Capstone Project **Trevor Clark**
Machine Learning Engineer Nanodegree

Supervised Learning for Star Cluster Classification

1 Introduction

The formation and death of stars are the dominant events in our universe. From these stars, the most simple hydrogen elements are transformed into the heavier elements of the periodic table. It is these complex elements that create the galaxies, comets, asteroids, planets, solar systems, and us. The conditions for the successful development of stars tend to occur in dense star clusters. The one star in our solar system is very unique in that solar systems observed throughout the universe are most frequently binary systems with two stars in orbit around each other.

2 Problem Statement

Astronomers and astrophysicists explore the various lifecycle phases of stars to better understand how solar systems evolve. N-body star cluster simulations provide a means to model the interactions between stars within a given cluster.[1] Some stars drop out of the simulation as they escape the star cluster. To better understand the development our own solar system, it is useful to determine the feature characteristics that lead to isolated stars. Supervised machine learning techniques have been applied to classify celestial bodies from astronomical observations. [2, 3] By using supervised machine learning classification algorithms (see Figure 2-1 below), the stars that will likely become isolated can be predicted. This approach provides a means for focusing our exploration efforts and resources on specific stars that support deepening our insight into the formation of our own Sun and solar system.

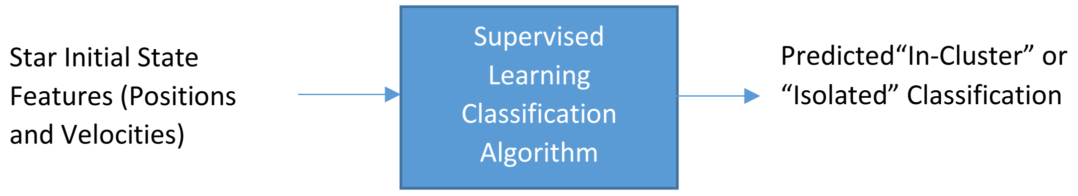

Star Initial State Features (Positions and Velocities)
Supervised Learning Classification Algorithm
Predicted"In-Cluster" or "Isolated" Classification

Figure 2-1 Supervised Learning Star Classification Model

3 Data Exploration

On kaggle.com[1], a dataset has been posted from a N-body star cluster simulation. The cluster simulation results include a star's position, velocity, mass and identification (ID) numbers as dataset features. Individual comma separated value (csv) files represent the feature data epoch time captures of the 64,000 stars within the cluster from the initial conditions to the end of the simulation. The positions and velocities are reflected as standardized N-body units. The star's initial position and velocity state data are distributed based on a King's model. Table 3-1 below is a sample of the N-Body star cluster dataset.

Table 3-1 N-Body Star Feature Sample

Star Position Data			Star Velocity Data			Star Mass	Star ID Number
x	y	z	vx	vy	vz		
0.791029	-1.16329	1.846175	0.000687	-0.0723	-0.01453	1.56E-05	22770
0.953431	-1.93228	0.441769	-0.409	0.456364	0.085676	1.56E-05	26771
0.995871	-0.99339	0.598057	-0.28609	0.025921	-0.15232	1.56E-05	35526
0.271007	-0.00523	1.080754	-0.03894	-0.24425	0.876209	1.56E-05	46544
-0.0522	-0.04324	0.523306	0.159853	0.510218	0.275074	1.56E-05	16733
0.499011	-0.22582	-0.10217	0.234844	0.648561	0.175006	1.56E-05	37382
-3.18051	-0.86475	0.336	0.200171	0.058656	-0.03269	1.56E-05	23650
-0.10222	0.174247	0.24045	-0.04045	-0.47619	-0.11163	1.56E-05	31785
-2.66139	1.892381	0.121634	-0.01654	0.000347	0.06725	1.56E-05	10466

3.1 Dataset Pre-processing

3.1.1 Star Classification Assignments

The star ID numbers that are included in the end of simulation dataset were used to establish the "In-Cluster" star classifications in the initial dataset. The missing star ID numbers in the end of simulation dataset were classified as "Isolated" stars in the initial dataset. Table 3.1.1-1 reflects a sample of the N-Body star cluster dataset with the star classifications.

Table 3.1.1 - 1 N-Body Star Feature/Classification Sample

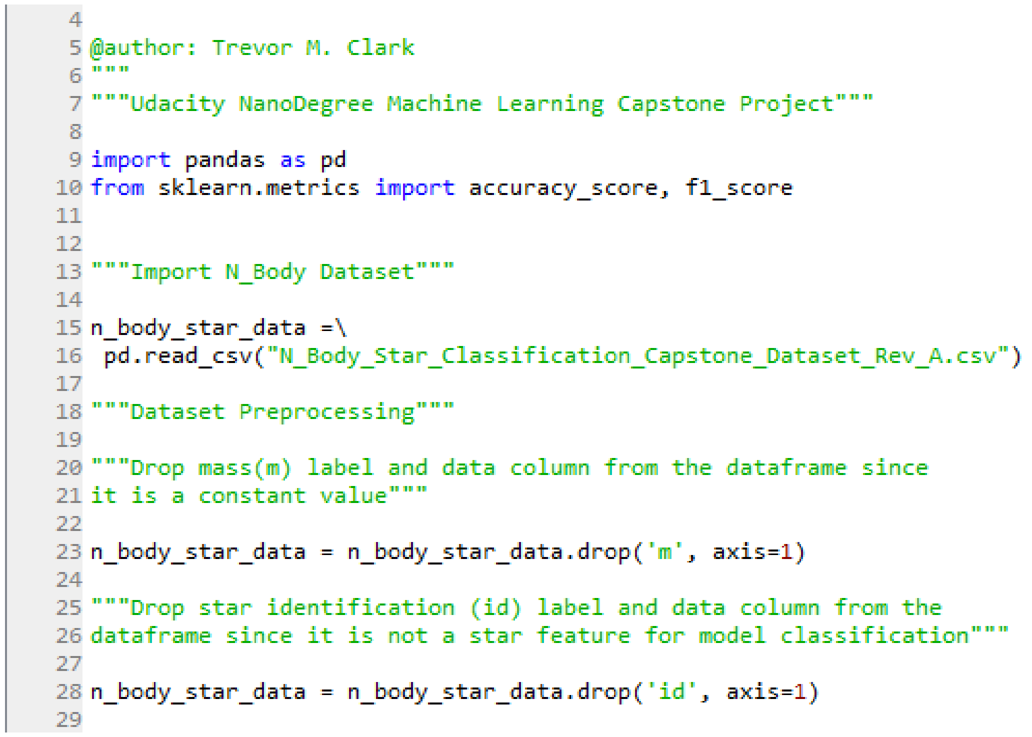
Star Position Data			Star Velocity Data			Star Mass	Star ID Number	Star Classification
x	y	z	vx	vy	vz			
0.791029	-1.16329	1.846175	0.000687	-0.0723	-0.01453	1.56E-05	22770	Isolated
0.953431	-1.93228	0.441769	-0.409	0.456364	0.085676	1.56E-05	26771	Isolated
0.995871	-0.99339	0.598057	-0.28609	0.025921	-0.15232	1.56E-05	35526	Isolated
0.271007	-0.00523	1.080754	-0.03894	-0.24425	0.876209	1.56E-05	46544	Isolated
-0.0522	-0.04324	0.523306	0.159853	0.510218	0.275074	1.56E-05	16733	In_Cluster
0.499011	-0.22582	-0.10217	0.234844	0.648561	0.175006	1.56E-05	37382	In_Cluster
-3.18051	-0.86475	0.336	0.200171	0.058656	-0.03269	1.56E-05	23650	Isolated
-0.10222	0.174247	0.24045	-0.04045	-0.47619	-0.11163	1.56E-05	31785	In_Cluster
-2.66139	1.892381	0.121634	-0.01654	0.000347	0.06725	1.56E-05	10466	Isolated

3.1.2 Training Dataset Sample Size Definition

To reduce the likelihood of the entire dataset (99.953125% - “In_Cluster”; 0.046875% - “Isolated”) biasing the supervised learning algorithm, a sample training/test dataset from the 64,000 stars was created with twenty (20) “Isolated” stars and 110 “In_Cluster” star. The training/test dataset “Isolated” stars were selected from the pool of “Isolated” star ID using a random number generator. The same random number generator approach was used to select the “In_Cluster” star IDs for the training/test dataset.

3.1.3 Dataset Dimensional Analysis

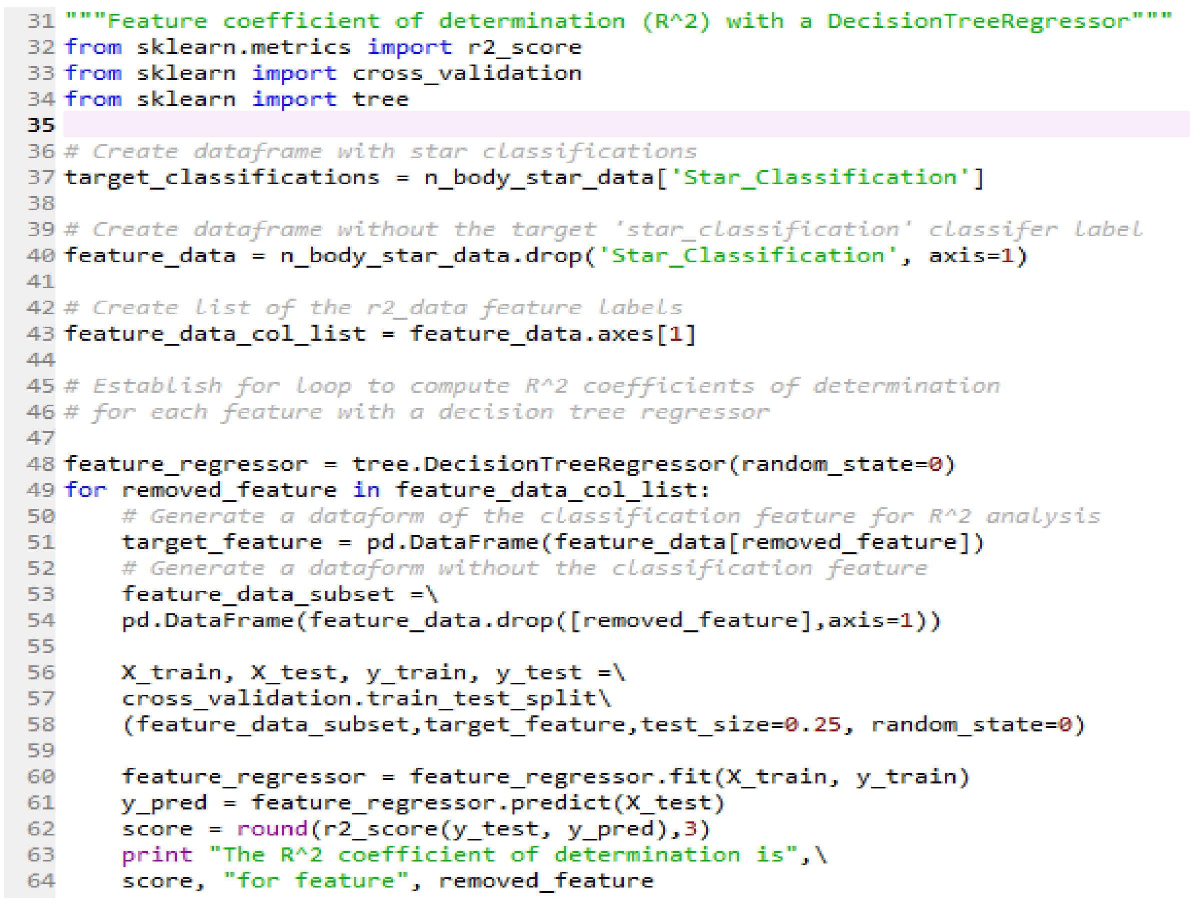
Since the mass of the stars were defined as a uniform distribution of the overall mass of the cluster, it was not a feature used (dropped) in the training/test dataset. Also, the star ID number was not considered a classification feature; therefore, it was removed (dropped) from the training/test dataset. The Python code to import the training/test dataset and accomplish the initial pre-processing is depicted in Figure 3.1.3-1.



```

@author: Trevor M. Clark
"""
"""Udacity NanoDegree Machine Learning Capstone Project"""

import pandas as pd
from sklearn.metrics import accuracy_score, f1_score


"""Import N_Body Dataset"""

n_body_star_data =\
 pd.read_csv("N_Body_Star_Classification_Capstone_Dataset_Rev_A.csv")

"""Dataset Preprocessing"""

"""Drop mass(m) label and data column from the dataframe since
it is a constant value"""

n_body_star_data = n_body_star_data.drop('m', axis=1)

"""Drop star identification (id) label and data column from the
dataframe since it is not a star feature for model classification"""

n_body_star_data = n_body_star_data.drop('id', axis=1)

```

Figure 3.1.3-1 CSV file Dataset Import and Initial Pre-processing Python Code

The coefficient of determination (R^2) for each of the remaining features was accomplished (see Python code in Figure 3.1.3-2) as an opportunity to further reduce the dimensions of the training/test dataset (see Table 3.1.3-1).



```python
"""Feature coefficient of determination (R^2) with a DecisionTreeRegressor"""
from sklearn.metrics import r2_score
from sklearn import cross_validation
from sklearn import tree

# Create dataframe with star classifications
target_classifications = n_body_star_data['Star_Classification']

# Create dataframe without the target 'star_classification' classifer label
feature_data = n_body_star_data.drop('Star_Classification', axis=1)

# Create list of the r2_data feature labels
feature_data_col_list = feature_data.axes[1]

# Establish for loop to compute R^2 coefficients of determination
# for each feature with a decision tree regressor

feature_regressor = tree.DecisionTreeRegressor(random_state=0)
for removed_feature in feature_data_col_list:
    # Generate a dataform of the classification feature for R^2 analysis
    target_feature = pd.DataFrame(feature_data[removed_feature])
    # Generate a dataform without the classification feature
    feature_data_subset =\
    pd.DataFrame(feature_data.drop([removed_feature],axis=1))

    X_train, X_test, y_train, y_test =\
    cross_validation.train_test_split\
    (feature_data_subset,target_feature,test_size=0.25, random_state=0)

    feature_regressor = feature_regressor.fit(X_train, y_train)
    y_pred = feature_regressor.predict(X_test)
    score = round(r2_score(y_test, y_pred),3)
    print "The R^2 coefficient of determination is",\
    score, "for feature", removed_feature
```

Figure 3.1.3-2 Star Feature Coefficient of Determination (R^2) Python Code

Table 3.1.3-1 Feature Coefficient of Determination (R^2) Analysis Results

Feature Labels	**Coefficients of Determination (R^2)**
X - Position	-1.703
Y - Position	-2.200
Z - Position	-0.470
Vx - Velocity	-2.216
Vy - Velocity	-2.032
Vz - Velocity	-2.197

With all the features coefficient of determination (R^2) results being negative, none of the star's position and velocity features will be removed from the training/test dataset.

4 Solution Statement

To predict the stars that will likely become isolated from the star cluster system, Support Vector Machine (SVM) and logistic regression machine learning algorithms were implemented. To ensure the model optimization approaches were repeatable, the cross validation random state was set equal to zero (0). The solution objective is to demonstrate SVM and/or logistic regression algorithm performance metrics meet or exceed the decision tree benchmark model.

4.1 Benchmark Model

Supervised learning decision tree models have been successfully used for the star and galaxy Digitalized Palomar Observatory Sky Survey (POSS) Two and it is considered the benchmark for N-body star isolation prediction modelling.[2] The initial POSS achieved between 30 to 90% classification accuracy with a neural network algorithm using star and galaxy intensity feature data. The D-POSS II implemented a decision tree model from eight (8) features to consistently classify stars and galaxies with 94% accuracy.

4.2 Algorithm and Techniques

SVM is a supervised machine learning algorithm that maximizes the training data decision boundary (hyperplane) margin. A SVM algorithm was anticipated to perform well with the complexity and high dimensions of this dataset. To address the billions of the stars that maybe classified by this supervised machine learning application, the logistic regression model was evaluated to take advantage of faster runtimes if this algorithm had the best prediction performance. Also, the binary classification aspects of the logistic regression model is perfectly suited for the scope of the star "In_Cluster"/"Isolated" classifications.

4.2 Evaluation Metrics

Initially, the sklearn accuracy_score function was the only performance metric used for evaluating the benchmark and the solution models. The accuracy_score was calculated via the following equation:

Accuracy_score = correct predicted classifications/total number predicted classifications

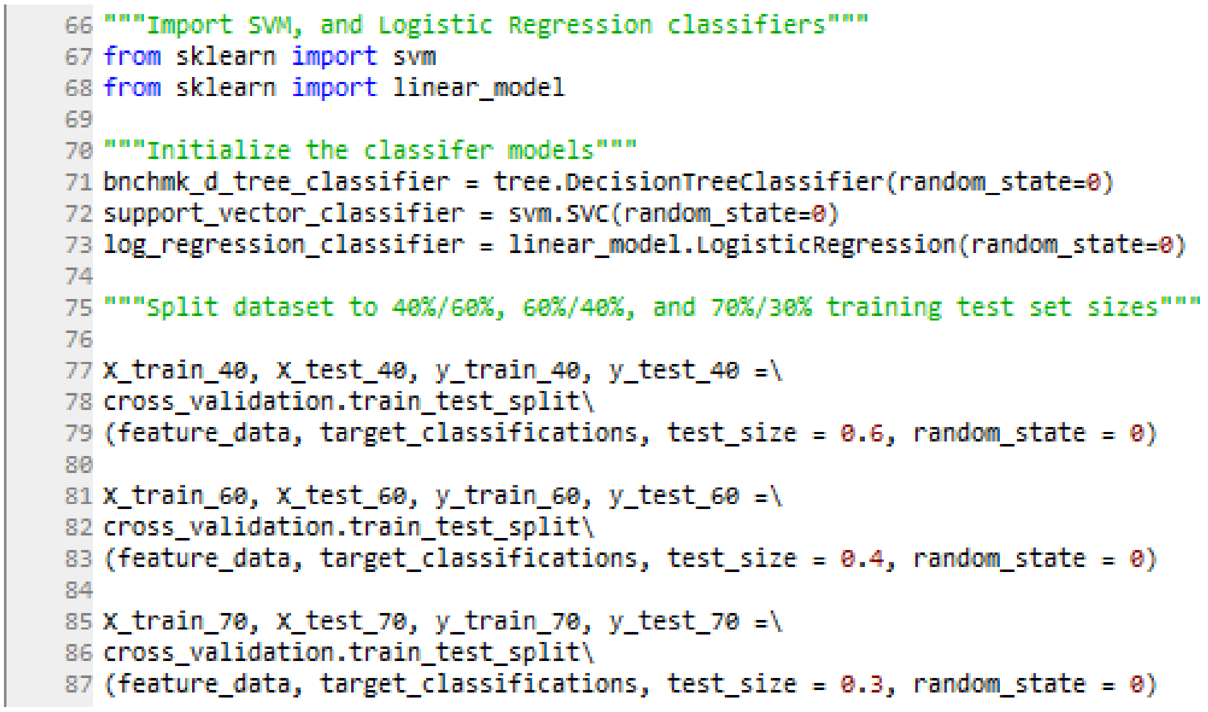
The N-Body dataset was first modeled with a decision tree algorithm and achieved a training and test accuracy of 1.000 and 0.7692, respectively. The 0.7692 algorithm test accuracy was used to evaluate the SVM and logistic regression algorithm's performance.

To better evaluate and compare the supervised learning models, the F1 scoring was also computed to assess the classifier's ability to correctly predict an outcome based on the number possible instances (recall) and addressed the how well the classifier correctly predicted the outcome based on the total number of predictions (precision) The F1 score provided a weighted average for the model's recall and precision via the following equation:

F1_Score = [2 * (precision * recall)]/(precision + recall)

5 Model Evaluation and Validation

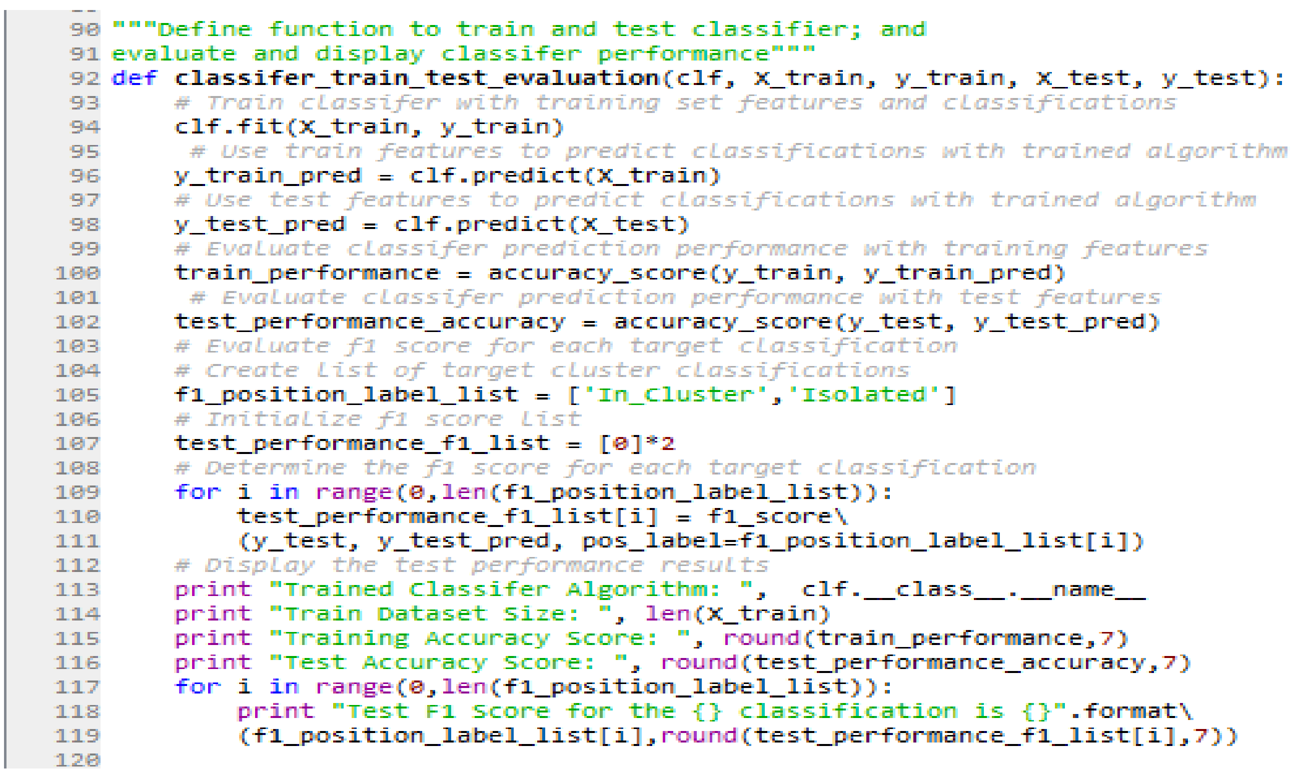
The Python code below in Figures 5-1 and 5-2 was created to train and cross validate the benchmark decision tree model; and the proposed SVM and logistic regression supervised learning algorithms.



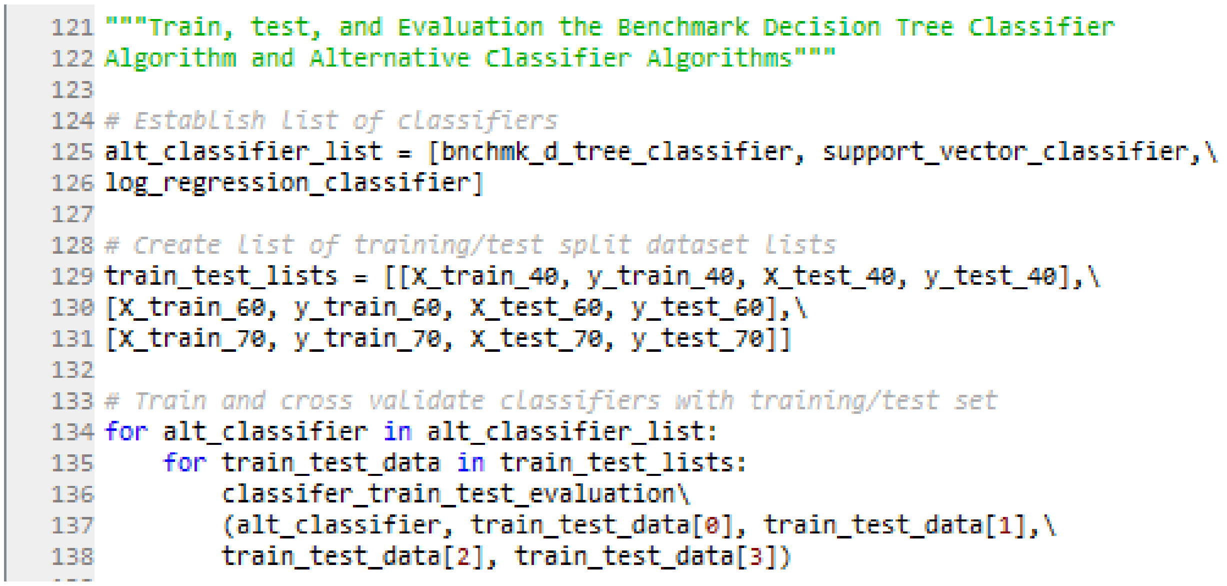
```python
"""Import SVM, and Logistic Regression classifiers"""
from sklearn import svm
from sklearn import linear_model

"""Initialize the classifer models"""
bnchmk_d_tree_classifier = tree.DecisionTreeClassifier(random_state=0)
support_vector_classifier = svm.SVC(random_state=0)
log_regression_classifier = linear_model.LogisticRegression(random_state=0)

"""Split dataset to 40%/60%, 60%/40%, and 70%/30% training test set sizes"""

X_train_40, X_test_40, y_train_40, y_test_40 =\
cross_validation.train_test_split\
(feature_data, target_classifications, test_size = 0.6, random_state = 0)

X_train_60, X_test_60, y_train_60, y_test_60 =\
cross_validation.train_test_split\
(feature_data, target_classifications, test_size = 0.4, random_state = 0)

X_train_70, X_test_70, y_train_70, y_test_70 =\
cross_validation.train_test_split\
(feature_data, target_classifications, test_size = 0.3, random_state = 0)
```

Figure 5-1 Classifier Algorithm Initialization and Dataset Cross Validation Split Python Code

The sample N-Body Star dataset was split with 40%, 60% and 70% for the classifier algorithm training; and the remaining 60%, 40%, and 70%, respectively, for model validation.



```python
"""Define function to train and test classifier; and
evaluate and display classifer performance"""
def classifer_train_test_evaluation(clf, X_train, y_train, X_test, y_test):
    # Train classifer with training set features and classifications
    clf.fit(X_train, y_train)
     # Use train features to predict classifications with trained algorithm
    y_train_pred = clf.predict(X_train)
    # Use test features to predict classifications with trained algorithm
    y_test_pred = clf.predict(X_test)
    # Evaluate classifer prediction performance with training features
    train_performance = accuracy_score(y_train, y_train_pred)
     # Evaluate classifer prediction performance with test features
    test_performance_accuracy = accuracy_score(y_test, y_test_pred)
    # Evaluate f1 score for each target classification
    # Create list of target cluster classifications
    f1_position_label_list = ['In_Cluster','Isolated']
    # Initialize f1 score list
    test_performance_f1_list = [0]*2
    # Determine the f1 score for each target classification
    for i in range(0,len(f1_position_label_list)):
        test_performance_f1_list[i] = f1_score\
        (y_test, y_test_pred, pos_label=f1_position_label_list[i])
    # Display the test performance results
    print "Trained Classifer Algorithm: ",  clf.__class__.__name__
    print "Train Dataset Size: ", len(X_train)
    print "Training Accuracy Score: ", round(train_performance,7)
    print "Test Accuracy Score: ", round(test_performance_accuracy,7)
    for i in range(0,len(f1_position_label_list)):
        print "Test F1 Score for the {} classification is {}".format\
        (f1_position_label_list[i],round(test_performance_f1_list[i],7))
```



```python
"""Train, test, and Evaluation the Benchmark Decision Tree Classifier
Algorithm and Alternative Classifier Algorithms"""

# Establish list of classifiers
alt_classifier_list = [bnchmk_d_tree_classifier, support_vector_classifier,\
log_regression_classifier]

# Create list of training/test split dataset lists
train_test_lists = [[X_train_40, y_train_40, X_test_40, y_test_40],\
[X_train_60, y_train_60, X_test_60, y_test_60],\
[X_train_70, y_train_70, X_test_70, y_test_70]]

# Train and cross validate classifiers with training/test set
for alt_classifier in alt_classifier_list:
    for train_test_data in train_test_lists:
        classifer_train_test_evaluation\
        (alt_classifier, train_test_data[0], train_test_data[1],\
        train_test_data[2], train_test_data[3])
```

Figure 5-2 Classifier Algorithm Training, Test and Scoring Python Code

Table 5-2 is a summary the classifier algorithm's accuracy and F1 metric scoring performance.

Table 5-2 Classifier Algorithm Performance Metrics Summary

		Accuracy Scoring		**F1 Scoring**	
Classifier Algorithms	Dataset Training%/Test%	Training	Test	In_Cluster	Isolated
Decision Tree (Benchmark)	40%/60%	1.0000	0.7564	0.8613	0.0000
	60%/40%	1.0000	0.7308	0.8372	0.2222
	70%/30%	1.0000	**0.7692**	**0.8657**	**0.1818**
SVM	40%/60%	0.9038	0.8333	0.9090	0.0000
	60%/40%	0.8590	0.8461	0.9167	0.0000
	70%/30%	0.8461	**0.8718**	**0.9315**	0.0000
Logistic Regression	40%/60%	0.9231	0.7949	0.8857	0.0000
	60%/40%	0.8718	0.7692	0.8696	0.0000
	70%/30%	0.8461	0.7692	0.8696	0.0000

The 70%/30% training/test dataset split provided the best overall metric performance for both the decision tree and SVM model algorithm. For the logistic regression classifier, the 40%/60% training/test dataset split had the highest overall metric performance within its training scope. When compared to the decision tree benchmark classifier algorithm, the SVM testing accuracy performed the best while the logistic regression model performed worse. The SVM had the highest F1 "In_Cluster" score with the logistic regression and decision tree algorithms being second and third, respectively. For the "Isolated" F1 scoring, the decision tree algorithm had the top score. Both the SVM and logistic regression had zeros for the "Isolated" F1 scores. Based on the overall accuracy and F1 scores, the SVM algorithm was considered the best prediction performer for N-Body star classification.

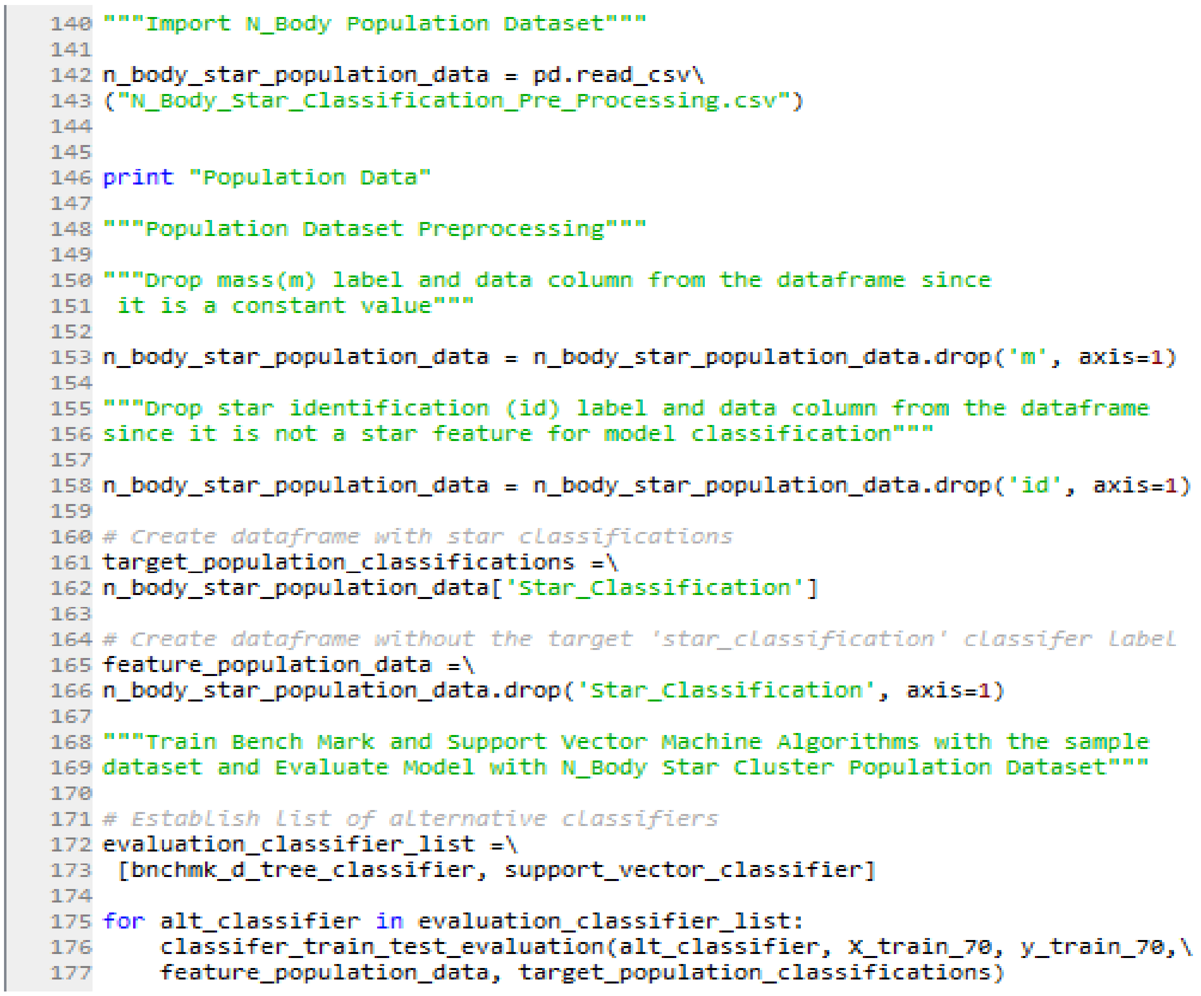
6 Conclusions

6.1 Summary

Isolated stars, like the Sun in our own solar system, are very rare in the Universe. This fact was observed by the very low number of “Isolated” stars in the N-Body Star Cluster dataset. It has been demonstrated that a dataset of star features can be pre-processed and sampled to train a supervised machine learning algorithm for accurately predicting “In_Cluster” and “Isolated” star classifications. By applying a supervised SVM machine learning algorithm to star position and velocity features, it can be trained to have higher prediction accuracies than the benchmark decision tree classifier that is typically used in astrophysics. The SVM machine learning algorithm’s test metric performance were 0.8718 accuracy score, 0.9315 In_Cluster F1 score, and 0.0000 “Isolated” F1 score.

6.2 Free-Form Visualization

To further validate the SVM’s prediction performance, the entire 64,000 N-Body star cluster population dataset was used as the test data for both the decision tree and SVM cross validation. The following Python code (see Figure 6.2-1) accomplished the N-Body star cluster pre-processing for the population dataset, model training with the sample dataset, and cross validation testing/scoring.



```python
"""Import N_Body Population Dataset"""

n_body_star_population_data = pd.read_csv\
("N_Body_Star_Classification_Pre_Processing.csv")


print "Population Data"

"""Population Dataset Preprocessing"""

"""Drop mass(m) label and data column from the dataframe since
 it is a constant value"""

n_body_star_population_data = n_body_star_population_data.drop('m', axis=1)

"""Drop star identification (id) label and data column from the dataframe
since it is not a star feature for model classification"""

n_body_star_population_data = n_body_star_population_data.drop('id', axis=1)

# Create dataframe with star classifications
target_population_classifications =\
n_body_star_population_data['Star_Classification']

# Create dataframe without the target 'star_classification' classifer label
feature_population_data =\
n_body_star_population_data.drop('Star_Classification', axis=1)

"""Train Bench Mark and Support Vector Machine Algorithms with the sample
dataset and Evaluate Model with N_Body Star Cluster Population Dataset"""

# Establish list of alternative classifiers
evaluation_classifier_list =\
 [bnchmk_d_tree_classifier, support_vector_classifier]

for alt_classifier in evaluation_classifier_list:
    classifer_train_test_evaluation(alt_classifier, X_train_70, y_train_70,\
    feature_population_data, target_population_classifications)
```

Figure 6.2-1 Classifier Algorithm Validation Python Code

The SVM algorithm was trained to have both higher accuracy scores (better than 11 percentage points) and F1 scores (almost an order of magnitude for the “Isolated” classification) metrics over the bench mark decision tree classifier (see Table 6.2-1).

Table 6.2-1 Decision Tree and SVM Classifier Algorithm Performance with the Population Dataset

	Accuracy Scoring		**F1 Scoring**	
Classifier Algorithms	Training	Test	In_Cluster	Isolated
Decision Tree (Benchmark)	1.000	0.8842	0.9385	0.0043
SVM	0.8461	**0.9992**	**0.9996**	**0.0408**

6.3 Refinement

The SVM’s F1 scoring performance for “Isolated” star classification could potentially be improved by refining the overall train/test dataset size and the split ratio. In addition, modelling optimization with GridSearch could be used to identify the best SVM parameter settings (e.g. kernel or variable *C*).

6.4 Reflection

It was interesting that the logistic regression classifier’s best performance was with the smallest training data subset. Based on this observation, evaluating training sets that are both much smaller and larger than the initial split point should be considered for future supervised learning projects. Also, it was surprising that the F1 scoring for the SVM (and logistics regression) classifier was zero with split test validation data, but achieved the best performance when the entire 64,000 N-Body star cluster dataset was used to evaluate the algorithm against the 70% training dataset. Overall, it has been demonstrated that astrophysicists can improve their chances to advance the understanding of our own solar system by identifying and monitoring developing isolated star systems by using SVM machine learning predictions.

References

[1] https://www.kaggle.com/mariopasquato/star-cluster-simulations

[2] Steven Salzberg, Rupali Chandar, Holland Ford, Sreerama K Murthy and Richard White. Decision Trees for Automated Identification of Cosmic-Ray Hits in Hubble Space Telescope Images. Publication of the Astronomical Society of the Pacific, 107: 279 - 288, March1995

[3] E M Howards. Machine Learning Algorithms in Astronomy. Macquarie University, Sydney, NSW, Australia