

SAMPLE SLIDES FOR SUPERVISED LEARNING

Feature Selection

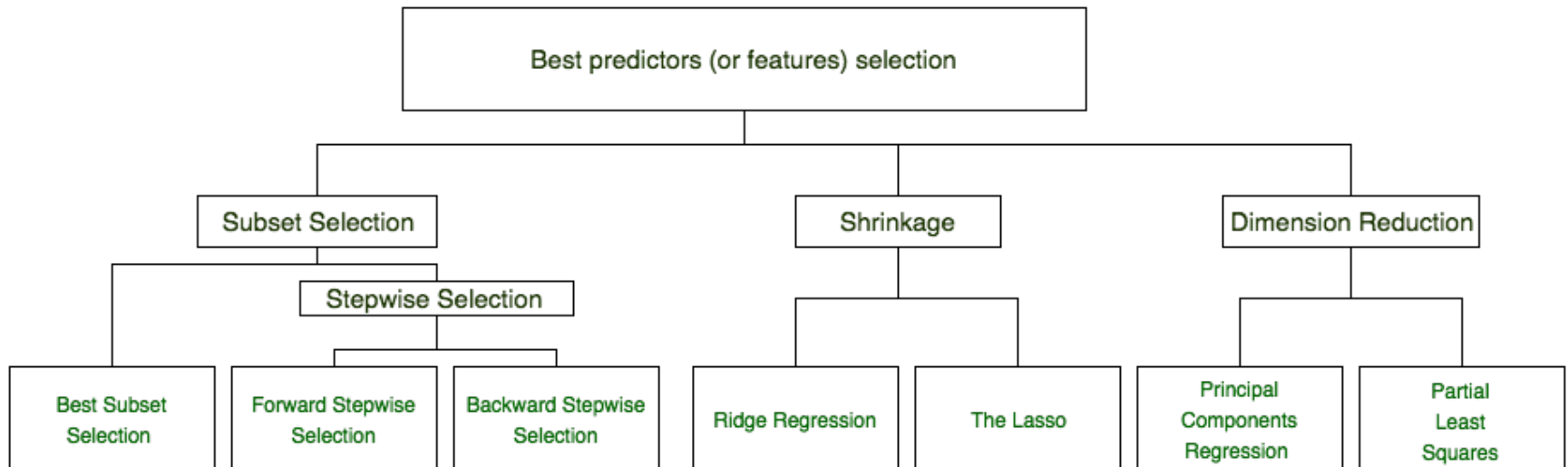
Overview

- Introduction
 - Methods of Linear Model Selection & Regularization
 - The elements of test MSE
 - Hands-on; Data Cleansing
- Subset Selection Methods
 - Best Subset Selection
 - Forward Stepwise Selection
 - Backward Stepwise Selection
 - Hands-on; Best Subset & Stepwise Methods
 - Choosing the optimal model
 - Hands-on; Subset Selection with Cross-Validation
 - The drawbacks
- Shrinkage Methods
 - Ridge Regression
 - Hands-on; Ridge Regression
 - Ridge Regression – The drawbacks
 - The Lasso
 - Hands-on; The Lasso with Cross-Validation
 - Comparing The Lasso & Ridge Regression
- High-Dimensional Data
 - Issues in High-Dimensions
 - Interpreting Results in High-Dimensions

Introduction – Methods of Linear Model Selection & Regularization

- In more detail, there are three basic categories of methods regarding predictors selection:
 - **Subset Selection:** This approach involves identifying a subset of the p predictors that we believe to be related to the response. Is actually an enumeration method.
 - **Shrinkage:** This approach involves fitting a model involving all p predictors. However, the estimated coefficients are shrunk towards zero relative to the least squares estimates. Depending on what type of shrinkage is performed, some of the coefficients may be estimated to be exactly zero.
 - **Dimension Reduction:** This approach involves projecting the p predictors into a M -dimensional subspace, where $M < p$. This is achieved by computing M different linear combinations, or projections, of the variables. Then these M projections are used as predictors to fit a linear regression model by least squares.

A summary of best predictors selection



- Each final method is represented in green
- In this lecture we will examine the **Subset Selection & Shrinkage Methods**

Variance Error

- Variance error, refers to the amount by which \hat{f} would change if we estimated it using a different training data set.
 - Our goal is to reduce that error
- If a method has high variance then small changes in the training data can result in large changes in \hat{f}
- In general, more flexible statistical methods have **higher variance**

Bias Error

- Bias error, refers to the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model.
 - Again, our goal is also to reduce that error.
- Generally, more flexible statistical methods have **lower bias**.

Bias-Variance tradeoff

- Variance error

- *In general, more flexible statistical methods have **higher variance**.*

- Bias error

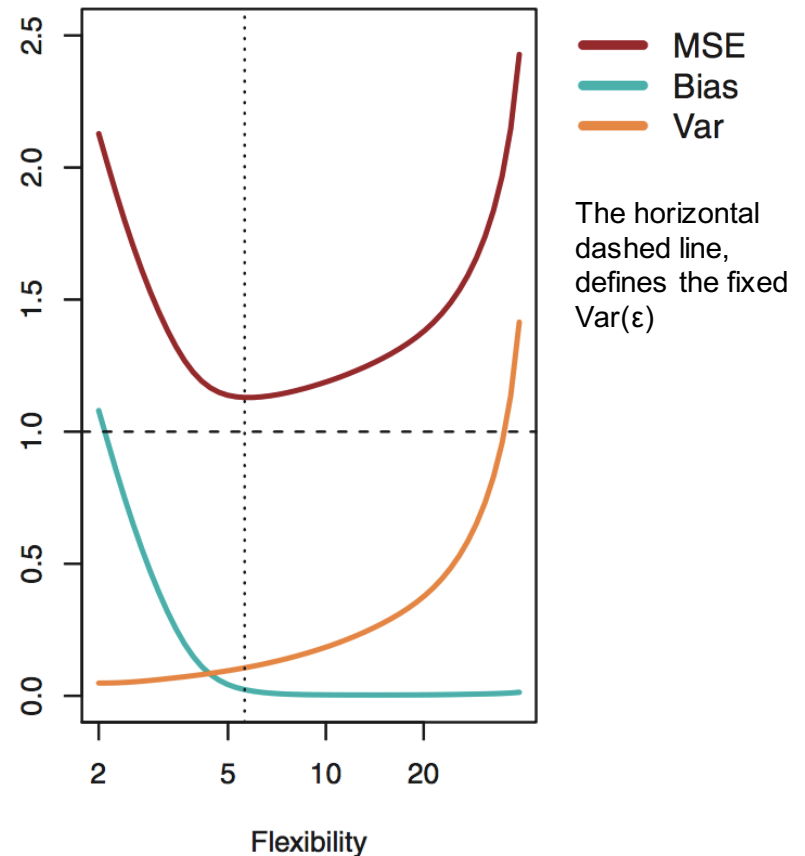
- *Generally, more flexible statistical methods have **lower bias**.*

- Remember!

- Our goal is to reduce both errors as possible in order to reduce the MSE of our test data
 - But it seems to be a “love & hate” connection between bias and variance error.
 - As bias error reduces , variance error increases and vice versa.
 - This phenomenon is called as **Bias-Variance tradeoff (or dilemma)**

Illustration of Bias-Variance trade-off

- As we have previously mentioned, as bias error reduces, variance error increases and vice versa.
- The challenge for us is to **define a model that can reduce both errors**
- This trade-off is one of the most important recurring themes in statistical learning



Shrinkage Methods – Ridge Regression

Scaling predictors

But there is one significant distinction between least squares & ridge regression

The standard least squares coefficient estimates are scale equivariant:

Regardless of how every predictor is scaled (e.g. in millions, thousands, units),

$X_j \hat{\beta}_j$ will remain the same.

In contrast, the ridge regression coefficient estimates can change substantially when multiplying a given predictor by a constant, due to the sum of squared coefficients term in the penalty part of the ridge regression objective function

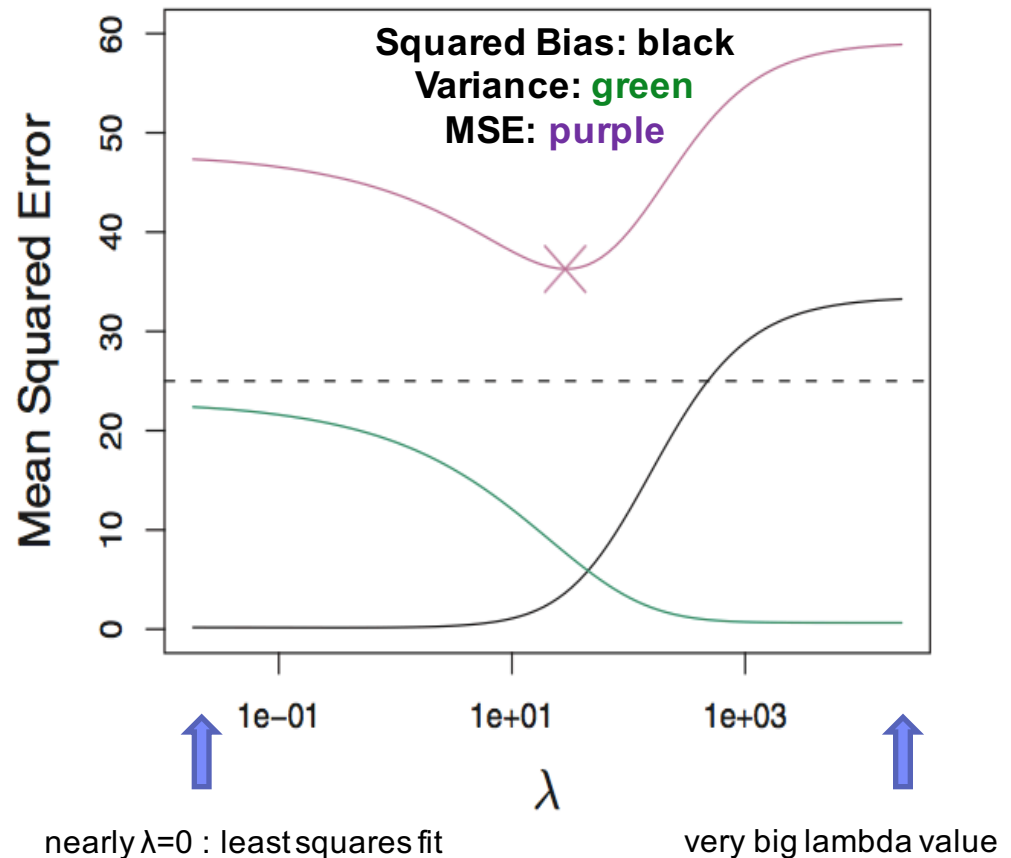
Therefore, it is best to apply ridge regression after standardizing the predictors, using the formula:

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2}}$$

Note that the packages in R, normally scale the predictors automatically.

Ridge Regression vs Least Squares

- Ridge regression's advantage over least squares is rooted in the bias-variance trade-off.
 - As λ increases, the flexibility of the ridge regression fit decreases, leading to decreased variance but increased bias.
 - But as λ increases, the shrinkage of the ridge coefficient estimates leads to a substantial reduction in the variance of the predictions, at the expense of a slight increase in bias.
 - At the least squares coefficient estimates, which correspond to ridge regression with $\lambda = 0$, the variance is high but there is no bias.



The Variable Selection Property of the Lasso

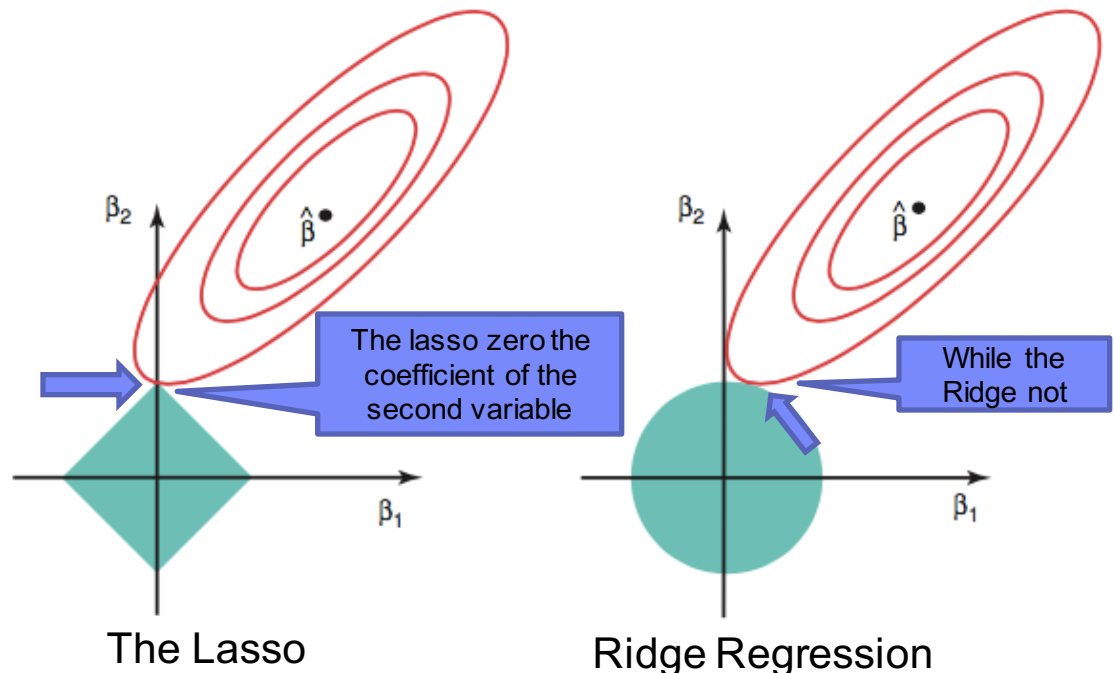
Geometric interpretation for two predictors (not predictor and response)

The least squares solution is marked as $\hat{\beta}$

The constraints are represented by the blue diamond and circle

If s , is sufficiently large, the coefficients will get the least squares value ($\lambda=0$)

The Lasso and Ridge Regression coefficient estimates are given by the first point at which an ellipse contacts the constraint region



The lasso constraint has corners at each of the axes, and so the ellipse will often intersect the constraint region at an axis. When this occurs, one of the coefficients will equal zero.

When $p=3$ constraints are represented with polyhedron & sphere respectively

Comparing The Lasso & Ridge Regression

It is clear that the lasso has a major advantage over ridge regression, in that it produces simpler and more interpretable models that involve only a subset of the predictors. However, **which method leads to better prediction accuracy?**

In general, neither ridge regression nor the lasso will universally dominate the other.

However, one might expect **the lasso to perform better when the response is a function of only a relatively small number of predictors.**

Ridge regression will perform better when the response is a function of many predictors, all with coefficients of roughly equal size.

But we also have to consider that the number of predictors that is truly related to the response is never known a priori (from the beginning) for real data sets.

Here (once more 😊) cross-validation can be used to determine the best method.

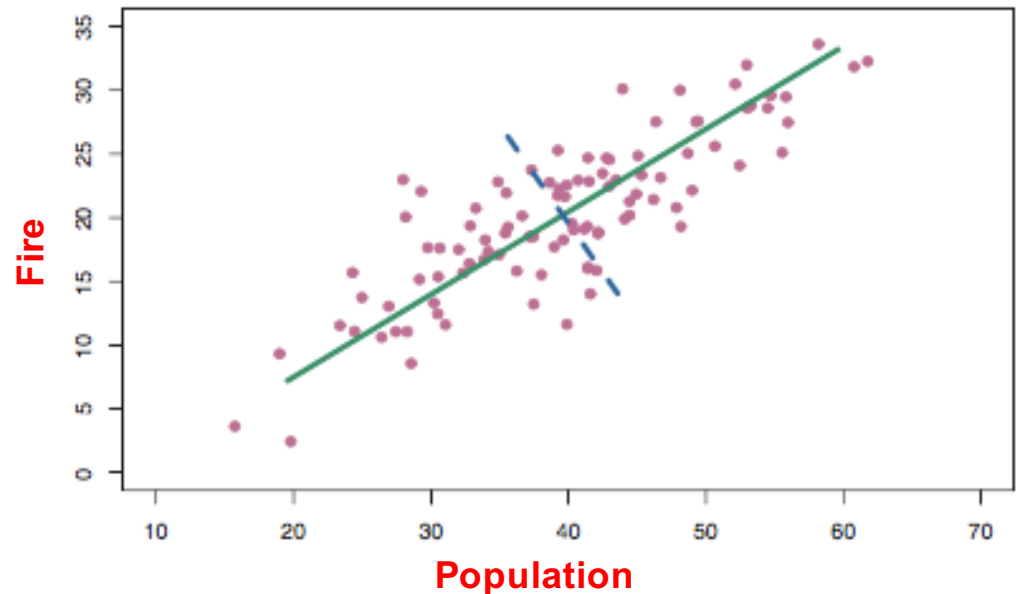
Principal Components Regression (PCA)

Geometric interpretation for **two predictors (not predictor and response)**

Let's assume that we have the predictors **population & fire** and the **response house sales**.

The biggest variation of the data is described with the green line (PC1)

And the second blue dashed line describes the second biggest variation (uncorrelated to the PC1) of the data. (PC2)



Want to know more about the concept of PCA with simple examples?
<https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues>

Issues in High-Dimensions

It turns out that best feature selections, such as forward stepwise selection, ridge regression, the lasso, and principal components regression, are particularly useful for performing regression in the high-dimensional setting.

Essentially, these approaches avoid overfitting by using a less flexible fitting approach than least squares.

There are three important points in these methods:

- Regularization or shrinkage plays a key role in high-dimensional problems
- Appropriate tuning parameter selection is crucial for good predictive performance
- **The test error tends to increase as the dimensionality of the problem (i.e. the number of features or predictors) increases, unless the additional features are truly associated with the response.**

The last point is also known as the “curse of dimensionality”

Issues in High-Dimensions

In general, adding additional **signal features** that are truly associated with the response will improve the fitted model, in the sense of leading to a reduction in test set error.

However, adding **noise features** that are not truly associated with the response will lead to a deterioration in the fitted model, and consequently an increased test set error.

We see that new technologies that allow for the collection of measurements for thousands or millions of features are a double-edged sword: they can lead to improved predictive models if these features are in fact relevant to the problem at hand, but will lead to worse results if the features are not relevant.

**And even if they are relevant, the variance incurred in fitting their coefficients may outweigh the reduction in bias that they bring.
(recall the bias-variance trade-off)**

Interpreting Results in High Dimensions

When we perform best predictor (or features) selection methods in the high-dimensional setting, we must be quite cautious in the way that we report the results obtained.

In previous lectures we have mentioned **multicollinearity, the concept that the variables in a regression might be correlated with each other.**

In the high-dimensional setting, the multicollinearity problem is extreme: any variable in the model can be written as a linear combination of all of the other variables in the model.

This means that we can never know exactly which variables (if any) truly are predictive of the outcome, and we can never identify the best coefficients for use in the regression.

At most, we can hope to assign large regression coefficients to variables that are correlated with the variables that truly are predictive of the outcome.

Hands-on

Forward and Backward Stepwise Selection

Now we will follow the same procedure for the Forward Stepwise Selection method

The command in this case will be:

```
regfit.fwd=regsubsets(house_sales_prices_mean ~ . , dataset,  
method = "forward", nvmax=11)
```

The summary and the selected variables for each model can be retrieved with:

```
summary(regfit.fwd)
```

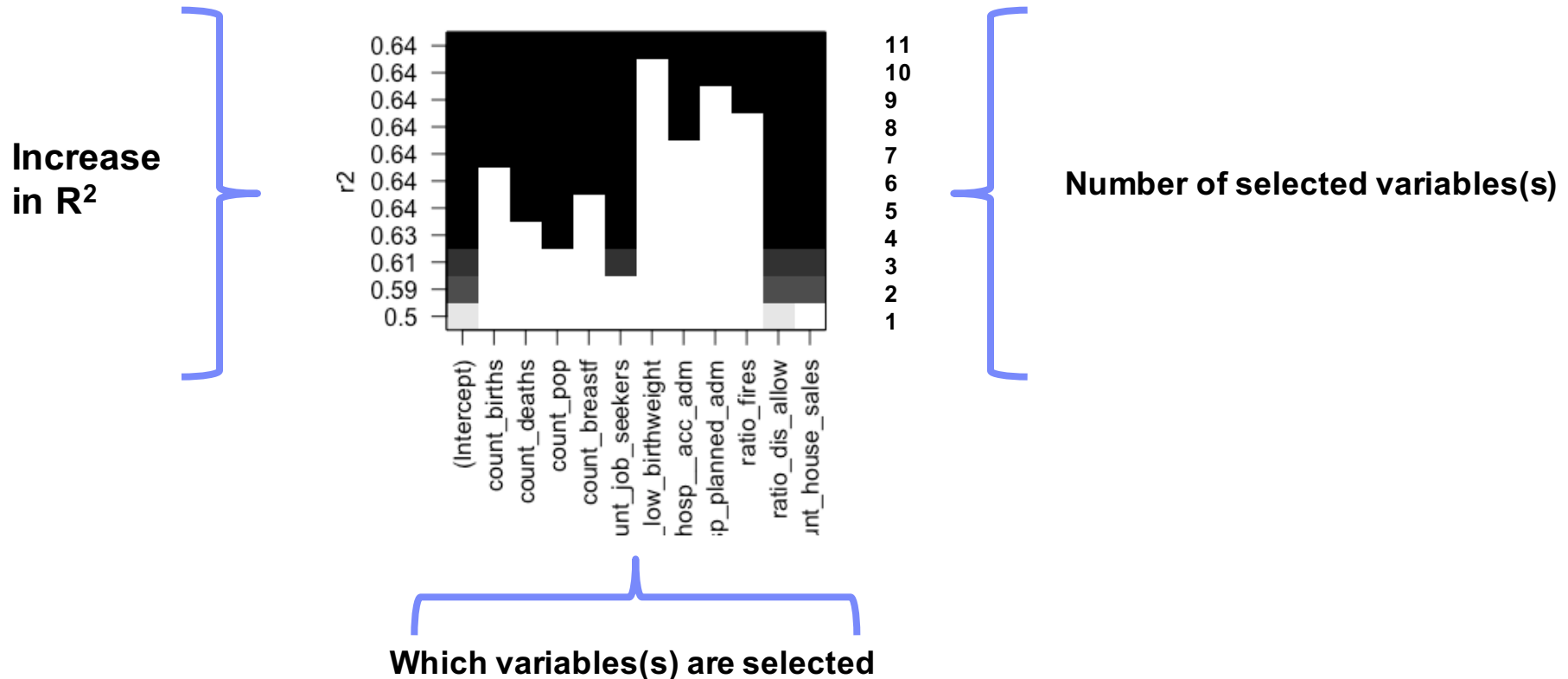
```
1 subsets of each size up to 11  
Selection Algorithm: forward  
count_births count_deaths count_pop count_breastf count_job_seekers ratio_low_birthweight ratio_hosp__acc_adm ratio_hosp_planned_adm ratio_fires ratio_dis_allow count_house_sales  
1 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11  
2 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11  
3 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11  
4 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11  
5 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11  
6 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11  
7 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11  
8 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11  
9 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11  
10 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11  
11 ( 1 ) 11 11 11 11 11 11 11 11 11 11 11
```

Hands-on

Forward and Backward Stepwise Selection

And again, we can get the selected variables for each model with:

```
plot(regfit.fwd, scale="r2")
```



Hands-on

Ridge Regression

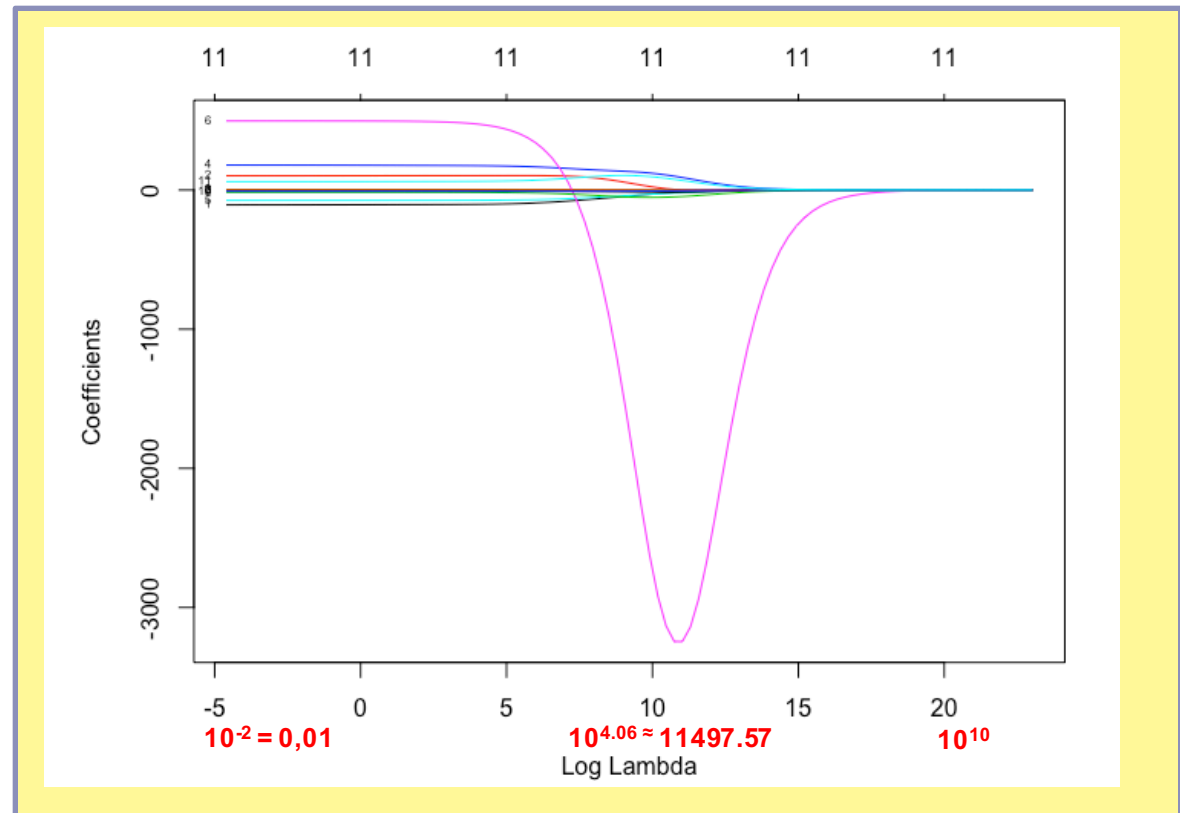
We can summarize all the previous computations with the following plot

```
plot(ridge.mod, xvar="lambda", label=TRUE)
```

Pay attention to the 6th predictor (purple line).

In our case is the
"ratio_low_birthweight".

If you check the
coefficients of this
variable for every
previous case and
lambda, you will also
be able to understand
the variation that
seems to be in the plot



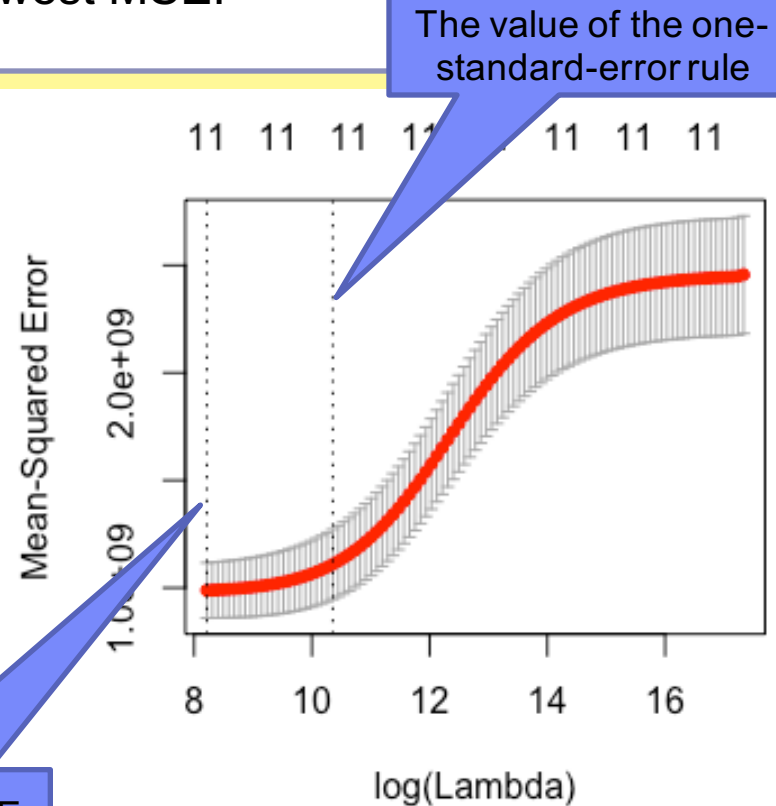
Hands-on

Ridge Regression – Cross-Validation

Now we will use the built-in ten-fold cross-validation function (`cv.glmnet`) to choose the best value of λ based on the lowest MSE.

```
set.seed(1)
train=sample(1:nrow(x), nrow(x)/2)
test=(-train)
y.test=y[test]
cv.out=cv.glmnet(x[train,],y[train],alpha=0)
plot(cv.out)
```

In this way, we can produce a plot which demonstrates the value of lambda and the equivalent test MSE.



Hands-on

Below you can find a summary of all cross-validated methods with its test MSEs

Method	Best Subset Selection	Forward Stepwise Method	Backward Stepwise Method	Ridge Regression	The Lasso -minimum MSE-	PCR	PLS
MSE	908.049.385	905.025.196	915.138.527	801.510.608	880.040.790	4.121.642.019	4.164.388.362
Number of Variables	7	5	7	All	9	--	--

Classification

Our Dataset

- For our hands-on we will import the dataclass.csv
- We have selected as geographic area, the 2001 data zones, as they can offer the highest level of granularity
- The file contain 5 variables in total
- The one which refers to “Urban Rural Classification” will be our response

COLUMNS (DATASET SLICES): 5

Dwellings by Number of Rooms (1 column)

Measure Type = Ratio, Number Of Rooms = 9, Reference Period = 2012

Hospital Admissions (2 columns)

Reference Period = 2012, Admission Type = Emergency, Age = All, Gender = All, measure type = Ratio

Reference Period = 2012, Admission Type = Disease Of The Digestive System (DDS), Age = All, Gender = Female, measure type = Ratio

Incapacity Benefit and Severe Disablement Claimants (1 column)

Measure Type = Ratio, Reference Period = 2012-Q2, Age = 16+, Benefit Type = IBSDA, Gender = All

Urban Rural Classification (6-Fold) (1 column)

Measure Type = Rank, Reference Period = 2011/2012

ROWS (GEOGRAPHIC AREAS): 6505

2001 Data Zones (6505 areas)

2001 Data Zones (6,505)

The categories of “Urban Rural Classification”

- The classification distinguishes between urban, rural and remote areas within Scotland and includes the following categories:
 1. Large Urban Areas Settlements of over 125,000 people;
 2. Other Urban Areas Settlements of 10,000 to 125,000 people;
 3. Accessible Small Towns Settlements of between 3,000 and 10,000 people and within 30 minutes drive of a settlement of 10,000 or more;
 4. Remote Small Towns Settlements of between 3,000 and 10,000 people and with a drive time of over 30 minutes to a settlement of 10,000 or more;
 5. Accessible Rural Settlements of less than 3,000 people and within 30 minutes drive of a settlement of 10,000 or more;
 6. Remote Rural Settlements of less than 3,000 people and with a drive time of over 30 minutes to a settlement of 10,000 or more.
- **So, every area (data zone) is classified into these 6 categories**

Reference Area ↓	Rank
2001 Data Zones	
S01000001	3
S01000002	1
S01000003	1
S01000004	1
S01000005	3
S01000006	3
S01000007	3
S01000008	3
S01000009	1

Transforming the response to boolean

- However, it seems that our response is not appropriate for a classification problem.
- Our response should clearly refer to only two situations;
 - if a given area is classified in **one category or not**.
- For this reason, we will need to transform our response into six sub-responses
 - This can easily be done (here for the first and second category) with the following command:

```
dataset$cat1 <- ifelse(dataset$urbanclass == 1 , 1, 0)  
dataset$cat2 <- ifelse(dataset$urbanclass == 2 , 1, 0)  
[...]
```

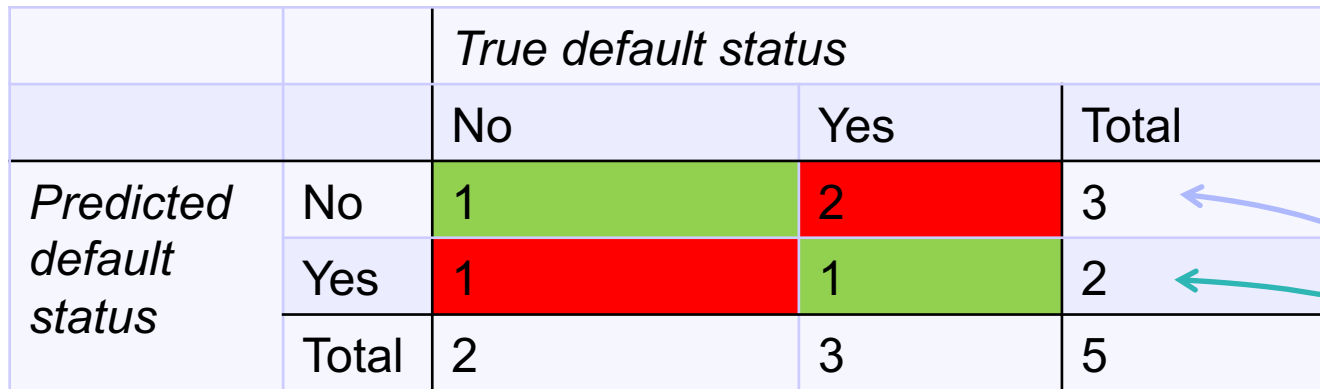
- Which creates a boolean (or binary) column for the category 1 & 2.
 - If the value is one, then indeed the area belong to the category.
 - If the value is zero it doesn't.
- The same scenario applies for each of the six categories.

	area	room_9	emerge_admiss	female_digest_admiss	ibsd_a_benefit	class	cat1
1	S01000001	2	6190	4049	3.9	3	0
2	S01000002	0	6735	2907	1.1	1	1
3	S01000003	0	5711	2222	0.7	1	1
4	S01000004	0	8183	4290	1.0	1	1
5	S01000005	1	6113	3236	1.0	3	0

Confusion Matrix

- So from the previous results and for threshold 0.4 we can assume the following:

		<i>True default status</i>		
		No	Yes	Total
<i>Predicted default status</i>	No	1	2	3
	Yes	1	1	2
	Total	2	3	5

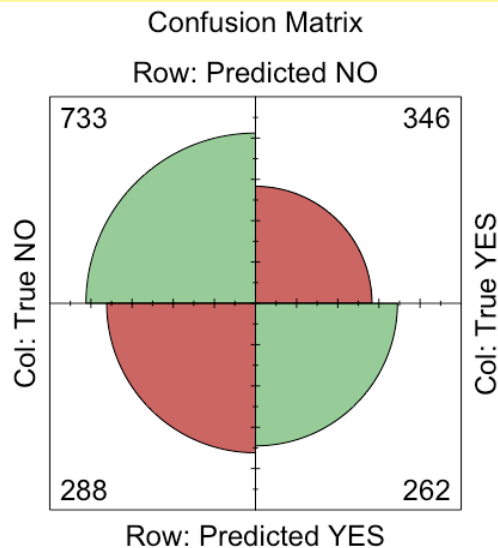


- *Confusion matrix*, concludes all the predictions and their relevance to true responses.
- For example, our model has predicted two areas to be in category 1. However, one was correct and one was wrong.
- But what about the rest areas?
 - All of them were classified from the model as not belonging to category one, but in reality two of them were truly belonging to category 1.
- The green boxes are actually the correct predictions

Four fold plot

- And what if we would like to better visualize our results?
 - Then we can use the built-in fourfoldplot() function

```
ctable <- as.table(matrix(c(res[1,1], res[1,2], res[2,1], res[2,2]), nrow = 2, byrow = TRUE))
rownames(ctable)= c("Predicted NO", "Predicted YES")
colnames(ctable) = c("True NO", "True YES")
fourfoldplot(ctable, color = c("#CC6666", "#99CC99"), conf.level = 0, margin = 1, main =
"Confusion Matrix")
```



Which is equivalent to our table...

		True default status		
		No	Yes	Total
Predicted default status	No	733 (1)	346 (2)	1079 (N)
	Yes	288 (3)	262 (4)	550 (Y)
	Total	1021 (N*)	608 (Y*)	1629 (T)

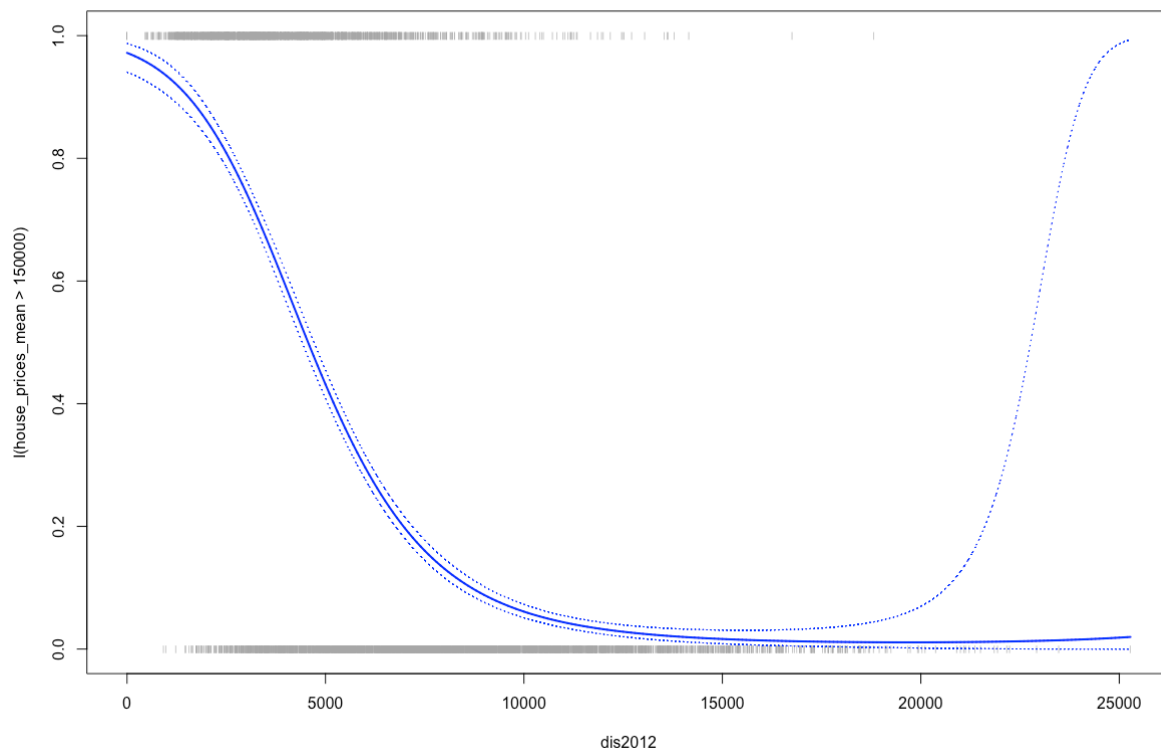
Flexible Fits

Polynomial Logistic Regression plot

- Then we plot the 4th degree regression line, the points and confidence intervals

```
plot(dis2012,l((house_prices_mean>150000)),xlim=dislims,type="n",ylim=c(0,1))
points(jitter(dis2012),l((house_prices_mean>150000)),cex=.5,pch="|",col="darkgrey")
lines(dis.grid,pfit,lwd=2,col="blue")
matlines(dis.grid,se.bands,lwd=1,col="blue",lty=3)
```

- We used the jitter() function to jitter the disability living allowance values a bit so that observations with the same age value do not cover each other up. This is also called a rug plot.

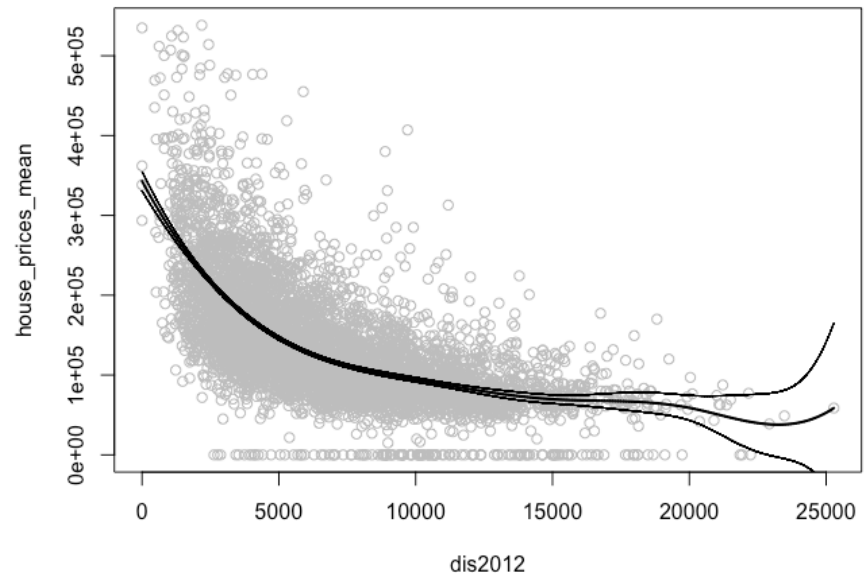


A Cubic Spline with three knots

- Now, we will create our first spline with three knots (disability living allowance at: 10000, 15000, 20000)
 - We will make also our predictions, and we will plot, both the regression spline and its confidence intervals

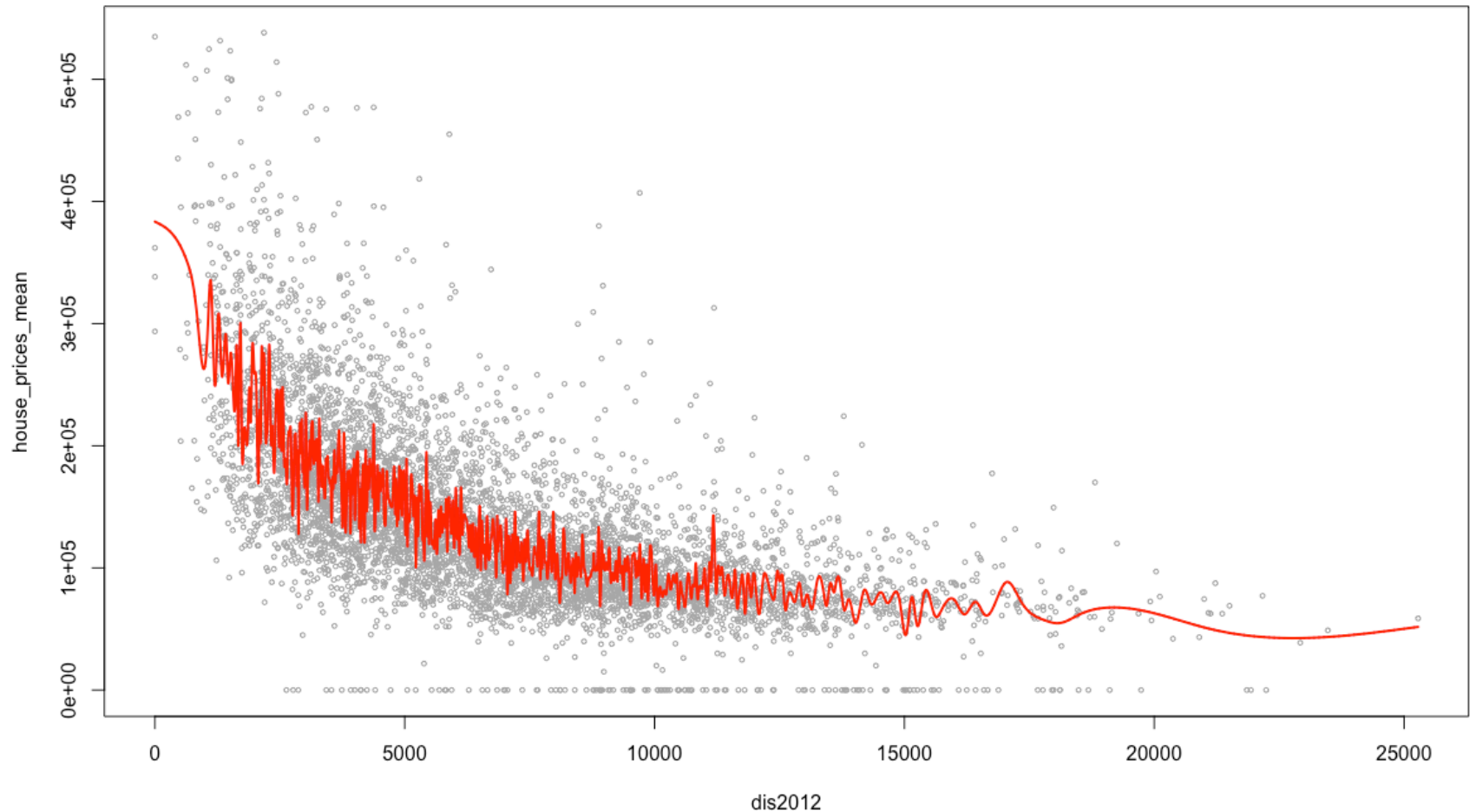
```
library(splines)
fit=lm(house_prices_mean~bs(dis2012,knots=c(10000,15000,20000)),data=dataset)
pred=predict(fit,newdata=list(dis2012=dis.grid),se=T)
plot(dis2012,house_prices_mean,col="gray")
lines(dis.grid,pred$fit,lwd=2)
lines(dis.grid,pred$fit+2*pred$se,lty="dashed")
lines(dis.grid,pred$fit-2*pred$se,lty="dashed")
```

- The `bs()` function generates the entire matrix of `bs()` basis functions for splines with the specified set of knots. By default, cubic splines are produced.



500 degrees of freedom

- What if we would like to retrieve a natural splines with 500 degrees of freedom?



High vs Cross-Validated degrees of freedom

- Then we plot them with the following commands

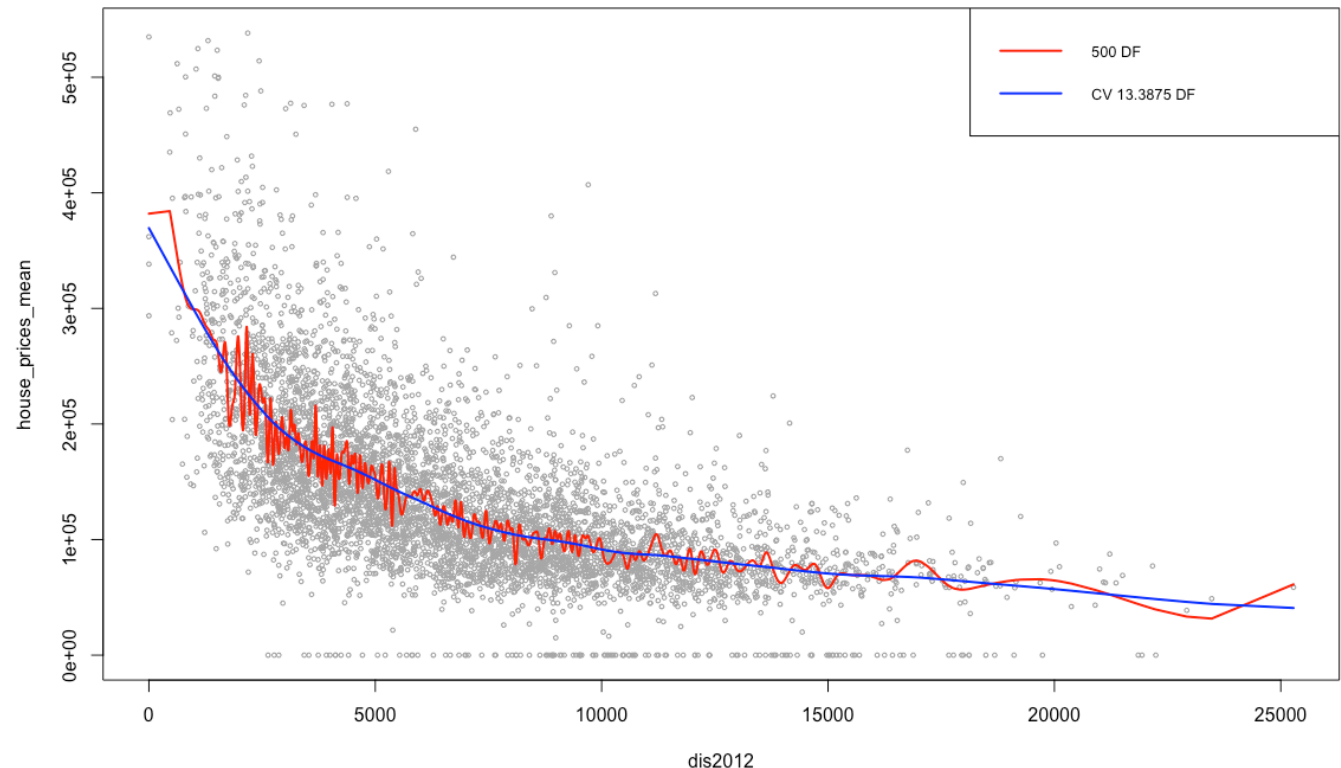
```
plot(dis2012,house_prices_mean,xlim=dislims,cex=.5,col="darkgrey")
title("Smoothing Spline")
lines(fit,col="red",lwd=2)
lines(fit2,col="blue",lwd=2)
df <- round(fit2$df, digits=4)
df <- as.character(df)
df <- paste("CV", df, "DF", collapse = "")
legend("topright",legend=c("500 DF", df ),col=c("red","blue"),lty=1,lwd=2,cex=.8)
```

Smoothing Spline

- fit2\$df returns the chosen df from Cross Validation

fit2\$df

13.38748



Decision Trees

Two-node & Three-node tree

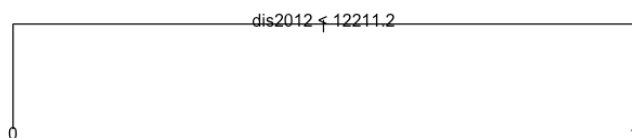
- We now apply the `prune.misclass()` function in order to prune the tree to obtain the two-node tree & three-node tree.

```
prune.cat1=prune.misclass(tree.cat1,best=2)
plot(prune.cat1)
text(prune.cat1,pretty=0)
title("category 1")
tree.pred=predict(prune.cat1,test.subset,type="class")
res_table <-table(tree.pred,test.subset$cat1)
res_table(res_table[1,1]+res_table[2,2])/nrow(test.subset)
nrow(test.subset)
```

```
> res_table
```

```
tree.pred  0  1
          0 999 551
          1  59 105
> (res_table[1,1]+res_table[2,2])/nrow(test.subset)
[1] 0.6441074
> nrow(test.subset)
[1] 1714
```

Category 1 tree - pruned; size:2

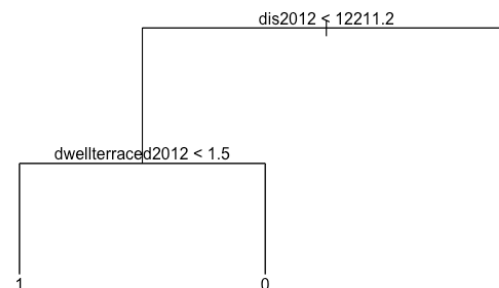


```
prune.cat1=prune.misclass(tree.cat1,best=3)
plot(prune.cat1)
text(prune.cat1,pretty=0)
title("category 1")
tree.pred=predict(prune.cat1,test.subset,type="class")
res_table <-table(tree.pred,test.subset$cat1)
res_table(res_table[1,1]+res_table[2,2])/nrow(test.subset)
nrow(test.subset)
```

```
> res_table
```

```
tree.pred  0  1
          0 908 452
          1 150 204
> (res_table[1,1]+res_table[2,2])/nrow(test.subset)
[1] 0.6487748
> nrow(test.subset)
[1] 1714
```

Category 1 tree - pruned; size:3



Indeed, the three node model results the best accuracy

Evaluating our model with test MSE

- In order to evaluate our regression tree we will split our dataset into train and test (75%-25%):

```
set.seed(1)
train_assign <- sample(c(TRUE,FALSE), nrow(dataset), TRUE, prob=c(0.75,0.25))
train.subset <- dataset[train_assign,]
test.subset <- dataset[!train_assign,]
prices.test <- dataset[!train_assign,"house_prices_mean"]
```

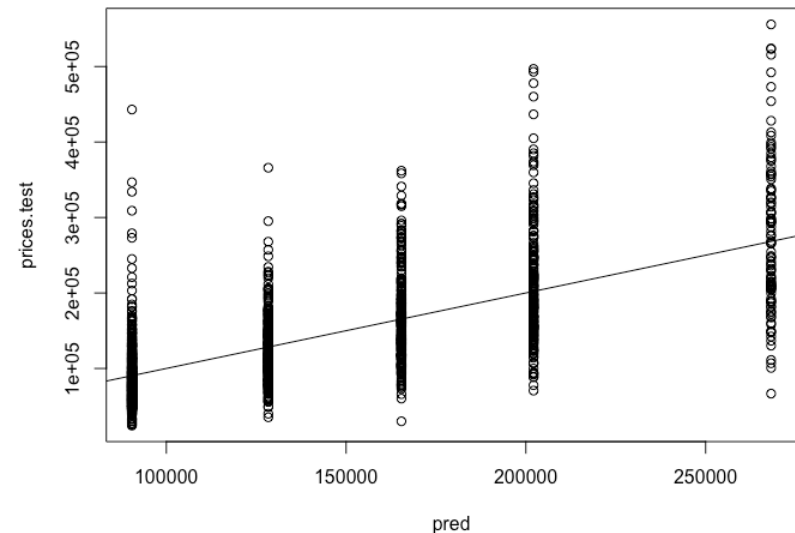
- And we will fit a new tree to find its test MSE

```
tree.prices=tree(house_prices_mean ~ immun2011+
dwellroom12012+ dwellterraced2012 + breast_2012 +
dis2012 + jobs2011 ,data=dataset,
subset=train_assign)

pred=predict(tree.prices, newdata=test.subset)
sqrt(mean((pred - prices.test)^2))

plot(pred, prices.test)
abline(0,1)
```

```
[1] 55610.68
```



Each column represents the final branches and the areas (prices) assigned to them

Pruning with CV

- We can also prune the tree with CV

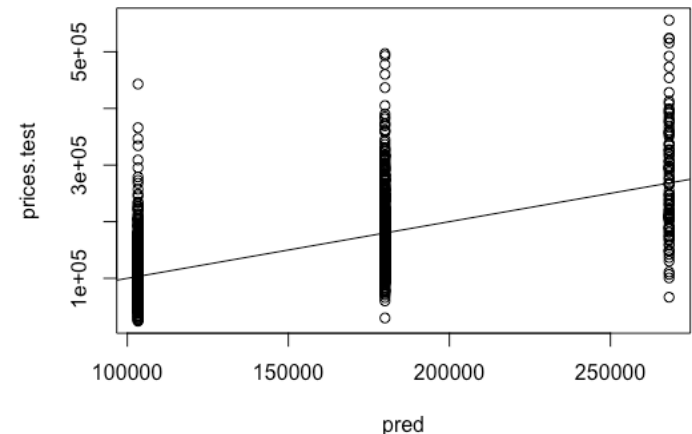
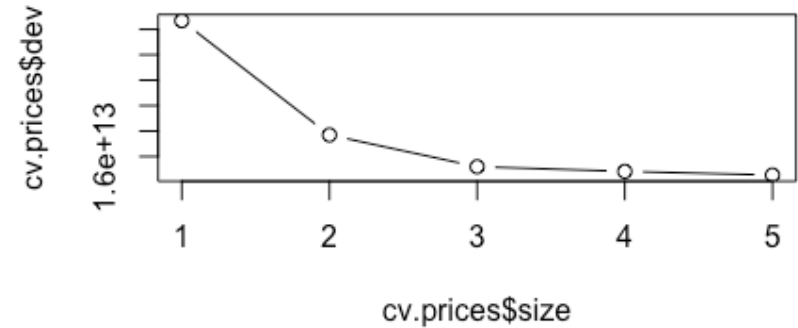
```
cv.prices=cv.tree(tree.prices)
plot(cv.prices$size,cv.prices$dev,type='b')
```

- However, it seems that a tree with all of its nodes (5) yields the best result. Next, will prune a tree to three nodes to prove this.

```
prune.prices=prune.tree(tree.prices,best=3)
pred=predict(prune.prices,newdata=test.subset)
sqrt(mean((pred - test.subset$house_prices_mean)^2))
plot(pred,prices.test)
abline(0,1)
```

[1] 58547.65

- So indeed, a lower node tree yields to a higher MSE.



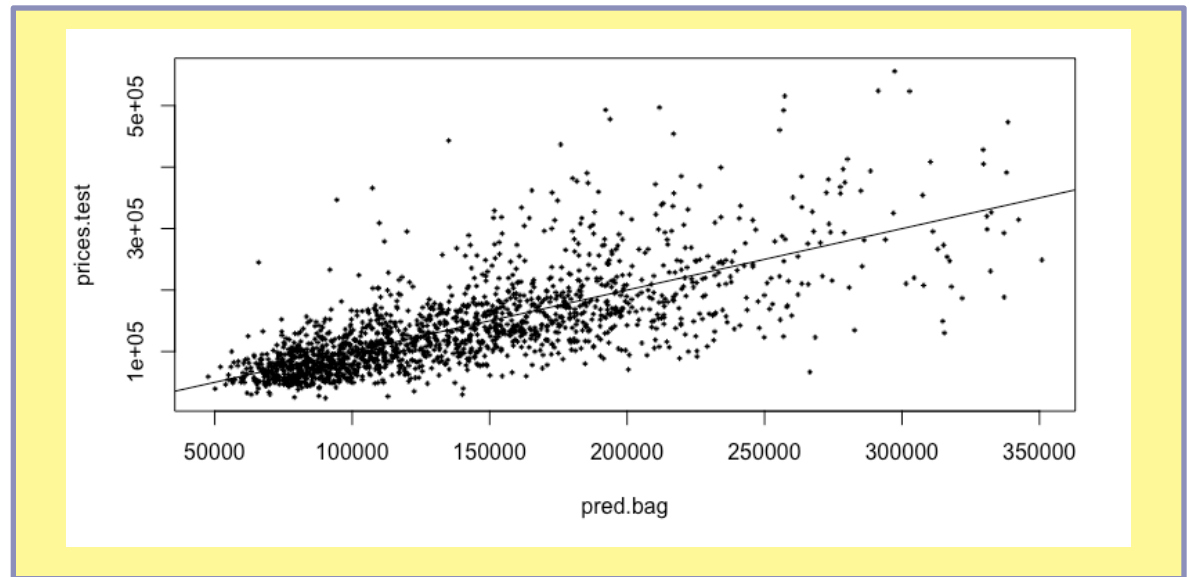
Plotting prediction with real response value.

- And now we will predict the response of test data and we will compute the test MSE.

```
pred.bag = predict(bag.prices,newdata=test.subset)
plot(pred.bag, prices.test, pch=8, cex=.3)
abline(0,1)
sqrt(mean((pred.bag-prices.test)^2))
```

```
[1] 54063.99
```

With *ntree=default (500)*



As a point is closer to the line, such greater is the accuracy for this observation.

Hands-on: Random Forests

- In the next example we will follow the same procedure but for Random Forests

```
ran.prices=randomForest(house_prices_mean ~ immun2011+ dwellroom12012+  
dwellterraced2012 + breast_2012 + dis2012 + jobs2011, data=dataset,  
subset=train_assign, ntree=200, importance=TRUE)
```

- In this case, we will omit the mtry argument
 - so mtry default will be $=\sqrt{p}$ (rounded) [$\text{sqrt}(6) = 2.444$, so mtry will be 2]
- The importance=TRUE argument will return a second measure of variable importance.
- In this fitting we use ntree=200 to fit only 200 trees and avoid further unneeded computations.
- And now we predict the y for test data and calculate the MSE.

```
pred.ran=predict(ran.prices,newdata=test.subset)  
sqrt(mean((pred.ran-prices.test)^2))
```

```
[1] 53069.38
```

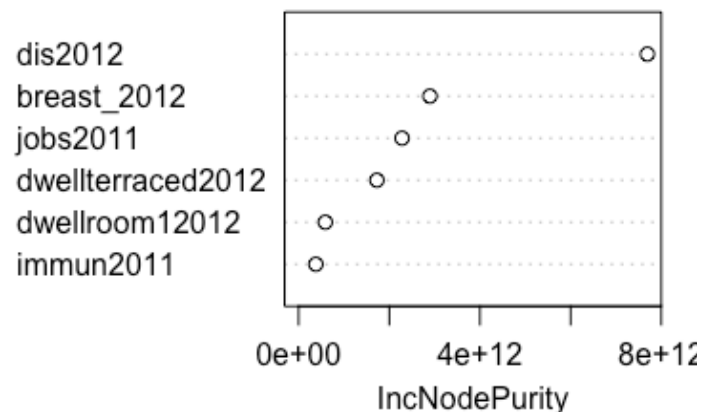
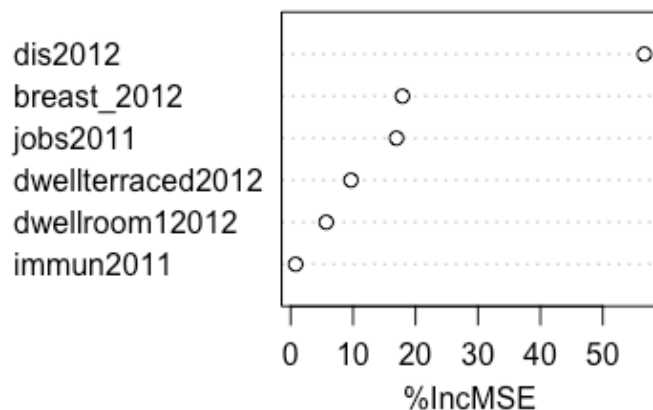
Variable Importance for Random Forests

- The importance and varImpPlot will return the Variable Importance Measures

```
importance(ran.prices)
varImpPlot(ran.prices)
```

ran.prices

	%IncMSE	IncNodePurity
immun2011	0.7957021	3.806502e+11
dwellroom12012	5.6998441	5.895269e+11
dwellterraced2012	9.6620279	1.726797e+12
breast_2012	17.9121102	2.896105e+12
dis2012	56.6589651	7.692913e+12
jobs2011	16.9531003	2.281943e+12



- The %IncMSE is based upon the mean decrease of accuracy in predictions on the out of bag samples when a given variable is excluded from the model.
- The IncNodePurity is a measure of the total decrease in node impurity that results from splits over that variable, average over all trees
 - In the case of regression trees, the node impurity is measured by the training RSS, and for classification trees by the deviance

Same model for different mtry

- For ntree=200 , **seed=1**
 - (not to be confused with the previous examples)

mtry	MSE
6 (bagging)	54201.32
5	54336.58
4	54102.63
3	53608.50
2 (default random forest)	53014.54
1	53360.29

So indeed the rule of $mtry = \sqrt{p}$ seems to offer the best results.

Boosting with higher λ

- Tuning parameter (λ) is by default 0.001 for the gbm function. In this example we set $\lambda=0.020$

```
set.seed(1)
boost.prices=gbm(house_prices_mean ~ immun2011+ dwellroom12012+ dwellterraced2012 +
breast_2012 + dis2012 + jobs2011, data=train.subset, distribution="gaussian", n.trees=5000,
interaction.depth=3 , shrinkage=0.020)

summary.gbm(boost.prices)
pred.boost=predict(boost.prices,newdata=test.subset,n.trees=5000)
sqrt(mean((pred.boost-test.subset$house_prices_mean)^2))
```

```
[1] 54075.31
```

*Which yields to a
slightly higher MSE*

```
> summary.gbm(boost.prices)
```

	var	rel.inf
dis2012	dis2012	65.247820
breast_2012	breast_2012	14.199474
dwellterraced2012	dwellterraced2012	7.322991
jobs2011	jobs2011	6.811241
dwellroom12012	dwellroom12012	3.390439
immun2011	immun2011	3.028035

Boosting for classification

- In the final example we will use boosting for a classification problem. To avoid any faulty assumptions or malfunctions in our code, we will use the CARET library.

```
install.packages('caret', dependencies = TRUE)
library(caret)set.seed(1)

fitControl = trainControl(method="cv", number=10, returnResamp = "all")

model.caret = train(cat1~ immun2011+ dwellroom12012+ dwellterraced2012 + breast_2012 +
dis2012 + jobs2011, data=train.subset, method="gbm",distribution="bernoulli",
trControl=fitControl, verbose=F, tuneGrid=data.frame(.n.trees=2000, .shrinkage=0.01,
.interaction.depth=6, .n.minobsinnode=1))
```

- In the fitControl variable we assign the method of CV (folds 10) and in the model.caret we create and assign the model of a classification boosting for 2000 trees.

Accuracy for classification

- Now we will make the prediction for the test.subset

```
mPred = predict(model.caret, test.subset)#postResample(mPred, test.subset$cat1)
confusionMatrix(mPred, test.subset$cat1)
```

Confusion Matrix and Statistics

	Reference	
Prediction	0	1
0	910	352
1	147	269

Accuracy : 0.7026
95% CI : (0.6801, 0.7244)
No Information Rate : 0.6299
P-Value [Acc > NIR] : 2.205e-10

Kappa : 0.3156
McNemar's Test P-Value : < 2.2e-16

Sensitivity : 0.8609
Specificity : 0.4332
Pos Pred Value : 0.7211
Neg Pred Value : 0.6466
Prevalence : 0.6299
Detection Rate : 0.5423
Detection Prevalence : 0.7521
Balanced Accuracy : 0.6470

'Positive' Class : 0

Finally, we can improve our accuracy to 70% (test subset) with use of boosting.