Assignment5

Ben Yang

2023-11-29

Introduction

With the advancement of next generation sequencing, sequencing of macromolecules have become fast and inexpensive. While so, annotations of genes and proteins remain a struggling process (Salzberg, 2019). A fundamental problem currently is that novel sequences are discovered at a faster rate than the speed of annotation, and at the same time, automated annotation methods produce inaccurate information, and thus require human intervention. This ultimately results in an overflowing number of novel sequences awaiting annotation. This is even more apparent in protein sequences as proteins require structure level understanding to annotate its function.

Protein structures can be classified into four general classes; all-alpha, which is composed of mostly alpha helices, with occasional isolated beta sheets; all-beta, which is composed of mostly beta sheets with small strands of alpha helices in the periphery; alpha+beta, which contains both alpha helices and beta sheets in no particular pattern; and alpha/beta, which is constructed with alternating alpha helices and beta sheets (Chou and Zhang, 1995). There can be a final structural class, which describes a protein with no particular unrecognizable secondary structure. These secondary structures are the result of individual amino acids in the polypeptide chain interacting non-covalently with each other. Amino acids can be categorized into four groups, polar, nonpolar, positive, and negative depending on the R group. The nonpolar group can be divided further into aliphatic, and aromatic, which is important in creating steric hindrances. These interactions work in unison to create functions such as binding, cleaving, transportation, etc.

While protein structure is a major first step in annotating a protein sequence, few studies have tried to associate structure with function in a single pipeline. Current prediction methods either tackles the structure, such as PSI-PRED and AlphaFold, or predicts the function or gene ontology of the sequence, such as PfamScan or InterProScan. In a previous study, we have identified the plausibility of using 3-mer profile of a sequence to classify different types of RNA with a neural network classifier. In this study, we will profile the amino acid sequence and compare the results of different classifiers such as random forest, K-nearest neighbors, support vector machine, and neural network to predict the protein structure class.

Methods

Software

The main softwares used in this study will be freely available in R. Libraries such as randomForest, class, e1071, and keras contains all the necessary tools to create classification models for this study. Other supporting libraries such as dplyr and gtools are used for simplification of code and data manipulation. The four classifiers compared in this study are random forest (RF), k-nearest neighbor (KNN), support vector machine (SVM) and simple neural network (NN). These four classifiers work differently to classify, but at the core, they are all supervised classifiers. RF create trees of randomly selected features at each level, while KNN takes K nearest neighbors of data points based on euclidean distance, and counts the number of categories. These two classifiers are nonlinear, compared to SVM and NN. SVM plots each point in N-dimensional space where N is the number of features and then subsequently finding the hyperplane that divides the points based

Table 1: Structural Class Count of 25PDB, FC699 and Total

class	n	class	n	class	n
a	443	a	130	a	573
b	443	b	269	b	712
c	346	c	377	c	723
d	441	d	82	d	523

on classes. NN performs matrix operations at each node to produce a probability map. These classifiers are widely used in the classification space, and gives good coverage of linear vs. nonlinear classification.

Datasets

Two datasets, 25PDB, and FC699, were used in this study. Both datasets were downloaded from Kurgan et al. (2008). 25PDB is a set of high resolution sequences from PDB with pairwise sequence identity less than 25%. There is a total of 1673 sequences in 25PDB, with roughly equal distribution of the four structure classes, which include 443 all-alpha sequences, 443 all-beta sequences, 346 alpha/beta sequences, and 441 alpha+beta sequences. This dataset is frequently used to benchmark structural prediction algorithms in literature. FC699 contains 699 sequences that share less than 40% pairwise identity with itself and less than 35% pairwise identity with 25PDB. This dataset is used as a testing data set by Kurgan et al. (2008), but we will use it as a training dataset. This data set contains 130 all-alpha sequences, 269 all-beta sequences, 377 alpha/beta sequences, and 82 alpha+beta sequences. In total, 573 all-alpha sequences, 712 all-beta sequences, 723 alpha/beta sequences, and 523 alpha+beta sequences. The mapping of structural classes are as follows, a = all-alpha, b = all-beta, c = alpha/beta, d = alpha + beta.

```r
# Import required library for this block
suppressMessages(library(dplyr))
suppressMessages(library(knitr))

# Read from CSV the protein sequences and combine them
PDB <- read.csv("25PDB.csv") %>%
  select(c("PDBid", "sequence", "actual.structural.class"))
FC <- read.csv("FC699.csv") %>%
  select(c("PDBid", "sequence", "actual.structural.class"))
names(PDB)[names(PDB) == "actual.structural.class"] <- "class"
names(FC)[names(FC) == "actual.structural.class"] <- "class"
data <- rbind(PDB, FC)

# Produce counts for each structural class
PDB.count <- PDB %>% group_by(class) %>% count()
FC.count <- FC %>% group_by(class) %>% count()
data.count <- data %>% group_by(class) %>% count()

# Create table
kable(list(PDB.count, FC.count, data.count),
      caption = "Structural Class Count of 25PDB, FC699 and Total")
```

```r
# Cleaning
rm(PDB.count, FC.count, data.count, FC, PDB)
```

Profiling

Profiling of the amino acid sequence is based on the category of individual amino acid, into one of the five groups. Nonpolar aliphatic, nonpolar aromatic, polar uncharged, positive, and negative. Category mapping is displayed in table 2.

```r
# Create table for amino acid categorization
AA <- read.csv("AAEncoding.csv")
kable(AA, caption = "Amino Acid Profile Categorization")
```

Table 2: Amino Acid Profile Categorization

class	name	letter	category
aliphatic	glycine	G	A
aliphatic	alanine	A	A
aliphatic	valine	V	A
aliphatic	leucine	L	A
aliphatic	isoleucine	I	A
aliphatic	proline	P	A
aliphatic	methionine	M	A
aromatic	phenylalanine	F	B
aromatic	tyrosine	Y	B
aromatic	tryptophan	W	B
polar	serine	S	C
polar	threonine	T	C
polar	cysteine	C	C
polar	asparagine	N	C
polar	glutamine	Q	C
negative	aspartic acid	D	D
negative	glutamic acid	E	D
positive	arginine	R	E
positive	histidine	H	E
positive	lysine	K	E

```r
# Separating the categories
mapping = list(AA$letter[c(1:7)],
               AA$letter[c(8:10)],
               AA$letter[c(11:15)],
               AA$letter[c(16:17)],
               AA$letter[c(18:20)])

# Given a sequence, return the corresponding category sequence
categorySequence <- function(sequence){
  # Split each sequence into individual characters
  sequence.split <- strsplit(sequence, "")[[1]]
  result <- "" # Category string to build upon
  # For each amino acid return the corresponding category number
  for (aminoAcid in sequence.split) {
    for (i in 1:5) {
      if (aminoAcid %in% mapping[[i]]) {
        result <- paste(result, LETTERS[i], sep = "")
      }
    }
  }
  return(result)
}

# For each sequence, obtain the category sequence and store it in data frame
for (row in 1:nrow(data)) {
  sequence <- data$sequence[row]
  data$category[row] <- categorySequence(sequence)
}

# Cleaning
rm(sequence, row, mapping, categorySequence)
```

Example: sequence “FTLQEG” would be converted into category sequence “BCACDA”. We then obtain the 3-mer frequency profile of the categorized sequence. 3-mers of the category sequence “BCACDA” is BCA, CAC, ACD, CDA each with a frequency of 0.25.

```r
# Import required library for this block
suppressMessages(library(gtools))

# Generate 3-mer data frame and convert to list of 3-mers
df <- permutations(5, 3, v = unique(AA$category), repeats.allowed = T) %>%
  as.data.frame()
threeMer.list <- paste(df$V1, df$V2, df$V3, sep = "")
# Generate empty data frame to insert data
df <- data.frame(matrix(nrow = 0, ncol = length(threeMer.list)))
colnames(df) <- threeMer.list

# Given sequence, return and append the 3-mer frequency data frame
threeMerFrequency <- function(sequence, row) {
  # Set row to all 0's
  df[row, ] <- 0

  # Sliding window to obtain the 3-mers
  sequence.split <- strsplit(sequence, "")[[1]]
  for (i in 1:(length(sequence.split) - 2)) {
    window <- paste(sequence.split[i:(i+2)], collapse = "")
    df[row, window] = df[row, window] + 1
  }
  # Divide all values of row by number of 3-mers
  df[row, ] = df[row, ]/(length(sequence.split) - 2)

  return(df)
}

# For each category sequence, obtain the 3-mer frequency
for (row in 1:nrow(data)) {
  sequence <- data$category[row]
  df <- threeMerFrequency(sequence, row)
}

# Adding name and class to frequency data frame
```

```r
df$class <- data$class
df$PDBid <- data$PDBid

# Print sample data frame
kable(df %>% select(c(127, 126, 1:3)) %>% head(n = 5),
      caption = "Sample 3-mer Frequency Probability Data Frame")
```

Table 3: Sample 3-mer Frequency Probability Data Frame

PDBid	class	AAA	AAB	AAC
1A1W_	a	0.0449438	0.0000000	0.0112360
1A56_	a	0.1265823	0.0126582	0.0632911
1A6M_	a	0.0805369	0.0067114	0.0335570
1AB3_	a	0.1046512	0.0000000	0.0465116
1ABV_	a	0.0757576	0.0151515	0.0378788

```r
# Cleaning
rm(AA, threeMer.list, row, sequence, threeMerFrequency)
```

Data Partitioning

We partition the data into 80% training and 20% testing data. To eliminate randomness, all classifiers will use the same training and testing data for classification. After partitioning, we obtain 2019 training sequences, and 512 testing sequences.

```r
set.seed(42) # Setting seed for reproducibility

# Create a list of 80% 1's and 20% 2'. 1 = Training, 2 = Testing.
indices <- sample(2, nrow(df), replace = T, prob = c(0.8, 0.2))
train <- df[indices == 1, ] %>% select(-PDBid)
test <- df[indices == 2, ] %>% select(-PDBid)

# Cleaning
rm(indices)
```

Results and Discussion

Random Forest

```r
# Import required library for this block
suppressMessages(library(randomForest))

# Constructing model
train$class <- factor(train$class)
classifier.RF <- randomForest(class~., data = train)
predict.RF <- predict(classifier.RF, newdata = test[1:125])

# Error of classifier
```

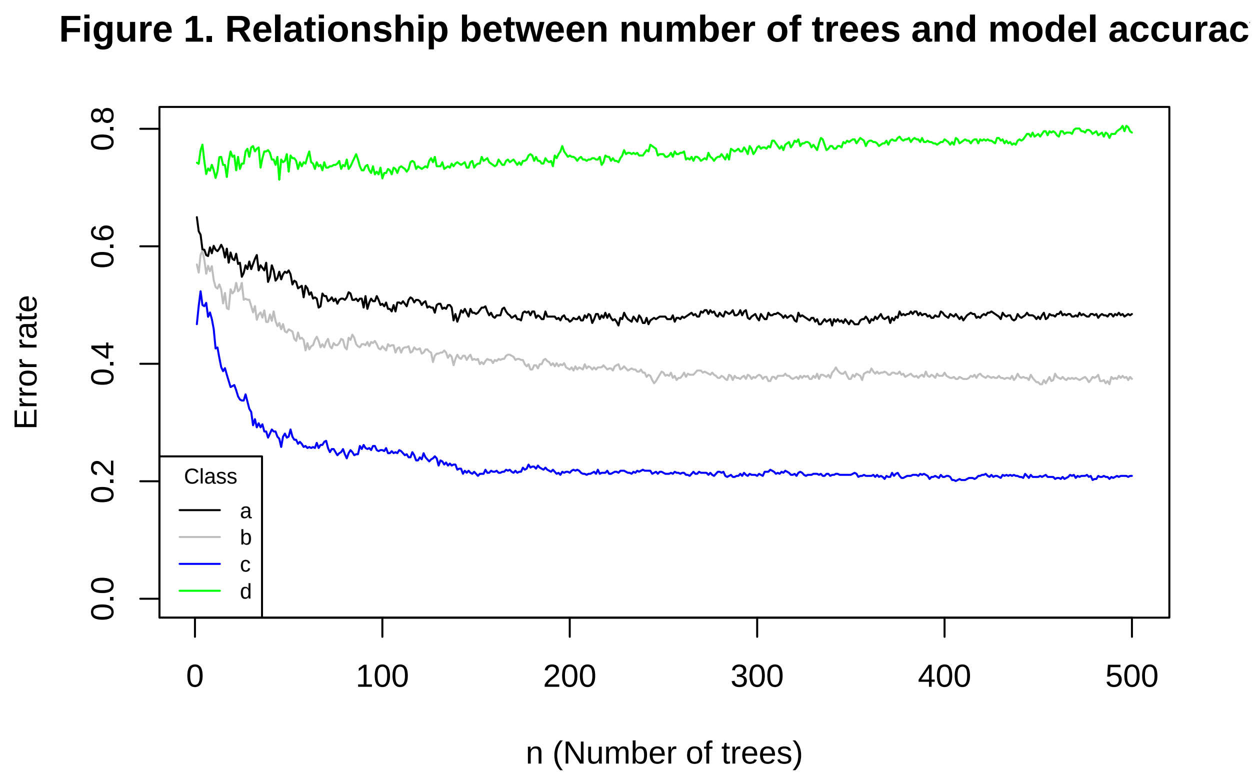
Table 4: Misclass error rate and Confusion Matrix of Random Forest Classifier (ntree = 500)

class	misclass
a	0.4956522
b	0.3269231
c	0.2142857
d	0.8620690
total	0.4218750

	a	b	c	d
a	58	28	14	15
b	19	105	22	10
c	11	12	121	10
d	29	25	21	12

```r
test$RF <- predict.RF != test$class
misclass.RF <- test %>% select(class, RF) %>% group_by(class) %>%
  summarise(misclass = mean(RF))
misclass.RF[nrow(misclass.RF) + 1, ] <- list("total",
                                             mean(predict.RF != test$class))

# Confusion matrix on testing data
confusionMatrix.RF <- table(test[, 126], predict.RF)

kable(list(misclass.RF, confusionMatrix.RF),
      caption = "Misclass error rate and Confusion Matrix of Random Forest Classifier (ntree = 500)")
```

```r
errorPerN <- classifier.RF[["err.rate"]] %>% as.data.frame() %>% select(-OOB)

# Plotting model
plot(rownames(errorPerN), ylim = c(0, max(errorPerN)),
     type = "n", cex = 1, pch = 20,
     xlab = "n (Number of trees)", ylab = "Error rate",
     main = "Figure 1. Relationship between number of trees and model accuracy")

colors <- c("black", "grey", "blue", "green")
for (column in seq_along(errorPerN)) {
  lines(x = rownames(errorPerN), y = errorPerN[, column],
        type = "l", cex = 1, pch = 20, col = colors[column])
}

legend("bottomleft", legend = c("a", "b", "c", "d"),
       col = colors, title = "Class", cex = 0.70, lty = 1)
```



```r
# Importance of features
importance.RF <- importance(classifier.RF) %>% as.data.frame()
importance.RF$feature <- rownames(importance.RF)
rownames(importance.RF) <- NULL
importance.RF <- importance.RF[order(-importance.RF$MeanDecreaseGini), ]
kable(importance.RF %>% select(feature, MeanDecreaseGini) %>% head(n = 10),
      caption = "Importance of features sorted by highest to lowest")
```

Table 5: Importance of features sorted by highest to lowest

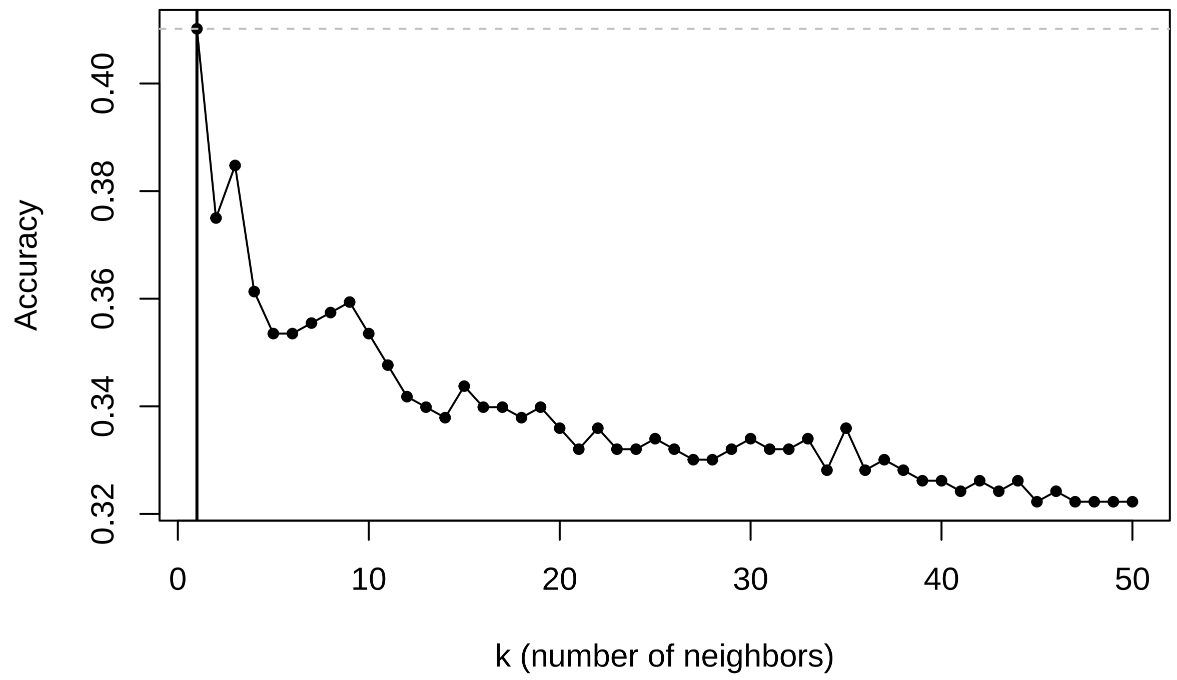
	feature	MeanDecreaseGini
1	AAA	40.85261
53	CAC	27.80911
4	AAD	26.34958
2	AAB	24.38875
76	DAA	24.35244
5	AAE	22.22015
11	ACA	22.21991
26	BAA	21.85142
6	ABA	21.70726
101	EAA	21.50994

The classification by Random Forest classifier of 500 trees is not optimal. For classes a, b, and c, the classifier was able to predict with majority of values being accurate. The prediction of class d is by far the worst with error percent as high as over 80%. The average error of the classifier around 40%, that changes with randomness of the model. This is summarized in table 4. Table 5 shows the top 10 important features determined by the classifier. We observe that feature "AAA", a triplet of non-polar aliphatic amino acids has an importance of 42%. This may be due to the flexibility of that chain being able to accomodate multiple protein folds without steric restriction.

K-Nearest Neighbor

```r
# Import required library for this block
suppressMessages(library(class))

# Scaling training and testing data sets
train.scale <- scale(train[, 1:125])
test.scale <- scale(test[, 1:125])

# Determine the best K https://daviddalpiaz.github.io/r4sl/k-nearest-neighbors.html
accPerK <- rep(x = 0, times = 50)
for (i in 1:50) {
  pred <- knn(train = train.scale, test = test.scale, cl = train$class, k = i)
  accPerK[i] = mean(test$class == pred)
}

# plot accuracy vs choice of k
plot(accPerK, type = "o", cex = 1, pch = 20,
     xlab = "k (number of neighbors)", ylab = "Accuracy",
     main = "Figure 2. Relationship between number of neighbors and model accuracy")
# add lines indicating k with best accuracy
abline(v = which(accPerK == max(accPerK)), lwd = 1.5)
# add line for max accuracy seen
abline(h = max(accPerK), col = "grey", lty = 2)
```

Figure 2. Relationship between number of neighbors and model accur



```r
# Model
classifier.KNN <- knn(train = train.scale,
                      test = test.scale,
                      cl = train$class,
                      k = which(accPerK == max(accPerK)))

# Error of classifier
test$KNN <- classifier.KNN != test$class
misclass.KNN <- test %>% select(class, KNN) %>% group_by(class) %>%
  summarise(misclass = mean(KNN))
misclass.KNN[nrow(misclass.KNN) + 1, ] <- list("total",
                                               mean(classifier.KNN != test$class))

# Confusion matrix on testing data
confusionMatrix.KNN <- table(test[, 126], classifier.KNN)

kable(list(misclass.KNN, confusionMatrix.KNN),
      caption = "Misclass error rate and Confusion Matrix of K-Nearest Neighbor Classifier (k = 1)")
```

The classification of K-Nearest Neighbor classifier performed worse than Random Forest classifier, with an average error of 59%, which unfortunately means it classifies more incorrects than corrects. KNN performed the best on class c with error of 17%, this is likely due to the distinct feature of class c sequences which contains alternating alpha helices and beta sheets. We further observe that the accuracy of KNN decreases with increasing k, this means that k = 1 will be the best performing model.

Table 6: Misclass error rate and Confusion Matrix of K-Nearest Neighbor Classifier (k = 1)

class	misclass
a	0.8000000
b	0.6987179
c	0.1753247
d	0.8505747
total	0.5898438

	a	b	c	d
a	23	18	67	7
b	15	47	81	13
c	13	9	127	5
d	6	12	56	13

Table 7: Misclass error rate and Confusion Matrix of Support Vector Machine Classifier (linear)

class	misclass
a	0.5739130
b	0.4294872
c	0.4220779
d	0.8965517
total	0.5390625

	a	b	c	d
a	49	16	31	19
b	21	89	23	23
c	30	19	89	16
d	28	28	22	9

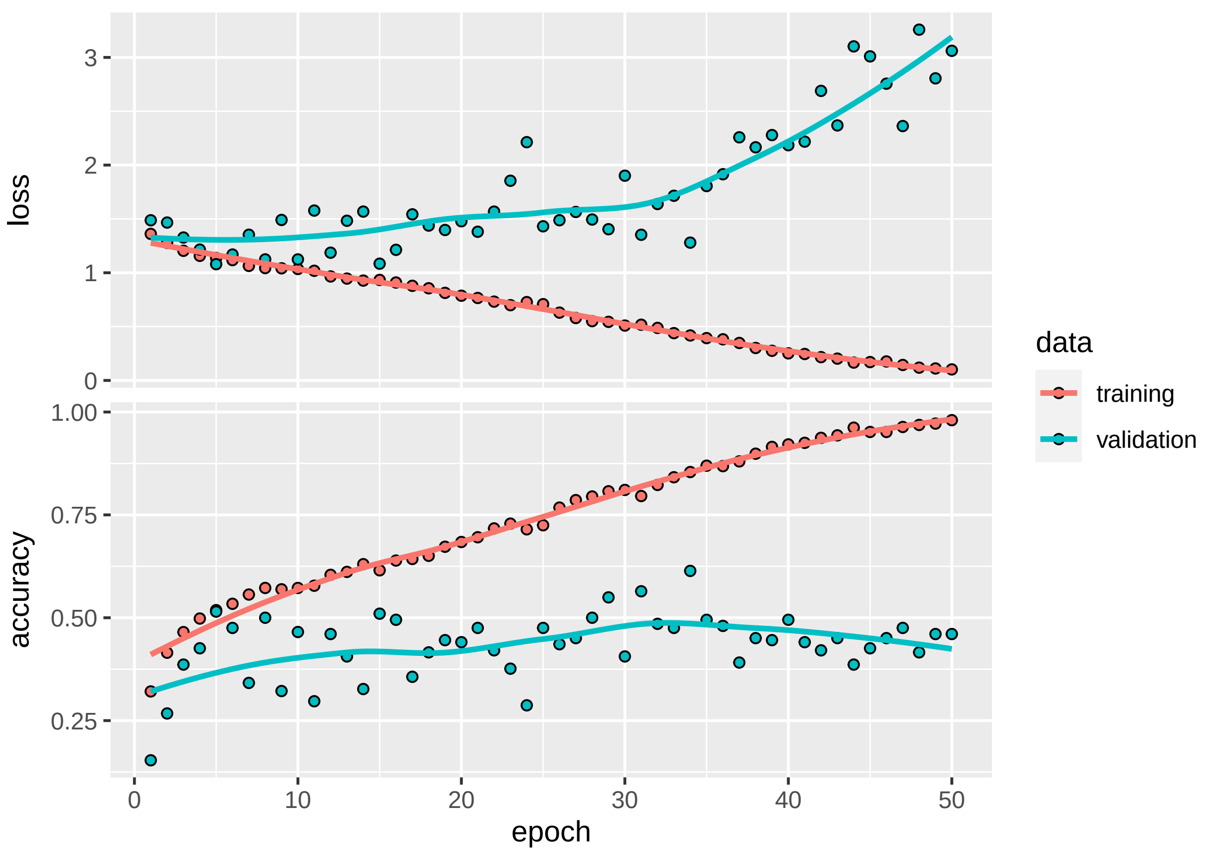
Support Vector Machine

```r
# Import required library for this block
suppressMessages(library(e1071))

classifier.SVM <- svm(class~., data = train,
                      type = "C-classification", kernel = "linear")
predict.SVM <- predict(classifier.SVM, newdata = test[1:125])


# Error of classifier
test$SVM <- predict.SVM != test$class
misclass.SVM <- test %>% select(class, SVM) %>% group_by(class) %>%
  summarise(misclass = mean(SVM))
misclass.SVM[nrow(misclass.SVM) + 1, ] <- list("total",
                                               mean(predict.SVM != test$class))

# Confusion matrix on testing data
confusionMatrix.SVM <- table(test[, 126], predict.SVM)

kable(list(misclass.SVM, confusionMatrix.SVM),
      caption = "Misclass error rate and Confusion Matrix of Support Vector Machine Classifier (linear)
```

Support vector machine classifier performed poorly as well with a grand misclass error rate of 54%. However, SVM also performed poorly in classifying class d structures like the previous classifiers, but surprisingly it also performed worse in classifying class c structures, which the previous two classifiers excelled on. SVM performed better than KNN, but worse than RF in classifying class a and b. This is surprising, but perhaps it is due to the distinct feature of class a and b are not apparent in the category string.

Neural Network

```r
# Import required library for this block
suppressMessages(library(keras))

# Function that convert to one-hot encoding matrix and then into data frame
oneHot <- function(df) {
    df$a <- as.integer(df$class == "a")
    df$b <- as.integer(df$class == "b")
    df$c <- as.integer(df$class == "c")
    df$d <- as.integer(df$class == "d")
    return(df)
}

map <- c("a", "b", "c", "d")
reverseOneHot <- function(x) {
  index <- match(1, x)
  return(map[index])
}

train.OH <- oneHot(train)
test.OH <- oneHot(test)

train.x <- subset(train.OH, select = 1:125) %>% as.matrix()
train.y <- subset(train.OH, select = c("a", "b", "c", "d")) %>% as.matrix()
test.x <- subset(test.OH, select = 1:125) %>% as.matrix()
test.y <- subset(test.OH, select = c("a", "b", "c", "d")) %>% as.matrix()

classifier.NN <- keras_model_sequential() %>%
  layer_dense(units = ncol(train.x), activation = "relu", input_shape = ncol(train.x)) %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 2048, activation = "relu") %>%
  layer_dense(units = ncol(train.y), activation = "softmax") %>%
  compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = "accuracy")

# Obtain and plot the history of training
history <- fit(classifier.NN, train.x, train.y, epochs = 50, batch_size = 128, validation_split = 0.1,
plot(history, main = "Figure 3. Neural Network Learning Curves")
```



```
# Confusion matrix on testing data
result <- data.frame(round(predict(classifier.NN, test.x)))
```

```
## 16/16 - 0s - 72ms/epoch - 5ms/step
```

```
test$predict <- apply(result, 1, reverseOneHot)
confusionMatrix.NN <- table(test[, c("class", "predict")])

# Error of classifier
test$NN <- test$predict != test$class
misclass.NN <- test %>% select(class, NN) %>% group_by(class) %>%
  summarise(misclass = mean(NN, na.rm = T))
misclass.NN[nrow(misclass.NN) + 1, ] <- list("total",
                                              mean(test$predict != test$class,
                                                   na.rm = T))

kable(list(misclass.NN, confusionMatrix.NN),
      caption = "Misclass error rate and Confusion Matrix of Neural Network Classifier (125 X 512 X 2048
```

As with the previous classifiers, a simple neural network also produced similar results with total error rate near 50%. With increasing epochs, accuracy stagnates at 50% at around 30 epochs. Similarly, NN also struggled with class d structures, while also performing poorly in classifying class c structures like the SVM. A common trend observed between the classifiers is that they perform poorly on class d structures. This, in conjunction of the poor classification on class a and class b structures gives insight that 3-mer of the categorical strings do not provide sufficient distinction between features for classifier to distinguish. Unlike

Table 8: Misclass error rate and Confusion Matrix of Neural Network Classifier (125 X 512 X 2048 X 4)

class	misclass
a	0.4642857
b	0.4932432
c	0.4285714
d	0.6867470
total	0.5000000

	a	b	c	d
a	60	11	6	35
b	27	75	8	38
c	27	10	84	26
d	29	18	10	26

linear classifiers, nonlinear classifiers performed well on class c structures by a wide margin, this is evidence that there is a nonlinear feature in the 3-mer of categorical strings that can be used for distinction. Although the attempt at classifying protein structure class was unsuccessful, we were able to observe that class c structures have a distinct feature that can be classified. Furthermore, RF classifier determined “AAA” is of high importance, an amino acid triplet that is known to have high flexibility in a protein. This agrees with the notion that the alternation between alpha and beta structure requires flexible amino acids for creating small turns and loops.

There are two major flaws in this study. One of which is the idea of a categorical string. Amino acids individually exists because of their function. Each amino acid cannot be easily replaced by another without causing minor or major structural changes. Thus the notion that amino acids can be categorized to predict structure is perhaps incorrect. Another problem is the samples. With 25PDB, each protein is dissimilar to each other, thus, we will produce dissimilar categorical strings. With limited samples, using higher k for k-mer would decrease the frequency of each k-mer appearing, while using lower k would result in not enough features. Perhaps with 10-fold the sample size, classifiers will have more confidence in their prediction.

The next steps of research would involve obtaining more protein from perhaps a higher similarity database like 85PDB to evaluate the hypothesis of categorical strings. Furthermore, we can experiment with different k-mers to find other features that may be missed by 3-mers. However, with the current results, I do not believe further pursuing the hypothesis of categorical strings is a good idea. With more protein sequences, we may just use the k-mer of the protein sequence instead.

Reflection

The most difficult part of the assignment was generating the idea. Completing the idea, although time consuming, did not take nearly as long as the generating the idea itself. I personally dislike doing a project on a theme that has been done, I like to explore at novelties, new potentials and possibilities, or ideas that may be overlooked by other researchers. My thesis involves machine learning, either classification or prediction, so this was a great opportunity to obtain hands-on experience in this field. Assignment 5 has been a very rushed assignment compared to the rest, timeline wise. This was due to the little downtime that we had between assignments, and the inclusion of assignment 4 that was layered on top. I do suggest future students to start this assignment earlier, at least earlier than me. I also encourage future students to explore beyond what was done, and include novelties in these assignments, it really makes this project oriented course like your own.

References

Chou, K.-C., & Zhang, C.-T. (1995). Prediction of protein structural classes. *Critical Reviews in Biochemistry and Molecular Biology*, *30*(4), 275–349. https://doi.org/10.3109/10409239509083488

Dalpiaz, D. (2017, August 28). *R for statistical learning.* daviddalpiaz.github.io. https://daviddalpiaz.github.io/r4sl/k-nearest-neighbors.html

GeeksforGeeks. (2020a, June 5). *Random Forest approach in R programming.* GeeksforGeeks. https://www.geeksforgeeks.org/random-forest-approach-in-r-programming/

GeeksforGeeks. (2020b, June 22). *K-nn classifier in R programming*. GeeksforGeeks. https://www.geeksforgeeks.org/k-nn-classifier-in-r-programming/

GeeksforGeeks. (2023, June 12). *Classifying data using support vector machines(svms) in R*. GeeksforGeeks. https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-r/

Kurgan, L., Cios, K., & Chen, K. (2008). SCPRED: Accurate prediction of protein structural class for sequences of twilight-zone similarity with predicting sequences. *BMC Bioinformatics*, *9*(1). https://doi.org/10.1186/1471-2105-9-226

Salzberg, S. L. (2019). Next-generation genome annotation: We still struggle to get it right. *Genome Biology*, *20*(1). https://doi.org/10.1186/s13059-019-1715-2