A MACHINE LEARNING APPROACH TO EXOPLANETS DISCOVERY

August 11th, 2016

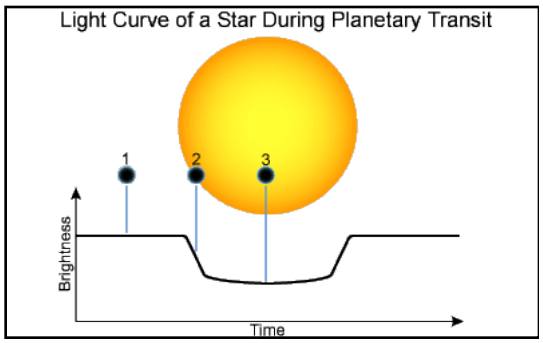
DEFINITION

Project Overview

In early October, twenty years ago, Michel Mayor and Didier Queloz of the Geneva Observatory in Switzerland made an announcement [1] destined to become history: the discovery of an Exoplanet orbiting a star similar to the Sun, the 51 Pegasi star, 48 light-years away towards the constellation of Pegasus. For the uninformed reader, an exoplanet is a planet outside our solar system orbiting a star other than our sun. Since then, the race to the discovery of exoplanets began. However, just recently, with the advancement in technology and the launch of space telescopes like Kepler, exoplanetary discovery has boomed. Since this is just the beginning to exoplanetary discovery, more mission are being scheduled to discover exoplanets, the purpose of this project is to create a model, using machine learning, that will allow scientist and researchers to simplify the process of discovery.

Problem Statement

The main problem in exoplanetary discovery is that scientist and researchers take a long time to check if there is a candidate planet orbiting a star. This project proposes an alternative method, using machine learning, to discover exoplanets orbiting a star. Before explaining the approach taken to solve this problem, some terminology has to be described:



- A **Light Curve** is a graph that represent the change in brightness of a star over time.
- A **Planetary Transit** is a astronomical method used for the discovery of exoplanets. The method consists in the detection of the decrease in brightness of the light curve of a star when a planet passes in front of the parent star. The decrease in brightness is related to the relative size of the parent star, the planet and its orbit. As you can see in the Figure on the right, as the planet passes through the star, the brightness of the star diminish. To understand this topic further, I suggest watching this youtube video.
- The **Kepler Telescope** is a space telescope whose purpose is the research and confirmation of Earth-like planets orbiting stars other than the Sun. The Kepler telescope was "specifically designed to monitor a portion of our region of the Milky Way and discover dozens of Earth-like planets in or near the habitable zone and determine how many of the billions of stars in our galaxy possess planets." [2]

The Kepler telescope, which has been in function since 2009, thanks to scientists, has confirmed 2290 planets, discovered 4120 False Positive exoplanets, and has still 2416 candidates Planets to be confirmed or debunked. The approach that I took was to work with the raw data collected by the Kepler telescope between 2009 and 2013, use the already confirmed exoplanets and false positive to train a machine learning algorithm to recognize future planets. The last step was to apply the trained algorithm to the still candidate exoplanet and check the results.

Metrics

During this project, one type of metrics have been used: Precision.

- ***Precision*** is a pretty common metric used in binary classifiers. As a matter of fact, differently from the f1 metric, which is a weighted average of precision and recall, precision only take into account true positive over true positive plus false positive. The choice of using precision is due to the fact that this is a discovery project; therefore; I am more concern of the true positive results. Indeed, if a planet is categorized as "false negative" it status will remain still candidate, because there are no strong evidences to promote it as false negative.

$$\text{Precision} = \frac{tp}{tp + fp}$$

Moreover I used the confusion matrix for a visual justification on how the model was performing, checking false positive and false negatives.

- The ***confusion matrix*** is another common metric used in binary classifiers. It is a table that describes the performance of a classifier, showing the number of true positive, false positive, true negative, false negative.

n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Frameworks and Libraries

Except for all the usual python Libraries used during the course of the Nanodegree such as Sklearn, Numpy, and Pandas, the following libraries have been used in the project:

- ***Kplr*** is a python interface to the Kepler data. It allows to search the Kepler data catalog and retrieve the light-curves of a planet, just referencing the KOI (Kepler Object of Interest) name. ***[Need to be installed]***
- ***PyFITs*** is a python library that allows reading and processing of FITs files. Flexible Image Transport System or more commonly (FITS) is an open standard file format useful for storage of scientific and other images. FITS files are most commonly used in astronomy. ***[Need to be installed]***
- ***Joblib*** is a python pipelining for simple parallel computing. ***[Need to be installed]***
- ***PyKE*** is a software developed by nasa that allows to process and normalize the Kepler Data. ***[Already included in the project]***

The first three libraries are easily installable using pip. Here are the commands for the terminal:

pip install kplr
pip install pyfits
pip install joblib

ANALYSIS

Data Exploration

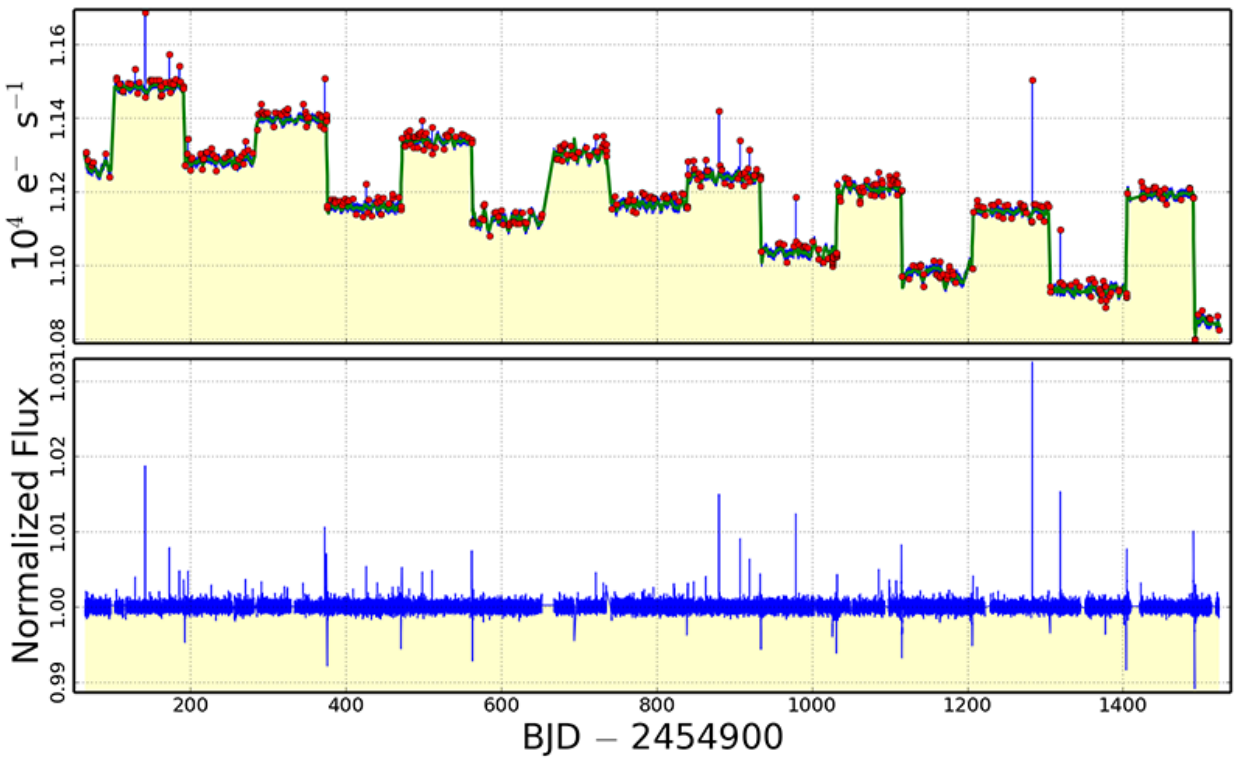
The first step towards analyzing and creating a model to recognize exoplanets was to find a reliable source to collect information about the exoplanets processed by the Kepler telescope. Thanks to the ***Kepler Exoplanet Archive***, I was able to find a list of all the planets processed by the Kepler telescope with their relative Kepler Object of Interest(KOI), and status (Confirmed, False Positive, and Candidate). Subsequently, I used the Kplr python library to pull the light curves of each planet retrieved from the Kepler Exoplanet Archive. The light curves are a time series representation; therefore, each time data point in the data of the light curves has some attribute related with it [3]. The attributes of the light curve stored in the FITs files include:

- ***TIME:*** The time at the mid-point of the cadence in BKJD (Kepler Barycentric Julian Day).
- ***TIMECORR:*** The barycenter correction calculated by the pipeline plus the time slice correction.
- ***SAP_FLUX:*** The flux in units of electrons per second contained in the optimal aperture pixels collected by the spacecraft.
- ***SAP_FLUX_ERR:*** The error in the simple aperture photometry as determined by PA in electrons per second.
- ***SAP_BKG:*** The total background flux summed over the optimal aperture in electrons per second.
- ***SAP_BKG_ERR:*** The 1-sigma error in the simple aperture photometry background flux.
- ***PDCSAP_FLUX:*** The flux contained in the optimal aperture in electrons per second after the PDC module has applied its co-trending algorithm.
- ***PDCSAP_FLUX_ERR:*** The 1-sigma error in PDC flux values.
- ***SAP_QUALITY:*** Flags containing information about the quality of the data.
- ***PSF_CENTR1:*** The column centroid calculated by fitting the point-spread function.
- ***PSF_CENTR1_ERR:*** The 1-sigma error in PSF-fitted column centroid.
- ***PSF_CENTR2:*** The row centroid calculated by fitting the point-spread function.
- ***PSF_CENTR2_ERR:*** The 1-sigma error in PSF-fitted row centroid.
- ***MOM_CENTR1:*** The column value for the flux-weighted centroid position of the target at this cadence.
- ***MOM_CENTR1_ERR:*** The 1-sigma error in the column value for the first moment centroid position.
- ***MOM_CENTR2:*** The row value for the flux-weighted centroid position of the target at each cadence.
- ***MOM_CENTR2_ERR:*** The 1-sigma error in the row value for the first moment centroid position.
- ***POS_CORR1:*** The column component of the local image motion calculated from the motion polynomials.
- ***POS_CORR2:*** The row component of the local image motion calculated from the motion polynomials.

The light curve files for each star are divided into quarters; thus, the quarters needed to be stitched together. Moreover, since the Flux of electrons came with different sorts of systematic and environmental errors, the flux needed to be normalized and detrended. This step was easily accomplished thanks to PyKE library, which allows stitching quarters together and detrending data. The picture below shows the light curve from the star K-752 before and after the normalization process.

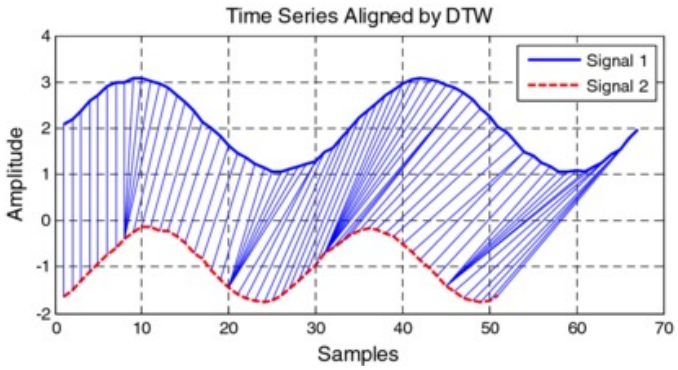
Subsequently, I needed a way to compare the time series of flux of two stars. Therefore, I ended up using an algorithm called Dynamic Time Warping (DTW), which is widely common for the comparison of two time series [4][5]. The Dynamic Time Warping algorithm finds an optimal match between two sequences of feature vectors which allows for stretched and compressed sections of the sequence. I

decided to choose a star that had a regular planetary transit in its light curve as a baseline for comparison.



One thing to be notice is that the DTW algorithm do not always find a comparison value for two time series. Therefore, after processing all the planets, form the 8825 planets processed by the Kepler telescope, 18.03% (1595 planets) is eliminated from the training data for the machine learning implementation.

The picture on the right gives a little bit of intuition on how the DTW algorithm works.

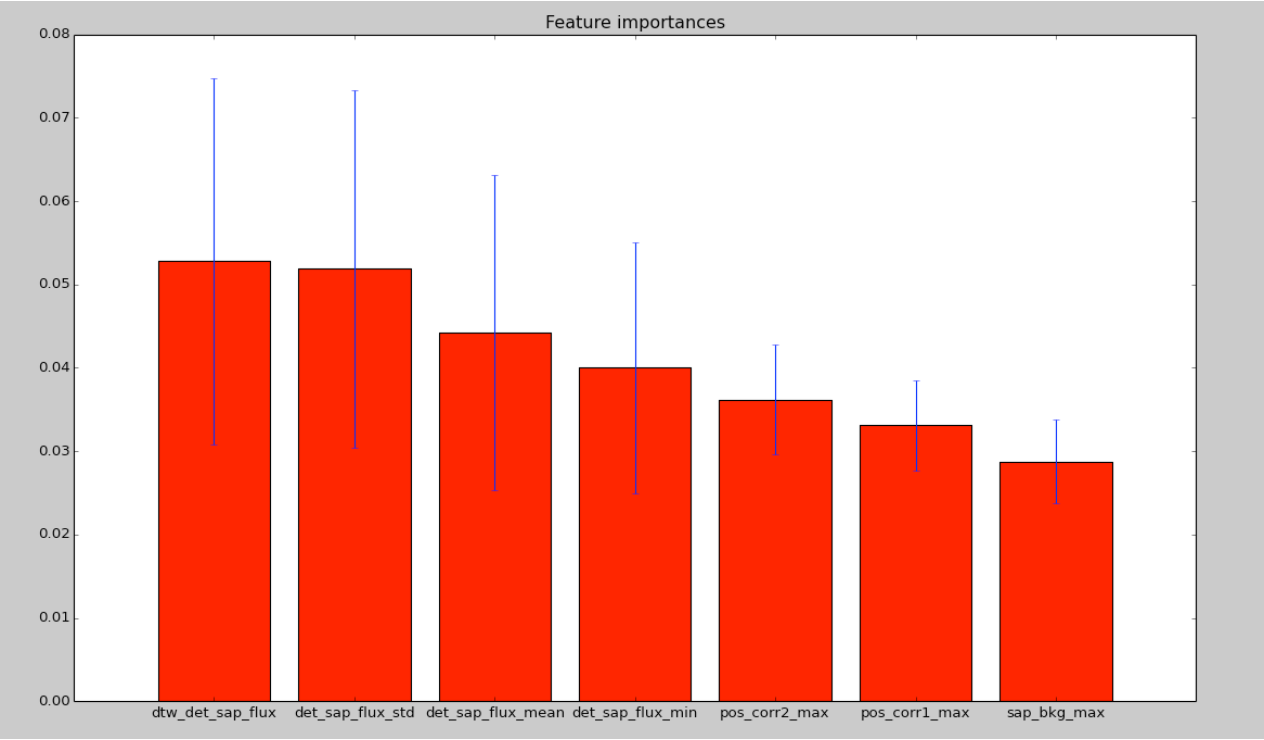


Finally, I created a table with the DTW comparison of the 3 different fluxes and the statistical attributes (mean, std, min, max) of the following: ***SAP_FLUX, SAP_BKG, PDCSAP_FLUX, DETSAP_FLUX, MOM_CENTR1, MOM_CENTR2, POS_CORR1, POSCORR2***. These values provided important features of a light curve data.

Exploratory Visualization

The figure below shows the 7 most relevant features of the data set. I chose seven oust for the sake of visualization. You can see all of them on the implementation section. As you can see, the choice of detrending the data and apply the Dynamic Time Warping algorithm on the light curve, really paid off. Indeed, the most relevant feature is the DTW of the detrended flux, and the second, third and forth

most relevant features are the standard deviation, mean, and minimum of the detrended flux. This graph has been generated fitting the entire data to a Random Forrest Classifiers, extracting the importance of each features. In the implementation phase you will see how this plays a role on reducing the number of features. The blue lines represent the standard deviation of each feature.



Algorithms and Techniques

The algorithms I will use for this project are the most useful methods used for classification problems. This includes:

- **Naive Bayes.** I choose Naive Bayes because is a super simple classifier. Indeed, if the conditional independence assumption holds, a Naive Bayes classifier will converge quicker than the other 3 classifiers I have chosen.
- **Logistic Regression.** I choose Logistic Regression because it is a classifier that has a lots of ways to regularize the model, and I don't have to to worry about correlated features, differently from Naive Bayes.
- **Decision Tree.** I choose Decision Tree because it is a non-parametric classifier and I don't have to worry if the data is not linearly separable.
- **Support Vector Machines (SVM).** I choose SVM because of it Hight accuracy and theoretical guarantees regarding overfitting.

I will use the precision metric to see which one performs best according to the dataset and then I will tune the parameters for that algorithm to improve its performance. I decided to follow this route due to the unpredictability and similarity of the dataset data points. As a matter of fact, each data point is very similar even if the two classes are very distinct (Confirmed, False Positive).

Benchmark

Since this is a research project, there are not real clear benchmarks. However, thanks to the Confusion Matrix and K-folds method, some metrics, to test the performance of the different algorithms, are available. I am expecting that the best algorithm will have an precision metric of at least 60%. I choose 60% because I believe that it will be able to fit the data; however; it will not be able to make good predictions due to the fact that the data set is really sensitive to and very close, even if the categories are really different. Moreover, when applying the model to the still candidate planets, which the researchers are still evaluating, I am expecting to find no more than 5 planets to be confirmed, with a discovery rate lower than 1%. This low margin is due to the fact that human have categorized the dataset of exoplanet as Confirmed or False Negative. Therefore, I do not believe that the algorithm, with that human bias pre-built into it, will able to recognize plates that the humans could not. I believe will find some exoplanets that the human eye could not recognize.

METHODOLOGY

Data Preprocessing

Processing the Kepler data was the most difficult task of the project. As a matter of fact, each start had approximately 1 GB of information. Moreover, the algorithms used to stitch and detrend the data were of order O(n^2), adding more delay to the process. To process all of the planets, I had to rent a supercomputer with 32 cores through ***Domino Data Labs***. If I would have processed all the data with my computer that has 4 cores, the process would have taken 30 Days. Instead, with Domino's platform, it took approximately 2 days.

The first step in the data analysis process was to read the list of planets from the Kepler telescope collected from the Kepler Exoplanet Archive. Subsequently, get some sense of the data.

The next step was to create a data-frame that would hold the processed information for each planet. Next step was to collect the information of the baseline light curve chosen prior and get the time series of the ***SAP_FLUX, SAP_BKG, PDCSAP_FLUX, DETSAP_FLUX.***

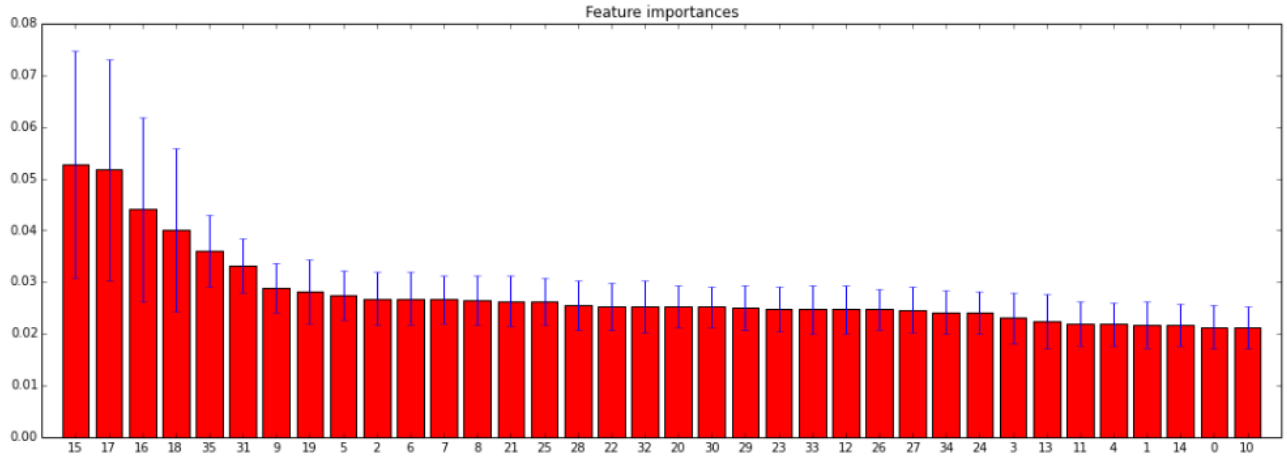
The next step was to create a function that processed the data for a single planet. This function takes the index of the planets data-frame, the planet data-frame, the output data-frame and the baseline data, processing all the data from a single planet, adding a row to the output data-frame.

Finally, I have implemented the parallel function that would process all the planet. For the sake of this notebook, I set up the number of core equal to 1, since not all the computers would be able to parallel compute this data. Moreover, since to process each planet it takes approximately 3 minutes, I set up a limit of 2 planets to be processed.

Lastly, the data-frame is converted to a CSV file, to be used later to apply Machine Learning, and the head of the data-frame is presented.

Implementation

The first step toward the machine learning analysis was to read the data processed in the previous step. Subsequently, since some values were NaN, I had to clean up the exoplanet data. The dataset reduced in size of 18.07%, dropping 1595 planets. Moreover, some useful information about the data can be found below.




```
['dtw_det_sap_flux', 'det_sap_flux_std', 'det_sap_flux_mean', 'det_sap_flux_min', 'pos_corr2_max', 'pos_corr1_max',
 'sap_bkg_max', 'det_sap_flux_max', 'dtw_sap_bkg', 'sap_flux_std', 'sap_bkg_mean', 'sap_bkg_std', 'sap_bkg_min', 'mom
_centr1_std', 'mom_centr2_std', 'pos_corr1_mean', 'mom_centr1_min', 'pos_corr2_mean', 'mom_centr1_mean', 'pos_corr1_m
in', 'pos_corr1_std', 'mom_centr1_max', 'pos_corr2_std', 'pd_sap_flux_std', 'mom_centr2_min', 'mom_centr2_max', 'pos_
corr2_min']
```

The next step was to choose the number of features and divide the dataset from those that are still candidate exoplanets from the already classified as confirmed or false positive. To perform feature selection, I have applied a Random Forest Classifier to each feature. The plot below shows the relevance of each feature.

Finally, I reduced the number of feature by 25%, eliminating some computational cost. The choice of 25% was determined due to the fact that I wanted eliminate some irrelevant feature, but at the same time I did not want to eliminate some features that could help for classification.

Since I have decided to choose multiple Machine learning algorithm, I have created different functions to help me simplify the process.

- ***train_classifier:*** Is a function that fits the training data to the chosen classifier.
- ***predict_labels:*** Is a function that returns the precision score for an already fitted classifier.
- ***predict:*** Is a function that returns the prediction values for an already fitted classifier.
- ***train_predict:*** Is a function that returns the prediction values of training and testing data for an already fitted classifier.
- **k_fold_train_predict:** Is a function that performs k fold on a specific classifier.

Subsequently, I initialized the different classifiers and performed a k-fold on each of them.

The DecisionTreeClassifier seemed to perform really well on training; However, it did not generalize very well. Logistic Regression and Naive Bayes did not seem to fit the data. Their respective training and testing score are represented in the table below.

Algorithm	Average K-fold Training Score	Average K-fold Testing Score	Training Score	Testing Score
Naive Bayes	0.3787	0.3791	0.3757	0.3849
Logistic Regression	0.4715	0.6095	0.5200	0.1176

Algorithm	Average K-fold Training Score	Average K-fold Testing Score	Training Score	Testing Score
Random Forest	0.9838	0.6171	0.9856	0.6323
SVM	0.9945	0.9645	0.9965	0.9648

From the following results, the SVC is the model that performed best on this data set, with a precision score for the test dataset equal to 0.9648. Even though SVC is negatively biased, preferring False Negative, I am okay with that bias. Indeed, this is a discovery project, therefore, I would prefer that the algorithm is negatively biased instead of being positively biased. Meaning that the planets the algorithm recognizes really are confirmed exoplanet.

If the algorithm classifies something as false positive, the status of the planet will remain still as Candidate.

Finally, I wanted to point out that there weren't any complications that occurred during the coding process of this section.

Refinement

I have tried to perform a GridSearch on the SVC classifier; however, it does seem that any improvement has been reported on the algorithm. As you can see from the confusion matrix nothing changes from the previous reported matrix (left picture) to the matrix after tuning (right picture).

The parameter I have tuned for the GridSearch, with their respective range, are: C [-2, 10], gamma [-5,5], and degree [1,6]. The Final model has C = 10, degree=10, and gamma = 0.0031622.

```
Confusion Matrix:
[[1418   16]
 [   5 2502]]

Tuned model has a training precision score of 0.9965.
Confusion Matrix:
[[192 304]
 [  7 811]]
Tuned model has a testing precision score of 0.9648.
```

Another thing to be noticed is that I have tried to perform normalization on the dataset; however, the SVC model did not work at all. This behavior is due to the fact that the data is been already pre-processed. Moreover, relevant features like DTW data points would loose their applicability on the overall model.

```
Training a SVC
Confusion Matrix:
[[1418   16]
 [   5 2502]]
Confusion Matrix:
[[192 304]
 [  7 811]]
Precision score for training set: 0.9965.
Precision score for test set: 0.9648.
```

RESULTS

Model Evaluation and Validation

The support Vector Classifier (SVC) works better than predicted, even though is a little bit negatively biased, as you can see from the confusion matrix. As a matter of fact, the model has been tested with various inputs to evaluate whether the model generalizes well to unseen data. Indeed, the average

precision score for the k-fold testing set is 0.9645 and the precision score of the entire dataset is 0.9648.

Further, I was pleasantly surprised to discover that the model prediction on the still candidate planets confirmed 144 exoplanets. I am not an astrophysicist, so there are no ways for me to validate this prediction. The only way would be scientist releasing results on the candidate planets and compare those results with the prediction from the model.

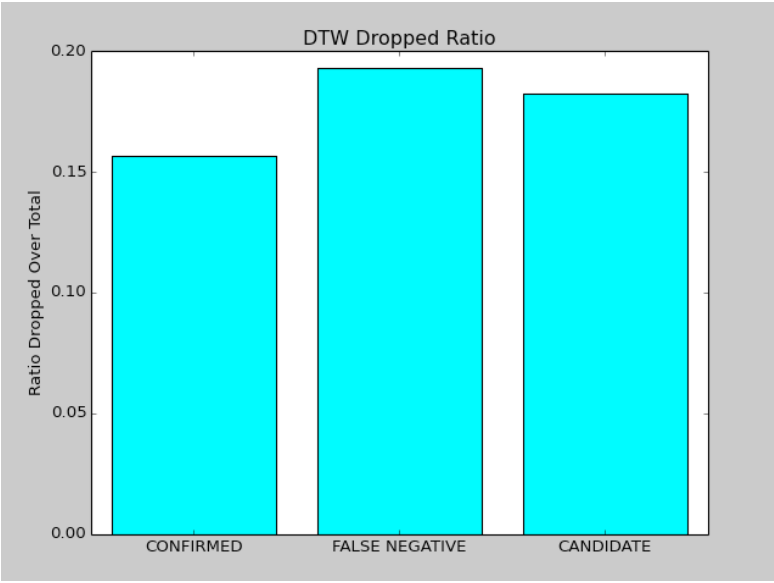
Justification

The final results found are stronger than the benchmark results reported earlier. Indeed, I predicted that the precision score would have been 60%; however, the score I got was 96.48%. I was not expecting such an amazing prediction. I believed that the data processing step contributed a lot to this results. The Dynamic Time Wrapping algorithm does an amazing job to compare the light curves, and without it I would have never reported this results. This machine learning approach would facilitate a lot the exoplanetary search for scientists.

CONCLUSION

Free-Form Visualization

One thing I want to point out is the fact that at the beginning I thought that the light curves the dynamic time wrapping algorithm could not process meant something. Therefore, I performed an analysis to see if there was any correlation between the number of dropped planets and their relative category (candidate, false positive, confirmed). As the plot below shows, there is not any correlation between the dropping rate and relative category of a planet. Indeed, the ratio of the number of each category dropped over the total number of planet for the respective category is between 0.15 and 0.20. This is consistent either between the 3 category, and with the total dropping rate of the dataset, which is 18.07%. I assume that my implementation of the DTW does not perform well with parallelized computing.



Reflection

This project started thanks to my passion for astrophysics. The first, and most difficult step was to collect the data of all the exoplanet, and processing it. Subsequently, the data had to be fitted to different models to check which one would perform best. Finally, with the best model, predict the faith of the still candidate exoplanets.

The entire project was difficult to put together, but at the same time rewarding thanks to the results obtained. I learned a lot about astrophysics, and the final results exceeded my expectations.

Improvement

One improvement that could be performed in the data processing section is to use a different algorithm to compare the two light curves. Indeed, while I was researching, I found this algorithm called Fast Time Series Evaluation (FTSE)[6], that could have given a faster and more accurate result compared with the Dynamic Time Warping algorithm. However, the research paper was not really clear on how the implement the algorithm, and I ended up using DTW.

Another improvement that could be performed is to refactor the code, setting the target labels to boolean values, and implement other statistical machine learning algorithms, such as perceptron, and k-nearest neighbors.

Another approach to this problem would have been using the photos taken from the Kepler telescope and apply a Deep Neural Networks to recognize planetary transit. However, this would have taken more time than required, and would have been out of scope.

CITATIONS

[1] *A Jupiter-mass companion to a solar-type star.* **Michel Mayor & Didier Queloz**. http://www.nature.com/nature/journal/v378/n6555/abs/378355a0.html Nature, 378, 355-359. November 23, 1995.

[2] *Kepler Mission/QuickGuide.* **Nasa Staff.** http://kepler.nasa.gov/Mission/QuickGuide/ July 25, 2016.

[3] *Kepler Archive Manual.* **Nasa.** http://archive.stsci.edu/kepler/manuals/archive_manual.pdf July 28, 2016.

[4] *Everything you know about Dynamic Time Warping is Wrong.* **Chotirat Ratanamahatana & Eamonn Keogh.** http://wearables.cc.gatech.edu/paper_of_week/DTW_myths.pdf University of California, Riverside. July 28, 2016.

[5] *Using Dynamic Time Warping to Find Patterns in Time Series.* **Donald J. Bemd & James Cliffor.** http://www.aaai.org/Papers/Workshops/1994/WS-94-03/WS94-03-031.pdf New York University, New York. July 28, 2016.

[6] *An Efficient and Accurate Method for Evaluating Time Series Similarity.* **Michael Morse & Jignesh M. Patel.** http://dbgroup.eecs.umich.edu/files/sigmod07timeseries.pdf University of Michigan Ann Arbor, Michigan. August 02, 2016.