Data Science Assignment 2

Sebastian Pusch (S5488079), Ivan Hegeman (s4789725)

```r
dat = read.table('./data/decision.dat', header = T)
subj8 = dat[dat$subjNo == 8,]

# prepare training data--make sure there are no duplicate columns , and also no columns that are consta
training_data = data.frame(x = dat[,c(1,3:5,7:8)], class = as.factor(dat$ER))
training_data = data.frame(x = subj8[,c(3:5,7:8)], class = as.factor(subj8$ER))
# set apart testing data
samp <- sample(nrow(training_data),round(0.2*nrow(training_data)))
testing_data <- training_data[samp,]
training_data2 <- training_data[-samp,]
subj8training <- subj8[-samp,]
# for the training data select an equal number of each class (we need the original subj8 to find the ER
errorInd <- which(subj8training$ER==1)
Nerrors <- length(errorInd)
correctInd <- which(subj8training$ER==0)
sampleCorr <- sample(correctInd,Nerrors)
equal_training_data <- training_data2[c(errorInd,sampleCorr),]
# Now run the SVM
svm.linear = svm(class~.,
data = equal_training_data,
kernel = "linear",
cost = 10,
scale = TRUE)
summary(svm.linear)
```

```
##
## Call:
## svm(formula = class ~ ., data = equal_training_data, kernel = "linear",
##     cost = 10, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  248
##
##  ( 122 126 )
##
##
## Number of Classes:  2
##
## Levels:
```

```
##  0 1
```

```
# do the stats on the training data
train.pred <- predict(svm.linear,equal_training_data)
# compute confusion matrix
err.tr1 <- table(equal_training_data$class,train.pred)
err.tr1
```

```
##    train.pred
##       0   1
##   0 126  88
##   1  21 193
```

```
# compute error rate
tr.err1 <- (1 - (sum(diag(err.tr1))/sum(err.tr1)))
cat("Training error rate:", tr.err1, "\n")
```

```
## Training error rate: 0.2546729
```

```
# now look at how the model does on the testing data
test.pred <- predict(svm.linear,testing_data)
err.te1 <- table(testing_data$class,test.pred)
err.te1
```

```
##    test.pred
##       0   1
##   0 268 193
##   1   5  44
```

```
te.err1 <- (1- (sum(diag(err.te1))/sum(err.te1)))
cat("Testing error rate:", te.err1, "\n")
```

```
## Testing error rate: 0.3882353
```

```
# for a linear SVM the feature importance can be directly extracted from the coefficients
coefficients <- t(svm.linear$coefs) %*% svm.linear$SV
importance <- abs(coefficients)
importance <- importance / max(importance)
cat("Importance:\n")
```

```
## Importance:
```

```
importance
```

```
##               x.RT     x.isLeft     x.cohVec   x.blocknum x.isDots
## [1,] 2.681114e-06 3.104217e-05 5.747787e-05 1.107897e-05        1
```

First, let's examine how the model performs at predicting the training set. As we can see from the confusion matrix, the model correctly predicted 131 negative class cases (i.e. predicted correctly that the participant did not make an error) and 182 positive class cases (i.e. predicted correctly that the participant made an error). Unfortunately, the model incorrectly predicted 21 negative class cases (i.e. incorrectly predicted that the participant did not make an error), and 72 positive class cases (i.e. incorrectly predicted that the participant made an error).

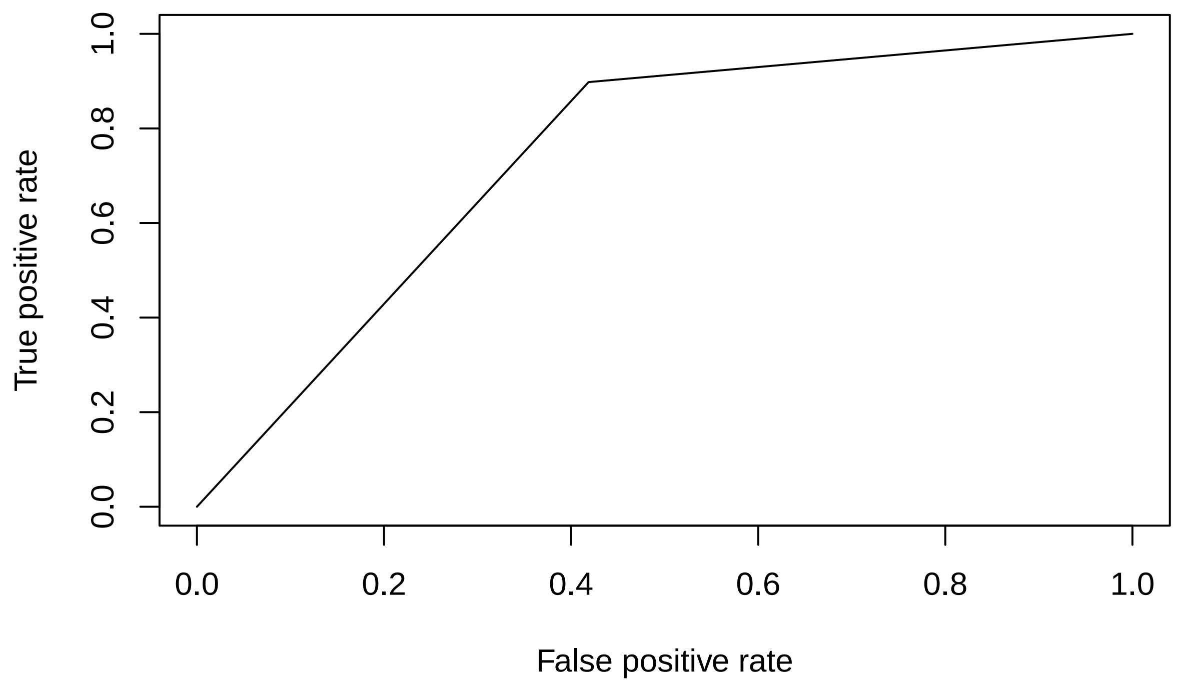
Now let's examine the test data predictions. As we can see from the confusion matrix, the model correctly predicted 247 negative class cases (i.e., correctly predicted that the participant did not make an error) and 55 positive class cases (i.e., correctly predicted that the participant made an error). However, it also incorrectly predicted 5 negative class cases (i.e., incorrectly predicted that the participant did not make an error), and 203 positive class cases (i.e., incorrectly predicted that the participant made an error).

From those confusion matrices, we can see that the model is relatively accurate when it comes to predicting incorrect answers, however the model really struggles with properly predicting correct answers.

For both the training data and the test data, the error rates are high (23% and 41% respectively). Our model is probably underfitting, which can be due to many things (such as the model being too simple, or the features selected being non-relevant). In our case, I would argue that the feature selection is suboptimal, as we can see from the importance table that the isDots variable's importance is extremely high (removing the control condition data would make sense here, which would allow us to remove this feature and only focus on the important ones). Also, the cohVec variable seems to be less important to the prediction of the model than I would have expected, however this can be explained by the fact that the coherence was adjusted for the participants to have approximately 70% and 90% correct (which means that some participants could have 70% correct for 10% coherence and other for 2% coherence), i.e. the coherence does not mean much wrt the correctness.

ROC analysis

```r
library(ROCR)
p <- predict(svm.linear,testing_data)
pr <- prediction(as.numeric(p),as.numeric(testing_data$class))
prf <- performance(pr, measure = "tpr", x.measure = "fpr")
plot(prf)
```



```
auc <- performance(pr, measure = "auc")
auc <- auc@y.values[[1]]
```

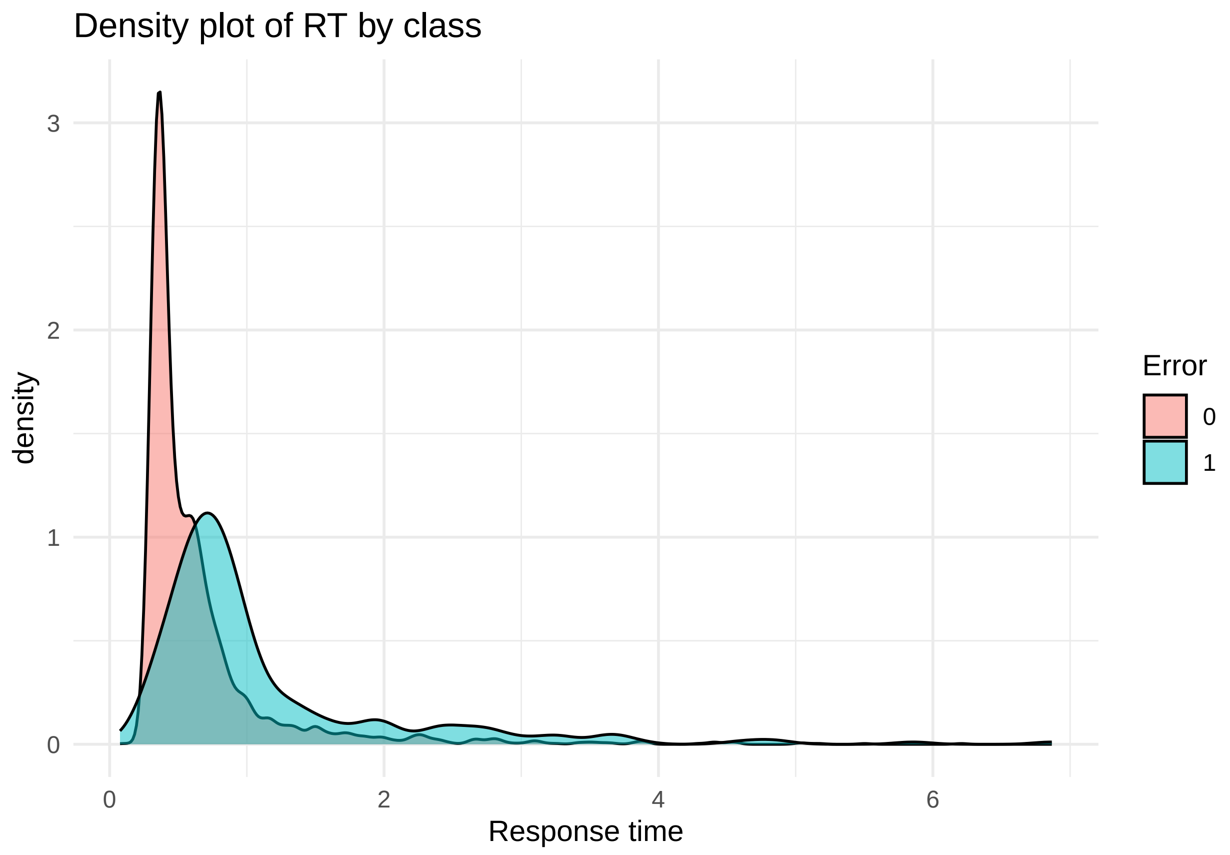
(a) ROC (Receiver Operator Characteristic) is a plot that display the performance of a model across different classification threshold. The y-axis represents the True Positive Rate, meaning the number of true positive over the number of actual positives. The x-axis represents the False Positive Rate, which is the number of false positives over the total number of actual negatives. Higher thresholds result in fewer false positives at the cost of missing more true positives, while lower threshold lead to capturing more true positive but with more false positives. The plot captures this trend (assuming a non-random classifier) and helps choosing an appropriate threshold.

(b) The ROC curve shows that the model performs better than random guessing, as it achieves a True Positive Rate of about 0.65 with a False Positive Rate below 0.2. This indicates the model is able to correctly identify a portion of the actual positives while keeping false positives relatively low at that threshold. While better than random guessing, a TPR 0.65 (the model identifies only 65% of the positive cases) is not particularly strong.

Data problems

```
# outliers
library(ggplot2)
ggplot(training_data, aes(x = x.RT, fill = class)) +
  geom_density(alpha = 0.5) +
```

```r
  labs(title = "Density plot of RT by class", x = "Response time", fill = "Error") +
  theme_minimal()
```



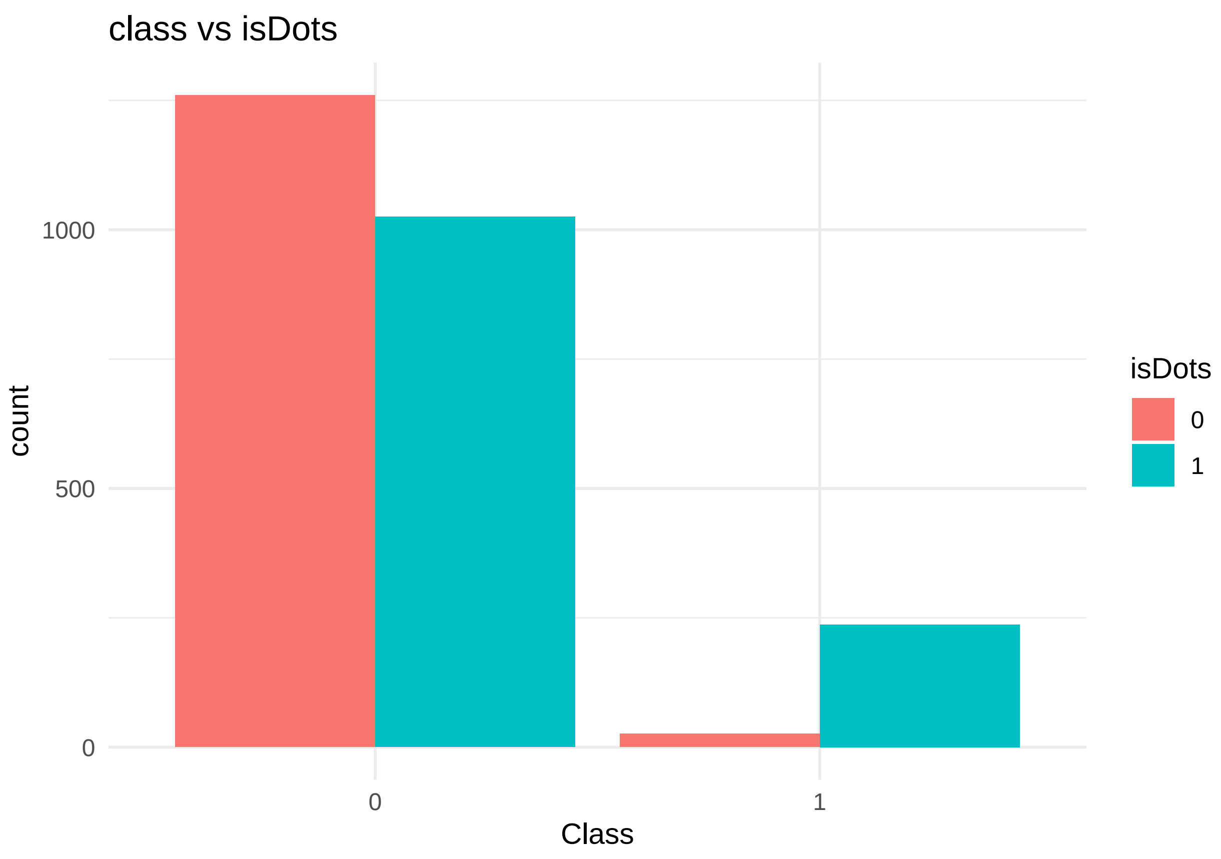
The density plot shows that the response time variable has a significant number of outliers, with a clear right tail for both class 0 (correct responses) and class 1 (errors). The right-skewed distribution is present in both classes, and indicates that there are instances with much longer response times. These longer response times could suggest issues like distractions or difficulty with the task.

```r
# correlation
cor(training_data[, c("x.RT", "x.cohVec", "x.blocknum")])
```

```
##                  x.RT    x.cohVec x.blocknum
## x.RT        1.00000000 -0.18445276 0.06883059
## x.cohVec   -0.18445276  1.00000000 0.01946211
## x.blocknum  0.06883059  0.01946211 1.00000000
```

As we can see from the correlation matrix, there is a slight negative correlation between response time and coherence, suggesting that as task difficulty decreases, response time tends to decrease slightly. However, this relationship is weak. The correlation between response time and block number is very weak, indicating little to no connection between these variables. Similarly, the correlation between coherence factor and block number is extremely weak, showing that these two variables are almost uncorrelated. Overall, the low correlation values suggest that these variables are not strongly related to one another.

```r
# useless variable
ggplot(training_data, aes(x = factor(class), fill = factor(x.isDots))) +
  geom_bar(position = "dodge") +
  labs(title = "class vs isDots", x = "Class", fill = "isDots") +
  theme_minimal()
```



From this graph, we can observe that in the control condition, participants almost always respond correctly, while in the actual test condition, there is a way higher likelihood of errors. This obviously makes sense. However, we do not see why this variable was included in training the SVM, as the control condition almost always results in correct responses, making it less informative for predicting errors. Additionally, given the noticeable difference between the control and actual test conditions, this variable is likely to have a significant impact on the SVM's performance, potentially skewing the results or dominating the model's predictions (which we can also see from the "importance" tests from part 1 of this assignment).

Model optimization

```
# find best model parameter
tune_out = tune(svm,
                class~.,
                data = equal_training_data,
                kernel = "linear",
                ranges = list(cost = c(0.001, 0.01, 0.1, 1,5,10,100)))
summary(tune_out)
```

```
##
```

```
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.2544297
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.4898671 0.11872025
## 2 1e-02 0.2544297 0.06347449
## 3 1e-01 0.2544297 0.06347449
## 4 1e+00 0.2544297 0.06347449
## 5 5e+00 0.2544297 0.06347449
## 6 1e+01 0.2544297 0.06347449
## 7 1e+02 0.2544297 0.06347449
```

```r
bestmod = tune_out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = class ~ ., data = equal_training_data,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  316
##
##  ( 158 158 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

After tuning the linear SVM using a range of cost values, we found that the best model achieved a cross-validation error rate of ~26.3% at a cost value of 1. This performance represents a noticeable improvement over models trained with either very low or very high cost values, which resulted in significantly higher error rates. The tuning process helped identify a better trade-off between underfitting and overfitting, allowing the model to generalize more effectively to unseen data.

Model comparison

```
# find best model parameter
tune_out = tune(svm,
                class~.,
                data = equal_training_data,
                kernel = "polynomial",
                ranges = list(cost = c(0.001, 0.01, 0.1, 1,5,10,100)))
summary(tune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.2267996
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.5514396 0.04696241
## 2 1e-02 0.5327243 0.04868342
## 3 1e-01 0.2571982 0.06844199
## 4 1e+00 0.2291805 0.06467456
## 5 5e+00 0.2291805 0.06467456
## 6 1e+01 0.2267996 0.06653495
## 7 1e+02 0.2407530 0.08168625
```

```
bestmod = tune_out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = class ~ ., data = equal_training_data,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  236
##
##  ( 131 105 )
##
##
```

```
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
# find best model parameter
tune_out = tune(svm,
                class~.,
                data = equal_training_data,
                kernel = "radial",
                ranges = list(cost = c(0.001, 0.01, 0.1, 1,5,10,100)))
summary(tune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.2334441
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.5560354 0.03503311
## 2 1e-02 0.5560354 0.03503311
## 3 1e-01 0.2542082 0.07198817
## 4 1e+00 0.2334441 0.07540951
## 5 5e+00 0.2473976 0.07295619
## 6 1e+01 0.2450166 0.08079327
## 7 1e+02 0.2543743 0.07784617
```

```
bestmod = tune_out$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = class ~ ., data = equal_training_data,
##     ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  240
##
##  ( 125 115 )
##
##
```

```
## Number of Classes:  2
##
## Levels:
##  0 1
```

We repeated the analysis using SVMs with non-linear kernels, specifically the `polynomial` and `radial` kernels. Both models outperformed the linear SVM in terms of prediction accuracy. The `polynomial` kernel achieved the lowest cross-validation error rate of approximately ~24.1%, with performance stabilizing at cost values above 5. The radial kernel also performed well, reaching an error rate of ~24.4% at a cost value of 100. These improvements indicate that the data likely has non-linear relationships that the linear kernel fails to capture. Overall, the results suggest that both radial and polynomial kernels provide a better fit for the data and are more effective at making accurate predictions.

Extending this to algorithms more generally

(a) You cannot use AIC on a classifier that detects humans in satellite images because such a model would not satisfy the assumptions of this method. One of the assumptions of AIC is that the model is a fixed parametric model estimated via maximum likelihood and intended to approximate the true data-generating process. However, deep learning classifiers such as CNNs, that are effective for complex image classification, are highly flexible, often overparameterized, and typically do not provide a well-defined likelihood function or interpretable parametric form, so AIC likely inapplicable. For non-parametric models like CNNs, cross-validation is a more appropriate approach for model selection and hyperparameter tuning, while a separate hold-out test set should be used to evaluate final model performance, as these methods do not rely on the strong parametric assumptions required by AIC.

(b) Overfitting can still occur even when cross-validation is used, due to data leakage. This happens when information from the test set influences the training process, for example through improper feature engineering or if the hyperparameters are tuned based on the noise rather than the actual useful features.

(c) Instead of using the validation set to determine the performance of the model, the lockbox method requires us to put aside part of the data that can only be accessed until the analysis protocol is clearly defined. This data is then only used to test the final performance of the model, once everything else has been decided, which allows us to see how well the model generalizes. This is also done in machine learning competitions, where the ultimate performance of the algorithm is evaluated on a separate set of data that is reserved for this purpose. This is very useful to prevent overfitting, as it ensures that the model has not been influenced by the test data during the training or model selection process.

(d) To prevent overfitting, there are many other techniques that can be used. First of all, training with more data will always result in the model having less chance to overfit, as it makes it less likely for the model to memorize specific details and more likely to generalize. Regularization, such as L1 and L2, adds a penalty to the loss function based on the number of the model's parameters, which prevents the model from fitting overly complex patterns. Early stopping is another useful strategy: we can stop training when the model starts to overfit, ensuring it doesn't learn noise or irrelevant details. Alternatively, reducing training time can help by preventing the model from completely memorizing the training data. Lastly, choosing the right model also plays a crucial role, as depending on the task a simpler or more complex model might be required. If the data consists of linear relationships for example, using a complex non-linear model would probably result in overfitting.