Module6L3

Neha Parulekar

March 13, 2016

# Additional packages needed

* If necessary install the followings packages.

install.packages("ggplot2");  
install.packages("e1071");  
install.packages("kernlab");

require(ggplot2)

## Loading required package: ggplot2

require(e1071)

## Loading required package: e1071

## Warning: package 'e1071' was built under R version 3.2.4

require(kernlab)

## Loading required package: kernlab

##   
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':  
##   
## alpha

* Go to the [UC Irvine Machine Learning Repository](https://archive.ics.uci.edu/ml/) and find a dataset for supervised classification. Every student MUST use a different dataset so you MUST get approved for which you can going to use. This can be the same dataset you used for the unsupervised clustering as long as the data has some labeled data.

## **About the dataset**

The dataset used is taken from UCI Machine learning Repository. The name of the dataset is **Yeast Dataset**. This dataset predicts the cellular localization sites of proteins. There are 8 attributes and 1484 instances. There are8 predictive and 1 outcome variables. The outcome class is localization sites.

Attribute Information. 1. Sequence Name: Accession number for the SWISS-PROT database 2. mcg: McGeoch's method for signal sequence recognition. 3. gvh: von Heijne's method for signal sequence recognition. 4. alm: Score of the ALOM membrane spanning region prediction program. 5. mit: Score of discriminant analysis of the amino acid content of the N-terminal region (20 residues long) of mitochondrial and non-mitochondrial proteins. 6. erl: Presence of "HDEL" substring (thought to act as a signal for retention in the endoplasmic reticulum lumen). Binary attribute. 7. pox: Peroxisomal targeting signal in the C-terminus. 8. vac: Score of discriminant analysis of the amino acid content of vacuolar and extracellular proteins. 9. nuc: Score of discriminant analysis of nuclear localization signals of nuclear and non-nuclear proteins.

# Loading the data   
# Here I take two columns in the yeast data and two class variable  
data\_url <-"https://archive.ics.uci.edu/ml/machine-learning-databases/yeast/yeast.data"  
YeastData <- read.csv(url(data\_url), header = FALSE, sep = "")  
head(YeastData)

## V1 V2 V3 V4 V5 V6 V7 V8 V9 V10  
## 1 ADT1\_YEAST 0.58 0.61 0.47 0.13 0.5 0.0 0.48 0.22 MIT  
## 2 ADT2\_YEAST 0.43 0.67 0.48 0.27 0.5 0.0 0.53 0.22 MIT  
## 3 ADT3\_YEAST 0.64 0.62 0.49 0.15 0.5 0.0 0.53 0.22 MIT  
## 4 AAR2\_YEAST 0.58 0.44 0.57 0.13 0.5 0.0 0.54 0.22 NUC  
## 5 AATM\_YEAST 0.42 0.44 0.48 0.54 0.5 0.0 0.48 0.22 MIT  
## 6 AATC\_YEAST 0.51 0.40 0.56 0.17 0.5 0.5 0.49 0.22 CYT

YeastData$V1 <- NULL  
YeastData$V2 <- NULL  
YeastData$V3 <- NULL  
YeastData$V6 <- NULL  
YeastData$V7 <- NULL  
YeastData$V8 <- NULL  
YeastData$V9 <- NULL  
head(YeastData)

## V4 V5 V10  
## 1 0.47 0.13 MIT  
## 2 0.48 0.27 MIT  
## 3 0.49 0.15 MIT  
## 4 0.57 0.13 NUC  
## 5 0.48 0.54 MIT  
## 6 0.56 0.17 CYT

summary(YeastData)

## V4 V5 V10   
## Min. :0.21 Min. :0.0000 CYT :463   
## 1st Qu.:0.46 1st Qu.:0.1700 NUC :429   
## Median :0.51 Median :0.2200 MIT :244   
## Mean :0.50 Mean :0.2612 ME3 :163   
## 3rd Qu.:0.55 3rd Qu.:0.3200 ME2 : 51   
## Max. :1.00 Max. :1.0000 ME1 : 44   
## (Other): 90

length(YeastData)

## [1] 3

nrow(YeastData)

## [1] 1484

# making a new data set which has only two class variables  
  
NewYeastData <- YeastData[which(YeastData$V10 == "CYT" | YeastData$V10 == "ME3"),]  
head(NewYeastData)

## V4 V5 V10  
## 6 0.56 0.17 CYT  
## 10 0.60 0.15 CYT  
## 13 0.57 0.35 CYT  
## 16 0.52 0.11 CYT  
## 17 0.50 0.11 CYT  
## 21 0.50 0.16 CYT

nrow(NewYeastData)

## [1] 626

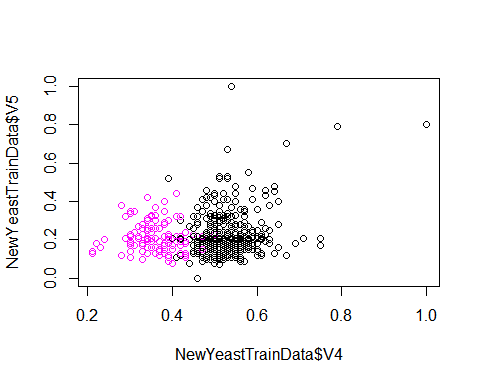
NewYeastTrainData <- NewYeastData[1:525,]  
nrow(NewYeastTrainData)

## [1] 525

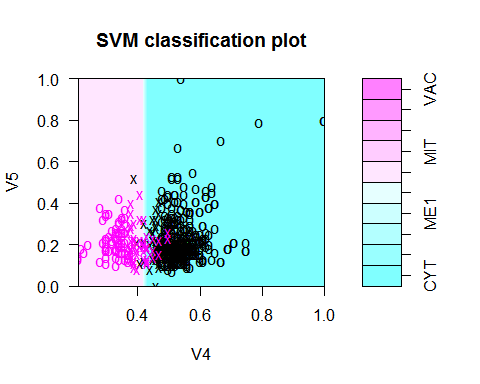
NewYeastTestData <- NewYeastData[526:626,]  
nrow(NewYeastTestData)

## [1] 101

# Plot   
  
plot(NewYeastTrainData$V4, NewYeastTrainData$V5, col = as.integer(NewYeastTrainData$V10))



#----------------------Training a model on the data ----  
  
svm.fit <- svm(V10 ~ V4 + V5, data = NewYeastTrainData, kernel = "linear", cost = 10, scale = FALSE)  
plot(svm.fit, NewYeastTrainData, V5 ~ V4, slice = list(V5 = 1, V4 = 2))



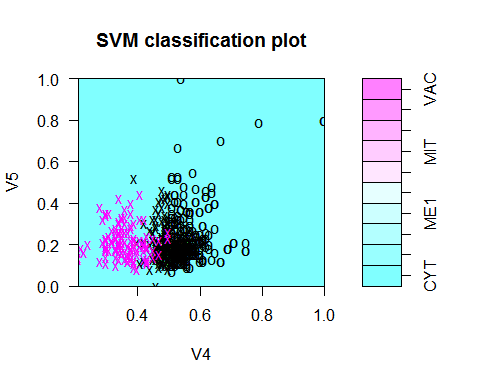
# List my Support Vectors  
svm.fit$index

## [1] 8 25 26 72 87 117 118 123 127 128 146 152 153 156 162 172 181  
## [18] 183 207 228 233 239 241 243 249 254 256 258 269 296 304 325 355 356  
## [35] 368 369 376 378 417 419 434 449 459 486 487 492 493 494 496 15 18  
## [52] 38 41 55 67 68 70 113 115 119 120 121 124 134 137 185 209 210  
## [69] 213 215 216 219 220 226 230 240 261 273 284 286 292 312 324 403 431  
## [86] 439 448 450 455 470 471 473 475 478 479 484 489

summary(svm.fit)

##   
## Call:  
## svm(formula = V10 ~ V4 + V5, data = NewYeastTrainData, kernel = "linear",   
## cost = 10, scale = FALSE)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 10   
## gamma: 0.5   
##   
## Number of Support Vectors: 97  
##   
## ( 49 48 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CYT ERL EXC ME1 ME2 ME3 MIT NUC POX VAC

# Let's chage Cost  
svm.fit1 <- svm(V10 ~ V4 + V5, data = NewYeastTrainData, kernel = "linear", cost = 0.1, scale = FALSE)  
plot(svm.fit1, NewYeastTrainData, V5 ~ V4, slice = list(V5 = 1, V4 = 2))



#Cross-Validation -Find Best Model for prediction   
tune.out <- tune(svm, V10 ~ V4 + V5, data = NewYeastTrainData, kernel = "linear", ranges = list(cost = c(0.001,0.01,0.1,1,5,10,100)))  
  
bestmodel = tune.out$best.model  
summary(bestmodel)

##   
## Call:  
## best.tune(method = svm, train.x = V10 ~ V4 + V5, data = NewYeastTrainData,   
## ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)),   
## kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 100   
## gamma: 0.5   
##   
## Number of Support Vectors: 46  
##   
## ( 23 23 )  
##   
##   
## Number of Classes: 2   
##   
## Levels:   
## CYT ERL EXC ME1 ME2 ME3 MIT NUC POX VAC

##-------------------- Predicting the Function ---  
  
  
#Predict testdata(i.e. testdf ) based on bestmodel  
  
ypred = predict(bestmodel, NewYeastTestData)  
ypred

## 1284 1285 1286 1287 1288 1289 1292 1293 1294 1295 1296 1297 1298 1301 1302   
## CYT CYT CYT CYT ME3 ME3 CYT CYT CYT CYT CYT CYT CYT CYT CYT   
## 1303 1304 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1321   
## CYT CYT CYT CYT CYT CYT CYT ME3 CYT CYT CYT CYT CYT CYT ME3   
## 1325 1327 1328 1329 1330 1334 1346 1347 1348 1349 1350 1351 1353 1354 1361   
## CYT CYT CYT CYT CYT CYT CYT CYT CYT CYT CYT ME3 CYT CYT ME3   
## 1362 1366 1368 1369 1371 1372 1376 1378 1379 1381 1382 1384 1386 1391 1393   
## ME3 ME3 CYT CYT CYT CYT ME3 ME3 CYT ME3 ME3 ME3 ME3 ME3 ME3   
## 1395 1397 1399 1401 1402 1403 1406 1407 1408 1412 1414 1415 1416 1417 1418   
## ME3 CYT ME3 ME3 ME3 ME3 ME3 ME3 ME3 ME3 ME3 CYT CYT CYT ME3   
## 1420 1422 1425 1427 1428 1429 1430 1431 1432 1434 1439 1441 1446 1447 1454   
## ME3 ME3 ME3 ME3 ME3 CYT CYT ME3 ME3 ME3 ME3 ME3 ME3 CYT ME3   
## 1458 1460 1467 1468 1474 1475 1476 1477 1478 1479 1484   
## CYT CYT ME3 CYT CYT CYT CYT CYT CYT CYT CYT   
## Levels: CYT ERL EXC ME1 ME2 ME3 MIT NUC POX VAC

table(pred = ypred,truth = NewYeastTestData$V10)

## truth  
## pred CYT ERL EXC ME1 ME2 ME3 MIT NUC POX VAC  
## CYT 55 0 0 0 0 6 0 0 0 0  
## ERL 0 0 0 0 0 0 0 0 0 0  
## EXC 0 0 0 0 0 0 0 0 0 0  
## ME1 0 0 0 0 0 0 0 0 0 0  
## ME2 0 0 0 0 0 0 0 0 0 0  
## ME3 1 0 0 0 0 39 0 0 0 0  
## MIT 0 0 0 0 0 0 0 0 0 0  
## NUC 0 0 0 0 0 0 0 0 0 0  
## POX 0 0 0 0 0 0 0 0 0 0  
## VAC 0 0 0 0 0 0 0 0 0 0

# look only at agreement vs. non-agreement  
# construct a vector of TRUE/FALSE indicating correct/incorrect predictions  
agreement <- ypred == NewYeastTestData$V10  
table(agreement)

## agreement  
## FALSE TRUE   
## 7 94

prop.table(table(agreement))

## agreement  
## FALSE TRUE   
## 0.06930693 0.93069307

##-------------- Improving model performance -----  
  
classifier\_rbf <- ksvm(V10 ~ V4 + V5, data = NewYeastTestData, kernel = "rbfdot")  
predictions\_rbf <- predict(classifier\_rbf, NewYeastTestData)  
predictions\_rbf

## [1] CYT CYT CYT CYT ERL ERL CYT CYT CYT CYT CYT CYT CYT CYT CYT CYT CYT  
## [18] CYT CYT CYT CYT CYT ERL CYT CYT CYT CYT CYT ERL ERL CYT CYT CYT CYT  
## [35] ERL CYT CYT CYT CYT CYT CYT ERL CYT CYT ERL ERL ERL CYT CYT CYT CYT  
## [52] ERL ERL CYT ERL ERL ERL ERL ERL ERL ERL CYT ERL ERL ERL ERL ERL ERL  
## [69] ERL ERL ERL CYT CYT ERL ERL ERL ERL ERL ERL ERL ERL CYT ERL ERL ERL  
## [86] ERL ERL ERL CYT ERL CYT CYT ERL CYT CYT CYT CYT CYT CYT CYT CYT  
## Levels: CYT ERL EXC ME1 ME2 ME3 MIT NUC POX VAC

agreement\_rbf <- predictions\_rbf == NewYeastTestData$V10  
table(agreement\_rbf)

## agreement\_rbf  
## FALSE TRUE   
## 47 54

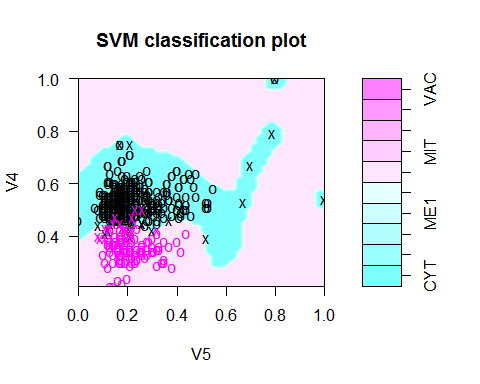
prop.table(table(agreement\_rbf))

## agreement\_rbf  
## FALSE TRUE   
## 0.4653465 0.5346535

####-------------------------- RADIAL KERNAL ( play with gamma and cost) ------------------  
  
svmfit1 <- svm(V10 ~ V4 + V5, data = NewYeastTrainData, kernel = "radial", gamma = 1, cost = 100000)  
svmfit1

##   
## Call:  
## svm(formula = V10 ~ V4 + V5, data = NewYeastTrainData, kernel = "radial",   
## gamma = 1, cost = 1e+05)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1e+05   
## gamma: 1   
##   
## Number of Support Vectors: 63

plot(svmfit1, NewYeastTrainData)



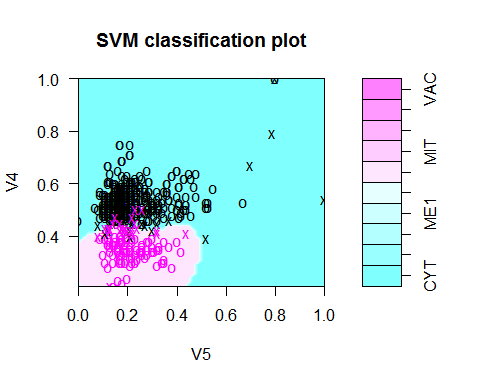
#------------- Cross Validation to set best choice of gamma and cost  
  
tune.out <- tune(svm, V10 ~ V4 + V5, data = NewYeastTrainData, kernel = "radial", ranges = list(cost = c(0.001,10,100,1000)), gamma = c(0.5,1,2,3,4))  
summary(tune.out)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 10  
##   
## - best performance: 0.03240203   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.22456459 0.06496653  
## 2 1e+01 0.03240203 0.03126508  
## 3 1e+02 0.03240203 0.02709297  
## 4 1e+03 0.03432511 0.03100439

bestmodel1<-tune.out$best.model  
bestmodel1

##   
## Call:  
## best.tune(method = svm, train.x = V10 ~ V4 + V5, data = NewYeastTrainData,   
## ranges = list(cost = c(0.001, 10, 100, 1000)), kernel = "radial",   
## gamma = c(0.5, 1, 2, 3, 4))  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 10   
## gamma: 0.5 1 2 3 4   
##   
## Number of Support Vectors: 54

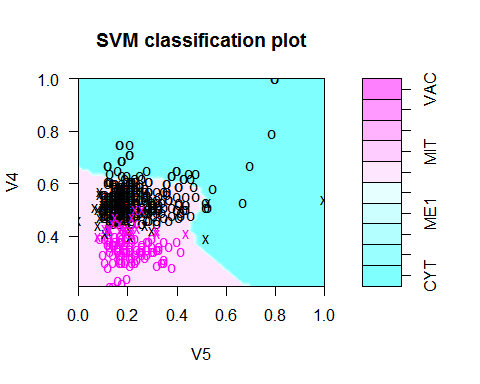
plot(bestmodel1, NewYeastTrainData)



### -------------- Polynomial KERNAL  
  
svmfitPoly<- svm(V10 ~ V4 + V5,data = NewYeastTrainData, kernel="polynomial",gamma=1, cost=100000)  
svmfitPoly

##   
## Call:  
## svm(formula = V10 ~ V4 + V5, data = NewYeastTrainData, kernel = "polynomial",   
## gamma = 1, cost = 1e+05)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 1e+05   
## degree: 3   
## gamma: 1   
## coef.0: 0   
##   
## Number of Support Vectors: 47

plot(svmfitPoly, NewYeastTrainData)



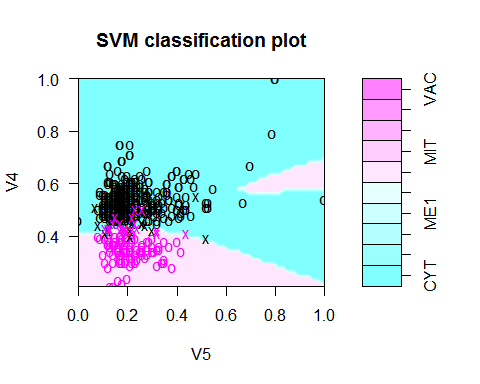
#------------------- Cross validation to set best choice of gamma and cost - POLYNOMIAL  
  
tune.out.poly<- tune(svm, V10 ~ V4 + V5,data = NewYeastTrainData , kernel="polynomial", ranges = list(cost=c(0.001,10,100,1000)), gamma=c(0.5,1,2,3,4))   
summary(tune.out.poly)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 100  
##   
## - best performance: 0.03991292   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.21509434 0.06717426  
## 2 1e+01 0.04190856 0.01746708  
## 3 1e+02 0.03991292 0.02066410  
## 4 1e+03 0.03991292 0.02066410

bestmodelPoly = tune.out.poly$best.model  
bestmodelPoly

##   
## Call:  
## best.tune(method = svm, train.x = V10 ~ V4 + V5, data = NewYeastTrainData,   
## ranges = list(cost = c(0.001, 10, 100, 1000)), kernel = "polynomial",   
## gamma = c(0.5, 1, 2, 3, 4))  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: polynomial   
## cost: 100   
## degree: 3   
## gamma: 0.5 1 2 3 4   
## coef.0: 0   
##   
## Number of Support Vectors: 44

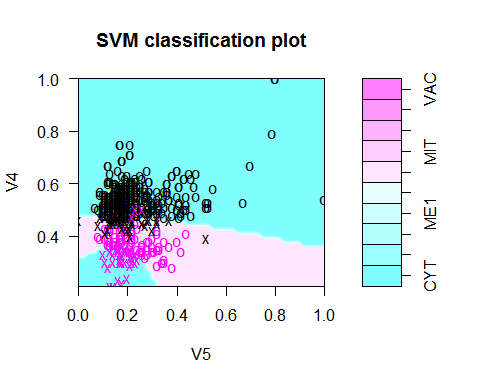
plot(bestmodelPoly, NewYeastTrainData)



###------------------------- SIGMOID KERNEL  
svmfitSig<- svm(V10 ~ V4 + V5,data = NewYeastTrainData, kernel="sigmoid",gamma=1, cost=100000)  
svmfitSig

##   
## Call:  
## svm(formula = V10 ~ V4 + V5, data = NewYeastTrainData, kernel = "sigmoid",   
## gamma = 1, cost = 1e+05)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: sigmoid   
## cost: 1e+05   
## gamma: 1   
## coef.0: 0   
##   
## Number of Support Vectors: 84

plot(svmfitSig, NewYeastTrainData)



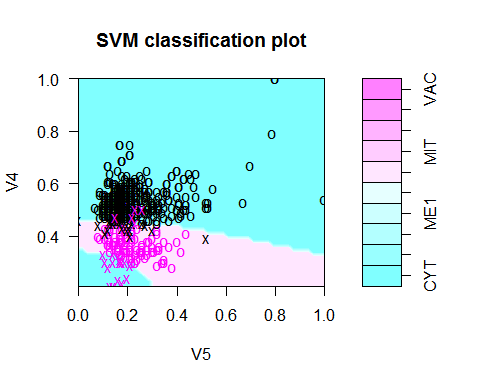
#---------------------------Cross validation to set best choice of gamma and cost - SIGMOID  
tune.out.sig<- tune(svm, V10 ~ V4 + V5,data = NewYeastTrainData, kernel="sigmoid", ranges = list(cost=c(0.001,10,100,1000)), gamma=c(0.5,1,2,3,4))   
summary(tune.out.sig)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 10  
##   
## - best performance: 0.08193033   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 1e-03 0.22463716 0.03212921  
## 2 1e+01 0.08193033 0.03494412  
## 3 1e+02 0.09346880 0.03791923  
## 4 1e+03 0.08962264 0.04264519

bestmodelSig = tune.out.sig$best.model  
bestmodelSig

##   
## Call:  
## best.tune(method = svm, train.x = V10 ~ V4 + V5, data = NewYeastTrainData,   
## ranges = list(cost = c(0.001, 10, 100, 1000)), kernel = "sigmoid",   
## gamma = c(0.5, 1, 2, 3, 4))  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: sigmoid   
## cost: 10   
## gamma: 0.5 1 2 3 4   
## coef.0: 0   
##   
## Number of Support Vectors: 42

plot(bestmodelSig, NewYeastTrainData)



* Classify your data using Support Vector Machines. You can use any method/package for SVMs. Answer the following questions:

\***How well does the classifier perform?** The classifier looks like a good one from the table and the probability.Except for the nine times of 101, test set has been classified properly. The probability of truth is 91%.

\***Try different kernels. How do they effect its performce?** Here I have tried linear, radial, polynomial and sigmoid kernels. After playing around with the gamma and cost values and using a function to find the best model to set the best choice for gamma values and costs and plotting the graphs, it looks like radial is the best classifier and Sigmoid looks good too, however linear and polynomial do not seem to be good. Tune methods finds out the best choice of gamma and cost. It makes is easier to classify classes from one another.

\***What might improve its performce?** The performance can be improved using the best kernel and gamma and cost. In this dataset I think radial is best kernel to improve performance. It classifies CYT from ME3 localisation site for the yeast protein.