2018

Application of SuperLearner Algorithm on Prostate Cancer Data

Ebo Essilfie-Amoah
eamoah88@hotmail.com

APPLICATION OF SUPERLEARNER ALGORITHM ON PROSTATE CANCER DATA

by

Ebo Essilfie-Amoah

A Major Research Paper
Submitted to the Faculty of Graduate Studies
through the Department of Mathematics and Statistics
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

APPLICATION OF SUPERLEARNER ALGORITHM ON PROSTATE CANCER DATA

by

Ebo Essilfie-Amoah

APPROVED BY:

M. Hlynka
Department of Mathematics and Statistics

A. Hussein, Advisor
Department of Mathematics and Statistics

October 15, 2018

Author's Declaration of Originality

I hereby declare that I am the sole author of this major Paper and that no part of this major paper has been published or submitted for publication.

I certify that to the best of my knowledge, my major paper does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my major paper, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted materials that surpass the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission of the copyright owner(s) to include such materials in my major paper and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my major paper, including any final revisions, as approved by committee and the Graduate Studies Office, and that this major paper has not been submitted for a higher degree to any other University or Institution. I authorize University of Windsor, Ontario Canada to lend this Major Paper to other institutions or individuals for the purpose of academic work.

Abstract

The objective of this major paper is to apply an ensemble method known as the SuperLearner algorithm to find the best ten gene expressions that can predict a high PSA (¿7) versus low PSA in prostate Cancer patients. We try to formulate techniques such the penalized logistic regressions, random forest and cross-validation in a way that is consistent with the formulation of the SupeLearner algorithm. To discover a patch of ten genes that can predict well the PDA level, we sample random patches of 10 genes from a pool of almost 47,000 genes and apply the superlearner algorithm to compute a cross-validated AUC. Consequently we choose groups of ten genes that have AUC exceeding 0.65. This exercise shows that many un-classified genes are correlated with high PSA.

Acknowledgments

I thank the Almighty God for His protection and security which I can not even purchased.

I am especially indebted to Dr. Hussein, my suprevisor, my teacher and mentor. He has taught me more than I could ever give him credit for here. He has shown me, by his example, what a good person should be.

I would like to thank Dr. Myron Hlynka for being my department reader. He taught me the basic knowledge of stochastic processes and regression analysis, which helped me build a solid foundation for further studies in Statistics.

I am grateful to all faculty and staff members,as well as the graduate students in the Department of Mathematics and Statistics at the University of Windsor, with whom I have had the pleasure to work during this program.Each of them has provided me extensive personal and professional guidance and taught me a great deal about life in general. Nobody has been more important to me in the pursuit of this career than the members of my family. I would like to thank my parents, whose love and guidance are with me in whatever I pursue. They are my ultimate role models. Most importantly, I wish to thank my loving and supportive wife, Deines, and my two wonderful children, Albert and Junior, who provide unending inspiration.

Contents

List of Figures

List of Tables

Chapter 1

The SuperLearner Algorithm

1.1 Background

One of the most common supervised statistical learning methodologies deals with classification of multi-category variables based on a given set of independent variables (known also as covariates or features). Historically, the logistic regression approach is a standard statistical technique addressing such classification problems. However, logistic regression models tend to over-fit the learning sample when the number p of features, or input variables, largely exceeds the number n of samples. This is referred to as the small n large p setting, commonly found in biomedical problems. An example of such is when one desires to correlate between disease incidence and the expressions of a large number of genes based on microarray data. Apart from the logistic regression, there is also a number of methodologies in supervised statistical learning such as regression-tree-based approaches and neural networks[2]

Often one would fit a few different types of models and may obtain, as a result, a few different answers (where an answer here is an estimated probability of belonging to one of the classes at hand). It is therefore quite difficult to make a subjective choice of which model to pick. This has led to the idea of model ensembles. Ensemble methods are approaches for pooling models together to produce one that could probably beat all of the individual model predictions. There are many ways to ensemble models and among those are the classical random forests, Bayesian model averaging and the more recent stacking approach called the superlearner algorithm [1,3,4].

Ensemble methods have been in existence for long time and therefore it is very difficult to trace when they began. The idea of deploying multiple models has been in use since 1990s. Breiman and Stehman conducted research in 1980s, where they found out that predictions made by the best single classifier are inferior to predictions made by the combination of a set of classifiers. It was concluded that predictions made by more classifiers most often give better and accurate result than made by a single classifier. In general, the phrase “Ensemble Methods” refers to building a large number of somewhat independent predictive models and then combining them by voting or averaging to yield very high performance [7]. Common types of ensemble learning are Bagging, Boosting and Random Forest all of which have the objective of improving performance in different ways. Also both Bagging and Random Forests were developed to overcome variance and bias issues.

The objective of this paper is to use a prostate cancer data set to illustrate the use of the SuperLearner algorithm in classification of a binary variable.

1.2 The SuperLearner and other relevant methods

Before proceeding to any of the learning algorithms that are used on the prostate data in the applications chapter, we first introduce several important general frameworks. For instance, many learning algorithms share the approach of cross-validation. Cross-validation as well as the learning algorithms that use them, can be viewed as formulas or procedures or even models that take the data as their input and output estimators of desired quantities. There are often two issues to deal with in building an estimator: the procedure of building the estimator itself and that of assessing its performance. These two procedures often share a fundamental tool known as loss function and its expectation which is known as risk function. Often, in building estimators (formulas or algorithms or models (M)), we maximize or minimize some sort of gain or loss function. Let us take an example of a binary classification problem. Let y_i be the outcomes and x_i be a vector of covariates (features) for the ith observation, so that the data set is $D_n = \{(y_i, x_i), i = 1, ..., n\}$. We will use the short hand y and x to denote the vectors of responses and that of features, respectively. One of the common models is the logistic regression model whose aim is to predict probability of success $p = P(y = 1|x)$ for a new subject given its covariates. The algorithm takes simply the D_n and maximizes a gain function of the form

$$L(y, \Psi(x, \beta)) = \sum_{i=1}^{N} \left[y_i(\beta_0 + \beta^T x_i) - log(1 + e^{(\beta^T x_i)}) \right].$$

That is, it obtains an estimate of a vector of parameters $\hat{\beta} = argmax_{\beta} L(y, \Psi(x, \beta))$. Notice here that the function $\Psi(.)$ is a function of the features x and possibly some unknown parameters. The function can also be nonparametric such as mean regressions using splines. The next step of the algorithm is to produce the predicted value for y given its features which is $\hat{p} = \hat{E}[y|x] = \frac{1}{1+e^{(-\hat{\beta}x)}}$ or even to give directly a predicted value of the response for a given set of features, $\hat{y} = 1$ if $\hat{p} > a$ and $\hat{y} = 0$ otherwise, where a is a probability threshold. Therefore, we have used a loss function to obtain a target parameter, which could be y for a new subject (hence, prediction) or p (probability of success for a new subject). Now the cross-validation operation can kick in as a method to assess the performance of the algorithm. That is, to answer the question of how good is $\hat{y}$ or $\hat{p}$. For such assessment to be unbiased, one should not use D_n which was used to build the model, M. So for that purpose, and if we are lucky, we will have a test data set D_m of size m that was not used in building the model M. Therefore, we will be able to estimate some sort of loss or gain measure aimed at measuring the performance of our model M. Let us take the example of the logistic regression when the target parameter is p and so a natural loss is the quadratic loss function $L(y, \hat{p}(x)) = (\hat{y} - \hat{p})^2$. Thus, the risk or the expected loss in using the model M to predict p for a new subject is $R = E[L(y, p(x))]$ which can be estimated by averaging over the test data (assuming the test data are independent realizations). That is, $\hat{R} = \sum_{i=1}^{m} 1/mL(y_i, \hat{p}(x_i))$. Another possible loss function is the missclassification error $L(y, \hat{p}(x)) = I(y \neq \hat{y})$ where $\hat{y}$ is decided by thresholding the $\hat{p}(x)$ as above. Also here, the risk can be estimated from the new testing data by averaging this function over the training observations $\sum_{i=1}^{m} \frac{1}{m} I(y_i \neq \hat{y}_i)$. Yet another gain function is the area under the Receiver Operating Characteristic curve (ROC), measured by AUC("area under the curve"). Higher values of the AUC correspond to better performance of the model in classifying subjects correctly to the binary classes.

The next concept is that of cross-validation. There could arise at least two scenarios in which cross-validation is required. One is when building the model itself requires the knowledge of a nuisance parameter (such as tuning parameter in the penalized regression models like lasso) or when one does not have test data to compute the loss function for assessing the performance of the model built on the training data. In such cases, one can still use cross-validation to estimate quantities of interest such as expectations of loss functions and or tuning parameters. Below is a generic $V-fold$ cross-validation procedure based on a loss function $L(y, \Psi(x))$.

Algorithm 1: Generic $v - fold$ cross-validation

```
step1: Divide $D_n$ randomly into $D_1, ..., D_V$, assuming that every partition has $n/V$, an integer
       sample size.
step2: For $v = 1, ..., V$,
       Fit the given algorithm, $M$, using $D_n - D_v$, i.e., all the data but the vth partition and
       call $M_{-v}$ the estimated model obtained by using the $D_{-v}$ data set.
       Compute the desired risk by averaging the loss function over the data partition $D_v$.
       That is
           $\hat{R}_v = \sum_{(y_i,x_i)\in D_v} \frac{1}{n/V} L(y_i, \hat{\Psi}_{M_{-v}}(x_i)).$
       Here we are using the notation $\hat{\Psi}_{M_{-v}}$ to indicate the target quantity (or parameter )
       estimated by using the model $M_{-v}$.
step3: Compute the final cross-validated loss as
           $\hat{R} = \sum_{v=1}^{V} \frac{1}{V} \hat{R}_v$
```

If a cross validation procedure sets $V = n$, then it is called leave-one-out (LOO). The LOO has been already in use in the survey literature under the name Jack-Knife, as one of the methods to obtain variances of finite population estimators. The cross-validation technique is not limited to just computing estimated versions of a loss function, but it can be used to generate cross-validated predicted values of the response itself, which is a key to many ensemble methods such as the superlearner algorithm.

In the next few sections we briefly describe some of the algorithms that we will use along with the superlearner algorithm.

1.3 Logistic regression with Lasso penalty

For more in depth treatment of the logistic regression with lasso penalty and its use as a classification method one can refer to [2]. We report a simple lasso logistic regression returning the predicted probabilities for a given fixed tuning parameter and one which uses v-fold cross-validation to choose optimal value for the tuning parameter based on AUC or misclassification error.

Algorith 2 (The logistic regression algorithm):

Step1: Given a data set D_n and a tuning parameter, α, compute

$$\hat{\beta} = argmax_{\beta} \sum_{i=1}^{N} \Big[y_i(\beta_0 + \beta^T x_i) - log(1 + e^{(\beta^T x_i)}) \Big] + \alpha \sum_{j=1}^{p} |\beta_j|.$$

step2: Return the model as the function acting on a vector of features, $x = (x_1, ..., x_p)$ and returning a vector of estimated probabilities

$$\hat{\Psi}(x) = \hat{p} = \hat{p}(x) \frac{1}{1 + e^{-\hat{\beta} x}}$$

Algorith 3 ($V - fold$ cross-validated logistic regression with lasso penalty and using AUC and missclassification error to choose the tuning parameter):

Step1: *Divide the data D_n into $D_1, ..., D_V$ disjoint and exhaustive subsets with n/V observations in each. Make a grid of tuning parameters $\alpha > 0$.*

step2: *For each α,*

do:

for each $v = 1, ..., V$:

pass the pair of objects α and $D_{-v} = D_n - D_v$ to **Algorithm 2** *and obtain $\hat{\Psi}(x)$.*

Compute the predicted probabilities from the data D_v as

$$\hat{\Psi}(x_i) = \hat{p} = \hat{p}(x_i) = \frac{1}{1 + e^{-\hat{\beta} x_i}}$$

For a sequence of cutoff points $t \in (0, 1)$ compute $\hat{y}_i = 1$ if $\hat{p}_i > t$ and $\hat{y}_i = 0$ otherwise for all i such that $y_i \in D_v$.

Compute

$$Sensitivity(t) = F(t) = \sum_{i: y_i \in D_v} I(y_i = 1, \hat{y}_i = 1)/(n/V)$$

and

$$1 - Specificity(t) = 1 - G(t) = 1 - \sum_{i: y_i \in D_v} I(y_i = 0, \hat{y}_i = 0)/(n/V)$$

.

Compute the area under the curve defined by the pairs $(1 - G(t), F(t))$ as the $AUC(\alpha, v)$.

```
        Compute the misclassification error as $mis(\alpha, v) = \sum_{i:y_i \in D_v} I(y_i \neq \hat{y}_i)/n$.
    end for:
    Compute the average AUC and the average misclassification error over $v = 1, ..., V$.
end do:
Choose the optimal $\alpha_{CV}$ as the one that leads to the model with the highest AUC or the least misclassification error.

step3: Return the model with the best $\alpha_{CV}$ as
```

$$\hat{\Psi}(x) = \hat{p} = \hat{p}(x)\frac{1}{1+e^{(-\hat{\beta}x)}}$$

with coefficients computed from algorithm 2 by using the α_{CV} and the entire data D_n.

1.3.1 Tree-based Methods

Here we limit ourself to the case of binary classification trees. The idea in such trees is to use binary splits based on some or all of the features. Explained in words, one would start by minimizing a loss function for all possible pairs of a feature and a cutoff point (also known as split point), (x_s, c) for $s = 1, ..., p$ and finding the pair that, for example, minimizes a loss function computed on the two subsets that result from the split of the data by the pair. For instance, we split the data D_n into two subsets $D_1 = \{(y_i, x_i) : x_{is} \leq c\}$ and $D_2 = D_1'$. Each of these subsets of data live in a node of the tree and therefore, the two nodes are daughters of whatever node they descended from. Once two nodes are obtained, a loss function is calculated on the data in each and the total loss is computed as $L(y, \Psi(x); D_n) = L_1(y, \Psi(x); D_1) + L_2(y, \Psi(x); D_2)$. Finally, the pair of (x_s, c) that minimizes the above loss is chosen as the splitting pair and the same procedure is repeated for each node until a prespecified stopping criteria is met. Such criteria could be reaching a prespecified maximum number of nodes or a minimum number of observations per node. At the end of this construction, the sample space R^p of the vector $x = (x_1, ..., x_p)$ will be partitioned into regions $N_1, ..., N_K$ corresponding to the K terminal nodes in the final tree.

There are many loss functions that can be used to decide the splits. For instance, in regression trees where the y is a continuous variable, a loss function such as the squared or absolute loss can be an appropriate measure to use, while if y is a binary or class variable, then measures such AUC or missclassification error can be maximized or minimized, respectively. Generally, in classification trees, loss functions are usually based on node impurity statistics. These are statistics that measure the heterogeneity in nodes by computing observed class membership for the data in the node concerned. For instance, a node is pure if all the y_i in that node belong to one class and it has maximum impurity when equal proportions of the y_i belong to each of the classes in consideration within that node.

An example of node impurity measure is the Gini index defined as

$$G_m = \sum_{k=1}^{K} \hat{P}_{mk}(1 - \hat{P}_{mk})$$

for the mth node in a given tree. This index measures total variance across the k classes and such interpretation is evident for the case of binary response variable (i.e., when $k = 2$). In fact, when $k = 2$, then

$$G_m = \hat{p}_{m0}(1 - \hat{p}_{m0}) + \hat{p}_{m1}(1 - \hat{p}_{m1} = 2\hat{p}_{m1}(1 - \hat{p}_{m1})$$

which is exactly twice the variance of the Bernoulli variable, y estimated from the data in the mth node. Thus, the loss used to build the tree is basically

$$L(y, \hat{\Psi}(x) = 2\hat{p}(1 - \hat{p})$$

where the $\Psi(.)$ here is the model which is simply the classification tree itself. To fix the idea and see how, for example, the Gini index is in fact a loss function based on the tree, suppose we have a tree with K terminal nodes and let $\Psi(.)$ be a vector of indicator functions,

$$\Psi(x) = (\Psi_1(x), ..., \Psi_K(x)) = (I(x \in N_1), ..., I(x \in N_K))$$

where N_k is the partition of the sample space of x corresponding to the kth terminal node of the tree.

Now clearly, the Gini index of the kth node for the binary case can be redefined as $L(y, \Psi_k(x)) = 2\hat{p}_k(1 - \hat{p}_k)$ where

$$\hat{p}_k = \sum_{i=1}^{n} \frac{y_i \Psi_k(x_i)}{\sum_{i=1}^{n} \Psi_k(x_i)}.$$

Therefore, the total loss for the entire tree could be computed as the sum of the losses over the terminal nodes of the tree.

An alternative to the Gini index is the cross-entropy, given by

$$D_m = -\sum_{k=1}^{K} \hat{P}_{mk} log \hat{P}_{mk}.$$

A tree can be pruned (penalized) by exploring all possible subtrees and choosing the one that minimizes a penalized loss. Any of the loss functions above can be penalized by a term $\alpha|\Psi|$ where α is a tuning parameter (a penalty parameter) and $|\Psi|$ is the size of the tree (i.e., number of terminal nodes). Notice here that we denote the tree itself by Ψ, because the function $\Psi(.)$ that we have defined above essentially represents the tree model.

Next, we will quote the classification tree algorithm as well as the random forest algorithm, before proceeding to the superlearner. Here we will use notational approach of Steingrimsson et al (2018), JASA

Algorithm 4 (Binary classification tree):

For a given data set D_n, a loss function $L(y, \Psi(x))$, and a stopping criteria, define as the current node the node that consists of all the observations in the data and

do:

a. *In the current node, identify all possible pairs (x_s, c) of a feature and a split point such that the data in the current node, is split into two parts by $x_s \leq c$ and $x_s > c$ so that each part forms a daughter node.*

b. Select the pair (x_s, c) of covariate and a split point that minimizes the given loss function computed by using the data in the parent node and divide this parent node into two mutually exclusive daughter nodes (subsets of data).

c. Check if the stopping criteria has been met for all of the current terminal nodes in the tree; if met, then return the tree Ψ and exit. Otherwise, set the node that is not meeting the criteria as the current parent node and repeat (b)-(c)

End Do:

Algorithm 5 (Random forest [James et al (2013) An Introduction to Statistical learning]):

Given a data set D_n, a loss function, two integers m, B, and a stopping criteria, do the following:

For $b = 1, ..., B$,

Do:

a. *Select a bootstrap sample of size n from the data*

b. *Grow a random forest tree Ψ_b based on the bootstrap sample by applying* **Algorithm 4** *in which step (b) is done only for m randomly selected variables among the p covariates.*

c. *Return the tree Ψ_b.*

end do:

Compute the target quantity (such as predicted probabilities of the event of interest or the estimated loss function) by averaging it over the tree $\Psi_b; b = 1, ..., B$.

The algorithm above is a typical example of an ensemble, as we are using a large number of trees and averaging out their predictions. The next topic is related to this idea and describes a recently proposed ensemble algorithm known as the superlearner.

1.3.2 The SuperLearner Algorithm

Stacked generalization is an ensemble method that allows researchers to combine several different prediction algorithms into one [8]. Since its introduction in the early 1990s, the method has evolved several times into its current form known as the SuperLearner.

In stacking, multiple layers of machine learning models are placed one over another where each of the models passes their predictions to the model in the layer above it and the top layer model takes decisions based on the outputs of the models in layers below it [8].

In the early 1990s, Wolpert developed a way to combine several machine learning algorithms with the goal of increasing predictive accuracy. He termed the approach as stacked generalizations, which later became known as stacking[8]. Later on, Breiman demonstrated how stacking can be used to improve the predictive accuracy in a regression context,and showed that imposing certain constraints on the higher-level model improved predictive performance[1].

More recently, van der Laan and colleagues proved that stacking possesses certain ideal theoretical properties and defined the superlearner algorithm. In particular, under reasonable constraints, Super Learner is guaranteed to perform asymptotically as well as the best performing algorithm included in the candidate set of algorithms. [7].

The Super Learner combines predictions from several methods by using V-fold cross-validation and minimizing a user-specified loss function, such as prediction error, negative-log-likelihood, or rank loss[6].

The algorithm is quite simple and it generally operates as follows. Given the covariates of a new observation, x, a target parameter to be predicted, $\Psi(x)$, and a library of algorithms with K members each designed to produce $\hat{\Psi}(x)$, compute the superlearner prediction as a linear combination of the individual predictions, $\hat{\Psi}_1(x), ..., \hat{\Psi}_K(x)$ as follows:

$$\hat{\Psi}_{SL}(x) = \sum_{l=1}^{K} \hat{\alpha}\hat{\Psi}_l(x)$$

where $\hat{\alpha}$ are computed in an optimal way using V-fold cross-validation.

Let us assume that we are dealing with a case of predicting the class probabilities for a binary classification situation. That is, what we are predicting is the probability $p_i = P(y_i = 1|x_i)$ so that our target parameter is $\Psi(x_i) = \hat{p}_i$. The computation of the optimal α in the superlearner can be described for a binary classification situation as follows.

Algorithm 6 (V-fold CV for computing the superlearner coefficients):

Given training data D_n, a library of K algorithms, $\Psi_1(.), ..., \Psi_K(.)$ that input x and output a prediction of a target parameter, $\hat{\Psi}_l(x)$, do the following:

a. *Partition D_n into V subsets, $D_1, ..., D_V$ with equal number of observations n/V.*

b. *For each $v = 1, ..., V$, let $D_{-v} = D_n$*

   D_v be the entire data save the vth subset.

   For $l = 1, ..., K$ build the lth algorithm using the data D_{-v} and use such an algorithm on the data in the subset D_v to obtain predictions $\{\hat{\Psi}_l(x_i)\}$ for $x_i \in D_v$. Store these K predictions for each of the n/V observations in the D_v data in a small $n/V \times K$ matrix.

c. *Stack the V matrices into a big one, Z, with all the $n \times K$ predictions. Consider the data (y, z) as new data set with K independent variables and n observations. Note that each algorithm represents a variable.*

d. *Use constrained ordinary least squares (COLS) to obtain $\hat{\alpha}_l$ for $l = 1, ..., K$ based on this pseudo data of (y_i, z_i) for $i = 1, ..., n$ where the $z_i = (z_{i1}, ..., z_{iK})$ is a vector of cross-validated predictions from the K algorithms trained on data excluding the D_v that contains (y_i, x_i). That is,*

$$\hat{\alpha} = argmax_\alpha \sum_{i=1}^{n} \left(y_i - \sum_{l=1}^{K} \alpha_l z_{il} \right)^2$$

subject to: $\alpha_l \geq 0$ *and* $\sum \alpha_l = 1$.

Chapter 2

Illustrating the use of the superlearner on PSA data

2.1 Description of the prostate cancer data sets

Prostate cancer is the most common malignancy (other than skin cancer) diagnosed in men[5]. Prostate cancer is a disease defined by the abnormal growth of cells. These abnormal cells can proliferate in an uncontrolled way and, if left untreated, form tumors which may spread to other parts of the body. Prostate cancer has the potential to grow and spread quickly, but for most men, it is a relatively slow growing disease[5]. According to Ministry of Health And Long Term Care in Canada,Of every 100 men, about 10 will be diagnosed with prostate cancer during their lifetime,and 3 of the 100 will die from the disease[5].

A PSA value of > 4.0 ug/L has often been defined in the literature as abnormal and is frequently used as a cut-point as indicated by the Ministry of Health And Long Term Care. However, a man's PSA level increases steadily as he ages, and some urologists advocate the use of age-related PSA cut-points, rather than using > 4 ug/L for all.

The table below shows suggested age-specific ranges. In our analysis we will set a binary

Table 2.1: Age-related normal PSA cut-points

Age Range (years)	Serum PSA Concentration(ug/L)
40-49	< 2.5
50-59	< 3.5
60-69	< 4.5
70-79	< 6.5

Source: Oesterling JE et al. JAMA 1993; 270:860

cutoff point of PSA > 7 and use the resulting classification variable as our main response. The data sets we use here are two: the Cambridge and the Stockholm data sets. Together, the two data sets contain observation on 212 men. Variables such as PSA values, age, Gleason scores as well as survival times are recorded. In addition, expressions for over 40k genes are also available for this group. This makes the current data a high-dimensional data. Our objectives from the current analysis is to find if there is a group of 10 genes whose expressions could be used to classify high versus low PSA after controlling for the effect of age. In the following, we perform the analysis using common R packages.

2.2 Analysis

Loading the required Rpackages

```
##Load the required libraries


library(ggplot2)

library(tidyr)

library(devtools)
library(dplyr)

library(party)

library(survival)

library(SuperLearner)

library(data.table)

library(glmnet)

library(randomForestSRC)
```

Below we manipulate the data sets inside the prostatecamcap and prostateStockholm R packages and combine them in one big data frame that contains gene expressions and patient demographics. We also remove genes with missing names.

```
library(prostateCancerCamcap)

data(camcap,package = 'prostateCancerCamcap')


pd_camcap <- tbl_df(pData(camcap))




fd_camcap <- tbl_df(fData(camcap))

fd_camcap=fd_camcap%>%

  mutate(Gene=replace(Symbol,Symbol %in% c("","NA"),NA))%>%

  filter(Gene != "NA")


exp_camcap <- tbl_df(data.frame(ID = as.character(featureNames(camcap))

,exprs(camcap)))
```

```
pd_stockholm <- tbl_df(pData(stockholm))

fd_stockholm <- tbl_df(fData(stockholm))


fd_stockholm=fd_stockholm%>%

  mutate(Gene=replace(Symbol,Symbol %in% c("","NA"),NA))%>%

  filter(Gene != "NA") exp_stockholm =tbl_df(data.frame(ID =



  as.character(featureNames(stockholm)),exprs(stockholm)))
```

Next, we store all the gene expressions for later use .. There are more than 40k genes.

```
genescam=fd_camcap%>%select(Gene)%>%unique()

genesstock=fd_stockholm%>%select(Gene)%>%unique()

genes0=genescam%>%inner_join(genesstock)

genes=genes0%>%mutate(Gene=droplevels(Gene))
```

We made a word cloud of the prefixes of the names of the genes. The word cloud shows that LOC and CORF are the most common gene prefixes. The LOC prefix means that the gene's functions are unknown. So the LOC is a sort of place holding until more informative name can be found. So, practically we are looking for a needle in a hey stack.

Figure 2.1: Word cloud for all the genes in the data base

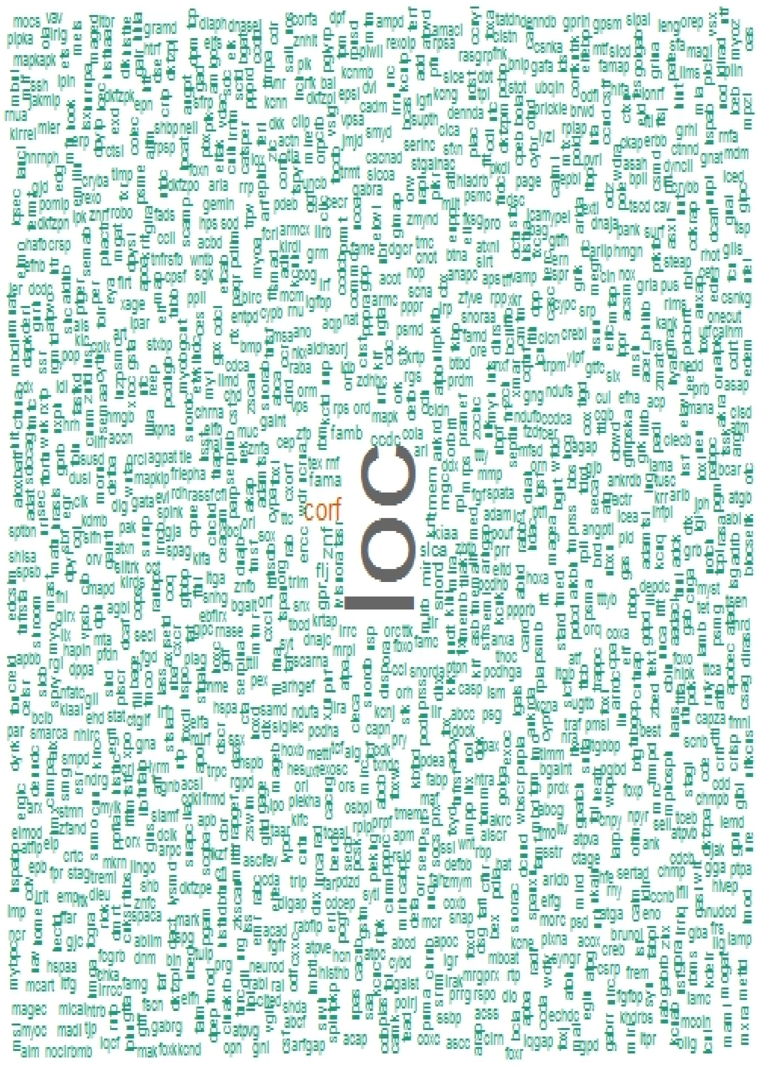
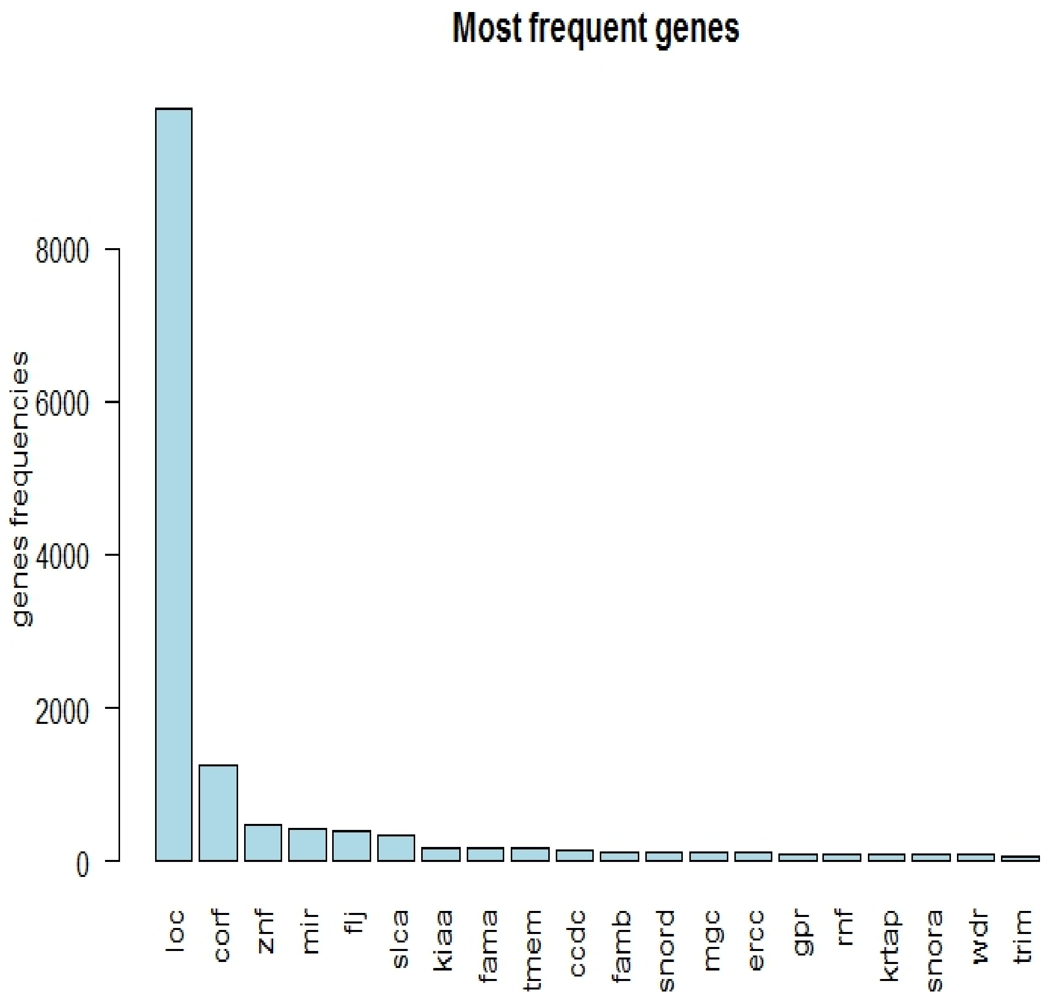


Figure 2.2: The first 100 most frequent genes (all the genes)



Here we start the main algorithm by creating training and test data indices. The total sample of individuals in the two sets after removing individuals with missing PSA or age is 212. We designate 80% to training data set and the remaining to validation (test).

```
trainindex=sample(1:212,168,replace=FALSE)
```

We sample 10 genes randomly from the pool of all genes in the data base, then pull out their expression data and then clean out repeated expressions (from different probes of the same gene), by choosing only the most variable one based on IQR statistic.

```
genes=genes0%>%sample_n(10, replace=FALSE)%>%select(Gene)
 probes <- fd_camcap %>% filter(Gene %in% genes$Gene) %>%
  select(ID) %>% unique %>% as.matrix %>%  as.character


data<- exp_camcap  %>% filter(ID %in% probes) %>%
  tidyr::gather(geo_accession,Expression,-ID)


 summary_stats <- data %>% group_by(ID) %>%
  summarise(mean=mean(Expression,na.rm=TRUE),sd=sd(Expression,na.rm=TRUE),
            iqr=IQR(Expression,na.rm=TRUE))


 data <- left_join(data,summary_stats) %>%
  mutate(Z = (Expression - mean) / sd)


mostVarProbes <- left_join(summary_stats,fd) %>%
  arrange(Symbol,desc(iqr)) %>%
  distinct(Symbol,.keep_all=TRUE) %>%
  select(ID) %>%  as.matrix %>%  as.character


data <- filter(data, ID %in% mostVarProbes)
data <- left_join(data, select(fd_camcap, ID, Symbol))
data <- left_join(data, pd_camcap)
```

Remove the auxiliary variables used such as the IQR, and z-scores. Then extract only age, PSA variables and the gene expression variables. Also, we classify the PSA level into above and below 7 (high and low).

```
data=data%>%select(-Z,-mean,-sd,-iqr,-ID)%>%
  tidyr::spread(Symbol,Expression)


data2=data%>%select(PSA,Age,names(data[,16:25]))%>%
  mutate(Age=as.numeric(as.character(Age)),


  PSA=as.numeric(as.character(PSA)))%>%
  na.omit()


data2cam=data2%>%mutate(PSAcp=ifelse(PSA>7,1,0))%>%select(-Age)
```

Next we do the same as above for the Stockholm data

```
###########Here starts same as above for Stockholm data
probes <- fd_stockholm %>% filter(Gene %in% genes$Gene)

 %>% select(ID) %>% unique %>% as.matrix %>%  as.character
```

```
data<- exp_stockholm  %>% filter(ID %in% probes) %>%

  tidyr::gather(geo_accession,Expression,-ID)


summary_stats <- data %>% group_by(ID) %>%

  summarise(mean=mean(Expression,na.rm=TRUE),

  sd=sd(Expression,na.rm=TRUE),iqr=IQR(Expression,na.rm=TRUE))


data <- left_join(data,summary_stats) %>%

 mutate(Z = (Expression - mean) / sd)


mostVarProbes <- left_join(summary_stats,fd) %>%

  arrange(Symbol,desc(iqr)) %>%

  distinct(Symbol,.keep_all=TRUE) %>%

  select(ID) %>%  as.matrix %>%  as.character


data <- filter(data, ID %in% mostVarProbes)

data <- left_join(data, select(fd_stockholm, ID, Symbol))

data <- left_join(data, pd_stockholm)


#data <- data %>% filter(!is.na(Time) & !is.na(Event))




data=data%>%select(-Z,-mean,-sd,-iqr,-ID)%>%
tidyr::spread(Symbol,Expression)


data2=data%>%select(PSA,names(data[,13:22]))%>%

mutate(PSA=as.numeric(as.character(PSA)))%>%na.omit()


data2stock=data2%>%mutate(PSAcp=ifelse(PSA>7,1,0))
```

We join the data sets into one dataset and print first few rows

```
 data22

# A tabble: 213 x 12

     PSA  PHB2  OSBP   CGA APBB3  TG   SCCPDH



   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>

 1  28     9.68 10.1   6.57  8.94  6.05   8.26

 2  99     9.62  9.55  6.44  9.84  5.94   9.01

 3  75.2   9.65 10.0   6.75  9.27  6.19   8.78

 4 215     9.77 10.2   6.56  9.66  6.09   8.27

 5  18.6   8.98  9.15  6.84  8.96  6.17   8.30

 6 200     9.62 10.6   6.61  9.63  5.94   8.04

 7  60     9.52 10.9   6.46  9.67  6.08   8.57

 8  48.6   9.27 10.1   6.80  9.40  5.93   8.62

 9 282     9.33  9.94  6.33  9.36  6.18   8.31

10 250     9.38 10.5   6.48  8.88  6.38   8.05
```

```
ERCC-00017` LOC100132658  LOC642367  LOC727878   PSAcp

<dbl>             <dbl>     <dbl>     <dbl>       <dbl>

6.12              8.02       8.71      6.22         1

6.07              7.57       7.74      6.10         1

5.95              8.12       8.04      6.40         1

6.10              7.58       7.93      6.36         1

5.91              8.02       7.80      6.28         1

6.21              8.48       7.58      6.19         1

6.06              8.34       7.57      6.25         1

6.08              7.83       7.85      6.27         1

6.24              8.33       8.27      6.38         1

6.28              7.34       8.56      6.31         1


# ... with 203 more rows
```

Take the index of the training data and create the response and covariates for the two sets. We then build the superlearner on the training data and compute the AUC on the validation set.

```
train_obs =  trainindex

data3=data22%>%select(-PSA,-PSAcp)

X_train = data3[train_obs, ]

X_holdout = data3[-train_obs, ]
```

```
outcome_bin = data22$PSAcp

Y_train = outcome_bin[train_obs]

Y_holdout = outcome_bin[-train_obs]
```

We now modify the ridge and lasso regressions that are used as two of the possible algorithms in the library of the superlearner so that these two algorithms use only $V = 4$ fold CV when choosing the penalty tuning parameter and we limit the number of the grid of tuning parameters to nlambda= 10. This is simply a modification of the wrappers of the glmnet that superlearner uses internally.

```
SL.glmnet.lasso = function(...) {

  SL.glmnet(..., nfolds = 4,nlambda=10)

}


SL.glmnet.ridge = function(...) {

  SL.glmnet(..., nfolds = 4,nlambda=10, alpha = 0)

}
```

```
library(SuperLearner)

SL.library <-  c("SL.mean", "SL.glmnet.lasso",


"SL.randomForest","SL.glmnet.ridge","SL.glm")
```

Next we run the main SL algorithm using lasso, ridge, usual logistic regression, simple average, and random forest as our base library of algorithms.

```
sl<- SuperLearner(Y = Y_train, X = X_train, SL.library = SL.library,
                          method = "method.AUC", family = binomial())
sl
```

```
                         Risk      Coef
SL.mean_All          0.6672714 0.1285674
SL.glmnet.lasso_All 0.6627362 0.1285674
SL.randomForest_All 0.5283447 0.5801448
SL.glmnet.ridge_All 0.6119426 0.1627204
SL.glm_All          0.5439153 0.0000000
>
```

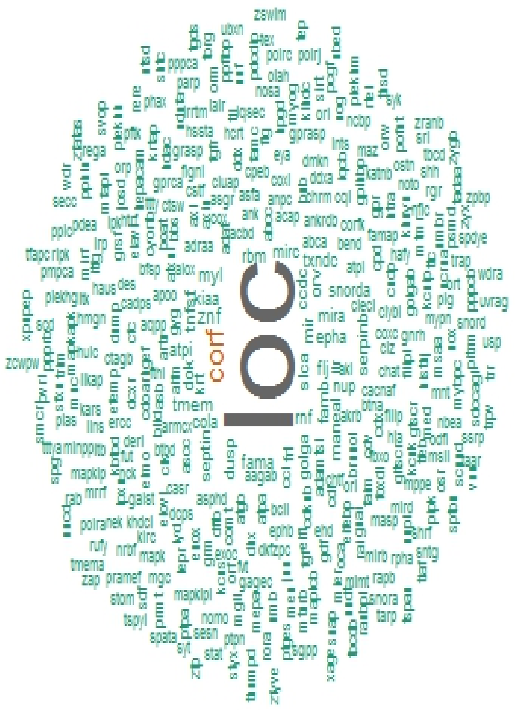
The output shows values of the cross-validated AUCs for each algorithm as well as the optimal α values. It is clear that the weights sum to one and the simple logistic regression algorithm has zero weight. The highest weight of 58% goes to the random forest, followed by ridge regression.

Here, we use the holdout data (test data set) to compute the AUC for the superlearner algorithm just build on the training data. Then the AUCs were stored along with names of the 10 genes that were chosen.

```
pred = predict(sl, X_holdout, onlySL = T)



pred_rocr = ROCR::prediction(pred$pred, Y_holdout)

auc = ROCR::performance(pred_rocr, measure = "auc",


x.measure = "cutoff")@y.values[[1]]

auc

   aucv[i]=auc

  genesstore[i,]=transpose(genes)
```

The above operation is then repeated for 2000 times and every time the 10 genes selected for the analysis and the resulting AUC are stored. Finally, below is a plot of the AUCs versus indices of the gene groups. Using the plot, we can see the patches of 10 genes that have the highest AUCs. We set a threshold of 0.8 and above for an AUC to be considered high. The resulting groups of gene from this criteria are then stored and their word cloud plotted.

Figure 2.3: Generate the Word cloud



The above word cloud clearly shows that again genes with prefixes Loc and Corf and znf are the three most important genes in the data that could be used to explain high PSA. The znf family appeared in some literature as correlated with PSA.

Figure 2.4: plot of frequencies of the first 10 frequent genes

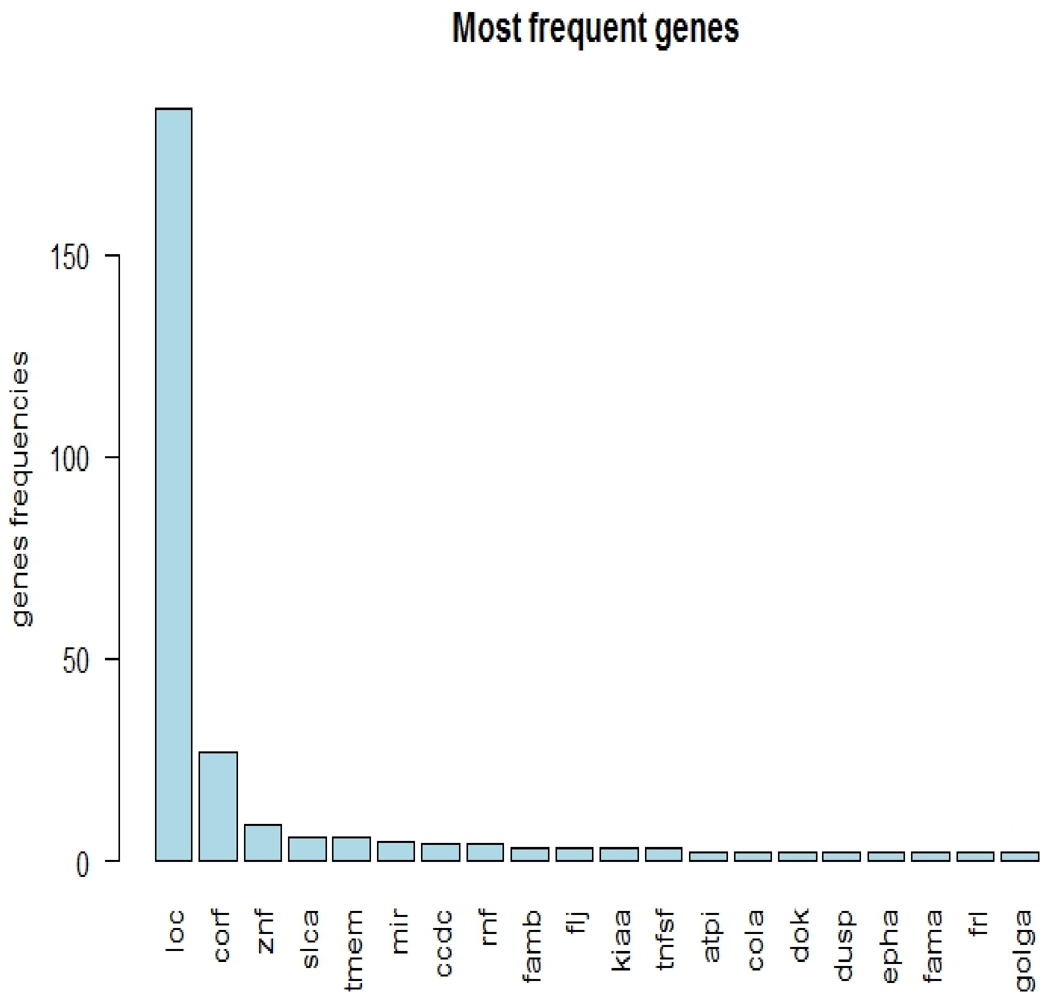
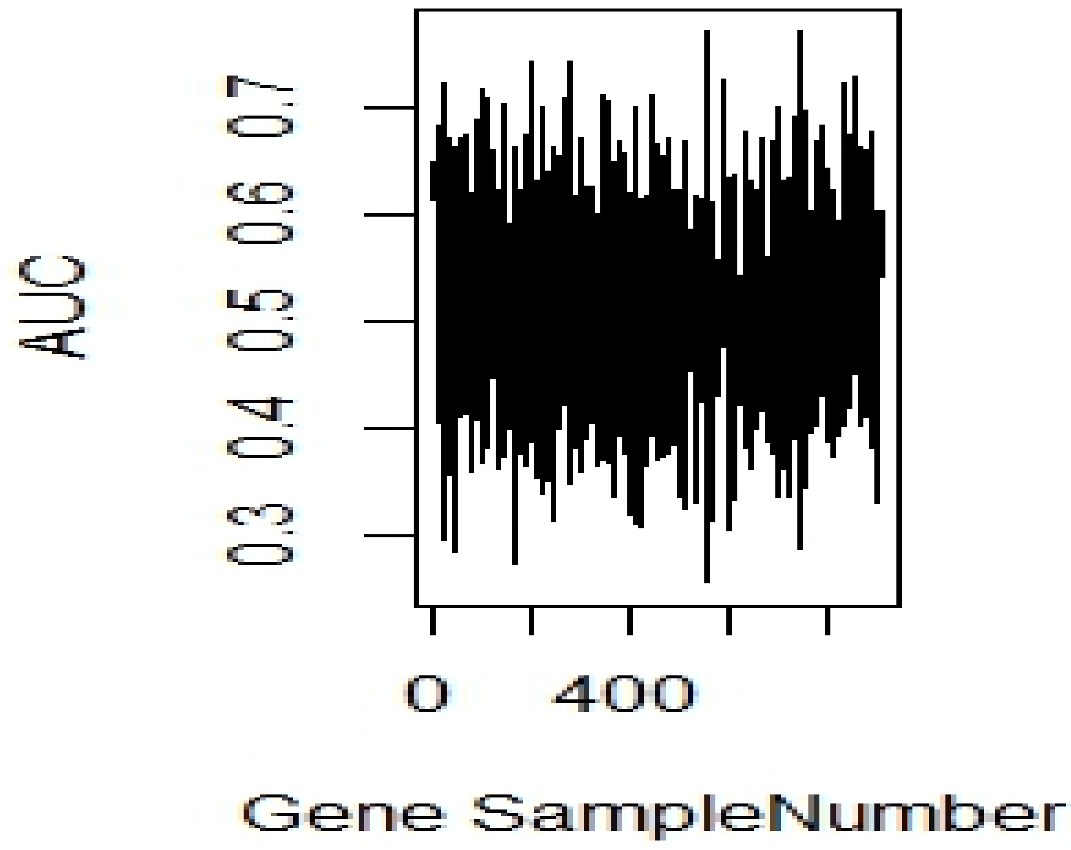


Figure 2.5: plot of AUC Vs Sample Number

vs Sample Number.jpeg



Chapter 3

Conclusions and Future Directions

3.1 Summary of the Results

In this MSc research project, we tried to illustrate the use of the superlearner algorithm of van de Laan et al[7] to classify 212 men with prostate cancer into two groups: a group with PSA less than 7 and group with PSA higher than 7. The age was used as covariate to adjust for and the classification ability of groups of 10 gene expressions randomly selected from a data base of over 47, 000 gene expressions was explored. The data was split into training (80%) and a test data sets. The operation of choosing 10 genes randomly from the pool, extracting their expression values, running a superlearner algorithm on the training data and building a superlearner model that can predict AUC was repeated about 1000 times. The superlearner algorithm is an ensemble method that combines the predictions from several supervised learning algorithms. We chose the logistic regression, lasso and ridge versions of logistic regression, a random forest and a simple mean model as the methods to be ensembled. The algorithm was then applied to the test data and an AUC was estimated for every run of this gene mining operation. Finally, the AUCs of the 1000 groups of 10 genes were examined to find those with AUCs exceeding 0.7 and this collection of gene names were reported in a word cloud.

Unfortunately, we have found that most of those genes had prefix LOC, meaning that they are not well known. We do not know if some these genes are interesting and really related to PSA or can serve as a proxy for high PSA. There are a few shortcomings of this type of exercise: (1) given the large number of genes whose expression data are available and given that most of genes are likely not related to PSA, sampling 10 at time may not be sufficient to capture a representative picture of the pool of genes, (2) given the small size of this data, dividing it into training and testing sets could have reduced the power of the approach, (3) the number of ensembled algorithms is too small.

Therefore, in the future, one may want to pool more observations from different studies to increase the sample size. Also, the number of genes samples should be increased substantially even at the cost of making the problem a high-dimensional one whereby ($p >> n$) and then deploy more learners that are designed for such situations (e.g., random forest, neural networks and similar algorithms).

This exercise, which we have accomplished in this major paper, was just to explore the possibility of applying superlearner on publicly available prostate cancer data. The more interesting undertaking would be that of using the superlearner learning about the survival of the PCa patients, and not so much about the PSA levels.

Bibliography

[1] L. Breiman.Stacked regressions.Machine Learning ,45:5-32,2001.

[2] G. James,D. Witten,T.Hastie and R.Tibshirani.*An Introduction to Statistical Learning.* Springer, New York,2013.

[3] E.LeDell. Statistician and Machine Learning Scientist ,H2o.ai

[4] E.LeDell, M. Petersen, and M. van der Laan. Computationally efficient confidence intervals for cross-validated area under the ROC curve estimates .*Electronic journal of statistics* , 9(1), 1583-1607,2015.

[5] J.E. Oesterling ,S.S. Jacobsen ,C.G. Chute,H.A. Guess and C.J.Girman.Serum prostate specific antigen in a community-based population of healthy men:*Establishment of age-specific ranges* . JAMA ,270(7):860-4,1993 .

[6] E.C. Polley ,E.LeDell ,M.Van der Laan, C.Kennedy . Super Learner in Prediction . *URL https://CRAN.R-project.org/package=SuperLearner. R package version.* 2.0-22,2016.

[7] J.A.Steingrimsson ,L.Diao and R.L.Strawderman.Censoring Unbiased Regression Trees and Ensembles .Rochester, New York,2018.

[8] M.J. van der Laan,E.C. Polley, and A.E.Hubbard. Super Learner. Statistical Applications of Genetics and Molecular Biology, 6, article 25. *http://www.degruyter.com/view/j/sagmb.2007.6.issue-1/sagmb.2007.6.1.1309/sagmb.2007.6.1.1309.xml,2007.*

[9] D.H Wolpert. Stacked Generalization. *Neural Networks* 5, 1992,241-259,1992.

Vita Auctoris

Ebo Essilfie-Amoah was born in Ghana, West Africa. He attended the University of Cape Coast, obtaining a BSc.Honours in Mathematics Dip. in Education, post Degree Diploma in Actuarial Science University of Waterloo(2008) and Master in Business Administration, University of Phoenix, USA (2010) . He is currently a candidate for the M.Sc.in Statistics from the University of Windsor.