Data Mining
Homework 10

Karl Ehatäht

April 13, 2016

1. First off, let's take a look at the points:

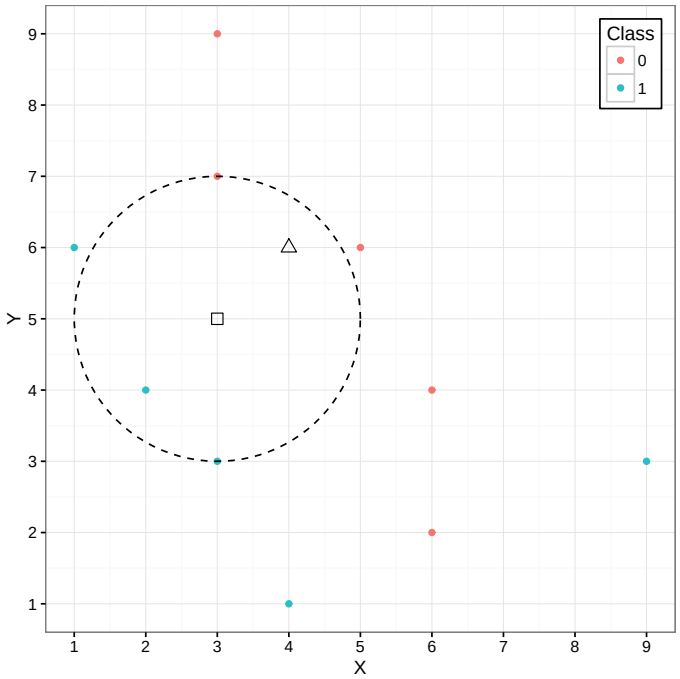

Figure 1: We try to classify points $A = (3, 5)$ (square) and $B = (4, 6)$ (triangle). If $k = 2$ for point A, then there are two candidates that fit to nearest neighbor criteria: $\{(2, 4), (3, 7)\}$ and $\{(2, 4), (3, 3)\}$.

Code:

```
knn.uri = paste0(curr_dir, "data/knn.txt")
knn.data = as.data.frame(read.csv(file = knn.uri, header = F))
knn.data[,"V1"] = NULL
colnames(knn.data) = c("x", "y", "class")
knn.data[,"class"] = as.logical(knn.data[,"class"])
ggplot2::ggplot(data = knn.data, mapping = ggplot2::aes(x = x, y = y)) +
  ggplot2::geom_point(mapping = ggplot2::aes(color = class)) +
  ggplot2::scale_x_continuous(breaks = 1:10) +
  ggplot2::scale_y_continuous(breaks = 1:11) +
  ggplot2::geom_point(data = data.frame(x = 3, y = 5), shape = 0, size = 3) +
  ggplot2::geom_point(data = data.frame(x = 4, y = 6), shape = 2, size = 3) +
  ggplot2::xlab("X") +
  ggplot2::ylab("Y") +
  ggplot2::theme_bw() +
  ggplot2::scale_color_discrete(name = "Class", breaks = c(F, T), labels = c(0, 1)) +
  ggplot2::theme(legend.justification = c(1, 1),
                 legend.position = c(1, 1),
                 legend.background = ggplot2::element_rect(color = "black")) +
  ggplot2::annotate("path", x = 3 + 2 * cos(seq(0, 2 * pi, length.out = 100)),
                            y = 5 + 2 * sin(seq(0, 2 * pi, length.out = 100)),
                    linetype = 2)
ggplot2::ggsave(filename = "knn.eps", width = 6, height = 4.5)
knn.data[,"class"] = 1 * knn.data[,"class"]
```

If we try to classify point $A = (3, 5)$ (c.f. fig. 1) with $k = 2$ (i.e. two nearest neighbors), then there are actually two candidates that fit to nearest neighbor criteria: $\{(2, 4), (3, 7)\}$ and $\{(2, 4), (3, 3)\}$.

To resolve that, I derived the following algorithm:

- if we select k nearest neighbors (NNs) and the $(k+1)$-th point closest to the given point we are about to classify is farther than the k-th NN, then the algorithm proceeds as usual:
  - count how many points out of k NN belong to a class and divide by k: this is the probability that the point belongs to that class;
  - the class with the highest probability wins (and thus we say that the point belongs to that class);
  - however, if there is a tie, we cannot really tell to which class the point belongs;
- however, if the k-th NN is as far as the $(k+1)$-th one, we do the following:
  - find all points that are at the same distance as the k-th one. It might be the case that $(k-m)$-th and $(k+n)$-th are at the same distance (m, $n > 0$), so that there $n-m+1$ points at the same distance as the k-th one.
  - generate all combinations $\binom{n-m+1}{k-m+1}$ of such points, i.e. we select all possible combinations that lie on the border of k-th NN;
  - for each combination we calculate the probabilities as described above;
  - then we average over the probabilities and report the class that had the highest probability.

To illustrate this, let choose point A and set $k = 2$. We see that $m = k = 2$ and $n = k + 1 = 3$, thus there are two $n - m + 1 = 2$ points that lie on the border. With the first combination we'll get

$$\{(2,\,4) \in \{1\},\, (3,\,3) \in \{1\}\} \;\Rightarrow\; P(\{1\}) = 1\,,\; P(\{0\}) = 0$$

and with the second combination we'll get

$$\{(2,\,4) \in \{1\},\, (3,\,7) \in \{0\}\} \;\Rightarrow\; P(\{1\}) = 0.5\,,\; P(\{0\}) = 0.5\,.$$

By averaging we'll get $P(\{1\}) = 0.75$ and $P(\{0\}) = 0.25$, so class 1 wins. Fair enough.

So I wrote the whole algorithm (c.f. listing 1), which produced the following results:

	A		B	
k	$P(\{0\})$	$P(\{1\})$	$P(\{0\})$	$P(\{1\})$
1	0	1	1	0
2	0.75*	0.25*	1	0
3	0.667	0.333	0.833*	0.167*

Table 1: Results of k NN algorithm. The values with an asterisk are calculated with custom method of combinations (as described above; c.f. listing 1). There were two such cases where two points were at the same border as the k-th NN.

It seems, that R's built-in `class::knn` chooses all points at the border. For instance,

```
class::knn(knn.data.noclass, data.frame(x = 3, y = 5), knn.data$class, 2, prob = T)
```

reports $P(\{1\}) = 0.667$ for point A with $k = 2$, which is the same as with $k = 3$.

Code:

```
knn.binary = function(df, x, y, k, debug = F)
{
  stopifnot(nrow(df) >= k)
  max_prob = 0
  max_class = NULL

  df.copy = df
  df.copy$d = knn.data = sqrt((knn.data$x - x)**2 + (knn.data$y - y)**2)
  df.copy = BBmisc::sortByCol(df.copy, "d", asc = T)

  l = k
  m = k
  while(l < nrow(df.copy) &
        isTRUE(all.equal(df.copy$d[l], df.copy$d[l + 1]))) l = l + 1
  while(m > 0 &
        isTRUE(all.equal(df.copy$d[l], df.copy$d[m - 1]))) m = m - 1
  nof_classes = -1

  if(l == k)
  {
    nn = df.copy[1:k,]
    classes = unique(nn$class)
    nof_classes = length(classes)
    probs = data.frame(matrix(ncol = nof_classes, nrow = 1, 0))
    for(cl in classes)
    {
      prob = nrow(subset(nn, class == cl)) / k
      probs[1, as.character(cl)] = prob
      cat(paste0("Class ", cl, ": ", round(prob, digits = 3)), "\n")
      if(prob > max_prob)
      {
        max_prob = prob
        max_class = cl
      }
    }
  }
  else # l > k
  {
    cat(paste("Encountered", l - m + 1, "items at the same distance for k =", k), "\n")
    nn.basic = if(m > 0) df.copy[1:m - 1,] else df.copy[0,]
    nn.add = df.copy[m:l,]
    combs = t(combn(l - m + 1, k - m + 1))
    classes = unique(df.copy[1:l,"class"])
    nof_classes = length(classes)
    probs = data.frame(matrix(ncol = nof_classes, nrow = nrow(combs), 0))
    colnames(probs) = as.character(classes)

    if(debug)
    {
      print(nn.basic)
      print(nn.add)
    }

    for(i in 1:nrow(combs))
    {
      nn.ext = nn.basic
      for(j in 1:ncol(combs))
        nn.ext = rbind(nn.ext, nn.add[combs[i,j],])
      for(cl in classes)
        probs[i, as.character(cl)] = nrow(subset(nn.ext, class == cl)) / k
      if(debug) print(nn.ext)
    }

    probs.means = colMeans(probs)
    if(debug) print(probs.means)

    for(cl in classes)
    {
      prob = probs.means[as.character(cl)]
      cat(paste0("Class ", cl, ": ", round(prob, digits = 3), "\n"))
      if(prob > max_prob)
      {
        max_prob = prob
        max_class = cl
      }
```

```
    }
  }
  if(isTRUE(all.equal(max_prob, 1 / nof_classes)) & nof_classes > 1)
    cat("Can't tell to which class the point should belong to\n")
  else
    cat(paste0("Given point (", x, ", ", y, ") likely belongs to class ", max_class), "\n")
}
knn.binary(knn.data, 3, 5, 2)
knn.binary(knn.data, 4, 6, 1)
```

Listing 1: Custom k NN algorithm that combines the values that are at the same distance as the k-th NN.

2. The code:

```
diab.url = paste0("https://archive.ics.uci.edu/ml/machine-learning-databases/",
                  "pima-indians-diabetes/pima-indians-diabetes.data")
diab.data = as.data.frame(read.csv(file = url(diab.url)))
colnames(diab.data) = c("nof_pregnant", "glucose_concentration", "blood_pressure",
                        "skin_thickness", "insulin", "bmi", "pedigree_function", "age", "class")
set.seed(123)
train.test.ratio = 0.8
nof_train = round(nrow(diab.data) * train.test.ratio)
train_idx = sample(nrow(diab.data), nof_train)
logit.train = diab.data[train_idx,]
logit.test = diab.data[-train_idx,]

logit.model = glm(formula = as.factor(class) ~ ., data = logit.train, family = "binomial")
summary(logit.model)
logit.pvalues = as.data.frame(coef(summary(logit.model)))
logit.pvalues = BBmisc::sortByCol(logit.pvalues, "Pr(>|z|)")
round(logit.pvalues, digits = 3)

logit.test$pred = predict(newdata = logit.test, logit.model, type = "response")
logit.test$pred_binary = ifelse(logit.test$pred <= 0.5, 0, 1)
logit.confu = table(actual = logit.test$class, prediction = logit.test$pred_binary)
logit.confu
logit.accuracy = sum(diag(logit.confu)) / sum(logit.confu)
logit.precision = logit.confu[4] / sum(logit.confu[,2])
logit.recall = logit.confu[4] / sum(logit.confu[2,])
logit.rec.prec = c(logit.recall, logit.precision)
logit.fscore = 1 / mean(1 / logit.rec.prec)
```

Here is the coefficient table sorted by P-values:

Predictor	Estimate	Standard error	z value	$P(>\|z\|)$	Significance code
(Intercept)	-8.890	0.829	-10.725	0.000	***
Plasma glucose concentration	0.036	0.004	8.572	0.000	***
Body mass index	0.104	0.018	5.897	0.000	***
Diabetes pedigree function	1.071	0.337	3.183	0.001	**
Number of times pregnant	0.112	0.037	3.049	0.002	**
Diastolic blood pressure	-0.015	0.006	-2.536	0.011	*
Age	0.014	0.011	1.230	0.219	
2-Hour serum insulin	-0.001	0.001	-0.716	0.474	
Triceps skin fold thickness	-0.003	0.008	-0.332	0.740	

Table 2: Results of simple logit model, where the response class is „tested positive for diabetes".

As can be seen from above table, the plasma glucose concentration really impact the odds ratio of having a diabetes – the higher it is (coefficient is positive[1]), the more likely a person has diabetes. As for the diabetes pedigree function, it's less significant, but significant nevertheless, because its associated p-value is below 0.05. Higher value of this function means higher odds of having a diabetes.

[1]Positive coefficient β_i in a logit model corresponds to odds ratio of e^{β_i}. Thus, if it's positive, then the odds of response variable is greater than 1.

We deduced it from the (two-tail) p-value of z-test (special case of Wald test), which gives us the probability for the coefficient having such value, assuming null hypothesis (i.e. coefficient is zero). If the z-score (maximum likelihood estimation over standard error) corresponding to a coefficient is far away from the mean of normal distribution $\mathcal{N}(0,\,1)$, then it's likely significant. The probability for it to happen is p-value (twice the area under $\mathcal{N}(0,\,1)$ from the absolute value of z-score to infinity). The precise difference between t-test (used in linear regression) and z-test still remains elusive to me... Running the model with a testing set (20% of the original dataset) outputs the following confusion matrix:

		Prediction	
		0	1
Actual	0	84	13
	1	22	34

Table 3: Confusion matrix for the logit model using the testing set. „1" means „tested positive for diabetes" and „0" means the opposite.

Corresponding scores are:

- accuracy: 0.771;
- precision: 0.723;
- recall: 0.607;
- F_1-score: 0.660.

3. Code:

```
knn.train = diab.data[train_idx,]
knn.test = diab.data[-train_idx,]
knn.train.class = knn.train$class
knn.test.class = knn.test$class
knn.train$class = NULL
knn.test$class = NULL
knn.pred.1 = as.numeric(as.vector(class::knn(knn.train, knn.test, knn.train.class, 1)))
knn.pred.3 = as.numeric(as.vector(class::knn(knn.train, knn.test, knn.train.class, 3)))

printf = function(s, ...) cat(paste0(sprintf(s, ...)), '\n')
knn_confu = function(pred, actual)
{
  tp = sum(pred & actual)
  tn = sum(!pred & !actual)
  fp = sum(pred & !actual)
  fn = sum(!pred & actual)
  confu = table(actual = actual, prediction = pred)
  acc = (tp + tn) / (tp + tn + fp + fn)
  recall = tp / (tp + fn)
  prec = tp / (tp + fp)
  prec.recall = c(prec, recall)
  fscore = 1 / mean(1 / prec.recall)
  printf("Precision: %.3f\nRecall: %.3f\nAccuracy: %.3f\nF1 score: %.3f",
         prec, recall, acc, fscore)
  return(confu)
}
knn.confu.1 = knn_confu(knn.pred.1, knn.test.class)
knn.confu.3 = knn_confu(knn.pred.3, knn.test.class)
```

I used $k = 1$ and $k = 3$, and the same 20/80 split for testing/training (actually, it's the same training and testing set as in the previous exercise). This produced the following confusion tables:

		Prediction	
		0	1
Actual	0	73	24
	1	24	32

Table 4: Confusion matrix for the kNN model with $k = 1$.

		Prediction	
		0	1
Actual	0	76	21
	1	28	28

Table 5: Confusion matrix for the kNN model with $k = 3$.

And the measures summed up (including the results of previous exercise):

Model	Precision	Recall	Accuracy	F_1-score
Logit	0.723	0.607	0.771	0.660
kNN, $k = 1$	0.571	0.571	0.686	0.571
kNN, $k = 3$	0.571	0.500	0.680	0.533

From the above table it is evident that our logit model has the highest predictive power (all scores higher than that of kNN models). As for the comparison between kNN models with $k = 1$ and $k = 3$, the former seems to be better (all but precision is higher).

4. & 7. We used the regular 20/80 split again, and also kept the factor variables in model construction. Root mean square errors (RMSE) was calculated for each model and for both training and testing samples. The results are depicted in the following figure:

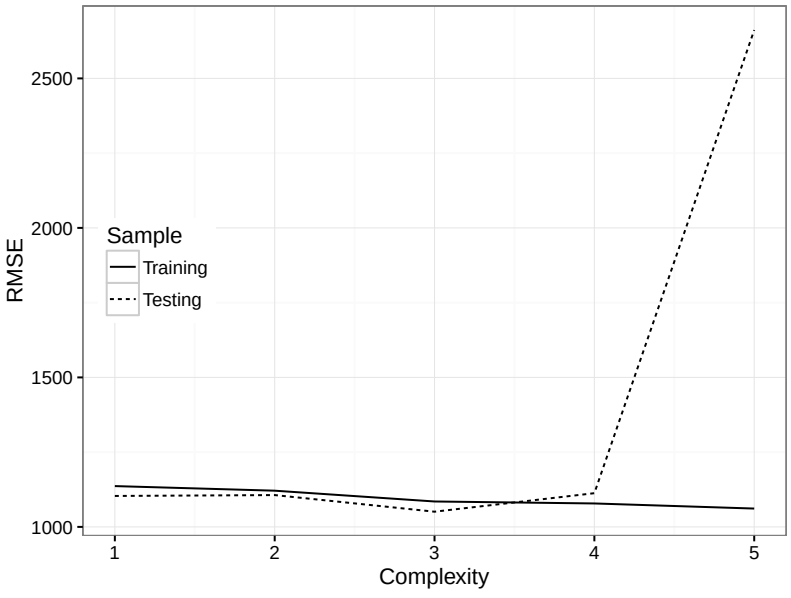

Figure 2: RMSE vs model complexity for bot testing sample (random 20% of the original sample) and training sample (80%). C.f. table 6 for explicit values.

It is obvious that the 4th and 5th model overfit the data: the training RMSE decreases (as expected), but the testing RMSE shoots up. The „best" model out of all these is the 3rd one, b/c the minimum testing RMSE is achieved with this model. As for the underfitting... it's hard to tell whether we underfit or not. Since we included all features into our regression model, the initial model was already complex enough.

Instead of trying out just ridge or lasso regression models, I experimented with general method – elastic net regularization – where both lasso and ridge are special cases. The objective function subject to minimization is

$$\min_{\{\beta_0,\boldsymbol{\beta}\}} \frac{1}{N}\sum_{i=1}^{N}\left(y_i-\beta_0-\mathbf{x}_i^T\boldsymbol{\beta}\right)^2+\lambda\left[(1-\alpha)||\boldsymbol{\beta}||_2^2+\alpha||\boldsymbol{\beta}||_1\right] ,$$

where β_0 is intercept, $\boldsymbol{\beta}$ the coefficients we want to find, y_i and $\mathbf{x}_i$ the i-th response and predictor (out of N training points), λ is complexity/regularization parameter and α mixing parameter. Setting α to 1 penalizes only L_1 norm (lasso), whereas $\alpha = 0$ penalizes L_2 norm (ridge). Here is the plot for the most complex (5th) model, where we vary α and calculate corresponding RMSE.

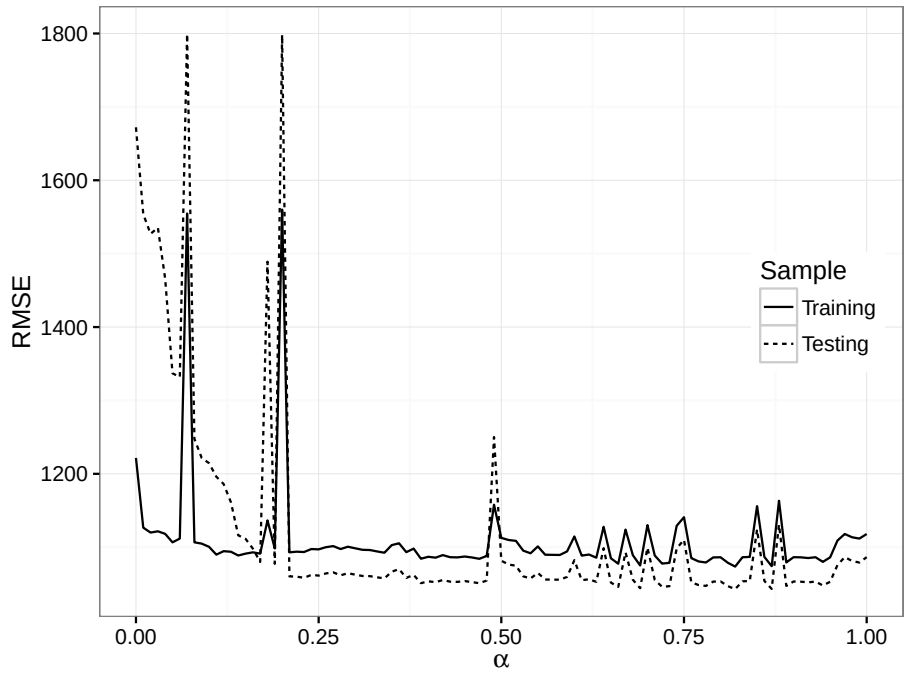

Figure 3: RMSE vs penalty parameter α for testing set in the most complex (5th) model. The model should be cross-validated against α in order to find optimal value for it (the method I used, `glmnet::cv.glmnet`, does 10-fold CV for finding optimal λ subject to MSE, given α). However, with a single trial on the whole testing set, global minima is achieved at $\alpha = 0.82$ (w/ corresponding testing RMSE of 1042.9). I turns out that this choice outperforms all simpler models as well (c.f. table 6 and table 7).

Since it is hard to distinguish smaller values of RMSE in 2 and 3, here are tables that explicitly state RMSE for each model:

Model complexity	Training RMSE	Testing RMSE
1	1136.5	1103.5
2	1121.1	1106.4
3	1085.0	1050.9
4	1078.4	1112.7
5	1061.4	2661.7

Table 6: RMSE values for different model complexities. See 2 for a visual.

Elastic net α, 5th model	Training RMSE	Testing RMSE
0.0	1221.7	1672.2
0.1	1100.5	1214.5
0.2	1560.0	1798.1
0.3	1098.5	1062.6
0.4	1087.0	1053.6
0.5	1112.7	1081.6
0.6	1114.7	1082.4
0.7	1130.1	1099.4
0.8	1086.3	1053.5
0.9	1086.4	1053.3
1.0	1118.0	1086.5

Table 7: Elastic net model, RMSE values (c.f. fig. 3 for visual guide). The global minimum is at $\alpha = 0.82$ w/ testing RMSE of 1042.9.

Code:

```
########### DIAMONDS #############

require("ggplot2")
set.seed(123)
data(diamonds)
train.test.ratio = 0.8
nof_train = round(nrow(diamonds) * train.test.ratio)
train_idx = sample(nrow(diamonds), nof_train)
diamonds.train = diamonds[train_idx,]
diamonds.test = diamonds[-train_idx,]

remove_factors = F
if(remove_factors)
{
  diamonds.test$cut = NULL
  diamonds.test$color = NULL
  diamonds.test$clarity = NULL
  diamonds.train$cut = NULL
  diamonds.train$color = NULL
  diamonds.train$clarity = NULL
}

diamond.default.formula = function(exceptions)
  colnames(diamonds.train)[!colnames(diamonds.train) %in% c(exceptions, "price")]
diamond.formula = function(exceptions, newvars)
  as.formula(paste("price ~",
    paste(c(diamond.default.formula(exceptions), newvars),
          collapse = " + ")))
diamonds.train.rmse = c()
diamonds.test.rmse = c()

diamonds.model.1 = lm(formula = price ~ ., data = diamonds.train)
diamonds.pred.1 = predict(newdata = diamonds.test, diamonds.model.1)
diamonds.train.rmse.1 = sqrt(mean(diamonds.model.1$residuals**2))
diamonds.test.rmse.1 = sqrt(mean((diamonds.test$price - diamonds.pred.1)**2))
diamonds.train.rmse = c(diamonds.train.rmse, diamonds.train.rmse.1)
diamonds.test.rmse = c(diamonds.test.rmse, diamonds.test.rmse.1)

diamonds.model.2 = lm(formula = diamond.formula(c(), c("I(carat^2)", "I(depth^2)")),
                      data = diamonds.train)
diamonds.pred.2 = predict(newdata = diamonds.test, diamonds.model.2)
diamonds.train.rmse.2 = sqrt(mean(diamonds.model.2$residuals**2))
diamonds.test.rmse.2 = sqrt(mean((diamonds.test$price - diamonds.pred.2)**2))
diamonds.train.rmse = c(diamonds.train.rmse, diamonds.train.rmse.2)
diamonds.test.rmse = c(diamonds.test.rmse, diamonds.test.rmse.2)

diamonds.model.3 = lm(formula = diamond.formula(c(), c("I(carat^2)", "I(depth^2)",
                                                       "I(carat^3)", "I(depth^3)")),
```

```
                      data = diamonds.train)
diamonds.pred.3 = predict(newdata = diamonds.test, diamonds.model.3)
diamonds.train.rmse.3 = sqrt(mean(diamonds.model.3$residuals**2))
diamonds.test.rmse.3 = sqrt(mean((diamonds.test$price - diamonds.pred.3)**2))
diamonds.train.rmse = c(diamonds.train.rmse, diamonds.train.rmse.3)
diamonds.test.rmse = c(diamonds.test.rmse, diamonds.test.rmse.3)

diamonds.model.4 = lm(formula = diamond.formula(c(), c("I(carat^2)", "I(depth^2)",
                                                       "I(carat^3)", "I(depth^3)",
                                                       "I(x^2)", "I(y^2)", "I(z^2)")),
                      data = diamonds.train)
diamonds.pred.4 = predict(newdata = diamonds.test, diamonds.model.4)
diamonds.train.rmse.4 = sqrt(mean(diamonds.model.4$residuals**2))
diamonds.test.rmse.4 = sqrt(mean((diamonds.test$price - diamonds.pred.4)**2))
diamonds.train.rmse = c(diamonds.train.rmse, diamonds.train.rmse.4)
diamonds.test.rmse = c(diamonds.test.rmse, diamonds.test.rmse.4)

diamonds.model.5 = lm(formula = diamond.formula(c(), c("I(carat^2)", "I(depth^2)",
                                                       "I(carat^3)", "I(depth^3)",
                                                       "I(carat^4)", "I(depth^4)",
                                                       "I(x^2)", "I(y^2)", "I(z^2)")),
                      data = diamonds.train)
diamonds.pred.5 = predict(newdata = diamonds.test, diamonds.model.5)
diamonds.train.rmse.5 = sqrt(mean(diamonds.model.5$residuals**2))
diamonds.test.rmse.5 = sqrt(mean((diamonds.test$price - diamonds.pred.5)**2))
diamonds.train.rmse = c(diamonds.train.rmse, diamonds.train.rmse.5)
diamonds.test.rmse = c(diamonds.test.rmse, diamonds.test.rmse.5)

diamonds.rmses = data.frame(train.rmse = diamonds.train.rmse,
                            test.rmse = diamonds.test.rmse)
diamonds.rmses$complexity = as.numeric(rownames(diamonds.rmses))
diamonds.rmses.melt = reshape2::melt(diamonds.rmses, id.vars = "complexity")
ggplot2::ggplot(data = diamonds.rmses.melt,
                mapping = ggplot2::aes(x = complexity, y = value)) +
  ggplot2::geom_line(mapping = ggplot2::aes(color = variable)) +
  ggplot2::scale_color_discrete(name = "Sample",
                                breaks = c("train.rmse", "test.rmse"),
                                labels = c("Training", "Testing")) +
  ggplot2::xlab("Complexity") +
  ggplot2::ylab("RMSE") +
  ggplot2::theme_bw() +
  ggplot2::theme(legend.justification = c(0, 0.5),
                 legend.position = c(0, 0.5))
if(remove_factors) {
  ggplot2::ggsave("rmse_wo_factors.eps", width = 4, height = 3)
} else {
  ggplot2::ggsave("rmse_w_factors.eps", width = 6, height = 4.5)
}
round(diamonds.rmses, digits = 1)

################ RIDGE, LASSO, ELASTIC NET ###############
diamonds.train.x = model.matrix(
  diamond.formula(c(), c("I(carat^2)", "I(depth^2)",
                         "I(carat^3)", "I(depth^3)",
                         "I(carat^4)", "I(depth^4)",
                         "I(x^2)", "I(y^2)", "I(z^2)")),
  diamonds.train
)
diamonds.train.y = as.matrix(diamonds.train$price, ncol = 1)
diamonds.test.x = model.matrix(
  diamond.formula(c(), c("I(carat^2)", "I(depth^2)",
                         "I(carat^3)", "I(depth^3)",
                         "I(carat^4)", "I(depth^4)",
                         "I(x^2)", "I(y^2)", "I(z^2)")),
  diamonds.test
)

set.seed(123)
diamonds.elnet.df = data.frame(alpha = numeric(0), train.rmse = numeric(0), test.rmse = numeric(0))
for(i in seq(0, 1, 0.01))
{
  printf("Going with alpha = %.2f", i)
  diamonds.elnet.cv = glmnet::cv.glmnet(
    x = diamonds.train.x,
    y = diamonds.train.y,
    type.measure = "mse",
    family = "gaussian", alpha = i
```

```
  )
  diamonds.elnet.test.predict = predict(
    diamonds.elnet.cv,
    s = diamonds.elnet.cv$lambda.1se,
    newx = diamonds.test.x
  )
  diamonds.elnet.train.predict = predict(
    diamonds.elnet.cv,
    s = diamonds.elnet.cv$lambda.1se,
    newx = diamonds.train.x
  )
  diamonds.elnet.test.rmse = sqrt(mean((diamonds.test$price - diamonds.elnet.test.predict)**2))
  diamonds.elnet.train.rmse = sqrt(mean((diamonds.train$price - diamonds.elnet.train.predict)**2))
  diamonds.elnet.df = rbind(diamonds.elnet.df, c("alpha" = i,
                                                 "train.rmse" = diamonds.elnet.train.rmse,
                                                 "test.rmse" = diamonds.elnet.test.rmse))
  colnames(diamonds.elnet.df) = c("alpha", "train.rmse", "test.rmse")
}
diamonds.elnet.df.molten = reshape2::melt(diamonds.elnet.df, id.vars = "alpha")
ggplot2::ggplot(data = diamonds.elnet.df.molten,
                mapping = ggplot2::aes(x = alpha, y = value)) +
  ggplot2::geom_line(mapping = ggplot2::aes(linetype = variable)) +
  ggplot2::xlab(latex2exp::TeX("$\\alpha$")) +
  ggplot2::ylab("RMSE") +
  ggplot2::scale_linetype_discrete(name = "Sample",
                                   breaks = c("train.rmse", "test.rmse"),
                                   labels = c("Training", "Testing")) +
  ggplot2::theme_bw() +
  ggplot2::theme(legend.justification = c(1, 0.5),
                 legend.position = c(1, 0.5))
ggplot2::ggsave(filename = "rmse_alpha_elnet.eps", width = 6, height = 4.5)

round(diamonds.elnet.df[seq(1, 101, by = 10),], digits = 1)
round(diamonds.elnet.df[which.min(diamonds.elnet.df$test.rmse),], digits = 1)
```

5. Done.