Machine Learning Summary

Maximilian Pfundstein

2018-12-28

Contents

1 Terms

1.1 Bagging

1.2 Bias-Variance-Trade-Off

1.3 Curse of Dimensionality

1.4 Degrees of Freedom

1.5 Generative vs Discriminative

1.6 Kernel Trick

1.7 Mean Squared Error (MSE)

1.8 Misclassification Rate

1.9 No Free Lunch Theorem

1.10 Ordinary Least Squares Regression (OLS)

1.11 Parametric and Non-Parametric Models

1.12 Types of Learning

1.12.1 Active Learning

1.12.2 Reinforcement Learning

1.12.3 Semi-supervised

1.12.4 Supervised Learning

1.12.5 Unsupervised Learning

1.13 Under- and Overfitting

2 Useful Code Snippets

2.1 Confusion Matrix and Misclassification Rate

```
confusion_matrix_train = as.matrix(table(spambase_predict_train, train$Spam))

error_rate =
  1 - sum(diag(confusion_matrix_train)/sum(confusion_matrix_train))
print(error_rate)
```

2.2 Custom Error Function

Gam and Tree

2.3 Feature Plot

```
ggplot(statedata, aes(x = MET, y = EX)) +  geom_point() + geom_smooth()
```

```
ggplot(adb_errors) +
  geom_line(aes(x = n, y = error_rate_training,
                colour = "AdaBoost Training"), linetype = "dashed") +
  geom_point(aes(x = n, y = error_rate_training), colour = "orange") +

  geom_line(aes(x = n, y = error_rate_validation,
                colour = "AdaBoost Validation")) +
  geom_point(aes(x = n, y = error_rate_validation), colour = "red") +

  geom_line(aes(x = n, y = error_rate_training,
                colour = "Random Forest Training"),
            data = rf_errors, linetype = "dashed") +
  geom_point(aes(x = n, y = error_rate_training),
             colour = "blue", data = rf_errors) +

  geom_line(aes(x = n, y = error_rate_validation,
                colour = "Random Forest Validation"), data = rf_errors) +
  geom_point(aes(x = n, y = error_rate_validation),
             colour = "steelblue2", data = rf_errors) +
  labs(title = "Random Forest and AdaBoost", y = "Error Rate",
       x = "Number of Forests", color = "Legend") +
  scale_color_manual(values = c("orange", "red", "blue", "steelblue2"))
```

2.4 Histogram

```
hist(pruned_tree_plot_dataframe$residual,
     col="orange", main = "Histogram of the Residuals", xlab = "Residuals", breaks = 20)
```

2.5 Importing Data

2.5.1 Splitting Data

```
data(spam)
n = dim(spam)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.4))
train_spam = spam[id,]
id1 = setdiff(1:n, id)
set.seed(12345)
id2 = sample(id1, floor(n*0.3))
val_spam = spam[id2,]
```

```
id3 = setdiff(id1,id2)
test_spam = spam[id3,]
```

2.5.2 .csv

```
emails = read.csv2("data.csv", fileEncoding = "ISO-8859-1", sep = ";")
emails$Conference = as.factor(emails$Conference)
```

```
stations = read.csv("stations.csv", encoding = "UTF-8")
temps = read.csv("temps50k.csv", encoding = "UTF-8")
st = merge(stations, temps, by="station_number")
```

2.5.3 .xls and .xlsx

```
creditscoring = read_excel("creditscoring.xls")
creditscoring$good_bad = as.factor(creditscoring$good_bad)
```

2.6 RMarkdown Template

2.6.1 Header

```
author: "Maximilian Pfundstein"
date: "2018-12-28"
output:
  pdf_document:
    toc: true
    toc_depth: 3
    number_sections: true
  html_document:
    df_print: paged
    toc_float: true
    number_sections: true
```

2.6.2 Setup

```
{r setup, include = FALSE}
knitr::opts_chunk$set(echo = FALSE, cache = TRUE, include = TRUE, eval = FALSE)
```

2.6.3 Source Code as Appendix

```
{r, ref.label=knitr::all_labels(), echo = TRUE, eval = FALSE, results = 'show'}
```

3 Models

3.1 Bayesian Classification

3.2 Boosting

3.2.1 AdaBoost

```
c_adaBoost = blackboost(formula = c_formula,
                        data = train_spambase,
                        family = AdaExp(),
                        control=boost_control(mstop=i))
```

3.2.2 Forward Stagewise Additive Modeling

3.2.3 Gradient Boosting

3.3 Elastic Net

```
# Elastic Net

# Predictor Variables. The -1 removes the intercept component
x_test = as.matrix(train_emails[, -ncol(train_emails)])
x_val = as.matrix(val_emails[, -ncol(val_emails)])

# Outcome variable
y_test = train_emails$Conference
y_val = val_emails$Conference

model_elastic_net = cv.glmnet(x = x_test, y = y_test, alpha = 0.5,
                              family = "binomial")

# Support vector machine with "vanilladot" kernel.
# Vanilladot means "Linear Kernel Function"
# set scale = FALSE to prevent "variable(s) `' constant. Cannot scale data.""
model_svm = ksvm(x = x_test, y = y_test, kernel = "vanilladot", scale = FALSE)

# Predictions
pred_val_elastic = predict(model_elastic_net, newx = x_val, type = "class")
pred_val_svm = predict(model_svm, x_val, type = "response")

matrix_elastic = table(y_val, t(pred_val_elastic))
matrix_svm = table(y_val, pred_val_svm)

kable(matrix_elastic)
kable(matrix_svm)

summary(model_elastic_net)
plot(model_elastic_net)
```

3.4 Generalized Additive Model (GAM)

Somehow uses cross validation.

```
gam_model = gam(formula = Mortality ~ Year + s(Week,
                k=length(unique(influanza$Week))), family = gaussian(),
                data = influanza, method="GCV.Cp")

print(gam_model)
summary(gam_model)
```

Probabilistic Model:

$$Model \sim \mathcal{N}(\beta_{year} * X_{Year} + S_{week} * X_{week} + \alpha, \sigma^2)$$

```
Model \sim \mathcal{N}(\beta_{year} * X_{Year} +S_{week} * X_{week} + \alpha,  \sigma^2)
```

3.4.1 Useful GAM Plots

```
time = influanza$Time
observed = influanza$Mortality
predicted = gam_model$fitted.values

mortality_values = data.frame(time, observed, predicted)

ggplot(mortality_values) +
  geom_line(aes(x = time, y = observed,
                colour = "Observed Mortality")) +
  geom_line(aes(x = time, y = predicted,
                colour = "Predicted Mortality")) +
  labs(title = "Observed vs Predicted Mortality", y = "Mortality",
       x = "Time", color = "Legend") +
  scale_color_manual(values = c("blue", "orange"))

ggplot() +
  geom_line(aes(x=influanza$Week,y=influanza$Mortality,
                          colour=as.factor(influanza$Year))) +
  labs(x='Week', y='Mortality', colour='Year',
       title='Mortality by Week and by Year')
```

3.4.2 Penalty Factor, Deviance, Degress of Freedom

```
penalties = seq(from = 0 , to = 20, by = 0.1)
index = 1

fitted_gam_with_penalties = data.frame()

for (penalty in penalties) {
  gam_model = gam(formula = Mortality ~ Year +
                    s(Week, sp = penalty, k = length(unique(influanza$Week))),
                  family = gaussian(), data = influanza, method="GCV.Cp")
```

```
  fitted_gam_with_penalties = rbind(fitted_gam_with_penalties,
        data.frame(list(penalty = penalty,
                        deviance = gam_model$deviance,
                        df = sum(influence(gam_model)))))
}

## Deviance VS Penalty
ggplot(fitted_gam_with_penalties) +
  geom_line(aes(x = fitted_gam_with_penalties$penalty,
                y = fitted_gam_with_penalties$deviance,
                colour = "Deviance/Penalty")) +
  labs(title = "Deviance VS Penalty", y = "Deviance",
       x = "Penalty", color = "Legend") +
  scale_color_manual(values = c("blue", "orange"))

## Penalty Factors vs Degrees of Freedom
ggplot(fitted_gam_with_penalties) +
  geom_line(aes(x = fitted_gam_with_penalties$deviance,
                y = fitted_gam_with_penalties$df,
                colour = "Degrees of Freedom/Deviance")) +
  labs(title = "Deviance VS Degrees of Freedom", y = "Degrees of Freedom",
       x = "Deviance/DF", color = "Legend") +
  scale_color_manual(values = c("blue", "orange"))

## Create Models for High and Low penalties
gam_model_low_pen = gam(formula = Mortality ~ Year + s(Week,
                  k=length(unique(influanza$Week)), sp = 0.1),
                  family = gaussian(), data = influanza, method="GCV.Cp")
gam_model_high_pen =
  gam(formula = Mortality ~ Year + s(Week, k=length(unique(influanza$Week)),
                                     sp = 20),
                  family = gaussian(), data = influanza, method="GCV.Cp")

high_low_df = data.frame()
high_low_df = rbind(high_low_df,
                    list(mortality = influanza$Mortality,
                         mortality_low = fitted(gam_model_low_pen),
                         mortality_high = fitted(gam_model_high_pen),
                         date = influanza$Time))

ggplot(high_low_df) +
  geom_line(aes(x = date, y = mortality, colour = "Mortality")) +
  geom_line(aes(x = date, y = mortality_low, colour = "Mortality (Low: 0.1)")) +
  geom_line(aes(x = date, y = mortality_high, colour = "Mortality (High: 20)")) +
  labs(title = "Mortalities vs Time", y = "Mortality",
       x = "Time", color = "Legend") +
  scale_color_manual(values = c("blue", "orange", "violet"))

## Observed vs Predicted Mortality
ggplot(mortality_values) +
  geom_line(aes(x = time, y = observed,
                colour = "Observed Mortality")) +
  geom_line(aes(x = time, y = predicted,
```

```
                colour = "Predicted Mortality")) +
  labs(title = "Observed vs Predicted Mortality", y = "Mortality",
       x = "Time", color = "Legend") +
  scale_color_manual(values = c("blue", "orange"))
```

3.4.3 Residuals

```
gam_model$residuals
```

Family parameter mgcv

3.5 Generalized Linear Model (GLM)

```
spambase_model = glm(Spam ~ ., data = train, family = "binomial")
spambase_predict_train =
  predict(object = spambase_model, newdata = train, type = "response")
spambase_predict_train =
  apply(as.matrix(spambase_predict_train), c(1),
        FUN = function(x) return(x > 0.5))
```

- Response Poisson distributed
- Canonical Link (log) is used for regression
- probabilistic expression for the fitted model

3.6 K-Nearest Neighbour (KNN)

```
kknn_model_train = kknn(formula = Spam ~ ., train = train, test = train, k = 30)
y_hat_train =
  apply(as.matrix(kknn_model_train$fitted.values), c(1),
        FUN = function(x) return(x > 0.5))
```

3.7 Lasso

Ridge Regression has the problem, that even with high $\lambda's$ the coefficients will never be zero, this would only be the case for $lim_{\lambda\to\infty}$. This time the *Shrinkage Penalty* is

$$\lambda \sum_j |\beta_j|$$

Due to the fact that high $\lambda's$ can set the coefficients to zero, LASSO perform a *Variable Selection*. This is what can bee seen in the plot, with increasing $\ln(\lambda)$ more and more variables are set to 0.

```
covariates_lasso = scale(tecator_data[,2:(ncol(tecator_data)-3)])
response_lasso = tecator_data$Fat

glm_model_lasso = glmnet(as.matrix(covariates_lasso),
                         response_lassoe, alpha = 1, family="gaussian")
plot(glm_model_lasso, xvar="lambda")
```

3.7.1 Cross-Validation

```
lasso_vc = cv.glmnet(y = response_lasso, x = covariates_lasso,
                     alpha = 1, lambda = seq(from = 0, to = 10, by = 0.01))
plot(lasso_vc)
print(paste(sep = "", "Best lambda score: ", lasso_vc$lambda.min))
```

3.8 Least Absolute Deviation Regression

3.9 Linear Regression

```
model = lm(formula = c_formula, data = tecator_data)
```

3.10 Logistic Regression

- Equation of decision boundary
- Plot classified data and decision boundary
- GLM
- Custom Classification

3.11 Naive Bayes

```
naiveBayesModel = naiveBayes(good_bad ~ ., data = train)
```

3.12 Nearest Shrunken Centroid Classification (NSCC)

```
rownames(train_emails) = 1:nrow(train_emails)
x_train = t(train_emails[,-ncol(train_emails)])
y_train = train_emails[[ncol(train_emails)]]
mydata_train = list(x=x_train ,y=as.factor(y_train), geneid =
                      as.character(1:nrow(x_train)),
              genenames = rownames(x_train))

nsc_model = pamr.train(mydata_train, threshold=seq(0, 4, 0.1))
nsc_model_cv = pamr.cv(nsc_model, mydata_train)

nsc_model_cv$threshold[which.min(nsc_model_cv$error)]

genes = pamr.listgenes(nsc_model, mydata_train, threshold = 1.3)

pamr.plotcen(nsc_model, mydata_train, threshold = 1.3)

print(nsc_model_cv)
pamr.plotcv(nsc_model_cv)

# Amount of features
nrow(genes)
```

```r
# 10 Most contributing features
kable(colnames(emails)[as.numeric(genes[1:10, 1])])

# Confusion Matrix
rownames(val_emails) = 1:nrow(val_emails)
x_val = t(val_emails[,-ncol(val_emails)])
y_val = val_emails[[ncol(val_emails)]]
#mydata_val = list(x=x_val ,y=as.factor(y_val),
# geneid = as.character(1:nrow(x_val)),
#               genenames = rownames(x_val))

pred_train = pamr.predict(nsc_model, newx = x_train, threshold = 2.5)
pred_val = pamr.predict(nsc_model, newx = x_val, threshold = 2.5)

matrix_train = table(train_emails$Conference, pred_train)
matrix_val = table(val_emails$Conference, pred_val)

kable(matrix_train)
kable(matrix_val)

error_train_nsc = 1 - sum(diag(matrix_train)/sum(matrix_train))
error_val_scn = 1 - sum(diag(matrix_val)/sum(matrix_val))

print(paste("Error Train:", error_train_nsc))
print(paste("Error Validation:", error_val_scn))
```

3.13 Neural Networks (NN)

Limitations and Types

3.13.1 Backpropagation Implementation

3.13.2 Implementation

Naive Bayes that uses nonparametric density estimation method Hint: density() function does not have predict() function but it evaluates predictions on a given grid. To make prediction for a vector of new values, you may call density() several times and specify one prediction point at a time, i.e. interval [a,b]=[x(i),x(i)].

3.13.3 Library

3.13.4 Regularization

3.14 Partial Least Squares Regression (PLS)

3.15 Quadratic Discriminant Analysis

3.16 Ridge Regression

λ, also called the *Shrinkage Parameter*, penalizes the coefficients to decrease the number of coefficients to prevent overfitting. This is due to the fact that the *Shrinkage Penalty*

$$\lambda \sum_j \beta_j^2$$

is added to the term of calculating the estimates $\hat{\beta}^R$. If $\lambda = 0$ we're back to least squares estimates.

```
covariates_ridge = scale(tecator_data[,2:(ncol(tecator_data)-3)])
response_ridge = tecator_data$Fat

glm_model_ridge = glmnet(as.matrix(covariates_ridge),
                         response_ridge, alpha = 0, family="gaussian")
plot(glm_model_ridge, xvar="lambda")
```

3.17 StepAIC (AIC)

```
model = lm(formula = c_formula, data = tecator_data)
model.stepAIC = stepAIC(model, direction = c("both"), trace = FALSE)
summary(model.stepAIC)
```

3.18 Support Vector Machines (SVM)

```
# Our three models depending on
model_svm_05 =
  ksvm(type ~ ., train_spam, kernel = "rbfdot",
       kpar = list(sigma = 0.05), C = 0.5)

model_svm_05_prediction = predict(model_svm_05, newdata = val_spam)

cm_svm_05 = table(val_spam$type, model_svm_05_prediction)

error_svm_05 = 1 - sum(diag(cm_svm_05)/sum(cm_svm_05))
```

3.19 Trees

```
decisionTree_deviance = tree(good_bad ~ ., data = train, split = "deviance")
decisionTree_gini = tree(good_bad ~ ., data = train, split = "gini")

prediction_deviance_train =
  predict(decisionTree_deviance, newdata = train, type = "class")
prediction_deviance_test =
  predict(decisionTree_deviance, newdata = test, type = "class")

summary(decisionTree_deviance)
summary(decisionTree_gini)
```

3.19.1 Optimal Tree

```
trainScore = rep(0, 15)
testScore = rep(0, 15)

for(i in 2:15) {
  prunedTree = prune.tree(decisionTree_deviance, best = i)
  pred = predict(prunedTree, newdata = valid, type = "tree")
  trainScore[i] = deviance(prunedTree)
  testScore[i] = deviance(pred)
}

## Add one as we trim the first index
optimalTreeIdx = which.min(testScore[-1]) + 1
optimalTreeScore = min(testScore[-1])

print(optimalTreeIdx)
print(optimalTreeScore)
```

3.19.2 Plot Deviance

```
plot(2:15, trainScore[2:15], type = "b", col = "orange", ylim = c(250,650),
     main = "Tree Depth vs Training/Test Score", ylab = "Deviance",
     xlab = "Number of Leaves")
points(2:15, testScore[2:15], type = "b", col = "blue")
legend("topright", legend = c("Train (orange)", "Test (blue)"))
```

3.19.3 Prune a tree and print it

```
optimalTree = prune.tree(decisionTree_deviance, best = optimalTreeIdx)
plot(optimalTree)
text(optimalTree, pretty = 1)
title("Optimal Tree")
```

3.19.4 Regression Tree with Deviance

```
# Create the model
reg_tree = tree(EX ~ MET, data = statedata, control =
                  tree.control(nobs = nrow(statedata), minsize = 8))

# Use cross validation
cross_val_reg_tree = cv.tree(reg_tree)

# Plot the deviance of the sizes
plot(cross_val_reg_tree)
```

3.19.5 Pruned tree with residuals

```
# Let's create the pruned the with best set to 3 and get it's prediction
pruned_tree = prune.tree(reg_tree, best = 3)
```

```
pruned_tree_prediction = predict(pruned_tree, newdata = statedata, type = "vector")

# We create a data.frame to save our values to make it easier to plot the data
pruned_tree_plot_dataframe =
  data.frame(statedata$MET, statedata$EX, pruned_tree_prediction,
             pruned_tree_prediction-statedata$EX)
names(pruned_tree_plot_dataframe) = c("met", "orignal_ex", "predicted_ex", "residual")

# Lets first plot the pruned tree
plot(pruned_tree)
text(pruned_tree, pretty = 1)
title("Optimal Tree with best = 3")

# Lets create a plot with the real and predictes values and highlight the
# residuals
ggplot(pruned_tree_plot_dataframe) +
  geom_point(aes(x = pruned_tree_plot_dataframe$met,
                 y = pruned_tree_plot_dataframe$orignal_ex),
             color = "black") +
  geom_point(aes(x = pruned_tree_plot_dataframe$met,
                 y = pruned_tree_plot_dataframe$predicted_ex),
             color = "darkblue") +
  geom_segment(mapping=aes(x=pruned_tree_plot_dataframe$met,
                           y=pruned_tree_plot_dataframe$orignal_ex,
                           xend=pruned_tree_plot_dataframe$met,
                           yend=pruned_tree_plot_dataframe$predicted_ex),
               color = "red", linetype = "dotted") +
  labs(title = "Original Data, Fitted Data and Residuals", y = "EX",
       x = "MET", color = "Legend")
```

3.19.6 Random Forest

```
c_randomForest =
    randomForest(formula = c_formula, data = train_spambase, ntree = i)
```

Decision, Blackboost, CART

4 Feature Reduction

4.1 Independent Component Analysis (ICA)

```
set.seed(12345)

ica_model = fastICA(X = nir_spectra_copy, n.comp = 2, alg.typ = "parallel",
                    fun = "logcosh", alpha = 1, method = "R", row.norm = FALSE,
                    maxit = 200, tol = 0.0001, verbose = FALSE)

W_dash = ica_model$K %*% ica_model$W

plot(W_dash[,1], main= "Column 1")
```

```r
plot(W_dash[,2], main= "Column 2")

ggplot(as.data.frame(W_dash)) +
  geom_point(aes(x = W_dash[,1],
                 y = W_dash[,2]),
             color = "orange") +
    labs(title = "Feature 1 vs Feature 2", y = "Feature 2",
       x = "Feature 1", color = "Legend")
```

4.2 Linear Discriminant Analysis (LDA)

lda() in package mass

4.2.1 Implementation

4.2.2 Library

4.3 Principal Component Analysis (PCA)

```r
# Copy to not modify the original dataset
nir_spectra_copy = nir_spectra
nir_spectra_copy$Viscosity = c()

# PCA
res = prcomp(nir_spectra_copy)

# Eigenvalues
lambda = res$sdev^2

# Proportion of variation
kable(head(sprintf("%2.3f",lambda/sum(lambda)*100)), caption = "Variance for each Feature")

# Plot
screeplot(res, main = "Variances for each Feature")

# PC1 vs PC2
ggplot(as.data.frame(res$x)) +
  geom_point(aes(x = res$x[,1],
                 y = res$x[,2]),
             color = "orange") +
    labs(title = "PC1 vs. PC2", y = "PC2",
       x = "PC1", color = "Legend")
```

4.3.1 Trace Plots

```r
U = res$rotation
plot(U[,1], main = "Traceplot, PC1")
plot(U[,2], main = "Traceplot, PC2")
```

4.3.2 Kernel PCA

kpca in kernlab

4.4 Regularized Discriminant Analysis

5 Miscellaneous

5.1 Benjamin-Hochberg Algorithm

```
# Orignal Data Set: emails
# H0: Feature has effect on Conference
# Ha: Feature has no effect on Conference

# 1) Calculate p-values
# 2) Assign ranks and sort
# 3) BH Critical Value
# 4) Find critical value and select all p < score

q_value = 0.05

## 1)

feature_names = c()
p_values = c()

# For each data point calculate the p_value

for (i in 1:(ncol(emails)-1)) {
  colname = colnames(emails)[i]
  c_formula = paste(sep = "", colname, " ~ Conference")
  p_value = t.test(as.formula(c_formula), data = emails,
                   alternative="two.sided")

  feature_names = c(feature_names, colname)
  p_values = c(p_values, p_value$p.value)
}

## 2)

# Sorting
bh_entries = data.frame(feature_names, p_values)
bh_entries = bh_entries[order(bh_entries$p_values, decreasing = FALSE),]
rownames(bh_entries) = NULL

## 3)

# Define the function for the CV-Score
getCV_BH = function(i, m, Q) {
  return(i/m*Q)
}
```

```r
cv_scores = c()

for (j in 1:nrow(bh_entries)) {
  current_val = getCV_BH(as.numeric(rownames(bh_entries)[j]),
                         nrow(bh_entries), q_value)
  cv_scores = c(cv_scores, current_val)
}

#cv_col = apply(bh_entries, 1, function(x)
#getCV_BH(as.numeric(rownames(bh_entries)), nrow(bh_entries), q_value))
bh_entries = data.frame(bh_entries, cv_scores)

## 4)
## Find all rows where p_values < cv_scores
bh_entries = bh_entries[bh_entries$p_values < bh_entries$cv_scores,]

kable(bh_entries)
```

5.2 Bootstrapping

5.2.1 Non-Parametric Bootstrap with Confidence Bands

```r
# We take the function given from the slides and adjust to the tree
# computing bootstrap samples
f_non_p_bootstrap = function(data, ind) {

  # First take the subsample
  data1 = data[ind,]

  # Now create a tree with the same hyperparameters from that subsample
  tree_model = tree(EX ~ MET, data = data1,
                    control = tree.control(nrow(data),minsize = 8))
  tree_model_pruned = prune.tree(tree_model, best = 3)

  # Use that model to predict on the real data
  prediction = predict(tree_model_pruned, newdata = data)
  return(prediction)
}

# Lets create the Bootstrap (again taken from slides)
res = boot(statedata, f_non_p_bootstrap, R = 1000)

# Confidence Bands using envelope
ci_non_p_bootstrap = envelope(res)
ci_non_p_bootstrap_df = as.data.frame(t(ci_non_p_bootstrap$point))
names(ci_non_p_bootstrap_df) = c("upper_bound", "lower_bound")
pruned_tree_plot_dataframe =
  data.frame(pruned_tree_plot_dataframe, ci_non_p_bootstrap_df)

# Plot the data
ggplot(pruned_tree_plot_dataframe) +
```

```
  geom_point(aes(x = pruned_tree_plot_dataframe$met,
                 y = pruned_tree_plot_dataframe$orignal_ex),
             color = "black") +
  geom_ribbon(aes(x = pruned_tree_plot_dataframe$met,
                  ymin = ci_non_p_bootstrap_df$lower_bound,
                  ymax = ci_non_p_bootstrap_df$upper_bound),
              alpha = 0.4, fill = "orange", color = "orange3") +
    labs(title = "Confidence Bands (non-parametric)", y = "EX",
       x = "MET", color = "Legend")
```

5.2.2 Parametric Bootstrap with Confidence Bands

```
# Again we take the sample from the slides and adjust it to our needs
# 1) Compute value mle
# 2) Write function ran.gen that depends on data and mle and which generates
# new data
# 3) Write function statistic that depend on data which will be generated by
# ran.gen and should return the estimator

## 1)
mle = pruned_tree

## 2)
rng = function(data, mle) {
  data1 = data.frame(EX=data$EX, MET=data$MET)
  n = length(data$EX)
  #generate new Price
  # summary needed to access the residuals
  data1$EX = rnorm(n, predict(mle, newdata=data1), sd(summary(mle)$residuals))
  return(data1)
}

## 3) f_non_p_bootstrap + distribution N
f_p_bootstrap = function(data) {

  # The index is not needed any more as we don't take a sub-sample

  # Now create a tree with the same hyperparameters from that subsample
  tree_model = tree(EX ~ MET, data = data,
                    control = tree.control(nrow(data), minsize = 8))
  tree_model_pruned = prune.tree(tree_model, best = 3)

  # Use that model to predict on the real data
  prediction = predict(tree_model_pruned, newdata = data)

  return(prediction)
}

# Bootstrap
res2 = boot(statedata,
            statistic = f_p_bootstrap, R=1000, mle=mle,
            ran.gen=rng, sim="parametric")
```

```r
# Confidence Bands using envolope
ci_p_bootstrap = envelope(res2)
ci_p_bootstrap_df = as.data.frame(t(ci_p_bootstrap$point))
names(ci_p_bootstrap_df) = c("upper_bound", "lower_bound")
pruned_tree_plot_dataframe_p =
  data.frame(pruned_tree_plot_dataframe, ci_p_bootstrap_df)

# Plot the data
ggplot(pruned_tree_plot_dataframe_p) +
  geom_point(aes(x = pruned_tree_plot_dataframe_p$met,
                 y = pruned_tree_plot_dataframe_p$orignal_ex),
             color = "black") +
  geom_ribbon(aes(x = pruned_tree_plot_dataframe_p$met,
                  ymin = ci_p_bootstrap_df$lower_bound,
                  ymax = ci_p_bootstrap_df$upper_bound),
              alpha = 0.4, fill = "orange", color = "orange3") +
    labs(title = "Confidence Bands (parametric)", y = "EX",
       x = "MET", color = "Legend")
```

5.2.3 Parametric Bootstrap with Prediction Bands

```r
# Prediction Bands

# from slides
f_p_bootstrap_pb = function(data) {

  # The index is not needed any more as we don't take a sub-sample

  # Now create a tree with the same hyperparameters from that subsample
  tree_model = tree(EX ~ MET, data = data,
                    control = tree.control(nrow(data), minsize = 8))
  tree_model_pruned = prune.tree(tree_model, best = 3)

  # Use that model to predict on the real data
  prediction = predict(tree_model_pruned, newdata = data)

  # Add the rnrom to the prediction
  prediction_normal = rnorm(nrow(data), prediction, sd(summary(mle)$residual))

  return(prediction_normal)
}

# Bootstrap
res3 = boot(statedata, statistic = f_p_bootstrap_pb,
            R=1000, mle=mle, ran.gen=rng, sim="parametric")

# Confidence Bands using envolope
pb_p_bootstrap = envelope(res3)
pb_p_bootstrap_df = as.data.frame(t(pb_p_bootstrap$point))
names(pb_p_bootstrap_df) = c("upper_bound", "lower_bound")
pruned_tree_plot_dataframe_p_pb =
  data.frame(pruned_tree_plot_dataframe, pb_p_bootstrap_df)
```

```
# Plot the data
ggplot(pruned_tree_plot_dataframe_p_pb) +
  geom_point(aes(x = pruned_tree_plot_dataframe_p_pb$met,
                 y = pruned_tree_plot_dataframe_p_pb$orignal_ex),
             color = "black") +
  geom_ribbon(aes(x = pruned_tree_plot_dataframe_p_pb$met,
                  ymin = pb_p_bootstrap_df$lower_bound,
                  ymax = pb_p_bootstrap_df$upper_bound), alpha = 0.4,
              fill = "orange", color = "orange3") +
    labs(title = "Prediction Bands (parametric)", y = "EX",
       x = "MET", color = "Legend")
```

5.3 Cross-Validation

5.3.1 Cross Validation Plot

5.3.2 K-Fold

```
c_cross_validation = function(k = 5, Y, X) {

  if (!is.numeric(X) && ncol(X) == 0) {
    y_hat = mean(Y)
    return(mean((y_hat-Y)^2))
  }

  Y = as.matrix(Y)
  X = as.matrix(X)
  X = cbind(1, X)

  # Create a list of 5 matrices with the appropiate
  # size (these will hold the subsets)
  X_subsets = list()
  Y_subsets = list()

  # We fill the list entries with the subsets
  for (i in 1:k) {
    percentage_marker = nrow(X)/k
    start = floor(percentage_marker*(i-1)+1)
    end = floor(percentage_marker*i)
    X_subsets[[i]] = X[start:end,]
    Y_subsets[[i]] = Y[start:end]
  }

  # Now we take one matrix at a time for training and
  # everything else as the testing
  scores = 0
  for (i in 1:k) {

    ## Initial
    X_train = matrix(0, ncol = ncol(X))
    Y_train = c()
```

```
    # Get validation and training data
    current_subset_X = X_subsets[-i]
    current_subset_Y = Y_subsets[-i]
    for (j in 1:(length(X_subsets)-1)) {
      X_train = rbind(X_train, current_subset_X[[j]])
      Y_train = c(Y_train, current_subset_Y[[j]])
    }

    # Because of R
    X_train = X_train[-1,]

    # Model
    betas =
      as.matrix((solve(t(X_train) %*% X_train)) %*% t(X_train) %*% Y_train)

    ## Select the training data and transform them to
    # one matrix X_test and one vector Y_test
    X_val = X_subsets[[i]]
    Y_val = Y_subsets[[i]]

    ## Now we get our y_hat and y_real. y_real is
    # only used to clarify the meaning
    y_hat = as.vector(X_val %*% betas)
    y_real = Y_val

    ## Get MSE and add to the scores list
    scores = c(scores, mean((y_hat - y_real)^2))

  }
  # Return the mean of our scores
  scores = scores[-1]
  return(mean(scores))
}

c_best_subset_selection = function(Y, X) {

  # Shuffle X and Y via indexes
  ids = sample(x = 1:nrow(X), nrow(X))
  X = X[ids,]
  Y = Y[ids]

  # Get all combinations
  comb_matrix = matrix(0, ncol = ncol(X))
  for (i in c(1:(2^(ncol(X))-1))) {
    comb_matrix =
      rbind(comb_matrix, tail(rev(as.numeric(intToBits(i))), ncol(X)))
  }

  results = c()

  # Do cross validation for each feature set
  for (j in 1:nrow(comb_matrix)) {
    comb = as.logical(comb_matrix[j,])
```

```
    feature_select = X[,comb]
    res = c_cross_validation(5, Y, feature_select)
    results = c(results, res)
  }
  models = matrix(results, ncol = 1)
  models = cbind(models, comb_matrix)

  # Add column with the sum of the features for plotting
  feature_sum = c()
  for (k in 1:nrow(comb_matrix)) {
    row_sum = sum(comb_matrix[k,])
    feature_sum = c(feature_sum, row_sum)
  }
  models = as.data.frame(cbind(feature_sum, models))
  colnames(models)[1:2] = c("Sum", "Score")
  print(ggplot(models, aes(x = Sum, y = Score, colour = factor(feature_sum))) +
    geom_point())
   stat_summary(fun.y = min, colour = "red", geom = "point", size = 5)
  return(models[min(models[,2]) == models[,2],])
}
```

5.3.3 Nested Cross Validation

5.3.4 Two-Fold

5.4 EM-Algorithm

Let's have a look at the mathematical equations and how we can derive our formulas for the matrix multiplication from that. Formulas without a source are either taken from the lecture slides or derived by previous formulas.

The first step is to calculate Z. We will divide that in first calculating Bx which contains $Bernoulli(x|\mu_k)$ and afterwards calculating $p(x)$ which we can use to calculate Z. The formulas (left side) will be assigned to a letter which you will find in the source code.

$$Bernoulli(x|\mu_k) = B_x = \prod_i \mu_{k_i}^{x_i}(1-\mu_{k_i})^{1-x_i}$$

```
Bernoulli(x|\mu_k) = B_x = \prod_{i} \mu^{x_i}_{k_i} (1 - \mu_{k_i})^{1-x_i}
```

For using matrix multiplication we need to get rid of the product and the exponents, so we use ln on both sides:

$$\ln(B) = \sum \ln(\mu_{k_i}^{x_i})x_i + \sum \ln(1-\mu_{x_i})(1-x_i)$$

```
\ln(B) = \sum \ln (\mu^{x_i}_{k_i}) x_i + \sum \ln (1-\mu_{x_i}) (1 - x_i)
```

Now let's get rid of the ln on the left side:

$$B_x = e^{\sum \ln(\mu_{k_i}^{x_i})x_i + \sum \ln(1-\mu_{x_i})(1-x_i)}$$

```
B_x = e^{\sum \ln (\mu^{x_i}_{k_i}) x_i + \sum \ln (1-\mu_{x_i}) (1 - x_i)}
```

This craves to be put into a neat matrix multiplication. Okay, let's look for $p(x)$.

$$p(x) = P_x = \sum_k \pi_k Bernoulli(x|\mu_k) = \sum_k \pi_k B$$

```
p(x) = P_x =\sum_k \pi_k Bernoulli(x|\mu_k) = \sum_k \pi_k B
```

We will use $P(x)$ later to calculate the likelihood L but for now let's calculate Z:

$$P(z_{nk}|x_n, \mu, \pi) = Z = \frac{\pi_k p(x_n|\mu_k)}{\sum_k p(x_n|\mu_k)}$$

```
P(z_{nk}|x_n,\mu,\pi) = Z = \frac{\pi_k p(x_n|\mu_k)}{\sum_k p(x_n|\mu_k)}
```

The likelihood L is given by the following equation which can be found in Pattern Recognition on page 433 equation 9.14.

$$\ln p(X|\pi, \mu, \Sigma) = L = \sum_{n=1}^{N} ln\{\sum_{k=1}^{K} \pi_k N(x_n|\mu_k, \Sigma_k)\}$$

```
\ln p(X|\pi,\mu,\Sigma) = L = \sum_{n=1}^{N} ln\{ \sum_{k=1}^{K}\pi_k N(x_n|\mu_k,\Sigma_k)
\}
```

As we already have the value inside the curly braces ($P(x)$) it's basically just the sum of the logarithms over n.

For calculating π we use the following.

$$\pi_k^{ML} = pi = \frac{\sum_k p(z_{nk|x_n|\mu|\pi})}{N}$$

```
\pi_{k}^{ML} = pi = \frac{\sum_k p(z_{nk|x_n|\mu|\pi})}{N}
```

And finally we use the following for calculating μ. Note that the nominator of π and the denominator of μ are the same.

$$\mu_k^{ML} = mu = \frac{\pi_k p(z_{nk|x_n|\mu|\pi})}{\sum_k \pi_k p(z_{nk|x_n|\mu|\pi})}$$

```
\mu_{k}^{ML} = mu = \frac{\pi_k p(z_{nk|x_n|\mu|\pi})}{\sum_k \pi_k p(z_{nk|x_n|\mu|\pi})}
```

Voila we're done, now the coding is actually just a few lines of code, you'll find it in the appendix.

```
set.seed(1234567890)
max_it = 100 # max number of EM iterations
min_change = 0.1 # min change in log likelihood between two consecutive EM iterations
N=1000 # number of training points
D=10 # number of dimensions
x = matrix(nrow=N, ncol=D) # training data
true_pi = vector(length = 3) # true mixing coefficients
true_mu = matrix(nrow=3, ncol=D) # true conditional distributions

true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)

# Producing the training data
for(n in 1:N) {
  k = sample(1:3,1,prob=true_pi)
```

```
  for(d in 1:D) {
    x[n,d] = rbinom(1,1,true_mu[k,d])
  }
}

plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")

kable(true_pi, caption = "true_pi")
kable(true_mu, caption = "true_mu")

##############################################################################################
# K = 3
##############################################################################################

set.seed(1234567890)
K = 3 # number of guessed components
z = matrix(nrow=N, ncol=K) # fractional component assignments
pi = vector(length = K) # mixing coefficients
mu = matrix(nrow=K, ncol=D) # conditional distributions
llik = vector(length = max_it) # log likelihood of the EM iterations
# Random initialization of the paramters
pi = runif(K,0.49,0.51)
pi = pi / sum(pi)
for(k in 1:K) {
  mu[k,] = runif(D,0.49,0.51)
}

for(it in 1:max_it) {
  # E-step: Computation of the fractional component assignments
  Bx = exp(x %*% log(t(mu)) + (1-x) %*% log(t(1-mu)))
  Px = Bx * rep(pi, nrow(Bx))
  Z = Px / rowSums(Px)
  #Log likelihood computation.
  L = sum(log(rowSums(Px)))
  llik[it] = L

  # Stop if the lok likelihood has not changed significantly
  if (it > 1 && abs(llik[it-1] - llik[it]) < min_change) break

  #M-step: ML parameter estimation from the data and fractional component assignments
  pi = colSums(Z) / N
  mu = (t(Z) %*% x) / colSums(Z)
}

kable(pi, caption = "pi")
kable(mu, caption = "mu")
kable(it, caption = "Number of Iterations")
kable(llik[it], caption = "Ln-Likelihood")
plot(mu[1,], type="o", col="blue", ylim=c(0,1))
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
```

```
plot(llik[1:it], type="o")
```

5.5 Holdout-Principle

5.6 K-Means Algorithm

5.7 Kernel Density Estimation

Epanechnikov kernel Exam 2015 2.3

5.8 Kernel Methods

```
cl = makeCluster(detectCores())

min_distance = function(a, b, ringSize) {

  boundOne = parSapply(cl, b, FUN = function(x) abs(a - x))
  boundTwo = parSapply(cl, b, FUN = function(x) abs(a - ringSize - x))
  boundThree = parSapply(cl, b, FUN = function(x) abs(a + ringSize - x))

  return(pmin(boundOne, boundTwo, boundThree))
}

## Kernels
## The definition for the Guassian Kernel is taken from the slides, page 6.

kernel_gauss_distance = function(pointA, pointB, smoothing) {

  # Use distHaversine() as a help
  m = cbind(pointB$longitude, pointB$latitude)
  u = distHaversine(m, c(pointA$longitude, pointA$latitude))
  u = u / smoothing
  return(exp(-(u^2)))
}

kernel_gauss_day = function(dayA, dayB, smoothing) {

  dayA_in_days = as.numeric(strftime(as.Date(dayA$date), '%j'))
  dayB_in_days = as.numeric(strftime(as.Date(dayB$date), '%j'))

  u = min_distance(dayA_in_days, dayB_in_days, 365)
  u = u / smoothing
  return(exp(-(u^2)))
}

kernel_gauss_hour = function (hourA, hourB, smoothing) {

  hourA_in_h = parSapply(cl, hourA$time, FUN = function(x)
    as.numeric(difftime(strptime(x, format = "%H:%M:%S"),
                        strptime("00:00:00", format = "%H:%M:%S"))))
```

```
  hourB_in_h = parSapply(cl, hourB$time, FUN = function(x)
    as.numeric(difftime(strptime(x, format = "%H:%M:%S"),
                        strptime("00:00:00", format = "%H:%M:%S"))))

  u = min_distance(hourA_in_h, hourB_in_h, 24)
  u = u / smoothing
  return(exp(-(u^2)))
}

# Students' code here
pred_temp_sum = predict_weather(latitude, longitude, date)

plot_data_frame =
  data.frame(as.POSIXct(paste(date ,pred_temp_sum$times),
                        format = "%Y-%m-%d %H:%M"), pred_temp_sum[2:3])
colnames(plot_data_frame) = c("hour", "predWithSum", "predWithProd")

ggplot(plot_data_frame) +
  geom_line(aes(x = hour, y = predWithSum, group = 1,
                colour = "Kernel Regression (Sum)")) +
  geom_point(aes(x = hour, y = predWithSum), colour = "orange") +
  geom_line(aes(x = hour, y = predWithProd, group = 1,
                colour = "Kernel Regression (Prod)")) +
  geom_point(aes(x = hour, y = predWithProd), colour = "blue") +
  labs(title = "Temperature Prediction with Kernel Regression (Sum and Prod)",
       y = "Temperature", x = "Hour of the Day", color = "Legend") +
  scale_color_manual(values = c("blue", "orange"))
```

5.8.1 Histogram Classification

5.8.2 Moving Windows Classification

5.9 Loss-Matrix

```
L = matrix(c(0, 10, 1, 0), nrow = 2)
colnames(L) = c("Predicted", "Predicted")
rownames(L) = c("good", "bad")
kable(L)

# Prediction
prediction_bayes_train_raw =
  predict(naiveBayesModel, newdata = train, type = "raw")
prediction_bayes_test_raw =
  predict(naiveBayesModel, newdata = test, type = "raw")

confusion_matrix_bayes_train =
  table(prediction_bayes_train_raw[,2]/
          prediction_bayes_train_raw[,1] > 10, train$good_bad)
confusion_matrix_bayes_test =
  table(prediction_bayes_test_raw[,2]/
          prediction_bayes_test_raw[,1] > 10, test$good_bad)
```

```
error_bayes_train_raw = 1 - sum(diag(confusion_matrix_bayes_train)/
                                sum(confusion_matrix_bayes_train))
error_bayes_test_raw =  1 - sum(diag(confusion_matrix_bayes_test)/
                                sum(confusion_matrix_bayes_test))
```

5.10 Probability Model and Log-Likelihood

From the engineers

5.11 ROC-Curve

5.11.1 Calculate ROC

```
# prediction optimal tree
prediction_optimalTree_test_p =
  predict(optimalTree, newdata = test, type = "vector")
# prediction naive bayes
prediction_bayes_test_p =
  predict(naiveBayesModel, newdata = test, type = "raw")

pi = seq(from = 0.00, to = 1.0, by = 0.05)
fprs_tree = c()
tprs_tree = c()
fprs_bayes = c()
tprs_bayes = c()

for (i in pi) {
  current_tree_pi_confusion =
    table(test$good_bad, factor(prediction_optimalTree_test_p[,2] > i,
                                lev=c(TRUE, FALSE)))
  current_bayes_pi_confusion =
    table(test$good_bad, factor(prediction_bayes_test_p[,2] > i,
                                lev=c(TRUE, FALSE)))

  # FPR = FP / N-
  # TPR = TP / N+
  fprs_tree = c(fprs_tree, current_tree_pi_confusion[1,1]/
                  sum(current_tree_pi_confusion[1,]))
  tprs_tree = c(tprs_tree, current_tree_pi_confusion[2,1]/
                  sum(current_tree_pi_confusion[2,]))

  fprs_bayes = c(fprs_bayes, current_bayes_pi_confusion[1,1]/
                   sum(current_bayes_pi_confusion[1,]))
  tprs_bayes = c(tprs_bayes, current_bayes_pi_confusion[2,1]/
                   sum(current_bayes_pi_confusion[2,]))
}

roc_values = data.frame(fprs_tree, tprs_tree, fprs_bayes, tprs_bayes)
```

5.11.2 Print ROC

```
ggplot(roc_values) +
  geom_line(aes(x = fprs_tree, y = tprs_tree,
                colour = "ROC Optimized Tree")) +
  geom_point(aes(x = fprs_tree, y = tprs_tree), colour = "orange") +

  geom_line(aes(x = fprs_bayes, y = tprs_bayes,
                colour = "ROC Naive Bayes")) +
  geom_point(aes(x = fprs_bayes, y = tprs_bayes), colour = "blue") +

  geom_abline(slope=1, intercept=0, linetype="dotted") +

  labs(title = "ROC for Optimized Tree and Naive Bayes", y = "TPR",
       x = "FPR", color = "Legend") +
  scale_color_manual(values = c("blue", "orange"))
```

5.12 Splines

Some sample code using GAM.

```
gam_model_additive = gam(formula =
                         Mortality ~ s(Year, k=length(unique(influanza$Year))) +
                         s(Week, k=length(unique(influanza$Week))) +
                         s(Influenza, k=length(unique(influanza$Influenza))),
                         family = gaussian(), data = influanza, method="GCV.Cp")

summary(gam_model_additive)
plot(gam_model_additive)

time = influanza$Time
observed = influanza$Mortality
predicted = fitted(gam_model_additive)

mortality_values = data.frame(time, observed, predicted)

ggplot(mortality_values) +
  geom_line(aes(x = time, y = observed,
                colour = "Observed Mortality")) +
  geom_line(aes(x = time, y = predicted,
                colour = "Predicted Mortality")) +
  labs(title = "Observed vs Predicted Mortality", y = "Mortality",
       x = "Time", color = "Legend") +
  scale_color_manual(values = c("blue", "orange"))
```