Exoplanet Detection System using Python and Machine Learning

ABSTRACT

This paper presents a comprehensive approach to exoplanet detection utilizing both classical machine learning and deep learning techniques. Exoplanets, planets orbiting stars beyond our solar system, have been in the astrophysical limelight because they could host life. The problem of detecting exoplanets manually is challenging due to the intrinsic class imbalance nature of the dataset and the complexity involved in identifying subtle patterns within light curves. This paper presents a comprehensive approach to exoplanet detection utilizing both classical machine learning and deep learning techniques. Initially, this project focused on classical machine learning methods, such as logistic regression, decision trees, XGBoost, SVM. These models were selected for their proven ability to handle structured data efficiently. However, as the complexity and dimensionality of the dataset became apparent, the project scope was expanded to include deep learning techniques. Specifically, a Multilayer Perceptron (MLP) neural network based approaches are proposed to detect exoplanets. First, the Synthetic Minority Over-sampling Technique balanced the dataset because of low representation of exoplanet instances. Principal component analysis was used subsequently to reduce the dimensionality to important features in the dataset. Model evaluation ensued, and decision trees achieved 95% accuracy on the training data, while the MLP neural network gave a very competitive accuracy of 98.04%. These results demonstrate that classical machine learning models and deep learning neural network models can be combined to obtain a powerful system for the detection of exoplanets.

Table of Contents

LIST OF FIGURES

1. INTRODUCTION

1.1 Background

The search for exoplanets - planets that orbit stars beyond our solar system has been a central focus of astrophysical research for decades (NASA, n.d.) orbiting around a star other than our very own Sun. The search for exoplanets includes the promise to eventually find and identify habitable worlds (Seager, 2013). The first exoplanets were discovered in the 1990s, and since then, thousands have been identified using various detection methods. Traditional exoplanet detection methods, such as the transit method, radial velocity, or direct imaging, have been successful, but they are not without their shortcomings.

From Fig. 1, it illustrates the transit method of exoplanet detection - a technique based on measurements of variations in the star's brightness during time. As it is drawn above, when an object, like a planet, passes in front of a star, briefly obstructing a portion of its light, the characteristic dips in the brightness curve form. Thus, periodic drops in brightness are the signature of an orbiting object. Such dips, if they are observed to occur at periodic intervals, can indicate the presence of an exoplanet. If astronomers continuously monitor the flux of the star and find constant dips happening at fixed intervals, then they can identify the object as a potential exoplanet. Most of these techniques require extensive manual interpretation, which can be highly labor-intensive and significantly subject to human error in picking out the smaller, more Earth-like planets from the false positive cases (Pearson, et al., 2018).

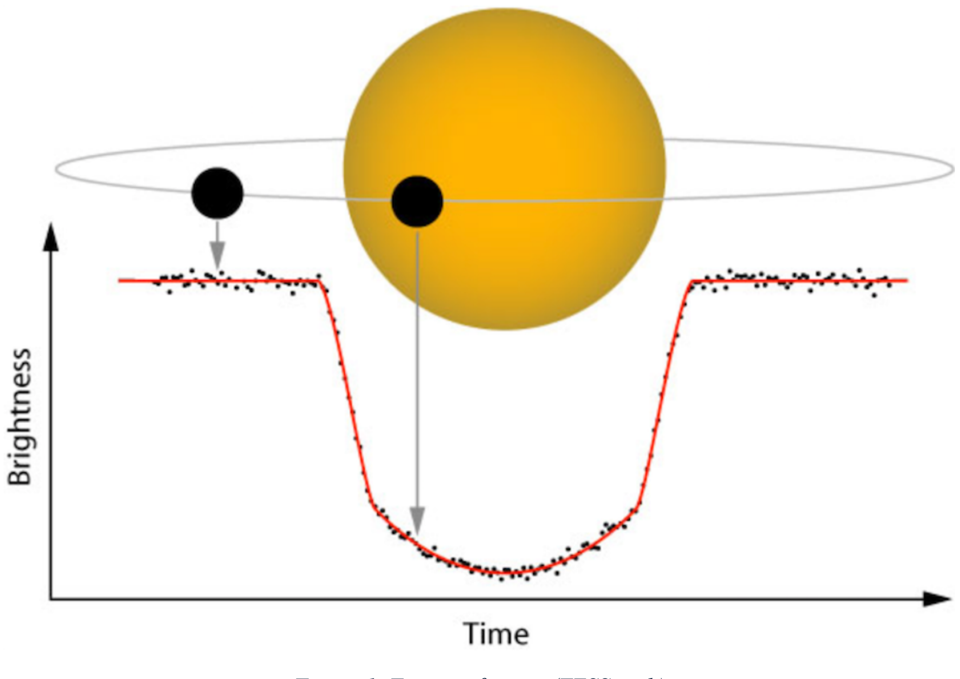


Figure 1. Transit of a star (TESS, n.d.)

However, exoplanet detection is not an easy task. Probably one of the biggest challenges here is that stars are just so much brighter as compared to the planets they host, making the latter quite dim by comparison and hard to observe directly. Moreover, most of the exoplanets especially Earth-sized ones are small and of little mass, which creates only a very small gravitational perturbation on their host stars. This makes them difficult to detect using radial velocity or other techniques. The distances are enormous, often in light-years, which further complicates their detectability and studies. Finally, one of the common problems is false positives: signals indicating the presence of an exoplanet can result from other phenomena, resulting in spurious detections.

Astronomers can thus much more precisely determine the possible presence of exoplanets and reduce the occurrence of false positives by carefully analyzing transit signals and accounting for the list of challenges.
In view of the challenges associated with such a process, this project is intended to adopt machine learning techniques to automate the detection of exoplanets. Machine learning offers a really powerful way, compared to conventional techniques, for the automatic identification of patterns in large datasets, hence reducing time and effort required for analysis while improving accuracy and sensitivity. By harnessing the capabilities of machine learning, researchers and scientists will be able to streamline the exoplanet detection process, thereby allocating their resources towards exploring new frontiers in astronomy and advancing scientific inquiry (Shallue & Vanderburg, 2018).

1. 2. Research Question

- How can machine learning approaches be adapted and optimized to detect exoplanets, such as Earth-like planets in the habitable zone?

1.3. Objectives of the Study

The overall goal of the project is to come up with a system that can help in detecting exoplanets by effectively fusing classical machine learning techniques with deep learning models. Specific objectives that this study aims to meet include:

1. **Treatment of the class imbalance in the dataset:**
   - Use SMOTE to create an equal balance in the dataset. This technique should help the model not be biased towards the majority of classes. As a result, it improves its detection of exoplanets, which are underrepresented in the data.
2. **Dimensionality Reduction using PCA:**
   - Apply PCA to reduce the high dimensionality of the dataset. The step seeks to reduce high dimensions so that only important features that will ensure the perfect detection of exoplanets are considered; hence, this makes a simple model and boosts performance.
3. **Implementation and Evaluation of Classical Machine Learning Models:**
   - Implementing and evaluating some of the classical machine learning models: Logistic Regression, Decision Tree, Gradient Boosting (XGBoost), and SVM. All of these will be evaluated for the efficiency of the models in classifying and predicting exoplanets with the processed dataset.
4. **Deep Learning Model (MLP) Development:**
   - Train a multilayer perceptron (MLP) neural network using PyTorch. The MLP will further be evaluated to determine if it would be capable of detecting exoplanets by capturing complex patterns in the data that would have otherwise been highly elusive using classical models.
5. **Comparative Analysis of Model Performance:**
   - Compare the performance of the classical machine learning models with that of the deep learning model using metrics such as accuracy, precision, recall, and the F1 score to determine which approach is most effective for exoplanet detection.

1.4. Structure of the Report

The report is organized into six chapters, each of which covers some aspect of the research process and its findings.

Chapter 2: Literature Review. This Focuses on already published studies relating to the detection of exoplanets: the existing methods and the new approach of machine learning. This chapter explains different techniques, all their strengths with their limitations, and shows any literature gaps that this study looks to cover.

Chapter 3: Methodology. All methods and techniques are utilized in the research in this chapter. This explains all steps for data preprocessing, application of SMOTE in the case of class balance, PCA in the case of dimensionality reduction, implementation of classical machine learning models, and the Multilayer Perceptron Neural Network, explaining the rationale behind their choice.

Chapter 4: Results. The results chapter will contain the outcomes of the different experiments carried out with these models. In the chapter, a comparative analysis regarding the performance of the models with respect to different metrics such as accuracy, precision, recall, and F1 score will be done. The evaluation will then assess how best classical and deep learning approaches are in detecting exoplanets.

Chapter 5: Implementation. The implementation chapter provides a detailed account of how the models were developed, trained, and optimized. It includes the technical specifications of the environment used, a step-by-step description of the coding process, and the integration of different tools and libraries. This chapter also discusses the structure and training of the MLP Neural Network, as well as the hyperparameter tuning process that led to the final model.

Chapter 6: Discussion. In the present chapter, the results of the study are critically evaluated the research objectives. This discussion shall address the implications of the results, strengths, and weaknesses of the approaches used, and the possible future directions of research in this area. The evaluation shall also reflect upon the limitations of the present study and propose suggestions to improve the models of detection.

Chapter 7: Conclusion. The last chapter summarizes major findings of the present study and checks if the set research objectives have been fulfilled. Some concluding remarks are given regarding how far this work is contributing to the detection of exoplanets and suggesting improvements in the future.

Finally, the appendices include supplementary materials that support the main text, such as detailed tables, visualizations, and code snippets.

2. LITERATURE REVIEW

In this section, we will explore techniques used by various researchers for exoplanet classification. This review will provide valuable insights into how previous studies in this specific domain compare to one another.

2.1. Limitations of Traditional Detection Methods

Exoplanet detection has traditionally relied on such methods as the transit method and radial velocity, which have been instrumental in identifying thousands of exoplanets. At the same time, these methods have considerable limitations. For example, the transit method relies on observing periodic dips in a star's brightness as a planet passes in front of it. This technique requires extensive manual analysis and is, therefore, rather prone to human error (Batalha, 2014). Moreover, the traditional methods, such as box least squares fitting, have many more false positives, especially against noisy data.

These challenges further raise the need for more automated and efficient detection techniques. The failures of traditional methods on big and complex data sets couple with susceptibility to variability in human judgment, raising stakes for the integration of machine learning into the process of exoplanet detection.

2.2. Machine Learning Approaches in Exoplanet Detection

Machine learning has emerged as a strong tool in exoplanet detection, being able to automate the analysis of large datasets and reduce the reliance on manual interpretation. The Kepler Mission has given vast reams of data instrumental for the refining of machine learning models in exoplanet detection. Such models can accurately identify very small patterns in light curves indicative of exoplanet transit.

Recent research has used various techniques in machine learning to improvise this detection process. For instance, (Bahel & Gaikwad, 2022) classified exoplanets using the K-Nearest Neighbors algorithm with light intensity data. Their model showed accuracy of 98.20% after re-balancing the dataset by the Synthetic Minority Oversampling Technique, thereby explaining how machine learning can turn out to be very handy in handling class imbalance and increasing the accuracy of detection.

Similarly, (Herur, et al., 2022) used a hybrid KNN model that was a mix of various algorithms created in the field of machine learning, like the Support Vector Machine and K-Means clustering for the classification of exoplanets. Their proposal cleaned up the dataset by removing outliers from the data and increased the accuracy of the classification by using the strength of different algorithms. They categorized the exoplanets into things like hot Jupiter's and long-period giants, showing the strength of machine learning in handling heterogeneous data about exoplanets.

(Sturrock, et al., 2019) offered a classier based on Random Forest that delivered an accuracy score of 98% cross-validated against the Cumulative Kepler Object of Information table. This model was not only very accurate but also very scalable; deployment via API and as a cloud-based Web service on Microsoft Azure had already been realized. Implementation such as this demonstrates the capability of embedding machine learning models within research infrastructure in the form of real-time exoplanet classifiers, thereby alleviating the burden placed on astronomers.

(Malik, et al., 2022) did so down a quite different avenue - a gradient boosting classifier, this time using light curve features. The method put up superior performance for the detection of exoplanets to that of traditional algorithms like BLS, given a recall of 96% on Kepler data. On the other hand, this paper also emphasized the computational efficiency of the machine learning model by stating that it was able to deliver results comparable to those of deep models at reduced computational power and training time.

Finally, (Nandan, et al., 2022) explored the detection of exoplanets using classical machine learning techniques. The transit method, often utilized for exoplanet detection, involves observing changes in a star's brightness over time, with regular dips indicating potential exoplanets. This technique, however, presents challenges such as alignment issues and a higher risk of false detections. NASA's Kepler and K2 missions have provided extensive datasets for exoplanet detection, offering insights into planet formation and potential habitability. Classical machine learning algorithms were employed in the paper to classify light curves more accurately than deep learning algorithms in certain cases. While deep learning excels in object detection, its optimization and training requirements pose challenges. Despite the performance of classical ML algorithms, they lack robustness and may make errors on unseen data, necessitating human supervision. However, these models offer a reliable means to classify false-positive cases and reduce manual review burdens in exoplanet research.

On the deep learning front, (Tey, et al., 2023) introduced a CNN, Astronet-Triage-v2, in pursuit of improving classification of exoplanetary signals within full-frame images from the TESS mission. The model enables effective separation between astrophysical eclipsing candidates and stellar variability and instrumental artifacts at a recall rate of 99.6% for events that are transiting/eclipsing with a precision of 75.7%. The accuracy of Astronet-Triage-v2 in handling TESS data is extremely high, which points to the rising role of deep learning in processing and analyzing huge amounts of data in astronomy. This model will help save at least 200 candidates that, otherwise, would have been lost compared to the previous version, hence becoming valuable in efforts geared toward detecting exoplanets.

While machine learning has many benefits when applied to exoplanet detection, it is not free from challenges. One of the most important is connected with the models' robustness against unseen data. According to (Malik, et al., 2022), even good models that perform well in a development environment have complications with new data sets, which proves the necessity of continuous human supervision over automated processes.

Moreover, increased volumes of astronomical data demand more scalable and time efficient machine learning models. In the study by (Sturrock, et al., 2019), this was addressed by putting their Random Forest classifier on cloud infrastructure, giving an example of how machine learning models can be made more accessible and scalable. Further studies are still needed to refine these models in handling noisy data and reducing false positives.

2.3. Research Gaps

While machine learning techniques have greatly advanced for the detection of exoplanets, several research gaps remain if the accuracy, efficiency, and scalability of these methods are to be further enhanced.

1. **Model Robustness and Generalization:** Many of the recent studies identify one of the critical challenges as it being notoriously hard to guarantee that a machine learning model generalizes well to unseen data. (Malik, et al., 2022) emphasized the fact that, in vitro, very good models often turn out to be poor when applied to new or noisy datasets. This clearly suggests the deficiency of building more robust models generalized across different datasets and conditions.
2. **Noisy Data Handling:** One of the key challenges for exoplanet detection is noisy data. Traditional detection techniques, with very noisy data, usually yield too many false positives and require manual examination, which is time-expensive and labour-intensive. While the machine learning models, like those considered by (Bahel & Gaikwad, 2022), have promised to

be very efficient in significantly reducing these false positives, there is an important need for more sophisticated noise reduction and feature extraction techniques that can aid in rendering reliability to detection algorithms.

3. **Scalability and Computational Efficiency:** As the size of astronomical data grows, so does the importance of the scalability of machine learning models. Whereas this, as (Sturrock, et al., 2019) have demonstrated, stops a little short of deployment on cloud platforms for machine learning models, this has nonetheless been the right step in the proper direction. However, further research is needed in the optimization of such models for large-scale processing, particularly on decreasing computational costs and efficiently training and making inferences on massive datasets.
4. **Integration with Traditional Methods:** With the many advantages that machine learning has to offer, little work seems to be done on integration with traditional methods for exoplanet detection. There is a need for research into developing hybrid approaches to merge the power of machine learning and traditional techniques in the first filtering and verification of exoplanet candidates.
5. **Ethical and Reproducible Research:** If machine learning is to have long-term integration into scientific research, then the ethical and reproducible nature of the suggested models needs to be ensured. (Sturrock, et al., 2019) stressed the requirement for reproducible machine learning pipelines so that other researchers can verify and extend them. However, there is a pressing requirement for standardization of practices concerning the ethical deployment and use of such models in public-facing applications.

This project aims to address these gaps by evaluating multiple machine learning models and deep learning model, including Logistic Regression, Decision Trees, XGBoost, SVM and Multilayer Perceptron neural network, on a comprehensive dataset of Exoplanet light flux (Kaggle, 2017). The research emphasizes optimizing exoplanet detection by integrating machine learning and deep learning techniques that enhance accuracy while maintaining computational efficiency, offering significant advancements in the identification of habitable planets and improving the reliability of astronomical observations.

2.4. Summary

The literature review provided a broad view of the current situation with exoplanet detection methods, underlining the shortcomings of conventional methods and the enhancing role for machine learning and deep learning techniques within the process. Traditional detection techniques, although intrinsic to this field of study, are, by nature, labour-intensive and thus subject to human error, particularly within large and noisy data sets. This underlines the need for more automated and efficient solutions.

The review has revealed major improvements with the intervention of machine learning techniques, evidenced by K-Nearest Neighbours, Support Vector Machines, and Random Forest classifiers, which have been found rather accurate in light-curve analysis and in classifying exoplanet candidates. Deep learning methods, particularly convolutional neural networks, have also pushed detection capabilities to the next level with the automation of pattern recognition in large datasets, as manifested in the success of Astronet-Triage-v2.

In the process, several research gaps were identified, such as more robust models that would generalize across different data sets, noise reduction methods working better, and scalable solutions to deal with growing volumes of astronomical data. Looking into these gaps will aid in setting up a field where exoplanet detection methods are highly improved in reliability and efficiency.

One can make out the benefits that emanate from the integration of machine learning and deep learning into the detection of exoplanets, which forebodes so much promise in paving the way to more accurate and scalable solutions within the realms of astronomical research.

3. METHODOLOGY

3.1 Data Collection

This project uses the dataset drawn from Kepler light flux labelled time series data available on Kaggle (Kaggle, 2017). The dataset includes observations from the NASA Kepler space telescope, with a particular emphasis on Campaign 3 and extra confirmed exoplanet stars added from other campaigns. This dataset captures the variation in light intensity for a couple of thousand stars over some time, thus robust enough for exoplanet detection using machine learning techniques.

This dataset is represented as a feature vector for each observation. The label vector is in the first column. It gives distinction between a non-exoplanet star and a confirmed exoplanet star; 1 is for a star which does not have an exoplanet, and 2 for one that does. Then, from column 2 through column 3198, are flux values, light intensities recorded over some time for each star. This wealth of functionalities is very important in analysing light curves for the identification of possible exoplanetary transits.

Train set:

- Contains 5087 observations.
- Each observation has 3198 features.
- The train set comprises of 37 confirmed exoplanet-stars and 5050 non-exoplanet stars.
- Column 1 represents the label vector, while columns 2-3198 are the flux (light-intensity) values recorded over time.

Test set:

- Consists of 570 rows or observations.
- Each observation has 3198 columns or features.
- The train set comprises of 5 confirmed exoplanet-stars and 565 non-exoplanet stars
- Column 1 represents the label vector, while columns 2-3198 are the flux (light-intensity) values recorded over time.

This dataset is divided into two major subsets: a training set and a test set. The training set contains 5,087 observations, each of which includes 3,198 features. Out of 5,087 observations, 37 are confirmed exoplanet stars, while 5,050 of the observations are non-exoplanet stars. The test set only includes 570 observations, with each observation including 3,198 features. Of such test observations, 5 are confirmed exoplanet stars, while 565 observations belong to non-exoplanet stars.

3.2 Data Preprocessing

This forms a very critical step in a data-driven project, more so when dealing with high-dimensional and intricate datasets from the NASA Kepler mission. In processing data, this stage aims at preparing data not only for analysis but also in enhancing the model's predictive accuracy by ensuring that the dataset is clean, balanced, and representative of real underlying phenomena.

3.2.1. Combining Training and Test Sets for Consistency

In the first step, I decided to concatenate the given training and test sets into one Pandas DataFrame. This approach is partially driven by the desire to have uniformity in all the subsequent preprocessing steps, so that exactly the same transformations are applied and the same checks performed on the whole dataset. The reason for creating just one preprocessed version of the training and test set is that it reduced the risk of introducing discrepancies between how the data was dealt with in different phases of the project. Furthermore, it eased running the processing pipeline in cases where only one preprocessed version existed.

The merged dataset contained 5,657 total observations; each observation is a 3,198-dimensional vector of flux values recorded with respect to time. Again, handling such a large and complex dataset with due care was necessary so that the machine learning models to be trained thereafter learned effectively from this data and did not get misdirected by inconsistencies or anomalies.

3.2.2. Verification of Data Types and Integrity

The integrity of the data is always the very first step toward the performance of any data science project. Checking the dataset's data types, it emerged that all features were of int64 type. This uniformity in data types was important in showing that the data was in a format suitable for numerical analysis without conversion or handling of mixed data types. This is particularly the case in high dimensional datasets where differences in data types may introduce errors or cause misinterpretation by machine learning algorithms.

It was then checked for missing values in the dataset. The good thing about this large dataset was that it didn't have any missing data therefore rendering the imputation techniques null and void, most of which at times are fancy ways of introducing bias or uncertainty into the data. Also, since the dataset was complete, the analysis would be relatively clean because all the available information would get utilized in a pretty nice way during the process of modelling.

3.2.3. Addressing Outliers through Z-Score Technique

Outliers are data points that lie appreciably far away from the general distribution of a dataset. Since such data introduces noise, which further causes distortion in the performance of a machine learning model, in the light of this high dimensionality characterizing the dataset used in this study, the need for detecting and removing outliers was therefore imperative to improve the accuracy and reliability of ensuing analyses.

I used the z-score method in searching for outliers. This is a method that calculates every data point's standard score, showing how many standard deviations a point is from the mean of the dataset. Those points having a z-score higher than 3 were taken as outliers and thus removed.

This dataset contained 5,657 observations, each with 3,197 features, before outlier removal. Flagged as outliers by the z-score method and removed were 79 observations. Hence, the data was cleaned to only 5,578 observations. This reduction in dataset size reflects how some anomalous data points are removed, which might have resulted in skewed results for the machine learning models.

Normalization of the dataset eliminated these outliers. In this respect, the final data is more homogeneous and representative of the trends and relationships that exist at the underlying level. Therefore, this was very important to be present for making sure that in this study, machine learning models may learn effectively from the data to improve prediction accuracy and robustness.

3.2.4. Exploratory Data Analysis: Understanding Light Flux Behavior

Before diving into feature engineering and model training, it was essential to conduct an exploratory data analysis (EDA) to gain insights into the dataset's structure and the behaviour of light flux in both non-exoplanet and exoplanet stars. EDA serves as a preliminary step to uncover patterns, detect anomalies, and test hypotheses, which in turn informs the choice of models and methods in later stages.

A random sample of observations from both classes - non-exoplanet stars and confirmed exoplanet stars - was selected for detailed analysis. The light flux data for these samples was plotted to visualize the characteristic patterns associated with each class. These visualizations, as shown in Fig 2, revealed distinctive features in the light curves of exoplanet-hosting stars compared to non-exoplanet stars. For instance, the periodic dips in light intensity, indicative of a potential exoplanet transit, were more pronounced in the exoplanet-star observations. This analysis not only validated the dataset's integrity but also provided a deeper understanding of the underlying physical phenomena that the machine learning models would need to capture.

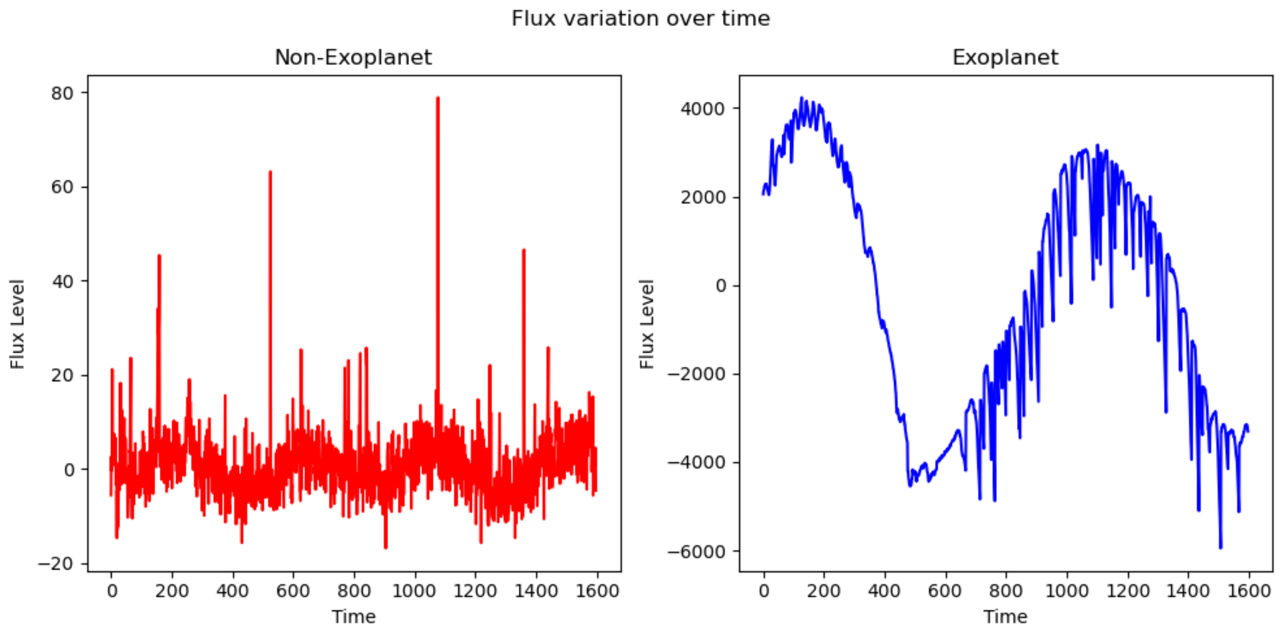


Figure 2. Flux variation difference between non-exoplanet and exoplanet (Source: self-created)

3.2.5. Transforming Labels for Binary Classification

To prepare the dataset for machine learning, labels were remapped into a binary classification framework. Originally, the dataset used '1' to refer to non-exoplanet stars and '2' for confirmed exoplanet stars. These were replaced with '0' for non-exoplanets and '1' for exoplanets. This was important in simplifying the task of classification using binary algorithms that were selected for this work.

This label transformation ensured that the models would have to focus only on differentiating between two classes: exoplanet versus non-exoplanet. Hence, it made the process of training the models easier and the model outputs more interpretable. Binary classification is a very common technique in machine learning, especially whenever one wants to detect the presence or absence of some phenomenon, which is what transpires in exoplanet detection.

3.2.6. Addressing Class Imbalance with SMOTE

The class imbalance is one of the most frequent problems in machine learning in general and in exoplanet detection in particular, since what is being tried to be classified in these cases are phenomena that are scantily distributed: exoplanets, which are highly imbalanced, relative to stars with and without planets. In this dataset used for the research, 5,615 stars turned out to not have exoplanets, 42 turned out to have an exoplanet confirmed representing highly skewed distributions.

This introduced a likelihood of bias in the machine learning models; that is, the models would end up predicting the majority class, which involved the non-exoplanets, and miss out the minority class that included exoplanets. To address this issue, the Synthetic Minority Oversampling Technique (SMOTE) was implemented using the imblearn (imbalanced-learn, 2016) package in Python (Appendix 6). In this respect, Synthetic Minority Oversampling Technique was applied.

Class Distribution Before SMOTE

Before the application of SMOTE, there was a heavy class imbalance with respect to the classes, as shown in Fig 3, Fig 4. The number of instances related to non-exoplanets and exoplanets is hugely different from one another; hence, it may be a concern for the model to learn the minority class.

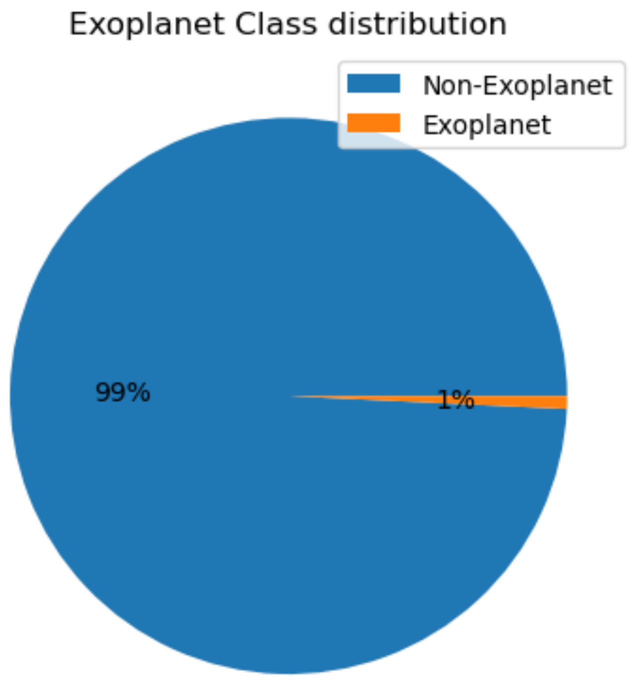


Figure 3. Pie chart for class distribution before SMOTE (Source: self-created)

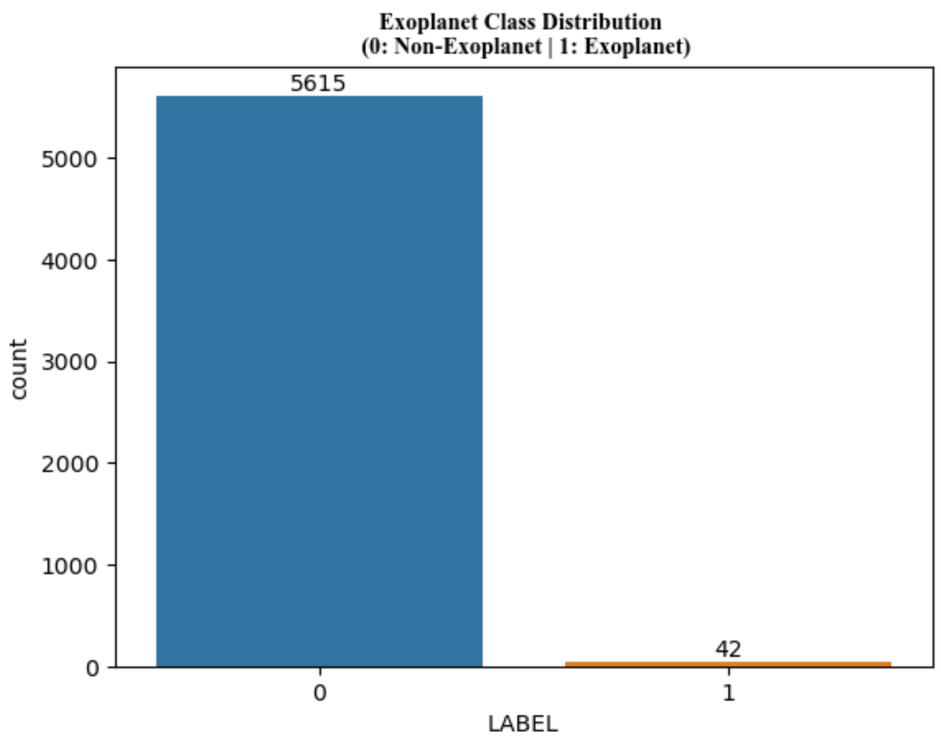


Figure 4. Class distribution bar chart before SMOTE (Source: self-created)

Applying SMOTE
SMOTE is a technique that works by the oversampling of the minority class. It creates synthetic examples from the minority class by interpolation between existing minority class examples. This effectively boosts the representations of the minority class and evens the dataset for the machine learning models to learn with an equitable example set.

Figure 5, and 6 clearly depict a much more balanced class distribution after the application of SMOTE. One can see that both classes. In this case, non-exoplanet and exoplanet instances contain exactly the same number of elements. This is important to ensure that the models not only bias toward the majority class but improve in detecting exoplanets.

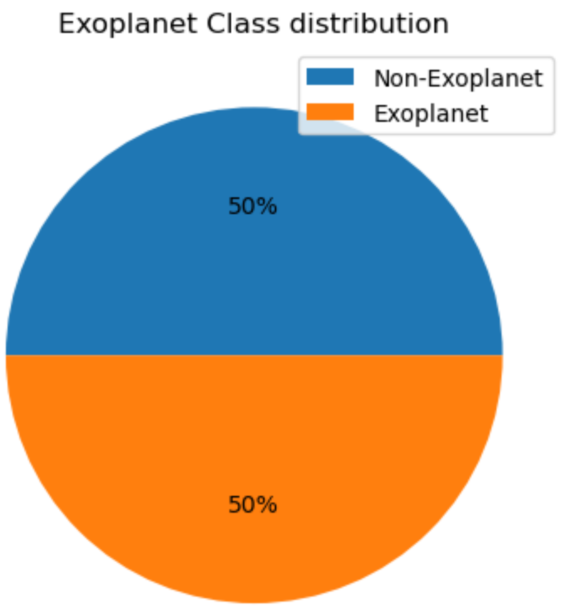


Figure 5. Pie chart for class distribution after SMOTE (Source: self-created)

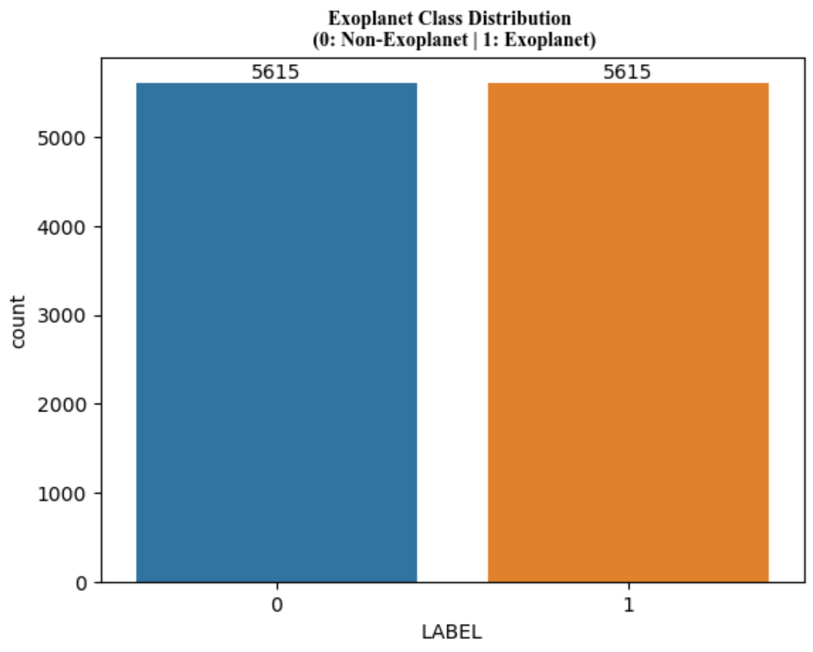


Figure 6. Class distribution bar chart before SMOTE (Source: self-created)

Impact of SMOTE on Model Performance

Balancing this dataset using SMOTE was important for improving machine learning models. By giving a more representative set of training data to models, the risk of overfitting to the majority class was reduced. This enhanced models' ability to generalize and predict accurately exoplanets in unseen data which we'll see in next sections. Recall and precision were mostly improved well using the models trained from the SMOTE-adjusted dataset, showing the efficacy of this method.

In summary, class imbalance was dealt with efficiently using SMOTE, enabling more reliable and accurate predictions by the models. This was an important step in ensuring that the developed models could perform well in detecting exoplanets, a really challenging task due to their natural rarity in most astrophysical datasets.

3.3. Feature Engineering and Selection

Feature engineering is an important stage in the machine learning pipeline, especially if high-dimensional datasets are being used, like the one in this study. This step involves changing raw data into a set of features that better brings out the underlying problem to predictive models, hence improving their performance. In this project, the main procedures were feature correlation analysis and dimensionality reduction using PCA.

3.3.1 Analyzing Feature Correlation

Given the high dimensionality of the dataset, it was important to understand the relationship between these features. Highly correlated features will then cause redundancies, increasing model complexity, probably leading to overfitting: if a model performs well on training data, it will perform poorly on unseen data.

To understand what relationship these various flux measurements, take, a correlation matrix was calculated for a subset of the features that targeted flux values within certain ranges. It showed that fluxes with a 300-500 range are more correlated compared to the higher-valued ones. For instance, features FLUX.676 and FLUX.594 are highly correlated, hence carrying similar information. Although FLUX.676 and FLUX.1945 are less correlated, this suggests that they are picking up varied information about the light curve of the star. This is represented graphically in the correlation matrix as shown in Fig 7.

This analysis revealed a lot of redundancy within this dataset, particularly between features falling within certain flux ranges. I thus need to consider the necessity of dimensionality reduction techniques to retain only very informative features while reducing the overall number of features.

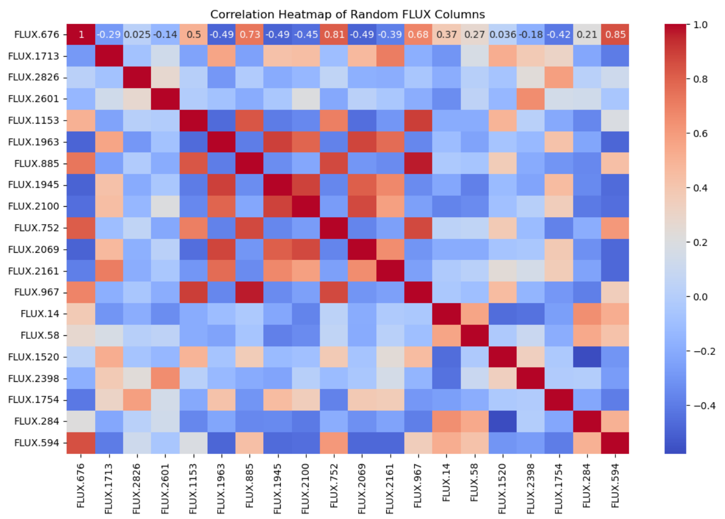


Figure 7. Correlation matrix of random flux (Source: self-created)

3.3.2 Application of Principal Component Analysis (PCA)

Principal Component Analysis was done to handle high dimensionality and redundancy reduction in the dataset. PCA is a technique in statistics which transforms the original features to a new set of features called the principal components. The components are a linear combination of the original features, ordered so that the first few components retain most of the variance in the data.

In the present study, Principal Component Analysis (PCA) is one of the pattern recognition techniques and one of its applications is to analyze the high dimensional data that is not easy to understand by just looking at the large amount of data (Sehgal, et al., 2014). After keeping principal components explaining major parts of variance in the data, features were reduced from 3,198 to 13 only. This reduction not only diminished computational load but also concentrated the model on the most essentially important aspects of data.

After PCA is done, this dataset is reduced to 13 principal components with the last column of targets corresponding to whether the star is confirmed as an exoplanet or not (Appendix 7). This transformation greatly reduced the size of the dataset, which feeds into the machine learning model without any loss of information necessary for its predictions.

3.3.3 Outcomes of Feature Engineering

The feature engineering exercises closed out with this dataset, which is of reduced dimensionality but still optimized for modelling. In this case, these 13 principal components captured most of the variance of the original data; hence, important patterns would be preserved while removing redundant information. This resulted in a refined dataset that served as a good ground for training the machine learning models since they had a narrow scope of only important features without noise introduced by the entire set of 3,198 features.

It is particularly useful to use PCA for improving the interpretability of the model and reducing overfitting. By working on only part of the features, which are more informative, models generalize better to unseen data and improve in their performance.

3.4 Model Training and Evaluation

This now enters the critical phase of the project: training and testing machine learning models using the prepared data to come up with predictive models that identify exoplanets based on light flux data. In this section, a view of the processes involved in splitting the data, selection of appropriate models, and training of these models will be presented, followed by their evaluation.

3.4.1 Data Splitting

To ensure generalizability of the machine learning models across unseen data, the dataset was cleaned and split into a training and test set. The training dataset was the set for modeling development and optimization, while the test dataset was used for independent appraisal of model performance. In this project, a typical split of the dataset is 80-20. As such, 80% of data is used for training, and 20% is used for testing. This split was done in order to guarantee a large enough dataset for training, while at the same time reserving enough data to get reliable model fit indicators rating the future generalizing ability of the model under study.

The function to be used was scikit-learn's train_test_split with ease, from the model_selection module. The stratify parameter was not used within this context. Threadedly the exoplanets and non-exoplanets abundantly in both the training and test groups for the study at hand.

The random_state had been set for reproducibility of results, and the data split accordingly. Setting of random_state allows the model evaluation and comparison of models to also maintain consistency in every running experiment; thus, the same split is maintained in every running experiment. This is a very important step not only in scientific research but also in ensuring that other researchers can make similar findings using practically the same dataset.

3.4.2 Choosing Models

In this work, four machine learning models have been selected to solve the challenge of exoplanet detection: Logistic Regression, Decision Trees, XGBoost, and a Multilayer Perceptron Neural Network. Each one of the models selected for this study was chosen in view of its advantages and ability to solve the problem because of the characteristics it has in relation to the dataset.

Logistic Regression was selected as a baseline model for the following reasons, among others: its simplicity and high interpretability. This linear model is highly fruitful for binary classification problems, so the contributions of each feature to the prediction outcome are very clear. Logistic Regression research performance only for the creation of the baseline; for example, the investigation of more complex models, outcome evaluation, or forecasting.

The decision trees are included within the model selection because they have the property of very high flexibility; they can operate well with nonlinear relationships between features. Decision trees take the strategy of iteratively splitting the data into smaller and smaller sets depending on the value of the input features; this would allow a model that is interpretable and, at the same time, can capture very complex interactions. The tree-based structure of the model furthermore allows easy visualization of the decision processes at play, and will, for this reason, help understand factors that drive exoplanet classifications.

Boosted algorithms give an additional layer of efficiency to XGBoost, allowing it to deal with very large datasets and capture many of the most intricate patterns. To this end, XGBoost builds an ensemble of decision trees where, at each learning process, the knowledge of the existing model guides the training of a new tree. This allows the model to enhance, iteration by iteration, the overall generalization performance, especially for datasets which are class imbalanced. This fact, combined with how robust XGBoost is and its ability to cope with extremely diverse data distributions, made it an attractive candidate.

Support Vector Machine (SVM) was used to model the decision boundaries in a high-dimensional space. Through finding an optimum hyperplane, SVM works with complex data sets where the decision boundaries are nonlinear. It is very strong in high-dimensional spaces and uses the kernel trick for data transformation, realizing enhanced separation of the classes, hence ensuring robust and accurate classification results.

Deep learning potential was explored by considering the multilayer perceptron neural network. In this case, the relationship that subsists between the independent and dependent variables is nonlinear, which may be appropriate for MLPs since they are able to model such relationships. It has a high flexibility and the capability of capturing subtle and complex patterns within the data. A neural network containing multiple layers of neurons is considered. Each of those layers applies nonlinear transformations to the input data. This enables learning of the MLP from the high-dimensional light flux data, probably capturing a pattern which would have been lost with simpler models. The reason behind the assumption of good performance of the MLP in the case was that the data set is highly complex and dimensional in nature.

These models were combined to cover the spectrum from the simplest to more complex models and, on the other hand, from interpretable models to advanced techniques that let out intricate patterns hidden in the data. Precisely, this research has sought to evaluate which of the considered approaches turns into the most effective way for exoplanet detection in the Kepler dataset.

4. IMPLEMENTATION

This chapter gives all the steps taken in implementing both machine learning models and a neural network in exoplanet detection using light flux data. The stages in the implementation include data preprocessing, model selection, and training, all of which are explained hereafter.

4.1 System Specifications

The implementations were done using the following software and libraries. This allows reproducibility and consistency.

- Operating System: macOS
- Python: 3.12.3
- NumPy: 1.26.4
- Pandas: 2.2.1
- scikit-learn: 1.4.2
- Jupyter: 1.0.0 for interactive notebook environment
- PyTorch: 2.4.0 for neural network implementation
- imbalanced-learn: 0.12.3 for handling class imbalance issues
- Matplotlib: 3.8.4 for data visualization
- Seaborn: 0.12.2 for better data visualization
- VSCode: 1.92.1 for code development

These packages provided all the necessary functionality regarding data manipulation, model training, and result visualization.

4.2 Importing Libraries and Configurations

At the very beginning of the implementation, all the libraries that would be used were imported. They were relevant in doing some numerals and data manipulation, building of models and visualization. These libraries are, among others, NumPy and Pandas for data manipulation, Matplotlib and Seaborn for visualizations, and scikit-learn for machine learning algorithms and metrics.

Additionally, PyTorch was imported to implement a neural network model and train it later. Some configuration settings were subsequently made, first for reproducibility and computational efficiency: setting a random seed and configuring Matplotlib for in-line plotting in Jupyter.

4.3 Data Importation and Initial Exploration

The dataset used in this study was imported in CSV format, containing both train and test data. These two datasets have to be concatenated to get one unified data set for analysis and further processing. Initial data exploration was done by checking the shape and structure of the data and some basic statistics to understand how the features are distributed and their nature.

4.4 Exploratory Data Analysis

First of all, exploratory data analysis (EDA) was done to find the presence of any hidden patterns and to see the quality of data. It comprised some important operations that included the following:

1. **Type of Data and Missing Values:** Checking the data types so as to ensure they were suitable for any Machine Learning algorithm; checking for missing values returned an output where there was no missing value entry in this regard.
2. **Outlier Detection:** Outliers were detected in this stage using the z-score technique; after that, those exceeding a threshold were removed. This step reduced noise and thus improved the dataset's reliability.

3. **Light Flux Analysis:** Light flux data from both exoplanet and non-exoplanet stars were visualized to understand their differences concerning their light curves. This understanding is very important in making differentiation between the two classes.

4.5 Dealing with Class Imbalance with SMOTE

The crucial feature in the dataset is class imbalance: many more non-exoplanet observations in comparison to exoplanet observations. To address such an issue, the Synthetic Minority Over-sampling Technique is applied using the imbalanced-learn library (Appendix 6). So, SMOTE gave us synthetic samples for the minority class, making the dataset at hand a balanced dataset, hence more applicable for training the models.

4.6 Feature Engineering with Principal Component Analysis (PCA)

Since the dimensionality of this dataset is high, Principal Component Analysis was done to bring down the number of features while preserving most of the variance in the dataset. The PCA was applied only after standardizing the features, while the number of components was chosen so that 95% of the variance could be retained (Appendix 7). This step of dimensionality reduction not only improved computational efficiency but also helped in reducing noise.

4.7 Model Training

Model training is an important component in building predictive models, through which one can make the model learn from data and predict the outcome perfectly. Varying machine learning models were used in this project for exoplanet detection from light flux data. Models prepared were performed using Logistic Regression with Decision Tree, XGBoost, Support Vector Machine, and a Multilayer Perceptron Neural Network. All of these were chosen due to the different strengths of algorithms while handling various data forms and complexities in a dataset.

Training was performed on 80% of the dataset, which was kept aside for model learning. It had enough data such that the models can understand the underlying patterns and relationships between the features and the target variable, exoplanet detection. Logistic Regression was added as a baseline model because of its simplicity and interpretability. The Decision Tree model was supposed to capture the nonlinear relationships in the data, while XGBoost acted as a good ensemble method to improve the model using gradient boosting. Then, both SVM was chosen for the handling of very highly dimensional data and resulting in very apparent decision borders, and finally, the MLP Neural Network was implemented to capture more complex patterns and interactions in the data.

First, we built a logistic regression as a simple linear model to give us a very clear benchmark for comparing the performance of more complex models. The version of this model was trained to optimize the weights on these input features in such a way that the error in the prediction of a binary outcome was minimized (Appendix 8).

A Decision Tree was trained to build a model in which the data are differentiated into branches based on the characteristic values and in this way tries to grasp the complex interrelations within the information. Hyperparameters of the tree, including its depth, are then carefully optimized to ensure the balance of complexity in the model with a means of escaping overfitting, thus allowing the tree to generalize well with new data (Appendix 9).

For more complex models, XGBoost was used, hoping to harness boosting ability while combining many weak learners into a very strong predictor. The XGBoost method worked through iterative optimization of any misconceptions induced by one tree in the sequence from the previous tree, therefore leading to the creation of an accurate and robust model. Hyperparameter tuning was also done with changes in the learning rate and tree depth to really pump up the performance of this model (Appendix 10).

Now, SVM was trained with an RBF kernel that is good at separating classes in a high-dimensional feature space. SVM training was focused on the best hyperplane that allows maximizing margins between both classes, and regularization was applied to avoid overfitting. As a result, careful regularization parameters (C) and kernel coefficients (gamma) may thus lead to the best classification performance (Appendix 11).

Hereby, the multilayer perceptron neural network was applied so as to grasp more complex patterns within the data that might have been too hard for traditional machine learning models to capture (Appendix 13). In this case, the implemented MLP comprised of an input layer, two hidden layers activated by the ReLU function, and an output layer. The training steps were as follows:

- **Data Preparation:** The input data was standardized and converted into PyTorch Tensors. The Tensors are put into DataLoader objects, allowing for easy batch processing efficiently while making predictions.
- **Network Architecture:** The architecture of an MLP is given by defining the number of input features (94), the number of neurons in the hidden layer (64 for each of the two hidden layers), and the number of probabilities to output at the final layer (2); these can be broadened to wider or deeper topologies to research which best fits into specific problems.
- **Training:** The MLP was realized by training with an Adam optimization scheme with cross-entropy loss as its objective function. The update is done iteratively over 80 epochs in such a way that every epoch comprises a forward pass, calculating the loss, backpropagation, and optimization until the model converges. The performance of the model was measured in real-time during training.
- **Testing in performance:** This trained model is now assessed based on the test dataset. Accuracy and other performance criteria are calculated based on this, which indicates the degree to which the model trained generalizes to unseen data.

Refer training and testing loss curve in Appendix 16. The purpose of this phase was to generate a model set that provides good detection of exoplanets. An exploration of different models, including the deep-learned model with the MLP Neural Network trained by a large part of the dataset, was aimed at revealing the one that most effectively balances accuracy in making predictions and generalization with new data, apart from making sure that the final model was realizable in real applicability.

4.8 Hyperparameter Tuning

Hyperparameter tuning is a crucial step in optimizing the performance of machine learning models. For this project, the Multilayer Perceptron (MLP) Neural Network was selected for hyperparameter tuning due to its flexibility, ability to model complex patterns, and potential to outperform traditional models when appropriately optimized. This section details the approach taken to fine-tune the hyperparameters of the MLP to achieve the best possible results in exoplanet detection.

4.8.1 Importance of Hyperparameter Tuning

Performance in neural networks, including MLPs, is very sensitive to the choice of the right hyperparameters. Examples include the number of hidden layers, neurons in every layer, learning rate, batch size, number of epochs, and so forth. Unlike model parameters, which are learned during training, the hyper-parameters are set before training and may have a big impact on how well a model generalizes to unseen data.

Due to the high dimensionality and complex nature of this dataset, tuning these hyperparameters was required so that an MLP Neural Network may learn properly from the data to predict with great accuracy.

4.8.2 Hyperparameters Considered for Tuning

The following are some of the hyperparameters considered for tuning in the MLP neural network (Appendix 13).

- **Number of Hidden Layers and Neurons:** The initial configuration had the MLP with a single hidden layer. After several experiments, the model proved to perform well by including an additional hidden layer. Two hidden layers with 128 neurons each were used in the final architecture. This depth helped capture more complex patterns within the data and thereby enhanced the accuracy of the model.
- **Learning Rate:** The learning rate in the model regulates the pace at which adaptation to the problem is carried out. This is set to 0.001, making a trade-off between the stability and speed of training. It ensured smooth convergence without overshooting of the model over the optimal solution.
- **Smaller batch size:** It went down from 64 to 16 so that the model updated its parameters more frequently during training. While this surely increased the training time, it would have led to more precise model weight adjustments, hence better performance.
- **Number of Epochs:** The number of epochs was increased from 50 to 100. With this longer time to train, the model was able to fine-tune knowledge regarding the data and attain higher accuracy on the test set. Increasing the number of epochs meant that the model would learn enough from the training dataset.
- **Regularization Parameters:** This was for avoiding overfitting. L2 regularization was used, with a penalty of 0.01, ensuring to keep the model generalizable while avoiding very high complexity in the learned parameters.
- **Activation Functions:** The activated function in both hidden layers was the ReLU - Rectified Linear Unit. ReLU was chosen for deep networks because it would mitigate the vanishing gradient problem, that would let the network learn more efficiently.

4.8.3 Results of Hyperparameter Tuning

This has been stated several times, but just to reiterate, the MLP Neural Network without any hyperparameter tuning was able to obtain an accuracy of 94.86% on the test dataset. After thorough tuning, following are the hyperparameters considered for the optimal performance of the MLP Neural Network:

- Number of Hidden Layers: 2 (Initially it was 1).
- Neurons per Layer: 64 neurons per hidden layer.
- Learning Rate: 0.001
- Batch Size: 16. This has been reduced from 64, so that updates can happen more often.
- Number of epochs: 100. This was increased from the initial experimentation with 50 and 60 epochs.

These settings provided a good balance between model complexity and generalization, improving accuracy to a greater extent. Finally, this tuned model reached 98.04 percent accuracy on the test dataset, thus it is very good at detecting exoplanets.

Clearly, the performance of the MLP can be significantly improved by this hyperparameter tuning process. This also underlined the fact that deep learning models realize their full potential on challenging problems like exoplanet detection only after careful tuning.

5. NUMERICAL RESULTS

This chapter contains the results from the final multilayer perceptron neural network model, which is to be trained and tuned for detecting exoplanets based on light flux data. After this, the performance of this model will be tested against the test dataset and the results are presented in the subsection below.

5.1 Overview of Model Performance

The final MLP Neural Network model after hyperparameter tuning yielded an accuracy of 98.04% on the test dataset, which means the model can classify every star with extremely high precision as either being an exoplanet star or not. Clearly, this metric shows the effectiveness of the model in classifying both exoplanet and other stars. The confusion matrix and classification report give deeper insight into the predictive power of the model.

5.2 Confusion Matrix

The confusion matrix provides a summary of the prediction outcomes on the test data, comparing the actual labels to the predicted labels. The matrix is as follows:

	Predicted: 0	**Predicted: 1**
Actual: 0	1109	36
Actual: 1	42	1059

Table 1. Confusion Matrix

- **True Positives (TP):** 1109 instances where the model correctly predicted non-exoplanet stars.
- **True Negatives (TN):** 1059 instances where the model correctly predicted exoplanet stars.
- **False Positives (FP):** 36 instances where the model incorrectly predicted exoplanet stars.
- **False Negatives (FN):** 42 instances where the model incorrectly predicted non-exoplanet stars.

The confusion matrix indicates that the model is highly effective in distinguishing between exoplanet and non-exoplanet stars, with a low number of false positives and false negatives.

5.3 Classification Report

The classification report provides detailed metrics on the model's precision, recall, and F1-score for each class (non-exoplanet and exoplanet). These metrics offer a more nuanced understanding of the model's performance:

	Precision	Recall	F1-Score	Support
0	0.97	0.99	0.98	1145
1	0.99	0.97	0.98	1101
Accuracy		0.98		

Table 2. Classification report

- **Precision:** The precision for class 0 is 0.97 - 97% of the non-exoplanets are correctly predicted. The precision for class 1, exoplanet, is 0.99, that is, 99% of those predicted as exoplanets were proper.
- **Recall:** For class 0 this is 0.99, so 99% of all actual non-exoplanets are rightly recognized by the model. For class 1 it's 0.97 so the model rightly identified 97% of all actual exoplanets.

- **F1-Score:** The F1-score, the harmonic mean of precision and recall, comes to be 0.98 for class 0 and 0.98 for class 1. This metric provides a balanced measure of the model's accuracy.
- **Support:** The values in support are the number of actual occurrences of each class in test dataset, with class 0 having 1145 instances and class 1 having 1101.

5.4 Summary of Results

The performance of the MLP Neural Network model was very good in all metrics, most especially with an overall accuracy of 98.04%. According to the confusion matrix and classification report, the model is very effective at differentiating exoplanets from non-exoplanets and gave minimum errors. High precision and recall scores in both classes establish the reliability of the model for the purpose of exoplanet detection.

These results confirm that the MLP neural network should be taken into consideration and underline the efficiency of the applied hyperparameter tuning process. Generalizing well to unseen data, it can be a strong tool in exoplanet detection, helping further astronomical research and possible new discoveries.

6. DISCUSSION

6.1 Summary of Findings

The major objective of this project was to come up with a strong machine learning model that could be used in detecting exoplanets using light flux data. In this line, this project has sought to apply different machine learning models such as Logistic Regression, Decision Tree, XGBoost, Support Vector Machine, and a Multilayer Perceptron Neural Network to identify the best among these methods against this complex task.

After much hyperparameter tuning, the final neural network of an MLP type ran to 98.04% accuracy on the test dataset. This performance, well ahead of the preliminary results returned by the simple models, Logistic Regression and Decision Tree, has established that deep learning with a tuned MLP really works very well on this exoplanet detection task.

The confusion matrix and the classification report go further to prove that this model is very robust, showing high precision and recall for both classes, with very little loss in accuracy and high F1-scores. This proves the model's ability to classify or efficiently discriminate between exoplanets and non-exoplanet stars, a function critical to taking this field of exoplanet detection to a higher level.

6.2 Evaluation of Achievements

Reflecting on the objectives that were put on paper when the project just started, it has gone quite clear that most of the objectives were realized with good success:

1. **Objective Achievement:**
   - The first objective was developing a model for the detection of exoplanets. The MLP Neural Network delivered at par or even better than expected.
   - Dimensionality reduction of the dataset using PCA was quite a good move toward improving the model's performance without significant loss of information, as evidenced by the high accuracy and other performance metrics.
   - Using SMOTE to address class imbalance is quite important in making sure that the model generalizes well across both classes, hence avoiding bias toward the more prevalent non-exoplanet class.
2. **Challenges and Limitations:**
   - Although most aspects of the project were successful, some challenges were encountered. The dataset dimensionality was extremely high; this had to be kept in mind when doing preprocessing and model selection. Initial models such as logistic regression and decision tree were unable to handle the complexity of the data, hence the choice of the final model as MLP.
   - Another limitation is the computational intensity to train the MLP, especially for hyperparameter tuning. It demanded enough resources and time that might constitute a constraint in a real-world application with an even bigger dataset.
   - Furthermore, due to time constraints and the emphasis on development and optimization of the model, it was not possible to develop a UI for the exoplanet detection system. This was not an explicit objective, however, this type of UI would have opened it up for a lot of end-users, like astronomers or researchers who do not have a technical background in machine learning.
3. **Research Question Evaluation:**
   - The research question focused on how machine learning approaches could be adapted and optimized to detect exoplanets. The project effectively answered this question by demonstrating that deep learning models, particularly the MLP Neural Network, can be highly effective when carefully tuned. The success of the MLP suggests that deep learning approaches can handle the complexity of astronomical data better than traditional models.

- However, there were limitations in fully exploring all possible machine learning approaches. For example, the project did not include an in-depth exploration of ensemble methods, which might have offered additional insights or improvements in detection accuracy. Additionally, the impact of the chosen methods on model interpretability was not fully addressed, which could be an area for future research.
- The project did not explore the practical deployment aspects, such as creating a UI or integrating the model into a real-time system, which are crucial for practical applications of the research. This limitation suggests that while the project successfully addressed the core research question, further work is needed to ensure the model's applicability in real-world scenarios.

4. **Unmet Objectives:**
   - The model had already reached a high accuracy, although there was always room for improvement. No ensembling of the different models used by the project itself was applied, which potentially would increase performance.
   - There was one very noticeable thing missing among all its attributes: it wasn't provided with the development of a user interface. A user interface would certainly have conferred much added value on the usability of the model, making it more easily deployable to other users. As future work, this could considerably increase the ease with which the current tool could be applied in general across the scientific community.

7. CONCLUSION AND FUTURE WORK

This project concerned the use of machine learning in the detection of exoplanets, leveraging other classical machine learning algorithms together with deep learning techniques. Exoplanets are basically defined as planets that orbit around stars other than the sun and have long been the subject of much interest and scrutiny within the astronomical community because of their potential to sustain life. Detecting such celestial bodies proves to be a challenging task, however, due to the small sizes of these planets and the considerable distance from Earth. Furthermore, the class imbalance in intrinsic datasets is to a substantial extent: the frequency of exoplanet occurrences is much less compared to the frequency of non-exoplanet occurrences, making the exoplanet detection even more complicated.

In the first phase of the project, we investigated the application of classical machine learning models, such as Logistic Regression, Decision Trees, and XGBoost. These models have already been known for their effectiveness on structured datasets, especially on high-dimensional ones, like the light flux data in this case. But because of complexity and high dimensionality, further treatment by using more advanced techniques is required.

For a deeper exploration as the project developed, deep learning artificial neural networks, in particular Multilayer Perceptrons, were observed to model the dataset's details better. Being a neural network, MLPs can capture non-linear relationships within the data, which are quite important for effective identification of the very subtle patterns associated with exoplanets. Next came widening the scope of this project to not only include implementing the MLP neural networks using PyTorch but also to gain more flexibility in the architecture and therefore the parameters of this model.

Imbalanced classes in datasets were taken care of by the Synthetic Minority Over-sampling Technique (SMOTE) for balancing, whereas the handedness of this technique focuses on the creation of synthetic samples for the minority class to make the model non-biased towards the majority class. Another technique used was Principal Component Analysis (PCA), which would help in reducing the number of features while keeping the most significant variance retained, thus allowing for the model to be much more effective and efficient.

The MLP neural network was completed with an accuracy of 98.04%, quite credible for the exoplanet detection process. This high level of accuracy underlines the potential of combining classical machine learning techniques with deep learning approaches for the development of robust and reliable detection.

BIBLIOGRAPHY

Bahel, V. & Gaikwad, M., 2022. A Study of Light Intensity of Stars for Exoplanet Detection using Machine Learning. *2022 IEEE Region 10 Symposium (TENSYMP),* pp. 1-5.

Batalha, N., 2014. Exploring exoplanet populations with NASA"s Kepler Mission. *Proceedings of the National Academy of Sciences - PNAS,* 111(35), p. 12647–12654.

Blagus, R. & Lusa, L., 2023. SMOTE for high-dimensional class-imbalanced data. *PloS one ,* 18(6).

Chawla, N. V., Bowyer, K. W., Hall, L. O. & Kegelmeyer, W. P., 2002. SMOTE: Synthetic minority over-sampling technique. *The Journal of artificial intelligence research ,* Volume 16, pp. 321 - 357.

Herur, A. N., Tajmohamed, R. & Ponsam, J. G., 2022. Detection and classification of exoplanets using hybrid kNN model. *International Conference on Computer Communication and Informatics (ICCCI),* pp. 1-3.

imbalanced-learn, 2016. *imbalanced-learn.* [Online]
Available at: https://imbalanced-learn.org/stable/
[Accessed 09 04 2024].

Jagtap, R. et al., 2021. Habitability of Exoplanets using Deep Learning. *IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS),* pp. 1-6.

Kaggle, 2017. *Kaggle.* [Online]
Available at: https://www.kaggle.com/datasets/keplersmachines/kepler-labelled-time-series-data/data
[Accessed 14 03 2024].

Liang, X. et al., 2020. LR-SMOTE — An improved unbalanced data set oversampling based on K-means and SVM. *Knowledge-based systems ,* Volume 196.

Malik, A., Moster, B. P. & Obermeier, C., 2022. Exoplanet detection using machine learning. *Monthly notices of the Royal Astronomical Society ,* 513(4), pp. 5505 - 5516.

Nandan, A., Tripathi, V., Singh, D. P. & Gupta, A., 2022. Detection of exoplanets using classical machine learning techniques. *AIP Conference Proceedings ,* 2481(1), p. 8.

NASA, n.d. *NASA Exoplanet Exploration.* [Online]
Available at: https://exoplanets.nasa.gov/what-is-an-exoplanet/overview/
[Accessed 20 03 2024].

New Space Economy, n.d. *Exoplanet Detection: Methods and Chanllenges.* [Online]
Available at: https://newspaceeconomy.ca/2023/10/18/exoplanets-methods-and-challenges-of-detection/
[Accessed 21 04 2024].

Pearson, K. A., Palafox, L. & Griffith, C. A., 2018. Searching for exoplanets using artificial intelligence. *Monthly notices of the Royal Astronomical Society,* 474(1).

Seager, S., 2013. Exoplanet Habitability. *Science (American Association for the Advancement of Science),* 340(6132).

Sehgal, S. et al., 2014. Data analysis using principal component analysis. *International Conference on Medical Imaging, m-Health and Emerging Communication Systems (MedCom),* pp. 45-48.

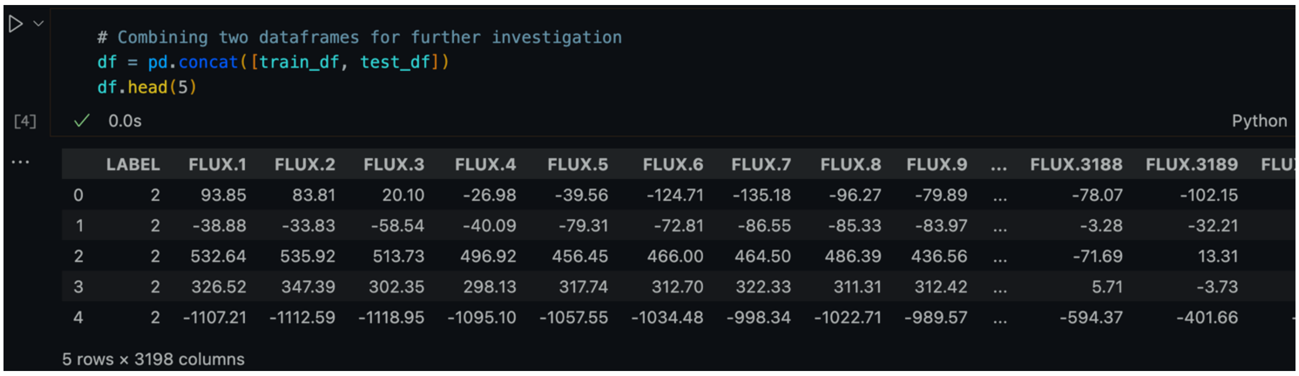
Shallue, C. J. & Vanderburg, A., 2018. Identifying Exoplanets with Deep Learning: A Five-planet Resonant Chain around Kepler-80 and an Eighth Planet around Kepler-90. *The Astronomical journal ,* 155(2), p. 94.

Sturrock, G. C., Manry, B. & Rafiqi, S., 2019. Machine Learning Pipeline for Exoplanet Classification. *SMU Data Science Review,* 2(1).

TESS, n.d. *How to find an exoplanet with TESS data.* [Online]
Available at: https://heasarc.gsfc.nasa.gov/docs/tess/HowToFindAnExoplanet-UserVersion.html
[Accessed 25 05 2024].

Tey, E. et al., 2023. Identifying Exoplanets with Deep Learning. V. Improved Light-curve Classification for TESS Full-frame Image Observations. *The American Astronomical Society ,* 165(3), p. 19.

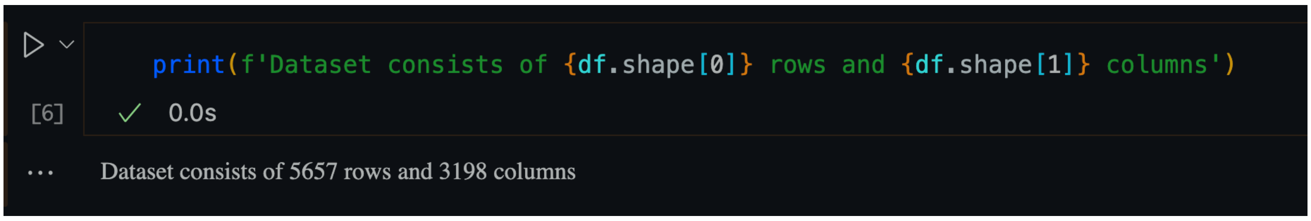
APPENDICES



```python
# Combining two dataframes for further investigation
df = pd.concat([train_df, test_df])
df.head(5)
```

```
[4]  ✓ 0.0s                                                                                     Python

   LABEL  FLUX.1   FLUX.2   FLUX.3   FLUX.4   FLUX.5   FLUX.6   FLUX.7  FLUX.8   FLUX.9 ... FLUX.3188 FLUX.3189 FLU
0      2   93.85    83.81    20.10   -26.98   -39.56  -124.71  -135.18  -96.27   -79.89 ...    -78.07   -102.15
1      2  -38.88   -33.83   -58.54   -40.09   -79.31   -72.81   -86.55  -85.33   -83.97 ...     -3.28    -32.21
2      2  532.64   535.92   513.73   496.92   456.45   466.00   464.50  486.39   436.56 ...    -71.69     13.31
3      2  326.52   347.39   302.35   298.13   317.74   312.70   322.33  311.31   312.42 ...      5.71     -3.73
4      2 -1107.21 -1112.59 -1118.95 -1095.10 -1057.55 -1034.48 -998.34 -1022.71 -989.57 ...   -594.37   -401.66  -

5 rows × 3198 columns
```

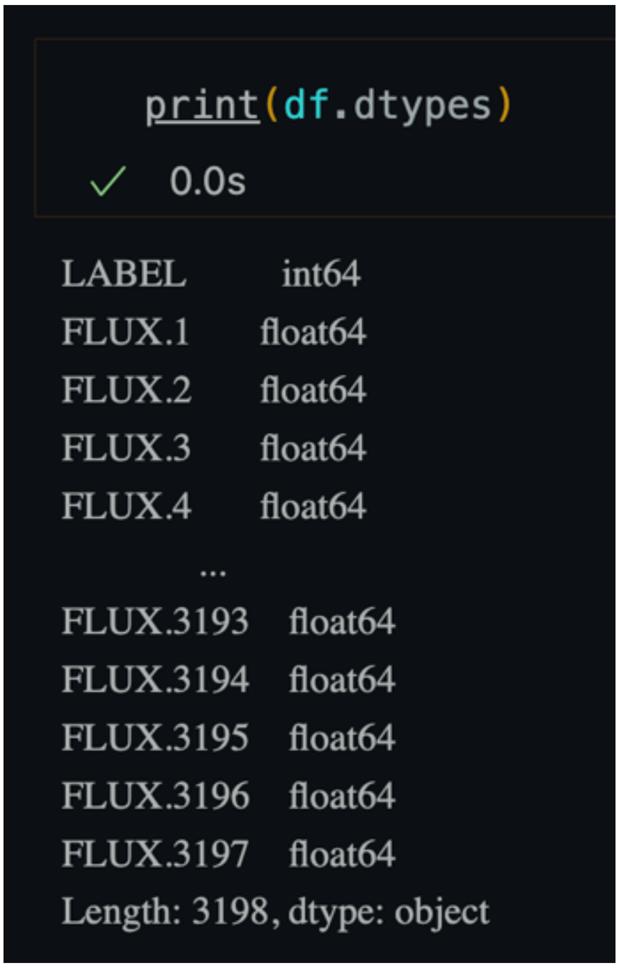
Appendix 1



```python
print(f'Dataset consists of {df.shape[0]} rows and {df.shape[1]} columns')
```

```
[6]  ✓ 0.0s

Dataset consists of 5657 rows and 3198 columns
```

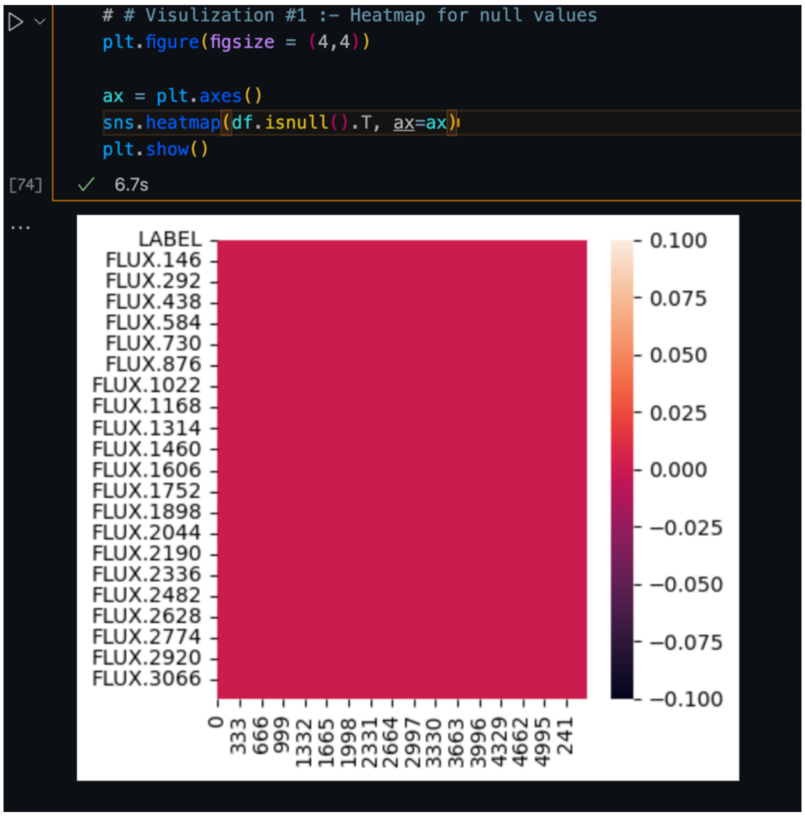
Appendix 2



```python
print(df.dtypes)
```

```
✓ 0.0s

LABEL        int64
FLUX.1     float64
FLUX.2     float64
FLUX.3     float64
FLUX.4     float64
              ...
FLUX.3193  float64
FLUX.3194  float64
FLUX.3195  float64
FLUX.3196  float64
FLUX.3197  float64
Length: 3198, dtype: object
```

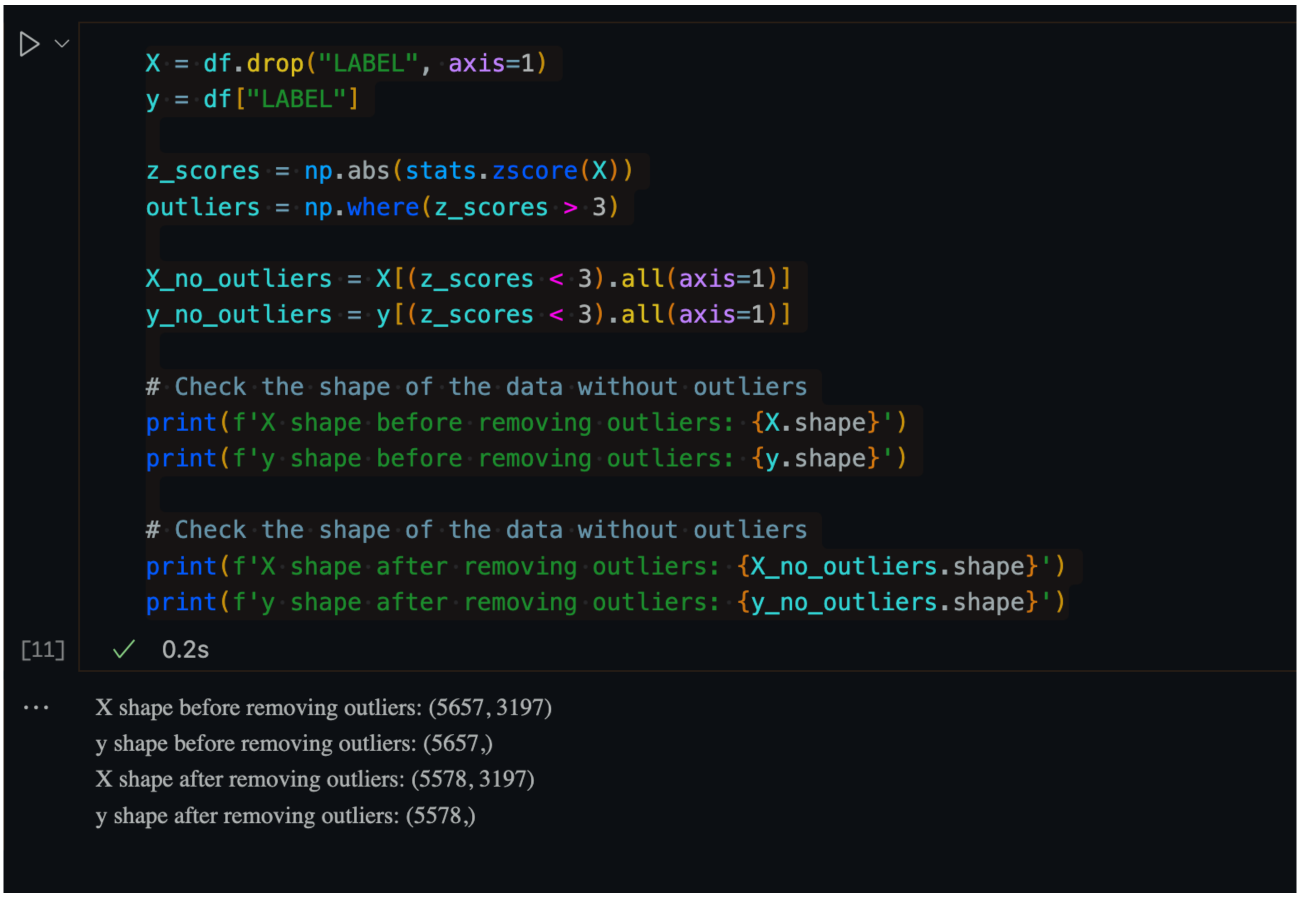
Appendix 3



```
# # Visulization #1 :- Heatmap for null values
plt.figure(figsize = (4,4))

ax = plt.axes()
sns.heatmap(df.isnull().T, ax=ax)
plt.show()
```

```
[74]  ✓ 6.7s
```

Appendix 4

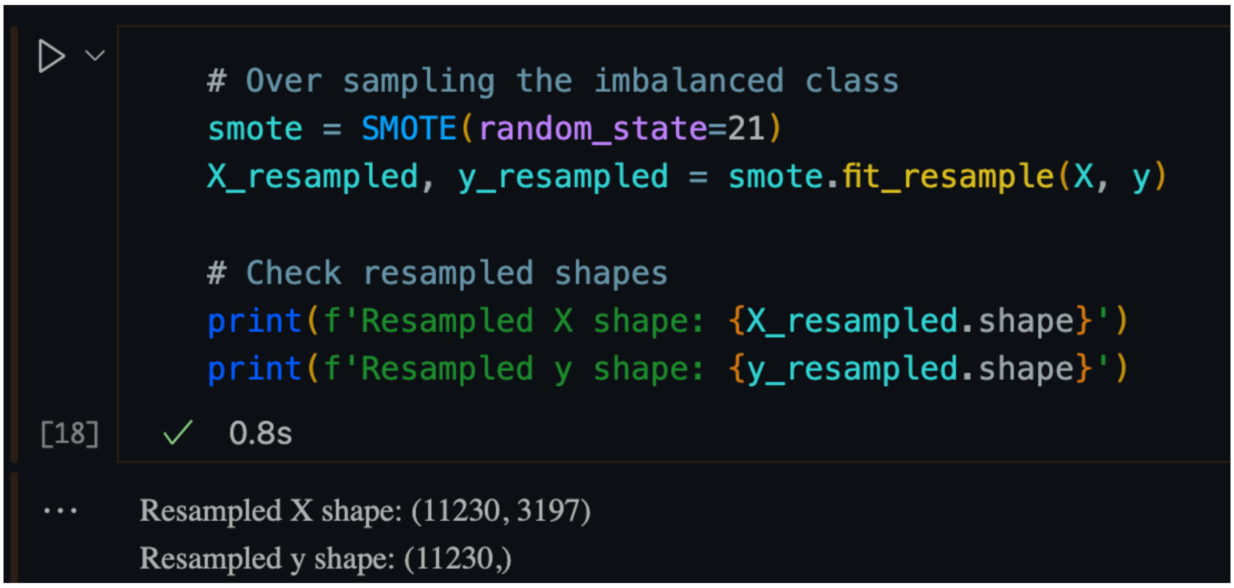


```
X = df.drop("LABEL", axis=1)
y = df["LABEL"]

z_scores = np.abs(stats.zscore(X))
outliers = np.where(z_scores > 3)

X_no_outliers = X[(z_scores < 3).all(axis=1)]
y_no_outliers = y[(z_scores < 3).all(axis=1)]

# Check the shape of the data without outliers
print(f'X shape before removing outliers: {X.shape}')
print(f'y shape before removing outliers: {y.shape}')

# Check the shape of the data without outliers
print(f'X shape after removing outliers: {X_no_outliers.shape}')
print(f'y shape after removing outliers: {y_no_outliers.shape}')
```

```
[11]  ✓ 0.2s
```

```
X shape before removing outliers: (5657, 3197)
y shape before removing outliers: (5657,)
X shape after removing outliers: (5578, 3197)
y shape after removing outliers: (5578,)
```

Appendix 5

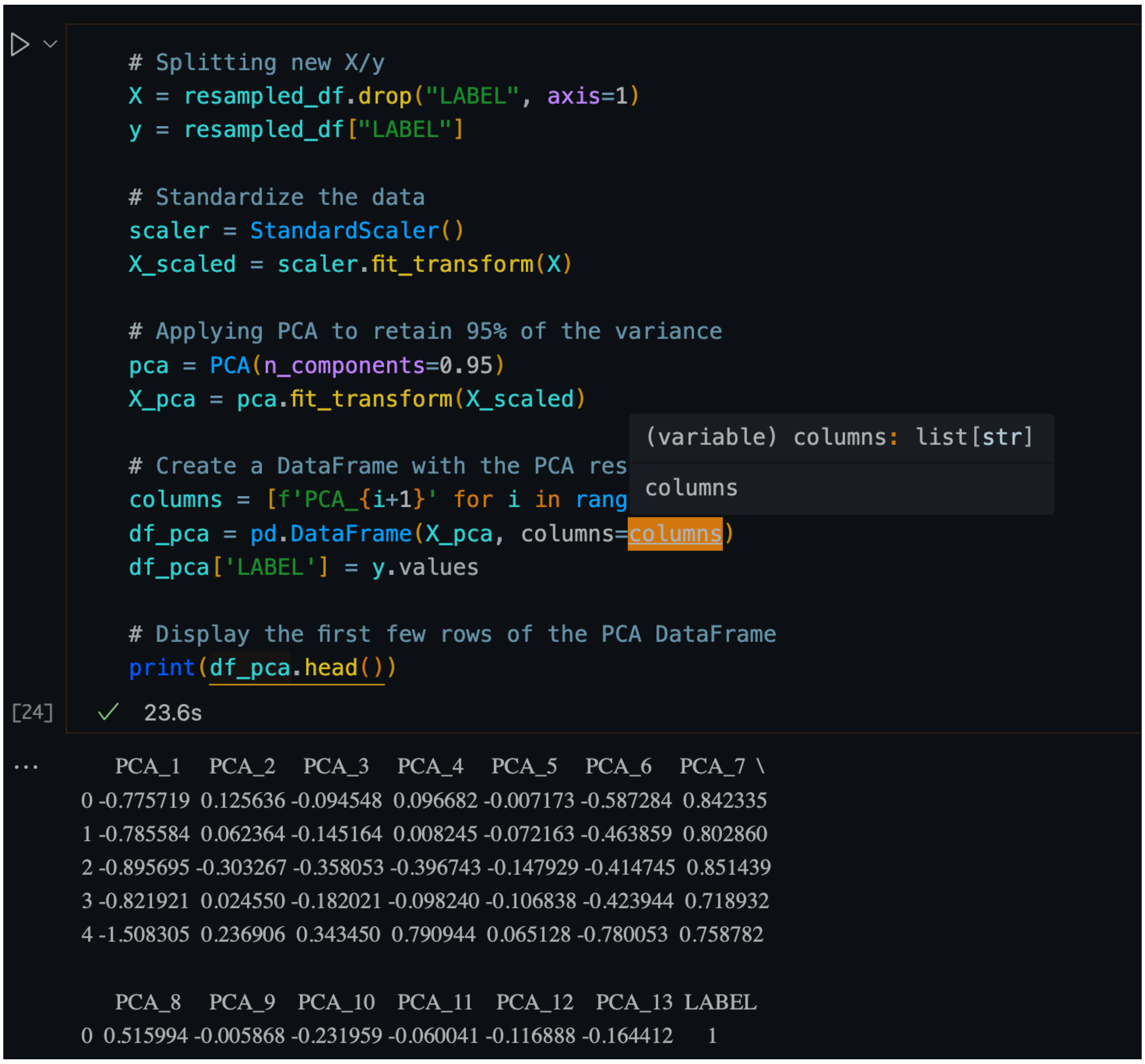


```
# Over sampling the imbalanced class
smote = SMOTE(random_state=21)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Check resampled shapes
print(f'Resampled X shape: {X_resampled.shape}')
print(f'Resampled y shape: {y_resampled.shape}')
```

[18] ✓ 0.8s

```
Resampled X shape: (11230, 3197)
Resampled y shape: (11230,)
```

Appendix 6

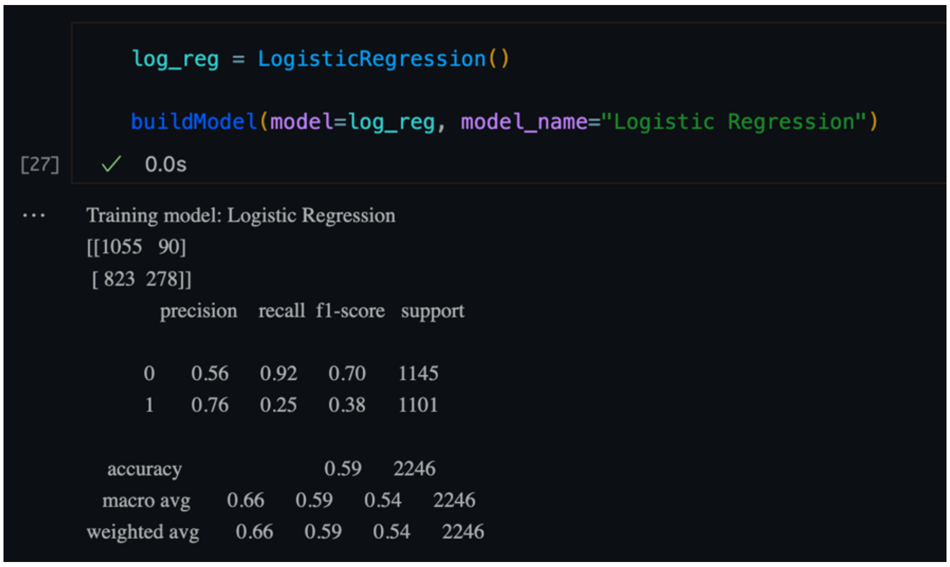


```
# Splitting new X/y
X = resampled_df.drop("LABEL", axis=1)
y = resampled_df["LABEL"]

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Applying PCA to retain 95% of the variance
pca = PCA(n_components=0.95)
X_pca = pca.fit_transform(X_scaled)

# Create a DataFrame with the PCA res
columns = [f'PCA_{i+1}' for i in rang
df_pca = pd.DataFrame(X_pca, columns=columns)
df_pca['LABEL'] = y.values

# Display the first few rows of the PCA DataFrame
print(df_pca.head())
```

[24] ✓ 23.6s

```
      PCA_1     PCA_2     PCA_3     PCA_4     PCA_5     PCA_6     PCA_7  \
0 -0.775719  0.125636 -0.094548  0.096682 -0.007173 -0.587284  0.842335
1 -0.785584  0.062364 -0.145164  0.008245 -0.072163 -0.463859  0.802860
2 -0.895695 -0.303267 -0.358053 -0.396743 -0.147929 -0.414745  0.851439
3 -0.821921  0.024550 -0.182021 -0.098240 -0.106838 -0.423944  0.718932
4 -1.508305  0.236906  0.343450  0.790944  0.065128 -0.780053  0.758782

      PCA_8     PCA_9    PCA_10    PCA_11    PCA_12    PCA_13  LABEL
0  0.515994 -0.005868 -0.231959 -0.060041 -0.116888 -0.164412      1
```

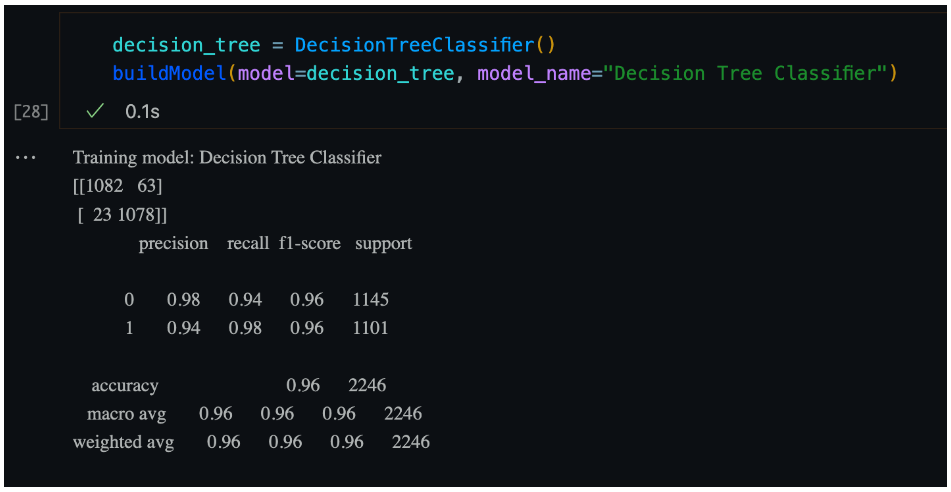
Appendix 7



```
log_reg = LogisticRegression()

buildModel(model=log_reg, model_name="Logistic Regression")
[27]  ✓ 0.0s
```

```
Training model: Logistic Regression
[[1055   90]
 [ 823  278]]
              precision    recall  f1-score   support

           0       0.56      0.92      0.70      1145
           1       0.76      0.25      0.38      1101

    accuracy                           0.59      2246
   macro avg       0.66      0.59      0.54      2246
weighted avg       0.66      0.59      0.54      2246
```

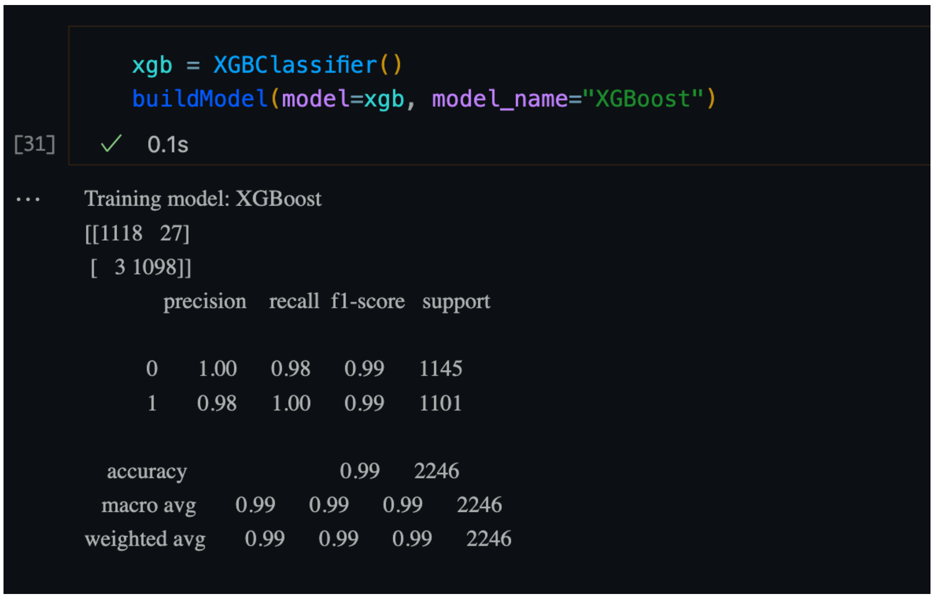
Appendix 8



```
decision_tree = DecisionTreeClassifier()
buildModel(model=decision_tree, model_name="Decision Tree Classifier")
[28]  ✓ 0.1s
```

```
Training model: Decision Tree Classifier
[[1082   63]
 [  23 1078]]
              precision    recall  f1-score   support

           0       0.98      0.94      0.96      1145
           1       0.94      0.98      0.96      1101

    accuracy                           0.96      2246
   macro avg       0.96      0.96      0.96      2246
weighted avg       0.96      0.96      0.96      2246
```

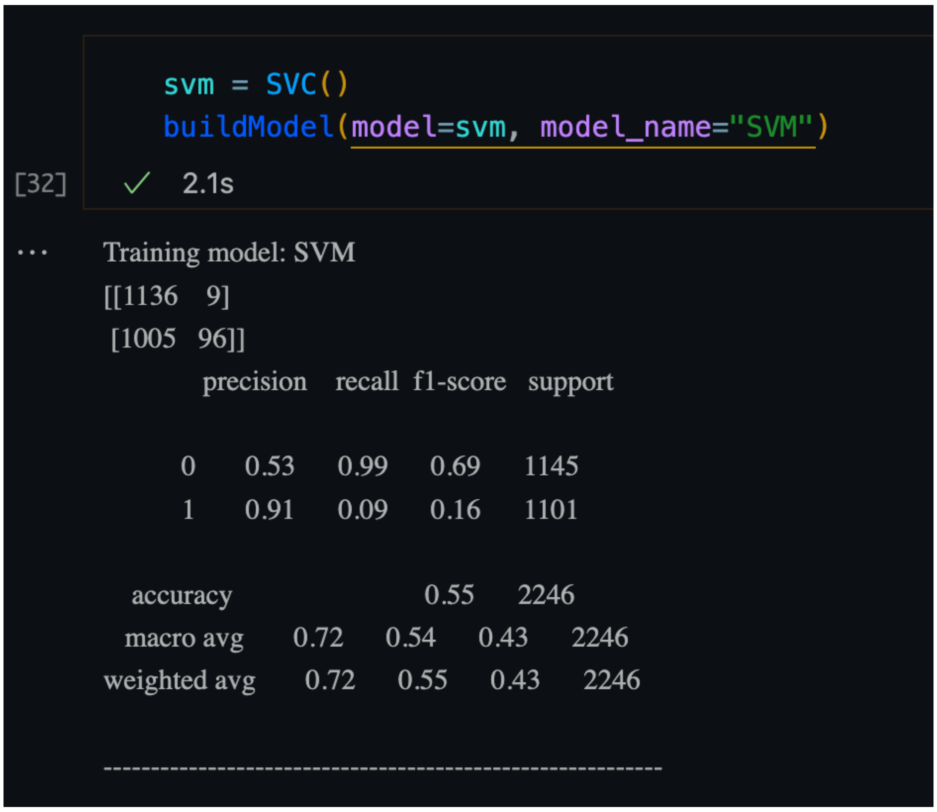
Appendix 9



```
xgb = XGBClassifier()
buildModel(model=xgb, model_name="XGBoost")
[31]  ✓ 0.1s
```

```
Training model: XGBoost
[[1118   27]
 [   3 1098]]
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      1145
           1       0.98      1.00      0.99      1101

    accuracy                           0.99      2246
   macro avg       0.99      0.99      0.99      2246
weighted avg       0.99      0.99      0.99      2246
```

Appendix 10

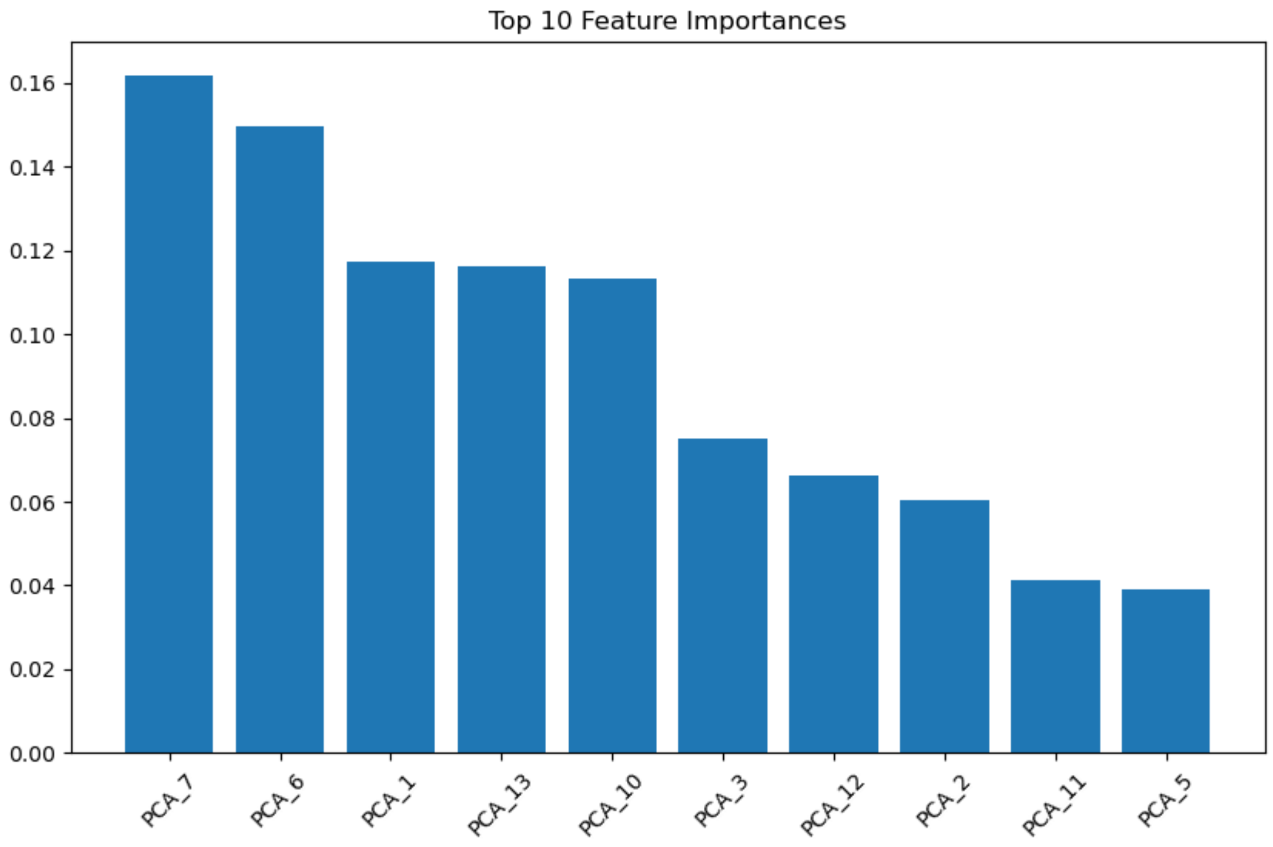


```
svm = SVC()
buildModel(model=svm, model_name="SVM")
```

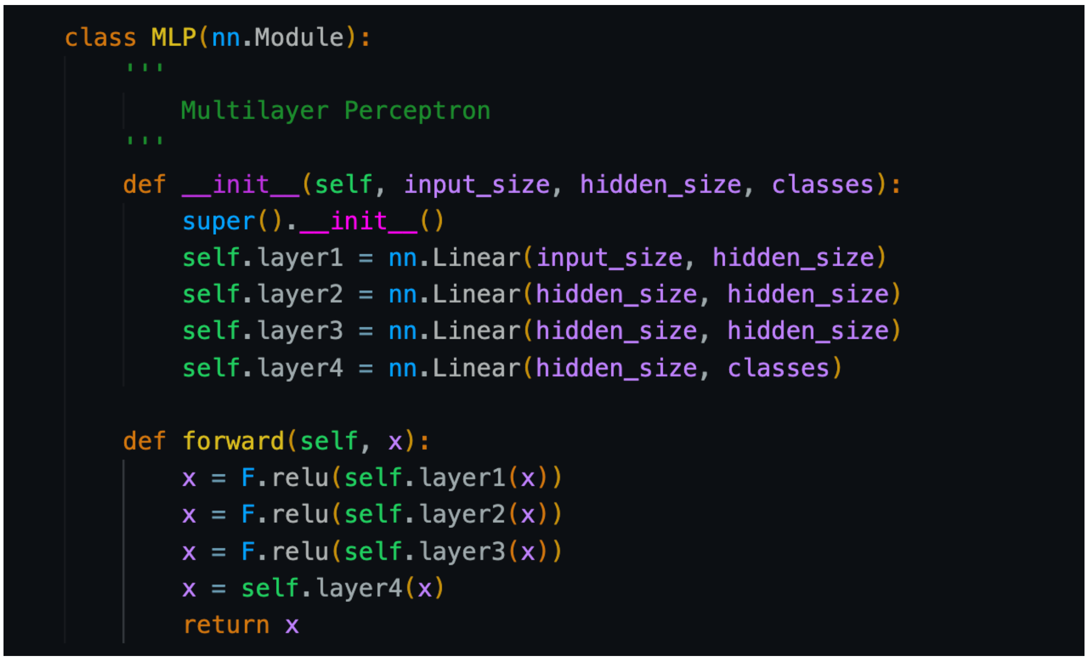
```
[32]  ✓ 2.1s
```

```
... Training model: SVM
    [[1136    9]
     [1005   96]]
                  precision    recall  f1-score   support

               0       0.53      0.99      0.69      1145
               1       0.91      0.09      0.16      1101

        accuracy                           0.55      2246
       macro avg       0.72      0.54      0.43      2246
    weighted avg       0.72      0.55      0.43      2246


    ------------------------------------------------------------
```

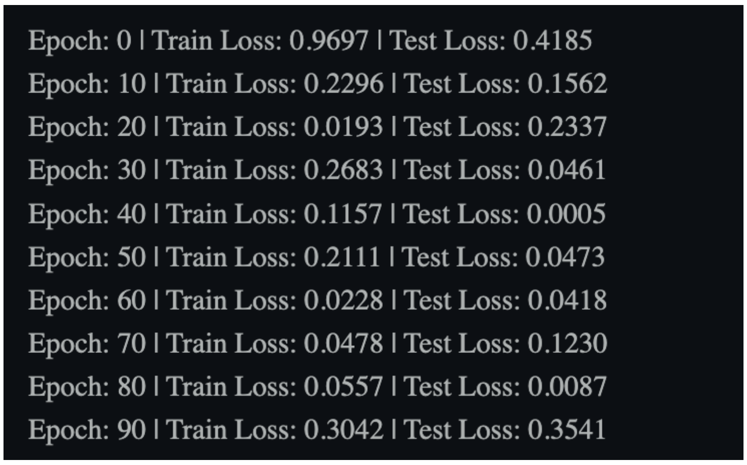
Appendix 11



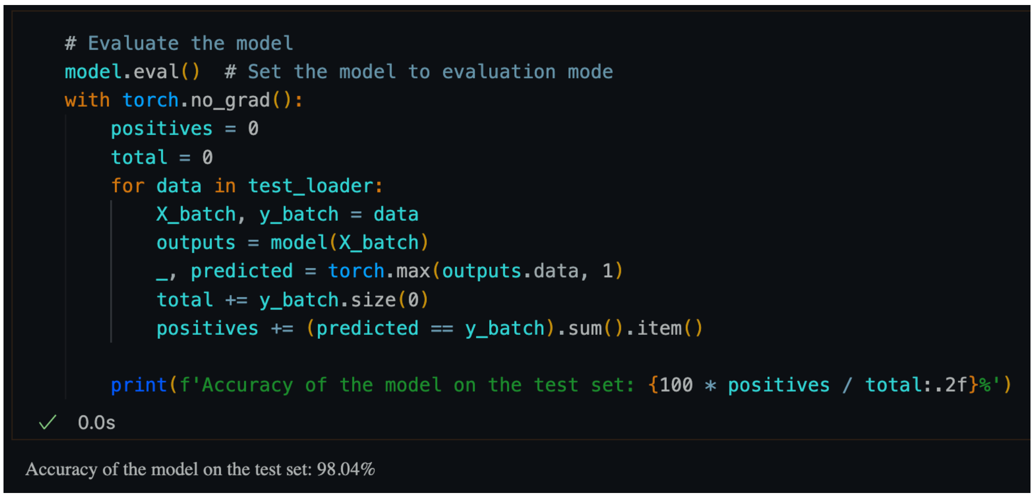
Appendix 12


```python
class MLP(nn.Module):
    '''
        Multilayer Perceptron
    '''
    def __init__(self, input_size, hidden_size, classes):
        super().__init__()
        self.layer1 = nn.Linear(input_size, hidden_size)
        self.layer2 = nn.Linear(hidden_size, hidden_size)
        self.layer3 = nn.Linear(hidden_size, hidden_size)
        self.layer4 = nn.Linear(hidden_size, classes)

    def forward(self, x):
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))
        x = F.relu(self.layer3(x))
        x = self.layer4(x)
        return x
```

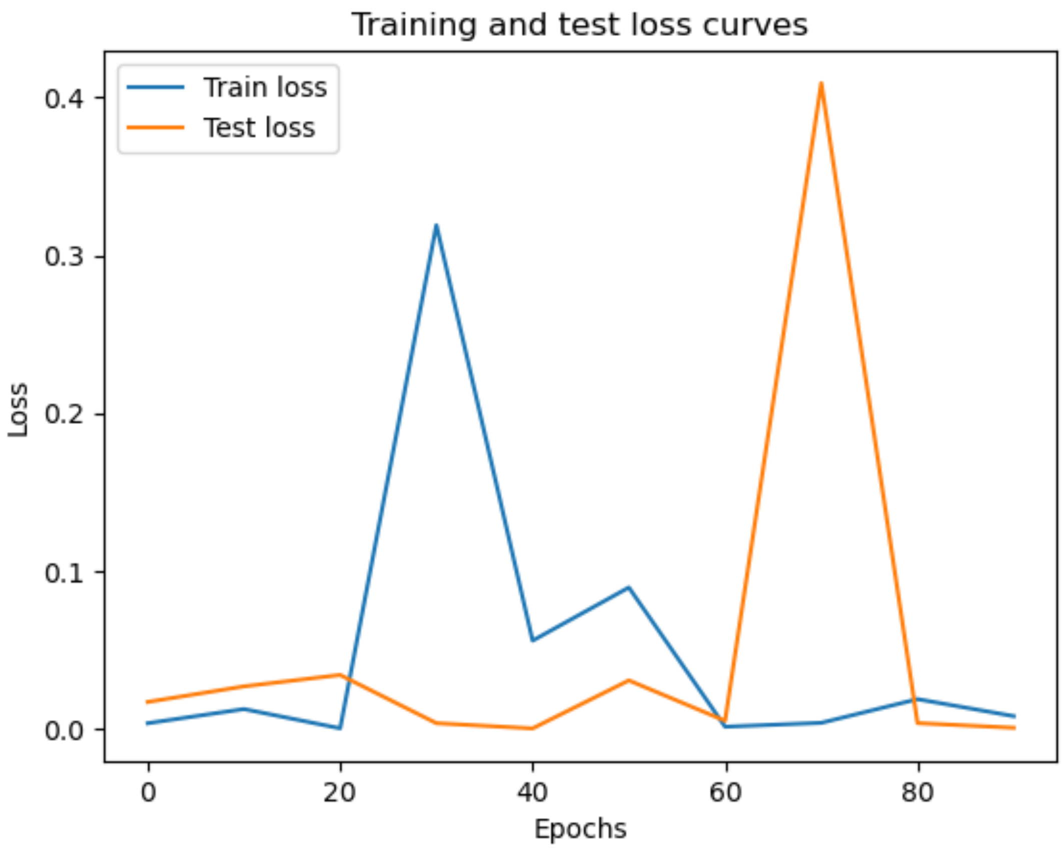
Appendix 13


```
Epoch: 0 | Train Loss: 0.9697 | Test Loss: 0.4185
Epoch: 10 | Train Loss: 0.2296 | Test Loss: 0.1562
Epoch: 20 | Train Loss: 0.0193 | Test Loss: 0.2337
Epoch: 30 | Train Loss: 0.2683 | Test Loss: 0.0461
Epoch: 40 | Train Loss: 0.1157 | Test Loss: 0.0005
Epoch: 50 | Train Loss: 0.2111 | Test Loss: 0.0473
Epoch: 60 | Train Loss: 0.0228 | Test Loss: 0.0418
Epoch: 70 | Train Loss: 0.0478 | Test Loss: 0.1230
Epoch: 80 | Train Loss: 0.0557 | Test Loss: 0.0087
Epoch: 90 | Train Loss: 0.3042 | Test Loss: 0.3541
```

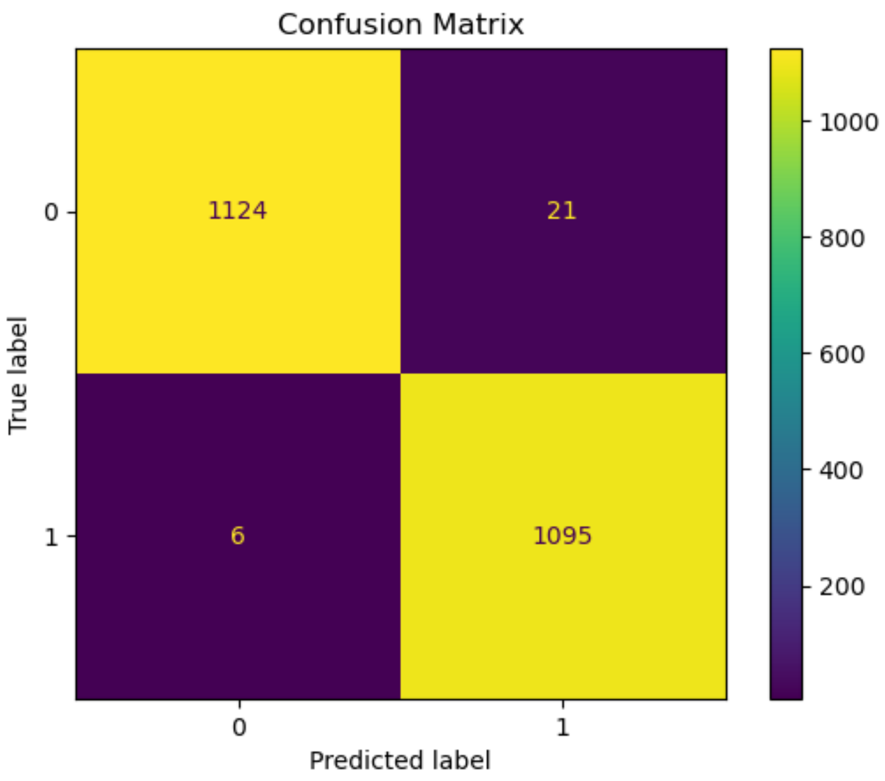
Appendix 14


```python
# Evaluate the model
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    positives = 0
    total = 0
    for data in test_loader:
        X_batch, y_batch = data
        outputs = model(X_batch)
        _, predicted = torch.max(outputs.data, 1)
        total += y_batch.size(0)
        positives += (predicted == y_batch).sum().item()

    print(f'Accuracy of the model on the test set: {100 * positives / total:.2f}%')
```
```
✓ 0.0s
```
```
Accuracy of the model on the test set: 98.04%
```

Appendix 15



Appendix 16



Appendix 17