Prediction of Confirmed Status of Detected TESS Exoplanet Candidates using Weka Machine Learning Classification Algorithms

Leonardo Valli and Alan Zhu

10/22/24
Dr. Yilmaz
Period 5

Table of Contents

1. Introduction and Goals

The search for exoplanets, planets orbiting stars other than our sun, is a growing field in astronomy. There are currently approximately 5,800 confirmed exoplanets. There are thousands more exoplanet candidates — possible planets identified by satellites — awaiting confirmation by scientists. However, the process of confirming exoplanet candidates is costly and time-consuming, requiring several hours of data collection using valuable time on research telescopes, followed by extensive data analysis done by scientists. The astronomy community is struggling to keep pace with verifying all of these candidates.

In our project, our goal was to develop machine learning models to predict true positive and false positive exoplanet candidates. Our models should be able to quickly and accurately predict if an exoplanet candidate is a true or false positive using data collected on the candidate.

2. Dataset

2.1. Dataset Overview

We use the Transiting Exoplanet Survey Satellite (TESS) dataset, which gathers data from a satellite and telescopic search for exoplanets. The dataset includes certain characteristics of the exoplanet, the star the exoplanet orbits around ("host star"), the position of the exoplanet with respect to earth, the time the data was collected, and identifying information of the exoplanet.

The TESS satellite identifies exoplanet candidates. Because the satellite might not be accurate, the candidates require follow-up validation using ground-based telescopes to verify it is a confirmed exoplanet, or if it is a false positive.

Link to dataset: https://exofop.ipac.caltech.edu/tess/view_toi.php.

The dataset is maintained by the Exoplanet Follow-up Observing Program, NASA Exoplanet Science Institute, operated by the California Institute of Technology, Infrared Processing and Analysis Center.

2.2. Attributes

The raw dataset has 61 attributes and 1 class. At the time of dataset acquisition, there were 7217 TESS candidates (instances).

The attributes are divided into 5 categories: TESS identification, position, planet properties, stellar properties, dates.

Table 1. Description of selected important attributes.

Attribute	**Meaning**
TOI	TESS Object of Interest number - unique identifier for the exoplanet candidate
Planet Orbital Period [days]	Time the planet takes to make a complete orbit around its host star.
Planet Transit Depth [ppm]	The relative flux (brightness) decrease caused by the orbiting body transiting in front of the star.
Planet Radius [R_Earth]	Radius of the planet.
TESS Magnitude	Brightness of the planet.
Stellar Radius [R_Sun]	Radius of the host star.
TFOPWG Disposition *(class)*	TESS Follow-Up Working Group Disposition. This is how the planet has been classified after the exoplanet candidate is validated using ground-based telescopes. APC=ambiguous planetary candidate CP=confirmed planet FA=false alarm FP=false positive KP=known planet PC=planetary candidate

2.3. Missing Values

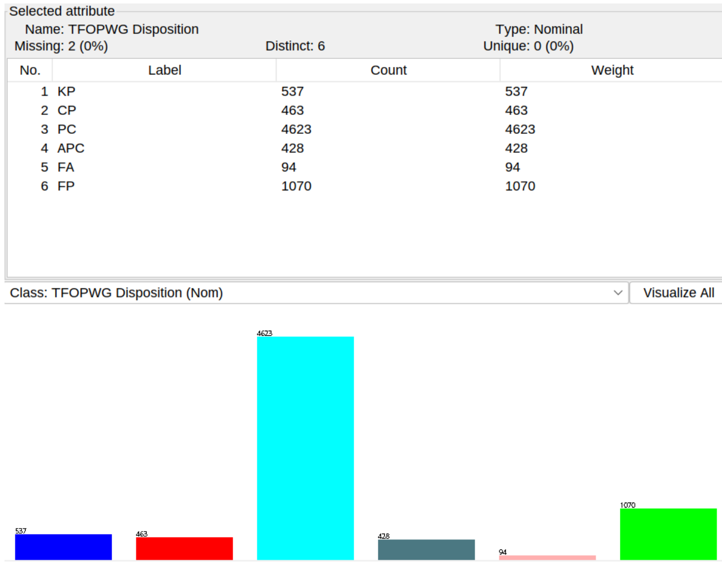
There are numerous missing values in nearly all of the columns. The number of missing values totals to 39072, meaning that ~9% of the data in this dataset is missing. These data points are determined by observations of the exoplanet host star by the TESS satellite and ground telescopes; as the dataset is still being continuously updated, so are these values, resulting in many missing values.

```
null_count = df.isnull().sum().sum()
print('Number of null values:', null_count)
```

2.4. Class

We are trying to predict the TFOPWG disposition in our classification model. Essentially, we are trying to classify, based on attributes of an exoplanet, whether it is actually an exoplanet (CP/KP) or if it is a false positive (FP).

Figure 1. Visualization of the distribution of the class variable.



As expected, there are many planetary candidates and substantial numbers of known planets and confirmed planets. There are also many false positives, which means TESS identified it as a candidate, but upon further observation it was found not to be an exoplanet. This distribution is not uniform.

Attributes found in the planetary and stellar properties attribute categories generally were found to be normally distributed when plotted.

3. Preprocessing

We did all of our data preprocessing in Python, in a Google Colab notebook.

3.1. Instance removal

First, we included in our dataset only the exoplanets that have already had a determination made to their TFOPWG status. This means we only used instances that had a class of confirmed planet (CP), known planet (KP), false positive (FP),or false alarm (FA) — we discarded PC (planetary candidate) and APC (ambiguous planetary candidate).

```
df = df.drop(df[df['TFOPWG Disposition'] == 'PC'].index)
df = df.drop(df[df['TFOPWG Disposition'] == 'APC'].index)
```

This brought us down to ~2,000 instances from ~7,000 in the original dataset. Then, we removed rows with no class. Next, we considered CP and KP as true positive exoplanet detections and FP and FA as false positives. We transformed the class into a boolean: true means it is an actual exoplanet, false means it is not an exoplanet.

```
df.dropna(subset=['TFOPWG Disposition'], inplace=True)
df['TFOPWG Disposition'].replace(['KP', 'FA'], ['CP', 'FP'], inplace=True)
df['TFOPWG Disposition'].replace(['CP', 'FP'], ['True', 'False'], inplace=True)
```

3.2. Attribute removal

We then removed attributes that we know intuitively have no predictive power. We also removed attributes that are based on calculations by astronomers intended to predict whether the candidate is an actual exoplanet (our model should operate only on the raw data).

Table 2. Attributes removed on grounds of no predictive power.

<table>
<tr><th>Attribute</th><th>Reason removed</th></tr>
<tr><td>TOI</td><td rowspan="4">Identifiers</td></tr>
<tr><td>TIC ID</td></tr>
<tr><td>Previous CTOI</td></tr>
<tr><td>Pipeline Signal ID</td></tr>
<tr><td>Master</td><td rowspan="8">Derived by astronomers to prioritize/predict if candidate is an actual exoplanet</td></tr>
<tr><td>SG1A</td></tr>
<tr><td>SG1B</td></tr>
<tr><td>SG2</td></tr>
<tr><td>SG3</td></tr>
<tr><td>SG4</td></tr>
<tr><td>SG5</td></tr>
<tr><td>TESS Disposition</td></tr>
<tr><td>Planet Name</td><td>Only for known planets.</td></tr>
<tr><td>Source</td><td rowspan="2">Which pipeline processing code detected the candidate.</td></tr>
<tr><td>Detection</td></tr>
<tr><td>RA</td><td rowspan="5">Location of the planets in the sky</td></tr>
<tr><td>Dec</td></tr>
<tr><td>PM RA (mas/yr)</td></tr>
<tr><td>PM Dec (mas/yr)</td></tr>
<tr><td>Sectors</td></tr>
<tr><td>Epoch (BJD)</td><td rowspan="4">Dates</td></tr>
<tr><td>Date TOI Alerted (UTC)</td></tr>
<tr><td>Date TOI Updated (UTC)</td></tr>
<tr><td>Date Modified</td></tr>
<tr><td>Comments</td><td>Comments on the candidate</td></tr>
<tr><td>TESS Mag err</td><td rowspan="2"></td></tr>
<tr><td>PM RA err (mas/yr)</td></tr>
</table>

Epoch (BJD) err	Error in attribute value
Period (days) err	
Duration (hours) err	
Depth (mmag) err	
Depth (ppm) err	
Planet Radius (R_Earth) err	
Stellar Distance (pc) err	
Stellar Eff Temp (K) err	
Stellar log(g) (cm/s^2) err	
Stellar Radius (R_Sun) err	
Stellar Metallicity err	
Stellar Mass (M_Sun) err	

This brought us down to 22 attributes, from the original 62.

3.3. Missing values

For columns that had greater than 75% values missing, we removed the attribute. Otherwise, we filled in missing values in the column with the mean of the attribute values. Using this method, we removed one attribute (stellar metallicity) and filled in using the mean for 10 attributes.

```
for col in df.columns:
  num_missing = df[col].isnull().sum()

  percent_missing = (num_missing / len(df)) * 100

  print(f"Column '{col}': {num_missing} missing values, {percent_missing:.2f}% missing.")

  # If percentage of missing values is greater than 75%, drop the column
  if percent_missing > 75:
      df.drop(columns=[col], inplace=True)
      print(f"Column '{col}' dropped due to high percentage of missing values.")
  elif percent_missing > 0:
    mean_of_col = df[col].mean()
    df[col].fillna(mean_of_col, inplace=True)
    print(f"Missing values in column '{col}' filled with mean value {mean_of_col}")
```

3.4. Normalization

We normalized all of the attributes using z-score normalization due to the presence of outliers in several of our columns.

```
for col in df.drop(columns=['TFOPWG Disposition'], axis=1).columns:
  mean = df[col].mean()
  std = df[col].std()

  for row in df.index:
    df.loc[row, col] = (df.loc[row, col]-mean)/std

df.head()
```

3.5. Train-Test-Validation split

We initially performed a train-test-validation split with a 70-15-15 ratio. However, we soon realized that Weka does not easily support the use of multiple separate sets, so in the end we did not use train-test-validation datasets. We used k-fold cross validation instead.

```
from sklearn.model_selection import train_test_split

X = df.drop(columns = ['TFOPWG Disposition'], axis=1)
y = df['TFOPWG Disposition']

X_train, X_temp, y_train, y_temp = train_test_split(X, y, stratify=y,
test_size=0.3)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp,
stratify=y_temp, test_size=0.5)
```

3.6. Final datasets

Here are links to our final datasets.
Preprocessed dataset: TOI-preprocessed.csv
Training dataset: train.csv
Testing dataset: test.csv
Validation dataset: validation.csv

4. Data Mining

4.1. Attribute selection

We performed attribute selection using Weka. We used four selection algorithms, and selected a set of attributes by ourselves.

4.1.1. Correlation

We used one of the most basic attribute selection methods, using the Pearson correlation coefficient to rank attributes based on correlation with the class.

```
Ranked attributes:
 0.3761   14 Planet Equil Temp (K)
 0.3109   20 Stellar Mass (M_Sun)
 0.2617   17 Stellar Eff Temp (K)
 0.2602   18 Stellar log(g) (cm/s^2)
 0.2548    6 Imaging Observations
 0.2363    4 Time Series Observations
 0.2334    5 Spectroscopy Observations
 0.1977   16 Stellar Distance (pc)
 0.1948   13 Planet Insolation (Earth Flux)
 0.1399   19 Stellar Radius (R_Sun)
 0.1342   15 Planet SNR
 0.0877   12 Planet Radius (R_Earth)
 0.0653    9 Duration (hours)
 0.0409   11 Depth (ppm)
 0.04      2 TSM
 0.0384    3 Predicted Mass (M_Earth)
 0.0296   10 Depth (mmag)
 0.0185    1 ESM
 0.0143    8 Period (days)
 0.012     7 TESS Mag
```

4.1.2. CFS Subset

The correlation-based feature subset (CFS) attribute selection method is similar to the correlation algorithm, but also takes into account the intercorrelation rate of the selected attribute subsets. This algorithm is unique because it also recommends the number of attributes to use.

```
Selected attributes: 4,5,6,8,10,12,13,14,15,16,18,20 : 12
                     Time Series Observations
                     Spectroscopy Observations
                     Imaging Observations
                     Period (days)
                     Depth (mmag)
                     Planet Radius (R_Earth)
                     Planet Insolation (Earth Flux)
                     Planet Equil Temp (K)
                     Planet SNR
                     Stellar Distance (pc)
                     Stellar log(g) (cm/s^2)
                     Stellar Mass (M_Sun)
```

4.1.3. Gain Ratio

This algorithm evaluates the worth of an attribute by using the information gain (entropy) associated with including an attribute.

```
0.1125    13 Planet Insolation (Earth Flux)
0.0846    14 Planet Equil Temp (K)
0.0823     4 Time Series Observations
0.0786    20 Stellar Mass (M_Sun)
0.0771    18 Stellar log(g) (cm/s^2)
0.0621    12 Planet Radius (R_Earth)
0.0609    19 Stellar Radius (R_Sun)
0.0592    17 Stellar Eff Temp (K)
0.0555    16 Stellar Distance (pc)
0.0519    10 Depth (mmag)
0.0519    11 Depth (ppm)
0.0505     8 Period (days)
0.0444     5 Spectroscopy Observations
0.0409    15 Planet SNR
0.0385     6 Imaging Observations
0.0327     1 ESM
0.0324     3 Predicted Mass (M_Earth)
0.0222     7 TESS Mag
0          9 Duration (hours)
0          2 TSM
```

4.1.4. Relief

The Relief-F algorithm evaluates the worth of an attribute by choosing an instance, finding the closest same-class and different-class instances to it, and updating a weight for that attribute to capture the association between the attribute and the class and inter-attribute relationships.

```
Ranked attributes:
 0.027226   14 Planet Equil Temp (K)
 0.015745   20 Stellar Mass (M_Sun)
 0.011189    4 Time Series Observations
 0.010021    7 TESS Mag
 0.008191   18 Stellar log(g) (cm/s^2)
 0.007213    5 Spectroscopy Observations
 0.006352    9 Duration (hours)
 0.006173    6 Imaging Observations
 0.005643   13 Planet Insolation (Earth Flux)
 0.004976   17 Stellar Eff Temp (K)
 0.004535   16 Stellar Distance (pc)
 0.003795    3 Predicted Mass (M_Earth)
 0.002523   15 Planet SNR
 0.002195    1 ESM
 0.001401   11 Depth (ppm)
 0.00132    12 Planet Radius (R_Earth)
 0.00099    10 Depth (mmag)
 0.000871   19 Stellar Radius (R_Sun)
 0.000841    2 TSM
 0.000662    8 Period (days)
```

4.1.5. Self-selected attribute set

We selected a set of attributes ourselves. We focused on including planetary and host star scientific characteristics. We selected these attributes prior to running the other selection algorithms, so it is “blind” to the other selection methods.

```
Depth (ppm)
Planet Radius (R_Earth)
Planet Insolation (Earth Flux)
Planet Equil Temp (K)
Planet SNR
Stellar Eff Temp (K)
Stellar log(g) (cm/s^2)
Stellar Radius (R_Sun)
Stellar Mass (M_Sun)
Time Series Observations
Spectroscopy Observations
Imaging Observations
```

4.1.6. Number of selected attributes

We proceeded with the 12 most highly-ranked attributes from each method. We chose 12 because there is generally a ranking gap near the 12th-ranked attribute, and also the CFS algorithm chose to use 12 attributes.

4.1.7. Train-test-validation datasets for selected attributes

We created a train, test, and validation dataset for each of the five attribute subsets. Because in the end we used *k*-fold cross validation, we also created a dataset with all of the instances for each of the five attribute subsets.

```python
# The values of the dictionary hold the names of the attributes in the attribute subsets
ATTRIBUTE_SELECTIONS = {
    "correlation": correlation_attributes,
    "gain_ratio": gain_ratio_attributes,
    "reliefF": reliefF_attributes,
    "cfs_subset": cfs_subset_attributes,
    "self_selected": self_selected_attributes,
}

MAX_ATTRIBUTES = 12

for name, attribute_set in ATTRIBUTE_SELECTIONS.items():
    attribute_set = attribute_set.strip().split("\n")[:MAX_ATTRIBUTES]
    attribute_set = [x.strip() for x in attribute_set]
    attribute_set.append("TFOPWG Disposition")
    print(f"{name}: {attribute_set}")

    # Save new train, test, validation sets using only the selected attributes
    all = df[attribute_set]
    train = X_train[attribute_set]
    test = X_test[attribute_set]
    val = X_val[attribute_set]

    all.to_csv(f"all-{name}.csv", index=False)
    train.to_csv(f"train-{name}.csv", index=False)
```

```
    test.to_csv(f"test-{name}.csv", index=False)
    val.to_csv(f"validation-{name}.csv", index=False)
```

4.2. Classification Algorithms

We then selected four Weka classification models to use in order to demonstrate the performances of varying kinds of algorithms on the classification of our TESS dataset. The models selected were as follows.

4.2.1. OneR

The OneR classification algorithm is one that loops through each attribute in the dataset, creating a rule that ties each value (for discrete attribute values) or range of values (for continuous attribute values) to a specific class value. In the end, the rule of the feature with the lowest error is taken as the final classification rule. Due to its simplicity, the OneR classification algorithm was predicted to have poor results when tested on our dataset. Basic pseudocode for the OneR classifier looks something like this:

```
For every attribute in the training data:
   For every value for that attribute:
       For every class value:
          Calculate error for the rule value -> class value
       Remember the best performing rule for this value
   Remember the best performing rules for this attribute
Final rules = best performing rules from one attribute
```

4.2.2. NaiveBayes

The NaiveBayes classification algorithm is another simple classification algorithm that we anticipated would not work well on our fairly complex data. The algorithm begins by finding the percentage of training instances that have each class value. Then, for every class value in the training set, the likelihood for an instance of that class to have each attribute value for each attribute is computed. Finally, predictions are made by finding the class whose percentages of occurrence multiplied by each of the likelihoods from each of its attribute values is the highest. That way, not only would more commonly-occurring classes be predicted more often, but the correlation between certain attribute values and class values is also accounted for. Basic pseudocode for the NaiveBayes classifier looks something like this:

```
For every class value:
   Find percentage of training data that is that class
   For every attribute:
       For every value of that attribute:
          Calculate likelihood of that class having that value

Classification = max(percentage * likelihoods)
```

4.2.3. RandomForests

Random Forests, which is an ensembling method, was the next classification model we tested. An ensembling machine learning model is one that takes a multitude of base models, and combines their predictions in some way to come up with the final prediction of the model. In the case of Random Forests, the base models that are used are Decision Trees, which classify instances by making multiple separations, or branches based on rules created with the attribute values in the training dataset. The Random Forests model creates an n number of Decision Trees, has each Decision Tree make a prediction for the instance, then makes a majority-rule decision to finally classify the instance. As the name suggests, the Random Forests algorithm introduces some randomness, leading to small variations in its construction and performance between trials. We had high confidence in the abilities of Random Forests to predict the classes of our data, as ensembling methods typically work well, especially with a solid base model like Decision Trees. Basic pseudocode for the RandomForests classifier looks something like this:

```
For i in range(n):
   Create decision tree
For i in range(n): #Classification
   Have the i-th decision tree make a prediction on the test instance
Classification = Class with the most votes from the decision trees
```

4.2.4. MultilayerPerceptron

Our final classification algorithm that we used was a Multilayer Perceptron, a type of feedforward neural network. A neural network of this kind is built and trained by looping through the training data, making predictions, computing the loss (how bad the predictions were), then adjusting the weights in the neural network accordingly via gradient descent. Weka's Multilayer Perceptron is a two-layer neural network. Our previous experience with neural networks led us to believe that these would perform well on our data. Basic pseudocode for the Multilayer Perceptron classifier looks something like this:

```
Initialize random base weights for network
For every instance in the training set:
    Make prediction using current weights
    Calculate loss (e.g., Mean Squared Error) by comparing prediction to actual class
    For every weight in the NeuralNet:
       Update via gradient descent
```

5. Results and Discussion

Here are the results from our total of 20 attribute selection algorithm/classification method pairs, tested via Weka's k-fold cross validation testing, with the default k value of 10. We have detailed the accuracies, area under ROC curve values, and confusion matrices for each of the pairs tested in this way.

Accuracy

		Attribute Selection Method				
		Correlation Attribute Eval	Gain Ratio Attribute Eval	ReliefF Attribute Eval	Cfs Subset Eval	Our Choice of Attribute Subset
Classification Method	OneR	66.54%	66.54%	67.05%	66.54%	66.54%
	NaiveBayes	65.53%	63.22%	67.51%	62.80%	65.99%
	RandomForests	87.38%	86.65%	86.83%	87.89%	86.46%
	MultilayerPerceptron	79.99%	79.02%	82.39%	80.73%	80.45%

ROC Area

		Attribute Selection Method				
		Correlation Attribute Eval	Gain Ratio Attribute Eval	ReliefF Attribute Eval	Cfs Subset Eval	Our Choice of Attribute Subset
Classification Method	OneR	0.665	0.665	0.670	0.665	0.665
	NaiveBayes	0.815	0.803	0.801	0.803	0.820
	RandomForests	0.945	0.937	0.941	0.949	0.943
	MultilayerPerceptron	0.887	0.869	0.886	0.884	0.881

Correlation Attribute Eval

OneR

		Predicted	
		True	False
Actual	True	663	337
	False	387	777

NaiveBayes

		Predicted	
		True	False
Actual	True	912	88
	False	658	506

RandomForests

		Predicted	
		True	False
Actual	True	873	127
	False	146	1018

MultilayerPerceptron

		Predicted	
		True	False
Actual	True	794	206
	False	227	937

Gain Ratio Attribute Eval

OneR

		Predicted	
		True	False
Actual	True	663	337
	False	387	777

NaiveBayes

		Predicted	
		True	False
Actual	True	925	75
	False	721	443

RandomForests

		Predicted	
		True	False
Actual	True	871	129
	False	160	1004

MultilayerPerceptron

		Predicted	
		True	False
Actual	True	796	204
	False	250	914

ReliefF Attribute Eval

OneR

		Predicted	
		True	False
Actual	True	665	335
	False	378	786

NaiveBayes

		Predicted	
		True	False
Actual	True	863	137
	False	566	598

RandomForests

		Predicted	
		True	False
Actual	True	867	133
	False	152	1012

MultilayerPerceptron

		Predicted	
		True	False
Actual	True	854	146
	False	235	929

Cfs Subset Eval

OneR

		Predicted	
		True	False
Actual	True	663	337
	False	387	777

NaiveBayes

		Predicted	
		True	False
Actual	True	906	94
	False	711	453

RandomForests

		Predicted	
		True	False
Actual	True	873	127
	False	135	1029

MultilayerPerceptron

		Predicted	
		True	False
Actual	True	808	192
	False	225	939

Our Choice of Attribute Subset

OneR

		Predicted	
		True	False
Actual	True	663	337
	False	387	777

NaiveBayes

		Predicted	
		True	False
Actual	True	921	79
	False	657	507

RandomForests

		Predicted	
		True	False
Actual	True	866	134
	False	159	1005

MultilayerPerceptron

		Predicted	
		True	False
Actual	True	808	192
	False	231	933

Generally, it can be seen that our predictions of the relative performances of the Weka machine learning models we used were correct. Random Forests and Neural Networks worked well, generally achieving accuracies of more that 80 percent. OneR and Naive Bayes classifiers resulted in accuracies of 60-70 percent when tested with Weka's k-fold cross validation.

The clear best-performing model during our tests was the Random Forests classifier model, achieving a highest accuracy of 87.89% with the CfsSubsetEval Attribute Selection method, followed by CorrelationAttributeEval with a 87.38% accuracy. We were surprised to see such a high accuracy coming from a Pearson Correlation attribute evaluator, since Pearson Correlation only determines linear correlation between attributes.

We know that Random Forests was our best model, because not only did it have the highest accuracy by 5%, but it also led in other performance metrics such as precision, recall, and area under ROC curve. Random Forests paired with CfsSubsetEval had an area under curve value of 0.949. Precision and Recall for the True class were also the highest for the pairing between CfsSubsetEval and Random Forests, with values of 873/(873+135) = 0.866 and 873/(873+127) = 0.873, respectively. Precision and Recall for the False class for this pairing were 1029/(1029+127) = 0.89 and 1029/(1029+135) = 0.884, respectively.

There were a few observations that we made about the results that we found interesting. Firstly we saw that all but one of the OneR models performed exactly the same. This made sense, as the OneR model simply picks an attribute to create a rule out of. However, we were surprised to see that one of the models performed slightly better than the rest, even though they all chose the same attribute (Planet Equil Temp (K)) to form a rule out of. In addition, we were surprised to see that the Random Forests classification model built off of the attributes chosen by CorrelationAttributeEval had the same precision (0.873) for the True class as the highest performing pair. Finally, while CfsSubsetEval resulted in the best performance for our best model, RandomForests, we noted that the ReliefF attribute selection algorithm resulted in the the best accuracies for each of the other three classification models we selected.

6. Conclusions

Our goal to create a fast and accurate classification model for the TESS dataset was a success. Using only 12 of the attributes collected at the moment of the detection of a potential exoplanet by a satellite, our Random Forests model can confirm whether the detected planet is an actual exoplanet or if it is a false alarm with an accuracy of 87.89%. Further research could be done in testing other more complicated models on this data in order to increase performance even further, but the performance we achieved is satisfactory for the purpose it would serve. Astronomers may now have a prediction from our model to back up their confirmed analysis-based conclusions of whether the detected planets are truly planets or not. We learned a lot about the use of attribute selection algorithms, various models used for classification, and the reporting of results using performance metrics like accuracy, precision, recall, and ROC area under curve measures. We also got practice writing a semi-professional level report of our work, and we enjoyed the process of laying our work out on paper to be seen by others.

7. Project Contributions

Alan found the dataset, and he knows the most about it, which resulted in him playing a big role in deciding which attributes in the dataset should not be kept. With Leo's experience with machine learning with python, he contributed a lot to the preprocessing done in our code. Leo chose and ran the attribute selection algorithms, and Alan created all the datasets we needed, and chose and ran the classification models. Leo created the tables for the accuracy and confusion matrix results, and Alan created the table for the ROC curve results. Alan wrote the abstract, introduction, dataset, attribute selection, and conclusions sections. Leo wrote the rest of the methodology section, results and discussion section, conclusions, and contributions section.