: for $b = 1, 2, \ldots, B$ do
3:     generate set $\mathcal{S}_b$ by sampling $n$ items with replacement from $\{(x_i, y_i)\}_{i=1}^n$.
4:     form prediction $\hat{f}^b$ by fitting the logistic regression model with training data $\mathcal{S}_b$.
5:     compute $\widehat{\text{err}}_b := \frac{1}{n}\sum_{i=1}^n \ell(y_i, \hat{f}^b(x_i))$
6: end for
7: return $\widehat{\text{err}}_{\text{boot}} := \frac{1}{B}\sum_{b=1}^B \widehat{\text{err}}_b$
```

Questions on the bootstrap.

1. Do you expect Algorithm 1 to provide a good estimate of the prediction error? Why or why not?

```
Yes, because if B is large enough, we can expect the law of large number happens
to all of the data points, that is to say the occurance of each data points should
around the same.
```

2. Bootstrapping introduces significant overlap between the set of data used for training and the validation set (the old training set). Show that each training sample gets picked with probability ≈ 0.632 for a fixed bootstrap dataset, as the number of training samples $n \to \infty$. What is the expected number of distinct samples used in forming a bootstrap dataset, as n grows large?

```
        For an arbitary data point
        P(not being selected)=(1-1/n)^n
        when n->∞ ,(1-1/n)^n=1/e
        P(being selected)=1-1/e=0.632
        N=P(being selected)*n=0.632*n
```

3. Using the `boot` function from the library with the same name, apply Algorithm 1 with $B = 10$ to the logistic regression problem, using the data in `df`.You will need to implement a function `boot_fn(data, train_id)` to pass to `boot`, which will do the following:

   (i) using the subset of `data` designated by the indices `train_id`, fit a logistic regression using `glm()` (and its `family` parameters; see the `R` help text).

   (ii) return the mean prediction error on the **whole** dataset using the `glm` fit.

```
boot_fn= function(data, train_id){
  glm.fit=glm(formula = y~x1+x2, data=data,subset=train_id,family = binomial )
  glm_mse=mean((y-predict(glm.fit,data))^2)
  return(glm_mse)
}
train_id1=sample(1000,1000,replace = TRUE)
test1=boot_fn(df,train_id1)
```

4. (Optional) In an attempt to combine the best of both worlds, the leave-one-out bootstrap estimate replaces Line 7 with the following estimate:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{|B_{\setminus\{i\}}|}\sum_{b\in B_{\setminus\{i\}}}\ell(y_i,\hat{f}^b(x_i)), \tag{1}$$

where $B_{\setminus\{i\}}$ is the set of bootstrap datasets which do not contain sample i from the original dataset. Note that we may swap the above as

$$\frac{1}{n}\sum_{b\in B}\sum_{i\notin \mathcal{S}_b}^{n}\frac{1}{|B_{\setminus\{i\}}|}\cdot \ell(y_i,\hat{f}^b(x_i)) \tag{2}$$

We will implement Algorithm 1 with this modification step-by-step in the next few questions:

(i) Write a function `boot_poly` that accepts two arguments, `train_id, deg`, where the first is the training indices and the second is the polynomial degree, and fits a logistic regression model with `deg`-polynomial terms for the predictors, using the samples corresponding to `train_id`, returning the fit model. (*Hint: Use the* `glm()` *function and is* `family` *parameter to specify that you are fitting a logistic reg. model.*

(ii) Write a function `boot_num` that accepts two arguments, an index `i` and a $n \times B$ **matrix** where column b contains the training indices of bootstrap b, and returns the number of bootstrap datasets that do not contain `i`.

(iii) Use the following code to fit $B = 10$ bootstrap datasets for given degree `deg = 2` as a guide:

```
> B <- 10
> deg <- 2
> all_inds <- matrix(sample(1:n, B * n, replace=TRUE), ncol=B)
> boot_fits <- apply(all_inds, 2,
+ function(trainind) return(boot_poly(df,trainind, deg)))
```

The above code returns a B-dimensional vector of `glm` objects. Using `bootNum`, write a function computing (2). (*Hint*: while not necessary, you may find the functions `apply, lapply, sapply` useful.)

Logistic regression with cross-validation. We want to fit logistic regression models of different degrees p, up to $p = 5$ (see the `family` parameter of the `glm()` function). First, hold out 100 samples of the dataset as a test set, and use the remainder as training set. Use k-fold cross validation with $k = 10$ and LOOCV to evaluate the performance for each degree p, and compare with LOOCV. Then, choose the best model from the k-fold validation step and compute its error on the test set. Some remarks:

- we are using the overall accuracy rate as the metric for the error.
- define a cost function using the following code, and pass that function as the cost argument to `cv.glm()`:

```
        cost_fn <- function(r, pi=0) {
                mean(abs(r - pi) > 0.5)
        }
```

  What does the `cost_fn` function do, and how does it relate to classification with logistic regression?

```
calculate the error rate of a logistic model given the list of predicted value
and true class.
r is predicted value, pi is the true class
If abs(r - pi)>0.5(0.5 is general threshold of logistic regression, then we will
 count this data point as' mis-classified, take the mean of it to get error rate
```

- when converting the predictions to integers, which threshold should you use?

```
    if(prediction >0.5)
```

Report your findings below:

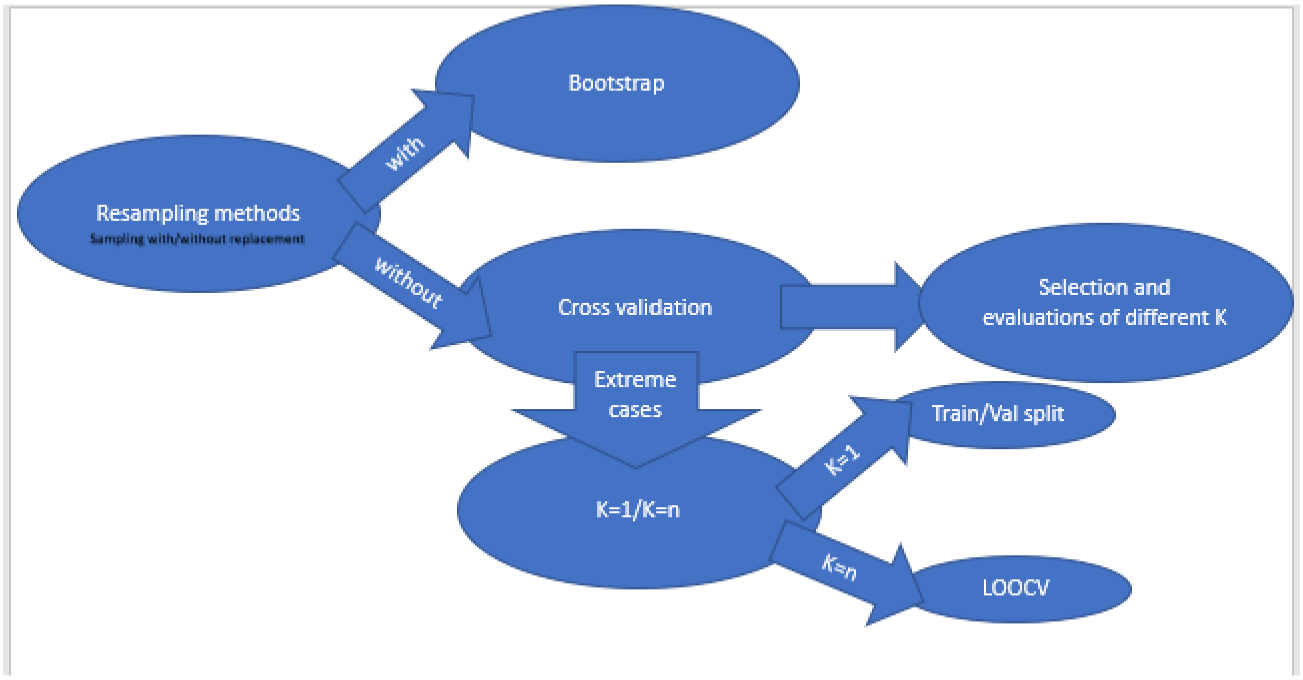
1. What are the validation errors you get for different degrees (k-fold vs. LOOCV)? Which model do they suggest you should choose? Is the choice reasonable?

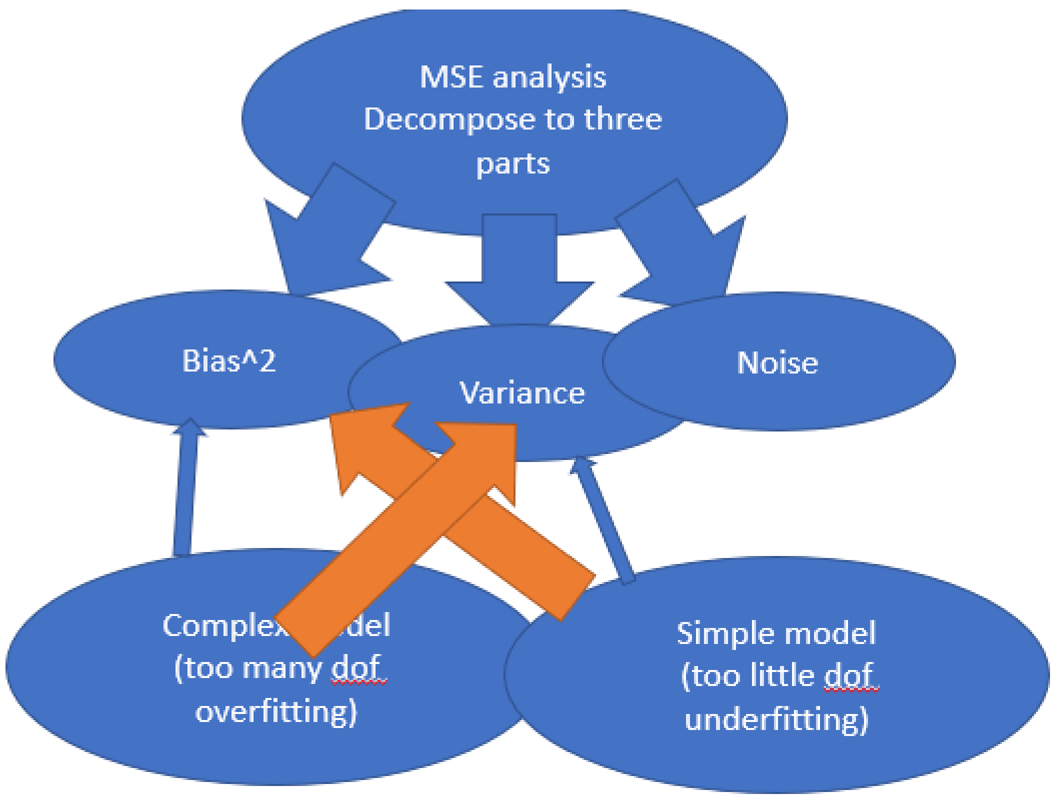
```
I select K={5,10,15,20,25}, all of them returns 0.224 as validation error
As for LOOCV, validation error=0.2239, which is quite similar to the k-fold
result
Thus, consider about computational efficiency, CV K=5 is the model I will chose
```

2. What is the test error of your chosen model?

```
Testing error=0.23
```

Concept map. Make a concept map for the topics covered by this lab. While there is no single right answer, you are expected to demonstrate that you understand the relationships between various topics in the lab.




MSE analysis
Decompose to three parts
Bias^2
Variance
Noise
Complex model (too many dof overfitting)
Simple model (too little dof underfitting)