Statistical programming languages - *House Prices: Advanced Regression Techniques*

Christian Koopmann
Felix Skarke
Enno Tammena

Ladislaus von Bortkiewicz Chair of Statistics
Humboldt–Universitaet zu Berlin
http://lvb.wiwi.hu-berlin.de

Motivation

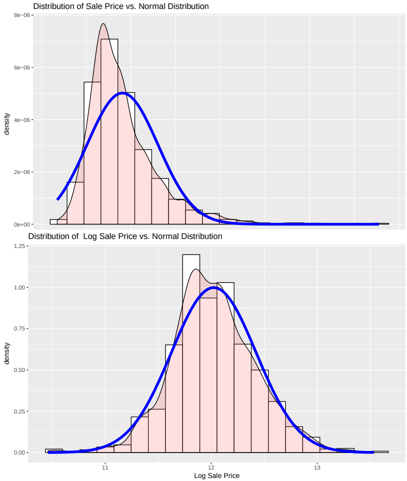
- demonstrate some of Rs machine learning capabilities
- develop and test different models to predict the price of houses in Ames, Iowa
- dataset: Ames Housing data published in a *kaggle* competition with 79 explanatory variables

Outline

Data Structure / Quality

- 2919 Observations separated into 1460 labelled training observations and 1459 unlabelled test observations to predict.
- Target Variable: Final Sale Price in USD
- 79 Explanatory Variables of which 36 are numeric (e.g. floor area) and 43 categoric (e.g. heating type).
- 19 variables contain NA's 16 of which are categoric variables
- NA usually means a variable is not applicable (Pool Quality if house has no pool) instead of missing data

Sale Price Distribution



- Original price distribution heavily skewed
- Log transformed House Prices seem to be normally distributed

Quantlet: Exploratory Data Analysis: Dependence

- aim of this quantlet: show relations between variables, especially the target variable SalePrice
- producing different graphical representations
  - correlation matrix
  - barplot
  - boxplots

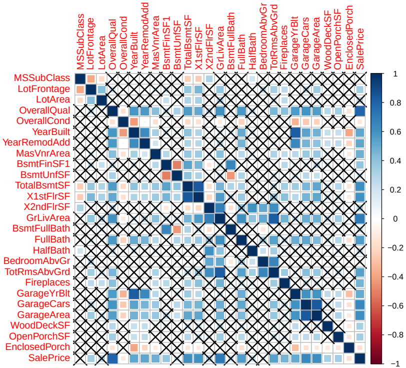
Correlation Matrix of all numeric variables: Code

```
corr.func = function(data, cut.value, corr.mat = FALSE, corr.test =
  FALSE, significance = 0.05) {
    corr.numeric =cor(na.omit(numeric.data))
    find.rows = apply(corr.numeric, 1, function(x) sum(abs(x)>
                abs(cut.value))>1)
    corr.numeric.adjusted = corr.numeric[find.rows, find.rows]


    pdf("Corrplot.pdf")
    if (corr.test == FALSE) {
        corrplot(corr.numeric.adjusted, method = "square")
    } else {
        corrplot(corr.numeric.adjusted,
        p.mat = correlation.test(corr.numeric.adjusted),
        sig.level = significance, method = "square")
    }
    dev.off()

    if (corr.mat == TRUE)
        return(corr.numeric.adjusted)
  }
```

Correlation Matrix of all numeric variables: Code ctd.

```
correlation.test = function(corr.data) {
    corr.data             = as.matrix(corr.data)
    n                     = ncol(corr.data)
    p.value.matrix        = matrix(NA, n, n)
    diag(p.value.matrix) = 0

    for (i in 1:(n - 1)) {
     for (j in (i + 1):n) {
      tmp = cor.test(corr.data[, i], corr.data[, j])
      p.value.matrix[i, j] = p.value.matrix[j, i] = tmp$p.value
              }
      colnames(p.value.matrix) = rownames(p.value.matrix) =
                          colnames(corr.numeric.adjusted))
        }
    return(p.value.matrix)
    }
```

Plot of Correlations (Corrplot)

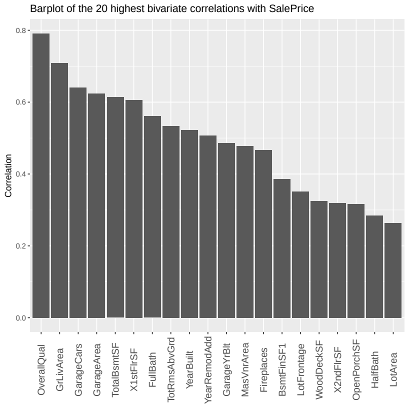


- A cut-off value of 0.3 was used: Variables with no correlation over 0.3 do not show up in the plot
- A significance level of 0.05 was used to test the correlations (crosses indicate no significance)

Barplot: Code

```
corr.barplot = function(numb.corr = 36) {
        correlation.vars     = names(numeric.data) %in% c("SalePrice")
        correlation.data     = numeric.data[!correlation.vars]
        correlations         = vector(length =
                              length(names(correlation.data)))
        names(correlations) = names(correlation.data)
        for (i in names(correlation.data)) {
            correlations[i] = cor(numeric.data$SalePrice,
                                  correlation.data[i], use = "pairwise.
              complete.obs")}

        y.plotting = correlations[order(abs(correlations),
           decreasing = TRUE)][1:numb.corr]
        x.plotting = names(y.plotting)
        names(y.plotting) = NULL
        df = data.frame(x.plotting, y.plotting)
        df$x.plotting = factor(df$x.plotting, levels =
           df[order(abs(df$y.plotting), decreasing = TRUE), "x.plotting
             "])

        ggplot(data = df, aes(x.plotting, y.plotting), fill = as.factor
           (x.plotting)) + geom_bar(stat = "identity") +
            theme(axis.title.x = element_blank(), axis.text.x = element
              _text(angle = 90, vjust = 0.5, size = 12)) +
            ylab("Correlation") + ggtitle(paste("Barplot of the", numb.
              corr, "highest bivariate correlations with SalePrice",
            sep = ""))
}
```

Ordered barplot for the correlations with SalePrice

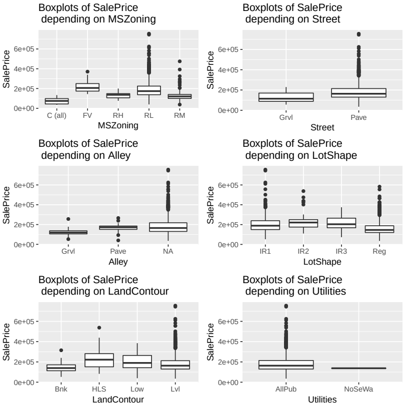


- The plot shows the 20 numeric variables that have the highest absolut correlation with SalePrice
- Out of 36 variables in the dataset only 10 have a correlation higher then 0.5

Boxplot: Code

```
boxplot.target = function(categoric) {
        categoric.x = data[, categoric]
        plot.data = as.data.frame(cbind(data$SalePrice, categoric.x))
        plot.data[[2]] = as.factor(plot.data[[2]])
        levels(plot.data[[2]]) = levels(categoric.x)

        ggplot(plot.data, aes(x = categoric.x, y = V1)) +
          geom_boxplot() +
          labs(title = paste("Boxplots of SalePrice",
            "\n", "depending on", categoric, sep = " "),
             x = categoric, y = "SalePrice")
}
```

Boxplots of SalePrice

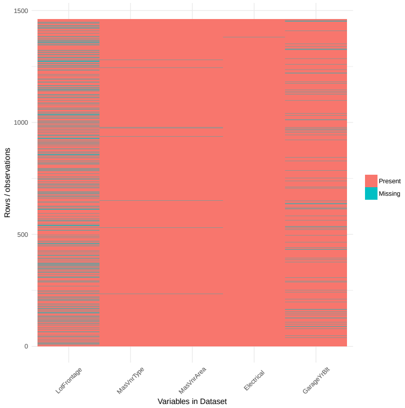


- Example of boxplots for SalePrice based on the levels of categorical variables
- Differences in median and spread across levels indicate a possibly good predictor variable

Quantlet: Data Preprocessing

- aim of this quantlet: ensure, that the data quality is sufficent for all types of models that will be used
- handling of missing data, including imputation
- reduction of dimensionality
  - merging of factor levels
  - principal component analysis
- detetection and handling of outliers

Missing Data



- based on data description, most NA's have a meaning (None/Other)
- imputation of remaining NA's
  - numeric variables: median
  - factor variables: mode

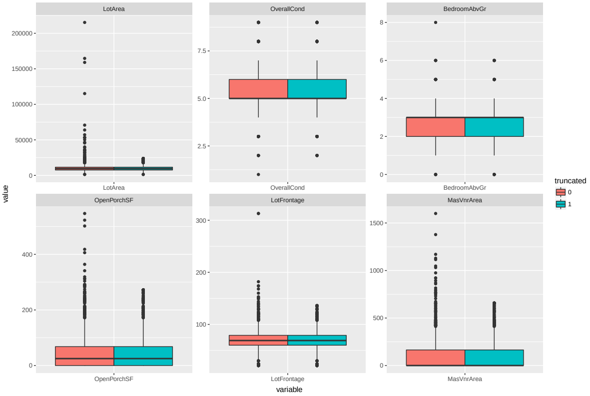
Missing Data: Code

```
Mode = function(x) {
      ux = unique(x)
      ux[which.max(tabulate(match(x, ux)))]
}
impute.mode = function(x){
      nas       = is.na(x)
      x[nas] = Mode(x[!nas])
      as.factor(x)
}
categoric.imputed = as.data.frame(sapply(categoric.
  data, impute.mode))
```

Outlier Handling: Code

```
outlier.count = function(x){
      low =as.numeric(quantile(x)[2] - IQR(x)*3)
      high=as.numeric(IQR(x)*3 + quantile(x)[4])
      sum(x >= high | x <= low)
}
outlier.truncate = function(x){
       low =as.numeric(quantile(x)[2] - IQR(x)*3)
       high=as.numeric(IQR(x)*3 + quantile(x)[4])
       x[x < low] = low
       x[x > high] = high
       return(x)
}
df_outlier_trunc = as.data.frame(sapply(df.temp.
  numeric, outlier.truncate))
```

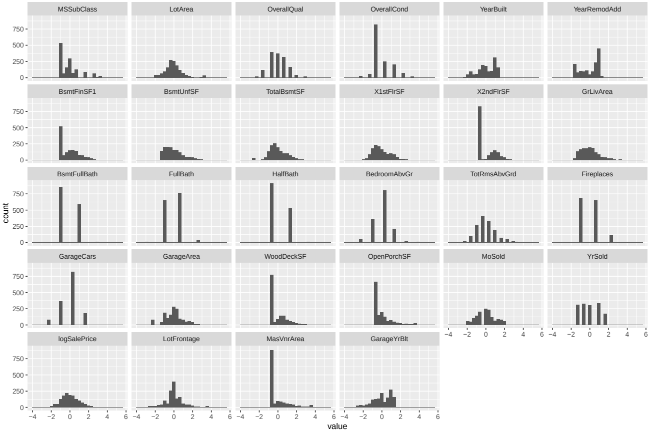
Outlier: Before and after truncation



Dimensionality Reduction: Code

```
single.factors = function(data) {
    for(var in names(data)) {
        if(is.factor(data[[var]])){
            tbl = table(data[[var]])
            ren = names(tbl)[tbl <=  20]
            levels(data[[var]])[levels(data[[var]]) %in% ren] = "Other"
            tbl     = table(data[[var]])
            tbl_sum = sum(tbl < 20)
            if(nlevels(data[[var]]) < 3 & tbl_sum >= 1  ) data[[var]] = NA
        }
    }
    return(data)
}
```

Preprocessing: Summary



Quantlet: Regression Models

- aim of this quantlet: select appropriate variables
- four selection procedures
  - Backwards Selection based on significance
  - Forward Selection based on AIC
  - LASSO
  - Ridge

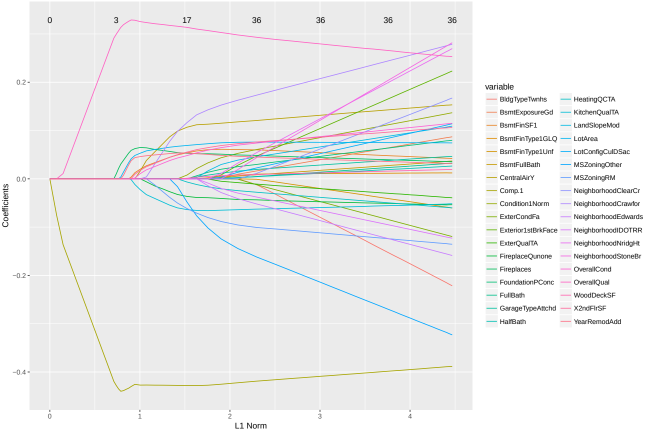
Significance Selection: Code

```
sign.select = function(dframe, y) {
pvals = 1, z      = 1, i      = 1
vars  = names(dframe)
vars  = vars[!vars %in% y]
while(z>0){
     df.lm = cbind(dframe[vars], dframe[y])
     lm1   = lm(formula(paste(y,"~ . ")) ,data=df.lm)
     pvals = summary(lm1)$coefficients[,4]
     pvals = pvals[!names(pvals) %in% "(Intercept)"]
     vars  = names(pvals[pvals<0.05])
     #z = sum(pvals>0.05),      i = i+1
     if(i==300){
          warning("... No signif. Vars in Data Set?")
          break
          }
}
return(vars)
}
```

LASSO: Code

```
lm.penal = function(type, x, y) {
    if (type == "lasso") {          alpha = 1
    } else if ( type == "ridge") {alpha = 0
    } else stop("type must be ridge or lasso")
    cvfit      = cv.glmnet(x, y, alpha = alpha, nfolds = 10)
    fit        = predict(cvfit,newx=x,s="lambda.1se")
    rsq        = 1 - sum((fit - y)^2)/sum(y^2)
    c          = coef(cvfit, s = "lambda.1se")
    inds       = which(c != 0)
    var = row.names(c)[inds]
    vars.sele = var[!variables %in% "(Intercept)"]
    coeftable = data.frame(var = var, coeff = c[inds], stringsAsFactors = F)
    output    = list(vars.sele, coeftable, c, cvfit, fit, rsq)
}
```

LASSO Results



Regression Models Results (Excerpt)

.variable	.stat	Sign. Selec.	AIC Selec.	Lasso	Ridge
(Intercept)	Estimate	-0.849***	-0.914***	-0.116	-0.229
LotArea	Estimate	0.112***	0.062***		-0.019
OverallQual	Estimate	0.245***	0.201***	0.076	0.057
OverallCond	Estimate	0.089***	0.104***	0.293	0.118
YearRemodAdd	Estimate	0.049***	0.044***	0.074	0.047
BsmtUnfSF	Estimate	-0.066***	-0.165***	0.045	0.052
X2ndFlrSF	Estimate	0.208***	0.215***	0.06	0.054
LotShapeOther	Estimate	-0.415***	-0.345***		0.034
LotConfigCulDSac	Estimate	0.141***	0.136***	0.048	0.043
LandSlopeMod	Estimate	0.119**	0.137***	0.009	0.029
NeighborhoodClearCr	Estimate	0.184**	0.225***		0.026
NeighborhoodCrawfor	Estimate	0.323***	0.291***		0.003
NeighborhoodEdwards	Estimate	-0.153***	-0.199***		-0.004
NeighborhoodIDOTRR	Estimate	-0.209***	-0.199***		0.033
ElectricalOther	Estimate				0.051
ElectricalSBrkr	Estimate				0.035
	N	1168	1168	1168	1168
	R2	0.895	0.914	0.88	0.89
	Number Vars	33	77	36	177

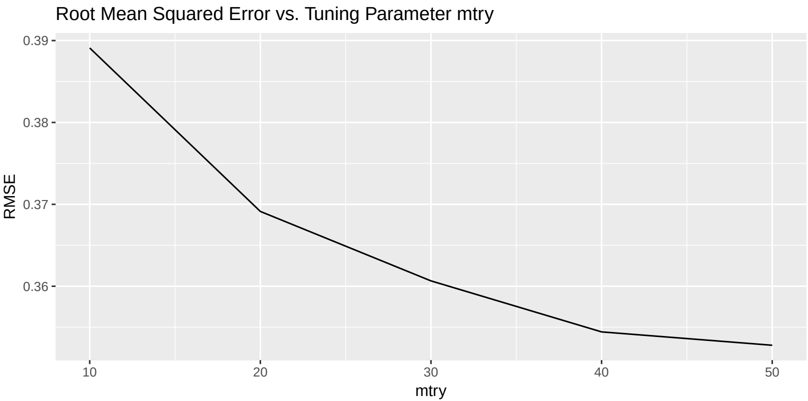
Random Forest

- Applied Random Forest Regression Model to the data
- Collection of Regression Trees, where the predicted value is the average value in the leaf node.
- Results are averaged across all Regression Trees in the Forest
- Tuning Parameter: *mtry* - Number of randomly selected variables per decision tree
- Model is tuned using the *caret* package and 5 Fold Cross Validation in a parallelized fashion using *doParallel*

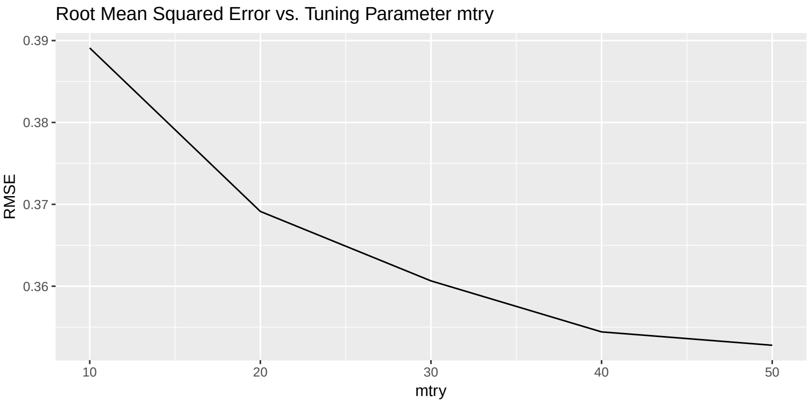
Random Forest: Code

```
CrossValidation = trainControl(method = "repeatedcv"
  , number = nfolds, p = ptrain, repeats = nrepeats)
rfGrid       = expand.grid(mtry = mtry.tuning)
cores        = max(c(detectCores() - 1, 0))
cl           = makeCluster(cores)
registerDoParallel(cl)
set.seed(123)
rftuned = train(log(SalePrice) ~ ., data = df,
  method = "rf", importance = TRUE, trControl =
  CrossValidation, verbose = TRUE,
    tuneGrid = rfGrid)
stopCluster(cl)
```

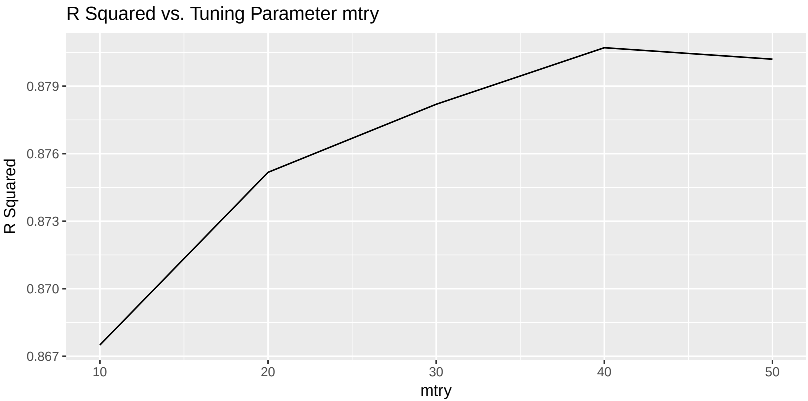
Random Forest Tuning Results - RMSE



Random Forest Tuning Results - RMSE



Random Forest Tuning Results - R Squared



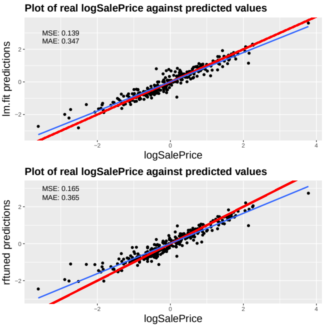
Quantlet: Model Comparison

- aim of this quantlet: after building and training all the models on the training data the goal is to measure how they perform on new data (=test data)
- computing different measures and show results graphically
  - Mean Squared Error
  - Bias
  - real Vs. predicted plots

Real Vs. Predicted: Code

```
predictions.lm = predict(lm.fit, newdata = test)
df.lm.fit      = data.frame(cbind(test$logSalePrice, predictions.lm ))

lm.plot = ggplot(df.lm.fit, aes(test$logSalePrice, predictions.lm)) + geom_point() + geom_segment(x = -4, y = -4, xend = 4, yend = 4, color = "red", size = 1.3) +
  stat_smooth(method = "lm", se = FALSE) +
  labs(title = "Plot of real logSalePrice against predicted values",
  x = "logSalePrice", y = "lm.fit predictions") + theme(axis.title = element_text(size = 16), plot.title = element_text(size = 16,
  face = "bold")) +
  annotate("text", label = paste("MSE:", comparison.result["MSE", "lm"], sep = " "), x = -3, y = 3) +
  annotate("text", label = paste("MAE:", comparison.result["MAE", "lm"], sep = " "), x = -3, y = 2.5)
```

Real Vs. Predicted Plots



- If the house prices were predicted perfectly, all points would lie on the red line
- The blue line comes from an OLS regression: The better the predictions, the more blue and red lines should align

Performance measure example: MSE Code

```
model.mse = function(model, test.data = test) {
    if (class(model)[1] %in% c("train", "lm")) {
        pred = predict(model, newdata = test.data)
        mse  = (1/ncol(test.data)) * sum((pred - test.data$logSalePrice)^2)
    } else {
        pred = predict(model, newx = as.matrix(test.data[!names(test.data) %in% "logSalePrice"]), s = "lambda.1se")
        mse  = (1/ncol(test.data)) * sum((pred - test.data$logSalePrice)^2)
    }
    return(mse)
}
```

Result comparing all models

	lm	fwd	lasso	ridge	gbm	rf
MSE	0.14	0.12	0.19	0.19	0.14	0.16
MSEtrain	0.70	0.57	0.94	0.88	0.48	0.14
MAE	0.35	0.33	0.40	0.40	0.37	0.36
BIAS	0.02	0.03	0.03	0.03	0.04	0.03
RSQ	0.91	0.92	0.88	0.88	0.91	0.90

- In comparison the self built linear model and the forward algorithm have the best results amongst all implemented models

Conclusions

- In the presented examples, the more complex models appear to overfit the training data
- All models perform reasonably well on the test set
- Linear models work better which suggests that the relationship between log House Prices and the input variables are indeed linear
- Follow our repository at: https://github.com/ckoopmann/SPL-HousePriceRegression