CYO Project: Codon Usage Analysis

HarvardX: PH125.9x Data Science: Capstone

Mario Träger

September 09, 2021

Contents

1 Overview

1.1 Introduction

The genetic code is a universal code used by all living systems to transfer biological information. In a first step, the genetic information stored in the DNA is transcribed into a messenger RNA (mRNA). In a second step, the information is translated into the amino acid sequence of proteins. The genetic information is coded by four "letters", the bases adenine (A), cytosine (C), guanine (G) and thymine (T). Three consecutive bases form a so called codon, which may code for an amino acid. This results in $4^3 = 64$ possible codon permutations. During the transcription from DNA to mRNA the base thymine is substituted by the base uracil (U). That is why the genetic code is expressed in the ACGU nomenclature. The following table shows the genetic code as expressed by base triplets of mRNA and their corresponding amino acids:

Codon1	Aminoacid1	Codon2	Aminoacid2	Codon3	Aminoacid3	Codon4	Aminoacid4
UUU	Phenylalanine	UCU	Serine	UAU	Tyrosine	UGU	Cysteine
UUC	Phenylalanine	UCC	Serine	UAC	Tyrosine	UGA	Cysteine
UUA	Leucine	UCA	Serine	UAA	stop codon	UGA	stop codon
UUG	Leucine	UCG	Serine	UAG	stop codon	UGG	Tryptophan
CUU	Leucine	CCU	Proline	CAU	Histidine	CGU	Arginine
CUC	Leucine	CCC	Proline	CAC	Histidine	CGC	Arginine
CUA	Leucine	CCA	Proline	CAA	Glutamine	CGA	Arginine
CUG	Leucine	CCG	Proline	CAG	Glutamine	CGG	Arginine
AUU	Isoleucine	ACU	Threonine	AAU	Asparagine	AGU	Serine
AUC	Isoleucine	ACC	Threonine	AAC	Asparagine	AGC	Serine
AUA	Isoleucine	ACA	Threonine	AAA	Lysine	AGA	Arginine
AUG	Methionine	ACG	Threonine	AAG	Lysine	AGG	Arginine
GUU	Valine	GCU	Alanine	GAU	Aspartate	GGU	Glycine
GUC	Valine	GCC	Alanine	GAC	Aspartate	GGC	Glycine
GUA	Valine	GCA	Alanine	GAA	Glutamate	GGA	Glycine
GUG	Valine	GCG	Alanine	GAG	Glutamate	GGG	Glycine

Since there are only 20 proteinogenic amino acids, most of the amino acids are coded by more then one codon. This fact is known as the degeneracy of the genetic code. Additionally, there are three codons (UAA, UAG, UGA) that do not code for amino acids. Instead, they are a signal to stop the translation. The codon AUG codes for the amino acid methionine and is also the start signal for the translation process.

The degeneracy of the genetic code leads to the assumption that different organisms have different codon preferences. Organisms that are closely related to each other are supposed to have a similar codon usage and organisms that are distantly related are supposed to have a clearly different codon usage. Closely related organisms share similar characteristics and are scientifically classified into the same group. These hierarchically organized groups are called taxa (singular: taxon).

1.2 Goal of the Project

The goal of this project was the prediction of taxa from the codon usage frequencies of different organism by the application of machine learning techniques. For this purpose, a data set of DNA codon usage frequencies from protein-coding genes of a large sample of diverse biological organisms from different taxa was used. Several algorithms should be applied to the data set to obtain an algorithm that predicts the taxa with high accuracy. A second goal was the comprehensive analysis of the data set and to gain further insights after application of the different techniques.

1.3 Description of the Data Set

The codon usage data set was provided by the UCI Machine Learning Repository[1]. The data originate from the CUTG (Codon Usage Tabulated from GeneBank) database[2] and were further processed[3].

Initially, the required libraries were loaded and the data set was downloaded, unzipped and read:

```
library(DiagrammeR)
library(tidyverse)
library(caret)
library(knitr)
library(gplots)
library(RColorBrewer)

url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/00577/codon_usage.csv.zip"
dl <- tempfile()
download.file(url, dl)
unzip(dl, "codon_usage.csv")
dat <- read.csv("codon_usage.csv")
```

The data set consists of 13028 samples and 69 attributes. The first five attributes are: Kingdom, DNAtype, SpeciesID, Ncodons, SpeciesName. The attributes 6 to 69 are the 64 codons. The columns are formatted differently:

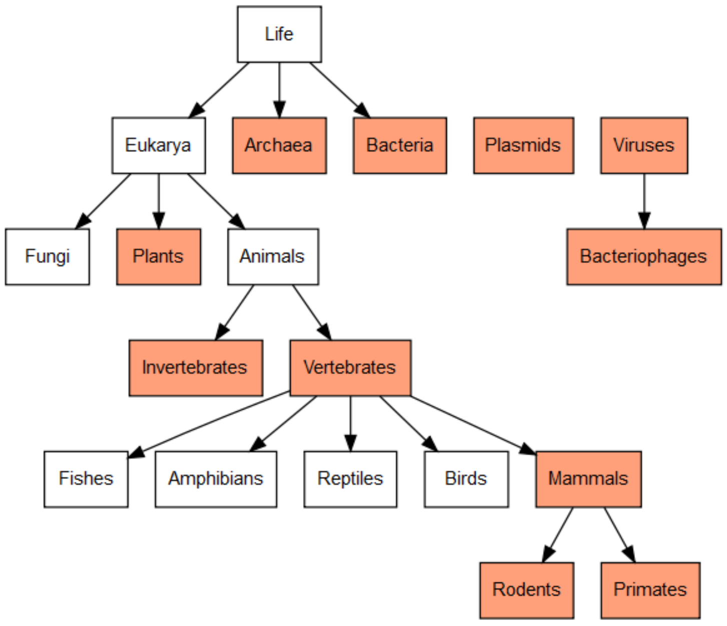
```
##     Kingdom     DNAtype   SpeciesID     Ncodons SpeciesName         UUU
## "character"   "integer"   "integer"   "integer" "character" "character"
##         UUC         UUA         UUG         CUU         CUC         CUA
## "character"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##         CUG         AUU         AUC         AUA         AUG         GUU
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##         GUC         GUA         GUG         GCU         GCC         GCA
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##         GCG         CCU         CCC         CCA         CCG         UGG
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##         GGU         GGC         GGA         GGG         UCU         UCC
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##         UCA         UCG         AGU         AGC         ACU         ACC
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##         ACA         ACG         UAU         UAC         CAA         CAG
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##         AAU         AAC         UGU         UGC         CAU         CAC
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##         AAA         AAG         CGU         CGC         CGA         CGG
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##         AGA         AGG         GAU         GAC         GAA         GAG
##   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##         UAA         UAG         UGA
##   "numeric"   "numeric"   "numeric"
```

The columns `Kingdom`, `SpeciesName`, `UUU` and `UUC` are formatted as `character`, the columns `DNAtype` and `SpeciesID` as `integer`. All other columns are `numeric`. The three stop codons are in the last three columns.

`Kingdom` is the column for the different taxa. There are 11 unique taxa: vrl, arc, bct, phg, plm, pln, inv, vrt, mam, rod, pri. These abbreviations stand for virus, archaea, bacteria, bacteriophage, plasmid, plant,

[1] https://archive.ics.uci.edu/ml/datasets/Codon+usage
[2] https://www.kazusa.or.jp/codon/
[3] Khomtchouk BB: 'Codon usage bias levels predict taxonomic identity and genetic composition'. bioRxiv, 2020.

invertebrate, vertebrate, mammal, rodent and primate, respectively. The figure below shows a simplified overview about the taxonomic relation between these taxa and how they are incorporated into the tree of life (taxa present in the data set in *red*). Bacteria, archaea and eukarya (organisms whose cells have a nucleus) are the three major domains of living organisms. Invertebrates comprise about 97 percent of animal species. Vertebrates contain mammals, and mammals contain rodents and primates, which all are categorized separately in the data set. Viruses (including bacteriophages, viruses that infect bacteria and archaea) are not considered to be organisms, because they lack an independent metabolism. Plasmids are small, extrachromosomal DNA molecules within (mostly bacteria) cells, that can replicate independently and are also not considered as organisms.



`DNAtype` specifies the DNA type of the entry: 0-genomic, 1-mitochondrial, 2-chloroplast, 3-cyanelle, 4-plastid, 5-nucleomorph, 6-secondary_endosymbiont, 7-chromoplast, 8-leucoplast, 9-NA, 10-proplastid, 11-apicoplast, and 12-kinetoplast.

`SpeciesID` is a unique ID for every organism in the data set, named in the `SpeciesName` column.

`Ncodons` (number of codons) is the algebraic sum of the numbers listed for the different codons in an entry of CUTG. Codon frequencies are normalized to the total codon count, hence the number of occurrences divided by `Ncodons` is the codon frequencies listed in the data file.

The codon columns contain the frequencies for each codon.

1.4 Key Steps

- Separation of a final hold-out test set
- Exploratory data analysis of the data set
- Training and tuning of different machine learning algorithms by cross-validation
- Performance comparison of the different models and selection of the best performing model
- Analysis of the variable importance
- Application of the best performing model on the hold-out test set
- Performance analysis of the final model

2 Methods and Analysis

2.1 Data Cleaning

First, the `Kingdom` column was transformed into a `factor` format, since the prediction of the taxa is a classification problem. Second, the first two codon columns, `UUU` and `UUC`, which were wrongly formatted as `character` due to text entries, were coerced to a `numeric` format, resulting in NA values (three in total) at the positions of the text entries. The NAs were replaced by 0s:

```r
dat <- dat %>%
  mutate(Kingdom = as.factor(Kingdom), UUU = as.numeric(UUU), UUC = as.numeric(UUC)) %>%
  replace_na(list(UUU = 0, UUC = 0))
```

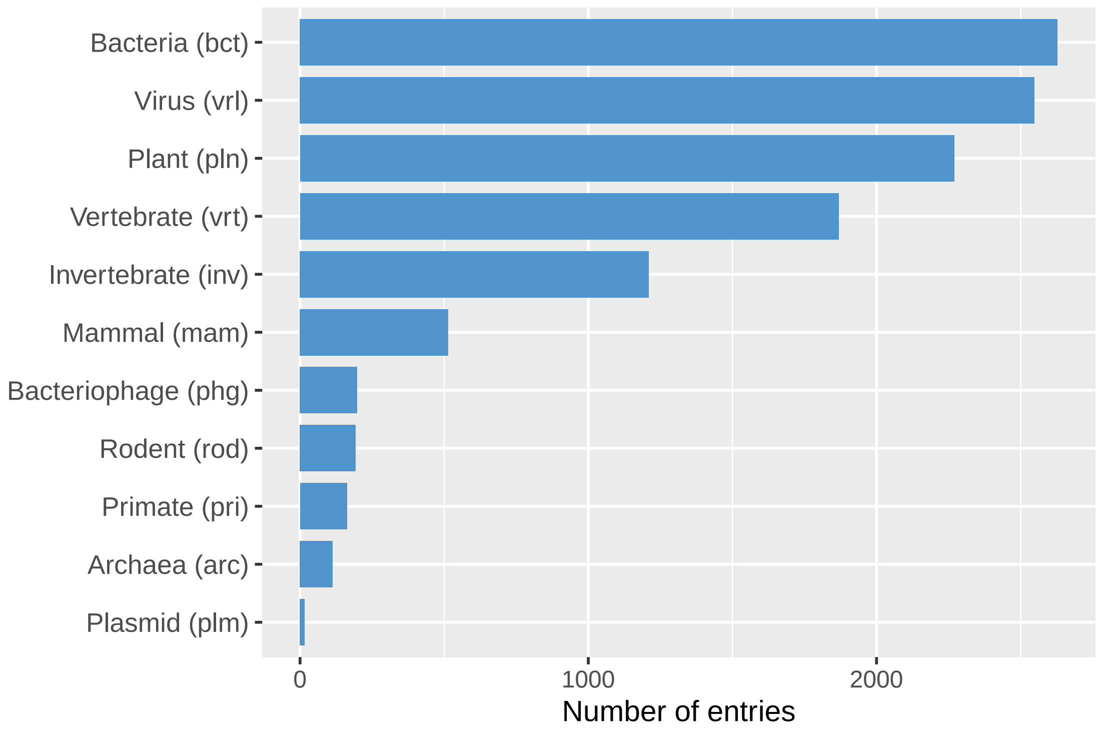
2.2 Separation of a Validation Set as Final Hold-out Test Set

In order to evaluate the performance of a final best performing model, a validation set was separated before the start of the analysis and considered as unknown and independent data. The remaining train set was used for analysis, training, tuning and model selection. In order to have as much data as possible to train, ninety percent of the data went to the train set and ten percent to the validation set:

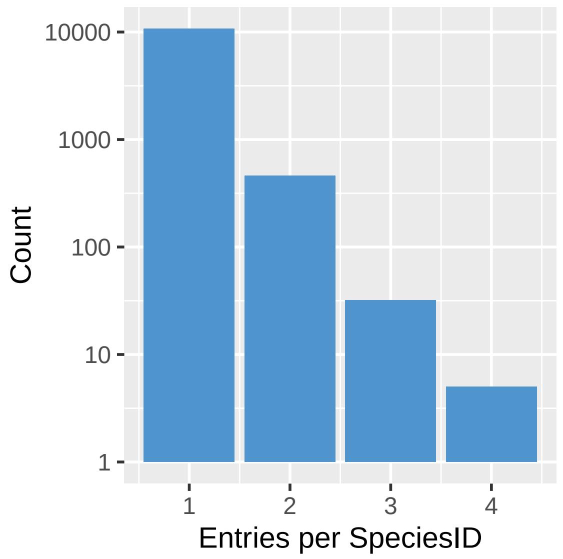
```r
set.seed(2005)
test_index <- createDataPartition(y = dat$Kingdom, times = 1, p = 0.1, list = FALSE)
train <- dat[-test_index,]
validation <- dat[test_index,]
```

2.3 Exploratory Data Analysis

The `Kingdom` column is the outcome column for the machine learning procedures. The bar plot below shows the number of entries for each taxon in the data set. Most of the entries belong to the bacteria, virus, plant and vertebrate taxa, respectively. Invertebrates have an average number of entries. Entries for plasmid, archaea, primate, rodent, bacteriophage and mammal are considerably less frequent in the data set.



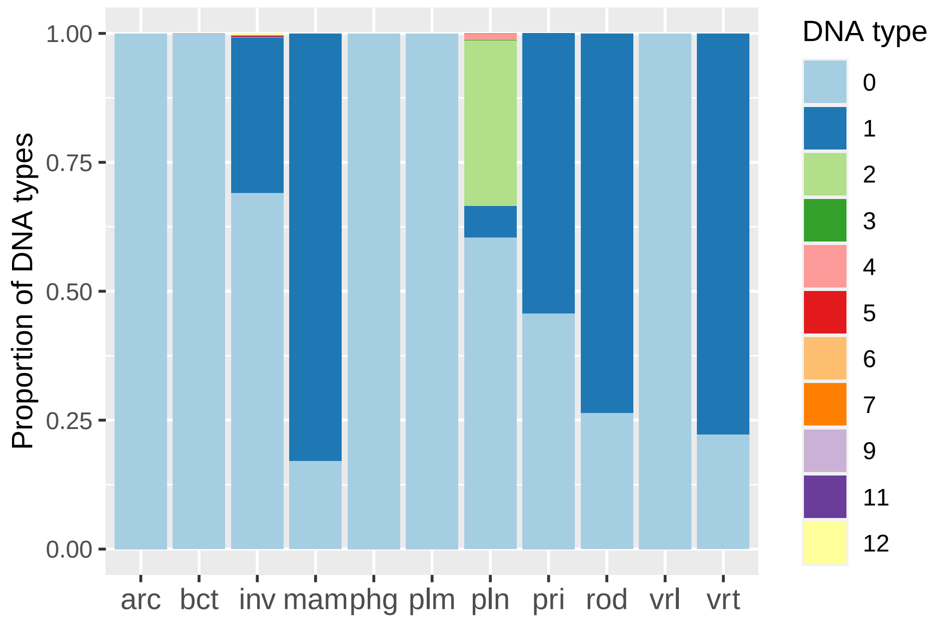
Looking at the number of entries for each `SpeciesID` shows for the vast majority of the species only one entry. However, there are also species with up to four entries in the data set.



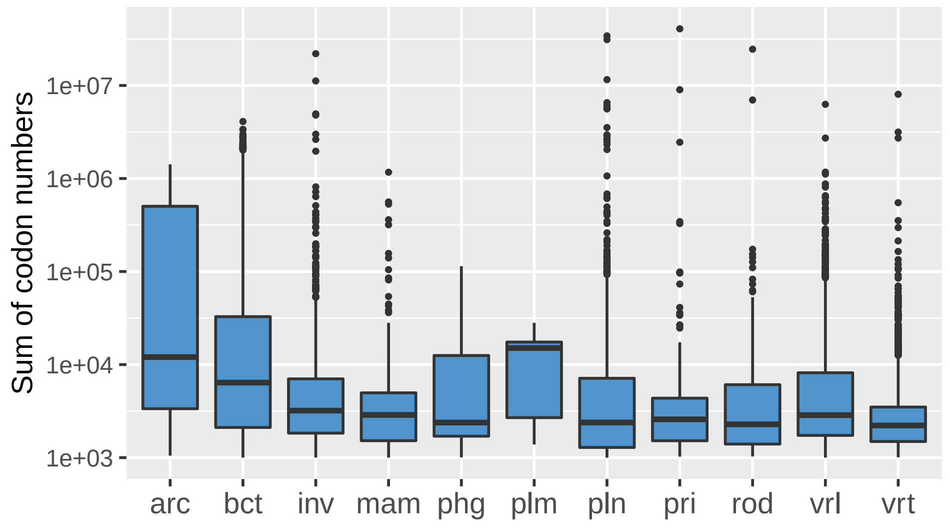
The explanation for this observation is that some species have multiple entries for different DNA types listed in the data set. The table below shows the number of entries for each DNA type. About 71 percent of the entries are for genomic DNA, followed by 22 percent for mitochondrial and 6 percent for chloroplast DNA. For all other DNA types there are only a few entries. DNA types 8 and 10 are not present in the data set and two entries are not specified (DNA type 9).

Name	DNAtype	Count	Share
genomic	0	8337	71.13
mitochondrial	1	2613	22.29
chloroplast	2	729	6.22
cyanelle	3	2	0.02
plastid	4	28	0.24
nucleomorph	5	2	0.02
secondary_endosymbiont	6	1	0.01
chromoplast	7	1	0.01
NA	9	1	0.01
apicoplast	11	2	0.02
kinetoplast	12	5	0.04

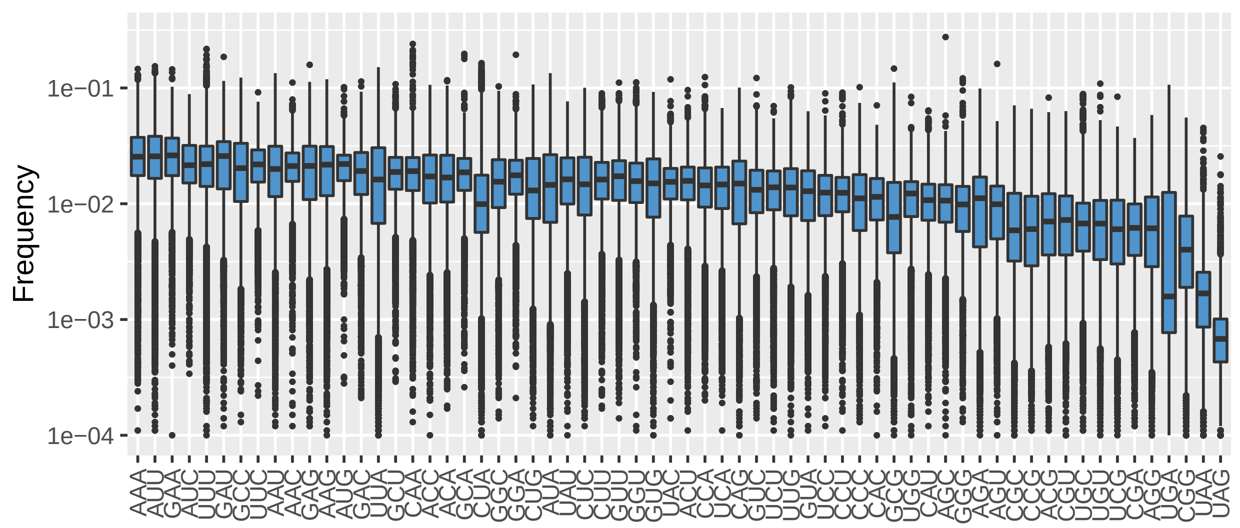
Stratification of the DNA type composition by each taxon in the bar plot below shows that for archaea, bacteria, bacteriophage, plasmid and virus all DNA data is genomic. That is not surprising because these taxa do not contain any additional organelles. For all vertebrate taxa the majority of the entries are from mitochondrial DNA. Chloroplast DNA is mainly present in plant entries, with a substantial share. The other DNA types from special organelles are only present in plants and invertebrates.



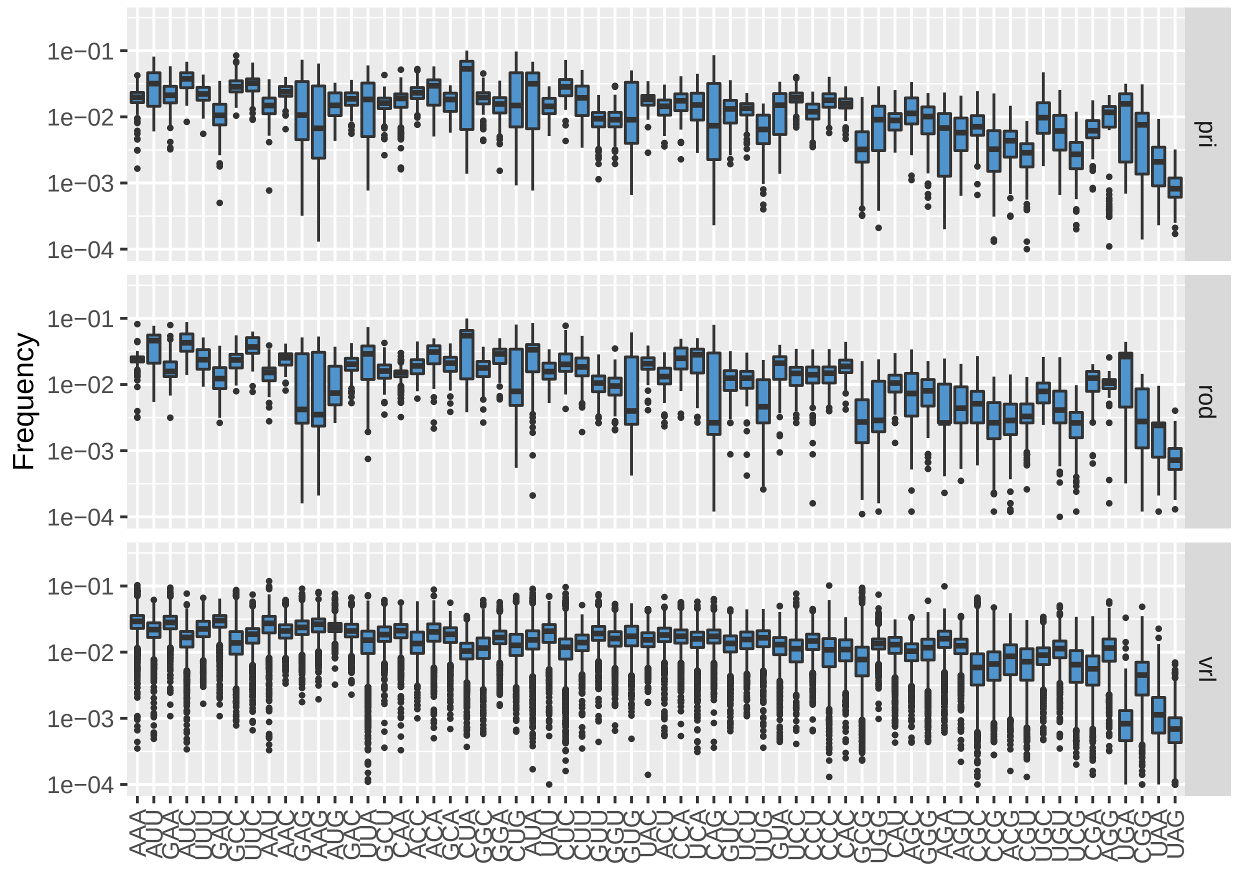
`Ncodons` is the total number of codons used to calculate the frequencies listed. The higher this number the higher is the reliability of the codon frequencies. The box plot of `Ncodons` stratified by `Kingdom` below shows similar distributions for most of the taxa with exception of archaea, bacteria and plasmids, which show considerably higher codon numbers.



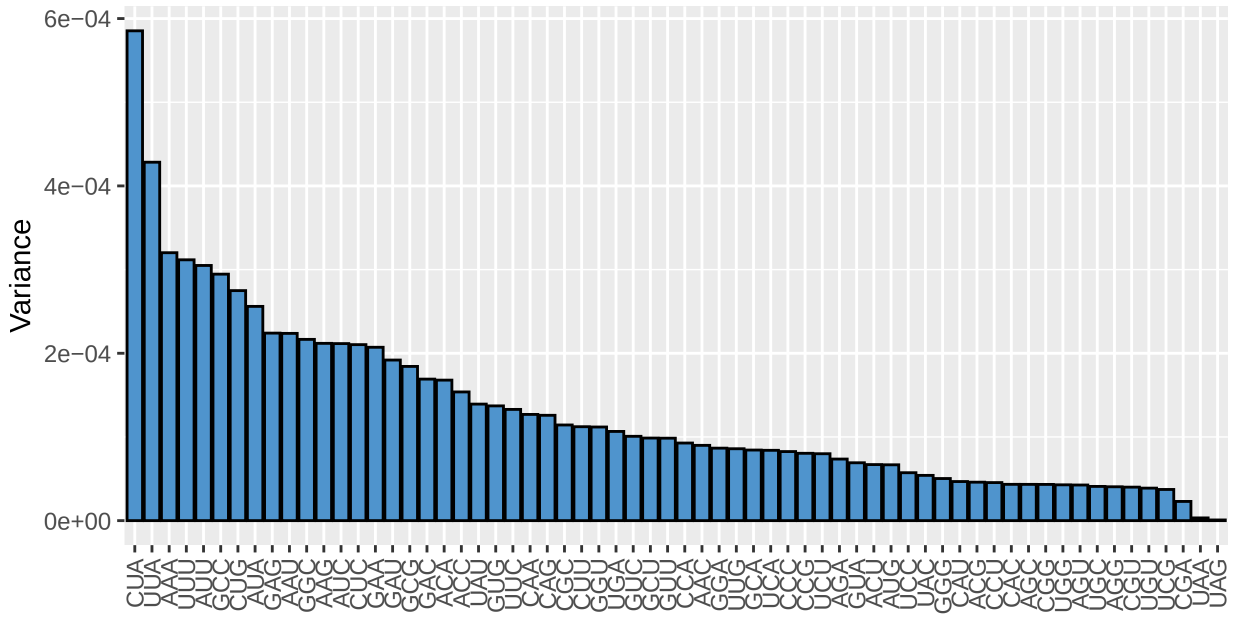
The codon frequencies are the predictors for the machine learning algorithms. The investigation of the distribution of the frequencies in the box plot below shows a great variability for each codon in the data set. The median frequencies are all in the same order of magnitude with exception of the three stop codons (UGA, UAA, UAG), whose median frequencies are about one order of magnitude lower.



One of the assumptions for the prediction of taxa from codon frequency data was that closely related taxa have a similar codon usage and distantly related taxa a distinct usage. By stratification of the codon frequency distributions by three different taxa (primate, rodent, virus) a high similarity between the codon usage of the highly related taxa primate and rodent emerges, but clear differences in comparison to the codon usage of viruses. The stop codon UGA, for instance, shows a considerably different frequency distribution in virus.

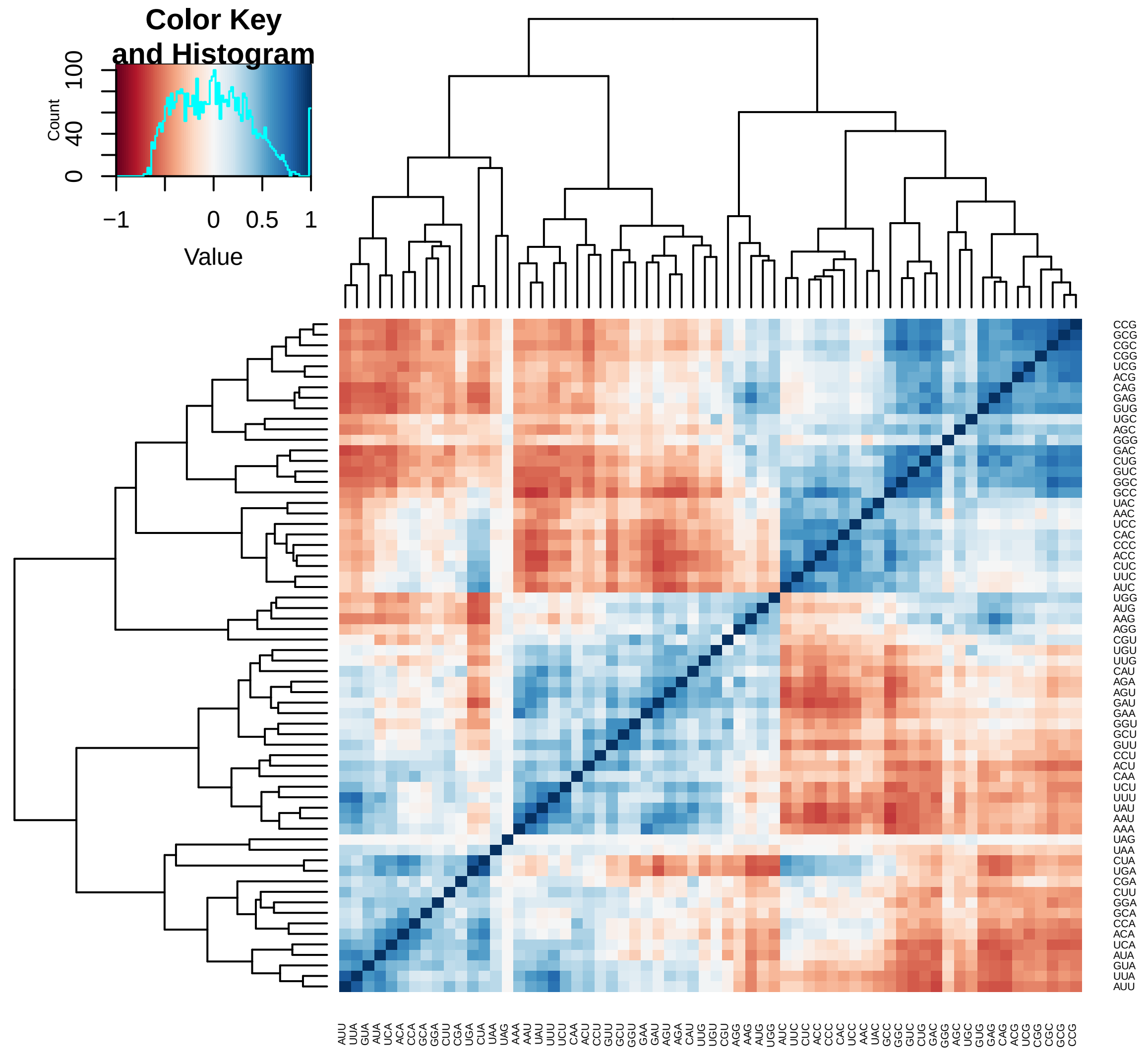


In order to distinguish different taxa in the data set, the variance of the predictors may be of great importance. The bar plot below shows the variance of the frequencies for each codon and reveals great differences. The codons CUA and UUA show by far the highest variance and suggest a high predictive power. The stop codons UAA and UAG show a very low variance and therefore suggest a low predictive power. In contrast, the stop codon UGA, which seems capable to distinguish between primates/rodents and viruses, has an average variance.



Since all frequencies of an entry add up to 1, a reduction of one codon frequency results in an increase of another one. This suggests positive or negative correlations between the codons in the data set. The heatmap of the correlations between the codons reveals clusters of codons with high correlations. The most variable codon CUA and the stop codon UGA show a high correlation of 0.84. Further high correlations become evident between the codons CCG, GCG, and CGC (0.81–0.87) and the codons AUU and UUA (0.82).

These highly correlated predictors and the two stop codons with very low variance are very good candidates for removal from the data set. However, in order to leave open the possibility to compare the codons to each other and make general statements about all codons, the predictors were not removed.



2.4 Removing Columns for Prediction

The columns `DNAtype`, `SpeciesID`, `Ncodons` and `SpeciesName`, which are dispensable for the predictions, were removed:

```
train <- train %>% select(-DNAtype, -SpeciesID, -Ncodons, -SpeciesName)
validation <- validation %>% select(-DNAtype, -SpeciesID, -Ncodons, -SpeciesName)
```

2.5 Modeling Approach

A variety of predictive models based on different mathematical procedures were trained and tuned on the train data set using the `caret` package and compared to each other. Eventually, the best performing model was applied on the validation set.

2.5.1 Caret Package

The `caret` package combines many machine learning techniques in one package under the same syntax. It allows for easy training of different methods by setting `method = "..."` and easy tuning of the parameters by the `tuneGrid` (or `tuneLength`) option. The `trControl` option controls the validation method. Additionally, a `preProcess` option allows for normalization of the data, but in this project all codon frequencies add up to 1 and therefore no normalization was necessary.

The different models were trained and tuned on the train data set using cross-validation methods. The tuning process was visualized by plotting the `train` results. The best result from the tuning process of each model was used for comparison of the model performances and the final model selection.

The final best performing model was trained without cross-validation on the train set using the fixed best performing tuning parameters. The `train` output was used for the prediction of outcomes in the validation set.

2.5.2 Accuracy as Metric

In order to evaluate the precision of the prediction models, the overall accuracy, as a measure of the overall proportion that is predicted correctly, was used. For the final model, the sensitivity (true positive rate) and the specificity (true negative rate) were used as additional accuracy metrics. All accuracy metrics are accessible through the `confusionMatrix` function of the `caret` package.

2.5.3 Cross-Validation

For the tuning of the parameters of the algorithms either a five fold or a ten fold cross-validation (CV) method was used. In the case of ten fold CV the data set is split into 10 subsets. Each of these subsets is used as a test set for the other nine subsets combined as train set. In this way ten different estimates for a given set of tuning parameters are calculated on the same data set. Averaging these estimates result in more stable final estimates. This approach is included in the `train` function of the `caret` package. The bootstrap method is the default, but in this analysis the CV was preferred in order to get faster calculation times.

2.5.4 Penalized Multinomial Logistic Regression

Logistic regression is a classification method that estimates the probability of class membership in binary classification tasks (0 or 1, yes or no). The decision which class is predicted depends on a set threshold for the probability. The logistic regression is a transformation of linear regression and therefore belongs to the family of generalized linear models (GLM). The multinomial logistic regression is an extension of this method for multiclass classification tasks. The tuning parameter of the penalized multinomial logistic regression is `decay`.

2.5.5 Linear Discriminant Analysis

The linear discriminant analysis (LDA) is a classification method that uses linear combinations of features to predict the probability of belonging to a given class. LDA uses the assumptions that the predictors are multivariate normal, the classes have homogeneous variance/covariance matrices and the correlation structure is the same for all classes. These assumptions lead to linear straight lines as class boundaries.

2.5.6 k-Nearest Neighbors

k-nearest neighbors (kNN) is a nonparametric classification method. After calculation of the distance between all samples, for each point the k-nearest points are combined and the class of this neighborhood is determined by a plurality vote. The value `k` is the tuning parameter of the kNN method.

2.5.7 Support-Vector Machines

Support-vector machines (SVM) are classification methods that identify the optimal decision boundaries that separates data points from different classes. The optimal decision boundaries might have the form of a linear straight line or have a non-linear form. In this analysis the linear SVM classifier was used for linear boundaries and the radial SVM classifier was used for non-linear boundaries. Both classifiers use `C` (cost) as tuning parameter, which determines possible misclassifications and basically imposes a penalty to the model for making an error. The radial classifier uses an additional `sigma` parameter.

2.5.8 Random Forests

Random forests is a classification method that uses bootstrap methods to randomly generate multiple independent classification trees from the same data set. A final prediction is made by averaging the predictions of all these trees. The tuning parameter `mtry` stands for the number of variables randomly sampled as candidates at each split and the parameter `ntree` is the number of trees to grow. The subsequent examination of the variable importance was carried out using the `varImp` command of the `caret` package on the `train` output.

3 Results

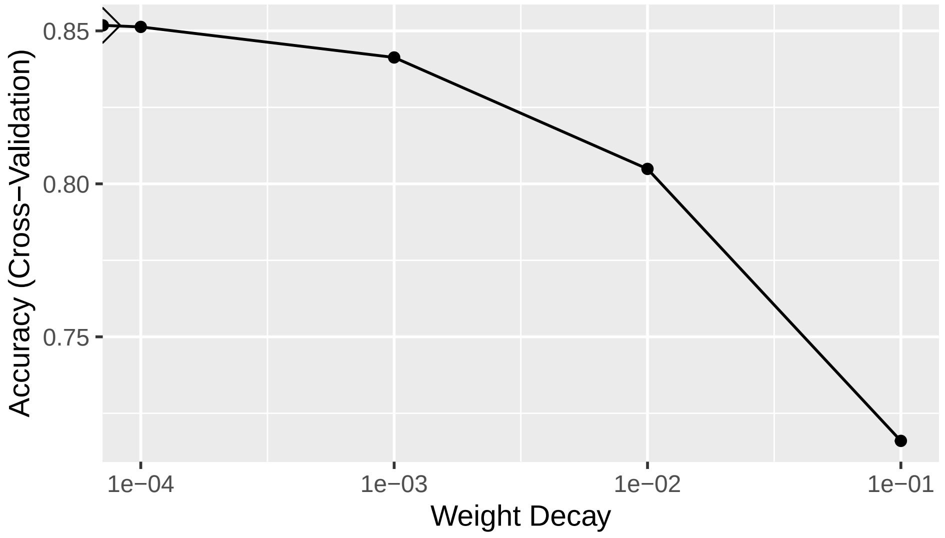
3.1 Training and Tuning of Models and Model Selection

3.1.1 Penalized Multinomial Logistic Regression Model

The multinomial logistic regression model was trained on the train data set with five different values of the tuning parameter `decay` using ten fold cross-validation:

```
set.seed(2005)
control <- trainControl(method = "cv", number = 10, p = .9)
multimodel <- train(Kingdom ~ ., data = train, method = "multinom",
                    trControl = control,
                    trace = FALSE,  # suppress iterations output
                    tuneLength = 5)
acc_multi <- max(multimodel$results$Accuracy)
```

The plot shows that 0 was the best tune for `decay`:



The application of this model yielded an overall prediction accuracy of 0.852. A table compares this result with following results:

Method	Accuracy
Multinomial Logistic Regression Model	0.852

3.1.2 Linear Discriminant Analysis Model

The linear discrimination analysis model was trained on the train data set using ten fold cross-validation:

```r
set.seed(2005)
control <- trainControl(method = "cv", number = 10, p = .9)
ldamodel <- train(Kingdom ~ ., data = train,  method = "lda",
                  trControl = control)
acc_lda <- max(ldamodel$results$Accuracy)
```

The LDA model yielded an overall accuracy of 0.806. This is still a decent accuracy but lower than the multinomial logistic regression:

Method	Accuracy
Multinomial Logistic Regression Model	0.852
Linear Discriminant Analysis Model	0.806

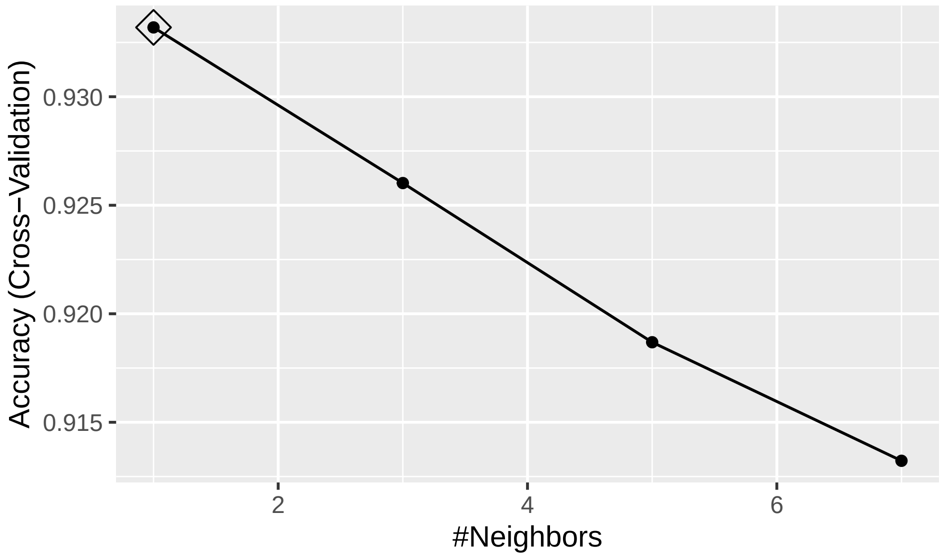
A possible explanation for the lower accuracy might be that the assumptions for multivariate normality and similar variances do not hold in this data set, as already seen in the data analysis section.

3.1.3 k-Nearest Neighbors Model

The kNN model was trained on the train data set with four different values of the tuning parameter `k` using ten fold cross-validation:

```r
set.seed(2005)
control <- trainControl(method = "cv", number = 10, p = .9)
knnmodel <- train(Kingdom ~ ., data = train, method = "knn",
                  trControl = control,
                  tuneGrid = data.frame(k = seq(1, 7, 2)))
acc_knn <- max(knnmodel$results$Accuracy)
```

The plot of the tuning process shows that `k` = 1 was the best tune:



The kNN model yielded an accuracy of 0.933. This is a strong increase in overall accuracy in comparison to the previous models:

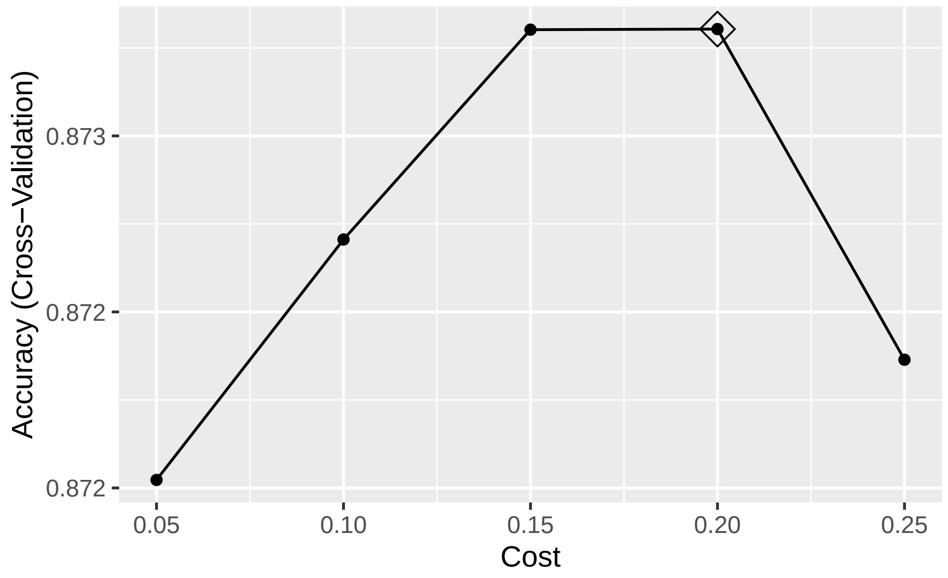
Method	Accuracy
Multinomial Logistic Regression Model	0.852
Linear Discriminant Analysis Model	0.806
k-Nearest Neighbors Model	0.933

3.1.4 Support Vector Machines with Linear Kernel Model

The SMV model with linear kernel was trained on the train data set with five different values of the tuning parameter `C` using ten fold cross-validation:

```r
set.seed(2005)
control <- trainControl(method = "cv", number = 10, p = .9)
svmlinmodel <- train(Kingdom ~ ., data = train, method = "svmLinear",
                     trControl = control,
                     tuneGrid = data.frame(C = seq(0.05, 0.25, 0.05)))
acc_svmlin <- max(svmlinmodel$results$Accuracy)
```

The plot of the tuning process shows that `C` = 0.2 was the best tune for the cost parameter:



The SVM linear model yielded an overall accuracy of 0.873. This is slightly better than the multinomial logistic regression model, but worse than the kNN model:

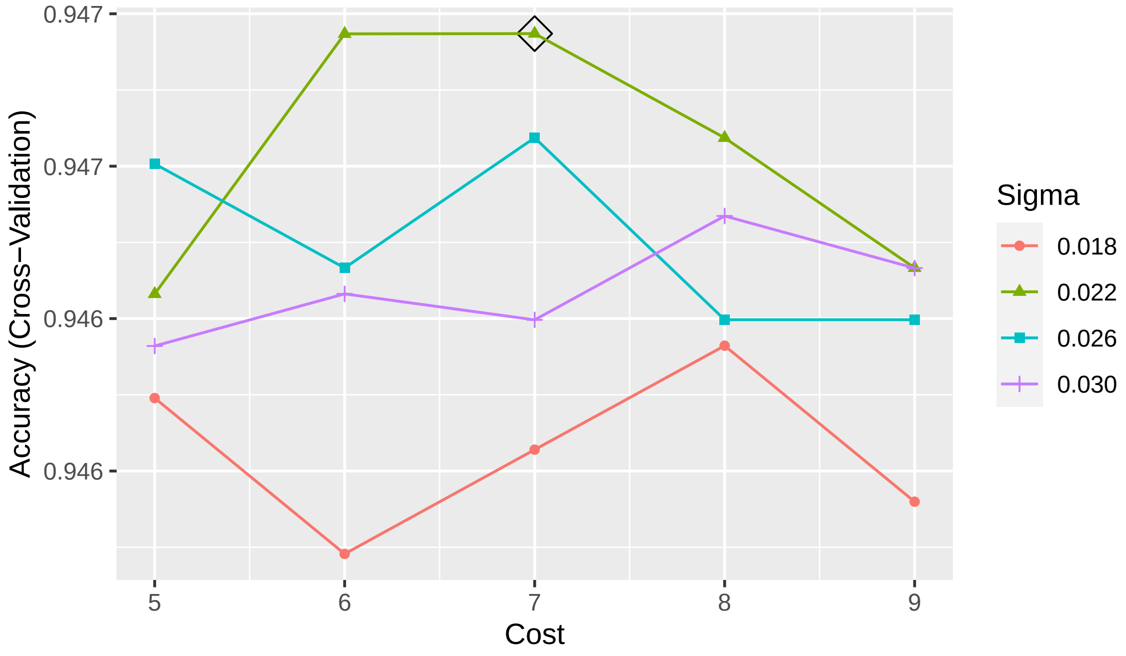
Method	Accuracy
Multinomial Logistic Regression Model	0.852
Linear Discriminant Analysis Model	0.806
k-Nearest Neighbors Model	0.933
Support Vector Machines Linear Model	0.873

3.1.5 Support Vector Machines with Radial Basis Function Kernel Model

The SMV model with radial kernel was trained on the train data set with five different values of the tuning parameter `C` and four values for the parameter `sigma` using five fold cross-validation:

```
set.seed(2005)
control <- trainControl(method = "cv", number = 5, p = .8)
svmradmodel <- train(Kingdom ~ ., data = train,  method = "svmRadial",
                     trControl = control,
                     tuneGrid = expand.grid(sigma = seq(.018, 0.030, 0.004),
                                            C = seq(5, 9, 1)))
acc_svmrad <- max(svmradmodel$results$Accuracy)
```

The plot of the tuning process shows that `C` = 7 and `sigma` = 0.022 were the best tune:



The SVM radial model yielded an overall accuracy of 0.947. The comparison to the linear classifiers, multinomial regression, LDA and SVM linear, shows that non-linear boundaries are better than linear lines to separate the different taxa in this data set. This result is even better than the kNN model:

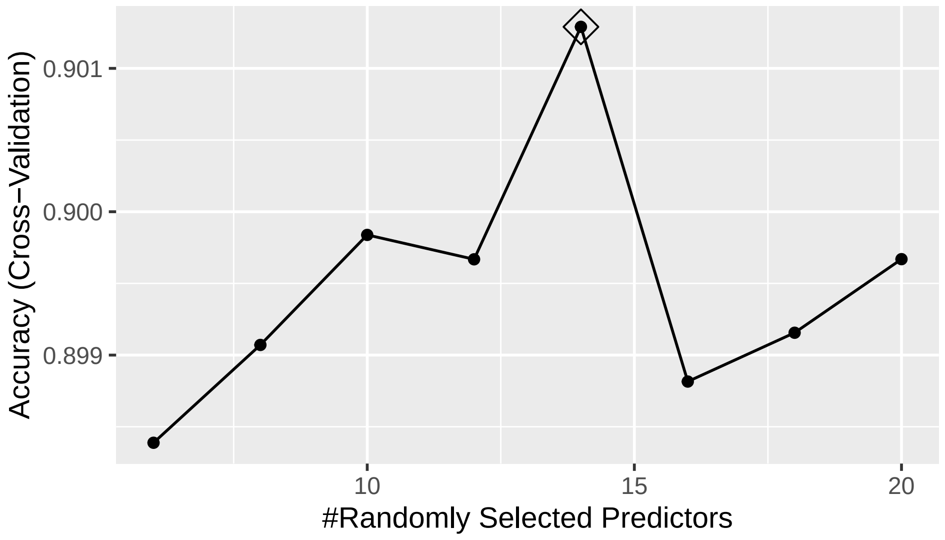
Method	Accuracy
Multinomial Logistic Regression Model	0.852
Linear Discriminant Analysis Model	0.806
k-Nearest Neighbors Model	0.933
Support Vector Machines Linear Model	0.873
Support Vector Machines Radial Model	0.947

3.1.6 Random Forests Model

The random forests model was trained on the train data set with eight different values of the, in `caret` implemented, tuning parameter `mtry` using five fold cross-validation:

```
set.seed(2005)
control <- trainControl(method = "cv", number = 5, p = .8)
mtrytune <- train(Kingdom ~ ., data = train, method = "rf",
                  trControl = control,
                  tuneGrid = data.frame(mtry = seq(6,20,2)))
```

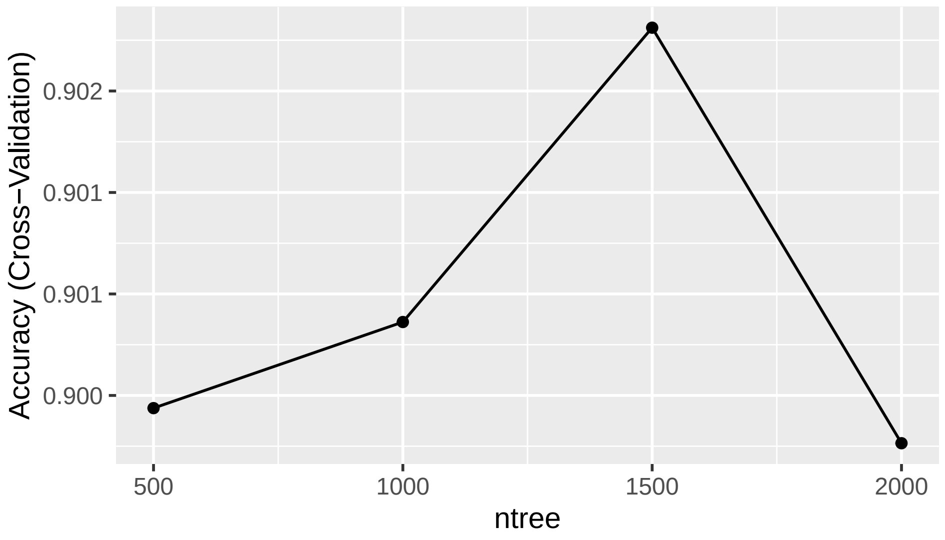
The plot of the tuning process shows that 14 was the best tune for `mtry`:



Additionally, the tuning parameter `ntree` was tuned with four different values using the `lapply` function and five fold cross-validation:

```
set.seed(2005)
control <- trainControl(method = "cv", number = 5, p = .8)
ntree <- c(500, 1000, 1500, 2000)
ntreetune <- lapply(ntree, function(nt){
  train(Kingdom ~ ., data = train, method = "rf",
        trControl = control,
        tuneGrid = data.frame(mtry = mtrytune$bestTune$mtry),
        ntree = nt)
  })
rf_accuracy <- c(ntreetune[[1]]$results$Accuracy, ntreetune[[2]]$results$Accuracy,
                 ntreetune[[3]]$results$Accuracy, ntreetune[[4]]$results$Accuracy)
acc_rf <- max(rf_accuracy)
```

The plot of the `ntree` tuning shows 1500 as the best tune:



The random forests model yielded an overall accuracy of 0.902. This result is better than the linear and the logistic regression models, but worse than the kNN and the SVM radial models:

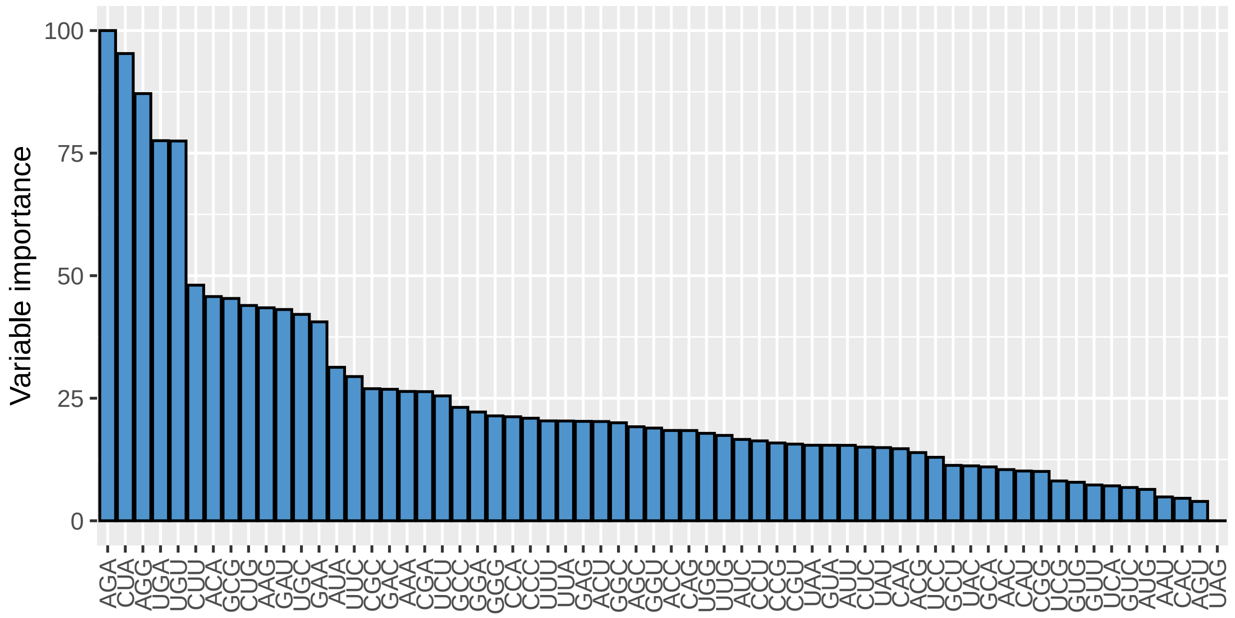
Method	Accuracy
Multinomial Logistic Regression Model	0.852
Linear Discriminant Analysis Model	0.806
k-Nearest Neighbors Model	0.933
Support Vector Machines Linear Model	0.873
Support Vector Machines Radial Model	0.947
Random Forests Model	0.902

The SVM radial model remained as the best of all the tested models and was used as the final model.

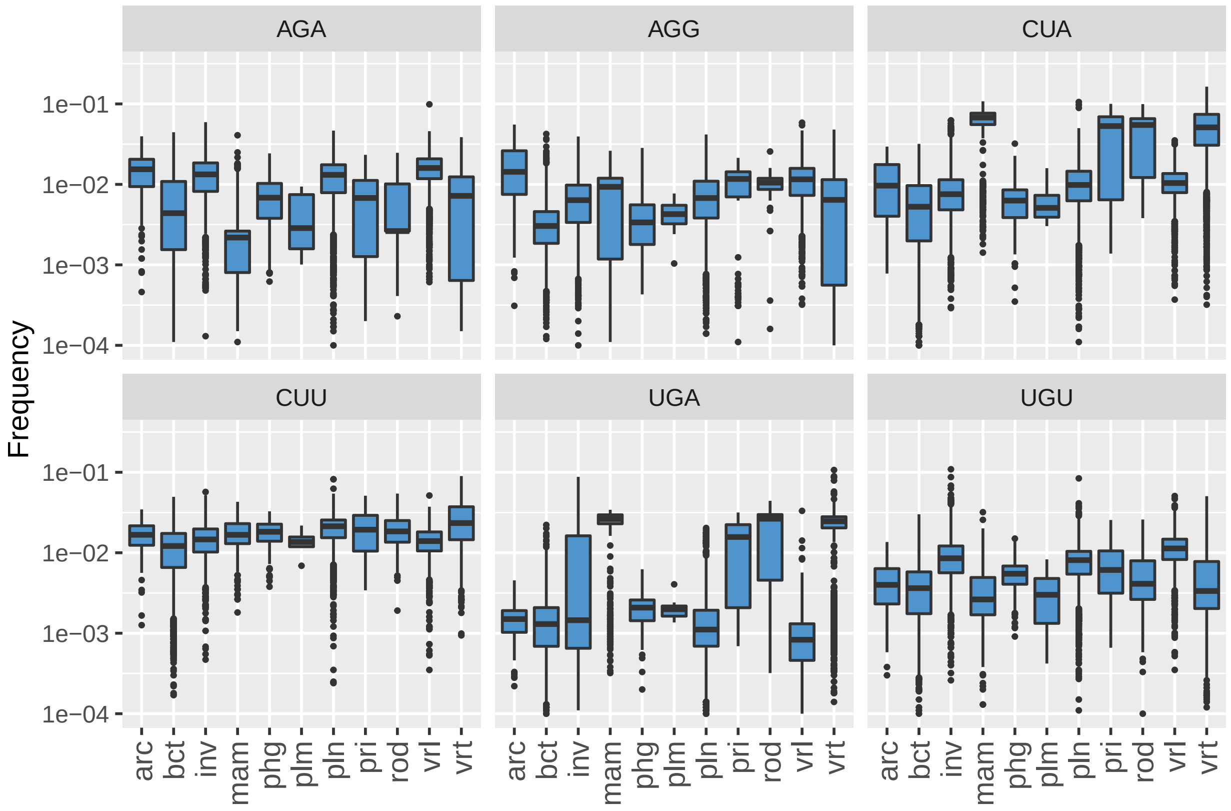
3.2 Analysis of Variable Importance

The random forests model does not perform as well as other model on this data set. However, an advantage of this model is that it allows for a well interpretable examination of the variable importance. The variable importance was calculated from the best performing random forests model and visualized in a bar plot:

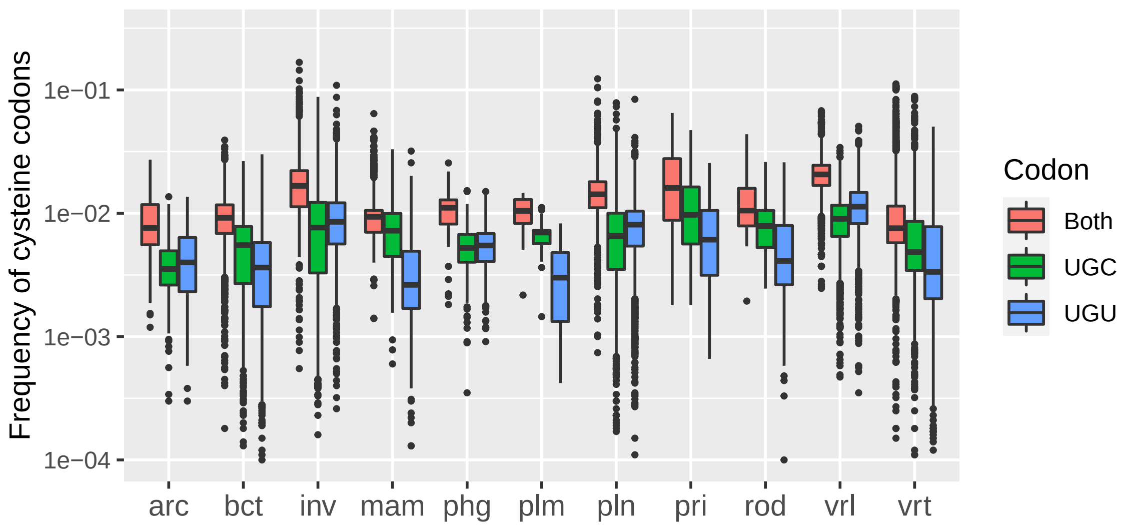
```
imp <- varImp(ntreetune[[which.max(rf_accuracy)]])
```



In this plot the codons AGA, CUA, AGG, UGU and UGA show by far the highest variable importance. By comparison of this result to the plot of the predictor variances, only the highly variable CUA and the stop codon UGA, which is highly correlated to CUA, also have a high variable importance. On the contrary, AGA, AGG and UGU have a rather low variance but a very high variable importance. The stop codons UAG and UAA showed a very low variance, but UAA still has a slightly less than average variable importance. Only UAG is last in both analyses. Hence, a high predictor variance does not automatically result in a high variable importance in this data set. The boxplot below shows the codon frequency distributions of the six codons with the highest variable importance stratified by each taxon. These codons show clear discrimination potential between different taxa.



The corresponding amino acids for these codons are leucine for CUA and arginine for AGA and AGG. Both amino acids are coded by six different codons, which allows for high flexibility, and therefore this result seems plausible. On the contrary, the codon UGU codes for the amino acid cysteine, which is only coded by two codons. This leads to the assumption that either the preference for each codon is taxon-dependent or that there is a different prevalence of cysteine in different taxa. The latter assumption is supported by the high variable importance of the second codon for cysteine, UGC. A comparative boxplot of the frequencies of both cysteine codons separate and in combination stratified by taxon reveals that both assumptions are true. There are different preferences for the cysteine codons (*green* and *blue*) between the taxa and the combined cysteine codon usage (*red*) also varies strongly between the taxa.



3.3 Application of the Final Model on the Validation Set

The SVM radial model emerged as the best performing model. In a final step, this model was trained on the train set using the tuned parameters `sigma` = 0.022 and `C` = 7 without cross-validation:

```r
finalmodel <- train(Kingdom ~ ., data=train, method="svmRadial",
                    trControl = trainControl(method="none"),
                    tuneGrid = data.frame(sigma = svmradmodel$bestTune$sigma,
                                          C = svmradmodel$bestTune$C))
```

The trained final model was used to predict the taxa in the validation hold-out set:

```r
predicted_classes <- predict(finalmodel, validation)
acc_final <- confusionMatrix(predicted_classes, validation$Kingdom)$overall["Accuracy"]
```

By comparison of the predicted taxa and the true taxa of the validation set an overall accuracy of 0.954 was achieved:

Method	Accuracy
Multinomial Logistic Regression Model	0.852
Linear Discriminant Analysis Model	0.806
k-Nearest Neighbors Model	0.933
Support Vector Machines Linear Model	0.873
Support Vector Machines Radial Model	0.947
Random Forests Model	0.902
Final Model on Validation Set	0.954

That is a very good prediction result that even surpasses the results from the model selection process. This suggests that no overtraining or oversmoothing occurred during training and tuning of the model.

3.4 Analysis of Final Model Performance

A disadvantage of the overall accuracy is that it does not take class imbalance into account. In order to get further insights into the performance of the model, the metrics sensitivity and specificity were investigated for each class separately:

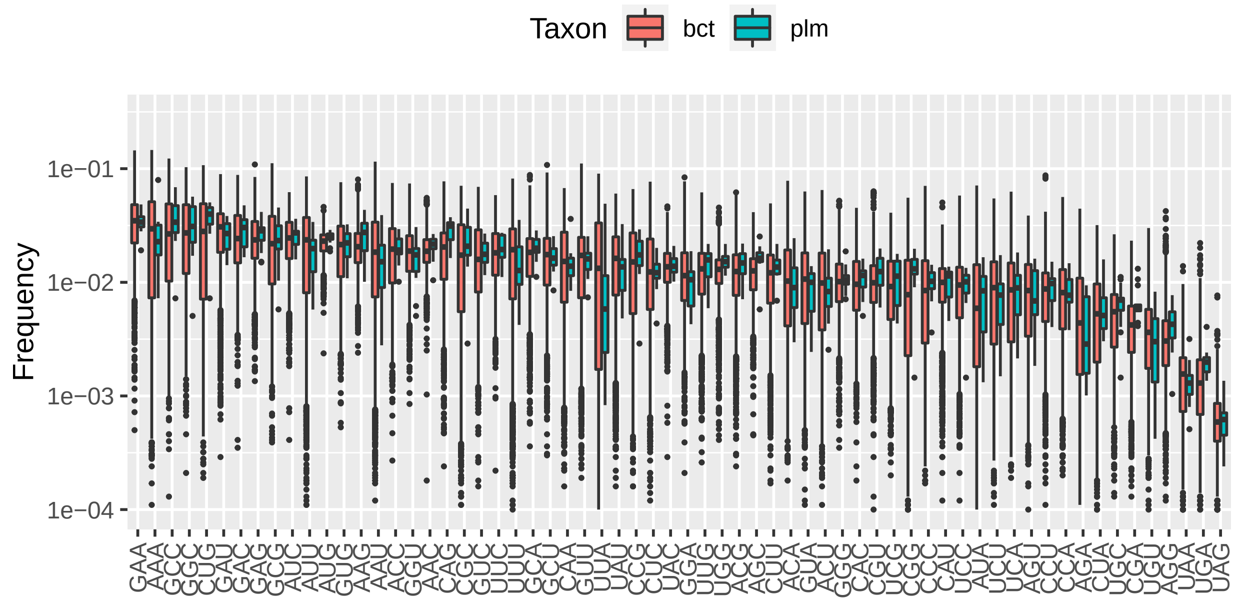
	Sensitivity	Specificity	Prevalence
Class: arc	0.923	0.999	0.010
Class: bct	0.969	0.988	0.223
Class: inv	0.911	0.991	0.103
Class: mam	0.862	0.998	0.044
Class: phg	0.773	1.000	0.017
Class: plm	0.000	1.000	0.002
Class: pln	0.964	0.993	0.194
Class: pri	0.667	0.996	0.014
Class: rod	0.864	1.000	0.017
Class: vrl	0.989	0.992	0.217
Class: vrt	0.990	0.987	0.159

The specificity (true negative rate) for all classes is close to 1, which means that only a very small fraction of the samples of each class was false positive. However, looking at the sensitivity (true positive rate) for each class reveals huge differences. Bacteria, plants, viruses and vertebrates have a very high sensitivity close to 1, which means that almost all true outcomes were predicted correctly. These are the four taxa with highest prevalence in the data set. On the contrary, plasmids have a sensitivity of 0, which means that none of the plasmids was predicted correctly. For primate samples only 67 percent and for bacteriophage samples only 77 percent were predicted correctly. Mammals, rodents and achaea were slightly better with sensitivities of 86, 86 and 92 percent, respectively. In conclusion, the very high overall accuracy is deceptive. Only the classes with the highest prevalence also showed very good prediction results. For the classes with low prevalence the algorithm provided only mediocre performance, the plasmids were not predicted at all.

The confusion matrix reveals the counts of misclassified samples (predicted taxa in the rows and reference taxa in the columns):

	arc	bct	inv	mam	phg	plm	pln	pri	rod	vrl	vrt
arc	12	1	0	0	0	0	0	0	0	0	0
bct	1	283	1	0	5	2	3	0	0	0	0
inv	0	3	123	1	0	0	4	0	0	1	1
mam	0	0	0	50	0	0	0	1	1	0	1
phg	0	0	0	0	17	0	0	0	0	0	0
plm	0	0	0	0	0	0	0	0	0	0	0
pln	0	3	4	0	0	0	244	0	0	0	0
pri	0	0	0	5	0	0	0	12	0	0	0
rod	0	0	0	0	0	0	0	0	19	0	0
vrl	0	2	3	0	0	0	1	2	0	281	0
vrt	0	0	4	2	0	0	1	3	2	2	206

The correct predictions are on the diagonal of the table, the prediction errors are outside the diagonal. There were only two plasmids in the validation set, both misclassified as bacteria. That is not surprising, because plasmids are usually present in bacterial cells and they rely on the machinery of their host cells. The boxplot below shows the very high similarity of the codon usage between bacteria and plasmids in the train set. Viruses also rely on the host machinery, which might be the cause for the confusion of viruses with far related taxa (3 of 284 misclassified). And even more, that might explain the substantial confusion of bacteriophages with bacteria (5 of 22 were misclassified as bacteria), but not with other viruses. More expected is the confusion of related taxa. For instance, primates and rodents are mainly confused with mammals and vertebrates. However, since some of the taxa are very underrepresented in the validation set, general statements are very speculative.



4 Conclusion

The codon usage varies substantially between different organisms from different taxa. The goal of this project was the development of a machine learning algorithm that predicts taxa from codon usage frequencies of different organisms with high accuracy. For this purpose, a data set of codon usage frequencies of a large sample of diverse biological organisms classified in eleven different taxa was used. Various models were trained and tuned on the train set using cross-validation and compared to each other. The investigation of the variable importance of the random forests model revealed the codons that were most relevant for the separation of the classes. Among the examined models, the support vector machines radial model turned out to be the best model for the prediction of taxa with an overall accuracy of 0.954 on the final hold-out test set. The investigation of the final predictions showed that taxa with a high prevalence in the data set were predicted with very high sensitivity and specificity. However, taxa with lower prevalence in the data set only have moderate sensitivities. For plasmids, the lowest prevalent taxon, no sample was predicted correctly.

Further improvements to this approach could be achieved by testing even more models, for instance the models of the well performing SVM family, or by an ensemble of different good performing models. Taking the different DNA types in consideration for the predictions might also improve the results. However, since some of the taxa have a very similar codon usage, such as bacteria and plasmids, a complete class separation seems hard to achieve. Nevertheless, besides the prediction problem, this data set certainly has a lot more secrets to reveal.