Statistical Analysis of Back Pain Causes and Symptoms

STAT40750

Hugh O’Brien

17204599

1.1 Abstract

This paper describes statistical analysis performed on data set pertaining to different kinds of back pain. Back pain can often be a difficult condition to properly diagnose as any diagnosis is usually formed based on anecdotal symptoms reported by the patient. This study attempts to improve the classifications of such back pain. Specifically that of Neuropathic and Nociceptive back pain. This paper describes that attempt to build a statistical machine learning model to predict cases of the two above forms of back pain. During analysis, multiple different algorithms were tested including logistic regression, decision trees and random forest classifiers with the later proving to be the most reliable classifier of the group. Using a random forest classifier, an average classification accuracy of around 95% was readily attainable. The model performed very well in generalization tests on unseen data also.

1.2 Introduction

The problem presented is a classic binary classification case with the two levels referring to Nueropathic and Nociceptive back pain. The data provided contains 380 samples with 31 explanatory variables. The target class included labels created from real world diagnosese from experts. The problem presented is to accurately distinguish between the two types of back pain, one being a symptom of more nefarious issues (Neuropathic) whilst the other is typically more benign in nature (Nociceptive). Unsurprisingly this is an important distinction to make and one that is not always very straightforward often due to the lack of concrete symptoms. Those attempting diagnose pain such as this must often rely on anecdotal symptoms provided by the patient. Due to the nature of the two types of pain, a misclassification could be potentially quite harmful to the patient as Neuropathic pain can be an indicator of damage to the nervous system or the threat of such damage. As such, catching damage such as this before it proliferates is very important. This is the issue

this study attempts to help resolve, that is what are the key indicators of such pain and can we use these vital indicators to make accuracy predictions for such pain?

1.3 Methods

A variety of machine learning algorithms and statistical evaluations techniques were used to provide some cursory answers to the above question. In particular the algorithms used were the following: Logistic regression, Decision trees, Bagging bases ensembles and Random Forest classifiers. The data was split into training, validation and test sets using an 80:10:10 split. This is opposed to the tradition 70:15:15 split, however due to the relatively smaller number of samples, a larger training set needed to be used to achieve an adequate convergence for the logistic regression.

To begin with all models were trained on the training data and an analysis of variable importance was conducted. Decision trees in particular are very susceptible to overfitting with a key cause of overfitting being overly high dimensional of the base data set. The original data set contains 31 dimensions excluding the target variable. The initial variable importance test is important for culling the variables that do not contribute to the predictive power of a potential model and will likely only supply noise for each model. Noise that might interfere with the generalization power of the chosen model.

Once the data set had been cleaning of frivolous dimensions, all models were trained on the new training data and then predictions were made on the validation set before getting accuracy scores for said validation set. A model was then selected depending on the performance on the validation set and subsequently tested on the final test set. The chosen model was then subject to a K-fold cross validation test. This K-fold test was performed 100 times and results were then averaged across all iterations in order to account for the variability inherit in sampling.

1.4 Results

1.4.1 Feature Importance

3 models were trained on the training set and the results were observed individually. The results were as follows:

Logistic Regression:

```
1] "PainLocationUni BK"          "PainLocationBack + Uni Leg" "SurityRating"
4] "Criterion4Present"           "Criterion8Present"          "Criterion9Present"
7] "Criterion20Present"
```

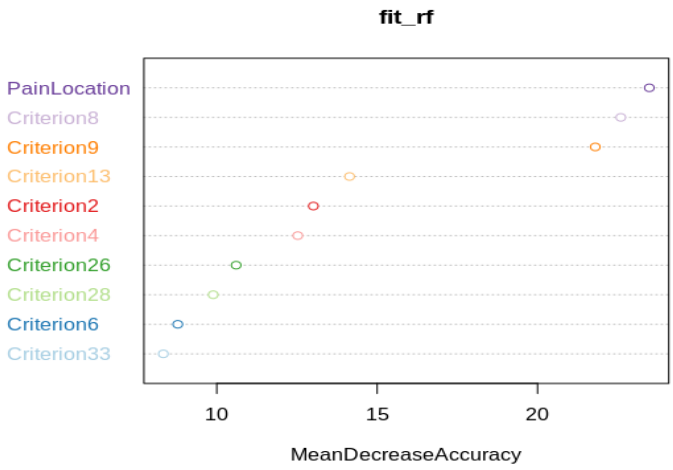
The above results indicate all variables significant at 90% confidence interval. For full regression results see Appendix A

Decision Tree:

```
Variable importance
  Criterion8  Criterion13  Criterion10  Criterion28 PainLocation  Criterion33   Criterion2
          26           15           12           12           10            9            8
  Criterion4   Criterion9           PF      SF36MCS
           3            2            1            1
```

Where each number indicates the sum decrease in node impurity after being converted to percentage (Naturally the higher the number the more important the variable

Random Forest:



Find Extra information on feature importance in Appendix C and D

1.4.2 Train, Test, Validation

Once the data had been stripped of weak variables, all models were trained on an 80% training set and then tested on a 10% validation set with the results being as follows (Note: Logisitic Regression was not tested as it performance was deemed to variable. Due to the size of the data set, the logisitic regression did not always converge and as such gave incomplete results)

	Random Forest	***Decision Tree***	***Bagging***	**K(3) NN**
Validation Score	0.9736842	0.9210526	0.9736842	0.9736842

The bagging and random forest and KNN outperformed the decision tree. Notice that there is no difference between the 3 models in terms of accuracy on the validation set. This is due to the relatively small size of the validation set. The Random Forest was selected for further analysis. The Random Forest subsequently received an accuracy score of 0.9736842 on the test set

1.4.3 K-Fold Cross Validation

A 5 folds cross validation was performed on the Random Forest and KNN 1000 times with the results be averaged across all 1000 iterations. The results were as follows:

Metric	Random Forest	KNN
Accuracy	0.9687895	0.9499526
Precision	0.99884	0.9502083
Recall	0.94186	0.95491
F1	0.9695135	0.9525533

A graph summarizing the results of the above test seen in Appendix E

1.4.4 Receiver Operating Characteristic

The Random Forest returned an AUC score of 0.95 and the full ROC curve can be found in Appendix F.

1.5 Discussion

Overall, the Random Forest achieved exceptional predictive power and indicated strong generalization. The most important variables appeared to be the location of the pain experience, Criterion 8 and Criterion 9 with C8 referring to “Localised to area of injury, dysfunction.” and C9 being “Referred in dermatomal, cutaneous distribution.”. Removing weak variables did not appear to hinder the predictive power of the model at all and removing unnecessary variables when dealing with decision trees like Random Forests is usually good practice as these types of models are often very susceptible to overfitting with one of the many causes of overfitting being too many explanatory variables. The KNN also performed quite well on the validation test, however the size of the validation set should be kept in mind when comparing the

strength of the models. Regardless, some might opt for the KNN as it is generally less computationally intensive. It is important to note however that KNN's are classed as 'lazy learners' with a large portion of the computation occurring at prediction time opposed to the 'earner learners' such as the Random Forest. With a dataset of this size, computation time and efficiency is likely a none issue however.

As previously mentioned the Random Forest proved quite robust when exposed to unseen data and this can be seen in its admirable performance during the cross validation. Other auxiliary metrics were recorded during this K-Fold test being, precision, recall and F1. All these scores were very higher with the recall being slightly lower than that of the precision. In medical cases, recall is often preferred over precision as it is often more desirable to incur a false positive where the cost of such an error may very well only be monetary opposed to a false negative which can lead to more grave implications down the line if an ailment isn't properly diagnosed. In this case, misdiagnosis could prove problematic as Nociceptive pain is generally considered quite benign whilst Neuropathic pain may lead to greater neurological damage if left untreated. In this specific case, this trade off between recall and precision should be considered relatively inconsequential as both score as very high and the difference between the two is minimal. However, if one were so inclined, a probability threshold for such models could be lowered from the traditional 0.5 to ensure a more conservative approach to false positives and negatives.

Conclusion

From the above analysis is can be seen that location of pain as well as C8 and C9 are all strong indicators when predicting types of back pain. 5 different types of models were tested with Random Forest appearing to perform the best with strong predictive scores and admirable generalization to unseen data. Consequently, this problem is one that is quite handled quite aptly by the Random Forest algorithm, however some allowances could be make in regards to probability thresholds so as to preserve as

high a recall as possible which is often a very desirable trait for machine learning algorithms specifically when dealing with medical cases.

Appendix

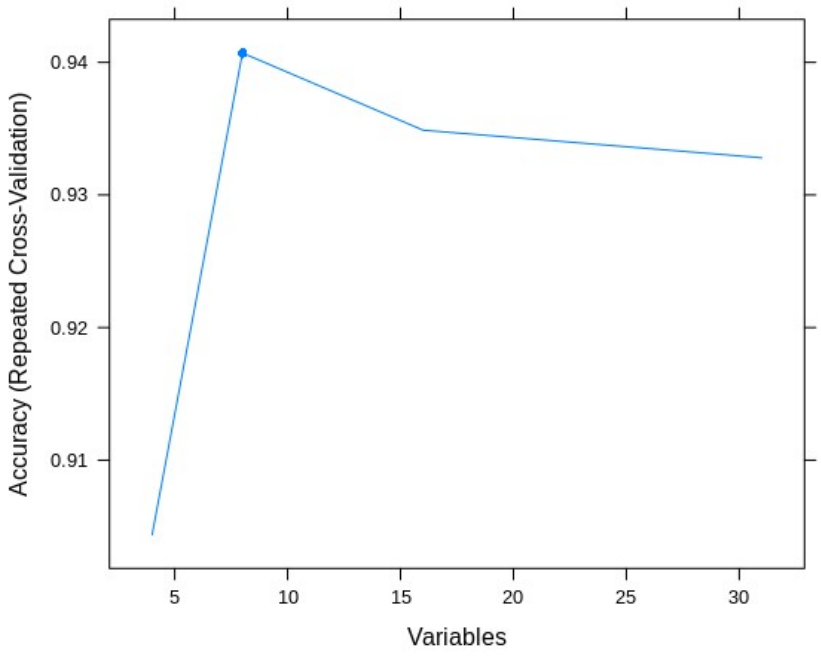
A

```
Coefficients:
                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                  -2.481e+01  1.982e+01  -1.252  0.21056
Age                           2.775e-02  8.527e-02   0.325  0.74487
GenderMale                   -1.623e+00  1.773e+00  -0.915  0.35994
DurationCurrent4-6 weeks      1.729e+00  3.697e+00   0.468  0.64006
DurationCurrent7-12 weeks    -5.975e-01  3.688e+00  -0.162  0.87130
DurationCurrent4-6 months     1.830e-01  3.817e+00   0.048  0.96177
DurationCurrent7-12 months    1.262e+00  5.121e+00   0.246  0.80540
DurationCurrent> 1 year      -2.314e+00  3.183e+00  -0.727  0.46729
PainLocationBack+Thigh        3.616e+00  2.349e+00   1.540  0.12361
PainLocationUni BK            7.091e+00  2.836e+00   2.500  0.01242 *
PainLocationBilat BK          2.486e+01  1.075e+04   0.002  0.99816
PainLocationBack + Uni Leg    6.219e+00  3.748e+00   1.659  0.09703 .
PainLocationBack + Bi-lat leg 1.668e+01  2.794e+03   0.006  0.99524
SurityRating                  1.694e+00  8.695e-01   1.949  0.05134 .
RMDQ                         -7.904e-02  2.499e-01  -0.316  0.75181
vNRS                          5.941e-01  4.382e-01   1.356  0.17518
SF36PCS                      -1.043e-01  3.819e-01  -0.273  0.78467
SF36MCS                      -2.081e-01  1.674e-01  -1.243  0.21394
PF                            2.904e-01  2.373e-01   1.223  0.22116
BP                           -1.562e-01  2.803e-01  -0.557  0.57729
GH                           -2.137e-03  1.590e-01  -0.013  0.98928
VT                           -1.753e-02  1.377e-01  -0.127  0.89866
MH                            2.203e-01  1.613e-01   1.366  0.17200
HADSAnx                      -2.290e-01  2.669e-01  -0.858  0.39084
HADSDep                       4.253e-01  4.276e-01   0.995  0.31981
Criterion2Present             4.053e+00  4.498e+00   0.901  0.36761
Criterion4Present             1.324e+01  6.875e+00   1.926  0.05405 .
Criterion6Present             4.667e+00  2.985e+00   1.564  0.11792
Criterion7Present             2.375e+00  2.125e+00   1.118  0.26377
Criterion8Present            -8.996e+00  4.211e+00  -2.137  0.03264 *
Criterion9Present             9.120e+00  3.284e+00   2.777  0.00548 **
Criterion10Present            2.235e+00  6.001e+00   0.372  0.70958
Criterion13Present            6.126e+00  4.857e+00   1.261  0.20720
Criterion19Present            1.192e-01  1.273e+00   0.094  0.92543
Criterion20Present           -4.575e+00  2.513e+00  -1.821  0.06864 .
Criterion26Present            1.681e+00  2.737e+00   0.614  0.53907
Criterion28Present           -5.277e-01  4.219e+00  -0.125  0.90047
Criterion32Present            2.460e+00  2.737e+00   0.899  0.36875
Criterion33Present            6.469e+00  5.052e+00   1.280  0.20040
Criterion36Present           -3.431e+00  2.602e+00  -1.319  0.18730
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

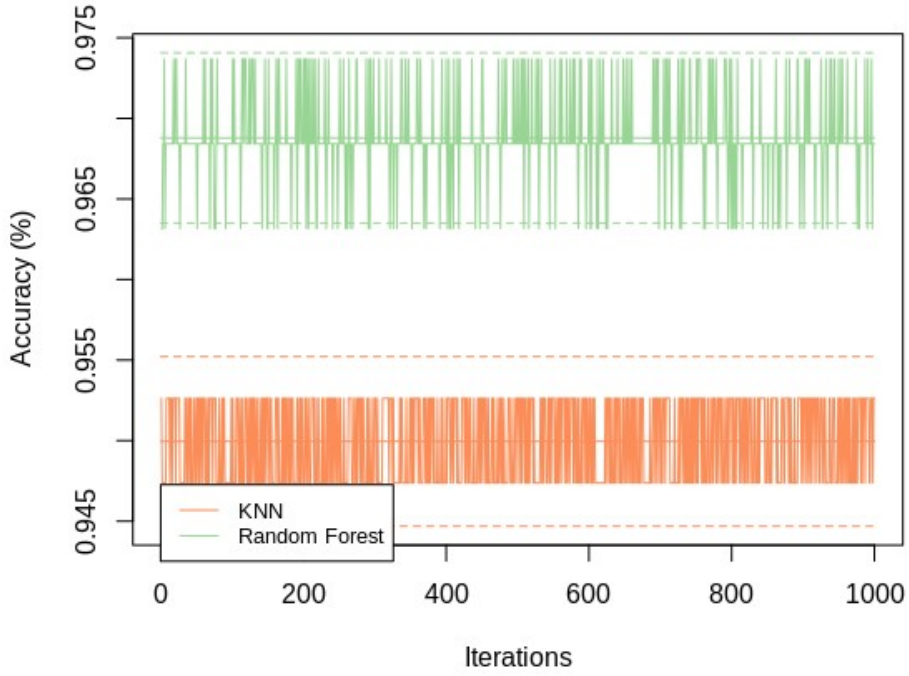
C

```
Recursive feature selection

Outer resampling method: Cross-Validated (10 fold, repeated 5 times)

Resampling performance over subset size:

 Variables Accuracy  Kappa AccuracySD KappaSD Selected
         4   0.9058 0.8103    0.04843 0.09723
         8   0.9432 0.8854    0.03980 0.08001        *
        16   0.9379 0.8751    0.04029 0.08060
        31   0.9294 0.8581    0.05084 0.10169

The top 5 variables (out of 8):
   PainLocation, Criterion9, Criterion8, Criterion2, Criterion13
```

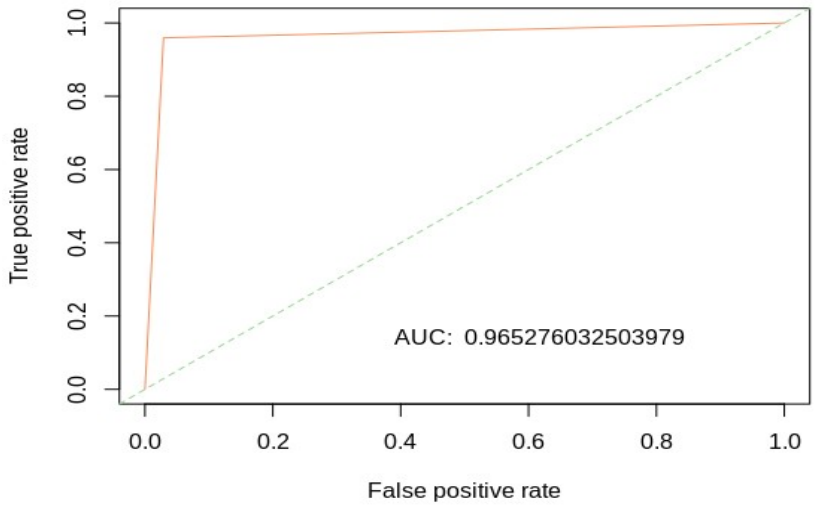
D



E


Accuracy (%)
0.945
0.955
0.965
0.975
0
200
400
600
800
1000
Iterations
KNN
Random Forest

F


True positive rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC: 0.965276032503979
False positive rate

References

1. Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2019). dplyr: A Grammar of Data Manipulation. R package version 0.8.0.1. https://CRAN.R-project.org/package=dplyr

2. Klaus K. Holst and Esben Budtz-Joergensen (2013). Linear Latent Variable Models: The lava-package. Computational Statistics, 28 (4), pp. 1385-1452. doi: 10.1007/s00180-012-0344-y.

3. Klaus K. Holst and Esben Budtz-Joergensen (2019). A two-stage estimation procedure for non-linear structural equation models. Biostatistics. doi: 10.1007/s00180-012-0344-y.

4. Gavin L. Simpson (2019). permute: Functions for Generating Restricted Permutations of Data. R package version 0.9-5. https://CRAN.R-project.org/package=permute

5. Max Kuhn. Contributions from Jed Wing, Steve Weston, Andre Williams, Chris Keefer, Allan Engelhardt, Tony Cooper, Zachary Mayer, Brenton Kenkel, the R Core Team, Michael Benesty, Reynald Lescarbeau, Andrew Ziem, Luca Scrucca, Yuan Tang, Can Candan and Tyler Hunt. (2019). caret: Classification and Regression Training. R package version 6.0-82. https://CRAN.R-project.org/package=caret

CODE:

```
rm(list=ls())
library(rpart)
library(RColorBrewer)
library(pROC)
library(randomForest)
library(permute)
library(adabag)
library(dplyr)
library(lava)
library(caret)
library(ROCR)

load("/home/hugh/stat_ML/Assignment6/
data_project_backpain.RData")

set.seed(100)

class(dat)
# shuffle dataset
dat = dat[sample(nrow(dat)),]

dat$PainDiagnosis = as.factor(dat$PainDiagnosis)
```

```
get_train_split = function(df){
    # split data set into train, validate, test sets (80:10:10)
    # note: with a 70:15:15 split, there was not enough data for the log
    # regression to converge so this split was used instead.
    train_index = sample(1:nrow(df), size = nrow(df)*0.8)
    df_train = df[train_index,]
    test_index = setdiff(1:nrow(df), train_index)
    df_test = df[test_index,]
    validate_index = sample(1:nrow(df_test), size=nrow(df_test)*0.5)
    df_validate = df_test[validate_index,]
    test_index = setdiff(1:nrow(df_test), validate_index)
    df_test = df_test[test_index,]

    return(list("train"=train_index, "test"=test_index,
"validate"=validate_index))
}

split = get_train_split(dat)

# fitting models
# note: due to the lack of data, the log regression doesn't always
converge properly
# so results may vary.
fit_log <- glm(PainDiagnosis ~ ., data = dat, family = "binomial",
subset=split$train)
fit_tree <- rpart(PainDiagnosis ~ ., data = dat, subset = split$train)
```

```
fit_rf <- randomForest(PainDiagnosis ~ ., data=dat,
subset=split$train, importance=TRUE)
fit_bag <- bagging(PainDiagnosis ~ ., data=dat, subset=split$train)
fit_knn <- knn3(PainDiagnosis ~ ., data = dat, subset=split$train)

summary(fit_log)
coefs = summary(fit_log)$coefficients
coefs
rownames(coefs)[which(coefs[,4] < 0.1)]
# important variables: PainLocation, C4, C6, C8, C9
summary(fit_tree)
# important variables: C8, C10, C13, C26, C28, C33, PainLocation
pal = brewer.pal(10, "Paired")
varImpPlot(fit_rf, type=1, n.var = 10, col=pal)
# important variables: PainLocation, C8, C9, C13, C26, C2

ctrl = rfeControl(functions=rfFuncs, method="repeatedcv",
repeats=5)
lmprofile = rfe(x=dat[,2:32], y=dat[,1], rfeControl=ctrl)
lmprofile
plot(lmprofile, type="l")
# According to the above 3 algorithms, the most important variables
are:
# PainLocation, C8, C10, C13, C9 among other less important
variables
```

```
# Decision trees are highly susceptible to overfitting so for this reason it
# is wise to limit the input dimensions to hopefully aid in future generalization

lean_df = select(dat, PainDiagnosis, PainLocation, Criterion8, Criterion10, Criterion13,
      Criterion26, Criterion9, Criterion4, Criterion6, Criterion33,
      Criterion2, Criterion28)



########################################
#      Train, Validate, Test       #
########################################

split = get_train_split(lean_df)

fit_log <- glm(PainDiagnosis ~ ., data = lean_df, family = "binomial", subset=split$train)
fit_tree <- rpart(PainDiagnosis ~ ., data = lean_df, subset = split$train)
fit_rf <- randomForest(PainDiagnosis ~ ., data=lean_df, subset=split$train, importance=TRUE)
fit_bag <- bagging(PainDiagnosis ~ ., data=lean_df, subset=split$train)
fit_knn <- knn3(PainDiagnosis ~ ., data = lean_df, subset=split$train)
```

```
model_fit = list("log"=fit_log, "tree"=fit_tree, "randomForest"=fit_rf,
"bag"=fit_bag, "knn"=fit_knn)

model_predict = list()
for(name in names(model_fit)){
  predict = predict(model_fit[name], newdata =
lean_df[split$validate,])
  model_predict = c(model_predict, c(name = predict))
}
# bagging confusion matrix
tab = model_predict$name.bag$confusion
tab
bagging_ac = sum(revdiag(tab))/sum(tab)
bagging_ac

tree_predict_class = c()
for(i in 1:nrow(model_predict$name.tree)){
  if(model_predict$name.tree[i, 1] > model_predict$name.tree[i, 2]){
    tree_predict_class = c(tree_predict_class, "Noiceptive")
  }
  else{
    tree_predict_class = c(tree_predict_class, "Neuropathic")
  }
}
tree_predict_class
```

```
tab <- table(lean_df$PainDiagnosis[split$validate], tree_predict_class)
tab
tree_ac = sum(revdiag(tab))/sum(tab)
tree_ac

model_predict$name.randomForest
tab <- table(lean_df$PainDiagnosis[split$validate],
model_predict$name.randomForest)
tab
rf_ac = sum(diag(tab))/sum(tab)
rf_ac

model_predict$name.knn
knn_predict_class = c()
for(i in 1:nrow(model_predict$name.knn)){
  if(model_predict$name.knn[i, 1] > model_predict$name.knn[i, 2]){
    knn_predict_class = c(knn_predict_class, "Noiceptive")
  }
  else{
    knn_predict_class = c(knn_predict_class, "Neuropathic")
  }
}

tab <- table(lean_df$PainDiagnosis[split$validate], knn_predict_class)
tab
tree_ac = sum(revdiag(tab))/sum(tab)
```

```
tree_ac

# random forest regression had the highest accuracy along with
# the knn on the validation
# set with a perfect classifcation score therefor I will use and fully
# evaluate this model from now on

prediction = predict(fit_rf, newdata=lean_df[split$test,])
tab <- table(lean_df$PainDiagnosis[split$test], prediction)
tab
rf_ac = sum(diag(tab))/sum(tab)
rf_ac

###################################
#       K-Fold Analysis       #
###################################

# test was initially performed with 1000 iterations
# results may be different from the report
iterations = 10
av_accuracy = 0
iter_ac_rf = numeric(iterations)
iter_recall_rf = numeric(iterations)
iter_precision_rf = numeric(iterations)
iter_ac_knn = numeric(iterations)
iter_recall_knn = numeric(iterations)
```

```
iter_precision_knn = numeric(iterations)
k = 5
n = nrow(lean_df)
folds = rep(1:k, ceiling(n/k))
folds = sample(folds)
folds = folds[1:n]

get_metrics <- function(tab){
  tab <- table(lean_df$PainDiagnosis[split$test], prediction)
  precision = tab[1]/(tab[1]+tab[3])
  recall = tab[1]/(tab[1]+tab[2])
  ac = sum(diag(tab))/sum(tab)
  output = list("ac"=ac, "recall"=recall, "precision"=precision)

}

for(j in 1:iterations){
  total_accuracy_rf = 0
  total_recall_rf = 0
  total_precision_rf = 0
  total_accuracy_knn = 0
  total_recall_knn = 0
  total_precision_knn = 0

  for(i in 1:k){
    train = which(folds != i)
```

```
    test = setdiff(1:n, train)

    fit_knn = knn3(PainDiagnosis ~ ., data = lean_df, subset = train)
    fit_rf = randomForest(PainDiagnosis ~ ., data = lean_df, subset =
train)

    # get metrics for random forest
    prediction = predict(fit_rf, newdata=lean_df[split$test,])
    tab <- table(lean_df$PainDiagnosis[split$test], prediction)
    rf_metrics = get_metrics(tab)

    #get metrics for knn
    prediction = predict(fit_knn, type="class",
newdata=lean_df[split$test,])
    tab <- table(lean_df$PainDiagnosis[split$test], prediction)
    knn_metrics = get_metrics(tab)

    # append metrics to respective running totals
    total_accuracy_rf = total_accuracy_rf + rf_metrics$ac
    total_recall_rf = total_recall_rf + rf_metrics$recall
    total_precision_rf = total_precision_rf + rf_metrics$precision

    total_accuracy_knn = total_accuracy_knn + knn_metrics$ac
    total_recall_knn = total_recall_knn + knn_metrics$recall
    total_precision_knn = total_precision_knn + knn_metrics$precision
  }
```

```
  iter_ac_rf[j] = total_accuracy_rf/k
  iter_recall_rf[j] = total_recall_rf/k
  iter_precision_rf[j] = total_precision_rf/k

  iter_ac_knn[j] = total_accuracy_knn/k
  iter_recall_knn[j] = total_recall_knn/k
  iter_precision_knn[j] = total_precision_knn/k
}

mean(iter_ac_rf)
mean(iter_recall_rf)
mean(iter_precision_rf)
2*(mean(iter_precision_rf)*mean(iter_recall_rf))/
(mean(iter_precision_rf)+mean(iter_recall_rf))

mean(iter_ac_knn)
mean(iter_recall_knn)
mean(iter_precision_knn)
2*(mean(iter_precision_knn)*mean(iter_recall_knn))/
(mean(iter_precision_knn)+mean(iter_recall_knn))

print_table = function(ac1, ac2){

  #calculate averages across all iterations
  av_ac1 = mean(ac1)
```

```
av_ac2 = mean(ac2)

max_acc = max(max(ac1), max(ac2))
min_acc = min(min(ac1), min(ac2))

sd_ac1 = sd(ac1)
sd_ac2 = sd(ac2)

#get upper and lower bounds for both models (2 standard deviations)
ac1_ubound = av_ac1 + 2*sd_ac1
ac1_lbound = av_ac1 - 2*sd_ac1

ac2_ubound = av_ac2 + 2*sd_ac2
ac2_lbound = av_ac2 - 2*sd_ac2

max_sd = max(max(ac1_ubound), max(ac2_ubound))
min_sd = min(min(ac1_lbound), min(ac2_lbound))

pal = brewer.pal(3, "Spectral")

plot(ac1, type = "l", col=pal[3], ylim = c(min_sd,max_sd),
    xlab = "Iterations", ylab = "Accuracy (%)")

lines(ac2, type = "l", col = pal[1])

lines(c(0, iterations), c(av_ac1, av_ac1), type="l", col=pal[3])
```

```
  lines(c(0, iterations), c(av_ac2, av_ac2), type="l", col=pal[1])
  lines(c(0, iterations), c(ac1_ubound, ac1_ubound), type="l",
col=pal[3], lty=2)
  lines(c(0, iterations), c(ac1_lbound, ac1_lbound), type="l",
col=pal[3], lty=2)
  lines(c(0, iterations), c(ac2_ubound, ac2_ubound), type="l",
col=pal[1], lty=2)
  lines(c(0, iterations), c(ac2_lbound, ac2_lbound), type="l",
col=pal[1], lty=2)

  par(xpd=TRUE)
  legend(x=0, min_acc-0.0001, legend = c("KNN", "Random Forest"),
col = c(pal[1], pal[3]), lty=1:1, cex=0.8)

  return(0)
}

print_table(iter_ac_rf, iter_ac_knn)

# overall, very high precision and slightly lower recall
# higher recall likely more desirable for cases such as this where
# a False negative can lead to serious health issues down the line
# in a sense it is better to predict positive more often than usual
# so as to avoid these 'expensive' misclassifications opposed
# to misclassifying a False Positive, in which case the only cost is
# the cost of treatment and not someone's long term health
```

```
####################################
#       Plotting ROC Curve         #
####################################
split = get_train_split(lean_df)
fit_rf <- randomForest(PainDiagnosis ~ ., data=lean_df,
importance=TRUE)
pred = predict(fit_rf, newdata=lean_df)
pal = brewer.pal(3, "Spectral")
plot.roc(roc_curve, col=pal, print.auc=TRUE, identity.col=pal[3])

predObj <- prediction(as.numeric(pred), dat$PainDiagnosis,
label.ordering=c("Nociceptive","Neuropathic"))

perf <- performance(predObj, "tpr", "fpr")
auc <- performance(predObj, "auc")@y.values
auc = paste("AUC: ", toString(auc))
plot(perf, col=pal, identity.col=pal[3])
abline(0,1,lty=2,col=pal[3])
text(0.616742, 0.1429984, labels=auc)
```