الكلية الجامعية بالجموم
Al Jumum University College

المملكة العربية السعودية
وزارة التعليم
جَامِعَةُ أمّ القُرىٰ

AI PROJECT

The prediction of Exoplanets using KNN

Student's names:

Raghad Mohammed Al-Lehyani (439004217)

Arwa Abdullah Al-Harbi (439004457)

Manar Hasan Al-Hutheifi (439009537)

Atheer khaled Al-Otaibi (439012694)

Reham Abdullah Al-Solami (439000913)

Instructor name:
Dr. Mashael Al-Luhaybi

Course name and number:
Artificial Intelligence - 23164402-4

11 November, 2021

Table of Contents:

Abstract

This project predicts Exoplanets that orbit around other stars in deep space beyond our solar system. It is not easy for astronomers to see them through telescopes. So, they use other methods to discover and study these distant planets. One way to hunt exoplanets is through indirect methods: measuring the dimming of a star that has a planet passed in front of it, called the transit method. So, based on this data which is the measuring of regular 'dimming' of the flux (the light intensity), we used it to predict that there may be an orbiting body around the star (Exoplanet) using the machine learning technique K-Nearest Neighbor (KNN). After building the model, we achieved an accuracy of **96%**.

Introduction

In 2009, NASA sent a spacecraft called Kepler to search for exoplanets. Kepler looked for it in a large-scale of sizes and orbits. Besides, these planets orbited around stars that varied in size and temperature.
So far, thousands of planets have been found by Kepler using the transit method. When a planet passes in front of a star, it is called a transit. So, when a planet transits in front of the star, it blocks out some of the star's light. Exoplanets are extremely dim. They do not emit light. They shine only with light reflected from their local stars. That means a star will look less bright when the planet passes in front of it. The dataset we used in our project is derived from this observation made by the NASA Kepler spacecraft. It describes the change in flux (light intensity) of many thousand stars at different times, representing the attributes. Each star has a binary label of 2 or 1. Label 2 indicates that the star is proved to have at least one exoplanet in its orbit, and label 1 indicates a non-exoplanet star. Since the labels are discrete, We will work on one of the supervised learning algorithms of classification type, the KNN algorithm. The final model will help astronomers discover and predict if a star has a planet that orbits around it or not and shows that exoplanets are very common in the universe.

- Attributes: FLUX(1) - FLUX(3197):
Which is the light intensity recorded for each star at a different point in time.

- LABELS:
2 is an exoplanet star.
1 is a non-exoplanet star.

The link of the dataset:
https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data
Which is: **exoTrain.csv**

Technique

In order to accomplish this system or make a predictable system by implementing some techniques, The first step that must be implemented is to identify a suitable dataset to solve the problem mentioned from reliable sources. The dataset for the implementation of this project is taken from NASA.
Next, we looked for software packages to help us solve the problem. It turns out that the Sklearn library can provide this help which includes many ready-made methods in machine learning, and therefore it includes a way to deal with the system to be developed.
The environment in which the automated system was developed was in a tool belonging to Google called Collab, a cloud platform from which we can download the required software packages. We will be using machine learning here to make an intelligent model.
The conclusions to be drawn here are to determine whether there is a planet orbit around a star? Or not? In other words, the results depend on the classification, which means we will work on Supervised Learning, a type of machine learning that contains many algorithms that will help prediction.
We have chosen the K-Nearest Neighbor algorithm to implement this project.

Algorithm

K-Nearest Neighbor (KNN) is a supervised machine learning algorithm, and it is a non-parametric method. It is additionally one of the best-known algorithms used for classification. The precept is that acknowledged information is organized in a space described via the selected features. When new data is furnished to the algorithm, the algorithm will differentiate the instructions of the k closest data to decide the class of the new data. The distance can be of any type, Euclidean or Manhattan, Etc. The analysis consists of the commentary of the affect parameters such as distance and classification policies on classification results. The essential advantage of the KNN classification is its simplicity, and it is additionally an environment-friendly method. However, computation instances can be lengthy with massive databases, notwithstanding its efficiency. The willpower of the wide variety of neighbors to use (k) requires trial and error. The algorithm is vulnerable to outliers, which can strongly affect its efficiency. Its principal negative aspects are that it is pretty computationally inefficient, and it is challenging to choose the 'correct' value of K, Poor overall performance on imbalanced data. However, the advantages of this algorithm are that it is versatile to specific calculations of proximity, it is very intuitive, and it is a reminiscence primarily based approach. In summation, KNN is a lazy learner. It works to assign a label to an unlabeled point primarily based on the proximity of the unlabeled point to all the different nearest labeled points.

Explanations

Here we will see how we can use Python's Scikit-Learn library to implement the KNN algorithm and build our model.

-1

Importing libraries:

The first and the essential step is to import the Python libraries we will use in this code, which features various functions like numpy as np for numerical calculation, array, and matplotlib. pyplot as plt to print the graph and show the output results, pands as pd for data manipulation operations. Still, in our code, we will need to install a pip implearn for using SMOTE Algorithm, Which we will see where we will need it later.

After that, from google.collab, we will import the drive, So that we can access the file that contains the dataset there, then using pd.read_csv() function that returns a new Data Frame with the data and labels from the file data.csv then we store it in the DS variable so we can use it. We use the dataset.shape & dataset.head methods to see the data's shape, size, and general details.

Code + Text

```
#1
#Importing libraries
!pip install imblearn
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#Importing the dataset from google drive
from google.colab import drive
drive.mount('/content/gdrive')
DS = pd.read_csv('gdrive/My Drive/AI LABS/Exoplanets_Dataset.csv')
#Shape of the dataset
print ("Dataset Shape: ", DS.shape)
#First 5 rows from the dataset
print ("Head of the Dataset: ")
print (DS.head())
```

-2

Separating the dataset into attributes and labels:

At first, we split the data into attributes and labels. We set the data location for both attributes and labels using iloc - an index always based on an integer. Attributes contain all rows and all the columns except the first. The label also contains all rows and the first column only.

Code + Text

```
#2
#Separating the dataset into attributes and labels
attributes = DS.iloc[:, 1:].values
label = DS.iloc[:, 0].values
```

Then we call a library in Python sklearn.model_selection that is used to manipulate, split, and aggregate data. One of the functions in this library is train_test_split, which divides the data into two parts, training and test sets. We split **70%** of the data into the training set while **30%** into the test set. Testing the model is essential to see how our model behaves, and this splitting helps us avoid the occurrence of overfitting.
We define a random_state to initialize a random number generator, which can be any integer to ensure that the results will be constant whenever we run(execute) code, and the train test datasets will have the same values each time.

```
Code  + Text

#Splitting the dataset into training and testing
from sklearn.model_selection import train_test_split
attributes_train, attributes_test, label_train, label_test = \
train_test_split(attributes, label, test_size=0.30,random_state = 100)
```

-3

Performing SMOTE:

Here we performed the synthetic minority oversampling (SMOTE) technique to achieve better-classifying performance since the dataset is imbalanced.
We use the Counter method imported from the collections library to count elements present in the container, which is the label_train before and after OverSampling.
We define a SMOTE instance with a default parameter that will create many new synthetic examples in the minority class (2), which balances it and then fits the dataset using the fit_resample() method. This will resample, create a new transformed version of the dataset.

```
Code  + Text

#3
#Performing synthetic minority oversampling
from imblearn.over_sampling import SMOTE
from collections import Counter
c = Counter(label_train)
print('Before OverSampling: ' , c)
s = SMOTE()
attributes_train_sm, label_train_sm = \
s.fit_resample(attributes_train, label_train)
c = Counter(label_train_sm)
print('After OverSampling: ' , c)
```

-4

Training the KNN and Prediction:

To train the KNN algorithm and make predictions, we must import the KNeighborsClassifier class from the sklearn.neighbours.

Next, this class is initialized with one parameter n_neigbours, the value of K. There is no outstanding value for K, and it is selected after testing and evaluation. However, We randomly chose 10 to be the value of K.

Instantiate the class, and call the fit() method, which fits our training set.

The latest step is to make predictions on our testing set attributes, which has **30%** samples of the total dataset that was hidden from the algorithm using predict() method.

```
+ Code  + Text

#4
#Training the KNN
from sklearn.neighbors import KNeighborsClassifier
prediction = KNeighborsClassifier(n_neighbors = 10)
prediction.fit(attributes_train_sm, label_train_sm)
#Prediction
label_pred = prediction.predict(attributes_test)
```

-5

Evaluating:

We evaluate the algorithm using confusion matrix, precision, recall metrics and accuracy. From sklearn.metrics, we import the confusion_matrix, classification_report and accuracy_score methods to calculate these metrics and get accuracy, passing them the testing set labels and the predicted values by KNN then print them to see the performance of the algorithm.

```
Code  + Text

#5
#Evaluating
from sklearn.metrics import classification_report, confusion_matrix , \
accuracy_score
print("Confusion Matrix: ")
print(confusion_matrix(label_test, label_pred))
print("Classification Report: ")
print(classification_report(label_test, label_pred))
#Accuray of the model
print("Accuracy: ", accuracy_score(label_test, label_pred))
```

-6

The best value for K:

We can not find which value of K yields the best results. We randomly chose 10 as the K value, and it just happened to result in 96% accuracy. one way to help us find the best value of K is to plot the graph of the K value and the corresponding error rate for the dataset. We will plot the mean error to the predicted values of the test set for all the K values between 1 and 20. First, we will calculate the mean of error for all the predicted values where K ranges from 1 and 20. To do this, we will execute a loop from 1 to 20. In every iteration, the mean error for predicted values of the test set is calculated, and the result is appended to the error list. After this step, we will plot the error values against K values. Moreover, determine figure size, colors, font type, and the plot's title. And the title of each of x label and y label.

Code + Text

```
#6
#Comparing Error Rate with the K Value
error = []
#Calculating error for K values between 1 and 20
for i in range(1, 20):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(attributes_train, label_train)
    pred_i = knn.predict(attributes_test)
    error.append(np.mean(pred_i != label_test))
plt.figure(figsize=(12, 6))
plt.plot(range(1, 20), error, color='magenta', linestyle='-.', marker='v',
         markerfacecolor='yellow', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Experimental results

In general, the results from training and testing data were somewhat accurate.
First, we print the data shape to explore our dataset, and it turns out that we have 5087 rows and 3198 columns. After that, we print the head of the dataset, which will show the first 5 records from our dataset. After that, we printed the labels before performing the SMOTE method, And most of them contained label 1 and very little of label 2, so this will cause a miss classification problem.

In order to solve the problem, we implemented the SMOTE method, then printed the labels again, and the percentage of label 2 increased due to the default data, which balanced the dataset.

```
Code  + Text

Dataset Shape:  (5087, 3198)
Head of the Dataset:
   LABEL   FLUX.1   FLUX.2   FLUX.3  ...  FLUX.3194  FLUX.3195  FLUX.3196  FLUX.3197
0      2    93.85    83.81    20.10  ...      39.32      61.42       5.08     -39.54
1      2   -38.88   -33.83   -58.54  ...     -11.70       6.46      16.00      19.93
2      2   532.64   535.92   513.73  ...     -11.80     -28.91     -70.02     -96.67
3      2   326.52   347.39   302.35  ...      -8.77     -17.31     -17.35      13.98
4      2 -1107.21 -1112.59 -1118.95  ...    -399.71    -384.65    -411.79    -510.54

[5 rows x 3198 columns]
Before OverSampling:  Counter({1: 3535, 2: 25})
After OverSampling:  Counter({1: 3535, 2: 3535})
```

After that, the confusion matrix is printed as it appears in the figure. We can notice that the number of correct predictions in class 1 is very high, which is 1461, while the false predictions are 54.
In class 2, the number of false predictions is very high, which is 7, while the correct predictions are 5 because of the imbalanced dataset, there is still a problem of miss classification, but the number of correct predictions of class 2 increased after performing the SMOTE method because before adding it we got **0%** predictions of class 2.

```
Code  + Text

Confusion Matrix:
[[1463   52]
 [   7    5]]
Classification Report:
              precision    recall  f1-score   support

           1       1.00      0.97      0.98      1515
           2       0.09      0.42      0.14        12

    accuracy                           0.96      1527
   macro avg       0.54      0.69      0.56      1527
weighted avg       0.99      0.96      0.97      1527

Accuracy:  0.9613621480026195
```

Under the confusion matrix, the classification report is printed, and it shows the results as follows:

Label 1:

It has 100% precision & 97% of recall & 98% of f1-score, and all of these are out of 1515 labels.

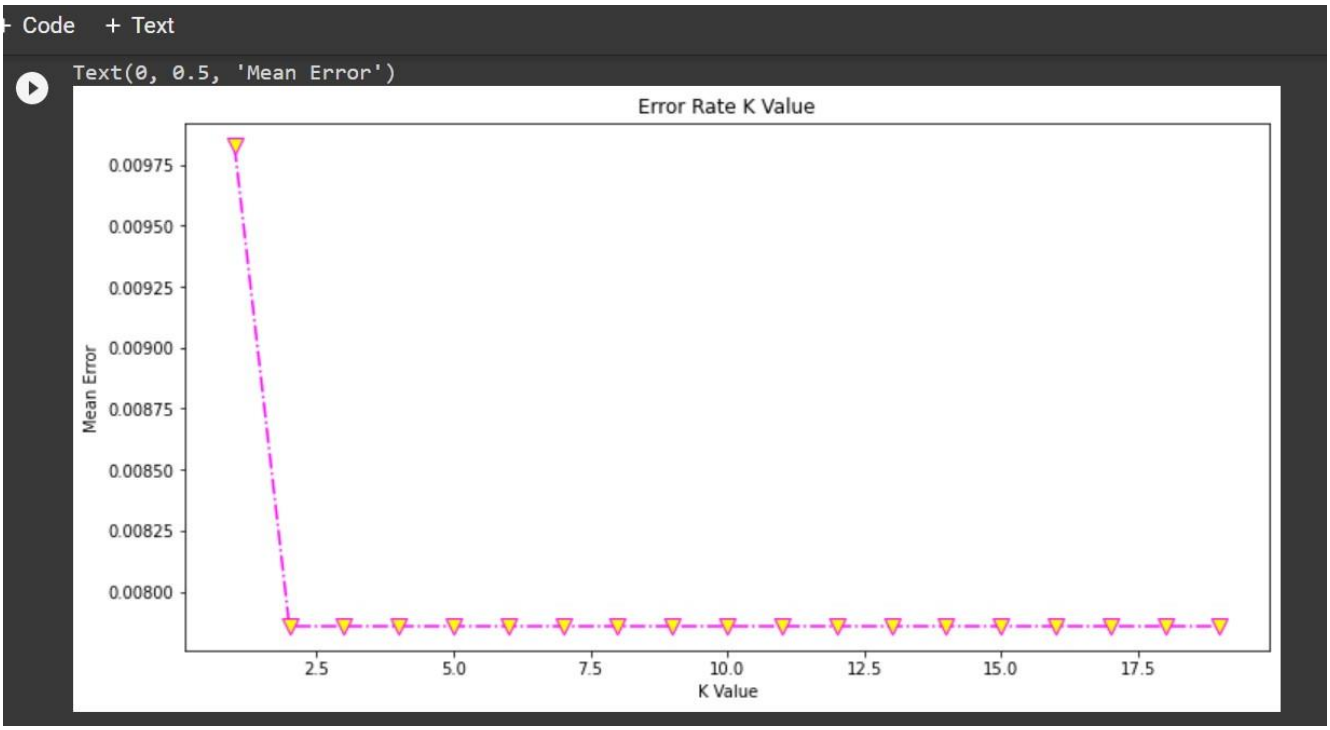
Label 2:

It has 0.9% precision & 42% of recall & 14% of f1-score, and all of these are out of 12 labels.

The model predicted all the 1468 records out of 1572 in the testing set.

And then, it shows the general accuracy the model has, equal to **96%**.

Finally, we have the graph that shows the error values against K values. From the graph, we can determine that the mean error is zero when the value of K is between 2 and 20, which are the best values to be the K.



Evaluation

Evaluating the machine learning algorithm is necessary for any project to see the model performance after building. So after splitting the dataset into training and testing sets, were testing set was of size **30%**, and **70%** of data went to the training set where we have trained the KNN algorithm. The model was capable of predicting all the 1468 records out of 1572 in the testing set with **96%** accuracy means our model predicts **4%** wrong predictions.
1463 records of label 1 are predicted right by the model with only 52 wrong predictions, and 5 records of label 2 are predicted right with 7 wrong predictions because of the imbalanced dataset.
These measures mean that our model did a great job.

Conclusion

At the end of our project, We have come out with many benefits on a personal level. We gained little knowledge of ways to find exoplanets and help astronomers detect them by the model we developed.
We delved into the details of the K-Nearest Neighbor (KNN) machine learning technique, such as implementing it, evaluating actual data, and predicting new data. We encountered some problems and tried to find solutions to them, which made us more efficient and more worthy than before, and we also enjoyed this work and the result of its final, which achieved an accuracy of **96%**.