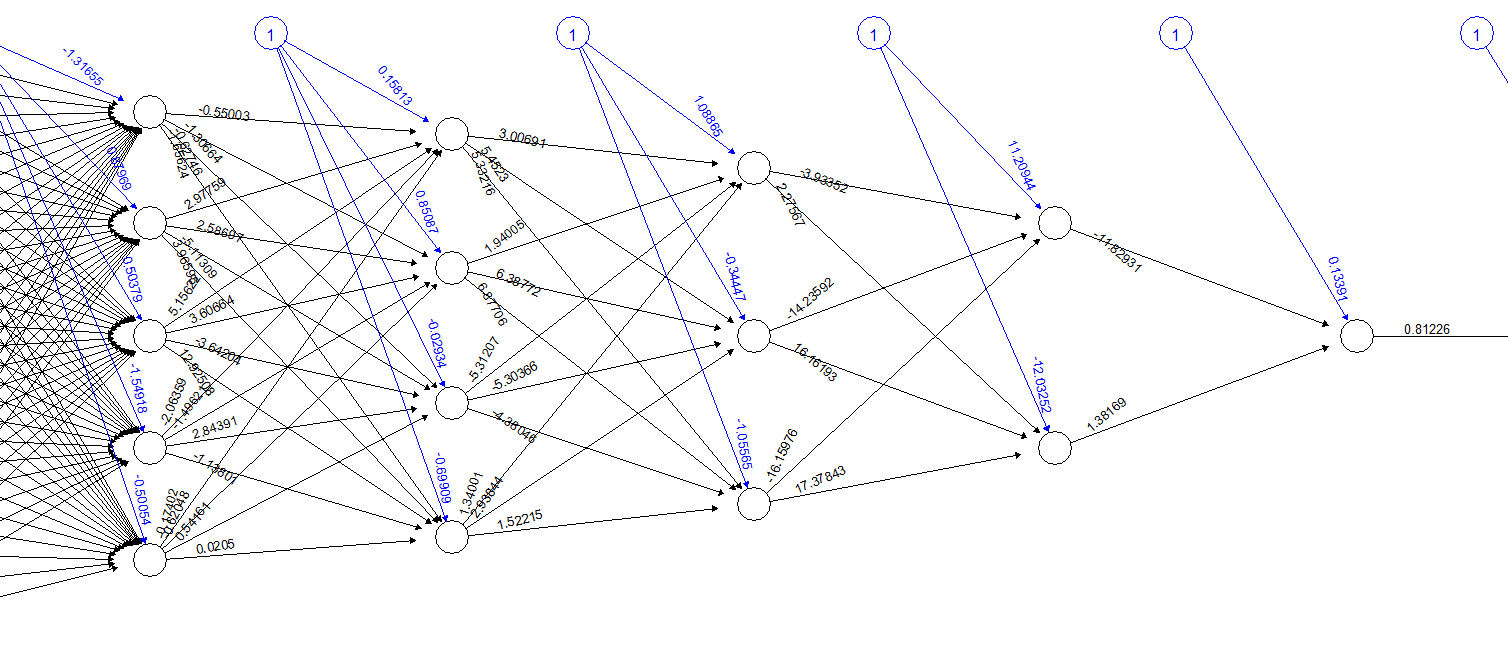
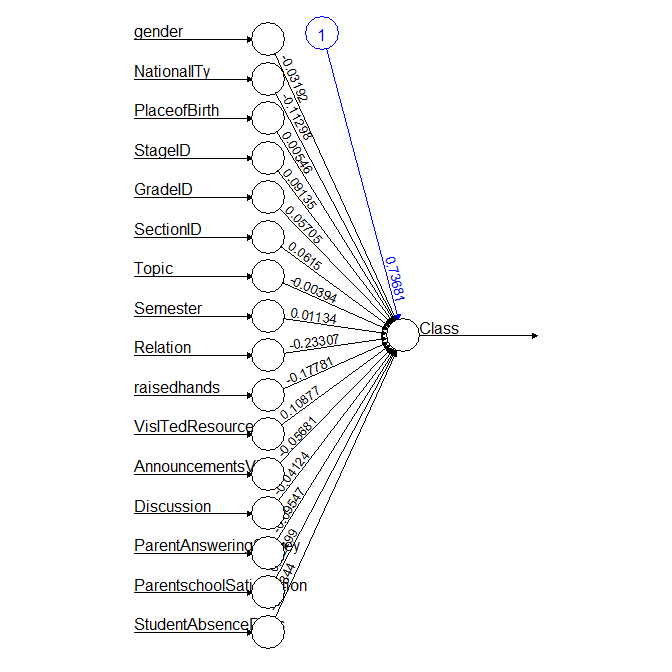
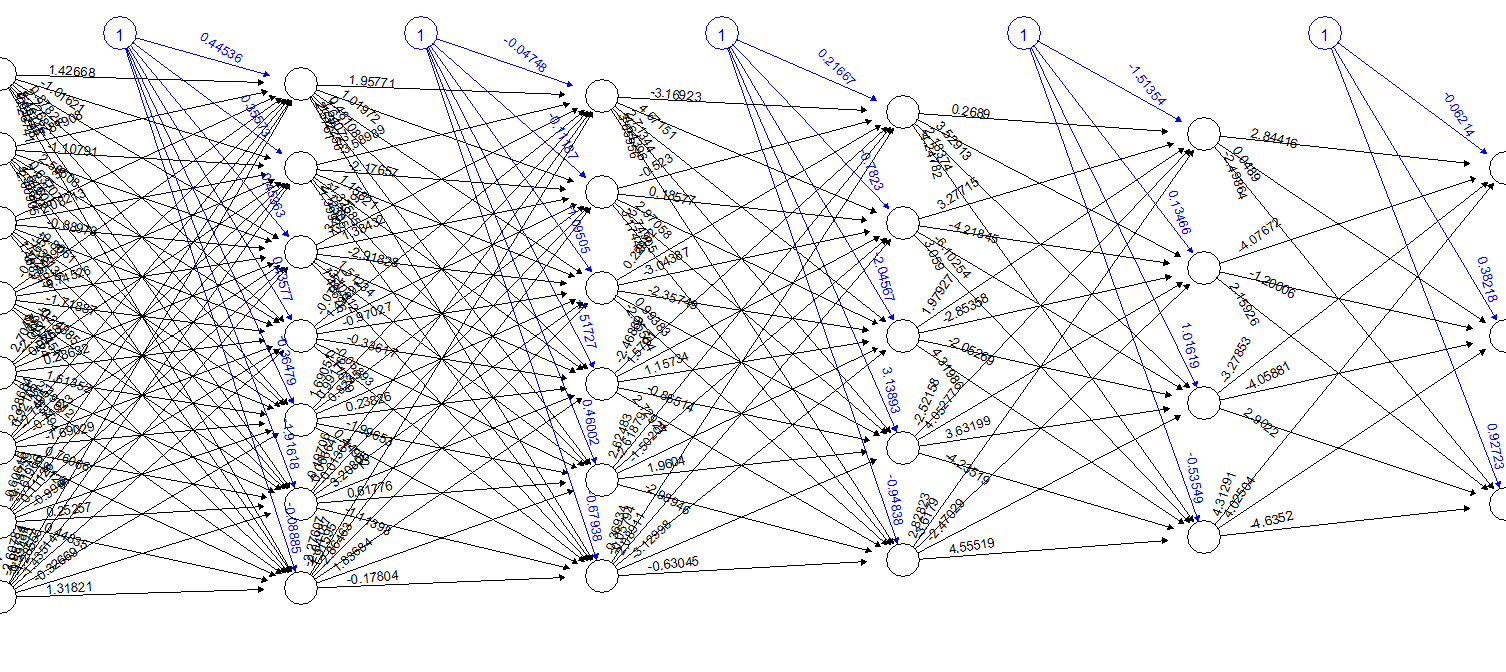
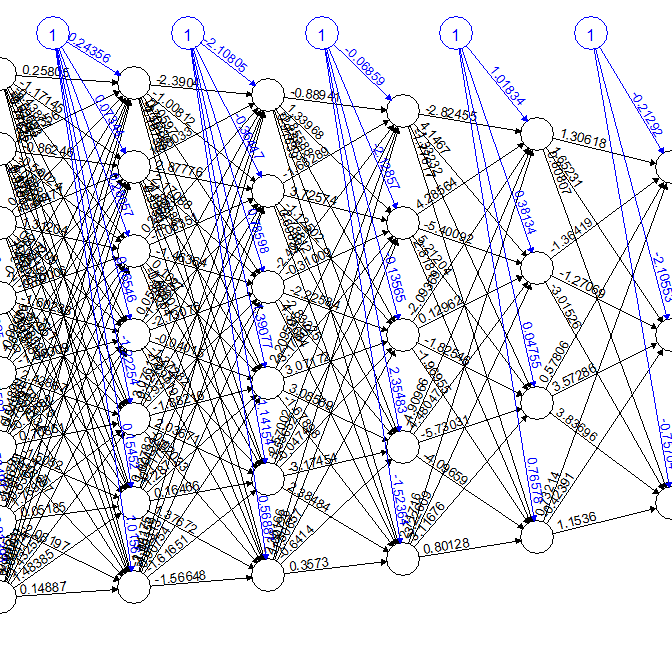
**Part I: Feed-Neural Network**



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model/Trial** | **#1** | **#2** | **#3** | **#4** | **#5** |
| **1** | 0.1661860669 | 0.16 | 0.16 | 0.16 | 0.16 |
| **2** | 0.200705277 | 0.31 | 0.36 | 0.33 | 0.23 |
| **3** | 0.16 | 0.242 | 0.249 | 0.287 | 0.274 |

More layers may result in better predictive data but that doesn’t seem to be the case from consistent basis standpoint, as shown above that the 5-layer neural network had higher correlation marks than the 10-layer. The magnitude of the dataset (parameters and length) favors a 5-layer neural network model.

Even with that said, more layers should and would incur higher correlation scores with more nodes and layers at the expense of higher time complexity. **Additional test runs on both 5- & 10-models** resulted that the 10-layer model scoring a correlation in the 45-46% percentile(10% probability, 1 / 10 run executions), as shown below:



The one-layer neuron model did not incur any changes from the lack of layers to further process and feed the data for learning.

The number of layers and nodes per layer will affect the time complexity. Judging by the eye-test, doing a backpropagation would be slower than a forward propagation, having to calculate and update nodes in previous layers.

**Part II: SVM**

2A. Linear

|  |
| --- |
| **Loaded excel file**  ksvm\_educ\_train <- educSVMNorm[1:336,]  ksvm\_educ\_test <- educSVMNorm[336:480,]  #training model with data kernel = linear  linear\_classifier <- ksvm(Class ~., data= ksvm\_educ\_train ,  kernel = "vanilladot")  #linear\_classifier  #check model performance  linear\_predictions <-predict(linear\_classifier, ksvm\_educ\_test)  #linear\_predictions  table(linear\_predictions, ksvm\_educ\_test$Class)  head(linear\_predictions)  # look only at agreement vs. non-agreement  # construct a vector of TRUE/FALSE indicating correct/incorrect predictions  agreement <- linear\_predictions == ksvm\_educ\_test$Class  table(agreement)  prop.table(table(agreement)) |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Linear** | **Rbf** | **poly** | **anova** | **laplace** | **Bessel** | **Tanhdot** |
| FALSE | .282 | 0.255% | 41 | **35** | 39 | 111 | **69** |
| TRUE | .717 | 0.744% | 104 | **110** | 106 | 34 | **76** |
| **OFV** | N/A | -345.75 | -8.8  -102.31  -67.38 | **-7.7**  **-88.16**  **-54.15** | -47  -145  -121 | -186  -140  -145 | **-48**  **-620**  **-409** |
| **Train err** | N/A | 0.976 | 0.1815 | **0.154** | 0.19 | 0.36 | **0.613** |

I utilized the kernels as the arguments would allow. Anova and tanhdot had the lowest and highest training error rates. From research, run-time complexity would be the same once the model has been trained at run-time respective of the kernel used.

I did not sense any execution that too long, likely because of the small dataset’s size. It is then presumed that time complexity will be higher on higher dimensions and higher data inputs. I would notice a time difference on those scenarios.

**SVM vs NN (Feed-forward variation)**

* Both are machine learning algorithms whose models are based on learning patterns formulated on the data fed.
* FF : Good for prediction or classification problems. Scalable and can handle higher dimensionality loads.
* Feed-forward networks are defined as a parametric model. SVM are non-parametric.

The hidden layers present in FF networks make it parameterized. The number of hidden layers would depend on the number of features (including bias parameters). Some features of FF make it a popular and powerful choice:

1. Backpropagation. There are input, hidden, and output neural layers. The algorithm calculates the neuron output per layer with the output making a prediction based on the differential between the actual and prediction error. Prediction error is then used to change the weights in neurons in all the previous layers. In other words, the neurons before, get a tune-up in hopes to better recognize the pattern concluded by the neurons ahead. This aims to improve overall prediction accuracy by the model.
2. The number of hidden layers and neurons per hidden layer also improve the network performance. This increase the network’s learning capacity as the lower layers can capture finer/lower-level details; same case also applies to middle-hidden (mid-level) and high-hidden (high-level), thus giving fine-grain tune performance.

NN has its disadvantages:

1. Requires a lot of data. Comparing the correlation and training error of NN and SVM, if comparable at all, SVM incurred a better result. Need more data for NN to make a difference.
2. Neural networks are best utilized with GPUs as matrix computations done there are blazing fast compared to multi-core processors. Consider other machine learning methods.
3. Require spatial property for convolution operation. Same reasoning in number two as means. The operations perform a task on a set of pixels/grouped data that are proximal to produce the ordering of features on that data. SVM are unaffected by this.
4. Parameterization complexity. NN will need to take account as much of that.
5. Non-trivial results inside. Intermediary results can be challenging to interpret/comprehend as it gets closer to the output.

SVM is equivalent to FF’s with a non-linear activation function. SVM is made of support vectors from a training set. Each vector has its own weight. This eventually creates hyperplanes to separate and classify data clusters. A hyperplane is a boundary and has margin spaces between it and the different support vectors.

**SVM employs kernel tricks and maximal margin concepts between classes to perform better in non-linear and high-dimensional tasks, making it better to generalize the differences**. A kernel trick involves mapping and transforming a non-linear space into a linear one for ease of computation. Without the kernel, the SVM is limited to linear pattern recognition. Outliers [outside of support vectors of interest] affect SVM however; they become support vectors and skew the data. Even a powerful SVM model, most of the times, would thus benefit from the proper feature selection and feature extraction/transformation techniques.

**Complexity**

**SVM**

* SVM: Number of examples (data), features, kernel function, regularization parameter. At its simplest form, running time f(x) for a trained SVM instance (binary) will be :

|  |  |
| --- | --- |
| **Kernel Name** | **Kernel Function** |
| Linear |  |
| Polynomial |  |
| RBF |  |

Linear kernel complexity will have O(*n*) complexity, n as number of inputs. We are just solving for a collection of single dot products.

Predicting linear SVM models strictly requires the hyperplane in input space. However, using non-linear kernels is different as calculating the hyperplane in its feature space is near impossible. Much of the reason being it’s computationally expensive to do so. As an alternative, the inner products between the hyperplane and test points in feature space can be computed using the kernel evaluations between support vectors and the test points. This is the *kernel trick.*

All things aside, the run time complexity for a SVM + kernel function would be

*P = number of support vectors*

**Neural Network**

NN is architecture-dependent but will likely have complexity usually above a linear kernel SVM. T

Normalization helps that it scales the data we have, removes bias in the model and makes converges faster during backpropagation. With the strategies to achieve better performance, getting time complexity for neural networks again still varies a lot depending on the type of network used. The time complexity will however be proportional to the number of nodes (activated) and among others.

**Conclusion**

**SVM** uses hyperplanes (marked gap space between support vectors, which are extreme datapoints) to separate clusters between identified data clusters. It is ideal for binary classification of X verses other variables, as well as various decision functions for the appropriate kernel function use, especially when dealing high dimensional spaces of *r* to the power of *n* samples. It performs poorly when the number of features are greater than the number of samples, exhausting power to produce a difference in clustering data. It doesn’t do well on predictive analyses. News, categorization are good use cases for SVM.

**FF** networks are best used in predictive analyses with sequenced Big Data that contain many features, the things the SVM isn’t ideal for. Image analysis, language processing are good use cases. This needs considerable set-up and huge amount of data to make something worthwhile in using neural networks.

|  |  |
| --- | --- |
| > library("kernlab", lib.loc="~/R/win-library/3.5")  > ksvm\_educ\_train <- educSVMNorm[1:336,]  > ksvm\_educ\_test <- educSVMNorm[336:480,]  > linear\_classifier <- ksvm(Class ~., data= ksvm\_educ\_train ,  + kernel = "vanilladot")  Setting default kernel parameters  > linear\_predictions <-predict(linear\_classifier, ksvm\_educ\_test)  > table(linear\_predictions, ksvm\_educ\_test$Class)    linear\_predictions H L M  H 40 0 22  L 0 23 1  M 12 6 41  > agreement <- linear\_predictions == ksvm\_educ\_test$Class  > table(agreement)  agreement  FALSE TRUE  41 104  > prop.table(table(agreement))  agreement  FALSE TRUE  0.2827586 0.7172414 | **Linear** |
| > rbf\_classifier  Support Vector Machine object of class "ksvm"  SV type: C-svc (classification)  parameter : cost C = 1  Gaussian Radial Basis kernel function.  Hyperparameter : sigma = 0.0391624817897649  Number of Support Vectors : 266  Objective Function Value : -22.6847 -107.2832 -83.5092  Training error : 0.154762  > #check rbf model performance  > rbf\_predictions <- predict(rbf\_classifier, ksvm\_educ\_test)  > table(rbf\_predictions,ksvm\_educ\_test$Class)    rbf\_predictions H L M  H 33 0 14  L 0 25 0  M 19 4 50  > #look vs agreement vs non-agreement  > #product boolean vector for predicting class  > agreement\_rbf <- rbf\_predictions == ksvm\_educ\_test$Class  > View(table(agreement\_rbf))  > prop.table(table(agreement\_rbf))  agreement\_rbf  FALSE TRUE  0.2551724 0.7448276 | Rbfdot |
| > poly\_classifier <- ksvm(Class ~., data= ksvm\_educ\_train,  + kernel = "polydot")  Setting default kernel parameters  > poly\_classifier  Support Vector Machine object of class "ksvm"  SV type: C-svc (classification)  parameter : cost C = 1  Polynomial kernel function.  Hyperparameters : degree = 1 scale = 1 offset = 1  Number of Support Vectors : 186  Objective Function Value : -8.8048 -102.3159 -67.3866  Training error : 0.181548  > #check poly model performance  > poly\_predictions <- predict(poly\_classifier, ksvm\_educ\_test)  > table(poly\_predictions, ksvm\_educ\_test$Class)    poly\_predictions H L M  H 40 0 22  L 0 23 1  M 12 6 41  > #look vs agreement vs non-agreement  > #product boolean vector for predicting class  > agreement\_poly <- poly\_predictions == ksvm\_educ\_test$Class  > View(table(agreement\_poly))  > prop.table(table(agreement\_poly))  agreement\_poly  FALSE TRUE  0.2827586 0.7172414 | poly |
| > anovadot\_classifier  Support Vector Machine object of class "ksvm"  SV type: C-svc (classification)  parameter : cost C = 1  Anova RBF kernel function.  Hyperparameter : sigma = 1 degree = 1  Number of Support Vectors : 191  Objective Function Value : -7.7046 -88.1623 -54.1542  Training error : 0.154762  > #check anovadot model performance  > anovadot\_predictions <- predict(anovadot\_classifier, ksvm\_educ\_test)  > #look vs agreement vs non-agreement  > #product boolean vector for predicting class  > agreement\_anovadot <- anovadot\_predictions == ksvm\_educ\_test$Class  > View(table(agreement\_anovadot))  > prop.table(table(agreement\_anovadot))  agreement\_anovadot  FALSE TRUE  0.2413793 0.7586207 | anovadot |
| > laplacedot\_classifier <- ksvm(Class ~., data= ksvm\_educ\_train,  + kernel = "laplacedot")  > laplacedot\_classifier  Support Vector Machine object of class "ksvm"  SV type: C-svc (classification)  parameter : cost C = 1  Laplace kernel function.  Hyperparameter : sigma = 0.0416653267021771  Number of Support Vectors : 312  Objective Function Value : -45.526 -142.7715 -118.6806  Training error : 0.184524  > #check laplacedot model performance  > laplacedot\_predictions <- predict(laplacedot\_classifier, ksvm\_educ\_test)  > #look vs agreement vs non-agreement  > #product boolean vector for predicting class  > agreement\_laplacedot <- laplacedot\_predictions == ksvm\_educ\_test$Class  > View(table(agreement\_laplacedot ))  > prop.table(table(agreement\_laplacedot ))  agreement\_laplacedot  FALSE TRUE  0.2689655 0.7310345 | laplacedot |
| > besseldot\_classifier <- ksvm(Class ~., data= ksvm\_educ\_train,  + kernel = "besseldot")  Setting default kernel parameters  > besseldot\_classifier  Support Vector Machine object of class "ksvm"  SV type: C-svc (classification)  parameter : cost C = 1  Bessel kernel function.  Hyperparameter : sigma = 1 order = 1 degree = 1  Number of Support Vectors : 294  Objective Function Value : -186.4177 -140.8432 -145.2793  Training error : 0.363095  > #check besseldot model performance  > besseldot\_predictions <- predict(besseldot\_classifier, ksvm\_educ\_test)  > #look vs agreement vs non-agreement  > #product boolean vector for predicting class  > agreement\_besseldot <- besseldot\_predictions == ksvm\_educ\_test$Class  > View(table(agreement\_besseldot))  > prop.table(table(agreement\_besseldot))  agreement\_besseldot  FALSE TRUE  0.7655172 0.2344828 | besseldot |
| > tanhdot\_classifier <- ksvm(Class ~., data= ksvm\_educ\_train,  + kernel = "tanhdot")  Setting default kernel parameters  > tanhdot\_classifier  Support Vector Machine object of class "ksvm"  SV type: C-svc (classification)  parameter : cost C = 1  Hyperbolic Tangent kernel function.  Hyperparameters : scale = 1 offset = 1  Number of Support Vectors : 221  Objective Function Value : -48.7944 -620.8788 -409.6121  Training error : 0.613095  > #check tanhdot model performance  > tanhdot\_predictions <- predict(tanhdot\_classifier, ksvm\_educ\_test)  > #look vs agreement vs non-agreement  > #product boolean vector for predicting class  > agreement\_tanhdot <- tanhdot\_predictions == ksvm\_educ\_test$Class  > View(table(agreement\_tanhdot))  > prop.table(table(agreement\_tanhdot))  agreement\_tanhdot  FALSE TRUE  0.4758621 0.5241379 | tanhdot |