# Deep Learning and other advanced methods in healthcare: non-imaging examples

Arun Aryasomayajula

Data Science

@PopHealthCare®

Greater Nashville Healthcare Analytics Meetup

20 Feb 2018

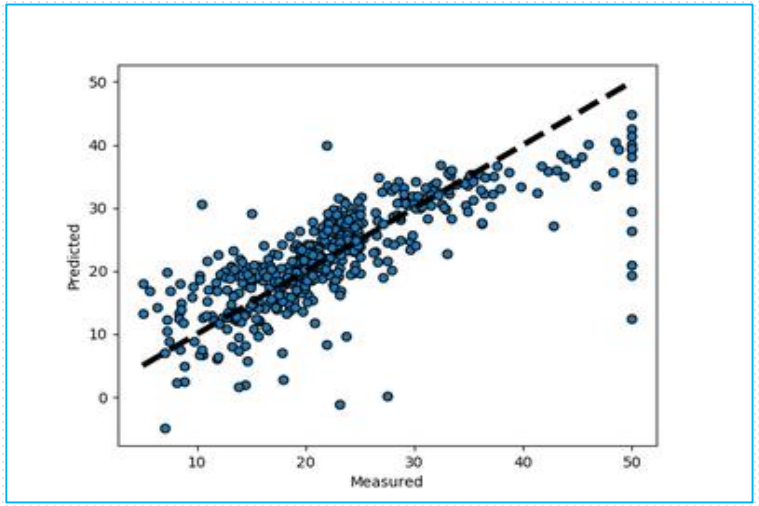
# Agenda

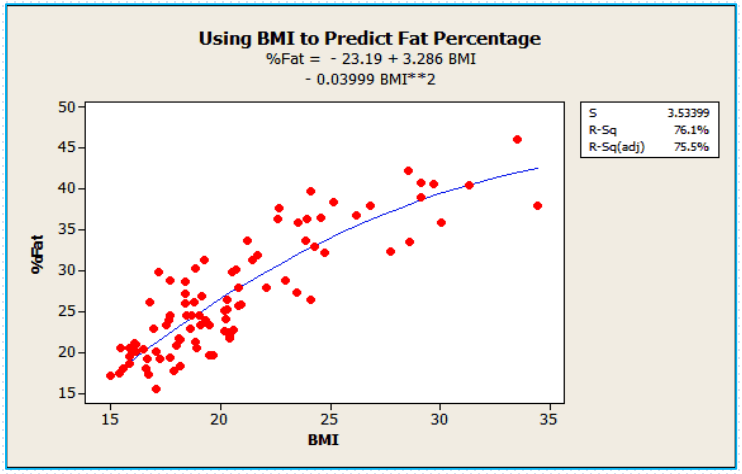
- About PopHealthCare®
- A primer on types of predictive algorithms
- Deep Learning architectures
- Autoencoders – a case study.
- CNN for cancer immunotherapy– a case study.
- NLP and Word Embedding Models.
- Questions
- Announcements

# About PopHealthCare®

- PopHealthCare (www.pophealthcare.com), PHC in short is an industry leader who partners with payers and providers to deliver proven risk adjustment, HEDIS, and high-risk population management programs.
- PHC has corporate offices in both Tempe, AZ and Nashville, TN.
- We currently serve over 40 health plan clients, impacting members in 49 states and Puerto Rico.
- PHC joined the GuideWell group of companies. http://www.guidewell.com/blog/guidewell-mutual-holding-corporation-acquires-pophealthcare-llc
- PHC makes informed decisions impacting its business with the help of Data Science and Analytics organization ranging from predictive modeling, machine learning, interactive visualizations etc.

# A primer on predictive algorithms

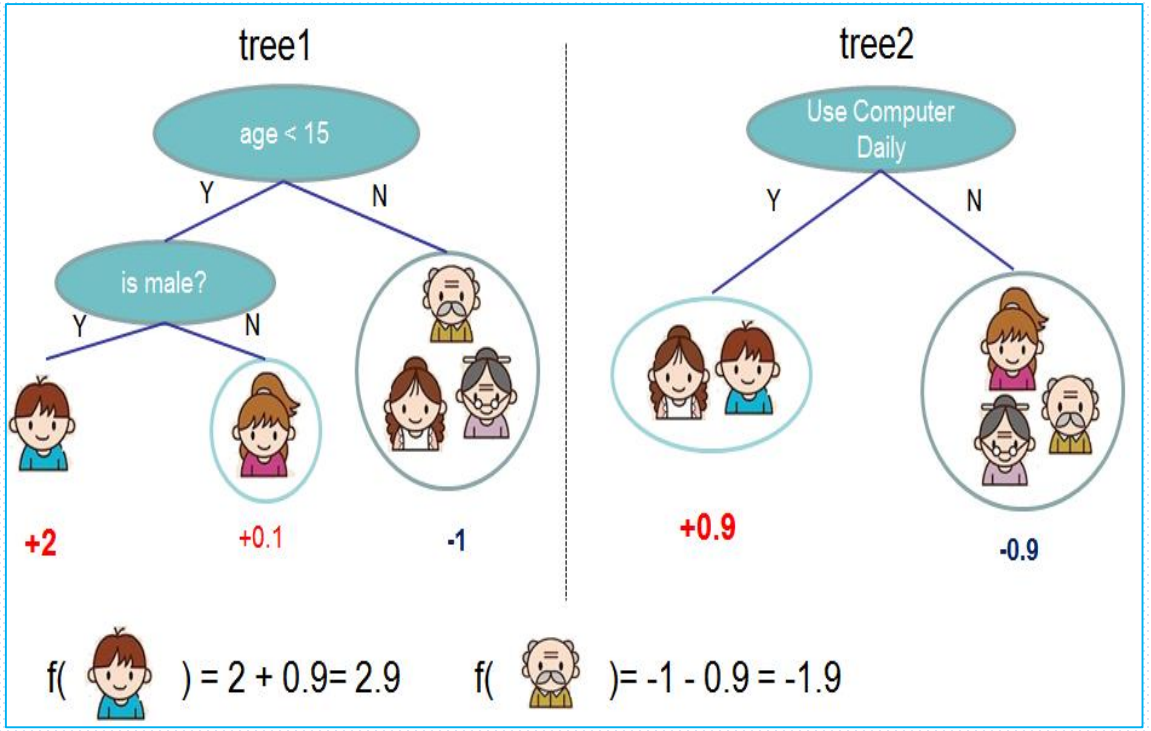


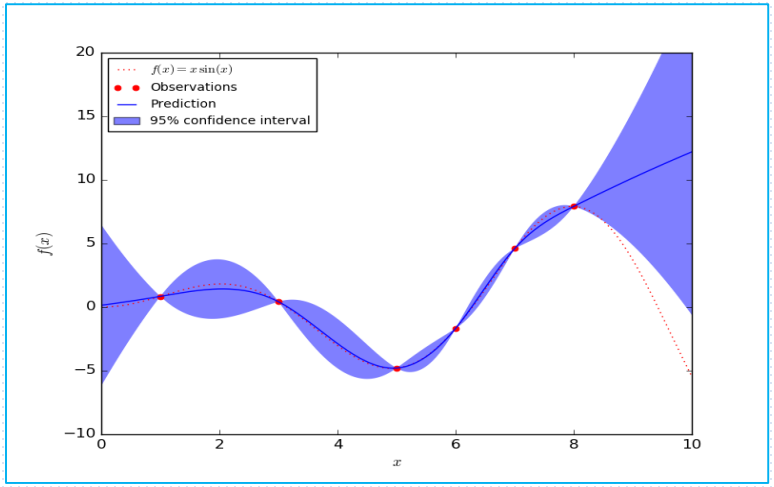

- Predictive algorithms can be categorized broadly into three types depending on what we create as an end product after training:
- Structural equations:
- Y = f(x) where f(x) is algebraically defined.
- Ex: Y = a*x+b*y+c or
- Y = exp(-(x-mu)/sig^2) etc.
- We start by assuming the form of relationship between our prediction and predictors.
- Eg: linear regression assumes a linear relationship.

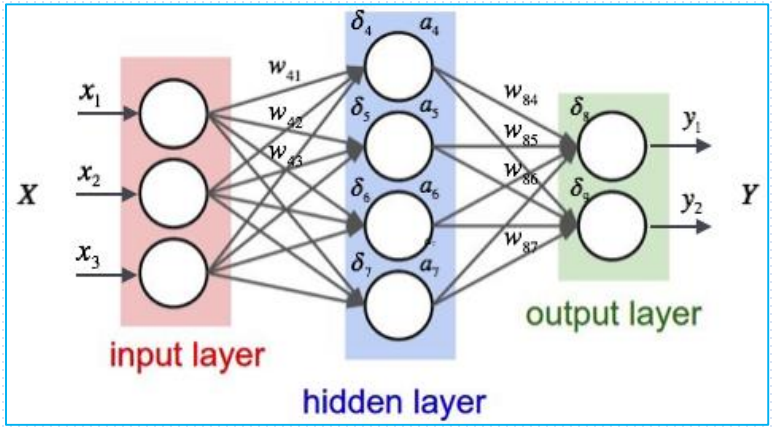
# A primer on predictive algorithms

- Tree based algorithms:
- These don’t assume a structural relationship.
- The algorithm creates a tree like structure with nodes and branches with each node representing a decision point. The data is divided at each node.
- These models in essence produce business rules derived from the data directly towards a goal like classification or regression problems.
- Very popular with lots of advanced ensemble methods available like RandomForest, Gradient Boosted Machines, XGBoost, Light-GBM, CatBoost etc.



# A primer on predictive algorithms




- Function approximation algorithms:
- These assume a structural relationship of the kind Y = f(X). The form of the function isn't assumed but is regressed from data.
- Eg: Gaussian Processes (some assumption required), Deep Learning Models (Aha!).
- For reference,

https://en.wikipedia.org/wiki/Gaussian_process

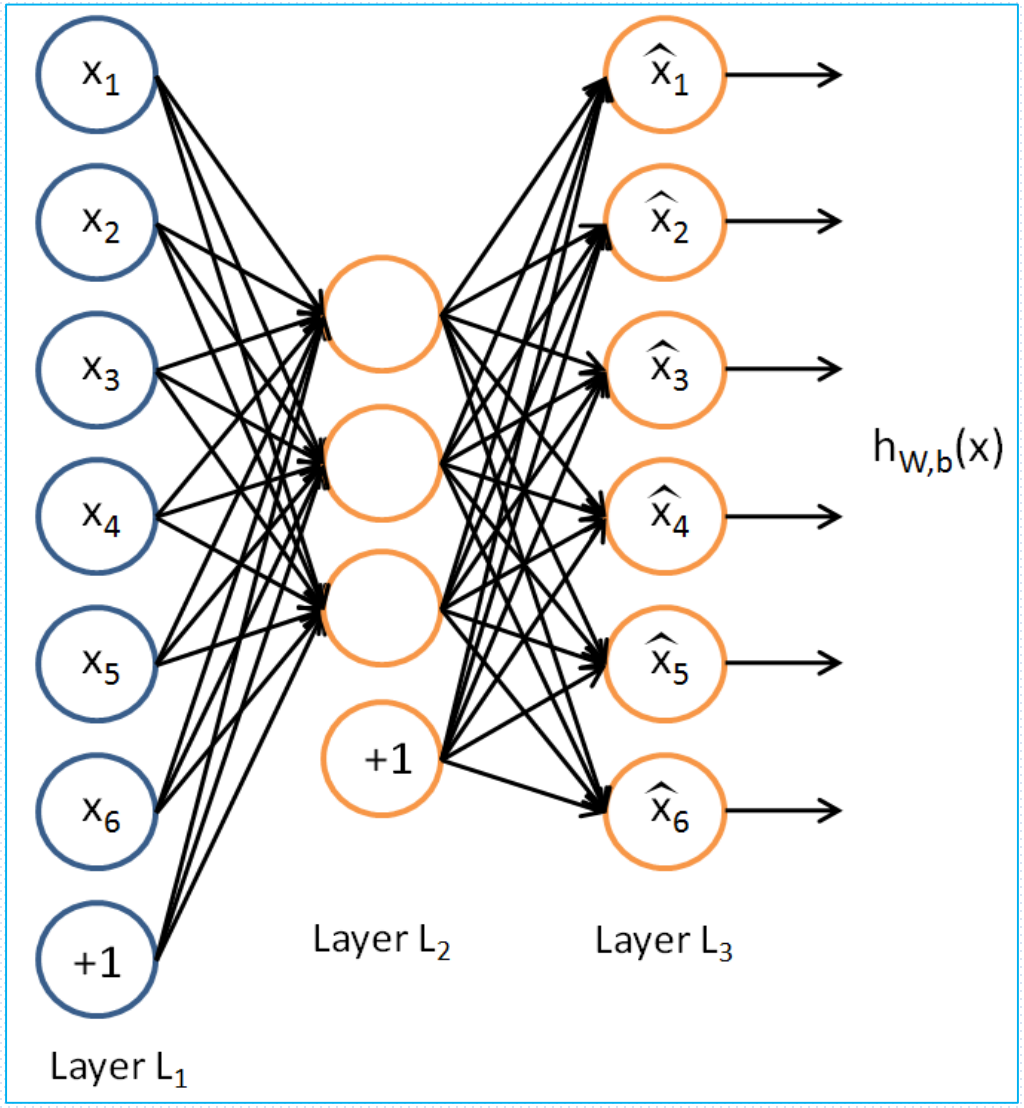
https://en.wikipedia.org/wiki/Deep_learning

# Types of Deep Learning Architectures

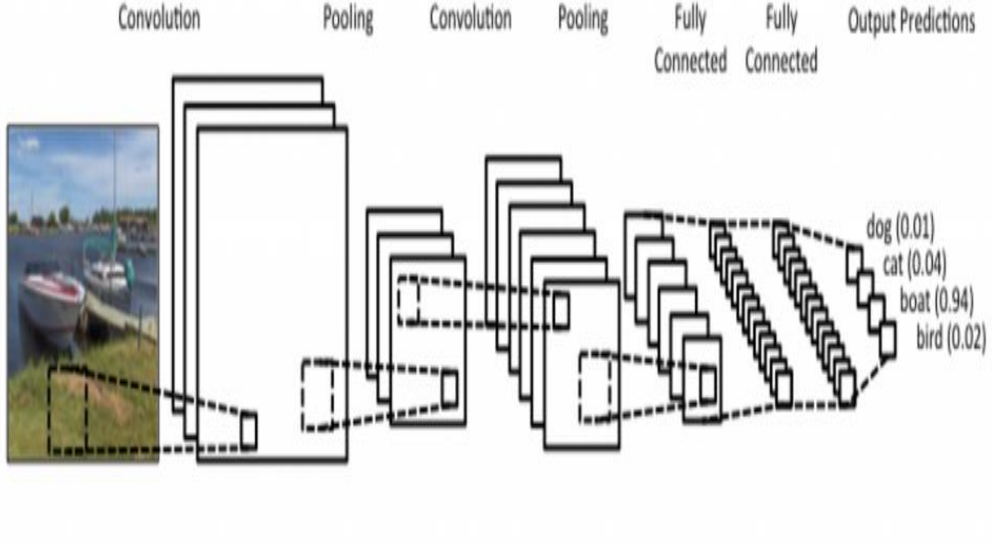
- Autoencoders
- Convolutional Neural Networks (CNN)
- Recurrent Neural Networks (RNN)
- Long Short Term Memory Networks (LSTM)
- Generative Adversarial Networks (GAN) and lots more (literally 1000s more!)

# Autoencoders

- An **autoencoder** neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs.
- The autoencoder tries to learn the identity function f(x) = x
- By putting constraints on the number of layers and units one can uncover a great deal about hidden patterns in the data.
- Autoencoders are related to PCA and other dimensionality reduction techniques, but can learn more complex mappings due to their nonlinear nature.
- A wide range of autoencoder architectures exist, including Denoising Autoencoders, Variational Autoencoders, or Sequence Autoencoders.
- Ref:[http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/](http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/)

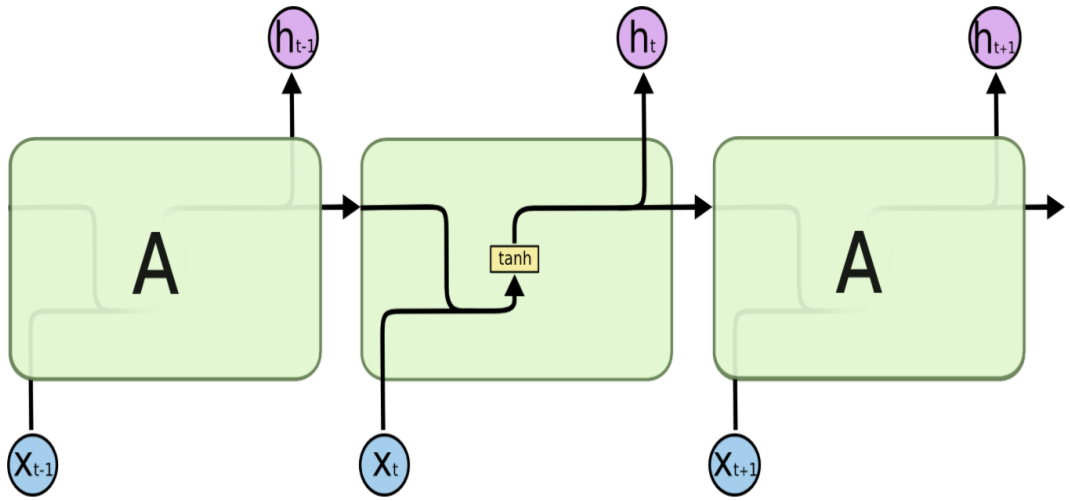


# Convolutional Neural Networks (CNN)




- A CNN uses convolutions to connected extract features from local regions of an input.
- Most CNNs contain a combination of convolutional, pooling and affine layers.
- CNNs have gained popularity particularly through their excellent performance on visual recognition tasks, where they have been setting the state of the art for several years.
- Stanford CS231n class – Convolutional Neural Networks for Visual Recognition
- Understanding Convolutional Neural Networks for NLP

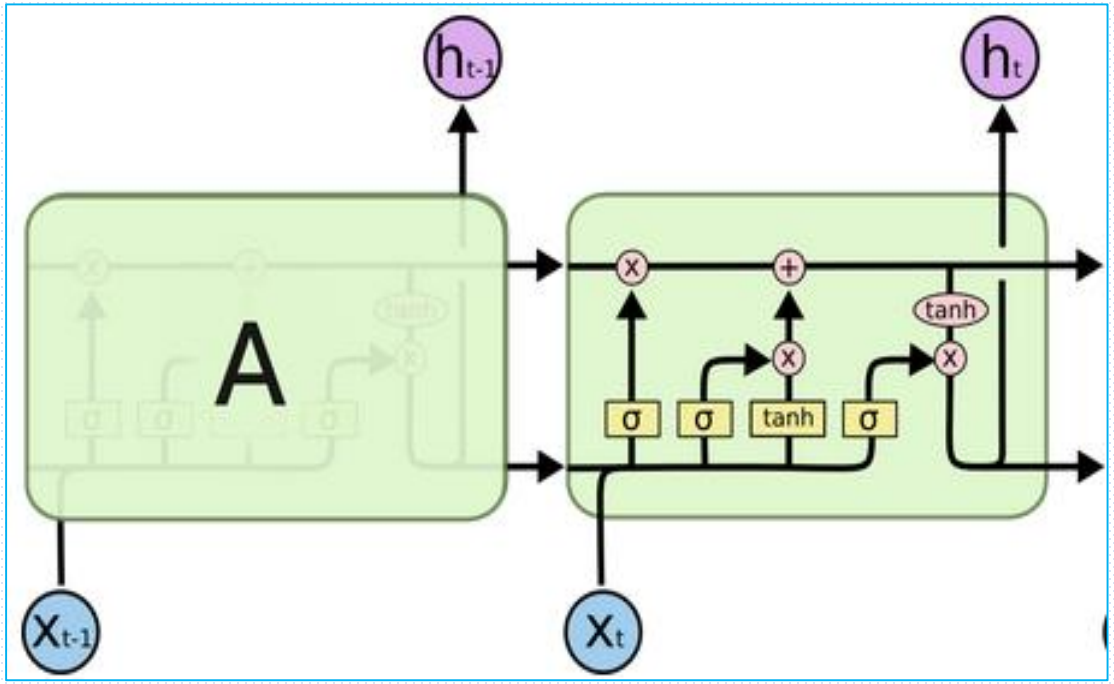
# Recurrent Neural Networks (RNN)




The repeating module in a standard RNN contains a single layer.

- A RNN models sequential interactions through a hidden state, or memory.
- It can take up to N inputs and produce up to N outputs. For example, an input sequence may be a sentence with the outputs being the part-of-speech tag for each word (N-to-N).
- An input could be a sentence, and the output a sentiment classification of the sentence (N-to-1). An input could be a single image, and the output could be a sequence of words corresponding to the description of an image (1-to-N).
- At each time step, an RNN calculates a new hidden state (“memory”) based on the current input and the previous hidden state.
- The “recurrent” stems from the facts that at each step the same parameters are used and the network performs the same calculations based on different inputs.
- Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs

# Long Short Term Memory Networks(LSTM)




- Long Short-Term Memory networks were invented to prevent the vanishing gradient problem in Recurrent Neural Networks by using a memory gating mechanism.
- Using LSTM units to calculate the hidden state in an RNN we help to the network to efficiently propagate gradients and learn long-range dependencies.
- Long Short-Term Memory
- Understanding LSTM Networks
- Recurrent Neural Network Tutorial, Part 4 – Implementing a GRU/LSTM RNN with Python and Theano

# Case Study 1: How to make better classifiers using AutoEncoders?

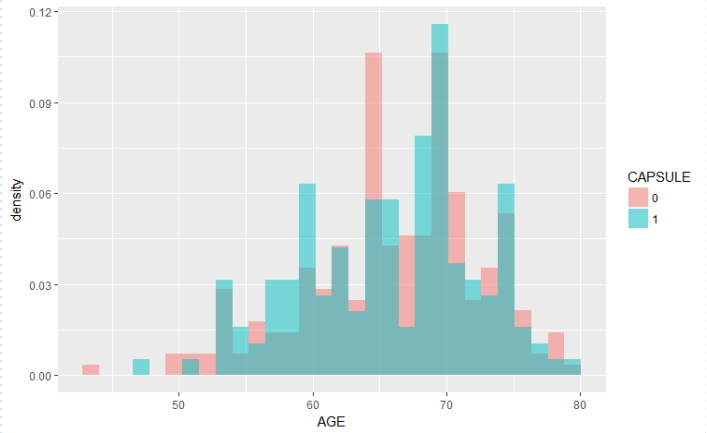
| |
|---|
| In this use case, a prostate cancer dataset is separated into “easy” vs “hard” to model subsets using autoencoders and to gain predictive accuracy in classifying cancer vs non-cancer. |
| The dataset has about 380 rows with 153 positives.<br>H2O.ai’s deep autoencoder and anomaly detection models are used within R. |
| H2O is an open-source software for machine learning and big-data analysis. It’s a JVM based truly parallel platform with both R and Python wrappers. (https://www.h2o.ai/download/ ) |
| It offers various models such as GLM, GBM and Random Forest, but more importantly, offers deep learning framework and a distributed computing architecture. |

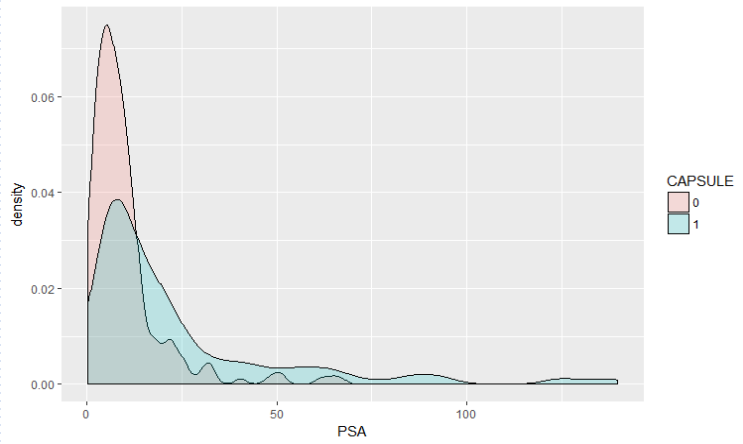
# Prostate Cancer dataset glossary

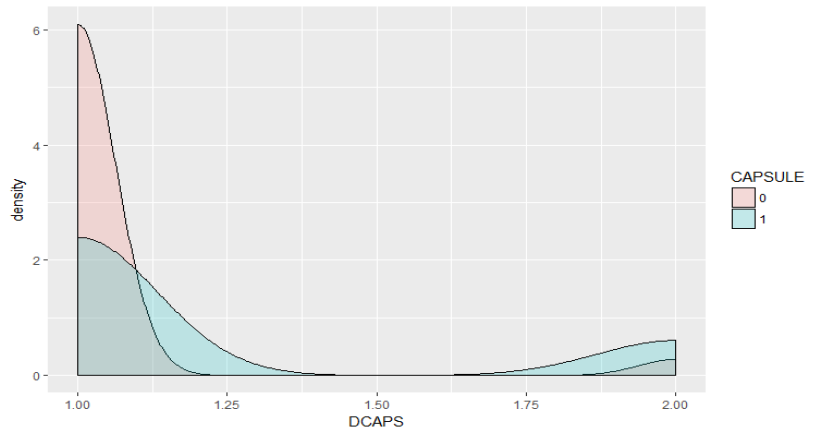
- Gleason Score: Pathologists grade prostate cancers using numbers from 1 to 5 based on how much the cells in the cancerous tissue look like normal prostate tissue under the microscope. This is called the *Gleason system*. Grades 1 and 2 are not often used for biopsies − most biopsy samples are grade 3 or higher. (integer from 1-10)
- PSA: Prostate-specific antigen, or PSA, is a protein produced by normal, as well as malignant, cells of the prostate gland. The PSA test measures the level of PSA in a man's blood. For this test, a blood sample is sent to a laboratory for analysis. The results are usually reported as nanograms of PSA per milliliter (ng/mL) of blood. (mg/ml)
- DCAPS: Capsular involvement on rectal exam(yes or no)
- DPROS: digital prostate exam (factor)
- VOL: Tumor Volume (cm3)
- CAPSULE: Tumor penetration of prostatic capsule (yes or no)
- RACE: Race (White or Black)

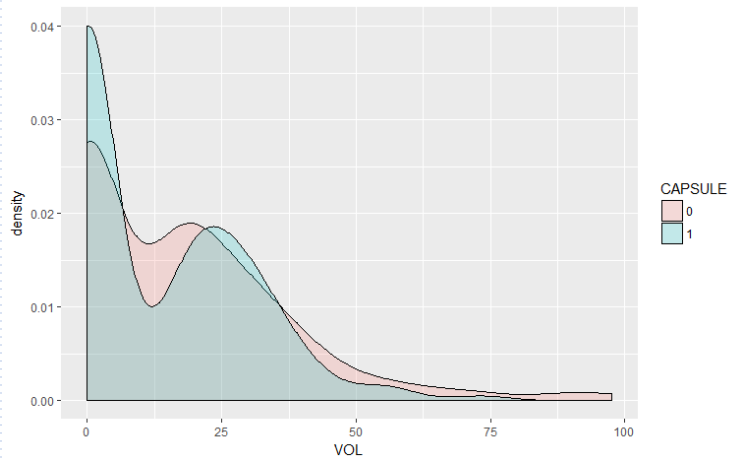
```
library(h2o)
prosPath = system.file("extdata", "prostate.csv",
package = "h2o")
prostate_df <- read.csv(prosPath)
```

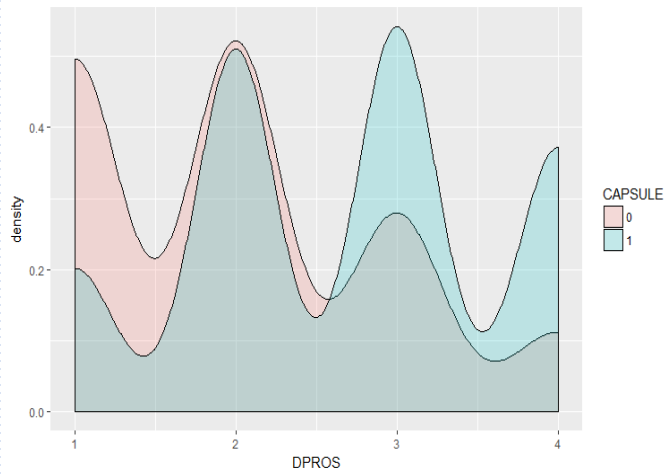
# Prostate Cancer Dataset

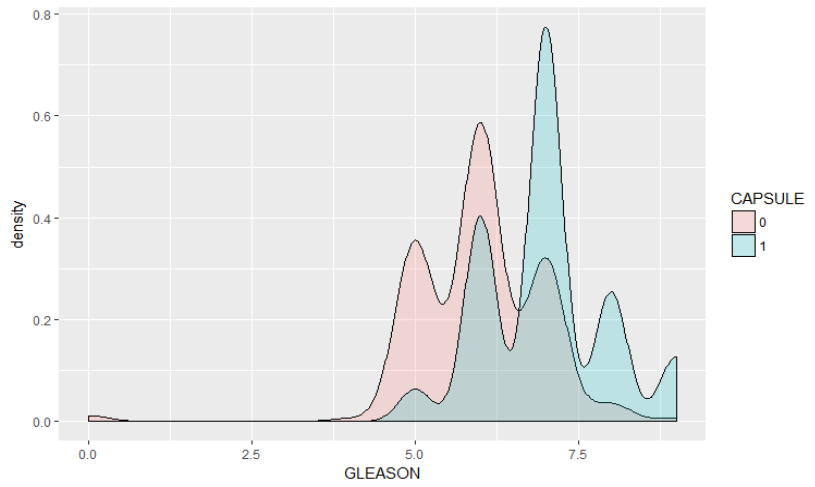


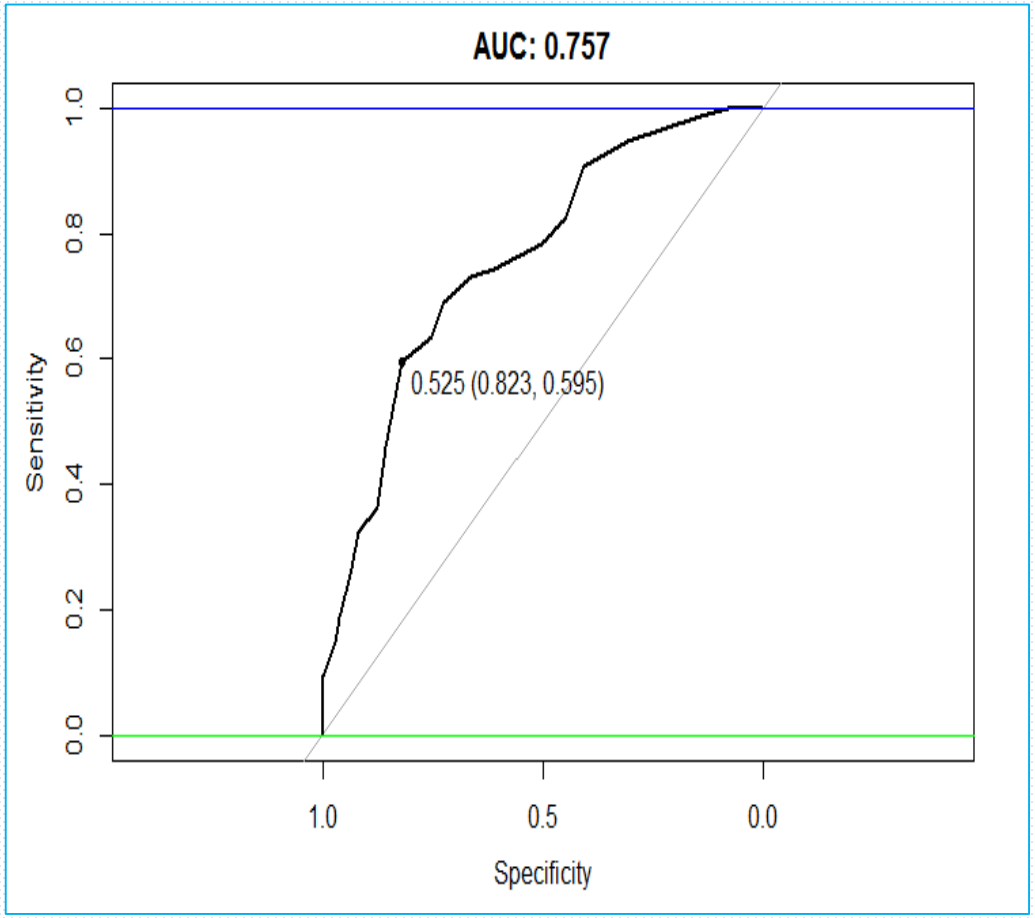



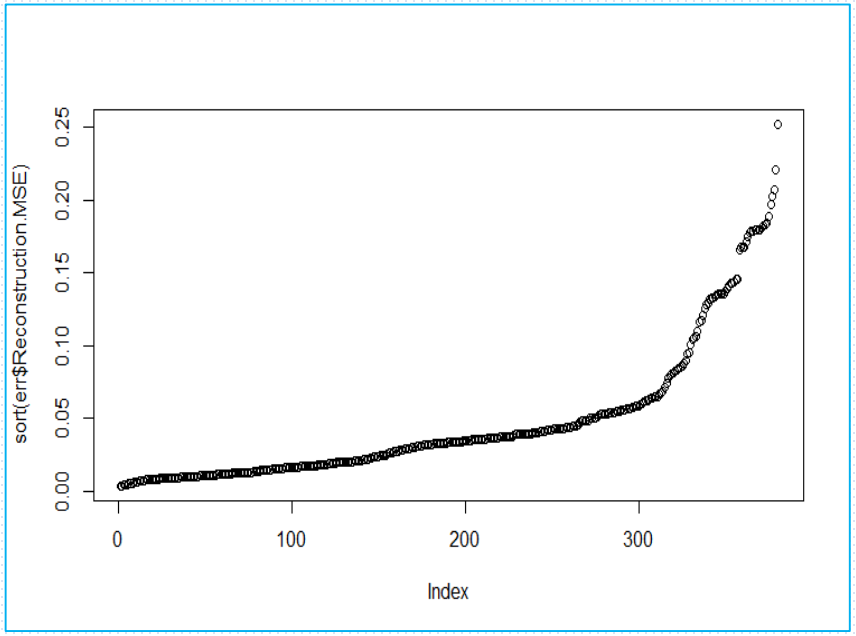
# Building RandomForest Model for Benchmarking in R

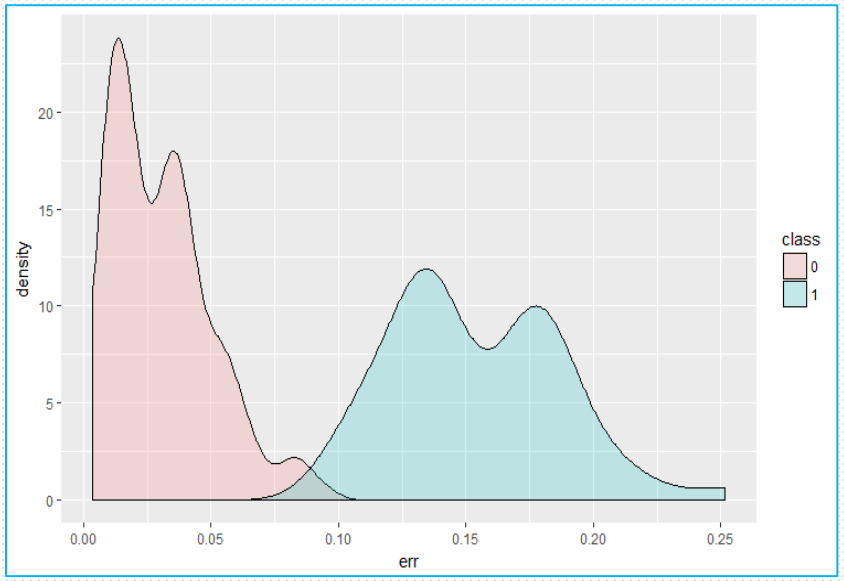
```
library(randomForest)
feature_names <- setdiff(names(prostate_df), “CAPSULE”)

set.seed(1234)
rf_model <- randomForest(x=train_df[,feature_names],
                         y=as.factor(train_df[,outcome_name]),
                         importance=TRUE, ntree=20, mtry = 3)

validate_predictions <- predict(rf_model,
newdata=validate_df[,feature_names], type="prob")
```

```
install.packages('pROC’)

library(pROC)

auc_rf = roc(response=as.numeric(as.factor(validate_df[,outcome_name]))-1,
             predictor=validate_predictions[,2])

plot(auc_rf, print.thres = "best", main=paste('AUC:',round(auc_rf$auc[[1]],3)))

abline(h=1,col='blue')
abline(h=0,col='green')
```



# Building autoencoder model in H2O



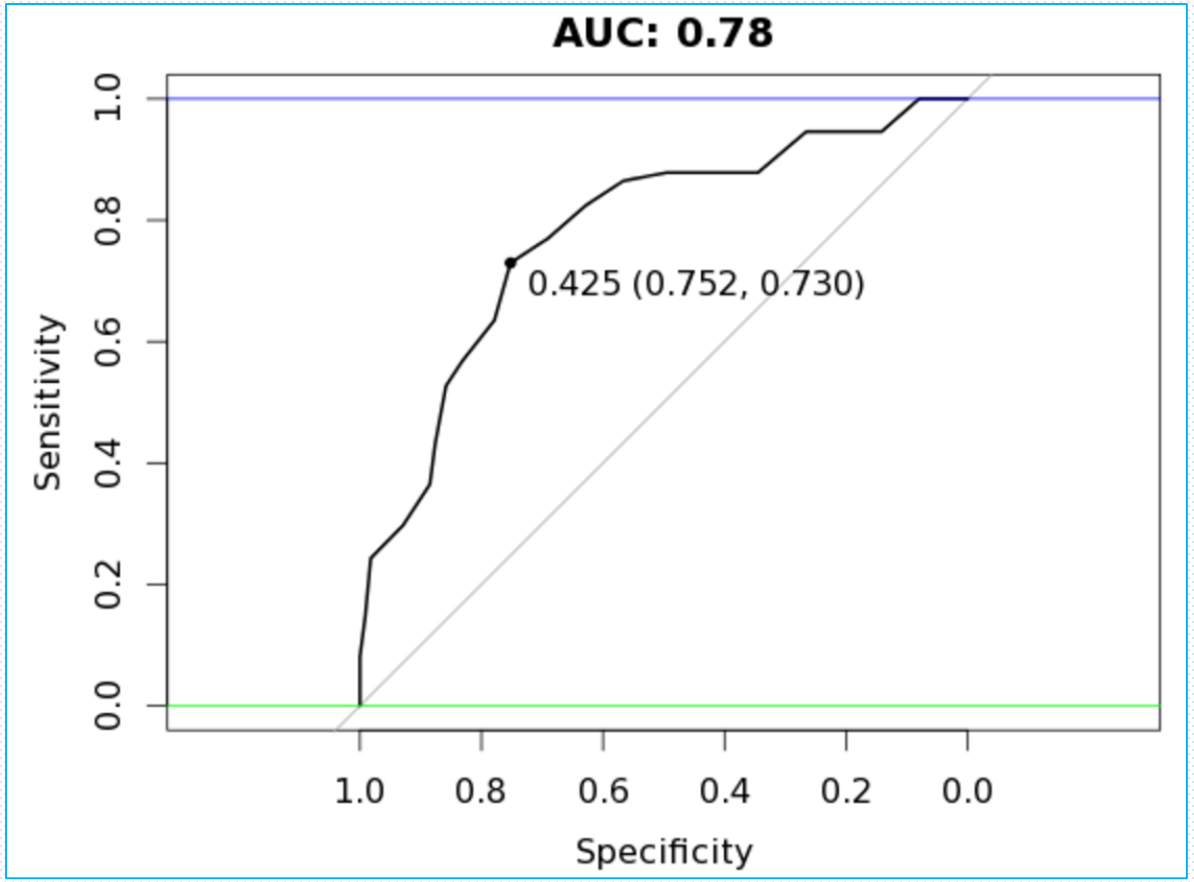

```
library(h2o)
localH2O = h2o.init()
prostate.hex<-as.h2o(prostate_df, destination_frame="train.hex")
prostate.dl = h2o.deeplearning(x = feature_names, training_frame = prostate.hex,
                               autoencoder = TRUE,
                               reproducible = T,
                               seed = 1234,
                               hidden = c(6,5,6), epochs = 50)

# interesting per feature error scores
 prostate.anon = h2o.anomaly(prostate.dl, prostate.hex, per_feature=TRUE)
 head(prostate.anon)
prostate.anon = h2o.anomaly(prostate.dl, prostate.hex, per_feature=FALSE)
head(prostate.anon)
err <- as.data.frame(prostate.anon)

# interesting reduced features (defaults to last hidden layer)
reduced_new  <- h2o.deepfeatures(prostate.dl, prostate.hex,layer=2)
head(reduced_new)
plot(sort(err$Reconstruction.MSE))
```

```
 DF.L2.C1   DF.L2.C2   DF.L2.C3 DF.L2.C4   DF.L2.C5
0.0000000 0.0000000 0.000000 0.0000000 0.04857907
0.4997092 1.0766817 1.600740 0.6755628 0.35198803
0.3719157 0.8298017 1.207336 0.4863282 0.03690014
0.0000000 0.0000000 0.000000 0.0000000 0.70372806
0.0000000 0.0000000 0.000000 0.0000000 0.37891354
0.4616853 1.0687052 1.568674 0.6392847 0.41436466
```

# Build RandomForest model: separate by reconstruction error

```
# rebuild train_df_auto with best observations

train_df_auto <- train_df[err$Reconstruction.MSE < 0.1,]

set.seed(1234)

rf_model <-
randomForest(x=train_df_auto[,feature_names],

                    y=as.factor(train_df_auto[,outcome_name]),

                     importance=TRUE, ntree=20, mtry = 3)

validate_predictions_known <- predict(rf_model,

        newdata=validate_df[,feature_names], type="prob")

auc_rf <-
roc(response=as.numeric(as.factor(validate_df[,outcome_
name]))-1, predictor=validate_predictions_known[,2])

plot(auc_rf, print.thres = "best",
main=paste('AUC:',round(auc_rf$auc[[1]],3)))

abline(h=1,col='blue')

abline(h=0,col='green')
```

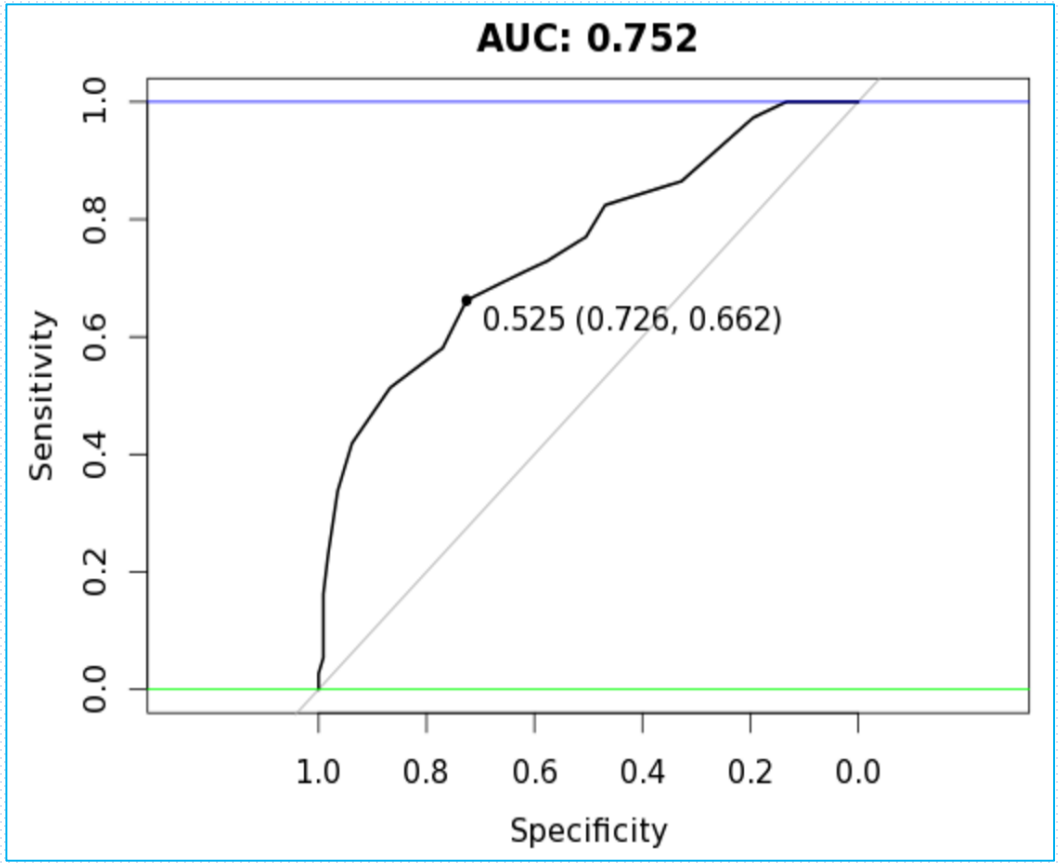
Reconstruction Error <0.1



# Build RandomForest model: separating by reconstruction error

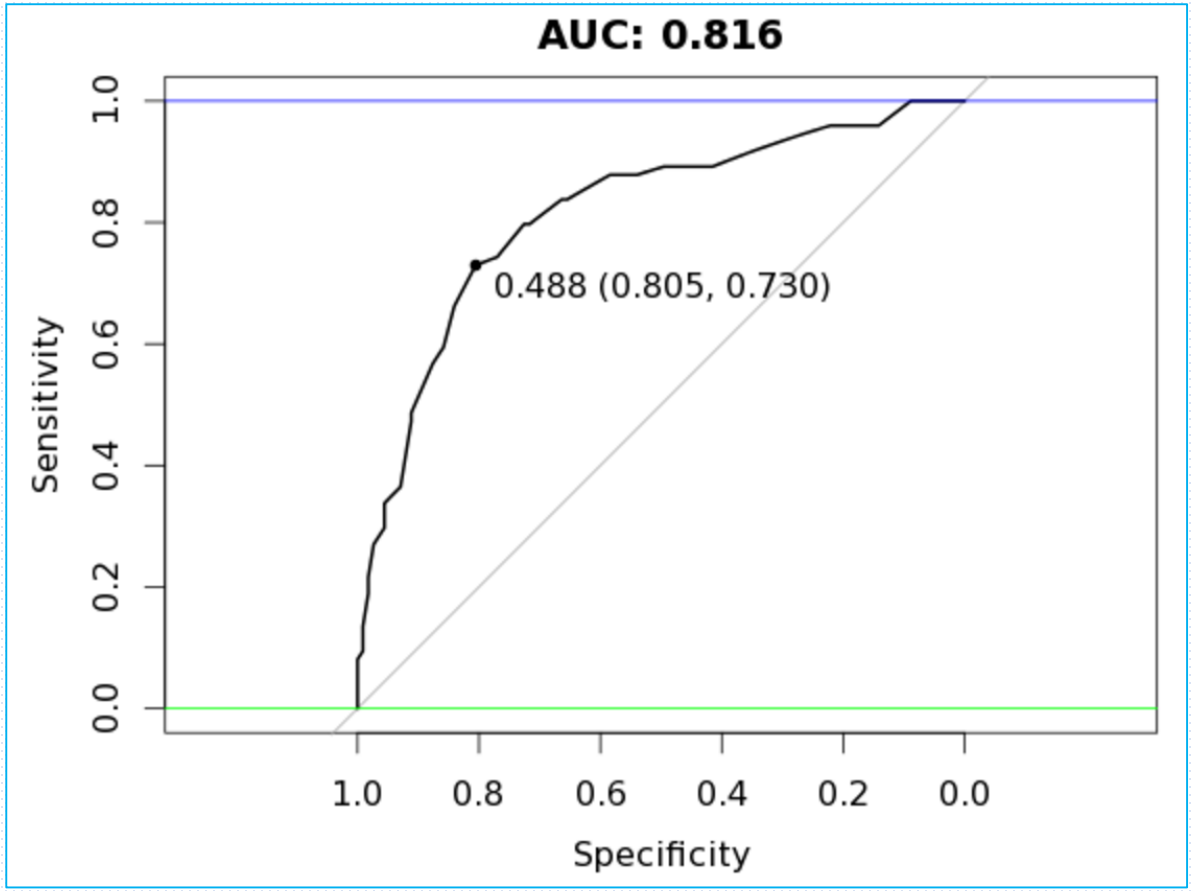
```
# rebuild train_df_auto with best observations

train_df_auto <- train_df[err$Reconstruction.MSE >= 0.1,]

set.seed(1234)

rf_model <- randomForest(x=train_df_auto[,feature_names],


y=as.factor(train_df_auto[,outcome_name]),

                                importance=TRUE, ntree=20, mtry = 3)



validate_predictions_unknown <- predict(rf_model,
newdata=validate_df[,feature_names], type="prob")

auc_rf =
roc(response=as.numeric(as.factor(validate_df[,outcome_name])
)-1,                predictor=validate_predictions_unknown[,2])

plot(auc_rf, print.thres = "best",
main=paste('AUC:',round(auc_rf$auc[[1]],3)))

abline(h=1,col='blue')

abline(h=0,col='green')
```

ReconstructionError >=0.1



# Bagging the results from both models

```
valid_all <- (validate_predictions_known[,2] +
validate_predictions_unknown[,2]) / 2


auc_rf =
roc(response=as.numeric(as.factor(validate_df[
,outcome_name]))-1,
            predictor=valid_all)


plot(auc_rf, print.thres = "best",
main=paste('AUC:',round(auc_rf$auc[[1]],3)))
abline(h=1,col='blue')
abline(h=0,col='green')
```



# Case Study 2:CNN Deep Learning for Cancer Immunotherapy

Peptide Classification Model

- The input data for this use case was created by generating 1,000,000 random 9-mer peptides by sampling the one-letter code for the 20 amino acids, i.e. ARNDCQEGHILKMFPSTWYV, and then submitting the peptides to MHCI binding prediction using the current state-of-the-art model netMHCpan.

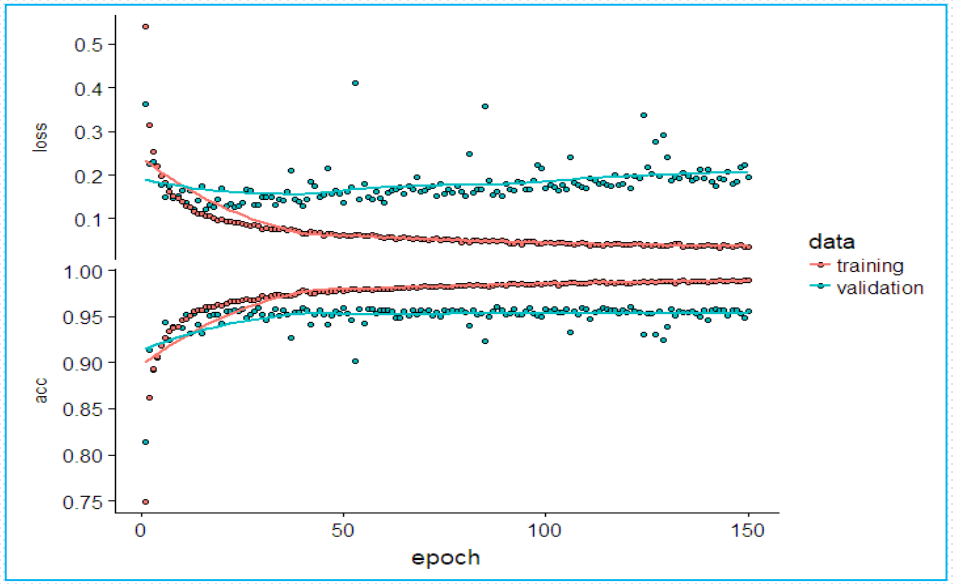
- For this use case, we applied three models to classify whether a given peptide is a ‘strong binder’ SB, ‘weak binder’ WB or ‘non-binder’ NB. to MHCI (Specific type: HLA-A*02:01). Thereby, the classification uncovers which peptides, will be presented to the T-cells.

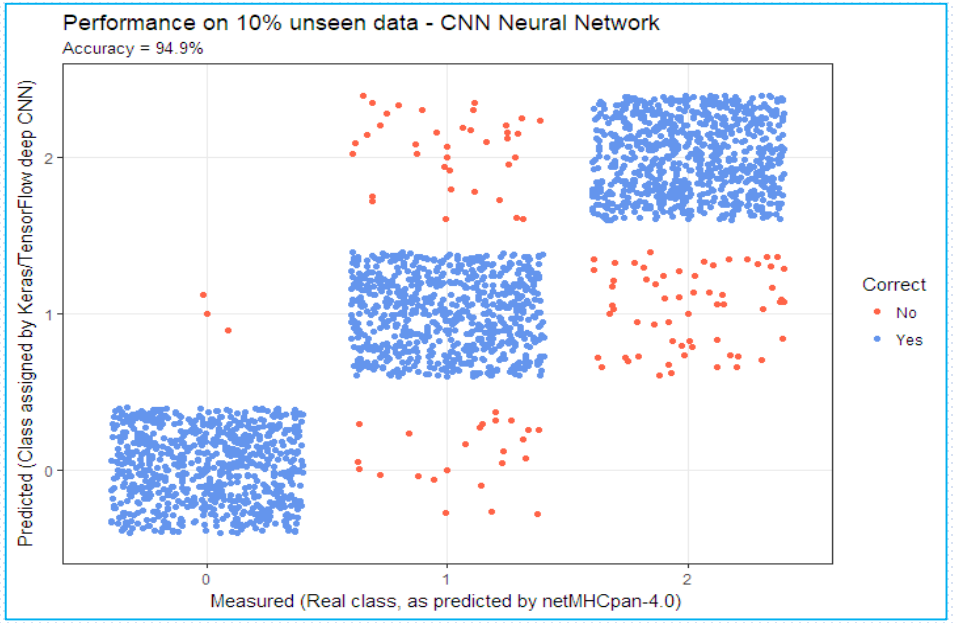
- Since n(SB) < n(WB) << n(NB), the data was subsequently balanced by down sampling, such that n(SB) = n(WB) = n(NB) = 7,920. Thus, a data set with a total of 23,760 data points was created. 10% of the data points were randomly assigned as test data and the remainder as train data. It should be noted that since the data set originates from a model, the outcome of this particular use case will be a model of a model. However, netMHCpan is very accurate (96.5% of natural ligands are identified at a very high specificity 98.5%).

- In the following each peptide will be encoded by assigning a vector of 20 values, where each value is the probability of the amino acid mutating into 1 of the 20 others as defined by the BLOSUM62 matrix using the pep_encode() function from the PepTools package. This way each peptide is converted to an ‘image’ matrix with 9 rows and 20 columns.

# Building a CNN model for Peptide classification

```
library(keras)

model <- keras_model_sequential() %>%
        layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu',
                      input_shape = c(9, 20, 1)) %>%
        layer_dropout(rate = 0.25) %>%
        layer_flatten() %>%
        layer_dense(units  = 180, activation = 'relu') %>%
        layer_dropout(rate = 0.4) %>%
        layer_dense(units  = 90, activation  = 'relu') %>%
        layer_dropout(rate = 0.3) %>%
        layer_dense(units  = 3, activation   = 'softmax')

model %>% compile(
        loss      = 'categorical_crossentropy',
        optimizer = optimizer_rmsprop(),
        metrics   = c('accuracy'))

history = model %>% fit(  x_train, y_train,
        epochs = 150,
        batch_size = 50,
        validation_split = 0.2)
```



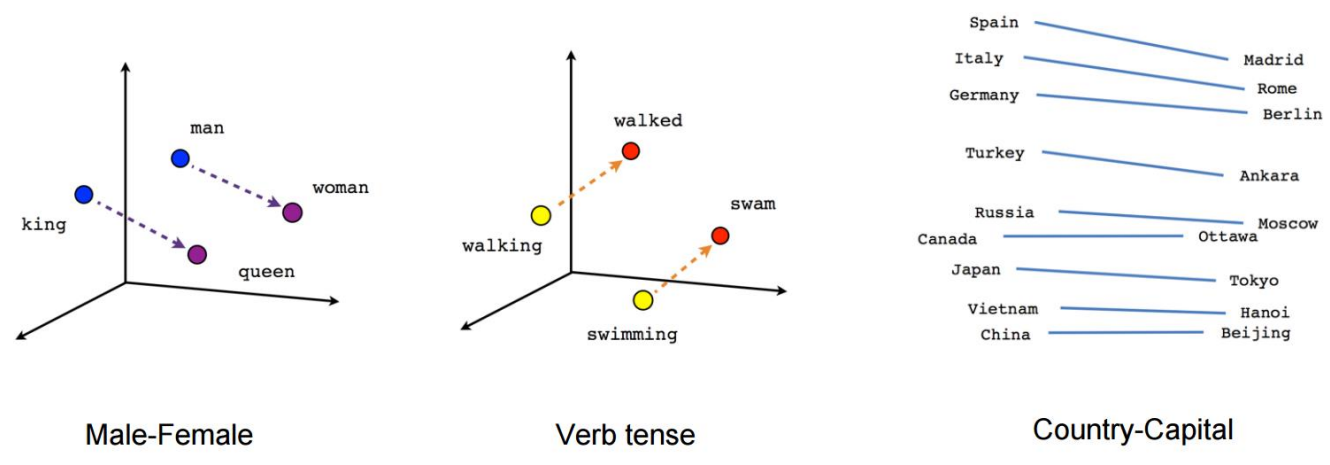



# Natural Language Programming and Deep Learning

Word2vec and Med2vec

# Word Embeddings and Vector models

- Word embedding is where words or phrases from the vocabulary are mapped to vectors of real numbers.
- Conceptually it involves a mathematical embedding from a space with one dimension per word to a continuous vector space with much lower dimension.

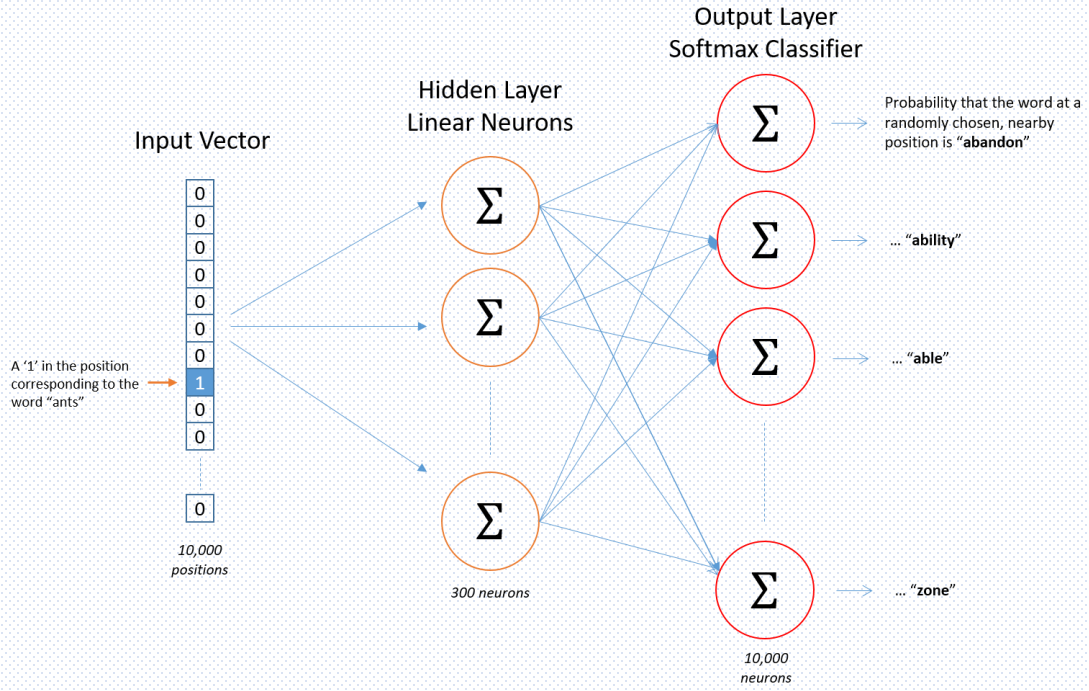


# Types of Vector Embedding Models

- Word2vec is a group of related models that are used to produce word embeddings.
- These models are shallow, two-layer neural networks that are trained to reconstruct linguistic contexts of words.
- Word2vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space.

- GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

- Facebook's FastText → https://github.com/facebookresearch/fastText
- Gensim → https://radimrehurek.com/gensim/tutorial.html



# Med2Vec for learning medical concepts

- Edward Choi etal. From GeorgiaTech teamed up Children Healthcare of Atlanta to create a 2-layer Word2Vec model which learn embeddings for medical codes (CPT, ICD codes) into a 200 dimensional vector space. The 2nd layer is an RNN that learns the temporal correlation between visits as well.

- The concepts learned can then be used in predicting:

  Clinical Risk Groups(CRG)

  and/or medical codes for future visits.

- Diagnosis representations learned through skip grams:
- http://mp2893.com/scatterplot/nnsg_h200e49_category10.html
- http://mp2893.com/scatterplot/skipgram.html
- http://mp2893.com/scatterplot/glove.html
- http://mp2893.com/scatterplot/gru_glove.html

# Language Generation models using LSTM

https://csaurav.shinyapps.io/SpeechPrediction/

https://github.com/saurav2608/speechPrediction

# References


Autoencoders

- https://wiseodd.github.io/techblog/2016/12/03/autoencoders/
- http://amunategui.github.io/anomaly-detection-h2o/
- https://blog.keras.io/building-autoencoders-in-keras.html

CNN/FFNN

- https://tensorflow.rstudio.com/blog/dl-for-cancer-immunotherapy.html

LSTM

- https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/

General

- https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

WordEmbedding and NLP

- https://nlp.stanford.edu/projects/glove/
- http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
- https://github.com/bmschmidt/wordVectors
- https://tensorflow.rstudio.com/blog/word-embeddings-with-keras.html
- https://github.com/mp2893/med2vec
- https://arxiv.org/pdf/1602.05568.pdf

Questions???

# PopHealthCare is Hiring!!!

Job Portal:

https://goo.gl/dZcsKX

LinkedIn:
https://www.linkedin.com/in/arran-hoek-3125215