Master's Thesis Project

VASCO: Developing AI-Crawlers for ML-Blink

UPPSALA UNIVERSITET

Department of Information Technology

Author:
Diego Castillo

Supervisor:
Kristiaan Pelckmans

June 20, 2019

Abstract

The “Vanishing and Appearing Sources during a Century of Observations” (VASCO) initiative aims at finding inexplicable effects among all-sky surveys. The VASCO project is a collaboration between astronomers and information technology researchers, and incorporates explicitly a component of citizen science. In an effort to efficiently mine the historical sky survey observations, an implementation of the ML-Blink algorithm – a machine learning algorithm which uses a data-driven approach to attempt to learn what features characterize interesting candidates – is proposed and evaluated as means to recommend interesting candidates from the historical sky survey observations. The proposed ML-Blink algorithm implementation consistently achieves an area under the curve in the 0.70 range and finds 2–4 artificial anomalies out of 7 in a dataset consisting 5005 observations from the USNO-B1.0 and Pan–STARRS1 datasets.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

The "Vanishing and Appearing Sources during a Century of Observations" (VASCO) initiative aims at finding inexplicable effects among all-sky surveys [21, 20]. The VASCO project is a collaboration between astronomers and information technology researchers, and incorporates explicitly a component of citizen science [1]. The study of differences among all-sky surveys could lead to interesting scientific findings, like new astrophysical phenomena or interesting targets for follow-up by the Search for Extraterrestrial Intelligence (SETI) observations.

Previous work done in [20], mostly based on manual comparisons, identified a vanishing point source by comparing the USNO-B1.0 sky survey catalog with the Sloan Digital Sky Survey (SDSS). The study of the night sky from multiple surveys to examine time variations is also described in [14], where a catalog with a total of 43,647,887 observations from USNO-B and SDSS was created and the issues encountered while doing so discussed. In both studies it is clear the enormous scale of existing sky surveys motivates the development of efficient computational tools, with an exciting role given to machine learning (ML) due to its capacity to deal with data-intensive processes.

The precise objective of this project is to implement and test an ML algorithm which uses a data-driven approach to attempt to learn what features characterize interesting candidates from the historical sky survey observations. The ML component is described as ML-Blink and it is based on methods of active and online semi-supervised learning. ML-Blink is named after the blink comparator; a 19th century viewing device invented by physicist Carl Pulfrich used by astronomers to discover differences between two images of the night sky [15].

[1]A more extensive description of VASCO can be found in [4].

Within the VASCO initiative, the ML-Blink algorithm will be used in order to identify anomalies that might be present in the historical sky survey observations. These surveys contain images from the same location in the night sky, but from distinct times. An arrangement of two images from the same location of the night sky from distinct datasets is defined as a mission. The goal of the ML-Blink algorithm is then to “crawl” these missions in order to recommend those that are more likely to contain an anomaly (i.e. a recommender system). In order to do so, the ML-Blink algorithm will learn what non–anomalies look like, select a set of missions to process, and recommend those that are most different from the non–anomalies it has learned. The recommended mission is referred to as a candidate.

1.2 Recommender Systems

A recommender system is a computer software which allows to provide product suggestions that serve a certain purpose to an entity. The entity to which such recommendation is provided is usually referred to as the user, while the product being recommended is commonly referred to as an item [6].

The usage of a recommender system is typically motivated by the existence of a set of predefined objectives to optimize and a possibly overwhelming number of items to choose from. A recommender system’s goal is to maximize the established set of objectives; a goal which can be accomplished by the use of a data–driven approach which attempts to learn existing dependencies among users and items.

As an example, consider an online bookstore that uses a recommender system to suggest books to its users. Such system might utilize explicit feedback such as a star rating system (e.g., 0–5), or implicit feedback like browsing for a title or buying a book to infer its users interests. The recommender system prediction based off the data aforementioned can then be used to increase profit and user engagement in the platform.

1.3 Outline of Thesis

The next chapter explains the ML-Blink algorithm from a theoretical point of view, along with topics which are required for the understanding of it. Chapter 3 discusses the methodology used to implement the ML-Blink algorithm and how it will be evaluated. Next, chapter 4 introduces the ML-Blink case study, where the ML-Blink algorithm will be used to aid astronomers in finding interesting observations for further analysis. In this chapter, the implementation of the user interface as well as the service to process and persist data are explained in detail. The evaluation results of the ML-Blink algorithm are discussed in chapter 4 too. Finally, chapter 5 is devoted to the conclusions of this work and suggestions for future work.

Chapter 2

Theory

2.1 Supervised Learning

Supervised learning is a function–fitting paradigm, where a model of the form $\mathbf{Y} = f(\mathrm{X}) + \epsilon$ is a fair premise. The goal of supervised learning is to learn f through a "teacher", which usually consists of a set of training observations of the form $\tau = (x_i, y_i), i = 1, ..., N$ where x_i is an input pattern and y_i is its corresponding label [10]. The model must also have the property that it can modify its input/output relationships in response to the differences between the predicted label and the true label of an observation. Once the learning process is completed, the expectation is that the outputs predicted by the learner will be similar to the true outputs such that the model is useful for all sets of inputs likely to be seen in practice [10].

2.2 Online Learning

Many common machine learning algorithms work by using batch learning; a paradigm where the entire training dataset is used to learn to recommend or predict an item [12]. In some occasions, doing so is in–feasible due to the size of the dataset, or because the model might need to actively adjust to new patterns in the data or user behavior; an scenario which is quite common in the field of recommender systems. In an online learning setting, data becomes available as a continuous stream, and the model uses these observations to update the current best recommendation or prediction at each time step [13].

2.3 Active Learning

Active learning is a paradigm in which a system attempts to learn the label of an observation by enabling users (or other sources) catalog unlabeled observations [17]. By doing this, the model aims to learn the relationship among the observations and their labels using as few observations as possible. Active learning seeks to overcome the labelling bottleneck, specially when there is a large amount of unlabeled data or when obtaining such labels is expensive [17]. Figure 2.1 shows an example active learning setup, where a user(s) is in charge of labeling data.

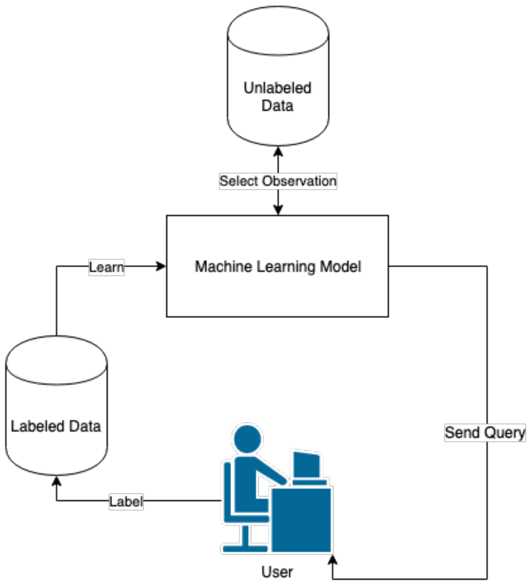


Figure 2.1: Diagram illustration of a possible active learning setup which relies in a user to label data.

2.4 The ML-Blink Algorithm

The main focus of this thesis is the study, implementation, and analysis of the ML-Blink algorithm. The ML-Blink algorithm was presented to me by my supervisor Kristiaan Pelckmans, and it was designed to recommend one item over another. The ML-Blink algorithm will determine how to recommend items based on a criteria it will learn using online and semi–supervised active learning techniques.

Formally, consider a pair of vectors $\mathbf{x}_i$ and $\mathbf{y}_j$ that represent the same information, but taken from different sources during distinct times. The goal is then to create a scoring function which is able to recommend pairs of items that are more likely to contain anomalies than those that do not. Since each pair of items represents essentially the same information, a pair of items is considered to contain an anomaly when something is present in one, but not in the other. Let the scoring function be defined as in equation 2.1, where $\mathbf{D}$ is the matrix that contains the weights that need to be learned by the model and it is initially $\mathbf{D} = 0$.

$$v = \mathbf{x}_i^T \mathbf{D} \mathbf{y}_j \tag{2.1}$$

The value v of a pair of items $\mathbf{x}_i$ and $\mathbf{y}_j$ is then defined as in equation 2.2.

$$v = \begin{bmatrix} x_{i,1} & x_{i,2} & \cdots & x_{i,n_x} \end{bmatrix} \begin{bmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,n_y} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,n_y} \\ \vdots & & & \\ d_{n_x,1} & d_{n_x,2} & \cdots & d_{n_x,n_y} \end{bmatrix} \begin{bmatrix} y_{j,1} \\ y_{j,2} \\ \vdots \\ y_{j,n_y} \end{bmatrix} \tag{2.2}$$

For the sake of readability, let us furthermore consider a pair of items $\mathbf{x}_i$ and $\mathbf{y}_j$ such that $n_x = 2$ and $n_y = 2$. The resulting formula is shown in equation 2.3.

$$\begin{aligned} v &= \begin{bmatrix} x_{i,1} & x_{i,2} \end{bmatrix} \begin{bmatrix} d_{1,1} & d_{1,2} \\ d_{2,1} & d_{2,2} \end{bmatrix} \begin{bmatrix} y_{j,1} \\ y_{j,2} \end{bmatrix} \\ &= \begin{bmatrix} x_{i,1}d_{1,1} + x_{i,2}d_{2,1} & x_{i,1}d_{1,2} + x_{i,2}d_{2,2} \end{bmatrix} \begin{bmatrix} y_{j,1} \\ y_{j,2} \end{bmatrix} \\ &= y_{j,1}x_{i,1}d_{1,1} + y_{j,2}x_{i,1}d_{1,2} + y_{j,1}x_{i,2}d_{2,1} + y_{j,2}x_{i,2}d_{2,2} \end{aligned} \tag{2.3}$$

Intuitively, the relevance of two features, say $y_{j,1}$ and $x_{i,1}$ is determined by the weight $d_{1,1}$. For example, by looking at the term

$$y_{j,1}x_{i,1}d_{1,1}$$

the weight assigned to $d_{1,1}$ will specify how important is the contribution of the $y_{j,1}$ and $x_{i,1}$ vectors' components according to what the ML-Blink algorithm has been taught. A large value of $d_{1,1}$ will therefore assign a high significance to $y_{j,1}$ and $x_{i,1}$, while a small value of it means $y_{j,1}$ and $x_{i,1}$ are not highly correlated with the objective value v. Finally, a value of $d_{1,1} = 0$ means the $y_{j,1}$ and $x_{i,1}$ features have no importance in terms of determining v.

As mentioned earlier, the matrix $\mathbf{D}$ will be learned using a combination of online and semi-supervised active learning techniques as defined in sections 2.2, 2.1, and 2.3 respectively, where users will catalog multiple pairs of items to determine whether these contain an anomaly or not. The ML-Blink algorithm will use this interaction to learn what non–anomalies look like and encode their features in the matrix $\mathbf{D}$. As a result, equation 2.1 will dictate "how much" like a non–anomaly does a pair of items "look like". That is, a resulting value of v that is large is unlikely to contain an anomaly, because it means the features of the pair of items at hand is highly correlated with what the ML-Blink algorithm has learned is a non–anomaly. In the other hand, a small value of v means that a particular pair of items is quite different from what the ML-Blink algorithm has learned, so it follows that the pair of items can possibly contain an anomaly.

How should the matrix $\mathbf{D}$ weights be learned? Given an unlabeled pool of observations, it is desirable to construct a query such that a pair of items with the minimum value of v is selected given what the ML-Blink algorithm currently knows in $\mathbf{D}$. That is, the ML-Blink algorithm will send a query to a user which contains a pair of items that the weights of the matrix $\mathbf{D}$ evaluates to contain an anomaly(s). The user will then determine whether the query contains an anomaly or not, and based on that the ML-Blink algorithm will then update the weights of the matrix $\mathbf{D}$ if necessary.

How should the weights of the matrix $\mathbf{D}$ be updated then? The query sent to the user contained what the ML-Blink algorithm evaluated to be an anomaly. As a result, if the query actually has an anomaly, there is nothing to change, since the matrix $\mathbf{D}$ weights correctly identified what corresponds to an anomaly(s). On the other hand, each time a certain pair $\left(\mathbf{x}_i, \mathbf{y}_j\right)$ was falsely recommended at iteration t because it led to a minimal value of $v_{t(i,j)} = \mathbf{x}_i^T \mathbf{D}_{t-1} \mathbf{y}_j$, $\mathbf{D}_{t-1}$ needs to be updated so that $(\mathbf{x}_i, \mathbf{y}_j)$ is not to be recommended in the near future. In other words, $\mathbf{D}_{t-1}$ needs to "learn" $\left(\mathbf{x}_i, \mathbf{y}_j\right)$ as normal. We do this by implementing one gradient step:

$$\mathbf{D}_t = \mathbf{D}_{t-1} + \mathbf{x}_i \mathbf{y}_j^T \tag{2.4}$$

with $\mathbf{x}_i \mathbf{y}_j^T$ the gradient of the evaluation $\mathbf{x}_i^T \mathbf{D}_{t-1} \mathbf{y}_j$ as

$$^1\nabla(\mathbf{x}_i^T \mathbf{D}_{t-1} \mathbf{y}_j) = \nabla(\text{trace}(\mathbf{D}_{t-1} \mathbf{y}_j \mathbf{x}_i^T)) = \mathbf{y}_j \mathbf{x}_i^T$$

where $\nabla(.)$ denotes the gradient with respect to $\mathbf{D}_{t-1}$. In this way, the next iteration will score the case $\left(\mathbf{x}_i, \mathbf{y}_j\right)$ higher. That is

$$\mathbf{x}_i^T \mathbf{D}_t \mathbf{y}_j = \mathbf{x}_i^T (\mathbf{D}_{t-1} + \mathbf{x}_i \mathbf{y}_j^T) \mathbf{y}_j = \mathbf{x}_i^T \mathbf{D}_{t-1} \mathbf{y}_j + 1$$

assuming that $||\mathbf{x}_i|| = ||\mathbf{y}_j|| = 1$. Hence, the case $\left(\mathbf{x}_i, \mathbf{y}_j\right)$ will not be low (and thus being recommended) in the next iteration. In other words, the algorithm has "learned" case $\left(\mathbf{x}_i, \mathbf{y}_j\right)$ as desired.

Equation 2.1 can also be implicitly represented. To start off, let us first re–write the update rule defined in equation 2.4. By construction, the matrix $\mathbf{D}$ can also be represented as

$$\mathbf{D} = \sum_{i \in A} \mathbf{v}_i \mathbf{w}_i^T$$

where $\mathbf{v}_i$ and $\mathbf{w}_i$ represent a pair of items that were learned by the model, and A is the set of all vectors that have been learned by the model, referred to as the

[1] The full proof of the ML-Blink algorithm and its mathematical properties will be addressed in a subsequent paper. This report focuses in the implementation and evaluation of the algorithm only.

active set. Hence, if the algorithm needs to compute the value v of a particular pair of items consisting of the vectors $\mathbf{x}_i$ and $\mathbf{y}_j$, equation 2.1 can be re–written as:

$$\begin{aligned} v &= \mathbf{x}_i^T \mathbf{D} \mathbf{y}_j \\ &= \mathbf{x}_i^T (\sum_{i \in A} \mathbf{v}_i \mathbf{w}_i^T) \ \mathbf{y}_j \\ &= (x_i^T \cdot v_1)(w_1^T \cdot y_j) + (x_i^T \cdot v_2)(w_2^T \cdot y_j) + \cdots + (x_i^T \cdot v_n)(w_n^T \cdot y_j) \end{aligned} \tag{2.5}$$

The advantages of using an implicit representation to describe what the ML-Blink algorithm has learned in the matrix $\mathbf{D}$ (or the active set) and compute the objective value of a pair of items $\mathbf{x}_i$ and $\mathbf{y}_j$ will be further studied in sections 3.3 and 3.4.

2.5 Normalization

Normalization refers to the process of accommodating the values of observations so that their unit of measurement does not affect their contribution when compared to one another. Normalization essentially drops the unit of measurement from the observations, and as a result, it allows to examine observations that come from distinct places in a notionally common scale [18].

As pointed out in section 2.4, the pair of items $\mathbf{x}_i$ and $\mathbf{y}_i$ are assumed to have been acquired from distinct sources, which means these sources might have used different devices and/or software processing techniques to collect the data. As a result, normalization is required in order to use a common "scale" between these two observations to avoid one unit of measurement dominating the other due to differences in the data acquisition step.

The ML-Blink algorithm uses the L2–norm as defined in equation 2.6 to normalize the input vectors. The normalization is performed by dividing each component of a vector by the vector's L2–norm. The resulting vectors have the characteristic that $||\mathbf{x}_i|| = ||\mathbf{y}_j|| = 1$, as required by equation 2.4.

$$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2} \tag{2.6}$$

2.6 Dimensionality Reduction

In machine learning and statistics, it is common to refer to the number of features that make up an observation as its dimensionality [18]. As the number of features that describe an observation increases, it is likely that one will encounter the so called "curse of dimensionality". The curse of dimensionality is the manifestation of all phenomena that occurs when dealing with high–dimensional data, and that have most often unfortunate consequences on the behavior and performances of learning algorithms [19].

Dimensionality reduction refers to the process of reducing the number of features that describe an observation. Dimensionality reduction can be performed by either using feature selection (selecting a subset of the original features) or feature extraction (deriving new features from the original features). Dimensionality reduction can help avoid the curse of dimensionality, eliminate unsuitable features, reduce noise, and reduce the amount of time and memory required by machine learning or statistical algorithms to execute [18].

The ML-Blink algorithm implementation written for this report was evaluated using two well known dimensionality reduction techniques: projections and pooling.

2.6.1 Projections

Projections use a linear inner product to project a pair of items $(\mathbf{x}_i, \mathbf{y}_j)$ to a lower dimension. Projections were chosen as one of the dimensionality reduction techniques to implement due to its simplicity and computational performance.

To better illustrate this method, consider a vector $\mathbf{x}_i$ where $n_x = 3$ and a matrix $\mathbf{P}$ of size 3×9 as in equation 2.7.

$$\begin{aligned} \mathbf{p} &= \mathbf{x}_i \cdot \mathbf{P} \\ &= \mathbf{x}_i \cdot \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} x_{i,1} + x_{i,2} + x_{i,3} & x_{i,4} + x_{i,5} + x_{i,6} & x_{i,7} + x_{i,8} + x_{i,9} \end{bmatrix} \end{aligned} \tag{2.7}$$

As shown in equation 2.7, the resulting vector $\mathbf{p}$ has only 3 dimensions. Each of these dimensions were created by adding a vector component and the next two consecutive components next to it until all elements in the initial vector $\mathbf{x}_i$ were processed. The linear inner product dimensionality reduction technique is essentially a form of feature extraction, as new features of the vector $\mathbf{x}_i$ were derived from its original components.

2.6.2 Pooling

The objective of pooling is to change a collective feature representation into a new, more usable one that maintains important information while eliminating irrelevant detail [5]. The pooling operation is typically a sum, average, or a max operation performed within a kernel.

The pooling implementation made for this report uses average pooling with non–overlapping kernels and replaces out–of–boundary pixel values intensities with zero. Pooling was selected as an alternative dimensionality reduction technique to evaluate whether the spatial structure of pooling neighborhoods (within the kernel) could benefit the representation of the input vector, and thus help the model to better encode features in the weight matrix **D**.

Figure 2.2 illustrates how average pooling with a kernel of size 2×2 works. Similar to projections, pooling is also a feature extraction dimensionality reduction technique.

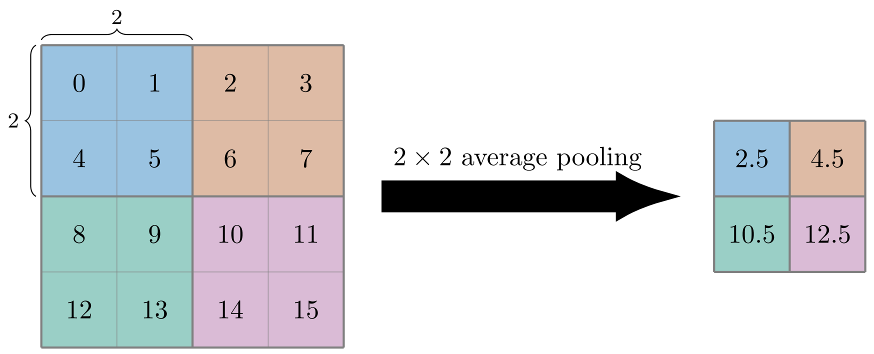


Figure 2.2: Example of how non–overlapping average pooling with a kernel of size 2×2 is performed.

Chapter 3

Methodology

3.1 Datasets

The ML-Blink algorithm was designed with the goal of aiding astronomers in the VASCO initiative to find interesting observations for further analysis. A subset of the USNO-B1.0 and Pan–STARRS1 datasets gathered by Johan Soodla during his master thesis project (referred to as β–pack in his written report) was used to implement and test the algorithm. Within the requirements elicited when the datasets' subsets were created, it was specified that the center of the image must contain a star, galaxy or artifact for at least 95% of the cases.

USNO-B1.0 is an all-sky catalog composed from multiple sky surveys during the interval from 1949 to 2002 [11] that indicates positions, proper motions, star/galaxy estimators and other astronomical features for 1,042,618,261 objects derived from 3,643,201,733 distinct observations [3]. Pan–STARRS is a system for wide-field astronomical imaging developed and operated by the Institute for Astronomy at the University of Hawaii. Pan–STARRS1 is the first part of Pan–STARRS to be completed and is the basis for both Data Releases 1 and 2 (DR1 and DR2). Pan–STARRS1 DR1 was released on December 19, 2016 [1].

The subset consists of a total of 1001 unique cases in each dataset, each described across different color–bands. Each of these color–bands represents a certain wavelength on the color spectrum. The USNO-B1.0 subset used a total of five bands (`blue1`, `blue2`, `red1`, `red2`, and `ir`), while Pan–STARRS1 subset used a total of three bands (`g`, `r`, and `z`). Consequently, the β–pack contains a total of 5005 images. Table 3.1 shows how each of the datasets' color–bands are related to one another in USNO-B1.0 and Pan–STARRS1 respectively. Lastly, the subsets' images were all in gray–scale format for all dataset bands.

USNO-B1.0 Band	Pan–STARRS1 Band
`blue1`	`g`
`blue2`	`g`
`red1`	`r`
`red2`	`r`
`ir`	`z`

Table 3.1: Mappings which specify how each color–band in USNO-B1.0 is related to a color–band in Pan–STARRS1 or vice–versa.

3.2 Crawling Candidates

Algorithm 1 shows the basic building block of what the ML-Blink algorithm does, where the time steps represent when the algorithm is called to generate a new candidate. The value v of a mission defines how similar it is to what the ML-Blink algorithm has learned in the matrix $\mathbf{D}$ (or active set). Since the ML-Blink algorithm is designed to learn what non–anomalies are, retrieving the mission with the minimum value v of all that were crawled represents the one that is most dissimilar to what the ML-Blink algorithm knows at that particular time step.

```
generate_candidate:
    for $t = 0, 1, 2, ...$ do
        Select a set of missions to crawl
        for mission in missions do
            Compute mission's $v$ value
        end
        Select mission with min($v$) as candidate
        return candidate
    end
end
```

Algorithm 1: Pseudo–code for the basic building block of the ML-Blink algorithm.

After a set of missions has been selected, computing their corresponding v value is what will differ depending on how the weights in the matrix $\mathbf{D}$ learned by the algorithm are represented. It is also important to note that the very first time the matrix $\mathbf{D}$ is retrieved, all of its weights are equal to 0 (i.e. it has not learned anything yet), and therefore any mission given to it will return a v value equal to 0. If multiple missions are tied for the minimum value v, the ML-Blink algorithm will randomly select a mission within those in the tie as the candidate.

3.3 Explicit Representation

As shown in equation 2.1, the explicit representation of the learned weights simply stores these weights in a matrix **D**. The matrix **D** is then used to compute the value v of a mission, as well as updating it to learn new information.

Algorithm 2 shows pseudo–code which given a mission setup will return the v value of such mission.

```
1 compute_v_explicit $i$, $j$:
2     Let $i$ be an image key, and $j$ be an image band
3     Retrieve image $\mathbf{x}_{i,j}$ according to $i$, $j$ from USNO-B1.0
4     Retrieve image $\mathbf{y}_{i,j}$ according to $i$, $j$ from Pan–STARRS1
5     Retrieve weights matrix $\mathbf{D}$
6     Compute $v = \mathbf{x}_{i,j}^T \mathbf{D} \mathbf{y}_{i,j}$
7     return $v$
8 end
```

Algorithm 2: Pseudo–code for computing the value v for a mission setup using the explicit definition of the matrix **D**.

Algorithm 3 shows how the weights of the matrix **D** are updated in order to learn new information.

```
1 update_d_explicit $i$, $j$:
2     Let $i$ be an image key, and $j$ be an image band
3     Retrieve image $\mathbf{x}_{i,j}$ according to $i$, $j$ from USNO-B1.0
4     Retrieve image $\mathbf{y}_{i,j}$ according to $i$, $j$ from Pan–STARRS1
5     $\mathbf{D} \leftarrow \mathbf{D} + \mathbf{x}_{i,j} \mathbf{y}_{i,j}^T$
6 end
```

Algorithm 3: Pseudo–code for updating the explicit representation of the matrix **D**.

The weights learned by the ML-Blink algorithm must be persisted in order for multiple crawlers to be able to read and write from the matrix **D** at different time steps. Even though the explicit representation using the matrix **D** provides a simple way to describe what the algorithm has learned and how it can be used to learn new information, storing, retrieving, and updating such weights might represent a performance issue depending on the size of the vectors in **x** and **y**.

3.4 Implicit Representation

Since the weight matrix **D** must provide an interface to easily access, update, and save its values, it is therefore desirable to create a data structure that can aid in creating such design. To do so, equation 2.1 can be implicitly represented as shown in equation 2.5.

Algorithm 4 shows the updated pseudo–code of the method `compute_v_explicit` renamed to `compute_v_implicit` used to calculate the value v of a mission.

```
1 compute_v_implicit i, j:
2     Let $i$ be an image key, and $j$ be an image band
3     Retrieve image $\mathbf{x}_{i,j}$ according to $i$, $j$ from USNO-B1.0
4     Retrieve image $\mathbf{y}_{i,j}$ according to $i$, $j$ from Pan–STARRS1
5     Retrieve all members of the active set $A$
6     Compute $v = \mathbf{x}_{i,j}^T ( \sum_{i \in A} \mathbf{v}_i \mathbf{w}_i^T ) \mathbf{y}_{i,j}$
7     return $v$
8 end
```

Algorithm 4: Pseudo–code for computing the value v for a mission setup using the implicit definition of the matrix $\mathbf{D}$.

Finally, when using the implicit definition of the weight matrix $\mathbf{D}$, in order to learn new information, all that is needed is to insert a mission to the active set A. Algorithm 5 shows the updated `update_d_explicit` method renamed to `update_d_implicit` used to updated the active set A when new information needs to be learned by the model.

```
1 update_d_implicit i, j:
2     Let $i$ be an image key, and $j$ be an image band
3     Retrieve image $\mathbf{x}_{i,j}$ according to $i$, $j$ from USNO-B1.0
4     Retrieve image $\mathbf{y}_{i,j}$ according to $i$, $j$ from Pan–STARRS1
5     $A \leftarrow A + \mathbf{x}_{i,j} \mathbf{y}_{i,j}^T$
6 end
```

Algorithm 5: Pseudo–code for updating the implicit representation of the matrix $\mathbf{D}$.

3.5 Image Retrieval

The ML-Blink algorithm retrieves images from the β–pack dataset and performs a few operations in order to pre–process the images to later on evaluate them. In addition to the aforementioned dimensionality reduction through projections (or average pooling) and normalization using the L2–norm, the images are also binarized.

Binarization is a process in which an input signal is transformed such that the resulting output consists of only two values. The ML-Blink algorithm uses binarization with a fixed threshold (one for each source) as a pre–processing technique when retrieving missions. Algorithm 6 shows pseudo–code which describes how a mission's vector is retrieved using binarization, dimensionality reduction (as described in section 2.6), and normalization (section 2.5). This process is applicable to both **x** and **y**, and it is described in terms of **z** for illustrative purposes only. Note line number 5 is replaced by average pooling as a dimensionality reduction technique when appropriate.

```
1 retrieve_vector $i$, $j$, $n_{projections}$:
2     Let $i$ be an image key, and $j$ be an image band
3     Retrieve image $\mathbf{z}_{i,j}$ according to $i$, $j$ from $\mathbf{z}$ source
4     $\mathbf{bw} \leftarrow \mathbf{z}_{i,j}$ binarized with fixed threshold $t_z$
5     $\mathbf{zs} \leftarrow \mathbf{bw} \cdot \mathbf{P}$ where $\mathbf{P}$ size is $n_{projections} \times n_z$ (or use average pooling)
6     return zs normalized using the L2–norm
7 end
```

Algorithm 6: Pseudo–code to retrieve a vector given an image key i, an image band j, and the desired number of projections to use for dimensionality reduction.

3.6 Parallelism

The ML-Blink algorithm also takes advantages of parallel computing in order to allow for faster processing of potential candidates. Algorithm 1 is slightly modified to simply split up the potential candidates processing among the available number of processors. Therefore, instead of a single for–loop processing all selected missions, each available processor computes the v value for each potential candidate it was assigned to using the implicit definition from section 3.4 in parallel. The result of each process is then "reduced" to correctly select the next candidate with $\min(v)$. Algorithm 7 shows the update pseudo–code to crawl for candidates using parallel processing.

```
generate_candidate_parallel:
    for $t = 0, 1, 2, ...$ do
        Select a set of missions to crawl
        Split missions among the number of available processors
        Process each missions' "chunk" in parallel
        Reduce each parallel job results
        Select mission with $\min(v)$ as candidate
        return candidate
    end
end
```

Algorithm 7: Slightly modified pseudo–code of the basic building block of the ML-Blink algorithm to allow for parallel processing.

3.7 Evaluation

As mentioned earlier, the ML-Blink algorithm is a recommender system. That being said, it can also be described as a binary classifier, since for any observation it assigns a label to it as either an anomaly or a non–anomaly. Hence, the receiver operating characteristic (ROC) curve is a suitable technique to study and understand the capability of an ML-Blink model as its discrimination threshold is changed.

Before diving into the ROC curve, let us first define a confusion matrix in terms of the ML-Blink algorithm. A confusion matrix is simply a table which allows to visualize the performance of a model by showing what label the model assigns to the observations in comparison to the real labels of these observations. A confusion matrix for the ML-Blink algorithm is shown in table 3.2.

		Actual	
		Anomaly	Non–anomaly
Predicted	Anomaly	TP	FP
	Non–anomaly	FN	TN

Table 3.2: A confusion matrix table for the ML-Blink algorithm. The model predictions are categorized as true positive (TP), false positive (FP), false negative (FP), and true negative (TN).

The ROC curve is defined as the a plot of the true positive rate (TPR) against the false positive rate (FPR). The TPR is defined as in equation 3.1, where TP corresponds to the total number of true positives (top–left quadrant of the confusion matrix in table 3.2), and P (positives) is the total number of real anomalies in the dataset. The FPR formula is shown in equation 3.2, where FP represents the false positive (top–right quadrant in table 3.2), and N (negatives) is the real number of non–anomalies in the dataset.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} \tag{3.1}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} \tag{3.2}$$

The ROC curve plot requires a model discrimination threshold to be changed, in the ML-Blink algorithm the threshold is going to be the range of v values computed for the missions that were selected to be crawled. The ROC curve will plot the TPR against the FPR "sweeping" the entire range of v values in the selected missions from $\min(v)$ up to $\max(v)$, and for each v value computing the TPR and FPR.

The ROC curve is a useful evaluation technique as it allows to easily visualize the trade-off between sensitivity (TPR) and specificity (FPR) [10]. The closer the curve being plotted is to the 45–degree diagonal, the less accurate the test. It is also common to show the Area Under Curve (AUC) when plotting the ROC curve. The AUC of a classifier is equivalent to the probability that the classifier will rank a randomly chosen positive observation higher than a randomly chosen negative observation [8] (assuming normalize units are being used). For instance, the 45–degree diagonal in the ROC space previously mentioned, has an AUC of 0.5, equivalent to a random predictor for a binary classification problem.

Chapter 4

Case Study

4.1 Introduction

This section describes the overall idea of the case study and how it was used to examine the ML-Blink algorithm. The case study consists of a user interface (UI) that allows users to match two images of the same location in the night sky from distinct datasets, and an Application Programming Interface (API) that processes data created by the UI. Figure 4.1 shows the UI where users can create a matching of two images.

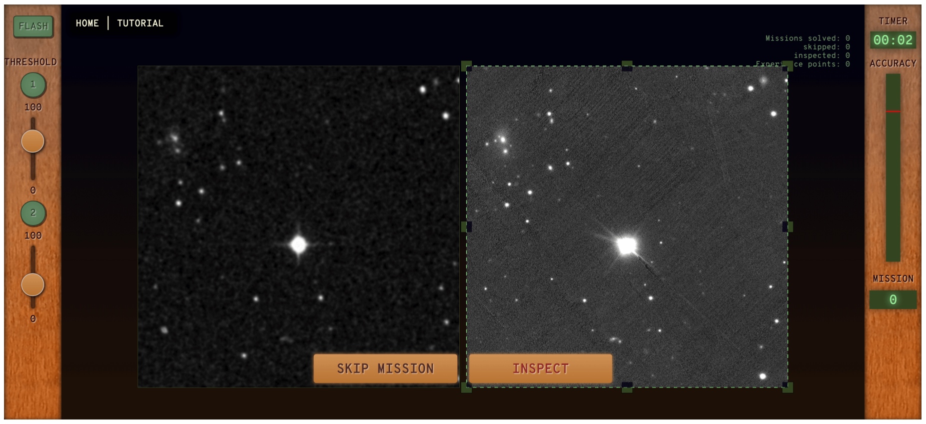


Figure 4.1: UI used to conduct the case study. Users are presented a mission where the goal is to create a matching. A matching is defined as the placing of the right side image on top of the other one, and obtaining an accuracy based on how good such matching is (i.e. how well the objects of one image align with the objects of the other image). The accuracy achieved by a matching is shown to the user in the right side of the UI.

4.1.1 Matching Accuracy

The UI uses an accuracy threshold shown to the user in the right side of the screen to determine whether a matching of two images is good or not. If a user is able to achieve an accuracy greater or equal than that of the accuracy threshold, the mission is then considered to be successfully completed and the two images are unlikely to contain an anomaly. If the accuracy achieved by the user is less than that of the accuracy threshold, then the images that represent the mission are considered to be potential anomalies.

The accuracy threshold used to conduct the case study was fixed at 80%. To fix it, a few missions from the datasets were randomly selected and manually altered to represent anomalies of interest, such as replacing a bright section of one of the images with its background, while the other image remained unaltered. Figure 4.2 shows an example of an anomaly that was manually created, where the center object of the image from the Pan–STARRS1 dataset has been replaced by its background. When missions like this were presented in the UI, it was found to be difficult, and in some occasions impossible, to get an accuracy greater than 80%, and thus why it was selected as the accuracy threshold.

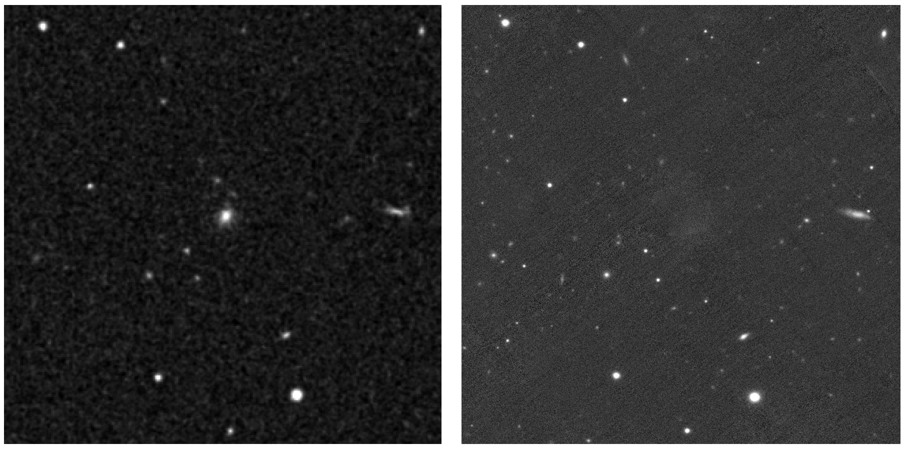

(a) USNO-B1.0 (b) Pan–STARRS1

Figure 4.2: Figures 4.2a and 4.2b show pictures manually altered taken from the same location in the night sky from USNO-B1.0 and Pan–STARRS1 respectively. An object close to the center of the Pan–STARRS1 image has been replaced by the image's background, while the USNO-B1.0 image is unaltered.

4.2 Architecture

The following section motivates and outlines the architecture choices made in order to create the tools required to conduct the case study. The software required to perform the case study is split into two independent applications. The first application focuses in user interaction and experience, referred to as the ML-Blink UI, and the second one is in charge of dealing with domain logic and providing the required resources to the ML-Blink UI, referred to as the ML-Blink API. Through out the report, these two applications are also described as the client and server respectively.

While the two applications could have been developed as a monolith, having two decoupled applications allows to clearly distinguish between client responsibilities and domain logic. This approach also allows to support multiple types of clients such as WEB, mobile, and desktop – all using the ML-Blink API to handle domain logic, as well as simpler delegation in the sense that a person(s) which only needs to work on the client will not need to setup the server dependencies; this person(s) can simply connect the local ML-Blink UI to a development instance of the ML-Blink API running in the cloud.

On the other hand, splitting the project into two separate applications comes with a higher operational overhead. For instance, deploying a new feature could require rolling out a new version of both the client and the server, while taking into account that and older version of the client might be cached in the user's browser. Figure 4.3 shows a summary of how the ML-Blink UI, ML-Blink API, and a user depicted by a computer interact with each other.

4.2.1 Server Architecture

The ML-Blink API runs in its own server using Ubuntu 16.04 and it is written in Python 3.5.2. It closely follows the Representational State Transfer (REST) software architecture, where applications provide consistent interface semantics (usually create, read, update, and delete for each resource) rather than arbitrary interfaces. Additionally, the REST interactions are "stateless" in the sense that a response provided by an application is dependent on the parameters it receives, not in the current state of it [2].

To aid in the design of a REST application, the ML-Blink API is build on top of the Flask microframework [9], which provides useful methods and abstractions to create web services that follow REST conventions with minimal effort. The ML-Blink API is served using the Apache HTTP Server in conjunction with `mod_wsgi`; an Apache HTTP Server module that enables Apache to serve Flask applications.

The majority of the data produced by the ML-Blink UI usually consists of a structure of nested objects, as a result MongoDB was selected for persisting data

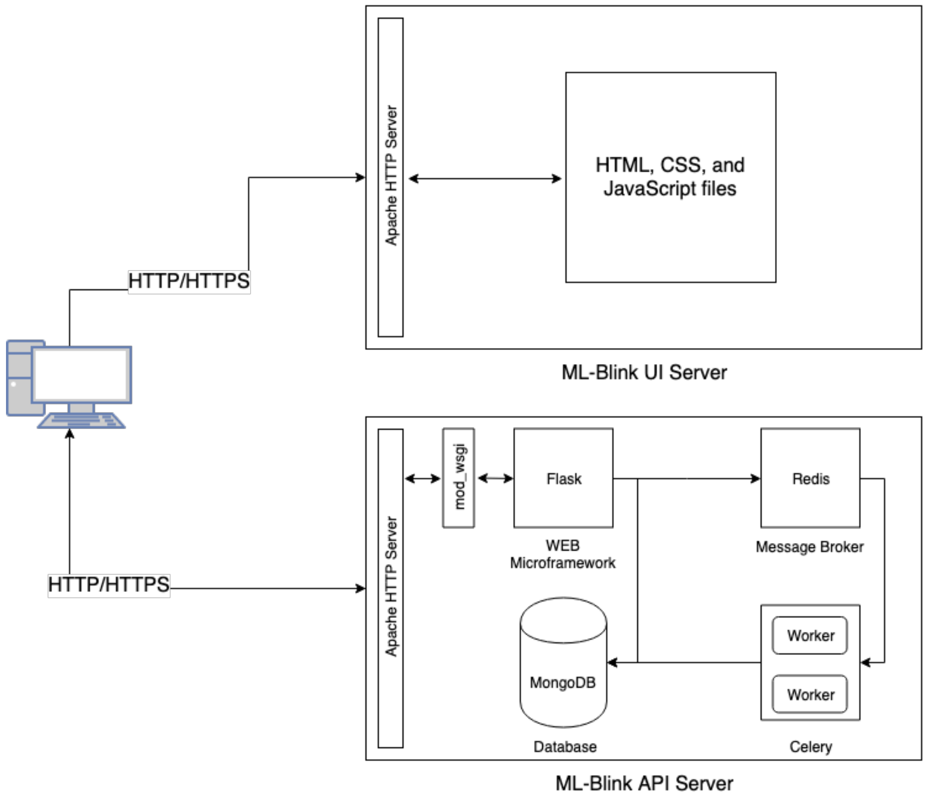


Figure 4.3: Summary of how the ML-Blink UI and the ML-Blink API interact with each other when used by a client depicted by a computer in this scenario.

in the ML-Blink API. Additionally, since the UI uses Asynchronous JavaScript and XML (AJAX) requests to communicate with the server, the ML-Blink API defines Cross–Origin Resource Sharing (CORS) rules to restrict that only the domain in which the ML-Blink UI is being served from can access the server's resources; such as when retrieving a mission setup, or creating a mission along with its achieved accuracy and other attributes.

Finally, the ML-Blink API uses the Celery distributed task queue [7] with Redis [16] as a message broker in order to execute time–consuming tasks in the background without blocking the API. The Celery distributed task queue was also selected as it comes with an API that allows to execute background tasks concurrently on multiple processors.

4.2.2 Client Architecture

The ML-Blink UI is entirely written in JavaScript. Since the UI allows for quite complex user interaction, the `Vue.js` framework [22] – an open-source JavaScript framework for building user interfaces – is used as the view layer to develop the user interface. Additionally, the application uses the Vue Command Line Interface (Vue CLI) to make local development, scaffolding, and deployment easier.

The ML-Blink UI communicates with the ML-Blink API through AJAX calls that closely follow the REST standard, as pointed out in section 4.2.1. These AJAX calls request the required data and resources to show to the user in the UI, as well as send data created by the user while using the UI to the ML-Blink API to be persisted in the database.

Lastly, similar to the ML-Blink API, the ML-Blink UI is hosted in its own server running in Ubuntu 16.04, and it uses the Apache HTTP server to serve requested resources as well.

4.3 Implementation

This section describes the implementation of the ML-Blink UI and ML-Blink API in detail, the algorithms these use, and the rationale behind these decisions.

4.3.1 ML-Blink UI

As described in section 4.1, the ML-Blink UI allows users to match images of distinct datasets of the same location in the night sky. The goal is to provide intuitive feedback about the quality of the current matching to the user, where a good matching receives a better score than a bad one.

The different steps taken to process a matching can be summarized as follows: computing the region of interest (ROI), smoothing, binarization, object detection, object size normalization, and computing accuracy. These steps in conjunction define the algorithm used to compute the quality of a matching, and it is referred to as the matching algorithm.

The matching algorithm runs in the main thread (also known as the UI thread) in the client hardware that renders the ML-Blink UI; which might be a fast desktop, or a slow mobile device. Therefore, it is important that the matching algorithm runs fast so that the UI thread does not "freeze" and the user experience degrade.

Every time the user changes the position of the Pan–STARRS1 image, a "debounced" function is created. A debounced function allows to wait a specified number of milliseconds before invoking another function. This allows the ML-Blink UI to avoid computing the aforementioned algorithm every time the user changes the position of the Pan–STARRS1 image, and instead wait until 250 ms have elapsed – and no further position or transformation has occurred – to execute the matching algorithm.

In order to better illustrate how the matching algorithm works, the following subsections will use figure 4.4 as an example. It is worth nothing that while all of these steps are performed when determining the accuracy of a matching, only step 4.3.1.1 is actually drawn behind the scenes in the UI. The arrays which represent the rest of these steps are only stored in memory, since there is no need to draw them on the UI. Additionally, except for the accuracy computation, all of these steps are only performed once for the USNO-B1.0 image when it is loaded, since it is static and there is no need to re–compute the same thing again.

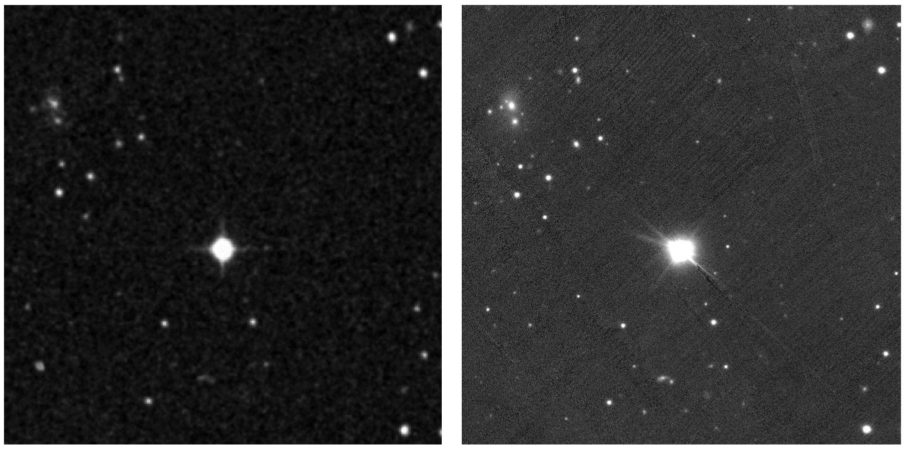

(a) USNO-B1.0 (b) Pan–STARRS1

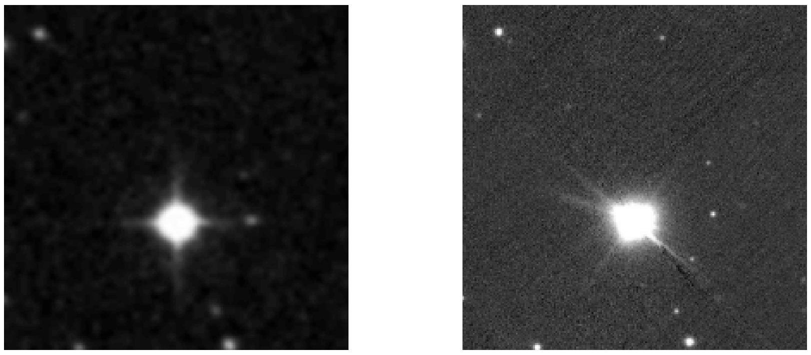
Figure 4.4: Figures 4.4a and 4.4b show pictures taken from the same location in the night sky from USNO-B1.0 and Pan–STARRS1 respectively.

As the user moves or transforms the Pan–STARRS1 image in the UI, the following debounced steps are executed:

4.3.1.1 ROI

The ROI is defined as a square of 200 pixels placed on top of the center of the USNO-B1.0 image. The resulting ROI of the USNO-B1.0 image is shown in figure 4.5a. The ROI of the Pan–STARRS1 image is defined from the exact location, which means that in order to match the two images, the user must place the Pan–STARRS1 on top of the USNO-B1.0 image, otherwise the resulting ROI will be completely dark (i.e. no pixels from the Pan–STARRS1 image were found in the same location as the USNO-B1.0 ROI). Figure 4.5b shows the resulting ROI of the Pan–STARRS1 image as if it was positioned exactly on top of the USNO-B1.0 image in the UI.

The choice to use a ROI to measure the quality of a matching was made based on the requirements established for the construction of the datasets used in the case–study (see section 3.1), as well as an approach to aid the design of an algorithm that evaluated faster.



(a) USNO-B1.0 ROI (b) Pan–STARRS1 ROI

Figure 4.5: Resulting ROI from the figures shown in figure 4.4. The ROI is computed as if the Pan–STARRS1 image was placed exactly on top of the USNO-B1.0 image in the UI.

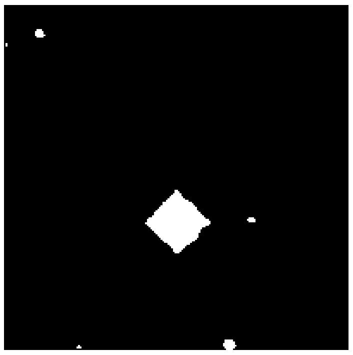
4.3.1.2 Smoothing

Once the ROI has been computed for both images, the pixel value intensities of the RGBA channels of each image in their respective ROI are retrieved. Since the images are in gray–scale format, all RGB pixel value intensities at the same location are equal, which means that the remaining computations can be performed using a single channel. As a result, only the pixel value intensities of the R channel are smoothed using a mean filter, which facilitates object detection in the upcoming steps.

The mean filter uses a 3×3 kernel which scans the entire ROI of each image and replaces the center pixel where the kernel is located at with the mean pixel value intensity of the elements within the kernel. The mean pixel value intensity of the entire ROI is used for out–of–boundary pixels when placing the kernel in the border pixels of the ROI.

4.3.1.3 Binarization

The next step is to binarize the smoothed pixel value intensities of each image's ROI. The binarization process simply replaces all pixel value intensities larger than a specified threshold with 255, while pixel value intensities less or equal to the threshold are set to 0. Figure 4.6 shows the resulting ROI after both images have been binarized. The matching algorithm uses a fixed threshold of 80 for USNO-B1.0 and 110 for Pan–STARRS1.


(a) Binarized USNO-B1.0

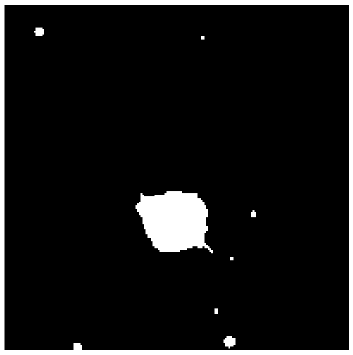

(b) Binarized Pan–STARRS1

Figure 4.6: Binarization result using a threshold value of 80 for figure 4.6a and 110 for figure 4.6b.

While a binarization technique that automatically finds what threshold to use could have been implemented, a fixed threshold is computationally faster and simpler. However, due to the threshold value not being optimal for all observations, it is also possible that a fixed threshold might mistakenly label a dark section as background, even if it is a “not so bright” object.

4.3.1.4 Object Detection

The binarized ROI are then fed through an object detection algorithm. The object detection algorithm uses connected components to label all pixels connected to an object before proceeding with the next pixel. Since the input image is binarized, any object to be identified must have a pixel value intensity of 255.

The algorithm works by first filtering the entire array of pixels by those which are cataloged as an object. Once these have been found, a kernel of size 3×3 placed on top of such pixels is used to find its neighbors. The neighbors are then filtered by those that are objects and have not been labeled yet. Following that, each of these neighbors are labeled using the label of that connected component, and kept track of in a queue data structure for follow up labelling. Only once all these neighbors have been labeled (i.e. the queue is empty), the algorithm continues with the next pixel value intensity cataloged as an object.

Figure 4.7 shows the connected components found by the object detection algorithm in each of the images’ ROI.

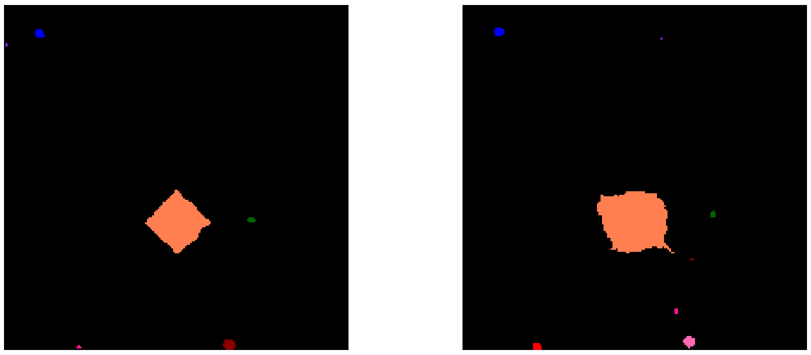

(a) Objects in USNO-B1.0 (b) Objects in Pan–STARRS1

Figure 4.7: Object detection result for both USNO-B1.0 (figure 4.7a) and Pan–STARRS1 (figure 4.7b) images' ROI. Note each of the objects label has been colorized for illustrative proposes only.

4.3.1.5 Object Size Normalization

The objects detected within each image's ROI are then normalized by replacing each of them by an equal sized square object. USNO-B1.0 objects are replaced by squares of size 9, while objects of size 13 for Pan–STARRS1 were found to result in a more intuitive score in the UI.

The object normalization algorithm takes the labeled objects as in figure 4.7, and for each of them, computes its center of mass. The center of mass is calculated for the x–axis and y–axis in a similar way. The x–axis center of mass is computed by adding the row of each pixel defined within an object and dividing the resulting total by the number of pixels the object has. The same procedure is performed for the y–axis center of mass, but adding the column of each pixel defined within the object instead. Once the center of mass has been computed in both axis, a squared object of a specified size is placed on top of the object's center of mass.

Figure 4.8 shows the resulting square objects after both the USNO-B1.0 and Pan–STARRS1 images' ROI objects size have been normalized.

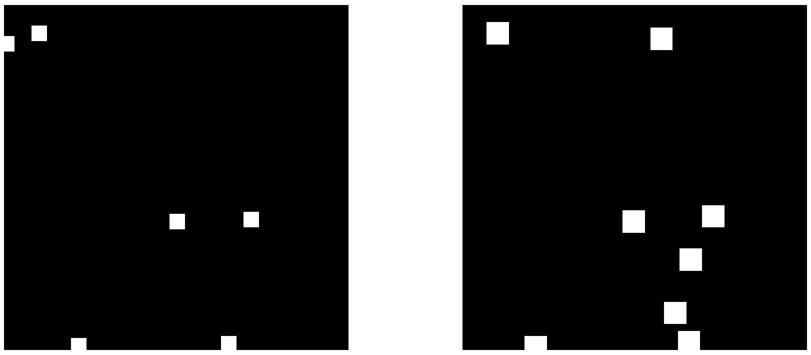

(a) Equal sized objects in USNO-B1.0 (b) Equal sized objects in Pan–STARRS1

Figure 4.8: Original objects from figure 4.7 replaced by equal sized objects in both USNO-B1.0 (figure 4.8a), and Pan–STARRS1 (figure 4.8b). Squared objects of size 9 are used for USNO-B1.0 images and 13 for Pan–STARRS1 images.

4.3.1.6 Accuracy

The last step is to compute the accuracy of a matching. The accuracy of a matching is defined as the number of remaining bright pixel value intensities (pixel values equal to 255) when the USNO-B1.0 image is subtracted from the Pan–STARRS1 image. In other words, let the USNO-B1.0 image in figure 4.8a be xs, the Pan–STARRS1 image in figure 4.8b be ys, and zs = xs − ys. Then the accuracy is defined as: follows:

$$\text{accuracy} = 100 - \frac{|\text{zs} \in \{255\}| \times 100}{|\text{xs} \in \{255\}|} \tag{4.1}$$

A fixed accuracy threshold of 80% is used in order to determine whether a mission is successfully completed or not. That is, if a user can create a matching where the accuracy as defined in equation 4.1 is greater or equal to the 80% fixed threshold, then the mission is considered as successfully completed (unlikely to be an anomaly), while the opposite means it is possible there is an anomaly between the two images.

It is worth noting the accuracy formula shown in equation 4.1 only works in one–direction. That is, it exclusively works to identify objects in USNO-B1.0 that have disappeared (or significantly moved) in Pan–STARRS1. An ideal solution would work both ways (i.e. it would also handle detecting appearing sources). Within the scope of this master thesis, the main goal was to create an evaluation technique for a matching that would make it difficult for an anomaly to be cataloged as a non–anomaly (since these are used by the ML-Blink algorithm to learn what to recommend). The outlined solution was the best method implemented that could fairly well handle the different type of artifacts present in the two datasets, and avoid anomalies being cataloged as non–anomalies.

4.3.2 ML-Blink API

The ML-Blink API implements the ML-Blink algorithm as described in section 3 using Python 3.5.2. It uses the MongoDB database in order to persist data, and the Celery asynchronous task queue to schedule and execute time consuming operations. The ML-Blink algorithm implementation can be split into three parts: the active set (4.3.2.1), potential anomalies (4.3.2.2), and crawling candidates (4.3.2.3).

4.3.2.1 The Active Set

The active set is a collection persisted in MongoDB comprised of missions which are considered to be non-anomalies. Table 4.1 shows the schema definition of a document of the active set.

`image_key`	An integer which identifies an image
`usno_band`	The USNO-B1.0 band of the image
`panstarr_band`	The Pan–STARRS1 band of the image
`usno_vector`	The resulting vector after processing the original USNO-B1.0 image.
`panstarr_vector`	The resulting vector after processing the original Pan–STARRS1 image.

Table 4.1: A description of the schema of a document of the active set collection.

When a user submits a mission in the ML-Blink UI, its data is received by the ML-Blink API and persisted in the database. Following that, a background task is created to further determine what to do with it.

The background task, named `tprocess_created_mission`, is in charge of defining whether a mission's data should be added to the active set or not. A mission is added to the active set if its `accuracy` achieved in the ML-Blink UI is at least as good as its `accuracy_threshold`. Additionally, the mission's data is analyzed to determine whether the user tried to at least do a matching by verifying whether the two images coordinates overlap (i.e. one image is placed on top of the other). If a mission must be inserted in the active set, the mission's details are pre–processed as described in algorithm 6. Finally, once the mission is successfully inserted in the active set collection, the `tcrawl_candidates` task is called to crawl a new candidate mission.

4.3.2.2 Potential Anomalies

The `tprocess_created_mission` is also in charge of inserting missions in the potential anomalies collection. A mission is added to the potential anomalies collection when a mission's `accuracy` is less than its `accuracy_threshold` and the mission's images overlap. If a mission is classified as a potential anomaly, the `tcrawl_candidates` task is not called.

The schema definition of a potential anomaly is shown in table 4.2. Its definition is similar to that of a document of the active set, but it differs in that it does not define the pre–processed vectors of the USNO-B1.0 and Pan–STARRS1 image specified by the `image_key`, `usno_band`, and `panstarr_band` attributes.

`image_key`	An integer which identifies an image
`usno_band`	The USNO-B1.0 band of the image
`panstarr_band`	The Pan–STARRS1 band of the image

Table 4.2: A description of the schema of a document of the potential anomalies collection.

4.3.2.3 Crawling Candidates

The `tcrawl_candidates` background task uses the Celery "chord" API to crawl for candidates. The Celery chord API allows to execute a task once a group of other parallel tasks have finished.

The `tcrawl_candidates` generates a list of image keys to analyze over all bands defined among the two datasets. This list is refereed to as the potential candidates list, since it contains the candidate that will be selected when the algorithm finishes. Missions which are known to be potential anomalies are filtered out of this list, since these have already been tagged for further analysis.

Once the potential candidates list has been generated and filtered, it is split into the available number of processes the machine where the ML-Blink API is running has. Each chunk is then processed in parallel by a different Celery worker. These workers compute the value of `v` for each potential candidate in its chunk using the implicit definition described in section 3.4 (algorithm 4). For each potential candidate, its v value is computed after it has been binarized with a fixed threshold value of 60 for USNO-B1.0 and 220 for Pan–STARRS1. The dimensionality of these potential candidates is also reduced and finally the remaining features normalized as described in the pseudo–code in algorithm 6.

Finally, once all chunks have been processed, a task is used to "reduce" the result, by retrieving the potential candidate that has the lowest `v` value among all the potential candidates analyzed by the different Celery workers. Since it is the most different to those in the active set (and therefore most likely to contain an anomaly), this candidate is inserted in the candidates collection. Table 4.3 shows the schema definition of a document of the candidates collection.

`image_key`	An integer which identifies an image
`usno_band`	The USNO-B1.0 band of the image
`panstarr_band`	The Pan–STARRS1 band of the image
`v`	The `v` value of this candidate computed by the AI–Crawler

Table 4.3: A description of the schema of a document of the candidates collection.

When the ML-Blink UI requests a new mission to the ML-Blink API, the ML-Blink API reads the candidates collection and selects the candidate with the minimum v value on it. If multiple candidates are tied for the minimum v value, the ML-Blink API randomly selects a candidate within those in the tie. Additionally, the selected candidate is removed from the candidates collection in MongoDB just before it is served to the ML-Blink UI. Finally, in the scenario where the ML-Blink UI requests a new mission and the candidates collection is empty, the ML-Blink API randomly selects a mission from the β–pack and serves it to the ML-Blink UI.

4.4 Results

The result section focuses in showcasing the evaluation of the ML-Blink algorithm and comparing it to randomly searching for anomalies in the β–pack dataset. To do so, the ML-Blink algorithm is executed in isolation (i.e. without using the ML-Blink UI). The reasoning behind this decision is that it is simpler and faster to evaluate the algorithm without having to depend on whether a user made a good matching or not. In order to enable this, the β–pack was slightly modified to contain a total of 7 anomalies. These anomalies were created by randomly selecting a few missions, and manually altering them as explained in section 4.1.1 and shown in figure 4.2. All anomalies consisted of removing objects close to the center from Pan–STARRS1 that are discernible in their USNO-B1.0 equivalent. Through out this section, the notation “`image_key.usno_band.panstarr_band`” will be used to refer to an image of the β–pack dataset.

While the β–pack dataset does contain the vanishing point identified in [20], the evaluation will not take it into account because the main purpose of this work is to study the overall performance of the ML-Blink algorithm, not its performance specifically targeted towards real astronomical anomalies. As such, the artificial anomalies were created so that these were easier to detect. Finally, artificial anomalies were constructed because of the possibly overwhelming class imbalance present in the dataset. That is, most observations are expected to be normal, while in some exceptional occasions anomalies could exist.

As seen in table 3.1, some Pan–STARRS1 color–bands are mapped into multiple

USNO-B1.0 color–bands. For instance, if a Pan–STARRS1 anomaly is manually created in color–band `g`, it must be mapped to both USNO-B1.0 `blue1` and `blue2`. Therefore, that single anomaly actually counts as two, since the ML-Blink algorithm is expected to find it across the two USNO-B1.0 bands. Table 4.4 shows the complete list of artificial anomalies that were created to evaluate the algorithm.

Image Key	USNO-B1.0 Band	Pan–STARRS1 Band
13	`blue1`	`g`
13	`blue2`	`g`
56	`blue1`	`g`
56	`blue2`	`g`
679	`ir`	`z`
831	`red1`	`r`
831	`red2`	`r`

Table 4.4: Complete list of anomalies that were created to evaluate the ML-Blink algorithm.

A naive solution for the requirements of the VASCO project would simply randomly attempt to find the anomalies in the β–pack. Since there are a total of 5005 observations in the β–pack, a random recommender system would recommend an item with probability of $1/5005 = 0.0002$. Furthermore, the ML-Blink algorithm will be tested by executing it for 200 time steps. A random recommender system that is ran 200 times, with the goal of finding 7 anomalies in a dataset of 5005 observations would be expected to find $7 \times 1/5005 \times 200 = 0.27$ anomalies.

Figure 4.9 shows the ROC curve created by running the ML-Blink algorithm for 200 time steps. The image vectors were reduced to 250 projections, and the binarization thresholds for USNO-B1.0 and Pan–STARRS1 were 60 and 220 respectively. These parameters were adjusted using trial and error by analyzing the results of the ROC curve, AUC, and the number of anomalies found during the specified number of time steps in the experiments. While there is some randomization involved in the algorithm, with the parameters previously specified the results are easily reproducible: the ML-Blink algorithm consistently achieves an AUC around 0.70, and finds 2 up to 4 anomalies out of 7. In the experiment shown in figure 4.9, the ML-Blink algorithm found 3/7 anomalies, and achieved the best AUC = 0.77 at $t = 100$.

As explained in section 3.7, the discrimination threshold used to create the ROC curve is “sweeping” the entire range of v values from $\min(v)$ up to $\max(v)$ in a time step t. It is thus reasonable that for all time steps, the ROC curve in figure 4.9 starts with an ascend (i.e. more true positives) since the v value is small and we expect more anomalies to evaluate to a v value in that range. On

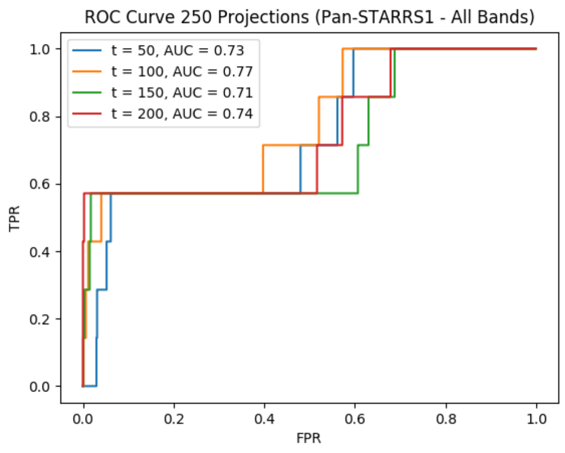


Figure 4.9: ROC curve and AUC of the evaluation of the ML-Blink algorithm using 250 projections for a total of 200 time steps.

the other hand, as the discrimination threshold is increased, the false positives rate starts to catch up and eventually dominates the ROC curve. The ML-Blink algorithm was consistently able to find the first 2 or 4 anomalies listed in table 4.4, but it could not find the last 3 - even when more time steps were used in the evaluation. This is also why the ROC curve in figure 4.9 has three “steps” before reaching a TPR = 1.0; these three steps are the 3 anomalies that the ML-Blink algorithm is unable to find because their v value is too large.

Figure 4.10 shows how the v value of known anomalies that were found versus a normal observation changes as the ML-Blink algorithm is taught over time. Additionally, the figure includes the $\min(v)$ value at each time step. As expected, the v value of the known anomalies is close to the $\min(v)$ value of each time step, while the normal observation results in a larger v value. Note the v value of an anomaly might be smaller than the $\min(v)$ of a time step only after it has been found. The reason for this is that once an anomaly is found, it is filtered out of the potential candidates to be crawled, otherwise the ML-Blink algorithm will continue to recommend it. In this scenario, the v value of an anomaly was computed after it has been found for illustrative purposes only. The anomalies in figure 4.10 were found in the following order: `13.blue2.g` in $t = 110$, `56.blue2.g` at $t = 158$, and finally `56.blue1.g` in $t = 189$.

The same plot is shown in figure 4.11, but this time it shows the anomalies the ML-Blink algorithm could not find. Except for anomaly `13.blue1.g`, the results are the opposite of what is expected, since the v value of the anomalies is closer to the normal observation than it is to the $\min(v)$ value at each time step.

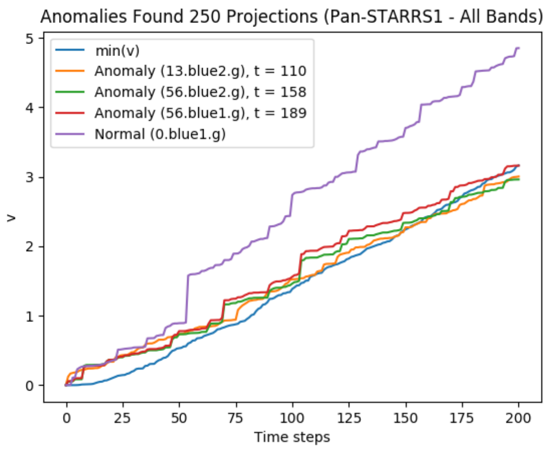


Figure 4.10: Evaluation of known anomalies (that the ML-Blink algorithm found) versus normal observations in comparison to the min(v) value at each time step.

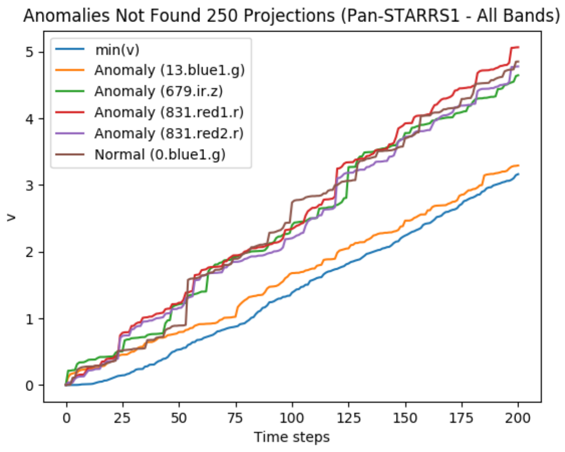


Figure 4.11: Evaluation of known anomalies (that the ML-Blink algorithm could not find) versus normal observations in comparison to the min(v) value at each time step.

The common denominator of the 4 anomalies the ML-Blink algorithm evaluated to be close to the min(v) value of any time step is that these are all defined in the `g` color–band of Pan–STARRS1. To further understand the capabilities of the ML-Blink algorithm when using projections for dimensionality reduction, each Pan–STARRS1 band was evaluated in isolation. The experiments were conducted one Pan–STARRS1 band at a time. That is, Pan–STARRS1 color–band `g` (versus USNO-B1.0 color–bands `blue1` and `blue2`) with a total of 2002 observations and 4 artificial anomalies was tested in isolation. Next, Pan–STARRS1 color–band `r` (versus USNO-B1.0 color–bands `red1` and `red2`) with a total of 2002 observations and 2 artificial anomalies was evaluated next. Finally, Pan–STARRS1 color–band `z` (versus USNO-B1.0 color–band `ir`) with a total of 1001 observations and 1 artificial anomaly was tested.

Figure 4.12 shows the ROC and AUC achieved by ML-Blink when evaluating the Pan–STARRS1 color–band `g` in isolation. As expected, ML-Blink does very well, consistently achieving an AUC of at least 0.90 for all time steps shown in the plot. It found 3 out of 4 anomalies in the following order: `56.blue2.g` in $t = 62$, `13.blue2.g` at $t = 82$, and `56.blue1.g` in $t = 105$. Figure 4.13 shows the evaluation of the v value of known anomalies (that were found when testing Pan–STARRS1 color–band `g` in isolation) versus a normal observation. Once again, as expected, the plot shows ML-Blink consistently evaluates the v value of known anomalies to be significantly smaller than that of a normal observation.

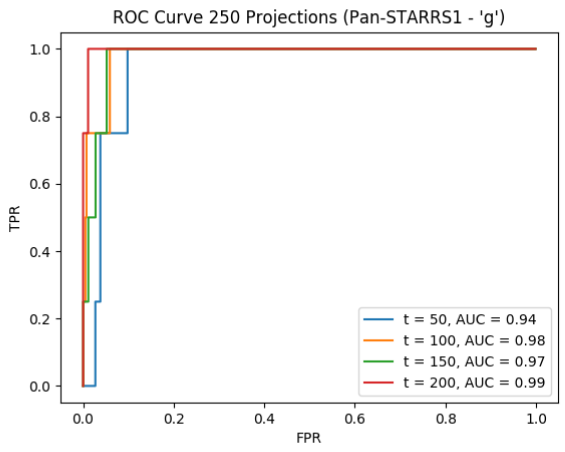


Figure 4.12: ROC curve and AUC achieved by the ML-Blink algorithm when evaluating observations in the Pan–STARRS1 color–band `g` only (versus USNO-B1.0 color–bands `blue1` and `blue2`) using 250 projections for dimensionality reduction.

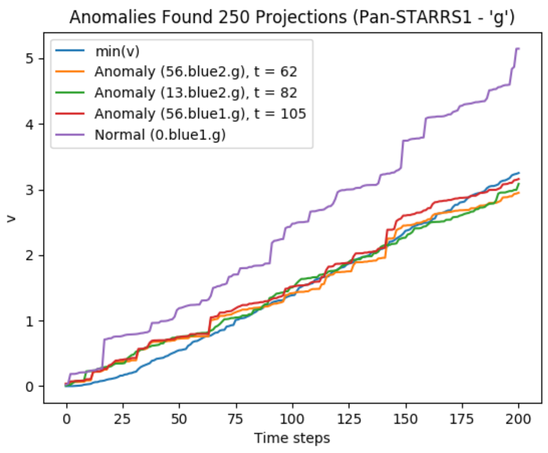


Figure 4.13: Evaluation of known anomalies that were found by the ML-Blink algorithm when evaluating the Pan–STARRS1 color–band `g` (versus USNO-B1.0 color–bands `blue1` and `blue2`) in isolation using 250 projections for dimensionality reduction.

Figure 4.14 shows the ROC and AUC of the evaluation of the Pan–STARRS1 `r` band in isolation. The results achieved are quite poor: it does not find any anomalies (out of 2), and achieves an AUC in the 0.5 range. Figure 4.15 shows the ML-Blink algorithm evaluates the anomalies in the dataset as if these were normal observations (their v values are closer to that of a normal observation than to the min(v) of any time step).

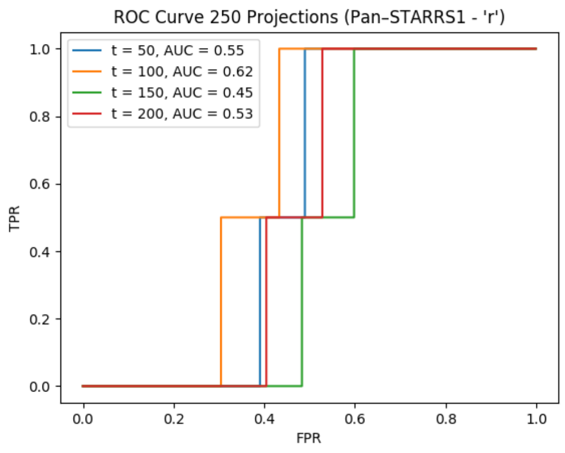


Figure 4.14: ROC curve and AUC achieved by the ML-Blink algorithm when evaluating observations in the Pan–STARRS1 color–band `r` only (versus USNO-B1.0 color–bands `red1` and `red2`) using 250 projections for dimensionality reduction.

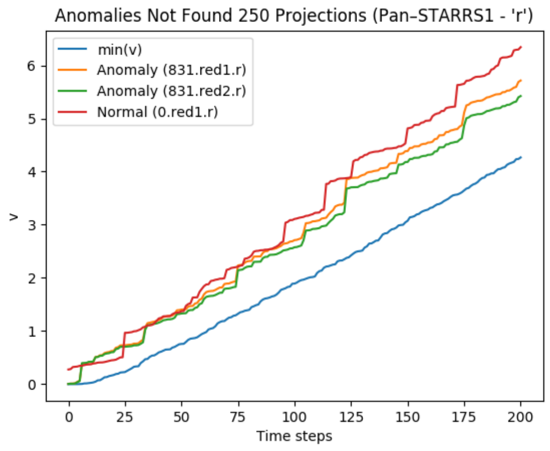


Figure 4.15: Evaluation of known anomalies that were not found by the ML-Blink algorithm when evaluating the Pan–STARRS1 color–band `r` (versus USNO-B1.0 color–bands `red1` and `red2`) in isolation using 250 projections for dimensionality reduction.

The results of the evaluation of Pan–STARRS1 color–band `z` are shown in figures 4.16 and 4.17. Once again, the ML-Blink algorithm performance is poor, with an overall AUC lower than 0.5, and unable to find the single anomaly defined in this experiment (`679.ir.z`). Moreover, similar to figure 4.15, the ML-Blink algorithm consistently evaluates the known anomaly closer to a normal observation than to the min(v) value of any time step.

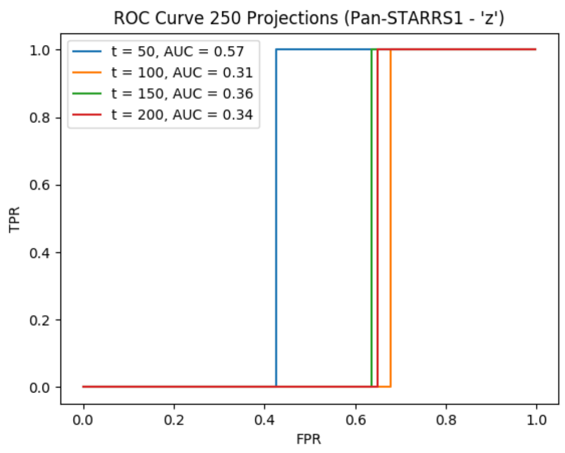


Figure 4.16: ROC curve and AUC achieved by the ML-Blink algorithm when evaluating observations in the Pan–STARRS1 color–band `z` only (versus USNO-B1.0 color–band `ir`) using 250 projections for dimensionality reduction.

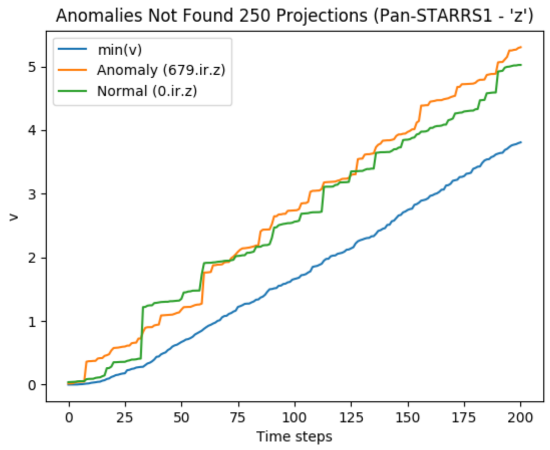
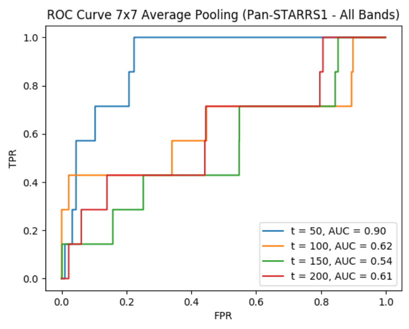


Figure 4.17: Evaluation of known anomalies that were not found by the ML-Blink algorithm when evaluating the Pan–STARRS1 color–band `z` (versus USNO-B1.0 color–bands `ir`) in isolation using 250 projections for dimensionality reduction.

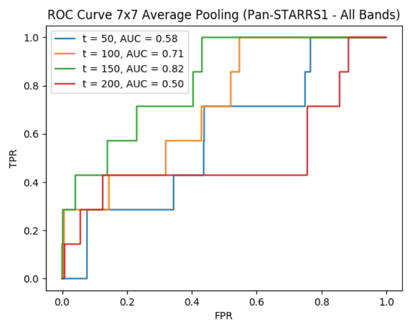
The same experiments were conducted but using average pooling with a kernel of size 7×7 for dimensionality reduction instead. The binarization thresholds for USNO-B1.0 and Pan–STARRS1 remained equal as when using projections.

The results of the ML-Blink algorithm when using average pooling for dimensionality reduction were unstable and difficult to reproduce. Figure 4.18 shows the ROC curve for two independent evaluations of the ML-Blink algorithm using average pooling for dimensionality reduction for a total of 200 time steps.

While it was able to find 2–3 anomalies, the AUC in sub–figure 4.18a deviates from 0.54 at $t = 150$ up to 0.90 in $t = 50$, and from 0.50 in $t = 200$ up to 0.82 at $t = 150$ in sub–figure 4.18b. This suggests the representation of the learned weights in the matrix $\mathbf{D}$ when using average pooling for dimensionality reduction is volatile, and small changes of the members that are inserted in the active set can greatly affect the outcome of the model.



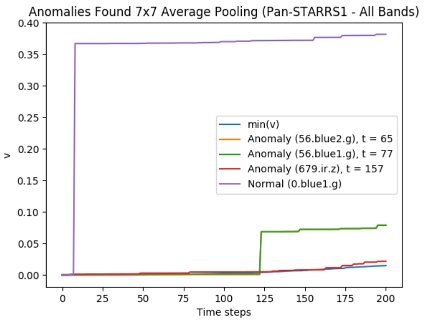
(a) Experiment 1 ROC Curve and AUC



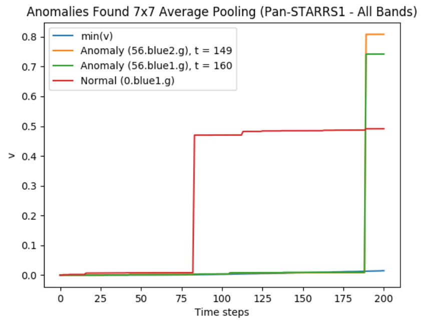
(b) Experiment 2 ROC Curve and AUC

Figure 4.18: ROC curve and AUC of two independent runs of the ML-Blink algorithm when using average pooling with a kernel of size 7×7 for a total of 200 time steps. The AUC in sub–figure 4.18a deviates from 0.54 at $t = 150$ up to 0.90 in $t = 50$, while the AUC of sub–figure 4.18b goes from 0.50 in $t = 200$ up to 0.82 at $t = 150$. The large changes in the AUC suggest the representation of the learned weights in the matrix $\mathbf{D}$ is volatile.

The instability can be better visualized when analyzing the v value of known anomalies versus a normal observation as the ML-Blink algorithm is taught over time. Figure 4.19 shows a plot of anomalies the ML-Blink algorithm found versus a normal observation for 200 time steps for both experiments in figure 4.18. Note the v value of the known anomalies in sub–figure 4.19b increases towards the end, and it goes even higher than the v value of a normal observation. This behavior is the opposite of what the ML-Blink algorithm is designed to do, which suggests average pooling with the configuration used when running these experiments does not capture a good representation of the features that are required to learn what is an anomaly. Additionally, the range of v values the ML-Blink computes in sub–figure 4.19a goes from 0 up to 0.40, while in sub–figure 4.19b it is from 0 until 0.80. When using projections, the v values grow is smoother and typically ranges in the 0–6 range regardless of what Pan–STARRS1 band is being crawled.



(a) Experiment 1 Anomalies Found



(b) Experiment 2 Anomalies Found

Figure 4.19: Evaluation of the v value of known anomalies and a normal observation as the ML-Blink algorithm is taught over time when using average pooling for dimensionality reduction. Notice the unstable change of the v value of the observations in comparison to using projections for dimensionality reduction (figure 4.10).

Similar results were obtained in other experiments when using average pooling for dimensionality reduction (both when crawling all Pan–STARRS1 color–bands or one at a time). The only major difference being that sometimes it would find the `679.ir.z` anomaly (as seen in sub–figure 4.19a), which projections could not. Regardless of that, the results obtained from multiple runs were highly volatile, and it was decided to not continue these experiments further due to their instability and lack of reproducibility.

Overall, when using all Pan–STARRS1 color–bands and projections for dimensionality reduction as seen in figure 4.9, the ML-Blink algorithm performs substantially better than randomly predicting the label of an observation. It also finds more anomalies when crawling 5005 observations (all Pan–STARRS1 color–bands) than what a random recommender system would be expected to do (i.e. $3 > 0.27$). Nonetheless, these results are primarily determined by the good performance of the ML-Blink algorithm in the Pan–STARRS1 color–band `g`. The ML-Blink algorithm performance significantly degrades when crawling the Pan–STARRS1 color–bands `r` and `z`. For both of these color–bands, it was shown the ML-Blink algorithm achieves an AUC in the 0.5 range (or worse). Additionally, when crawling the Pan–STARRS1 color–bands `r` and `z` in isolation, the ML-Blink algorithm was unable to find any of the artificial anomalies defined in these color–bands.

Chapter 5

Conclusion and Future Work

After a general introduction to recommender systems, supervised, online, and active learning, the ML-Blink algorithm was introduced and some of its mathematical properties were intuitively explained. The relation of the ML-Blink algorithm to the aforementioned learning techniques was described, and finally, the concepts of normalization and dimensionality reduction were defined and illustrated in terms of how these are used in the ML-Blink algorithm.

The methodology section outlined how the theoretical definition of the ML-Blink algorithm was approached for this particular work. A short explanation of the dataset the ML-Blink algorithm would be evaluated on was given as well. Next, the implicit representation of the weight matrix $\mathbf{D}$, how each observation from the dataset is retrieved, and the parallelization of the ML-Blink algorithm were studied. Lastly, an evaluation technique based on the ROC curve, AUC, and the number of anomalies found was presented.

The case study section described in detail a UI that allows users to match two images of the same location in the night sky from distinct datasets, and an API that processes and persists data created by the UI. The architecture constructed for the UI and API to interact with each other along with the required technologies to handle asynchronous tasks execution and data persistence were motivated and explained.

The results section focused on evaluating the ML-Blink algorithm in isolation, that is, without the ML-Blink UI. The ML-Blink algorithm using projections for dimensionality reduction was able to achieve an AUC around the 0.70 range when crawling all Pan–STARRS1 color–bands, and find 2–4 artificial anomalies out of 7. Since the artificial anomalies the ML-Blink algorithm found were always in the Pan–STARRS1 color–band g, in order to better understand the algorithm's performance, each of the Pan–STARRS1 color–bands was evaluated in isolation. The ML-Blink algorithm achieved an excellent AUC, typically above 0.90, when crawling the Pan–STARRS1 color–band `g` in isolation. Nonetheless, the ML-Blink algorithm performance when crawling the Pan–STARRS1 color–bands `r` or `z` was poor, consistently achieving an AUC below 0.5 and no artificial anomalies were found at all. Finally, a few configurations of average pooling as a dimensionality reduction technique were tested, but ultimately no further experiments were conducted with this technique due to the instability and lack of reproducibility encountered in these evaluations.

Before introducing the binarization step when retrieving an observation from the dataset, the ML-Blink algorithm was completely unable to find any of the anomalies manually inserted in the β–pack observations. This suggests more work can be done in the pre–processing steps. For instance, the binarization threshold is fixed, and furthermore, it has the same value regardless of what Pan–STARRS1 color–band is being crawled. It might be worth experimenting with automatic thresholding, and even consider using different pre–processing steps for each color–band. Additionally, it could be beneficial to assess the algorithm using other dimensionality reduction techniques to further evaluate if these could create a better representation of the features that are needed to identify anomalies in the datasets' observations.

The current implementation of ML-Blink crawls the same 5005 observations from the β–pack in all time steps. Eventually, when using a larger dataset, it must be decided how to crawl each observation. The larger datasets could use a complete crawler from start to finish (e.g. 0–999, 1000–1999, 2000–2999, etc), or a specified number of observations might be randomly selected in each time step within predefined lower/upper bound limits.

Within the proposed VASCO software architecture, this work focused the least in the ML-Blink UI. Nonetheless, the ML-Blink UI is essential for the rest of the project, as it provides the UI that allows users to label observations and ultimately feed the ML-Blink algorithm. Possible improvements to the ML-Blink UI include gamification strategies to incentivize users to solve more missions with a high degree of accuracy and the addition of a rotation feature which allows to match observations that could not be matched otherwise.

Chapter 6

Acknowledgments

I want to genuinely express my gratitude with my supervisor Kristiaan Pelckmans, who was the one who initially introduced me to the VASCO project and motivated me to contribute to it. Kristiaan created the ML-Blink algorithm, and the work presented in this thesis would not have been possible without his support and constant guidance. I also express my warm thanks to my reviewer Mikael Laaksoharju for providing timely feedback, and a vision for the overall ML-Blink software ecosystem.

Most importantly, I am grateful with my family, and especially my wife Yassibel Duque, for their constant support and encouragement throughout the past two years that conclude with this project.

Bibliography

[1] The pan-starrs1 data archive home page. `https://panstarrs.stsci.edu`. Accessed April 19, 2019.

[2] Relationship to the world wide web and rest architectures. `https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest`. Accessed April 3, 2019.

[3] Usno-b1.0 catalog. `https://www.ap-i.net/skychart/en/news/usno-b1.0_catalog`. Accessed March 9, 2019.

[4] The vanishing and appearing sources during a century of observations project. `https://vasconsite.wordpress.com/`. Accessed May 23, 2019.

[5] Y-Lan Boureau, J Ponce, and Yann Lecun. A theoretical analysis of feature pooling in visual recognition. *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*, pages 111–118, 11 2010.

[6] Charu C. Aggarwal. *Recommender Systems*. Springer Science+Business Media, 01 2016.

[7] Celery Core Team. Celery: Distributed task queue. `http://www.celeryproject.org/`. Accessed April 19, 2019.

[8] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861 – 874, 2006. ROC Analysis in Pattern Recognition.

[9] Flask Core Team. Flask: A python microframework. `http://flask.pocoo.org/`. Accessed April 19, 2019.

[10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2001.

[11] Davy Kirkpatrick. United states naval observatory b1.0 catalog. `https://irsa.ipac.caltech.edu/data/USNO_B1/usnob1_description.html`. Accessed March 9, 2019.

[12] Nick Littlestone. From on-line to batch learning. *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 269–284, 12 1989.

[13] Richard M. Karp. On-line algorithms versus off-line algorithms: How much is it worth to know the future? volume 1, pages 416–429, 01 1992.

[14] G. J. Madsen and B. M. Gaensler. A Precision Multi-band Two-epoch Photometric Catalog of 44 Million Sources in the Northern Sky from a Combination of the USNO-B and Sloan Digital Sky Survey Catalogs. *The Astrophysical Journal Supplement*, 209(2):33, Dec 2013.

[15] Axel Petzold and Eckhart Pitz. The historical origin of the pulfrich effect: A serendipitous astronomic observation at the border of the milky way. *Neuro-Ophthalmology*, 33, 07 2009.

[16] Redis Core Team. Redis: In–memory data structure store. `https://redis.io/`. Accessed May 23, 2019.

[17] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[18] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[19] Michel Verleysen and Damien François. The curse of dimensionality in data mining and time series prediction. *Lecture Notes in Computer Science*, 3512:758–770, 06 2005.

[20] Beatriz Villarroel, Inigo Imaz, and Josefine Bergstedt. Our Sky Now and Then: Searches for Lost Stars and Impossible Effects as Probes of Advanced Extraterrestrial Civilizations. *The Astronomical Journal*, 152(3):76, Sep 2016.

[21] Beatriz Villarroel et al. The Vanishing & Appearing Sources during a Century of Observations project: Paper I. A sample of USNO objects missing from modern sky surveys, and the follow-up observations of an old "missing" star. 2019. To be submitted.

[22] Vue Core Team. Vue.js: The progressive javascript framework. `https://vuejs.org/`. Accessed April 19, 2019.