Final Project - Oliver Zhao - soz63

First we load the relevant libraries and datasets for this project:

```
# Import libraries
library(randomForest)
library(ggplot2)
library(mosaic)
library(tidyverse)
library(caret)
library(ltm)
library(matrixStats)

# Import NCB dataset
Data <- read.csv("~/Github/SDS323_Spring2020/NCBFeatures.csv")

# Set random seed for entire markdown document
set.seed(123)
```

Abstract

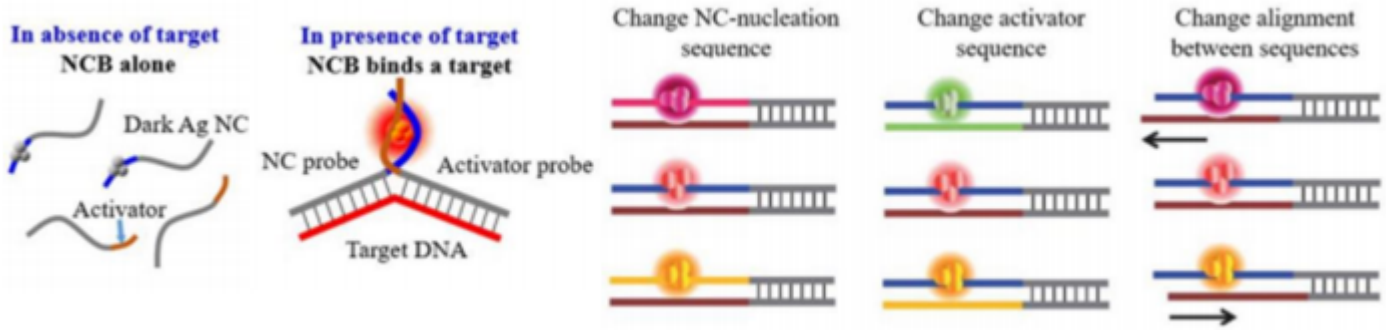
NanoCluster Beacons (NCBs) are an emerging class of fluorescent activatable molecular probes with the ability to detect proteins, metabolites, and cancer cells. The fluorescent response of the NCBs can be fine-tuned by altering the DNA sequence of the associated activator strand, allowing NCBs to be customized as versatile probes. However, with the DNA activator strand consisting of 18 nucleotides, even modern high-throughput methods that are able to screen ~12,000 sequences at once are not able to fully characterize the possible variants of NCBs.

While developing the appropriate DNA sequence to target specific molecules remains a challenging task, another relevant issue is ensuring that the NCBs are sufficiently bright in order to act as effective fluorescent markers. To address this challenge, we create a random forest model that predicts whether a NCB is bright enough based on its DNA sequence, achieving an accuracy of XX.XX%, sensitivity of XX.XX%, and specificity of XX.XX%. The features are extracted through a third-party software package, in which we implement correlation based feature selection and implicit feature selection to isolate relevant features for our final model. The model accelerates the development of new NCBs by identifying whether the proposed DNA sequence satisfies the desired fluorescent intensity.

Introduction

Silver NanoCluster Beacons (Ag NCBs) are DNA-templated molecules consisting of several silver atoms, serving as cheap and customizable molecular markers. Under normal conditions, NCBs are dark and elicit no fluorescent signal. Upon binding to an appropriate molecular target, the conformation of the DNA-templated NCB changes to emit strong fluorescent signals (Figure 1). The NCB DNA sequence can be customized to bind to different targets, such as specific proteins or molecules. In addition to the NCB DNA sequence, there is a seperate activator DNA sequence. The activator DNA sequence does not affect what type of molecules the NCB binds to, but rather, it controls how bright the NCBs are when bound to the target molecule. The design rules of the activator DNA sequence are unclear, reducing the development of the activator DNA sequence to laborious and often frustrating trial-and-error. Recent high-throughput methods that can screen up to approximately 12,000 unique activator DNA sequences help mitigate this challenge, but is exceedingly expensive and requires significant materials such as

advanced microscopy imaging systems and customized molecular assays. In addition, with the activator DNA sequence being 18 nucleotides long, the ability to screen 30,000 unique activator DNA sequences is still a small fraction of the possible 4^{18} activator sequences. To bridge this gap, we develop a machine learning model that can predict the brightness of the NCBs based on the activator DNA sequence, with the training and testing data derived from these expensive, high-throughput experiments.



Methods

Dataset

The Chip-Hybridized Associationg Mapping Platform (CHAMP) allows for the screening of 12,288 distinct DNA activator sequences, each with roughly 100-200 duplicate clonal DNA clusters. The resulting data consists of activator DNA sequences ranked in descending order of fluorescent brightness. To avoid the ambiguity of “bright” and “dark” DNA activator sequences, we classify “bright” sequences as DNA activator sequences belonging to the top 30% ranked sequences and “dark” sequences as DNA activator sequences belonging to the bottom 30% ranked sequences, with the middle 40% of sequences discarded. The feature extracted dataset is provided as “NCBFeatures.csv”. The data is collected from the NanoBiosensors and Molecular Tracking Lab at UT Austin.

Feature Extraction

With the raw dataset containing only a list of activator DNA sequences in rank order of brightness, different DNA features must still be extracted. The DNA features are extracted using a third-party Python software package - PyFeat - which is available at https://github.com/mrzResearchArena/PyFeat/ (https://github.com/mrzResearchArena/PyFeat/). Through this software, we extract 16,171 different DNA features (Figure 2), stored in the dataset we use (“NCBFeatures.csv”). Most features consist of different DNA motifs, where **N** indicates a variable nucleotide and **_** indicates a gap of 0 to 5 nucleotides. For example, the feature **A_T** may represent **ATT**, **AT**, or **AGCGCT**. For each of these motif features, the value indicates the number of occurences of each motif. Other features include basic holistic characterizations of the activator DNA sequence, such as the % of bases that are G or C (GC content) or the AT-GC ratio.

Features	Description	Number of Features
DNA Motif Structure (No Blanks)	**N, NN, . . ., NNNNN**	1,364
DNA Motif Structure (With Blanks)	**N_N, N_NN, NN_N, NNN_N N_NNN, NN_NN, NN_NNN, NNN_NN**	14,800
Z Curve	Decomposes DNA Sequence in 3D Curve	3
GC Content	% of Bases that are G or C	1
AT/GC Ratio	$(\sum A + \sum T)/(\sum G + \sum C)$	1
GC Skew	$(\sum G - \sum C)/(\sum G + \sum C)$	1
AT Skew	$(\sum A - \sum T)/(\sum A + \sum T)$	1

Feature Selection

Features that contain the same value across all data points were removed. Then the point-biserial correlation coefficient for each feature, relative to the DNA activator sequences brightness class, was calculated. Features with a correlation coefficient between -0.05 to 0.05 were excluded due to the likelihood of having lower discriminatory power. Finally, a random forest model with 50 trees was trained on the full dataset with the remaining features that were not removed due to low discriminatory power. Afterwards, the importance of each feature was calculated for each of these features. All other hyperparameters were default settings. Due to the high dimensionality of the dataset, overfitting is a possibility. To address this issue, features were excluded based on a feature importance threshold, ranging from 0.10 to 15. Then, these feature sets were used to train random forest models with 50 trees each with 10-fold cross validation to determine which number of features is optimal.

Model Hyperparameter Tuning

Once the optimal number of features - based on an feature importance threshold - was determined, we implement a traditional grid search of the hyperparameters `mtry` and `maxnodes`, which control the number of variables tested at each node and the maximum number of terminal nodes, respectively. These two hyperparameters are the most likely to affect the performance of the model, so only these two hyperparameters were tuned to avoid having to test too many model variants. Once the ideal `mtry` and `maxnodes` is determined, we expand the number of trees from 50 to 500, as a higher number of trees elicits better performance up to a certain point. Previously, the number of trees were limited to 50 to avoid heavy computational burden. A random forest model was selected because it is easily scalable with increasing number of trees, can be trained in parallel (opposed to boosting methods) to reduce computational burden, and requires minimal feature selection and feature scaling.

Results

Feature Selection

Of the 16171 extracted features, 6512 features contain homogenous values for all data points and are removed, resulting in 9659 remaining features. After excluding features with a point-biserial correlation coefficient between -0.05 to 0.05, another 6333 features were excluded to result in 3326 remaining features. Finally, after implicit feature selection through a random forest model, the number of features ranged from 67 to 1733 features depending on the feature importance cutoff threshold (Figure 3).


16171 Features
Homogenous Features Removed (6512)
9658 Features
Low Correlation Features Removed (6333)
3324 Features
Implicit Feature Selection
66-1732 Features

```
# Preprocess dataset
Data$X1 <- as.factor(Data$X1)  # factorize class
n <- nrow(Data)                # number of data points
f <- ncol(Data)-1              # number of features

# Correlation Feature Selection: Remove Low Point-Biserial Correlations
Cor <- vector("double", f)
CorData <- Data
for (i in f:1){
  Cor[i] <- biserial.cor(CorData[,c(i)],CorData$X1)
  if (is.na(Cor[i])){
    CorData <- CorData[,-c(i)]
  }
  else{
    if (abs(Cor[i]) < 0.05){
      CorData <- CorData[,-c(i)]
    }}
}

# Implicit Feature Selection: Train Random Forest on Reduced Dataset
rf <- randomForest(X1 ~ .,
                   data= CorData,
                   ntree = 50)
Features <- importance(rf)

# Model Complexity Analysis for Overfitting and Underfitting
CV <- 10
Folds <- seq(0,n,length.out = CV+1)
Importances <- c(0.10,0.20,0.30,0.50,1,2,8,15)
FeatureCount <- c(0,0,0,0,0,0,0,0)
TreeAcc <- matrix(0, nrow = CV, ncol = length(Importances))
TreeSen <- matrix(0, nrow = CV, ncol = length(Importances))
TreeSpe <- matrix(0, nrow = CV, ncol = length(Importances))
TreePpv <- matrix(0, nrow = CV, ncol = length(Importances))
TreeNpv <- matrix(0, nrow = CV, ncol = length(Importances))

for (j in 1:length(Importances)){

  # Isolate subset with most important features
  s = ncol(CorData)-1
  RFSelected <- CorData
  for (i in s:1){
    if (Features[i] < Importances[j]){
      RFSelected <- RFSelected[,-c(i)]
    }
  }
  FeatureCount[j] <- ncol(RFSelected)

  # Shuffle rows
  rows <- sample(nrow(RFSelected))
  RFSelected <- RFSelected[rows,]

  for (i in 1:CV){
```

```
    # Split train and test sets by CV fold
    train <- RFSelected[ c(Folds[i]:Folds[i+1]),]
    test  <- RFSelected[-c(Folds[i]:Folds[i+1]),]

    # Train random forest
    rf <- randomForest(X1 ~ .,
                       data=train,
                       ntree = 50)

    # Predict on test set
    pred <- predict(rf, newdata=test)

    # Create confusion matrix
    results <- confusionMatrix(pred,test$X1)

    TreeAcc[i,j] <- results[3]$overall[1]
    TreeSen[i,j] <- results[4]$byClass[1]
    TreeSpe[i,j] <- results[4]$byClass[2]
    TreePpv[i,j] <- results[4]$byClass[3]
    TreeNpv[i,j] <- results[4]$byClass[4]
  }
}
```

The best feature importance cutoffs were 0.20 and 0.30, depending on the performance metric being evaluated as shown below with the best performing metrics bolded (Figure 4). Based of these results, we use a feature importance cutoff of 0.30 with a set of 939 features for further model tuning.

Importance Cutoff	Number of Features	Accuracy	Sensitivity	Specificity	Positive Predictivity	Negative Predictivity
0.10	1732	95.18 ± 0.51	95.82 ± 0.37	94.54 ± 1.15	94.62 ± 1.09	95.76 ± 0.35
0.20	1248	95.45 ± 0.30	**95.99 ± 0.55**	94.91 ± 0.77	94.97 ± 0.72	**95.95 ± 0.52**
0.30	939	**95.56 ± 0.28**	95.85 ± 0.69	**95.28 ± 0.82**	**95.32 ± 0.77**	95.83 ± 0.65
0.50	663	95.38 ± 0.38	95.86 ± 0.64	94.90 ± 1.03	96.97 ± 0.93	95.83 ± 0.60
1.00	376	95.41 ± 0.31	95.75 ± 0.49	95.06 ± 0.68	95.10 ± 0.64	95.72 ± 0.48
2.00	215	95.17 ± 0.42	95.24 ± 0.78	95.10 ± 0.82	95.12 ± 0.76	95.24 ± 0.73
8.00	99	95.21 ± 0.30	95.34 ± 0.79	95.08 ± 1.04	95.10 ± 0.98	95.33 ± 0.74
15.00	66	94.37 ± 0.21	95.48 ± 0.51	93.25 ± 0.83	93.41 ± 0.74	95.38 ± 0.47

```
FeatureResults <- data.frame(Importances)
FeatureResults$NumFea <- FeatureCount
FeatureResults$AccAvg <- colMeans(TreeAcc)
FeatureResults$SenAvg <- colSds(TreeSen)
FeatureResults$SpeAvg <- colSds(TreeSpe)
FeatureResults$PpvAvg <- colSds(TreePpv)
FeatureResults$NpvAvg <- colSds(TreeNpv)
FeatureResults$AccStd <- colSds(TreeAcc)
FeatureResults$SenStd <- colSds(TreeSen)
FeatureResults$SpeStd <- colSds(TreeSpe)
FeatureResults$PpvStd <- colSds(TreePpv)
FeatureResults$NpvStd <- colSds(TreeNpv)

FeatureResults
  Importances NumFea    AccAvg      SenAvg      SpeAvg      PpvAvg      NpvAvg
1         0.1   1724 0.9517801 0.003657793 0.011487255 0.010887959 0.003478059
2         0.2   1189 0.9545091 0.005485160 0.007721144 0.007160218 0.005243294
3         0.3    917 0.9556400 0.006980529 0.008247492 0.007696207 0.006528123
4         0.5    612 0.9538306 0.006426226 0.010294542 0.009284211 0.005987465
5         1.0    359 0.9540568 0.004913555 0.006830378 0.006417354 0.004842003
6         2.0    217 0.9517197 0.007791513 0.008236453 0.007627268 0.007267614
7         8.0     94 0.9520665 0.007924248 0.010414824 0.009800177 0.007461952
8        15.0     60 0.9436528 0.005090599 0.008327051 0.007430456 0.004680291
       AccStd      SenStd      SpeStd      PpvStd      NpvStd
1 0.005066056 0.003657793 0.011487255 0.010887959 0.003478059
2 0.003047564 0.005485160 0.007721144 0.007160218 0.005243294
3 0.002816931 0.006980529 0.008247492 0.007696207 0.006528123
4 0.003760995 0.006426226 0.010294542 0.009284211 0.005987465
5 0.003092436 0.004913555 0.006830378 0.006417354 0.004842003
6 0.004218995 0.007791513 0.008236453 0.007627268 0.007267614
7 0.003041975 0.007924248 0.010414824 0.009800177 0.007461952
8 0.002092047 0.005090599 0.008327051 0.007430456 0.004680291
```

Hyperparameter-Tuning of Classifier Model

The best performing random forest model had `mtry = 50` and `maxnodes = 99999`, with the latter indicating that there is essentially not limit on the maximum number of nodes. The results are provided in a table format for viewing convenience (Figure 5).

```
# Isolate subset with most important features
s = ncol(CorData)-1
RFSelected <- CorData
for (i in s:1){
  if (Features[i] < 0.3){
    RFSelected <- RFSelected[,-c(i)]
  }
}

# Shuffle rows
rows <- sample(nrow(RFSelected))
RFSelected <- RFSelected[rows,]

# Model Hyperparameter Tuning
CV <- 10
Folds <- seq(0,nrow(RFSelected),length.out = CV+1)
Tries <- c(10,30,50)
Nodes <- c(25,100,99999)

# Create dataframe to store results
Model <- data.frame(mtry = rep(Tries, each = length(Nodes)),
                    maxnodes = c(Nodes,Nodes,Nodes),
                    AccAvg = NA, AccStd = NA,
                    SenAvg = NA, SenStd = NA,
                    SpeAvg = NA, SpeStd = NA,
                    PpvAvg = NA, PpvStd = NA,
                    NpvAvg = NA, NpvStd = NA)

for (k in 1:length(Tries)){

  for (j in 1:length(Nodes)){

    TreeAcc = rep(0,CV)
    TreeSen = rep(0,CV)
    TreeSpe = rep(0,CV)
    TreePpv = rep(0,CV)
    TreeNpv = rep(0,CV)

    for (i in 1:CV){

      # Split train and test sets by CV fold
      train <- RFSelected[ c(Folds[i]:Folds[i+1]),]
      test  <- RFSelected[-c(Folds[i]:Folds[i+1]),]

      # Train random forest
      rf <- randomForest(X1 ~ .,
                         data=train,
                         ntree = 50,
                         mtry = Tries[k],
                         maxnodes = Nodes[j])

      # Predict on test set
      pred <- predict(rf, newdata=test)
```

```
      # Create confusion matrix
      results <- confusionMatrix(pred,test$X1)
      TreeAcc[i] <- results[3]$overall[1]
      TreeSen[i] <- results[4]$byClass[1]
      TreeSpe[i] <- results[4]$byClass[2]
      TreePpv[i] <- results[4]$byClass[3]
      TreeNpv[i] <- results[4]$byClass[4]
    }

    # Store CV results
    n <- (k-1)*3+j

    Model$AccAvg[n] <- mean(TreeAcc)
    Model$SenAvg[n] <- mean(TreeSen)
    Model$SpeAvg[n] <- mean(TreeSpe)
    Model$PpvAvg[n] <- mean(TreePpv)
    Model$NpvAvg[n] <- mean(TreeNpv)

    Model$AccStd[n] <- sd(TreeAcc)
    Model$SenStd[n] <- sd(TreeSen)
    Model$SpeStd[n] <- sd(TreeSpe)
    Model$PpvStd[n] <- sd(TreePpv)
    Model$NpvStd[n] <- sd(TreeNpv)
  }
}

# Display GridCV results
Model
```

	mtry	maxnodes	AccAvg	AccStd	SenAvg	SenStd	SpeAvg
1	10	25	0.9324650	0.005878636	0.9657867	0.002574879	0.8991453
2	10	100	0.9523228	0.004216537	0.9612968	0.003802405	0.9433644
3	10	99999	0.9523831	0.004073920	0.9610842	0.004352800	0.9436936
4	30	25	0.9420245	0.004716026	0.9636140	0.003939368	0.9204256
5	30	100	0.9549464	0.004552756	0.9590684	0.004650493	0.9508499
6	30	99999	0.9541774	0.003924201	0.9587330	0.003998609	0.9496402
7	50	25	0.9478748	0.003737295	0.9618960	0.004290717	0.9338510
8	50	100	0.9550369	0.003447429	0.9581320	0.005052605	0.9519610
9	50	99999	0.9551424	0.002911609	0.9575586	0.004327614	0.9527420

	SpeStd	PpvAvg	PpvStd	NpvAvg	NpvStd
1	0.012387582	0.9055903	0.010739780	0.9633287	0.002561436
2	0.008803241	0.9444398	0.008375460	0.9605761	0.003720074
3	0.007431287	0.9447143	0.007128005	0.9603833	0.004286037
4	0.011632284	0.9238832	0.010092413	0.9619839	0.003679659
5	0.009282435	0.9513272	0.008949922	0.9587170	0.004493770
6	0.008917969	0.9501743	0.008553697	0.9583478	0.003780152
7	0.009331613	0.9357710	0.008339329	0.9608053	0.004070516
8	0.008064533	0.9523296	0.007747514	0.9578740	0.004781878
9	0.008165163	0.9530529	0.007794314	0.9573530	0.004007285

Final Model Performance

Upon expanding the final model to 500 trees, we see that our final model has an accuracy of XX.XX%, sensitivity of XX.XX%, specificity of XX.XX%, positive predictivity of XX.XX%, and negative predictivity of XX.XX%.

```
# Model Hyperparameter Tuning
CV <- 10
Folds <- seq(0,nrow(RFSelected),length.out = CV+1)

# Initalize memory to story metrics
TreeAcc <- matrix(0, nrow = CV, ncol = 1)
TreeSen <- matrix(0, nrow = CV, ncol = 1)
TreeSpe <- matrix(0, nrow = CV, ncol = 1)
TreePpv <- matrix(0, nrow = CV, ncol = 1)
TreeNpv <- matrix(0, nrow = CV, ncol = 1)

for (i in 1:CV){

  # Split train and test sets by CV fold
  train <- RFSelected[ c(Folds[i]:Folds[i+1]),]
  test  <- RFSelected[-c(Folds[i]:Folds[i+1]),]

  # Train random forest
  rf <- randomForest(X1 ~ .,
                     data=train,
                     ntree = 500,
                     mtry = 30)

  # Predict on test set
  pred <- predict(rf, newdata=test)

  # Create confusion matrix
  results <- confusionMatrix(pred,test$X1)
  TreeAcc[i,1] <- results[3]$overall[1]
  TreeSen[i,1] <- results[4]$byClass[1]
  TreeSpe[i,1] <- results[4]$byClass[2]
  TreePpv[i,1] <- results[4]$byClass[3]
  TreeNpv[i,1] <- results[4]$byClass[4]
}

mean(TreeAcc)
[1] 0.9557606
mean(TreeSen)
[1] 0.9592185
mean(TreeSpe)
[1] 0.9523265
mean(TreePpv)
[1] 0.9527537
mean(TreeNpv)
[1] 0.958949
sd(TreeAcc)
[1] 0.003400278
sd(TreeSen)
[1] 0.005241484
sd(TreeSpe)
[1] 0.009506616
sd(TreePpv)
[1] 0.009079455
```

```
sd(TreeNpv)
[1] 0.004892455
```

Conclusions

Unlike many medical applications, where sensitivity is a critical measurement, the primary performance metric we are interested in is positive predictivity. For example, we do not care if we “miss” an DNA activator sequence that makes an NCB bright, as there are many such sequences and we only need **one** to work. In contrast, if we run a proposed activator sequence that through the model and it is classified as “bright”, we really want that DNA sequence to elicit a truly bright fluorescent response in the NCB, lest we have to develop and test another DNA strand. Our model has a positive predictivity of XX.XX%, which means we are nearly guaranteed to be able to design a DNA strand that is desirable, prior to testing it experimentally. In addition, unlike other application areas where an improvement in 1% accuracy can mean a direct 1% decline is patient deaths (any life is significant!), the precise accuracy of our model is not critical as long it generally performs well. For example, having a positive predictivity of 90% opposed to 95%, indicates that our design will fail once every ten times instead of twenty times. This is not a huge deal, compared to without **any** model, we may have to design several DNA sequences before one is ultimately successful in terms of desired brightness.

As such, we have developed a model that effectively allows us to design our NCB molecular sensors *in silico*, opposed to the traditional method of generating several expensive sequences, running tedious experiments, and essentially crossing our fingers hoping that the NCB turns out the way we want it to.