Stars, Galaxies or Quasars – Classifying New Stellar Observations

Date of Submission: Dec 6th, 2022

Authors: Grigorii Bulava, Steven Cheng, Chang Liu, Harsh Shekhar

Course: STAT 441 Classification

Instructor: Yeying Zhu

Abstract

The goal of this project is to develop a model to classify new stellar object observations accurately. We decided to use data on properties related to spectroscopy and discovery order of stellar bodies. Specifically, the properties are Right Ascension Angle, Declination Angle, Red Light and Green Light emissions, Near Infrared, Infrared and Ultraviolet radiation emissions, Redshift, Modified Julian Date, Unique ID and Plate Number. After performing exploratory data analysis and feature selection, we used KNN, Random Forest, and single and multi-layer Neural Networks to classify stellar objects. We found Neural Networks and Random forest achieved approximately 97% accuracy while KNN achieved only around 94% accuracy.

Introduction

Astrophotography is a rapidly advancing field. From blurry photos of Neptune in the beginning of 2000s (NASA, 2022), NASA went on to obtain black hole images in the year of 2019 (NASA, 2019). On the other hand, despite all the advances in the field of astrophotography, it still took 19.3 million dollars and over 5 years to photograph the black hole, so astrophotography is still far from being cost-effective or practical on a large scale. Moreover, NASA estimates that there are at least 100 billion stars in our galaxy, the Milky Way (NASA, 2015), and there are over 2 trillion (NASA, 2016) galaxies and 12.8 million quasars in the observable universe (NASA, 2022). If humanity wants to map the entire sky, it is of the utmost importance to develop methods that could assign a class to a stellar object accurately, quickly, and economically.

Problem of Interest

In this study we are working with three stellar bodies: Stars, Galaxies, and Quasars – the last one being a combination of a black hole and gas that spirals around this black hole at incredible speeds, creating very strong luminosity (NASA, 2015). The goal is to train a model to identify objects' classes accurately. Our target accuracy rate is 95% since every percentage of error means possible misclassification for millions of objects. In this study we want to explore astrophotography data for stellar objects that was obtained quickly and economically, and to then train a model that can assign the stellar objects' class accurately. We removed arbitrary variables such as run_ID that were not related to our study from the dataset. Below are eleven explanatory variables that might provide useful information with the classification of an astronomical body:

Right Ascension Angle (alpha) – object's angular distance from the sun at Vernal Equinox (daytime equal to nighttime).

Declination angle (delta) – angle of the object relative to the Celsius Equator during the Vernal Equinox (daytime equal nighttime).

Green Filter (g) and **Red Filter (r)** – object's green and red light wave measurements (photometric system)

Near infrared Filter (i) – measure of the object's radiation that's near to the infrared light in the photometric system

Redshift – object's increase in wavelength based on distance, respective frequency change

Ultraviolet Filter (u) – measure of the object's UV Radiation strength in the photometric system. (3)

Infrared Filter (z) - measure of the object's infrared light in the photometric system

MJD – Modified Julian Date, indicates when a piece of SDSS data was taken.

Unique ID (spec_obj_id) – object's unique ID.

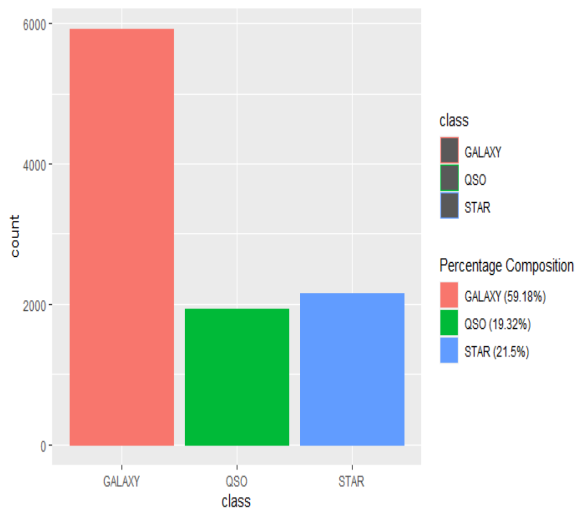
Plate – ID of the object's plate, in SDSS

We only included variables that are scientifically relevant to our statistical analysis. For example, we know Infrared radiation emitted by quasars and stars is drastically different (Xie, Ho, Zhuang, & Shangguan, 2021). Generally, we expect distinct types of stellar bodies' emissions to have different wave composition - and this is the reasoning behind the variables *g,r,i,z* and *u* in our model. Some things are included as a measure of observation control: *Right Ascension Angle* and *Declination Angle* represent the object's position in the sky relative to Earth, and variables *MJD*, *Plate* and *Unique ID* relate to the order and date of the object's classification. Finally, there is the variable *Redshift*, which is a measurement of how an object's wavelength increases with distance (NASA, 2022).

Methodology

To approach this problem, we first take a look at our data by conducting an initial exploratory data analysis (EDA). Then, we utilized Best Subset Selection and LASSO to select the variables of importance for our classification model. Based on our EDA results, we determined that K Nearest Neighbors, Random Forest and Neural Network are the best techniques to classify our data. We fitted models using the three approaches and arrived at our final model with the highest accuracy.

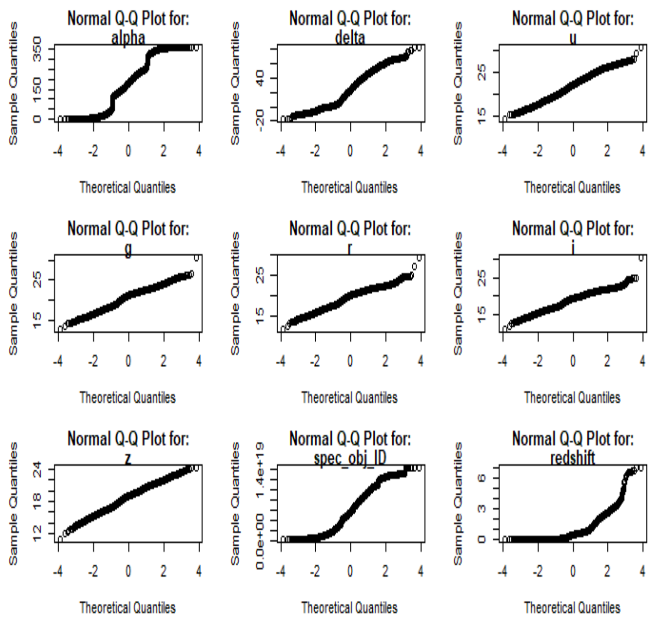
Exploratory Data Analysis



In Figure 1, we look at the distribution of object classes. We observe that in our data, approximately 60% of the observations are of Galaxies, and the others are of Quasar (QSO) and Stars each compromising 20% of our data.

In Figure 2, we look at the Normal QQ plot for all numeric variables and we observe that all the variables apart from alpha, delta, spec_obj_id, and redshift are normally distributed. alpha, delta, and spec_obj_id are uniformly distributed whereas redshift has an exponential distribution. There also does not seem to be a significant skew or kurtosis in the normally distributed variables.

Figure 1: Normal qqplot of Numerical Variables



Looking at the correlation heatmap of the numerical variables in Figure 3, apart from the photometric variables, other variables such as alpha, delta, spec_obj_id, and redshift do not seem to be highly correlated to other variables.

Figure 3: Correlation Heatmap of Numeric Variables

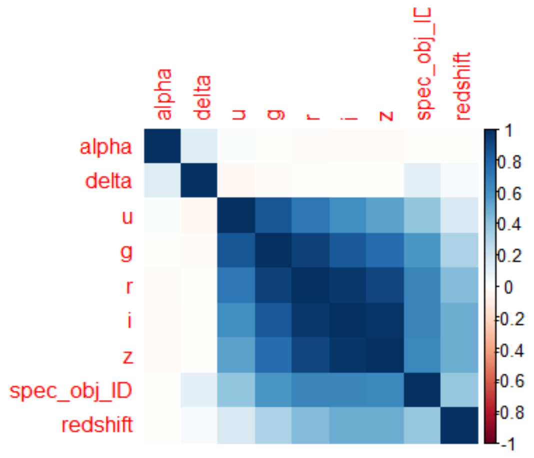


Figure 2: Normal QQ-plot of Numerical Variables

In Figure 4, we look closer at the correlation of the numerical variables. We observe that alpha is trimodal which may be related to each of the three classes. We also observe that the photometric variables do indeed have high correlation with each other. Most statistical practices would call for the removal of these variables since multicollinearity usually indicates poor quality data. It is, however, important to acknowledge that emissions of Ultraviolet, Green light, Red light, Near-infrared and Infrared emissions all follow the rule of wavelength and frequency being inversely proportional to each other (Qualitative Reasoning Group, Northwestern University, 2022). Therefore, the object's frequency will equally affect the wavelength of every variable mentioned above, creating some seemingly very strong correlation and providing a reason why it might be worth including these variables into our models.

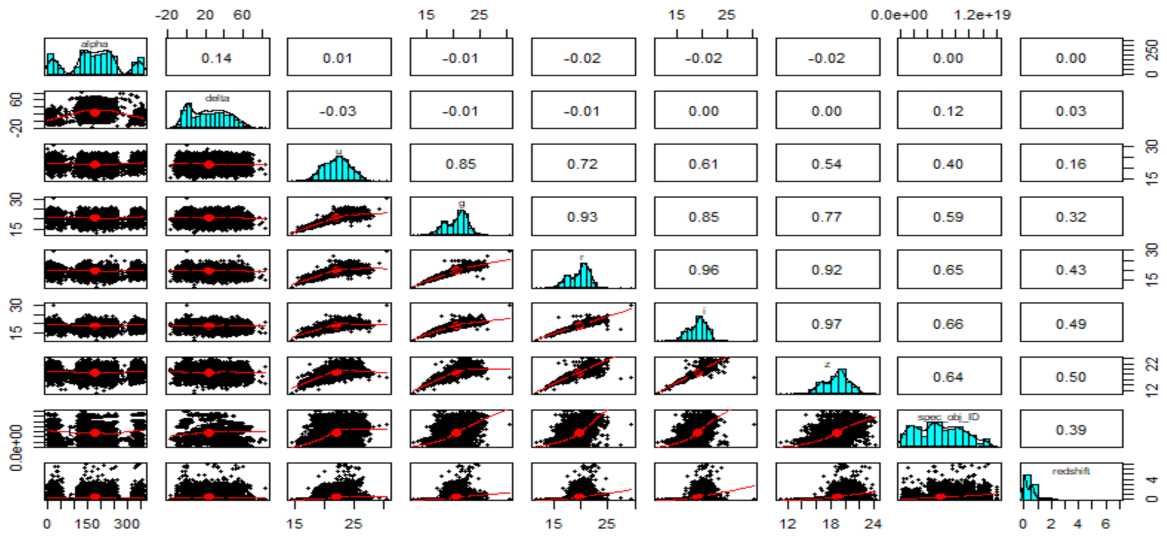


Figure 4: Correlation Plot for Numerical Variables

In figure 5, we graph the scatter plot for all variables against redshift color coded by class and we find that the trimodal distribution of alpha is not indicative of class. We also observe that plotting any variable against redshift coloured by class, there is an obvious separation amongst classes in these variables. We have plotted redshift against three variables here, alpha, u, and spec_obj_id, but the results are similar for every other variable too.

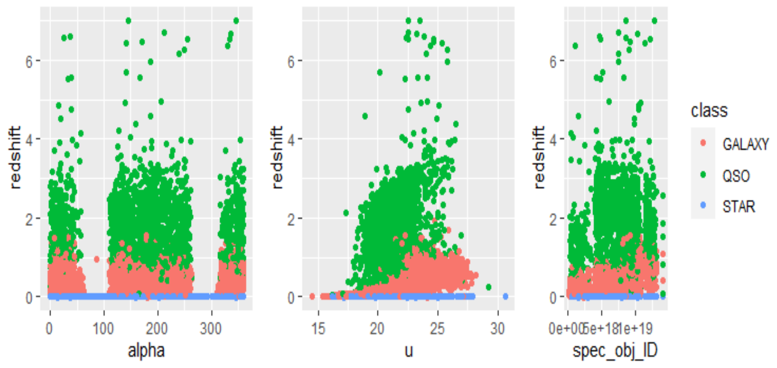


Figure 5: Scatterplot of {alpha, u, spec_obj_id} vs redshift by class

Looking at the Kernel Density Estimate of *ln(redshift)* in Figure 7, we observe that the values are different for different classes. We understand that the objects further away from us, the more red shifted their wavelengths will be (The European Space Agency, 2022). Quasars are the furthest away discovered objects till now (University of Oregon, 2022). We have observed a lot of stars around our galaxy. Then there are galaxies further away from stars but closer than quasars. The understanding of these distances combined with the working of redshift clearly explains our observation in Figure 7. This also means that redshift may be an important explanatory variable in our classification model.

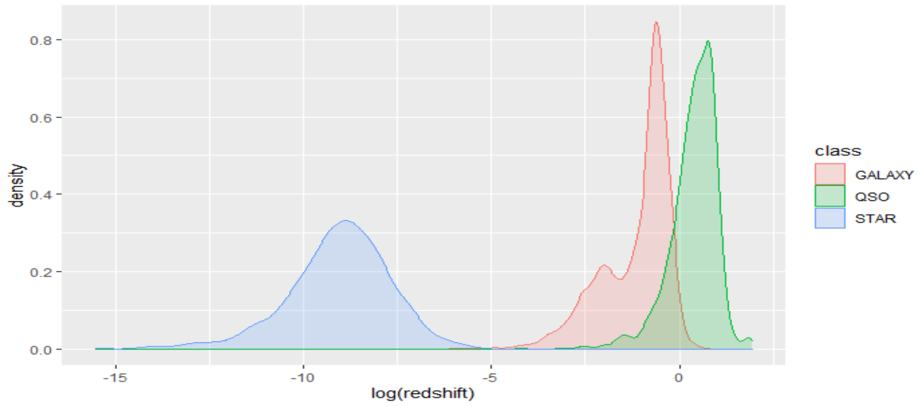


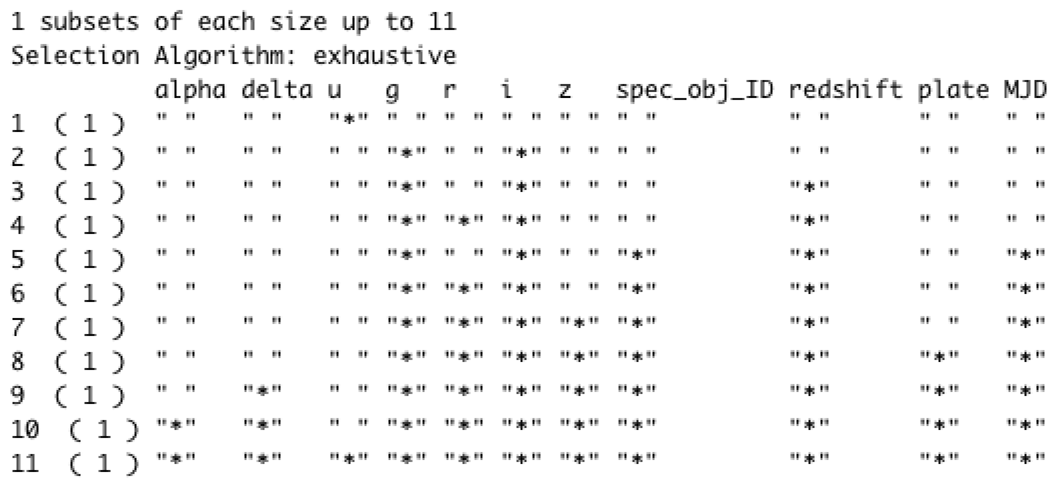
Figure 6: Kernel Density Estimate of *ln(redshift)* by Class

Feature Selection

After cleaning the data and removing unrelated predictors, there are 11 covariates remaining. However, considering the 10,000 data points we have, model training will be computationally expensive. To reduce the time needed for model training and include only essential covariates in our model, we used Best Subset selection and LASSO to further reduce the feature space.

Best subset selection

After we performed best subset selection, we utilized BIC, Mallow's CP, RSS, R^2 and adjusted R^2 value to evaluate our model and determine the most suitable feature space. BIC chose the a subset with 7 variables, Mallow's CP and RSS chose the subset with 11 variables, while adjusted R^2 and R^2 chose the subset with 1 variable. We decided not to follow adjusted R^2 and R^2 because these two methods chose a model with only one parameter and an underspecified model would produce biased results. Using the Law of Parsimony, we decided to use the subset feature space obtained from BIC because it chose a model with 7 variables.



```
1 subsets of each size up to 11
Selection Algorithm: exhaustive
          alpha delta u   g   r   i   z   spec_obj_ID redshift plate MJD
1  ( 1 )  " "   " "   "*" " " " " " " " " " "         " "      " "   " "
2  ( 1 )  " "   " "   " " "*" " " "*" " " " "         " "      " "   " "
3  ( 1 )  " "   " "   " " "*" " " "*" " " " "         "*"      " "   " "
4  ( 1 )  " "   " "   " " "*" "*" "*" " " " "         "*"      " "   " "
5  ( 1 )  " "   " "   " " "*" " " "*" " " "*"         "*"      " "   "*"
6  ( 1 )  " "   " "   " " "*" "*" "*" " " "*"         "*"      " "   "*"
7  ( 1 )  " "   " "   " " "*" "*" "*" "*" "*"         "*"      " "   "*"
8  ( 1 )  " "   " "   " " "*" "*" "*" "*" "*"         "*"      "*"   "*"
9  ( 1 )  " "   "*"   " " "*" "*" "*" "*" "*"         "*"      "*"   "*"
10  ( 1 ) "*"   "*"   " " "*" "*" "*" "*" "*"         "*"      "*"   "*"
11  ( 1 ) "*"   "*"   "*" "*" "*" "*" "*" "*"         "*"      "*"   "*"
```

Figure 7: Best Subset Selection Result

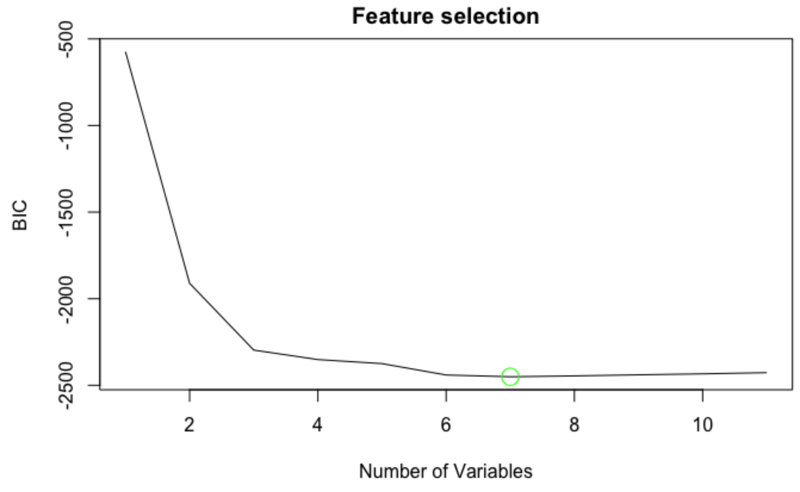


Figure 8: Best Subset Selection using BIC

LASSO

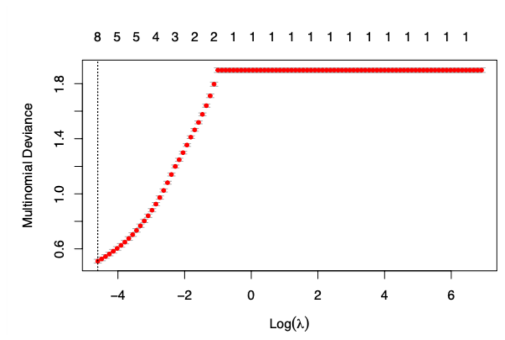


Figure 9: Multinomial Deviance vs Log(λ)

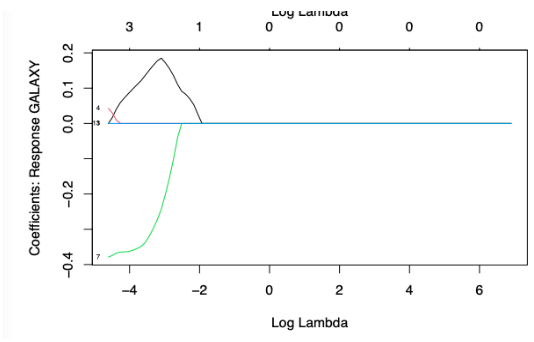
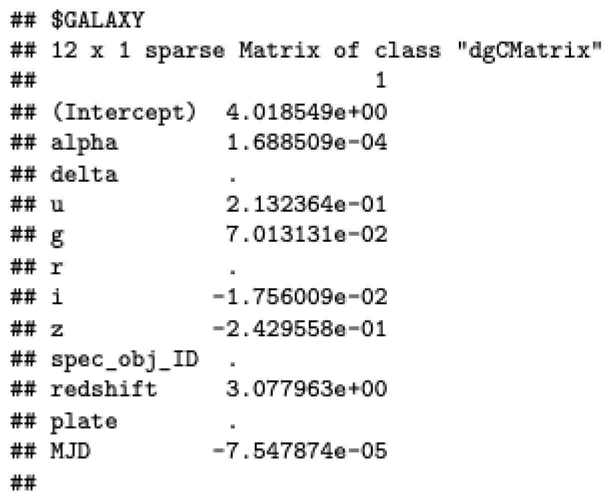


Figure 10: Coefficient Response vs Log(λ) for Galaxy

LASSO selected 7 variables as observed in Figure 11. During EDA (Figure 4), we found that the photometric variables are highly correlated. LASSO does not do well with group selection because LASSO tends to choose a variable arbitrarily out of the groups of correlated variables rather than selecting the best variable. In addition, our goal is to infer new astronomical observations and predict its category based on existing data we have. Best subset selection usually results in models with better prediction accuracy, and LASSO is not as suitable for inference. Therefore, we decided to use the feature space selected with the best subset selection.



```
## $GALAXY
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept)  4.018549e+00
## alpha        1.688509e-04
## delta        .
## u            2.132364e-01
## g            7.013131e-02
## r            .
## i           -1.756009e-02
## z           -2.429558e-01
## spec_obj_ID  .
## redshift     3.077963e+00
## plate        .
## MJD         -7.547874e-05
##
```

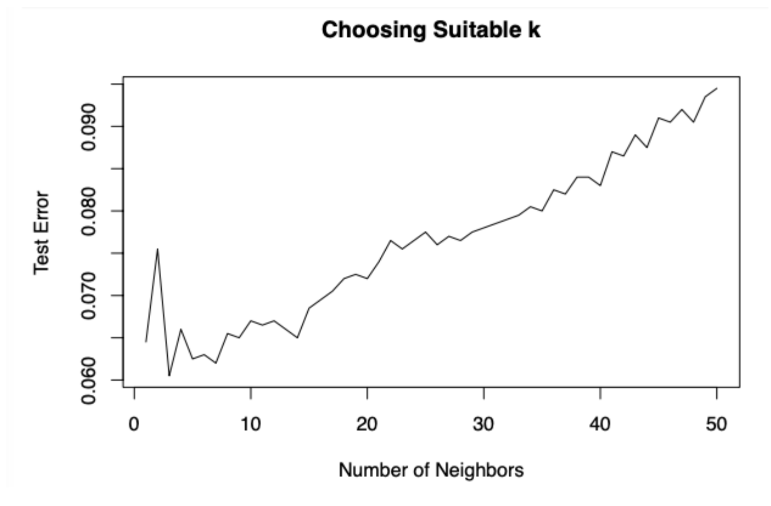
Figure 11. Variables selected by LASSO

Model Fitting

From the exploratory data analysis, we found that the underlying distribution of our data is likely not normal (Figure 2). Therefore, LDA and QDA are unsuitable for our data. Furthermore, we found each covariate has correlation with one another. Since similar astronomical bodies have similar properties and our covariates are all astronomical measurements, our statistical results correspond to the physical situation.

Considering the non-normality of our data and the correlation between variables from the EDA section, we are fitting KNN, Random Forest, and Neural Nets to our data because these models are nonparametric and make no assumption about our underlying model.

KNN

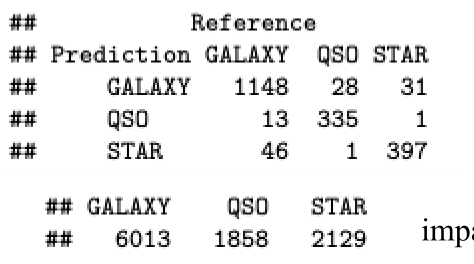


We first choose a suitable number of neighbors for our KNN model.

From the graph here, we can observe the number of neighbors corresponding to the smallest test error is *k = 3*. Therefore, we are fitting a 3NN to our data.

Figure 12: Shrinkage vs Error of Val and Test

From here, we can see that KNN performed quite well on our testing set with the majority of predictions being accurate. KNN has a 94.2% accuracy.



```
##             Reference
## Prediction GALAXY  QSO STAR
##     GALAXY   1148   28   31
##     QSO        13  335    1
##     STAR       46    1  397
```

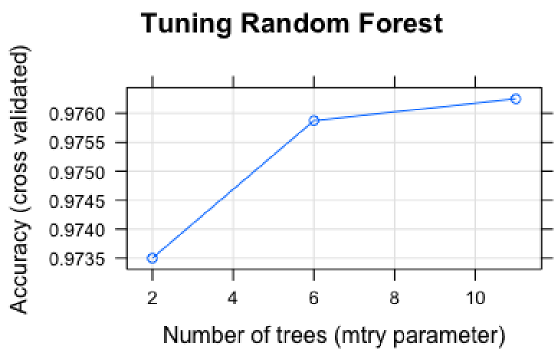
```
## GALAXY    QSO   STAR
##   6013   1858   2129
```

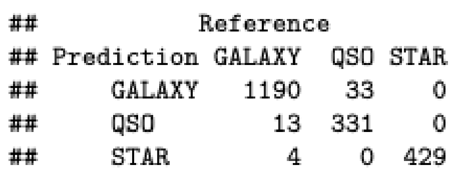
Note that for both Quasar and Stars, we are more likely to predict them to be galaxies. This is because of the class imbalance and there are many more galaxy data points than quasar or star data points. Since KNN makes predictions based on nearby data points, class imbalance has a large impact on the accuracy of prediction.

Figure 13: KNN Confusion Matrix

Random Forest

We used 10 fold cross validation to fit our random forest model. After tuning the mtry parameter with the number of trees, we found the optimum number of trees to be 6 with an accuracy of 97.6%.





```
##             Reference
## Prediction GALAXY  QSO STAR
##     GALAXY   1190   33    0
##     QSO        13  331    0
##     STAR        4    0  429
```

Figure 14: Random Forest Tuning and Confusion Matrix

Note that for our data, Random Forest performed much better than KNN. Although Random Forest still did predict some of the objects “Quasar” and “Star” to be of “Galaxy” class, and the class imbalance issue still exists, this problem affects Random Forest much less than KNN because Random Forest automatically balances sets of the data when one class is more frequent than other classes.

Neural Networks

Firstly, we consider why a neural networks model would be chosen for the star classification problem. All covariates are shown to be important in the variable selection process, so there are 11 variables with different levels of interaction; this complex balance may be captured by an equally complicated neural network. Despite covariates' distribution and scaling varying a lot, neural networks can handle this well by standardization. Additionally, the large sample size is a necessity for training neural networks to be powerful predictors, which is important when a small error rate still results in a large number of misclassifications.

We first fit single-layer neural networks, using the ‘nnet’ package (Venebles, 2002), see Appendix A for the network models. The single-layer neural networks approaches utilized were built upon each other in order: single-repetition, tuning size, tuning shrinkage, multiple repetition, and bagging. Additionally, the ‘caret’ library (Kuhn, 2022) was used to tune a single-layer neural network.

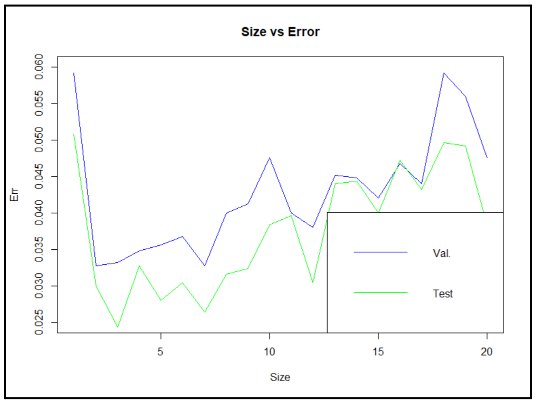


Figure 15: Size vs Error of Val and Test

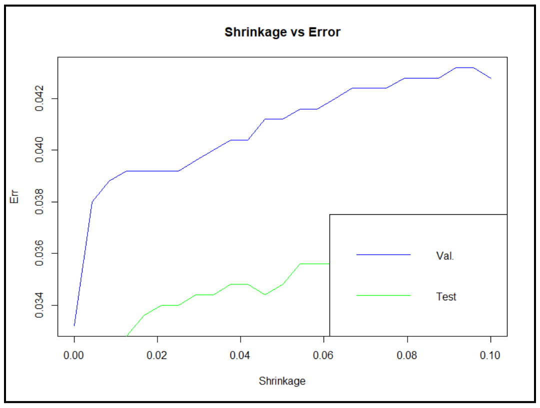


Figure 16: Shrinkage vs Error of Val and Test

Size is the number of nodes in the single-layer. Too few nodes restricts model flexibility to capture nonlinearities in data, while too many can lead to overfitting (Hastie, 2009, p. 400). The optimal size was found to be 2, which is not too complex. Shrinkage is a form of model regularization used to prevent overfitting and is combined with size to shrink weights down. In theory, larger sized models are better when properly regularized (James, 2013, p. 438). However, computational bounds and diminishing returns enforce more practical limits. Our optimal shrinkage of 0.0004 was on our lower end. This makes sense when considering that the number of coefficients in the size 2 model is dwarfed by the training set size.

Model: Hidden Layers x Nodes	**Number of Coefficients**
2 x 1	33
3 x 5	138
5 x 5	198

Table 1: Relation between product Hidden Layers x Nodes vs Coefficients #

Our approach is simplified, but flawed, in that first the optimal size is found, then the optimal shrinkage for that size is found. A better approach would be to do a grid search with multiple size and shrinkage combinations, which is seen with the ‘caret’ trained model. Appendix B features a caret boxplot of how test error varies with model size and shrinkage.

Another consideration in the fitting of neural networks is high variance, which can be traced to how neural networks tend to local minima due to the nonconvex nature of the objective function (James, 2013, p. 434). Thus, various starting points lead to various solutions. A solution is to have multiple repetitions, and take the majority of the repetitions as the predicted class. Another approach is to apply bagging, which has the advantage of avoiding overfitting. A further regularization technique, which we do not explore, is dropout learning, which parallels random forests. We fitted models of the single layer, size then shrinkage optimized neural network with 100 repetitions and 100 bootstrap repetitions. The number of repetitions itself could also be treated as a tuning parameter.

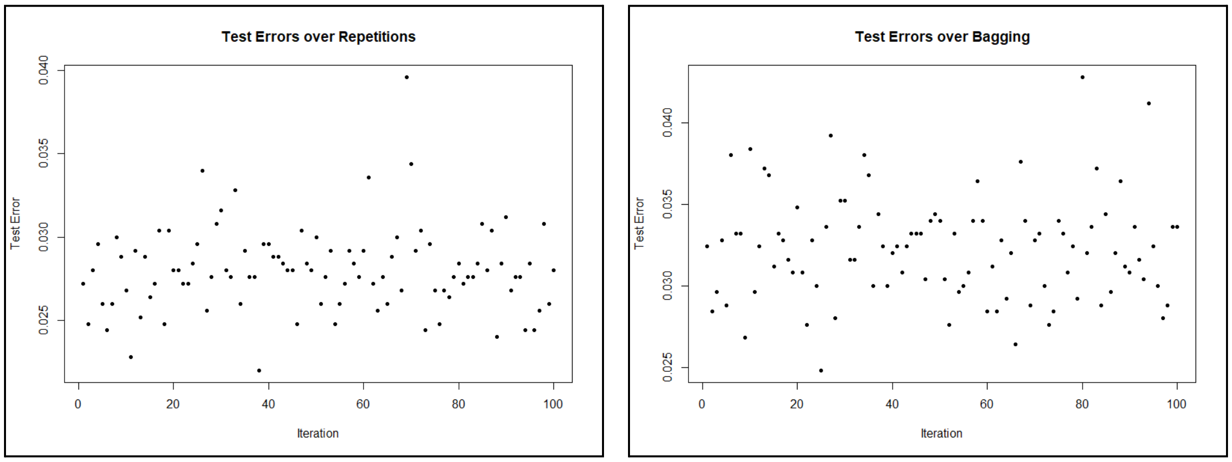


Figure 17: Test Errors over repetitions without bootstrap sampling on the left. With bootstrap sampling on the right.

We also explored multi-layer neural networks, fitted from the ‘neuralnet’ package (Wright, 2020), see Appendix B. We fitted 2 layers by 5 nodes over 3 repetitions, 3 layers by 5 nodes over 2 repetitions, and 5 layers by 5 nodes over 1 repetition. The advantage of a multi-layer neural network is that they are easier to train compared with a comparable single-layer neural network. Shrinkage, size, number of layers, and number of repetitions can be tuned as well, through the ‘caret’ package, but is not explored here due to computational bounds.

Neural Networks - Model Results

Model	**Test Error**	**Run Time (s)**
Single layer	0.0508	~1.03
Tuned size then shrinkage	0.0332	~203.38
>Repeated 100 times	0.0304	~145.97
>Bagged 100 times	0.03	~145.99
Single layer 'caret' tuned	0.0256	476.09
Multi-layer, 2 by 5, 3 reps	0.0276	65.49
Multi-layer, 3 by 5, 2 reps	0.03	54.13
Multi-layer, 5 by 5, 1 rep	0.0276	204.6

Table 2: Test Error and Runtime for Neural Network Models. Run times with leading tilde are approximate.

We notice that the single layer model which is properly tuned can perform as well as the multi-layer models, as in *An Introduction to Statistical Learning* (James, 2013, p. 407). However, a comparably accurate multi-layer model can be fitted much faster. Additionally, we don't see noticeable improvements between the multi-layer models, although the 5 by 5 model takes considerably more time despite less repetitions. The multi-layer models are not shrinkage regularized, which should improve the performance of the larger models.

Looking at our confusion matrices, the single layer models versus the caret tuned and multilayer models differ in their types of errors, see Appendix C. The single layer models do not falsely classify galaxies as stars, but are less accurate overall, while the multi-layer models do falsely classify galaxies as stars.

Conclusion

After performing feature selection, we noticed that *g, r, i, z, spec_obj_ID, MJD* are the variables of importance related to the class of stellar objects. Hence, we utilized these covariates as our reduced feature space for later modelling.

For our model training, we obtained some highly accurate models, competent enough to classify stellar objects with approximately 97% accuracy. Models that have performed especially well are Random Forest, Single layer "caret" tuned Neural Network, and Multi-layer, 5 by 5, 1 repetition Neural Network. These models all were able to obtain accuracy rates more than 97%. Even our KNN model with the lowest accuracy is still over 94% accurate. Overall, employing the Law of Parsimony, it can be concluded that the Random Forest model is best suited for classifying new stellar observations because it is highly accurate, not very computationally expensive and relatively simple to interpret.

Model	**Accuracy**
KNN	94.2%
Random Forest	97.6%
Single Layer Caret Tuned Neural Network	97.44%
Multi-Layer, 2 by 5, Neural Network	97.24%

Table 3: Model Performance

Contributions

Grigorii was involved in helping find the dataset of use, research to help explain the EDA and analysis, and editing the document. Harsh was involved in helping find the dataset, forming the research question, and the EDA. Chang was involved in feature selection using Best Subset and LASSO, as well as KNN, and Random Forest. Steven was involved in implementing neural nets and research to help explain the findings. Everyone assisted each other with their parts whenever necessary and we all believe everyone had an equal contribution in this project.

References

Beck MW (2018). “NeuralNetTools: Visualization and Analysis Tools for Neural Networks.” Journal of Statistical Software, 85(11), 1–20. doi:10.18637/jss.v085.i11.

Hastie, Tibshirani, R., Friedman, J. H., & Friedman, J. H. (Jerome H. . (2009). *The elements of statistical learning data mining, inference, and prediction* (2nd ed.). Springer.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013) *An Introduction to Statistical Learning with applications in R*, https://www.statlearning.com, Springer-Verlag, New York

Kuhn, M. et al. (2022). *Classification and Regression Training R package*. GitHub. Retrieved December 4, 2022, from https://github.com/topepo/caret/.

NASA. (2015, July 22). *How Many Stars in the Milky Way?* Retrieved from NASA: https://asd.gsfc.nasa.gov/blueshift/index.php/2015/07/22/how-many-stars-in-the-milky-way/

NASA. (2015, December 11). *Quasar (kway zar).* Retrieved from NASA: https://www.nasa.gov/audience/forstudents/k-4/dictionary/Quasar.html

NASA. (2016, October 13). *Hubble Reveals Observable Universe Contains 10 Times More Galaxies Than Previously Thought*. Retrieved from NASA: https://www.nasa.gov/feature/goddard/2016/hubble-reveals-observable-universe-contains-10-times-more-galaxies-than-previously-thought

NASA. (2019, April 10). *Black Hole Image Makes History; NASA Telescopes Coordinated Observations*. Retrieved from NASA: https://www.nasa.gov/mission_pages/chandra/news/black-hole-image-makes-history

NASA. (2022, December 5). *APOD: 2000 February 18 - Neptune through Adaptive Optics*. Retrieved from Pinterest: https://www.pinterest.ca/pin/247205467018974272/

NASA. (2022, December 5). *How Many Quasars Are There?* Retrieved from NASA: https://spacemath.gsfc.nasa.gov/Calculus/6Page107.pdf

Qualitative Reasoning Group, Northwestern University. (2022, December 5). *How are frequency and wavelength related?* Retrieved from Northwestern University:

https://www.qrg.northwestern.edu/projects/vss/docs/communications/2-how-are-frequency-and-wavelength-related.html

The European Space Agency. (2022, December 5). *What is 'red shift'?* Retrieved from The European Space Agency: https://www.esa.int/Science_Exploration/Space_Science/What_is_red_shift#:~:text=Ever%20since%201929%2C%20when%20Edwin,is%20%27red%2Dshifted%27.

University of Oregon. (2022, December 4). *Quasars*. Retrieved from University of Oregon: http://abyss.uoregon.edu/~js/cosmo/lectures/lec12.html#:~:text=In%20the%201930%27s%2C%20Edwin%20Hubble,farthest%20objects%20detected%20were%20quasars.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*, Fourth edition. Springer, New York. ISBN 0-387-95457-0, https://www.stats.ox.ac.uk/pub/MASS4/.

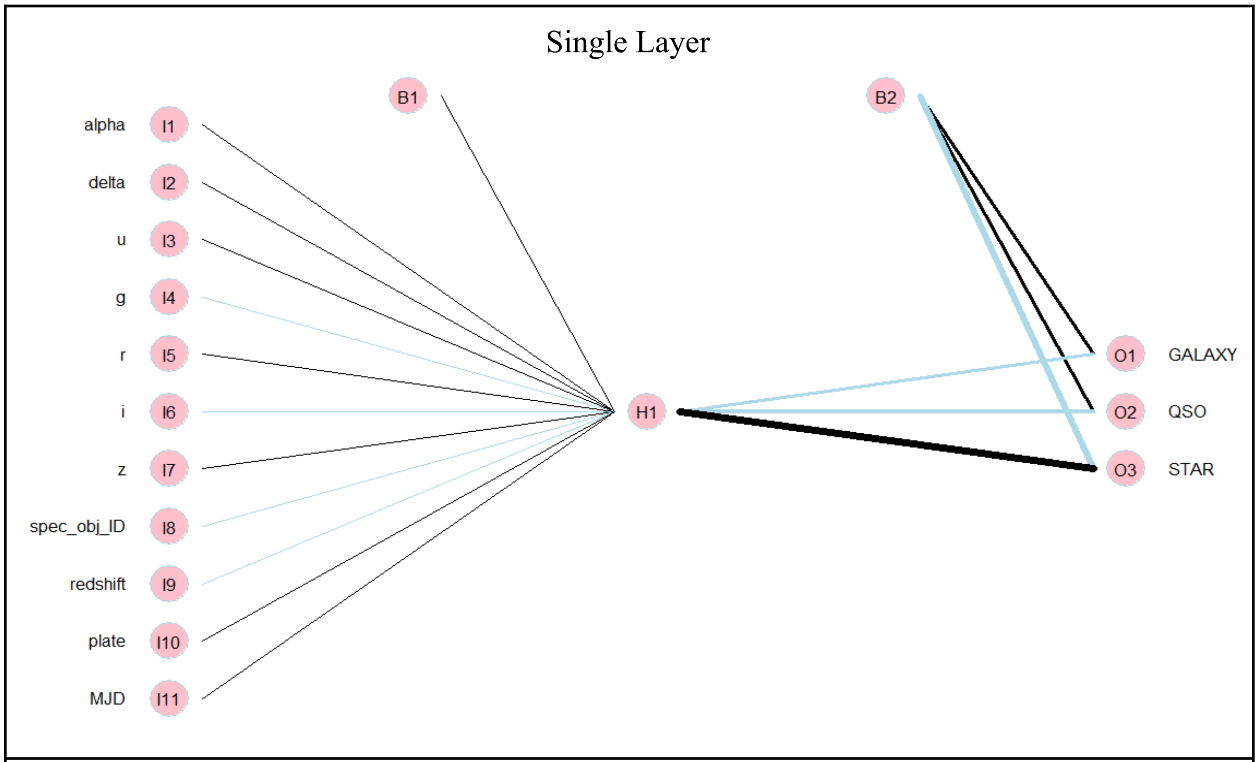
Wright, M. N. et al. (2020). *neuralnet: Training of Neural Networks.* GitHub. Retrieved December 4, 2022, from https://github.com/bips-hb/neuralnet.

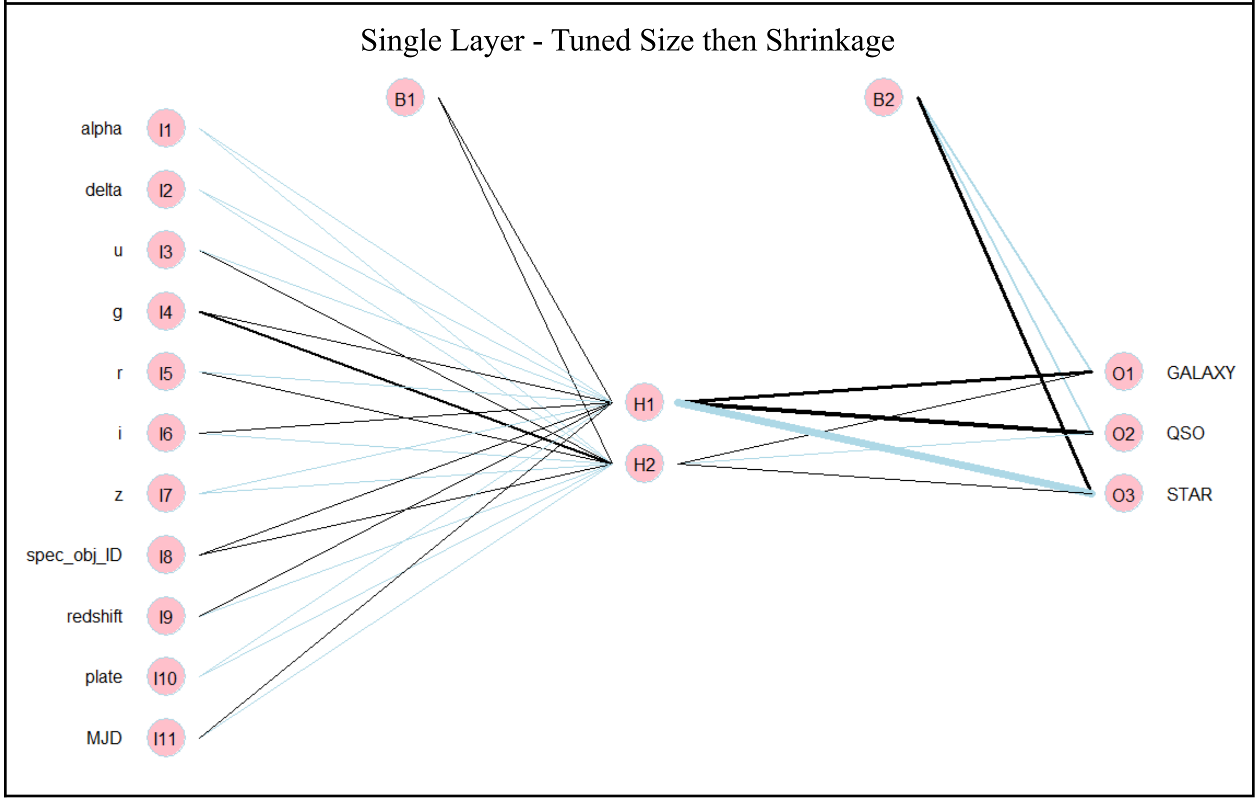
NASA. (2022, December 6). *Redshift and Hubble's Law*. Retrieved from Star Child: https://starchild.gsfc.nasa.gov/docs/StarChild/questions/redshift.html

Xie, Y., Ho, L. C., Zhuang, M.-Y., & Shangguan, J. (2021, April 5). The Infrared Emission and Vigorous Star Formation of Low-redshift Quasars. *The Astrophysical Journal*, 124. doi:10.3847/1538-4357/abe404

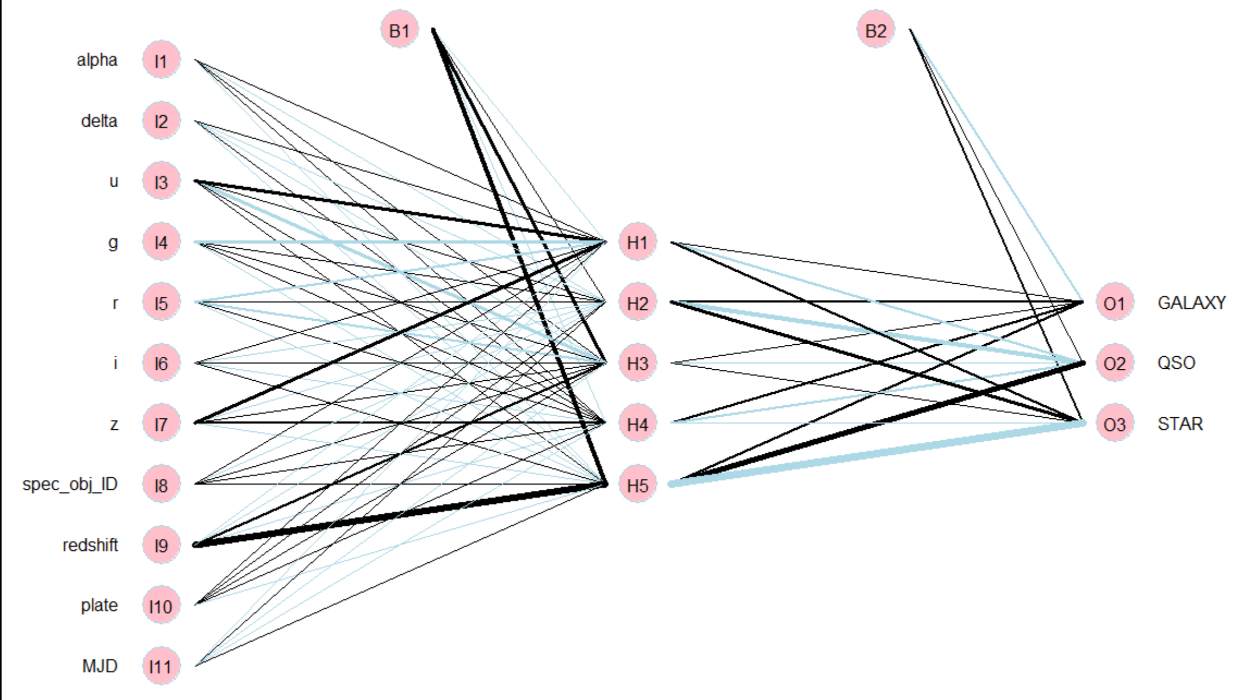
Appendix A: Neural Network Plots

Line thickness indicates weight magnitude. Black indicates positive weights whilst blue indicates negative weights. Created using the NeuralNetTools library (Beck, 2018).

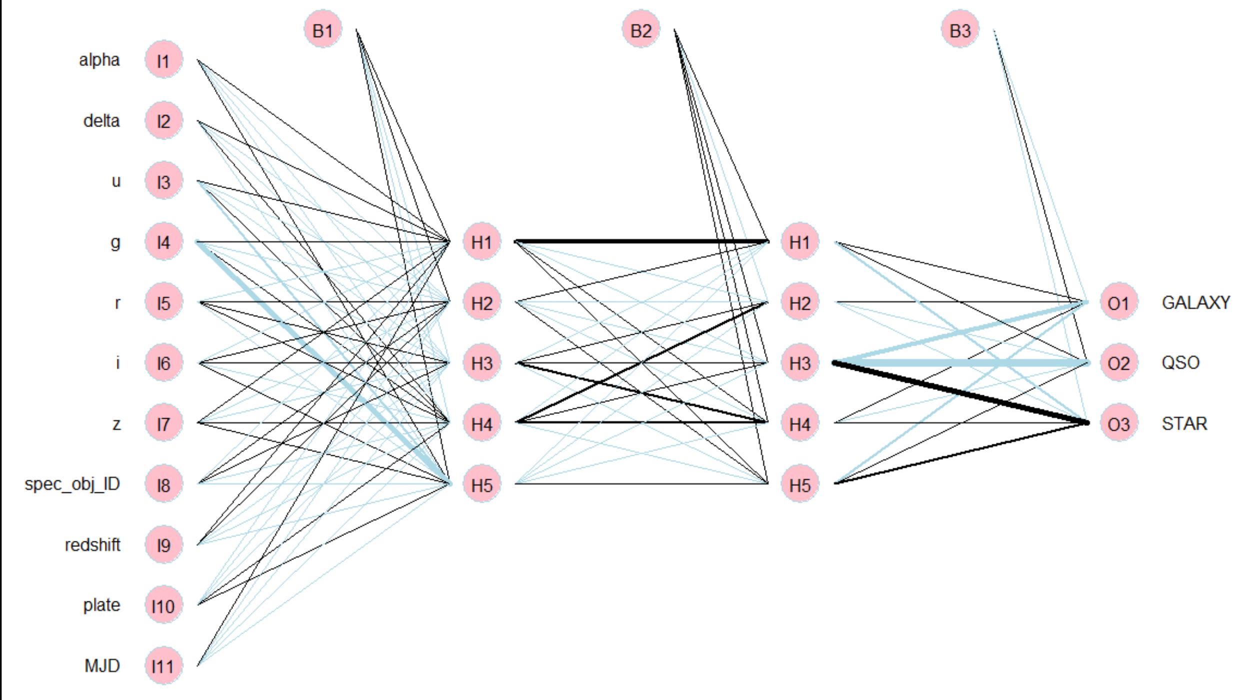


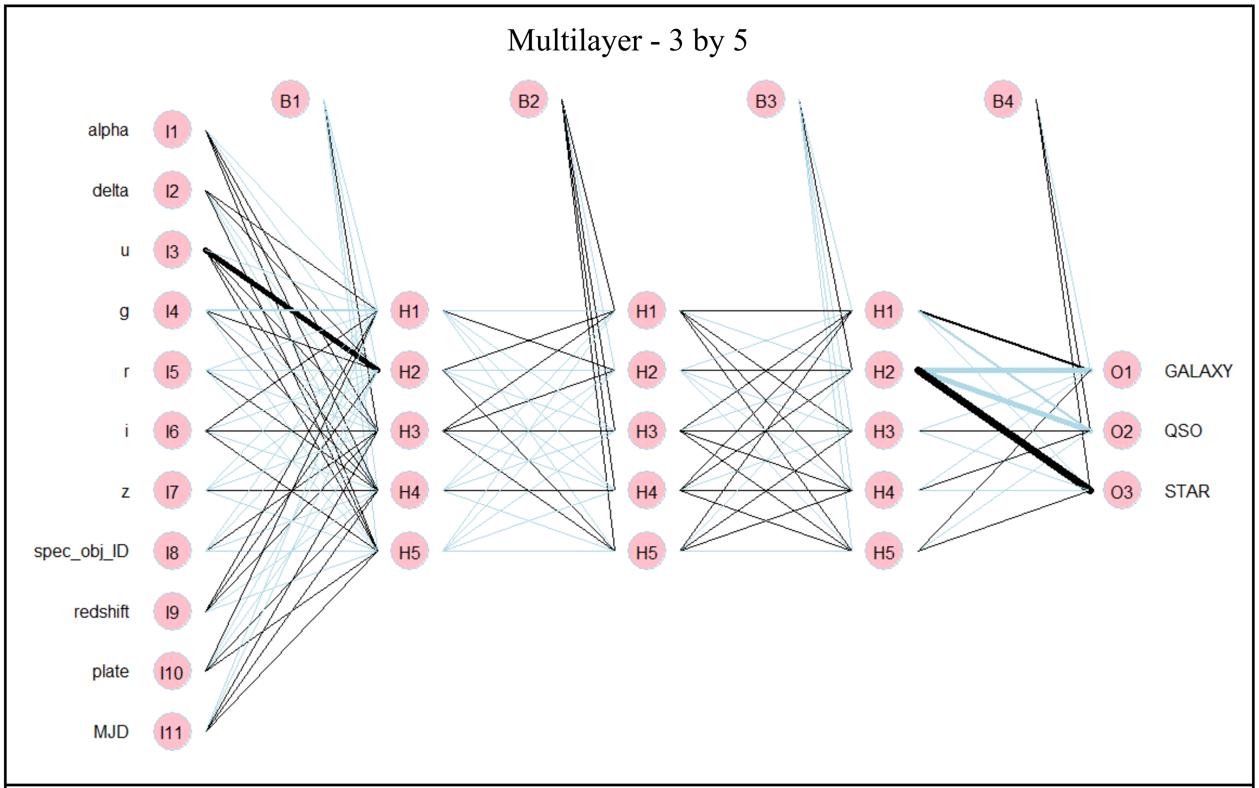


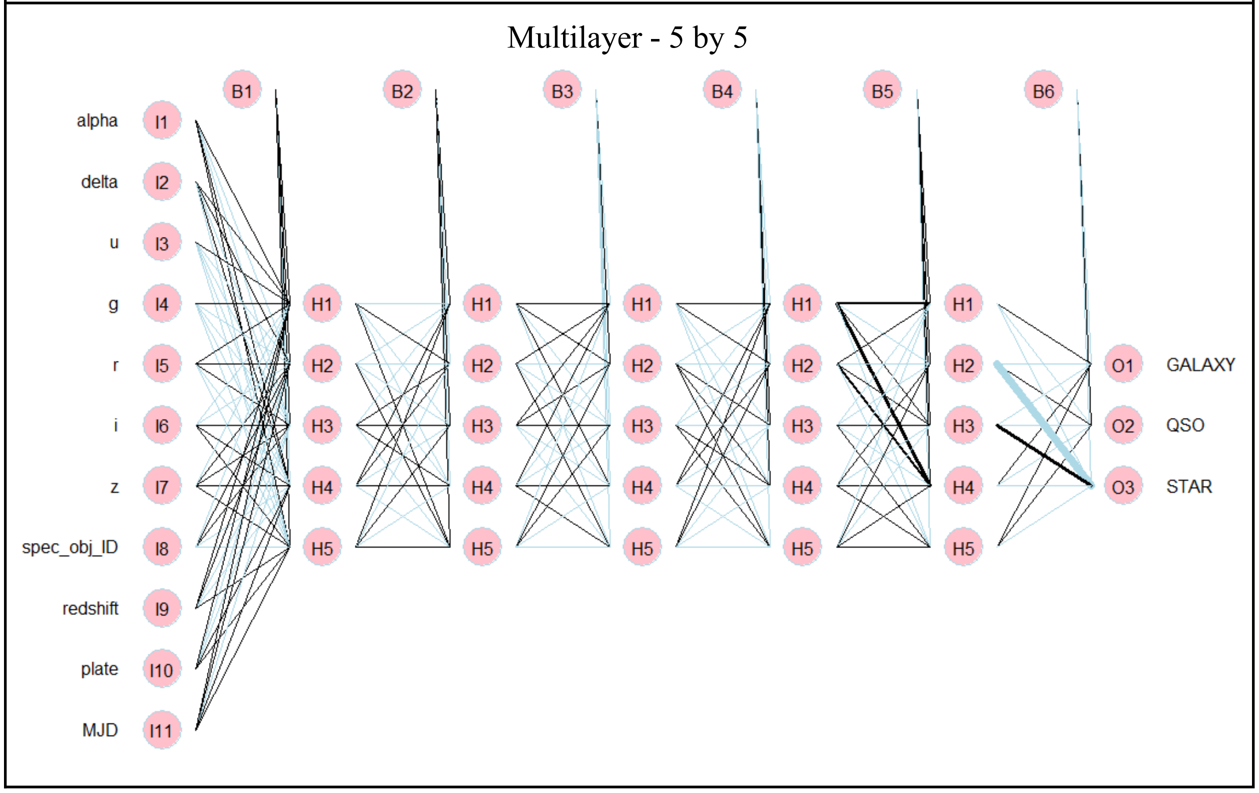
Single Layer Caret



Multilayer - 2 by 5

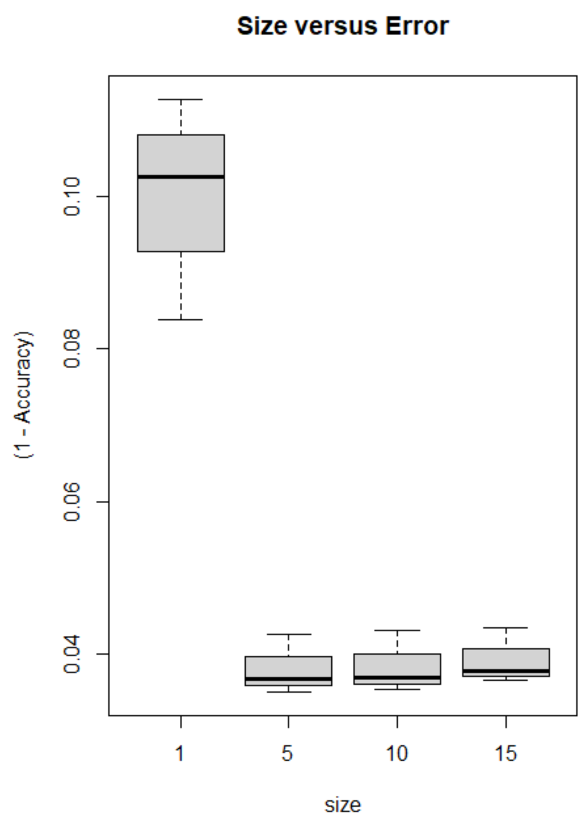


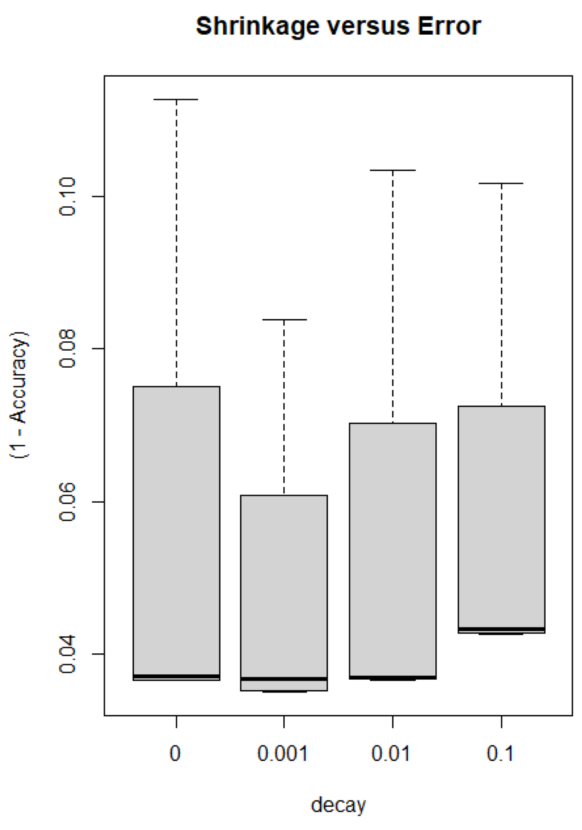

Multilayer - 3 by 5
B1
B2
B3
B4
alpha
delta
u
g
r
i
z
spec_obj_ID
redshift
plate
MJD
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
H1
H2
H3
H4
H5
O1
O2
O3
GALAXY
QSO
STAR


Multilayer - 5 by 5
B1
B2
B3
B4
B5
B6
alpha
delta
u
g
r
i
z
spec_obj_ID
redshift
plate
MJD
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
H1
H2
H3
H4
H5
O1
O2
O3
GALAXY
QSO
STAR

Appendix B - Boxplot of Neural Network Tuning Parameters

Left is a boxplot of the “Caret” Size vs Error. Right is a boxplot of the “Caret” Shrinkage vs Error.





Appendix C - Neural Network Confusion Matrices

Single Layer Neural Net

```
           Reference
Prediction GALAXY  QSO STAR
    GALAXY   1554   73    0
    QSO        31  330    0
    STAR       23    0  489
```

Single Layer with Tuned Size then Shrinkage

```
           Reference
Prediction GALAXY  QSO STAR
    GALAXY   1569   37    0
    QSO        21  366    0
    STAR       18    0  489
```

Single Layer with Repetition

```
           Reference
Prediction GALAXY  QSO STAR
    GALAXY   1578   31    0
    QSO        14  372    0
    STAR       16    0  489
```

Single Layer with Bagging

```
           Reference
Prediction GALAXY  QSO STAR
    GALAXY   1580   33    0
    QSO        12  370    0
    STAR       16    0  489
```

Single Layer Caret

```
           Reference
Prediction GALAXY  QSO STAR
    GALAXY   1578   32    2
    QSO        14  371    0
    STAR       16    0  487
```

Multilayer 2 by 5 Neural Net

```
           Reference
Prediction GALAXY  QSO STAR
    GALAXY   1566   23    4
    QSO        22  380    0
    STAR       20    0  485
```

Multilayer 3 by 5

```
           Reference
Prediction GALAXY  QSO STAR
    GALAXY   1560   24    3
    QSO        21  379    0
    STAR       27    0  486
```

Multilayer 5 by 5

```
           Reference
Prediction GALAXY  QSO STAR
    GALAXY   1566   22    4
    QSO        22  380    0
    STAR       20    1  485
```

Appendix D: EDA

```r
set.seed(1)
sc<-read.csv("C:/Users/harsh/Desktop/442_Final_Project/star_classification.csv")

df<-sc[c(2,3,4,5,6,7,8,13,14,15,16,17)]
df$class<-as.factor(df$class)
num_of_samples = 10000
sample_rows <- sample(1:nrow(sc), num_of_samples)
df<-df[sample_rows,]
```

Figure 1 and Figure 7

```r
class_percentage<- summary(factor(df$class))*100/length(df$class)
b1<-ggplot(df,aes(x=class,colour=class,fill=class))+geom_bar() +
  scale_fill_discrete(name="Percentage Composition",breaks = c("GALAXY", "QSO", "STAR"),
                     labels= c("GALAXY (59.18%)", "QSO (19.32%)", "STAR (21.5%)")) ## Figure 1
b2<-ggplot(df,aes(x=log(redshift),fill=class,colour=class)) +geom_density(alpha=0.2) ##Figure 7
b3<-ggplot(df,aes(x=alpha,fill=class,colour=class)) +geom_density(alpha=0.2)
b4<-ggplot(df,aes(x=delta,fill=class,colour=class)) +geom_density(alpha=0.2)
b5<-ggplot(df,aes(x=plate,fill=class,colour=class)) +geom_density(alpha=0.2)
b6<-ggplot(df,aes(x=u,fill=class,colour=class)) +geom_density(alpha=0.2)
#grid.arrange(b1,b2,b3,b4,b5,b6,ncol=3)
```

Figure 2 and Figure 4

```r
numeric_df<-df[c(1,2,3,4,5,6,7,8,10)]
pairs.panels(numeric_df) ## Figure 4
par(mfrow=c(3,3))
for (i in 1:ncol(numeric_df)){
  qqnorm(numeric_df[,i], main= c("Normal Q-Q Plot for:", names(numeric_df)[i]))
} ## Figure 2
par(mfrow=c(1,1))
```

Figure 3

```r
M<-cor(numeric_df)
corrplot(M,method="color")
```

Figure 6

```r
p1<-ggplot(df,aes(x=alpha,y=log(redshift),colour=class))+geom_point()
p2<-ggplot(df,aes(x=delta,y=log(redshift),colour=class))+geom_point()
p3<-ggplot(df,aes(x=u,y=redshift,colour=class))+geom_point()
p4<-ggplot(df,aes(x=g,y=log(redshift),colour=class))+geom_point()
p5<-ggplot(df,aes(x=r,y=log(redshift),colour=class))+geom_point()
```

```r
p6<-ggplot(df,aes(x=i,y=log(redshift),colour=class))+geom_point()
p7<-ggplot(df,aes(x=z,y=log(redshift),colour=class))+geom_point()
p8<-ggplot(df,aes(x=spec_obj_ID,y=log(redshift),colour=class))+geom_point()
p9<-ggplot(df,aes(x=plate,y=log(redshift),colour=class))+geom_point()
p10<-ggplot(df,aes(x=MJD,y=log(redshift),colour=class))+geom_point()
#arrange<-grid.arrange(p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,ncol=5) # Zoom to see clearly, very interesting
```

Appendix E: Feature Selection, KNN and Random Forest R Code

```
knitr::opts_chunk$set(echo = TRUE, eval=FALSE)
library(leaps)
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-4
```

```
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(randomForest)
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(nnet)
```

```
star = read.csv("star_classification.csv")
summary(factor(star$class))
num_of_samples = 10000
sample_rows <- sample(1:nrow(star), num_of_samples)


star <- star[c(2,3,4,5,6,7,8,13,14,15,16,17)]
star <- star[sample_rows, ]

star$class <- factor(star$class)
summary(star$class)
```

Feature Selection

```
set.seed(1)
head(star)
train = sample(nrow(star), nrow(star)*0.8)
star.train = star[train,]
star.test = star[-train,]
```

```
regfit.full <- regsubsets(factor(class)~., star.train)
reg.summary <- summary(regfit.full)
reg.summary
print("The number of variables for the best subset is: ")
which.min(reg.summary$bic)
which.min(reg.summary$cp)
which.min(reg.summary$rss)
which.min(reg.summary$adjr2)
```

```r
which.min(reg.summary$rsq)
plot(reg.summary$bic, xlab = "Number of Variables",
ylab = "BIC", main = "Feature selection", type = "l")
points(7, reg.summary$bic[7], col = "green", cex = 2, pch = 1)

print("The coefficients and corresponding values are:")
coeff <- coef(regfit.full, 7)
print(coeff)
```

LASSO

```r
star.train$class <- relevel(star.train$class, ref = "STAR")
x = model.matrix(class~., data = star.train)[, -1]
x_test = model.matrix(class~., data = star.test)[, -1]
y_test = star.test$class
y = star.train$class


grid =10^seq(-2, 3, length=100)
lasso.model = glmnet(x, y, alpha = 1 , lambda = grid, family = "multinomial")
plot(lasso.model, xvar = "lambda" , label = TRUE)
plot(lasso.model$lambda, lasso.model$df, xlab = "Lambda", ylab = "Model Size")

cv.fit = cv.glmnet(x , y, alpha = 1 , nfolds = 10, lambda = grid, family = "multinomial",
                   type.multinomial = "grouped", standardize = TRUE)

coef(cv.fit)

plot(cv.fit)
bestlambda = cv.fit$lambda.min

print("The coefficients for lambda.min are:")
coef(cv.fit, s = "lambda.min")

print("The coefficients for lambda.1se are:")
oneselamda = cv.fit$lambda.1se
coef(cv.fit, s = "lambda.1se")
```

```r
cv.fit
print("The test error using lambda.min is: ")
pred = predict(cv.fit, newx = x_test, s = "lambda.min" )

print("The test error using lambda.1se is: ")
pred = predict(cv.fit, newx = x_test, s = "lambda.1se" )
```

KNN

```r
library(class)
knn.train_x = star.train[c("g", "i", "z", "spec_obj_ID", "redshift", "MJD")]
knn.test_x = star.test[c("g", "i", "z", "spec_obj_ID", "redshift", "MJD")]

num_test_iter = 50
testerror.knn = rep(NA, num_test_iter)
```

```r
for (j in 1:num_test_iter) {
  set.seed(441)
  knn.pred = knn(scale(data.frame(knn.train_x)), scale(data.frame(knn.test_x)),
                 star.train$class, k=j, prob=TRUE)
  testerror.knn[j] = mean(knn.pred != star.test$class)
}
```

```r
plot(testerror.knn, xlab = "Number of Neighbors",
ylab = "Test Error", main = "Choosing Suitable k", type = "l")
which.min(testerror.knn)
knn.pred= knn(scale(data.frame(knn.train_x)), scale(data.frame(knn.test_x)),
              star.train$class, k=3, prob=TRUE)
confusionMatrix(knn.pred, star.test$class)$table
```

```r
summary(factor(star$class))
```

RF

```r
rf.model <- train(class~., data = star.train, method = 'rf',
                  trControl = trainControl(method = 'cv', number = 10))
```

```r
rf.model
plot(rf.model, main="Tuning Random Forest", ylab="Accuracy (cross validated)",
     xlab = "Number of trees (mtry parameter)")
rf.model$importance
rf.pred <- predict(rf.model, newdata = star.test)
confusionMatrix(factor(rf.pred), factor(star.test$class))
```

Appendix F: Neural Networks R Code

```r
# Neural Networks
library(nnet)

# Create training, validation, and test sets
set.seed(1)
trainSet = df[1:5000,]
validSet = df[5001:7500,]
testSet = df[7501:10000,]

# Rescale variables (even categorical) to have (mean,sd)=(0,1)
dfscaled = as.matrix(df[,-9])

for (i in 1:ncol(dfscaled)) {
  mx_mean = mean(dfscaled[,i])
  mx_sd = sd(dfscaled[,i])
  dfscaled[,i] = (dfscaled[,i]-mx_mean)/mx_sd
}

trainMx = dfscaled[1:5000,]
validMx = dfscaled[5001:7500,]
testMx = dfscaled[7501:10000,]

# Responses for SOFTMAX classification must be matrix of indicators
#  for each category
# Created by class.ind()

trainY = class.ind(trainSet[,9])
validY = class.ind(validSet[,9])
testY = class.ind(testSet[,9])

# Single Hidden Layer regression
# Classification: "softmax=true"
# No shrinkage and single node
set.seed(1)
nn.1.0 = nnet(x=trainMx, y=trainY, size=1, maxit=1000, softmax=TRUE)
# Test set error
predTest.1.0 = predict(nn.1.0, newdata=testMx, type="class")
testMisclass.1.0 <- mean(ifelse(predTest.1.0 == as.factor(testSet$class),
                                yes=0, no=1))

# Use 2 tuning parameters: decay (shrinkage) and size
# Number of Units: "size=" more units give more flexibility in model,
#  but more likelihood to overfit
sizes = seq(from=1,to=20)
nn.size.errs = sapply(sizes, function(s){
  set.seed(1)
  p = nnet(x=trainMx, y=trainY, size=s, maxit=1000, softmax=TRUE)
  predValid = predict(p, newdata=validMx, type="class")
  validErr = mean(ifelse(predValid == as.factor(validSet$class), yes=0, no=1))
  predTest = predict(p, newdata=testMx, type="class")
  testErr = mean(ifelse(predTest == as.factor(testSet$class), yes=0, no=1))
  return(list(validErr = validErr, testErr = testErr))
```

```
})
plot(sizes,nn.size.errs[1,],main="Size vs Error",xlab="Size",
     ylab="Err",col="blue",type="l", ylim= c(0.025,0.06))
points(sizes,nn.size.errs[2,],type="l",col="green")
legend("bottomright",legend=c("Val.","Test"),col=c("blue","green"),lty=c(1,1))

# Select optimal size
optSize = sizes[which.min(unlist(nn.size.errs[1,]))]
set.seed(1)
nn.size.opt = nnet(x=trainMx, y=trainY, size=optSize, maxit=1000, softmax=TRUE)
nn.size.opt.predTest = predict(nn.size.opt, newdata=testMx, type="class")

# Optimization: adding shrinkage
# Shrinkage: "decay=" value from [0.0001,0.1]
# Only 25 values tested due to computational bounds
decay = seq(from=0.0001,to=0.1,length.out=25)
nn.shrinkage.errs = sapply(decay, function(d){
  set.seed(1)
  p = nnet(x=trainMx, y=trainY, size=optSize, decay=d, maxit=1000, softmax=TRUE)
  predValid = predict(p, newdata=validMx, type="class")
  validErr = mean(ifelse(predValid == as.factor(validSet$class), yes=0, no=1))
  predTest = predict(p, newdata=testMx, type="class")
  testErr = mean(ifelse(predTest == as.factor(testSet$class), yes=0, no=1))
  return(list(validErr = validErr, testErr = testErr))
})
# Plot Shrinkage vs Error
plot(decay,nn.shrinkage.errs[1,],main="Shrinkage vs Error",xlab="Shrinkage",
     ylab="Err",col="blue",type="l",)
points(decay,nn.shrinkage.errs[2,],type="l",col="green")
legend("bottomright",legend=c("Val.","Test"),col=c("blue","green"),lty=c(1,1))

# Select decay to be the minimum from our validation error
optDecay = decay[which.min(unlist(nn.shrinkage.errs[1,]))]
set.seed(1)
nn.shrinkage.opt = nnet(x=trainMx, y=trainY, size=optSize, decay=optDecay,
                        maxit=1000, softmax=TRUE)
nn.shrinkage.opt.predTest = predict(nn.shrinkage.opt, newdata=testMx,
                                    type="class")

# Repetition: required to adjust for fact that local minimums and possibility
#  for overfitting can occur given various random starting weights
# Takes majority prediction
set.seed(1)
nn.repeat.predictors = lapply(seq(1,100),function(i){
  nnet(x=trainMx, y=trainY, size=optSize, decay=optDecay, maxit=1000,
       softmax=TRUE)
})
nn.repeat.predictions = sapply(nn.repeat.predictors, function(p){
  predTest = unlist(predict(p, newdata=testMx, type="class"))
})

# Plot test errors of all repetitions
nn.repeat.errors = apply(nn.repeat.predictions, MARGIN = 2, function(x){
```

```r
  mean(ifelse(x == as.factor(testSet$class), yes=0, no=1))
})
plot(seq(1,100), nn.repeat.errors, main="Test Errors over Repetitions",
     xlab="Iteration", ylab="Test Error", pch=20)

# Take Majority Prediction
nn.repeat.class = apply(nn.repeat.predictions, MARGIN = 1, function(x){
  uniquex = unique(x)
  uniquex[which.max(tabulate(match(x,uniquex)))]
})
testErr.repeat = mean(ifelse(nn.repeat.class == as.factor(testSet$class),
                             yes=0, no=1))

# Bagging: Alternative to repetition, creates different samples each time
set.seed(1)
nn.bagging.predictors = lapply(seq(1,100),function(i){
  trainSample = sample(nrow(trainSet),nrow(trainSet),replace = TRUE)
  bagTrainMx = dfscaled[trainSample,]
  bagTrainY = class.ind(trainSet[trainSample,9])
  nnet(x=bagTrainMx, y=bagTrainY, size=optSize, decay=optDecay, maxit=1000,
       softmax=TRUE)
})
nn.bagging.predictions = sapply(nn.bagging.predictors, function(p){
  predTest = unlist(predict(p, newdata=testMx, type="class"))
})

# Plot test errors of all repetitions
nn.bagging.errors = apply(nn.bagging.predictions, MARGIN = 2, function(x){
  mean(ifelse(x == as.factor(testSet$class), yes=0, no=1))
})
plot(seq(1,100), nn.bagging.errors, main="Test Errors over Bagging",
     xlab="Iteration", ylab="Test Error", pch=20)

# Take Majority Prediction
nn.bagging.class = apply(nn.bagging.predictions, MARGIN = 1, function(x){
  uniquex = unique(x)
  uniquex[which.max(tabulate(match(x,uniquex)))]
})
testErr.bagging = mean(ifelse(nn.bagging.class == as.factor(testSet$class),
                              yes=0, no=1))

# Try grid search with caret library
library(caret)
set.seed(1)
tuned.nnet <- caret::train(x=trainSet[,-9], y=trainSet[,9], method="nnet",
                           preProcess="range", trace=FALSE,
                           tuneGrid=expand.grid(.size=c(1,5,10,15),
                                                .decay=c(0,0.001,0.01,0.1)))

tuned.nnet$results[order(-tuned.nnet$results[,3]),]
tuned.nnet$bestTune

# Try boxplots to see variability of the two tuning parameters
```

```r
x11(h=7, w=10)
par(mfrow=c(1,2))
boxplot((1-Accuracy) ~ size, data=tuned.nnet$results,
        main = "Size versus Error")
boxplot((1-Accuracy) ~ decay, data=tuned.nnet$results,
        main = "Shrinkage versus Error")

# We try NN with the bestTune parameters from grid search
set.seed(1)
nn.caret = nnet(x=trainMx, y=trainY, size=tuned.nnet$bestTune$size, maxit=1000,
                decay=tuned.nnet$bestTune$decay, softmax=TRUE)
nn.caret.pred = predict(nn.caret, newdata=testMx, type="class")

table(nn.caret.pred, as.factor(testSet$class),  dnn=c("Predicted","Observed"))
misclass.caret <- mean(ifelse(nn.caret.pred == as.factor(testSet$class),
                               yes=0, no=1))

# Multilayer neural networks using neuralnet package
library(neuralnet)

trainData = cbind(trainY,trainMx)
nnFormula = as.formula(paste("GALAXY + QSO + STAR ~",
                             paste(names(trainSet[,-9]), collapse = " + ")))

# 2 Layer x 5 Nodes
set.seed(1)
neural.3 = neuralnet(nnFormula, trainData, hidden=c(5,5), rep=3, err.fct="sse",
                     act.fct="logistic", threshold = 1, stepmax = 10000,
                     lifesign = 'minimal', linear.output=FALSE)

nn.fancy.3.preds = compute(neural.3,testMx)$net.result
colnames(nn.fancy.3.preds) = c("GALAXY", "QSO", "STAR")
nn.fancy.3.class = data.frame("class" =
                                ifelse(max.col(nn.fancy.3.preds[ ,1:3])==
                                         1,"GALAXY",
                                       ifelse(max.col(nn.fancy.3.preds[ ,1:3])==
                                                2, "QSO", "STAR")))
# Confusion Matrix
caret::confusionMatrix(as.factor(testSet[,9]),as.factor(nn.fancy.3.class[,1]))
# Test Error
misclass.fancy.3 <- mean(ifelse(nn.fancy.3.class[,1] == as.factor(testSet$class), yes=0, no=1))

# 3 Layer x 5 Nodes
set.seed(1)
neural.4 = neuralnet(nnFormula, trainData, hidden=c(5,5,5), rep=2,
                     err.fct="sse", act.fct="logistic", threshold = 1,
                     stepmax = 10000, lifesign = 'minimal', linear.output=FALSE)

nn.fancy.4.preds = compute(neural.4,testMx)$net.result
colnames(nn.fancy.4.preds) = c("GALAXY", "QSO", "STAR")
nn.fancy.4.class = data.frame("class"=ifelse(max.col(nn.fancy.4.preds[ ,1:3])==
                                                1, "GALAXY",
                                       ifelse(max.col(nn.fancy.4.preds[ ,1:3])==
                                                2,"QSO", "STAR")))
# Confusion Matrix
caret::confusionMatrix(as.factor(testSet[,9]),as.factor(nn.fancy.4.class[,1]))
# Test Error
misclass.fancy.4 <- mean(ifelse(nn.fancy.4.class[,1] ==
                                  as.factor(testSet$class), yes=0, no=1))

# 5 Layer x 5 Nodes
set.seed(1)
neural.5 = neuralnet(nnFormula, trainData, hidden=c(5,5,5,5,5), rep=1,
                     err.fct="sse", act.fct="logistic", threshold = 1,
                     stepmax = 20000, lifesign = 'minimal', linear.output=FALSE)

nn.fancy.5.preds = compute(neural.5,testMx)$net.result
colnames(nn.fancy.5.preds) = c("GALAXY", "QSO", "STAR")
nn.fancy.5.class = data.frame("class" = ifelse(max.col(nn.fancy.5.preds[ ,1:3])
                                               ==1, "GALAXY",
                                       ifelse(max.col(nn.fancy.5.preds[ ,1:3])==
                                                2, "QSO", "STAR")))
# Confusion Matrix
caret::confusionMatrix(as.factor(testSet[,9]),as.factor(nn.fancy.5.class[,1]))
# Test Error
misclass.fancy.5 <- mean(ifelse(nn.fancy.5.class[,1] == as.factor(testSet$class)
                                , yes=0, no=1))
# Comparison of models:
# Single Layer, Single Layer w/optimal size, Single Layer
#  w/optimal shrinkage and size, Single Layer bagging, Caret Single Layer,
#  2 Layer x 5 Nodes, 3 Layer x 5 Nodes, 5 Layer x 5 Nodes

# Test errors
layer1.size1.err = testMisclass.1.0
layer1.optsize.err = min(unlist(nn.size.errs[1,]))
layer1.optshrink.err = min(unlist(nn.shrinkage.errs[1,]))
layer1.repeat = testErr.repeat
layer1.bagging = testErr.bagging
layer1.caret.err = misclass.caret
layer2.size5.err = misclass.fancy.3
layer3.size5.err = misclass.fancy.4
layer5.size5.err = misclass.fancy.5

# Confusion Matrices
layer1.size1.cm = caret::confusionMatrix(as.factor(predTest.1.0),
                                         as.factor(testSet[,9]))$table
layer1.optshrink.cm =
  caret::confusionMatrix(as.factor(nn.shrinkage.opt.predTest),
                         as.factor(testSet[,9]))$table
layer1.repeat.cm = caret::confusionMatrix(as.factor(nn.repeat.class),
                                          as.factor(testSet[,9]))$table
layer1.bagging.cm = caret::confusionMatrix(as.factor(nn.bagging.class),
                                           as.factor(testSet[,9]))$table
layer1.caret.cm = caret::confusionMatrix(as.factor(nn.caret.pred),
                                         as.factor(testSet[,9]))$table
layer2.size5.cm = caret::confusionMatrix(as.factor(nn.fancy.3.class[,1]),
                                         as.factor(testSet[,9]))$table
layer3.size5.cm = caret::confusionMatrix(as.factor(nn.fancy.4.class[,1]),
                                         as.factor(testSet[,9]))$table
layer5.size5.cm = caret::confusionMatrix(as.factor(nn.fancy.5.class[,1]),
                                         as.factor(testSet[,9]))$table

# Neural Net Plots
library(NeuralNetTools)
plotnet(nn.1.0, neg_col="lightblue", circle_col = "pink")
# We do not use the size optimized plot
plotnet(nn.shrinkage.opt, neg_col="lightblue", circle_col = "pink")
# Repeat and Bagging approach does not have a model
plotnet(nn.caret, neg_col="lightblue", circle_col = "pink")
plotnet(neural.3, neg_col="lightblue", circle_col = "pink")
plotnet(neural.4, neg_col="lightblue", circle_col = "pink")
plotnet(neural.5, neg_col="lightblue", circle_col = "pink")

# Run times timed by R
caret.time = tuned.nnet$times$everything
# neuralnet calls are timed, but information not stored
# Thus, R times 2 by 5, 3 by 5, 5 by 5 neural nets
```