NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Adjustable Publisher/Subscriber system with Machine Learning

Ioannis G. Kalopisis

SUPERVISOR: Alexandros Ntoulas, Assistant Professor NKUA

ATHENS

October 2020

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αυτορυθμιζόμενο σύστημα Εκδότη/Συνδρομητή με Μηχανική Μάθηση

Ιωάννης Γ. Καλοπίσης

ΕΠΙΒΛΕΠΩΝ: Αλέξανδρος Ντούλας, Επίκουρος Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

Οκτώβριος 2020

BSc THESIS

Adjustable Publisher/Subscriber system with Machine Learning

Ioannis G. Kalopisis
SN: 1115201500059

SUPERVISOR: Alexandros Ntoulas, Assistant Professor NKUA

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αυτορυθμιζόμενο σύστημα Εκδότη/Συνδρομητή με Μηχανική Μάθηση

Ιωάννης Γ. Καλοπίσης
ΑΜ: 1115201500059

ΕΠΙΒΛΕΠΩΝ: **Αλέξανδρος Ντούλας**, Επίκουρος Καθηγητής ΕΚΠΑ

ABSTRACT

The rapid development of the Internet of Things-IoT leads to the development of many distributed systems and smart applications. These applications generate and demand huge amounts of data every day. It is therefore easily understood that a system is needed to transfer this data. In order not to limit the development of large-scale applications, this system should both be independent and have a decentralized character. This transfer is undertaken by Publisher/Subscriber type messaging systems, such as Apache Kafka. This system functions as the intermediate link between a producer and a consumer for the transmission of messages.

These systems can be hosted on server clusters scattered around the world, depending on the size of the application they serve and the size of the data stream. We can understand that these are huge systems that adapt to the needs of the user. Therefore, the system parameters must be adjusted each time according to their application, use, type and data flow. However, apart from the tedious and time consuming process, the result does not always lead to optimal system performance.

In this project we present an attempt to automate the process of automatically optimizing system performance for pub/sub systems using ML. By using algorithms and Machine Learning techniques such as regression and classification, we try to predict the parameters of the Kafka Publisher/Subscriber system, aiming at specific system requirements.

You can find the code for this thesis, as well as the data, images, and results at the following link: https://github.com/GiannisKalopisis/Adjustable-pub-sub-system.

SUBJECT AREA: Machine Learning and Publisher/Subscriber systems optimization

KEYWORDS: Machine Learning, Kafka, Publisher/Subscriber, Regression, Classification

ΠΕΡΙΛΗΨΗ

Η ταχεία ανάπτυξη του Internet of Things-IoT οδηγεί στην ανάπτυξη πολλών κατανεμημένων συστημάτων και έξυπνων εφαρμογών. Οι εφαρμογές αυτές παράγουν αλλά και ζητάνε τεράστιες ποσότητες δεδομένων καθημερινά. Γίνεται λοιπόν εύκολα αντιληπτό ότι χρειάζεται ένα σύστημα για τη μεταφορά αυτών των δεδομένων. Για να μην περιορίζεται η ανάπτυξη των εφαρμογών μεγάλης κλίμακας, θα πρέπει το σύστημα αυτό να είναι ανεξάρτητο και να έχει έναν αποκεντρωμένο χαρακτήρα. Τη μεταφορά αυτή αναλαμβάνουν συστήματα μετάδοσης μηνυμάτων τύπου Εκδότη/Συνδρομητή, όπως το Kafka της Apache. Το σύστημα αυτό αποτελεί τον ενδιάμεσο κρίκο, μεταξύ ενός παραγωγού και ενός καταναλωτή για τη μετάδοση μηνυμάτων.

Τα συστήματα αυτά μπορούν να φιλοξενούνται σε συμπλέγματα διακομιστών, διασκορπισμένα σε όλο τον κόσμο, ανάλογα με το μέγεθος της εφαρμογής που εξυπηρετούν αλλά και το μέγεθος της ροής δεδομένων. Μπορούμε να καταλάβουμε ότι πρόκειται για τεράστια συστήματα που προσαρμόζονται ανάλογα με τις ανάγκες του χρήστη. Έτσι λοιπόν θα πρέπει κάθε φορά να ρυθμίζονται οι παράμετροι του συστήματος ανάλογα με την εφαρμογή, τη χρήση, το είδος και τη ροή των δεδομένων. Εκτός όμως από επίπονη και χρονοβόρα διαδικασία, το αποτελέσμα δεν οδηγεί πάντα στη βέλτιστη απόδοση του συστήματος.

Στην εργασία αυτή παρουσιάζουμε μία προσπάθεια αυτοματοποίησης αυτής της διαδικασίας. Με τη χρήση αλγορίθμων και τεχνικών Μηχανικής Μάθησης, όπως η παλινδρόμηση και η κατηγοριοποίηση, προσπαθούμε να προβλέψουμε τις τιμές των παραμέτρων του συστήματος Εκδότη/Συνδρομητή Kafka, έχοντας ως στόχο συγκεκριμένες απαιτήσεις από το σύστημα.

Μπορείτε να βρείτε τον κώδικα για αυτήν τη πτυχιακή, καθώς και τα δεδομένα, τις εικόνες και τα αποτελέσματα στον ακόλουθο σύνδεσμο: https://github.com/GiannisKalopisis/Adjustable-pub-sub-system.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Μάθηση και βελτιστοποίηση συστημάτων Εκδότη/Συνδρομητή

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Μηχανική Μάθηση, Kafka, Εκδότη/Συνδρομητή, Παλινδρόμηση, Κατηγοριοποίηση

To my family and friends.

ACKNOWLEDGEMENTS

I would firstly like to thank my supervisor, Asst. Prof. Alexandros Ntoulas, who gave me the opportunity to work on a very interesting and rapidly growing topic for my thesis. I am deeply appreciative of his support throughout the elaboration of this project. I would also like to thank him for the equipment and resources he has provided to me so that I can carry out the necessary experiments and complete this thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This project was developed in Athens, Greece between January 2020 and October 2020, and constitutes my thesis. For the first part, it was important to get acquainted with the messaging technologies, especially with the Publisher/Subscriber system Apache Kafka, that we used. As a result, it was necessary to learn, in depth, how to use the tool so as to best harvest measurements and data. However, the second part which dealt with data processing and the application of Machine Learning algorithms on them, was equally important, as we had to understand the data and interpret the results of the algorithms, the same results that led us towards the next steps of the project.

1. INTRODUCTION

The development of the Internet has significantly changed the scale of distributed systems, which contain entities scattered around the world. The behavior and, obviously, the location is different for each system. The advent of the Internet of Things, in addition to development of distributed systems, brings with it a vast variety of smart applications.

The huge flow of data through applications and the nature of distributed systems, as mentioned earlier, led us to seek a solution for flexible communication and data transmission, which reflects the dynamic and decentralized nature of applications. Individual point-to-point and modern communication solutions make applications rigid, rendering the development of dynamic large-scale applications impossible. It therefore becomes clear that the burden of communication and data transfer from application designers should be removed and transferred to an intermediate software infrastructure, based on an appropriate communication format.

Many systems have been proposed since this problem became apparent. One of the most dominant models today, is the Publisher/Subscriber. The Pub/Sub interaction scheme provides the loosely linked interaction format required in such a large-scale communication system.

A Publisher/Subscriber system allows subscribers to connect and express their interest in a topic, content, etc. (depending on the subscription model of system), in order to be notified of any event that suits their interests, and published from a publisher. Accordingly, the publisher generates and publishes events and data on a software bus (or event manager), which is the backbone of the system. Consumers connected to it subscribe to the information they want to receive, depending on their interests. The central system is, essentially, a notification management system that provides storage, efficient management and promotion of data/events.

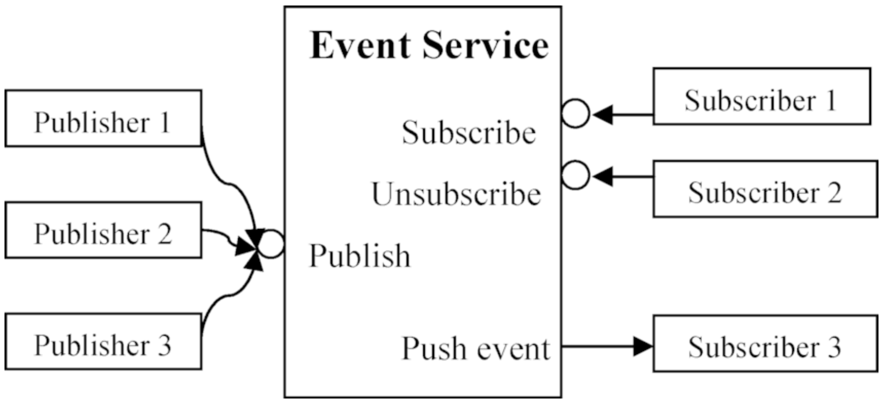


Figure 1.1: Publisher/Subscriber communication model

The power of such a system based on the interaction with events, is itself based on the full decoupling of space, time and synchronization between the two parties; the publisher and the subscriber.

As we can easily understand, the extent of such an industrial-level system is large. In order for the system to withstand time and the constant pressure exerted by the users' submissions, the ever-increasing flow of data through it, and its proper performance, regardless of the host machine or machines, it should be able to do the appropriate adjustments. Therefore, all Publisher/Subscriber systems, particularly the one we are studying in this project, i.e., Apache Kafka, have many parameters that are adjusted accordingly, enabling each user to achieve optimal performance for their requirements.

In recent years, the development of computer science has highlighted new fields that are constantly evolving. Some of these fields include Machine Learning (ML) and Artificial Intelligence (AI). These new fields are based on algorithms and a wealth of data that are provided to them to train their ”mind” and be able to make decisions for us, or give us solutions to problems more easily, dependably and much faster than the human mind. As mentioned earlier, the development of the internet and technology provides us daily with the abundance of data we need to continue advancing such fields and observing the advantages they offer to humanity. The development of these fields itself gives us new fields of science that we can explore, as well as new problems to solve.

This project is based on one of the most well-known Publisher/Subscriber systems, Apache Kafka. We firstly try to explain exactly what these systems are and how they work. The next major pillar of this work regards data collection. In general, data collection is one of the most basic parts of such studies, particularly when fields of science such as Data Mining, Machine Learning and Artificial Intelligence, are involved. Alongside data collection, other procedures such as visualizing and clearing data from possible noise, follow. Filling in missing values from datasets is a common occurrence, as is turning their values into useful and easy-to-process ones. Then, using various Machine Learning techniques such as regression and classification, we try to navigate through the data and discover various patterns that may exist within them. The ultimate goal of the thesis is to investigate the feasibility of predicting the system parameter values automatically, for a set of given requirements. This could predict parameters values much faster and more accurately than exploring each user blindly to adjust the system to its optimal performance levels.

2. RELATED WORK

In recent years, the development of the internet and distributed applications have pushed Publisher/Subscriber systems to their limits. The rapid increase in traffic and data transferring has led many developers to look for new ways to increase data transfer performance.

As we have mentioned, Apache Kafka is one of the most well-known and widely used Pub/Sub systems. Many users are trying to increase its performance. Generally, the performance of such systems or similar systems are proposed having the corresponding improvements. There are various studies and approaches to the subject. Some try to increase its efficiency by studying and changing the network itself, such as by modifying the network lines, i.e. the materials through which the information is transmitted, or the protocols [1], [2]. Unfortunately, this has the disadvantage of the network not being the same everywhere; therefore it does not provide a universal solution. Others, on the other hand, attempt to increase performance through hardware adjustments and improvements. The issue in this solution lies the fact that optimal performance is closely related to the specific hardware. Likewise, this is not an efficient solution either, as not all users can have the same hardware. Furthermore, the hardware itself is being improved at the same rapid pace as the software.

In another light, others try to study the system itself to find the best way to configure it. Through studying its behavior, they attempt to adjust its parameters according to their problem. Unfortunately, this method is also not efficient, as it can be both time-consuming, as well as not being possible at all.

After thorough study, we decided to suggest an alternative way to solve this problem through this thesis. We did not find any additional work that is related to the work in this thesis. To the best of our knowledge, the combination of Machine Learning and Artificial Intelligence techniques with Publisher/Subscriber systems, such as Kafka, for the sake optimal configuration, is a relatively new field that has not been sufficiently studied.

3. BACKGROUND

3.1 Alternative Communications Systems - Predecessors

Before the introduction of the Publisher/Subscriber scheme, many other communication architectures had been proposed. All these models are, essentially, the predecessors of the Publisher/Subscriber scheme, as they have all offered some of their features in its development. They all had benefits and drawbacks, depending on their level of abstraction. We will try to give a general assessment of the main communication models that were proposed [3].

3.1.1 Message Passing

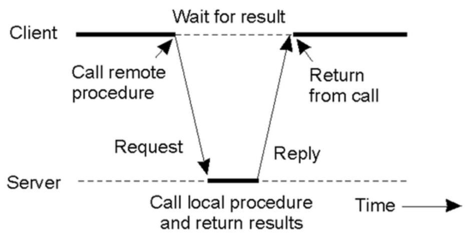
Message passing is a form of low-level communication in which the producer sends messages through a communication channel, asynchronously, and the consumer listens to the channel synchronously, and receives the messages. We observe that the two sides are coupled in space and time, as they must be active at the same time are aware of each other. This model could be said to be the initial predecessor of distributed communications [4], [5].

3.1.2 Remote Procedure Call - RPC

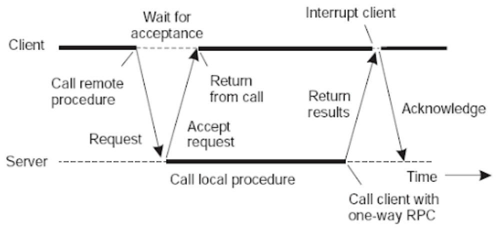
Remote Procedure Call, or RPC, is one of the most well-known forms of distributed communication. In distributed systems, an RPC occurs when a program triggers the execution of a subroutine on another computer (usually into a shared network). The call is encrypted, as if it were a common case, i.e. originating from the local computer. The developer does not implement the details of the remote communication; in other words, the code remains the same regardless of whether the subroutine was local or remote. This form of client-server interaction is typically implemented through a request-response messaging system. By making remote interactions the same as local ones, the RPC model makes distributed application development and scale up much easier and more efficient [6].

The RPC model implies a level of location opacity, i.e. local and remote calls are very similar. However, they are not identical. Remote calls are usually slower and less reliable, and need to be addressed explicitly. Various approaches have been proposed to deal with this problem which, again, had both pros and cons.

In the first image, we observe the RPC model in which the customer does not receive confirmation from the remote user. The second image features a variation of this model, where the customer receives confirmation. This variant is very common in communication systems and is often implemented by them, though it is up to the user whether to activate it or not. This variant provides confidence in the transmission of requests/messages but increases communication time. It is not suitable in cases where we are not so interested



(a) RPC without acknowledge



(b) RPC with acknowledge

Figure 3.1: Two implementations of RPC model

in the security of the messages, in contrast to the communication speed. The second variant with the confirmation messages is also used by the TCP transfer protocol, which guarantees the secure transfer of our data to the internet, unlike the UDP protocol, which is interested in the speed of message transmission [7], [8], [9], [10].

3.1.3 Notifications

The purpose of this communication model is to achieve synchronization decoupling. It achieves this by splitting the remote call into two asynchronous calls. The first call is sent by the client to the server and contains information about what the client is requesting, as well as information about callback to the client. The second part of the call is sent from the server to the client, who simply returns the answer. Although we provided a description of an one-to-one communication, the model can easily extend to many-to-many communication as well [3].

3.1.4 Shared Spaces

Communication through shared memory is one of the oldest ways of communication. Currently, it is mainly used by operating systems for process communication. Distributed shared memory (DSM) is a variation of this type of communication. Shared memory consists of a distributed shared system, where participants synchronize and communicate through access to shared data [11], [12].

This model provides space decoupling, as producers and consumers do not know who is on the other side, and time decoupling, as both sides do not need to be connected at the same time. Its main disadvantage is that it does not provide synchronization decouple; as a result, the scalability of the model is limited [3].

3.1.5 Message Queuing

The message queue is the closest structure to a distributed Publisher/Subscriber system. The message queue does not have a specific architecture, but primarily involves a protocol of communication between producer and consumer. Many times such communication schemes are incorporated into Pub/Sub systems, forming a communication structure known as Message-Oriented Middleware (MOM). The main pillar of communication in such systems is the messages, hence the name [13], [14].

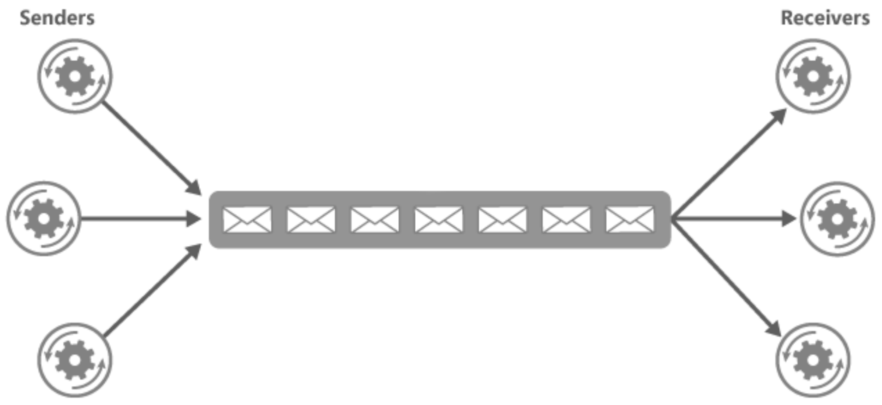


Figure 3.2: Example of simple Message Queue

Message queues are essentially places where messages published by producers are stored. Consumers receive the messages depending on the organization of the queue, i.e. either in the order in which they came (first in first out - FIFO), or in some other priority-based order provided by the queue. The system provides time and order guarantees to users. It is also important to note the key difference between a message queue and a Publisher/Subscriber system. As soon as a message is read/processed in the message queue, it is deleted from the queue and cannot be read by anyone else, unlike a Publisher/Subscriber system, where the messages are retained for a period of time.

Finally, the system provides decoupling in space and time. This is due to the fact that producers and consumers do not know who is on the other side and do not need to be connected at the same time [3].

3.1.6 Summary

The different communication models we have described so far have a lot in common. Typically, each new communication model proposed is either a variation of an older one, or a feature combination of older models. The purpose of each new model is to improve the pathogens of older systems. Thus, the Publisher/Subscriber system that we will study in this project has been influenced by all these models we have made reference to. Moreover, it has improved some of their points so as to achieve faster message transmission, easier to scaling, as well as adaptation to the needs of internet.

3.2 Publisher/Subscriber Systems

In the previous chapter we identified several types of communication systems. They all had advantages and disadvantages. In this chapter, we will further delve into what a Publisher/Subscriber system is. We will also take a look at some alternative subscription models. Finally, we will analyze the basic Publisher/Subscriber system Apache Kafka, that we used as a main tool in this project.

3.2.1 What is a Publisher/Subscriber system?

The Publish/Subscribe pattern, also known as Pub/Sub, is an architectural design pattern that provides a framework for exchanging messages between publishers and subscribers [3], [15], [16], [17].

The three main parts of such architectures are the publishers, the subscribers and the central system [15]. Let us take a closer look at what exactly each part of the system does.

- **Publisher:** publishers are applications that connect to the central system and publish events with a specific topic or content, depending on the system subscription model. In the modern age of the Cloud and the Internet of Things, these applications are made up of smaller, independent components that are easier to develop and maintain. Publishers do not know who is interested in the event they have published, or whether someone is connected to the other side to receive their message. They can also be connected and disconnected from the system without requiring any special procedure or burdening the system.
- **Subscriber:** subscribers, like publishers, are applications that also consist of smaller independent components. Their main job is to process the events published by the publishers. They log in to the system, indicate their interest in one (or more) topics and, when an event is published on that topic, they are notified and process it as soon as they receive it. Similarly to the publisher, the subscriber does not know who the message came from, nor does he know when the next message will come. His only job is to process the messages/events. The subscriber can essentially originate from a simple event processing application, to a large external system or database.
- **Event manager system:** a Publisher/Subscriber system mainly consists of a central messaging system. Publishers and Subscribers are applications connected to it. This central system accepts the events that are published and is responsible for transmitting to different parts of the system asynchronously. This transmission is done with the help of topics. A topic is a lightweight mechanism that resembles message queues, which helps to convey messages in the form of events, as well as connecting publishers and subscribers to the system. The difference between the topic mechanism and the message queue is that the topic mechanism pushes

events directly to subscribers, or with a slight delay. Additionally, when a message is read by an interested party it is not deleted, but remains there so that all interested parties on the same subject can receive it. Therefore, we can understand that the central system is, in essence, an event management system. In the literature we can also encounter it with this name [18].

The combination of these three parts provides us with a modern communication system that satisfies the needs of modern applications, communications, as well as the Internet.

3.2.2 Why a Pub/Sub system? Pros and Cons.

The architecture of the Publisher/Subscriber systems has some special features that made it stand out, compared to other messaging systems, and establish itself as one of the basic messaging schemes. These features that provides are the decoupling in time, space and synchronization [3]:

- **Time Decoupling:** the two sides do not need to be active at the same time. The publisher can publish something that will be read when a subscriber logs in, and the subscriber can be notified of a publish when the publisher is no longer logged in. This may be because the publisher logged out immediately after publishing, or because of the system settings, or because of an external application that processed and redirected the message back to the system.
- **Space Decoupling:** the interacting members are unaware of the existence of other members. As described above, the publishers publish the events through the event management system and, on the other hand, the subscribers process them. Publishers do not know how many and which subscribers process the events and, respectively, subscribers usually do not know which publisher the event is from, how many publishers post on the topic they are interested in, or when a new event will come.
- **Synchronization Decoupling:** publishers and subscribers are not blocked from producing or consuming new events. This is possible because these processes take place at the ends of the system and not in its main body.
- **Clean Design and Scalability** its design is simple and clean. This leads developers to design applications without taking into account the burden of communication. Its design also makes it easily scalable and customizable, as adding or removing system parts is a relatively easy and straight-forward process.

The features we have described make the system easily scalable. They also remove a large part of the implementation from the applications and, thus, make the system suitable for applications relying on document-centric communication and distributed environments such as the Cloud and the Internet of Things. Despite these advantages that make it a basic communication system of the modern era, there are some drawbacks that are worth noting:

- **Poor-consistency:** a Publisher/Subscriber system is definitely not suitable for critical systems such as those of military or financial nature due to the partial lack of guarantees in messaging. The possible loss of messages and the lack of guarantees for the delivery of messages make the system additionally unsuitable for applications that require accuracy and high consistency in the delivery of messages.
- **Increased Latency:** the intermediate level of communication (event management system) adds delay to communication time. Moreover, as publishers and consumers increase, it becomes more and more difficult for the system to provide time guarantees for the delivery of messages.
- **Security:** due to the loose nature of the system, it can become vulnerable to attacks. Intruders, i.e. malicious publishers, can invade the system and breach it. This can lead to bad messages being published and subscribers having access to messages that they should not normally receive [17].

We acknowledge that these downsides are due to the loose nature of the system in terms of time, space and synchronization. Though, again, we detect in practice that the advantages of a Publisher/Subscriber system have gained the trust of developers, making it, thus, a key pillar of communication of applications on the Internet. Of course, many systems, such as Kafka, eliminate these drawbacks with a variety of user-enabled features, sacrificing some of the advantages of the Publisher/Subscriber architecture during this process.

3.2.3 Subscription Models

The main members of a Publisher/Subscriber system are the publishers, the subscribers and, in particular, a central system that essentially constitutes an event management system. Subscribers connect to the system and express their interest in the events produced by the publishers. This interest can be expressed in many ways, depending on the architecture of the chosen system. Subscriptions architecture is, at heart, a way of categorizing events that are published, as well as showing the interest of subscribers. Depending on the subscription model, events are processed via the event management system and distributed within it. We will take a look at some of the most basic ones in the following sections.

3.2.3.1 Topic - Based Publisher/Subscriber

One of the first Publisher/Subscriber programs was based on the Topic concept, which is an extension of the group concept. According to the former, the interested parties—publishers and subscribers—express their respective interest for various specific topics, not necessarily one. The publishers, when they publish an event, identify the topic to which the event belongs; the subscribers, when connecting to the system, identify the topics that interest them in the meantime [15].

Topics, in essence, constitute an abstract tool to share the ”space” of the system with users. Each topic is a part of the system that is dedicated to its service. Of course, it is not the perfect solution. Unfortunately, although it can be a good solution for sharing the space of application or event processing by infrastructure, or the production and consumption of events by publishers and subscribers, it takes away much of the expressiveness of the users, either producers or consumers [2].

Various solutions have been proposed for this issue; one of them being the combination of the topic system with other systems, which we will cover below, such as content-based systems. Another solution is to add hierarchies to the organization of topics. This organization is usually executed by the developers. All modern topic-based systems offer this solution. In essence, we are talking about a tree-type structure, where each node is a topic; the higher the node is (on the structure/tree), the more general the topic it contains, is. Underneath this node exist topics that are perfectly related to the topic of the original node, though they are more specialized. When a subscription is made to a node in the hierarchy/tree, this then implies that there is interest in all the topics that exist under this node [3].

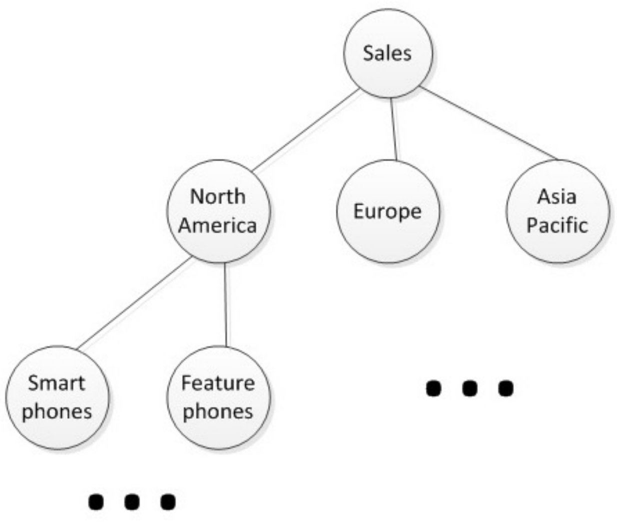


Figure 3.3: Topic hierarchy/tree model

We assume that a subscriber has shown interest in the topic ”North America”, as we see in the picture. This means that the subscriber automatically showed interest for the topics ”Smart Phones”, ”Feature Phones” and everything else included below that. Therefore, when a producer publishes an event for the topic of ”Smart Phones”, our subscriber will have expressed interest for this topic as well.

3.2.3.2 Content - Based Publisher/Subscriber

In practice, researchers have found that, even with some improvements proposed for the topic-based Publisher/Subscriber system, it was sometimes not enough to merely improve the limited expressiveness of such systems. This is why some variations of this model began to be proposed.

One of them was the content-based Publisher/Subscriber scheme. This format routes messages based on their content rather than a specific destination IP address, or a strictly predefined topic. That is, the properties/content of the message itself determine which subscriber will receive it.

Subscribers, when connecting to the system, can set some types of filters that can be applied to messages. These filters constitute data restrictions and determine which messages will reach subscribers. For example, such filters can be logical or numerical on data or message fields. In essence, the subscriber interest that goes through the filters is a set of topics, similar to those we saw in the previous section [1].

However, as was the case before, this scheme does not only have advantages. The application of the restrictions and filters we have mentioned, unfortunately, puts a brake on the response time of the system. The calculations needed at each node to find the path that the message will follow are complex and time-consuming. For this reason, such systems are not used in applications that require a very fast response or real-time response [3].

3.2.3.3 Type - Based Publisher/Subscriber

Through the use of Publisher/Subscriber systems, we have noticed that topic-based systems have events with similarities not only in content, but also in structure. This led to the creation of a new scheme for the Publisher/Subscriber systems oriented to the type of messages. By structure we mean the code structure given by the developers as they define the messages. This can be done using various techniques and abstractions provided by the respective language used by the developers for the application. We will provide an example to make it more understandable.

In the example 3.4, we observe a subscription based on the type of message, which is buying the car. We notice from the code that there are many different types in which a user/buyer may be interested. It seems that the subtyping offered by Java is a powerful tool for the user to express their interest for more than one type of message. Other languages, such as C and C++, offer similar capabilities.

It is becoming apparent that type-based events closely link the programming language to the messaging system. This can offer many advantages, such as the security provided by the language, in terms of types, perhaps in the speed of message processing, but also giving the necessary expressiveness to the user [3], [19].

```java
public class CarMarket {
    public String getCar(){...}
}

public class Car extends CarMarket {
    public String getCarName(){...}
    public String getCarBrand(){...}
    public float getCarPrize(){...}
}
public class CarForSale extends Car {
    public float getCarPrize(){...}
    public float getCarWarranty(){...}

}

public class CarForRent extends Car {
    public float getCarPrize(){...}
    public float getCarPetrolConsumption(){...}
}
```

```java
public class CarForSale extends Car {
    public float getCarPrize(){...}
    public float getCarKilometers(){...}
}

public class CarBuyer implements Buyer<CarForSale> {
    public Car buyCar(CarForSale c){
        if (c.getCarBrand() == 'Mercedes'){
            //...
        }
    }
}

Buyer<CarForSale> buy = new CarBuyer();
```

Figure 3.4: Type-based example in Java

3.2.3.4 Summary

In this section we referred to some of the most basic Publisher/Subscriber systems. As noted above, topic-based systems are static but very efficient, as they offer high speeds due to the short processing time they need, making them suitable for applications sensitive to response time. On the other hand, content-based systems can be more dynamic and with more expressive capability, though they add a lot of burden to the response time, as they need a lot of processing to get the messages routed properly. Finally, exploiting message types can increase the expressiveness of topic-based systems and reduce the time complexity of content-based systems.

Naturally, there are many variations and combinations, either on research or industrial level, and others are constantly being suggested. Thus, we understand that choosing the right system is not a cure-all. Everyone can customize or choose the system according to their needs.

3.2.4 Quality of Service - QoS

When looking at a Publisher/Subscriber messaging system, one of the key criteria for its suitability, especially when it comes to a large-scale industrial use system, is the quality of the services it offers. The end-to-end evaluation of QoS characteristics of a Publisher-/Subscriber system proves to be a challenge, as the loose connection of publishers and subscribers to the central system makes it difficult to define a single QoS measurement policy. In addition to the disconnection of the publisher and the subscriber from the system, a problem is the disconnection of system nodes from the central system due to its distributed nature. Consequently, we see that the non-deterministic behavior of the system introduces a lot of unreliability into the system, and its exact behavior is difficult to determine and measure. Nevertheless, we will attempt to introduce three key aspects of service quality: reliable messaging, on-time delivery and security [3], [14], [20].

3.2.4.1 Reliability

Reliability is one of the most important features of a distributed messaging system. It is imperative that strong guarantees exist for the sake of reliable data transfer within the system. Reliability is, substantially, a guarantee that subscribers will receive the published event. The processing of events that take place within the system, as well as the network hops that each data packet makes due to the distributed system architecture, adds an additional possible source of non-determinism, since each node of the system is likely be unavailable due to overload or general network uncertainty.

The persistence of events, their long stay in the system, their retransmission, and the existence of copies in the system, are techniques to reduce uncertainty and non-determinism. In general, the longer the event remains in the system, the lower the risk of losing it. Having copies can also be tricky, as it can overload the system and lead to adverse results. As a result, we understand that the balance is delicate. We conclude that the longer the message remains in the system, the less non-determinism decreases. Many systems, such as the one we are considering (Kafka), provide storage via files where we can establish their size and duration of stay. This technique completely eliminates the non-determinism of the system, as the message cannot be lost in the system.

3.2.4.2 On-Time Delivery

Many applications that use messaging systems are real-time and have strict requirements for controlling the delivery time of messages to consumers. Applications that require such restrictions have specialized Publisher/Subscriber systems and strictly defined environments that are relatively controlled. But to have such systems, there must be specialized point-to-point connections so that delays can be controlled. The connections and disconnections of publishers and subscribers should also be checked so that the network balance is maintained. These limitations, however, eliminate the advantages of the Publisher/Subscriber networks mentioned above.

3.2.4.3 Security

The security of a Publisher/Subscriber system is one of the most important issues faced by both the creators and the users of such systems.

The most obvious problem is giving the system access to authorized users. These users can belong to producers and consumers, making the issue even more complex. Due to the distributed architecture of such applications and their extent, usually, control over who connects and disconnects from the system ranges from difficult to almost impossible. This problem could be reduced by forcing users to use special security codes and keys, or for infrastructure nodes to run on dedicated servers where access from either producers or consumers is more precisely controlled. It is important to mention the aforementioned solutions merely reduce the problem; they do not eliminate it.

Another important problem that arises which may not be obvious, as the scale of the infrastructure is small but becomes quite obvious as it grows, is the trust between publisher and subscriber—the two ends of communication. On one hand, the subscriber wants to be certain of the authenticity of the message, as well as its content, i.e. that it will not contain malicious information or data that may have been altered. On the other hand, the system will want to receive data from trusted subscribers and have guarantees that the subscribers themselves will not try to negatively affect the system for their own benefit. Similarly, the system should make sure that publishers do not try to affect the system by overloading it, or modifying it in any other way. These problems stem in part from the uncertainty about who is connected to the system mentioned above.

We generally understand that the larger the system, the greater the number of nodes contained in the infrastructure. This fact, in addition to the uncertainty it adds for the secure transmission of the message, forces publishers and subscribers to rely on more nodes for the transmission of messages and subscriptions respectively. To be able to deal with this, it is very likely that we will sacrifice some of the advantages of the Publisher/Subscriber systems, as we noted in Section 3.2.2.

3.2.4.4 Summary

From what we have seen, we, therefore, understand that a Publisher/Subscriber system like Kafka should provide a consistent quality of service, especially when it comes to industrial use. Fluctuations in the performance of the data stream are not acceptable. The system should be able to maintain a steady rate of data transmission when the prevailing conditions do not change. It should also provide guarantees to users for the security of the transmission of messages.

3.3 Apache Kafka

After a careful look at some older messaging schemes, as well as several key points of a Publisher /Subscriber scheme, it is time to analyze and pay attention to the main parts of the Publisher/Subscriber system we used as a tool for this project, the Apache Kafka [15], [21].

3.3.1 Introduction to Kafka

Since the beginning of the internet, we have been writing and using programs that store information in Databases. Databases treat the world as objects that have a state, which is their information, stored in the system. This worked quite well for years, although, little by little, we realized that it is more efficient, instead of thinking of information as states of the world, to treat it as events.

Events are, basically, an indication in time of the state of the object. Though instead of storing events in databases, which is not efficient, we store them in logs. Logs practically refer to an ordered sequence of events. When an event occurs, it is written in one, or sometimes more, log. The structure we described above can very easily be scaled up and distributed to servers around the world. At heart, Apache Kafka is a system that manages these logs, but in Kafka's language the logs are called topics. Management includes many things such as:

- **publish** events in the system
- **subscribe** to events
- **store** them for as long as needed, either for a short period of time or permanently
- **process** events from various applications connected to the system
- **distribute** events correctly to users

Kafka is not a monolithic application. The topics it manages can be distributed across multiple servers around the world, depending on the needs of the user and the application. These servers can be physical machines, specialized hardware or even virtual machines. Developers can write many small standalone programs that interact with topics managed by Kafka, as well as consume or generate events from them. These programs can also perform functions on events such as real-time analysis, storage, editing, etc. All of this is done through an API provided by the system.

3.3.2 System Architecture

3.3.2.1 Main Parts of Kafka

Apache Kafka is a distributed messaging system with a topic-based architecture. We could say that the system is divided into 2 logical main parts, the **servers** that make up the system and do the basic work, and the **clients** that interact with the system. We note that the terminology is the same as that of the server-client system, the logic behind the process being quite similar since, in essence, the same type of interaction takes place. More specifically:

- The **servers** can be a cluster, small or large, depending on the needs of the users. These servers are, in simple terms, the backbone of the system. Some of them are called brokers and are responsible for maintaining the logs we mentioned before. Others are responsible for the communication of Kafka with external systems, such as another user system, a database etc.

- **Clients** are essentially programs and interfaces with which users can interact with Kafka. This communication can be achieved through distributed applications and microservices written by the developers.

The connection of these two parts gives us the messaging system called Kafka.

3.3.2.2 Abstractions and Terminology

In order to have a well-structured distributed system that can be scaled up and easily used by developers, the appropriate logical abstractions from its architecture must be made. If the system were complex and allowed many things to be implemented by the developer, its widespread use (i.e. Kafka), would range from being difficult to impossible. Thus, we present the most basic abstractions of Kafka, alongside how they are combined with each other to make the system function.

- **Events:** every interaction with the system takes the form of events. Every event is a registration in the system that indicates that ”something happened” in the outside world. The main parts of an event, which characterize its uniqueness, are a key that usually consists of a number, a value or information and the time stamp for when it took place.
- **Producers and Consumers:** producers are a type of client that we mentioned in the previous section, who interact with Kafka through events, like everyone else. In essence, producers are applications that publish events to the system. Consumers, on the other hand, subscribe to the system so that they have the opportunity to read the events that are published and interest them. As with most Publisher/Subscriber systems, Kafka publishers and subscribers are completely decoupled in time, space, and synchronization. In practice, the producers can log in and publish events on the system, and the consumers can read it whenever they want. Typically, both sides are completely unaware of the existence of the other side. Producers simply publish events on the system and do not care about whether they will be consumed; likewise, the consumers merely read and process events without concern on who published them.

- **Topics:** as we mentioned in the introduction, events are stored in logs which, in Kafka’s language, are called topics. Topics are essentially an ordered list of events. In essence, Kafka topics are a distributed file system. There is no limit to how many producers can publish on a topic, nor to how many consumers can subscribe to a topic. Even a zero number of producers and consumers is acceptable. Moreover, the processing of an event by a consumer does not delete the event from the topic, as would be the case with a classic message queue. Instead, events are retained in the topic until a specific amount of time—established by the system administrator—has elapsed, or they might forever exist in the topic.

- **Partitioning:** a topic can be distributed across multiple servers called brokers. This automatically means that the log we described as an unbroken log can consist of many smaller logs distributed across multiple servers. At heart, it constitutes a large file which is simply distributed. The chronological order of events within the log, when distributed, is maintained for each log separately and not for the entire log as a whole.

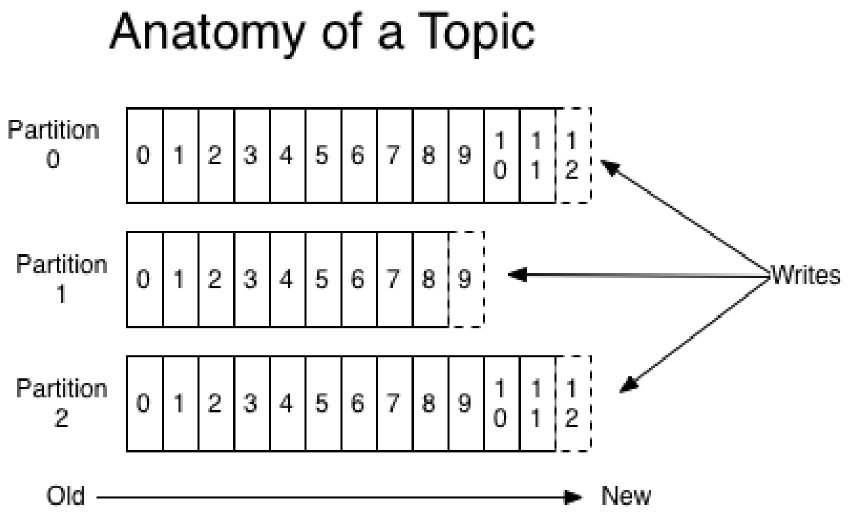


Figure 3.5: Multiple partitions of a Topic

  The topic partition we have described is one of the most important features of Kafka. This distribution of data/events is a key factor in scaling the system, as both publishers and subscribers can interact with multiple brokers at the same time. The events that are published and have the same key are written in the same partition, i.e. in the same broker, and Kafka guarantees that those who read from this partition will read the events exactly in the order in which they were published.

- **Replication:** errors are very likely to occur at messaging systems and particularly in distributed systems such as Kafka. These errors usually lead to data loss. To keep the data safe and the system fault-tolerant, Kafka creates copies/replications of the data in multiple brokers spread across the system, or even in databases. A common replication factor is 3, which denotes that there are 3 copies of the data in the system.

- **Broker:** a Kafka broker, a Kafka server and a Kafka node all refer to the same concept and are synonyms. A Kafka broker is a server that hosts multiple topics. Kafka can combine many brokers by transferring data between them and create a cluster, with the help of Zookeeper. This cluster hosts various partitions of one topic or more.

  A Kafka notice receives messages from publishers on various topics and saves them on disk with a unique offset. It respectively allows subscribers to receive these messages depending on the topic and partition they are interested in. In essence, as

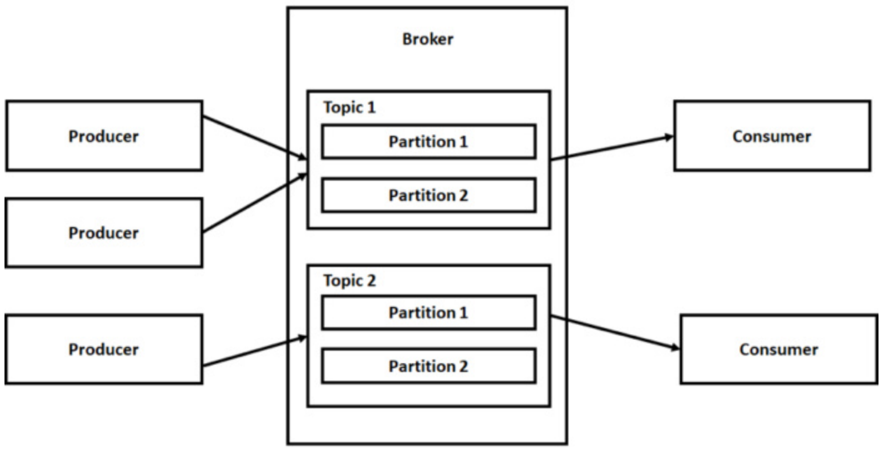


Figure 3.6: Kafka broker with multiple publishers and subscribers

in the real sense of the broker, a Kafka broker is the mechanism that mediates the communication between publisher and subscriber.

3.3.3 Zookeeper

3.3.3.1 What is it and how does it work?

Zookeeper is software developed by Apache and is used to configure and synchronize distributed systems such as Kafka. Zookeeper watches the status of Kafka's brokers, topics and partitions, etc. It is, practically, the administrator who allows the simultaneous readings and subscriptions of the publishers and subscribers of the system. The Zookeeper Atomic Broadcast(ZAB) protocol is the actual mechanism that allows the software to control the entire system [22].

The data within Zookeeper is split into several nodes to achieve the consistency required by the system. In case one node fails, Zookeeper automatically selects another, in real time, so communication is not disrupted .

3.3.3.2 Zookeeper and Apache Kafka

Having overall observed what Zookeeper is and how it works, let us proceed to how it combines with Kafka to give us the ultimate reliable and flexible system, and what tasks it takes on.

- **Controller-Leader Election:** the controller is one of Kafka's key broker entities and is responsible for maintaining the leader-follower relationship for all partitions on each topic. Each partition has a broker that is the partition leader and is the one who

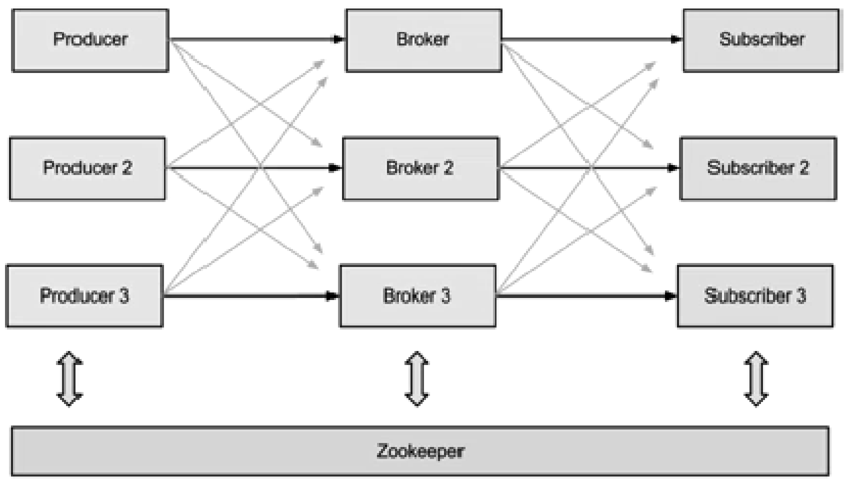


Figure 3.7: Zookeeper and Kafka interaction scheme.

manages all the read and write requests for the partition, while the follower passively reproduces the leader. If, for any reason, a node is terminated, the controller-leader is responsible for telling the followers to act as leaders in order to perform the tasks of the node that terminated, until it elects someone else and ensures that there is only one controller node.

- **Topic configuration:** Zookeeper manages the entire list of topics and everything related to it. It deals with the partitions of each topic and in which broker they are hosted; it also knows where to find the replicas of the data, which node is the most suitable leader for each topic, etc.
- **Members of cluster:** Zookeeper also knows which brokers the system consists of so that, in case of a broker shutdown, it can maintain the functionality of the system and maintain its performance.

In closing, we consider it mandatory to mention that Zookeeper is an integral tool for the operation of Kafka as, in order to be able to perform any Kafka service, we must first run a Zookeeper server [23].

4. DATA GATHERING

4.1 The importance of data

Machine Learning data analysis uses algorithms to continuously improve itself over time, though quality data is necessary for these models to operate efficiently. A single row of dataset is called an instance. Datasets are a collection of instances that all share a common attribute. Machine learning models will generally contain several different datasets, each used to fulfill various roles in the system.

For machine learning models to understand how to perform various actions, training datasets must first be fed into the machine learning algorithm, followed by validation datasets (or testing datasets) to ensure that the model is interpreting this data accurately. Once we feed these training and validation (or test) sets into the system, subsequent datasets can then be used to improve one's machine learning model going forward. The more data we feed into the ML system, the faster that model can learn and improve [24], [25].

4.2 Kafka setup and data gathering procedure

As in most Machine Learning systems, the main problem we faced in this thesis was data collection. It was one of the most important aspects of this project, as well as one of the most time-consuming ones.

We first had to understand how Kafka works in order to be able to use it. The next step was to see how we could set up the system to test it and do the necessary experiments. We initially experimented to seek an efficient and correct way to set up the system. Through tests, we discovered various Kafka devices that were sufficient to perform experiments and take representative measurements. Each time, the cluster served only 1 topic and consisted of 1, 2 or 5 brokers/servers that represented the replication factor of the topic. Thus, for each broker (1, 2, or 5) and topic, we attempted different partitions of the topic (1, 2, or 5) in order to eventually obtain a wide range of measurements.

4.3 Parameters tested/measured

In order to assess how the system reacts to different configurations for different parameter values and, then, to be able to predict its performance for different parameter values, we had to test many parameters. Hence, the setup we discussed in the previous section remained the same in every parameter we checked so as to achieve objective results.

In this work we focused primarily on the producer; particularly, how the parameters of the producer affect the system [26]. As we will mention at the end, future work or an extension of this work could be the study of how the parameters of brokers or consumers can also affect performance.

To that end, the parameters we chose to change and test how they affect system performance include [27], [28]:

Table 4.1: Producer parameters tested

Parameter	Values
message size	[10, 20, 50, 100, 200, 500, 1000]
batch.size	[16384, 50000, 100000, 200000, 500000]
linger.ms	[0, 1, 2, 5, 10]
max.request.size	[500000, 1048576, 2000000, 5000000, 10000000]
buffer.memory	[10000000, 33554432, 100000000, 200000000]
acks	[-1, 0, 1]

It is necessary, however, for better understanding of the work, to see what each parameter essentially represents. This way, we can better understand how their changes affect performance. The parameters of the producers, therefore, regard:

- **message size**: This parameter is one of the most important parameters of the system. It involves the size of the messages/data that the producer will send. Of course, the server has its own parameter, which also limits the size of the messages and can differ from that of the producer.

  The parameter type is integer and has a range of [1, . . .].

- **batch.size**: The producer will attempt to batch records together into fewer requests whenever multiple records are being sent to the same partition. This helps performance on both the client and the server. This configuration controls the default batch size in bytes.

  No attempt will be made to batch records larger than this size.

  Requests sent to brokers will contain multiple batches, one for each partition with data available to be sent.

  A small batch size will make batching less common and may reduce throughput (a batch size of zero will disable batching entirely). A very large batch size may use memory a bit more wastefully, as we will always allocate a buffer of the specified batch size in anticipation of additional records.

  The parameter type is integer; it has a default value of 16384 bytes and a range of [0, . . .].

- **linger.ms**: The producer groups together any records that arrive in between request transmissions into a single batched request. Normally, this occurs only under load when records arrive faster than they can be sent out. However, in some circumstances the client may want to reduce the number of requests even under moderate load. This setting accomplishes this by adding a small amount of artificial delay;

that is, rather than immediately sending out a record, the producer will wait for up to the given delay to allow other records to be sent, so that the sends can be batched together. This setting gives the upper bound on the delay for batching: once we get *batch.size* worth of records for a partition, it will be sent immediately regardless of this setting; however, if we have fewer than this many bytes accumulated for this partition, we will *linger* for the specified time waiting for more records to show up. Setting linger.ms=5, for example, would have the effect of reducing the number of requests sent but would add up to 5ms of latency to records sent in the absence of load.

The parameter type is long, has a default value of 0 milliseconds and a range of [0, . . .]. The default value 0 means no delay.

- **max.request.size**: The maximum size of a request in bytes. This setting will limit the number of record batches the producer will send in a single request to avoid sending huge requests. This is also effectively a cap on the maximum uncompressed record batch size. Note that the server has its own cap on the record batch size (after compression if compression is enabled), which may be different from max.request.size.

  The parameter type is integer, has a default value of 1048576 bytes and a range of [0, . . .].

- **buffer.memory**: The total bytes of memory the producer can use to buffer records waiting to be sent to the server. If records are sent faster than they can be delivered to the server, the producer will block for *max.block.ms*, after which it will throw an exception.

  This setting should correspond roughly to the total memory the producer will use, but is not a hard bound since not all memory the producer uses is used for buffering. Some additional memory will be used for compression (if compression is enabled), as well as for maintaining in-flight requests.

- **acks**: The number of acknowledgments the producer requires the leader to have received before considering a request complete. This controls the durability of records that are sent. The following settings are allowed:

  - *acks=0*: If set to zero, then the producer will not wait for any acknowledgment from the server at all. The record will be immediately added to the socket buffer and considered sent. No guarantee can be made that the server has received the record in this case, and the retries configuration will not take effect (as the client won't generally know of any failures). The offset given back for each record will always be set to -1.
  - *acks=1*: This will mean the leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers. In this case, should the leader fail immediately after acknowledging the record but before the followers have replicated it, then the record will be lost.

- *acks=all*: This means the leader will wait for the full set of in-sync replicas to acknowledge the record. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive. This is the strongest available guarantee. This is equivalent to the acks=-1 setting.

The parameter type is string, has a default value of 1 and a range of [all, -1, 0, 1].

We initially tested these parameters one by one to examine how the system is affected by each parameter. Then, we combined variables in pairs to see how the system is affected if we change two variables at the same time. That is, for each variable and pair of variables, we repeated the measurements as described in Section 4.2. Furthermore, the variable acks was so important that we were compelled to include it in all the measurements. In other words, for each measurement, whether a separate variable or a combination thereof, we repeated the process for all three values of the variable acks (-1, 0, 1) and recorded them in the same file to facilitate comparisons. Consequently, the variables and their pairs we measured include:

Table 4.2: Parameters and combinations tested

1 Parameter	2 Parameters
message.size	batch.size + buffer.memory
batch.size	batch.size + linger.ms
linger.ms	batch.size + max.request.size
max.request.size	buffer.memory + linger.ms
buffer.memory	linger.ms + max.request.size

Each entry in the table is a separate file with measurements made for each parameter listed in the table.

The measurements of these parameters, essentially, show us how the system reacts to the different values of the parameters, or even by combining these values. By system reaction, we mean how its performance is affected. So with these measurements as a guide, we will try to make ML models, which by taking values for these parameters, as training data, will try to predict what will be the possible performance of the system. These forecast will cover four specific targets. The first two targets will be ”Records/sec” and ”MB/sec”, which show us the performance of the system in terms of the amount of information it can transmit, and the other two will be the ”Avg Latency” and ”Max Latency”, that show us the delay in data transmission.

4.4 Procedure and code

We believe it is necessary to offer a few words on the code with which the measurements were created. Kafka itself provides shell scripts intended for brokers and the zookeeper server [29]. We also utilized the script provided by Kafka to test the performance of the producer. This generates random messages/data and sends them into the system according to the settings of producers, consumers and brokers.

The procedure we followed to carry out the experiments, as well as a sample of the commands we used, are presented as such:

- **Start Zookeeper Server**: We must first start the Zookeeper server. If Zookeeper does not work, neither will Kafka, as the former is a necessary element of the latter's architecture.

```
 # Starting server
 $ bin/zookeeper-server-start.sh config/zookeeper.properties

 # Killing server
 $ bin/zookeeper-server-stop.sh
```

- **Start Kafka Server**: After starting Zookeeper, we can start as many Kafka brokers/servers we want:

```
 # Starting broker
 # server.properties is the properties file of the server
 $ bin/kafka-server-start.sh server.properties
```

- **Topic handling**: These commands are for topic management:

```
 # Create Topic
 $ bin/kafka-topics.sh --create zookeeper localhost:2181 --replication
-factor {num} --partitions {num} --topic {my-topic}

 # Delete Topic
 $  bin/kafka-topics.sh --delete --zookeeper localhost:2181 --topic {
my-topic}

 # Describe Topic
 $ ./bin/kafka-topics.sh --describe --zookeeper localhost:2181 {
Optional: --topic {my-topic}}

 # Listing Topic
 $ bin/kafka-topics.sh --list --zookeeper localhost:2181 {Optional: --
topic {my-topic}}
```

- **Producer Performance Test**: This command is the script provided by Kafka to test the performance of the producer. It produces statistics for every X bytes it sends and, at the end, it calculates the average of statistics:

```
 # producer.properties is the properties file of the producer
 $ bin/kafka-producer-perf-test.sh --topic {my-topic} --num-records {
num} --record-size {num} --throughput {-1, 0, 1} --producer.config
producer.properties

```

```
 # Example: (The last row is the average statistics and the others rows are the statistics for every batch)

 $ bin/kafka-producer-perf-test.sh --topic test-topic --num-records 5000000 --record-size 100 --throughput 0 --producer.config producer.properties

 $ 3289742 records sent, 657948.4 records/sec (62.75 MB/sec), 280.0 ms avg latency, 590.0 ms max latency.
 $ 5000000 records sent, 781860.828772 records/sec (74.56 MB/sec), 184.33 ms avg latency, 590.00 ms max latency, 1 ms 50th, 543 ms 95th, 575 ms 99th, 588 ms 99.9th.
```

Additionally, to add to the automation of the process, we formulated some python scripts in order to automatically create the commands, transfer data from .txt file to excel file and edit them. To make the measurements more accurate, we repeat the same measurement 3 times (i.e. with the same configurations) and then we calculate their average with a Python script.

4.5 Machine specs

For the measurements we utilized the server provided by my supervisor, Mr. Doulas. Every broker of the system, as well as the zookeeper server, were running in a different virtual machine on the server. The technical specs of the server were:

Table 4.3: Machine specs

Part	Value
CPU	Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz
Architecture	x86_64
Max Boost	4GHz
Cores/Threads	4/8
L1d	32K
L1i	32K
L2	256K
L3	8MB
RAM	32GB
RAM speed	2400 MHz
RAM type	DDR4
Disk 1 type	SSD
Disk 1 capacity	256GB
Disk 2 type	HDD
Disk 2 capacity	1TB
Network speed	1Gbit/s
OS	Ubuntu 18.04.5 LTS
Kernel version	GNU/Linux 4.15.0-101-generic

5. REGRESSION ALGORITHMS

Regression algorithms belong to the family of supervised Machine Learning algorithms, which is a subset of Machine Learning algorithms. One of the main features of supervised learning algorithms is that they model dependencies and relationships between the target output and input attributes to predict the value for new data. The purpose of regression is to predict the value of one or more continuous target variables t when we are given the value of a D-dimensional vector x of input variables. The methodology regards the algorithm that builds a model based on the characteristics of training data and uses that model to predict the value for new data [30], [31].

For a given set of learning N elements observations x_n, where $n = 1, 2, \ldots, N$ together with the corresponding target values t_n, our aim will be to predict the value of t for a new value of x. In the simplest approach, this can be done by constructing a suitable function $Y(x)$, whose values for new entries are predictions for the corresponding values of t. In general, from a probabilistic point of view, we aim to model the predictive distribution $p(t|x)$, as it expresses our uncertainty about the value of t for each value of x. From this conditional distribution we can make predictions t, for each value of x, in such a way as to minimize the expected value of the loss function [32].

In this chapter, therefore, we will examine some basic regression models that we used in this thesis to predict new values for Kafka.

5.1 Linear Regression

Linear regression is likely one of the most important and widely used regression techniques. It is also one of the simplest methods of regression. One of its main advantages is the ease of interpreting the results.

When we implement the linear regression algorithm of a partially dependent variable y, the set of independent variables $x = (x_1, x_2, \ldots, x_r)$, where r is the number of predictors, we consider a linear relationship between the unknown variable y and the variables x. This relationship is described by the regression equation: $y = b_0 + b_1x_1 + \ldots + b_rx_r + \epsilon$. The parts that make up the equation are:

- $b_0, b_1, \ldots, b_r$: the regression coefficients
- $x_1, x_2 \ldots, x_r$: the independent variables x
- ϵ: the random error or noise that is added so that we do not have overfitting

The regression coefficients together with the independent variables give us a linear combination that is the value of the predicted variable y. The noise or random error ϵ is added to the predicted value to counterbalance the possibility of overfitting the data. Sometimes ϵ is omitted.

Linear regression calculates the estimators of the regression coefficients $b_0, b_1, \ldots, b_r$ which, in essence, are the predicted weights of the independent variables $x_1, x_2, \ldots, x_r$. Thus, we can derive a regression function that records the dependence between inputs and outputs: $f(x) = b_0 + b_1x_1 + \ldots + b_rx_r$. The estimated or predicted response, $f(x_i)$, for each observation $i = 1, 2, \ldots, n$, must be as close as possible to the corresponding actual response y_i. The differences $y_i - f(x_i)$ for all observations $i = 1, 2, \ldots, n$, are called residuals. Regression refers to the determination of the optimal predicted weights, i.e. the weights corresponding to the smallest residuals [33], [34], [35].

5.1.1 Linear Regression Performance

To get the best weights, we usually minimize the sum of squared residuals (SSR) for all observations $i = 1, 2, \ldots, n$:

$$SSR = \sum_{i=1}^{n}(y_i - f(x_i))^2 \tag{5.1}$$

This approach is called the method of ordinary least squares.

The coefficient of determination, denoted by R^2, tells us how much of the variance in y can be explained by the dependence of the values on x, for each linear regression model. But first, let us consider how the formula of R^2 emerges:

- The average value of the observed data:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{5.2}$$

- The total sum of squares (proportional to the variance of the data):

$$SST = \sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{5.3}$$

Through the combination of equations (5.1) and (5.3), the final formula of R^2 is:

$$R^2 = 1 - \frac{SSR}{SST}$$

Larger R^2 indicates a better fit and denotes that the model can better explain the variation of the output with different inputs. The value $R^2 = 1$ corresponds to $SSR = 0$—which is the perfect fit—since the values of predicted and actual responses fit completely to each other. An absolute fit, however, may indicate overfitting of the model on the data, which can cause large generalization error [34].

5.1.2 Multiple Linear Regression

Multiple linear regression is a case of linear regression with two or more independent variables. It is the generalized case of linear regression and is what we discussed in the section 5.1. The simple case of regression is the same but with only one variable x_1. If there are n independent variables, the estimated regression function is $f(x_1, x_2, \ldots, x_n) = b_0 + b_1x_1 + \ldots b_nx_n$. It represents a regression plane in a (n+1)-dimensional space. The goal of regression is to determine the values of the weights $b_0, b_1, \ldots, b_n$ such that this plane is as close as possible to the actual responses and yields the minimal SSR [34], [33].

5.2 Lasso Regression

Lasso regression, or Least Absolute Shrinkage and Selection Operator, is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean value. The Lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This particular type of regression is well-suited for models showing high levels of multicollinearity or when one wants to automate certain parts of model selection, such as variable selection/parameter elimination. Multicollinearity generally occurs when there are high correlations between two or more predictor variables. In other words, one predictor variable can be used to predict the other. This creates redundant information, skewing the results in a regression model [36].

5.2.1 L1 Regularization

Regularization is a way to avoid overfitting by penalizing high-valued regression coefficients. In simple terms, it reduces parameters and shrinks (simplifies) the model. This more ”compact” model will likely perform better at predictions. Regularization adds penalties to more complex models. The less overfitted the model is, the more likely it is to have very good predictive power.

This principle is called the ”*Principle of Economy*” or ”*Principle of Simplicity*” and is known as the Ockham’s razor. Ockham had claimed that ”*No one should make more guesses than necessary*” or, alternatively, ”*When one has to choose one of two models with identical predictions, the simplest is chosen*” [37].

Lasso regression performs L1 regularization, which adds a penalty equal to the absolute value of the magnitude of coefficients. This type of regularization can result in sparse models with few coefficients. Some coefficients can reach zero and, as such, be eliminated from the model. Larger penalties result in coefficient values closer to zero, which is the ideal for producing simpler models. On the other hand, L2 regularization (used in Ridge regression) does not result in elimination of coefficients or sparse models. This makes the Lasso far easier to interpret than the Ridge [36], [38].

5.2.2 Performing Lasso Regression

When performing the Lasso Regression algorithm, the goal is to minimize that quantity:

$$\min \left\{ \sum_{i=1}^{n} (y_i - b_0 - \sum_{j=1}^{p} x_{ij} b_j)^2 - \lambda \sum_{j=1}^{p} |b_j| \right\} \tag{5.4}$$

which is the same as minimizing the sum of squares with constraint $\sum_{j=1}^{p} |b_j| \leq s$. The tuning parameter λ controls the strength of the L1 penalty. λ is basically the amount of shrinkage. When λ = 0, no parameters are eliminated. The estimate is equal with Linear Regression. As λ increases, more and more coefficients are set to zero and eliminated. Theoretically, when λ = ∞, all coefficients are eliminated. In addition, bias increases, where bias is the tendency of a statistic to overestimate or underestimate a parameter. Last, but not least, when λ decreases, variance increases, with variance being a way of measuring how far a data set is spread out. It is mathematically defined as the average of the squared differences from the mean [36], [38], [39], [40].

5.3 LassoLARS Regression

Least-Angle Regression (LARS) is a regression algorithm for high-dimensional data, developed by Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani. LARS is similar to forward stepwise regression [41].

The LARS procedure works roughly as follows. As with classic Forward Selection, we start with all coefficients equal to zero, and find the predictor most correlated with the response, say x_{j1}. We take the largest step possible in the direction of this predictor until some other predictor, say x_{j2}, has as much correlation with the current residual. At this point, LARS separated by Forward Selection. Instead of continuing along x_{j1}, LARS proceeds in a direction equiangular between the two predictors until a third variable x_{j3} earns its way into the “most correlated” set. LARS then proceeds equiangularly between x_{j1}, x_{j2} and x_{j3}, that is, along the “least angle direction”, until a fourth variable enters, and so on [42], [43].

In simpler terms and without mathematics, it could be described as follows: at each step, it finds the feature most correlated with the target. When multiple features are having equal correlation, instead of continuing along the same feature, it proceeds in an equiangular direction between the features.

5.3.1 Pros and Cons of LARS algorithm

The LARS algorithm features many advantages, some of which we will list below [41]:

- It is numerically efficient in contexts where the number of features is significantly greater than the number of samples.
- It is computationally just as fast as forward selection and has the same order of complexity as ordinary least squares.
- It produces a full piecewise linear solution path, which is useful in cross-validation or similar attempts to tune the model.
- If two features are almost equally correlated with the target, then their coefficients should increase at approximately the same rate. The algorithm, thus, behaves as intuition would expect; additionally, it is more stable.
- It is easily modified to produce solutions for other estimators, like the Lasso.

The algorithm, of course,also has a drawback [41]:

- Because LARS is based upon an iterative refitting of the residuals, it would appear to be especially sensitive to the effects of noise.

5.3.2 Mathematical Formulation

LassoLARS is a Lasso model implemented using the LARS algorithm and, unlike the implementation based on coordinate descent, this yields the exact solution, which is piecewise linear as a function of the norm of its coefficients.

The algorithm is similar to forward stepwise regression, though instead of including features at each step, the estimated coefficients are increased in a direction equiangular to each one's correlations with the residual. Instead of giving a vector result, the LARS solution consists of a curve denoting the solution for each value of the l_1 norm of the parameter vector [42], [44].

Finally, the optimization objective function of LassoLARS is:

$$\frac{1}{2 \cdot n_{samples}}||y - Xw||_2^2 + \lambda||w||_1 \tag{5.5}$$

5.4 CART Regression

Classification and Regression Trees—or CART, for short—is a term to refer to Decision Tree algorithms that can be used for classification or regression predictive modeling problems.

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while, at the same time, an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches, each representing values for the attribute tested. Leaf node represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data [45], [46].

5.4.1 Pros and Cons of CART algorithm

To see whether this algorithm is worth utilizing or not, we have to look at the advantages and disadvantages it offers [44]. Some advantages of decision trees include:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalization; dummy variables need to be created and blank values to be removed.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

On the other hand, there are disadvantages that do not allow us to use this algorithm in every problem. Some of them are:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node, or setting the maximum depth of the tree, are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality, and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm, where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is, therefore, recommended to balance the dataset prior to fitting with the decision tree.

These are the pros and cons of decision trees. In our case, we have to compare them each time to observe whether the algorithm is suitable for our type of problem, as well as for our data.

6. CLASSIFICATION ALGORITHMS

In Machine Learning and Statistics, Classification is a supervised learning technique, just like the Regression we examined earlier. Classification is the process by which a computer program recognizes which category (from a set of known categories) a new observation belongs to. This decision is based on a set of training data containing other observations whose categories are known. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance. One of the best known examples of classification is the Iris Flower dataset [47] which includes 3 flower categories (Iris Virginica, Iris Versicolor, Iris Setosa) and new ones are categorized according to their four characteristics:

- Leaf length of calyx flower in centimeters (cm)
- Leaf width of calyx in centimeters (cm)
- Horseshoe length in centimeters (cm)
- Horseshoe width in centimeters (cm)

We will analyze the problem of classification from a more mathematical point of view. The goal in classification is to take an input vector x and assign it to K distinct classes C_k where $k = 1, 2, \dots, K$. In the most common case, classes are considered unrelated, therefore each entry is assigned to a single class. The entry area is therefore divided into $decision\ areas$ whose boundaries are called $decision\ boundaries$ or $decision\ surfaces$.

In comparison to the regression methods, the target variable t was simply the vector of real numbers whose value we intended to calculate. In the case of classification, we can interpret the value of the target variable t_k as the input vector x belongs to the class C_k [33].

After having gained an understanding of the key points of classification, in this chapter we will take a look at some techniques, as well as analyze their effectiveness on our data. We will also observe how we can tackle the problem of continuous data classification.

6.1 K-NN Classification

6.1.1 K-NN algorithm

$K - NearestNeighbors$ is a neighbors-based classification. This type of classification is an instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned to the data class which has the most representatives within the nearest neighbors of the point [48].

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point [49].

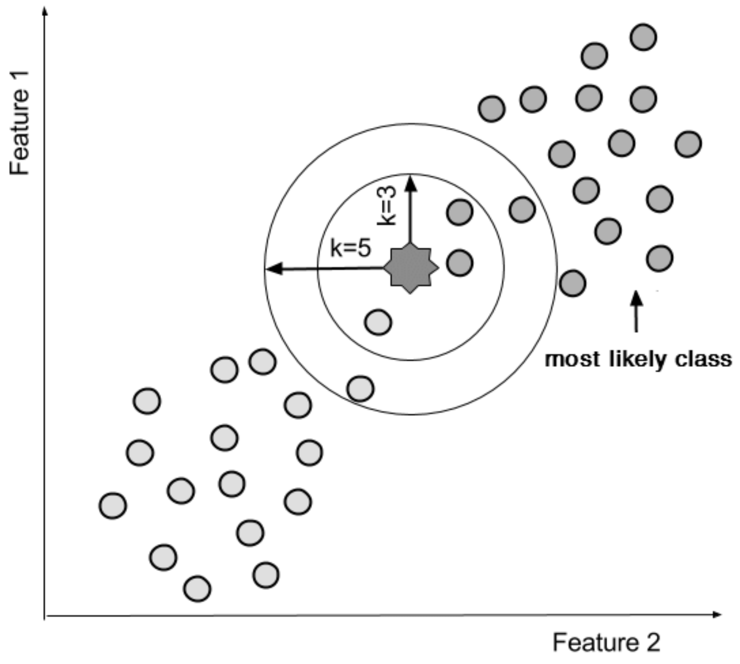


Figure 6.1: K-NN classification example

As we can note from the picture, where the new data will be classified depends on how many neighbors are close to it from each class. Of course, we observe that as k changes, the type/class of neighbors close to k can also change. From this we understand that the correct choice of the hyperparameter, is key.

6.1.2 Parameter selection

The best choice of k depends on the data. Overall, larger values of k reduce effect of the noise on the classification, but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques. The special case where the class is predicted to be the class of the closest training sample (i.e. when k = 1), is known as the nearest neighbor algorithm.

The accuracy of the k-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or when the feature scales are not consistent with their importance. Much research effort has been put into selecting or scaling features to improve the classification. A particularly popular approach is the use of evolutionary algorithms to optimize feature scaling. Another popular approach is to scale features according to the mutual information of the training data with the training classes.

It is easy, therefore, to grasp the importance of the hyperparameter k for the performance of the algorithm. It becomes apparent that the performance and number of neighbors the user needs to define, is directly related to the type of problem, as well as the data. Another parameter that can affect the performance of the algorithm through the number of neighbors is the metric function we use to measure the distance of new unclassified data from neighbors. The best choice of k can be made using techniques such as cross-validation or GridSearch, which is essentially a form of exhaustive search.

6.1.3 Distance functions

To calculate distances, three distance metrics [50] that are often used are:

- **Euclidean Distance**: Euclidean distance is one of the most commonly used distance metric. It is calculated using Minkowski Distance formula (see below) by setting hyperparameter p's value to 2. This will update the distance d formula as follows:

$$d_{euclidean} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{6.1}$$

  Euclidean distance formula can be used to calculate the distance between two data points in a plane.

- **Manhattan Distance**: We use Manhattan Distance when we need to calculate the distance between two data points in a grid-like path. Distance d will be calculated using an absolute sum of difference between its Cartesian co-ordinates, as indicated below:

$$d_{manhattan} = \sum_{i=1}^{n} |x_i - y_i| \tag{6.2}$$

  where, n-number of variables, x_i and y_i are the variables of vectors x and y, respectively, in the two dimensional vector space. i.e. $x = (x_1, x_2, x_3, \ldots)$ and $y = (y_1, y_2, y_3, \ldots)$. Manhattan distance is also known as Taxicab Geometry, City Block Distance, etc.

- **Minkowski Distance**: Minkowski distance is a metric in Normed vector space, on which a norm is defined. The distance can be calculated using the formula below:

$$d_{minkowski} = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}} \tag{6.3}$$

  Minkowski distance is the generalized distance metric. In this context, generalized means that we can manipulate the above formula to calculate the distance between two data points in various ways. We can manipulate the value of *hyperparameter p* and calculate the distance in three different ways:

1. p = 1, Manhattan Distance
2. p = 2, Euclidean Distance
3. p = ∞, Chebychev Distance

6.2 SVM Classification

A Support Vector Machine (SVM) is a supervised Machine Learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features we have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by constructing a hyper-plane or set of hyper-planes in a high or infinite dimensional space. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin) since, in general, the larger the margin, the lower the generalization error of the classifier [51]. The figure below shows an example of a decision function:

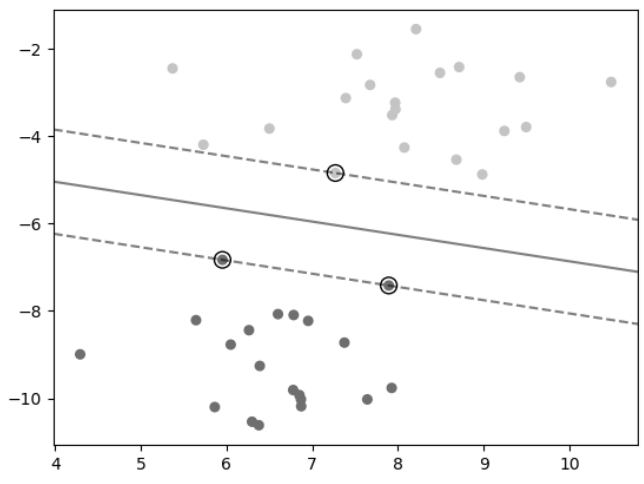


Figure 6.2: SVM classification example

6.2.0.1 Large-margin hyperplane

SVM is known as a large margin classifier. The distance between the line and the closest data points is referred to as the margin. The best or optimal line that can separate the two classes is the line that has the largest margin. This is called the large-margin hyperplane.

The margin is calculated as the perpendicular distance from the line to only the closest points.

6.2.0.2 Support Vectors

As the margin is calculated by taking into account only specific data points, support vectors are data points that are closer to the hyperplane and influence its position and orientation. By using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. They are the points that help us build our SVM.

In other words, support vectors are imaginary or real data points that are considered landmark points in order to determine the shape and orientation of the margin. The objective of the SVM is to find the optimal separating hyperplane that maximizes the margin of the training data [49], [51].

6.2.1 Linear SVM Classification

According to the previous section, the linear SVM is the basic case of the SVM classifier according to which we find the maximum-margin hyperplane and classify our data according to the side of the hyperplane they are on. As such, in this section we will explore the basic mathematical concepts of linear SVM in more detail.

Our training dataset consists of n points of the form: $(\overrightarrow{x_1}, y_1), (\overrightarrow{x_2}, y_2), \ldots, (\overrightarrow{x_n}, y_n)$ and the y_i is either 1 or -1, depending on the class to which $\overrightarrow{x_i}$ belongs. Each $\overrightarrow{x_i}$ is a p-dimensional real vector. As mentioned earlier, our purpose is to find the ”maximum-margin hyperplane” that separates the dataset of $\overrightarrow{x_i}$ that have $y_i = 1$, from $\overrightarrow{x_i}$ that have $y_i = -1$, and to maximize the distance of each class from the hyperplane.

Each hyperplane can be written as a set of $\overrightarrow{x}$ points that satisfy the relation:

$$\overrightarrow{w} \cdot \overrightarrow{x} - b = 0 \tag{6.4}$$

where $\overrightarrow{w}$ is the normal vector to the hyperplane. The parameter $\frac{b}{||\overrightarrow{w}||}$ determines the offset of the hyperplane from the origin along the normal vector $\overrightarrow{w}$ [49], [51].

As we can see in the picture, an area is created around the hyperplane. What the algorithm practically tries to do is maximize this area. Unfortunately, however, this is not always easy; as a result, there are 2 subcategories of the algorithm that we believe are worth mentioning:

- **Hard-margin**: Essentially, it describes the aforementioned algorithm, according to which the data can be perfectly separated into two classes and, consequently, we attempt to maximize the distance between them. In this case, the maximum-margin hyperplane is that which lies halfway between the two.

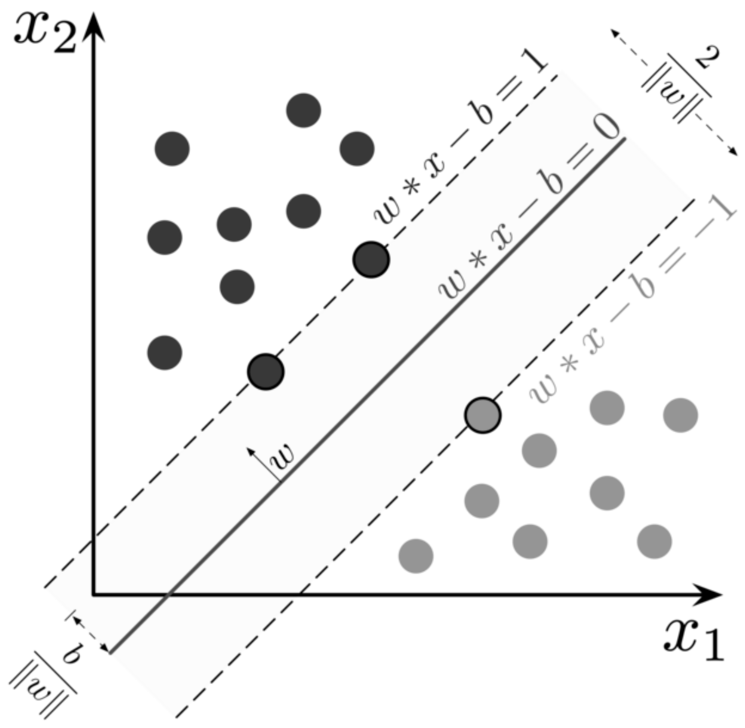


Figure 6.3: Linear SVM maximum-margin hyperplane

Geometrically, the distance between these two hyperplanes is $\frac{2}{\vec{w}}$, so, to maximize the distance between the planes, we aim to minimize $\vec{w}$. The distance is computed using the distance from a point to a plane equation. We also have to prevent data points from falling into the margin, and thus we add the following constraint for every point:

- $\vec{w} \cdot \vec{x} - b \geq 1$, if $y_i = 1$,
- $\vec{w} \cdot \vec{x} - b \leq -1$, if $y_i = -1$

These constraints state that each data point must lie on the correct side of the margin. An important consequence of this geometric description, is that the max-margin hyperplane is completely determined by those $\vec{x_i}$ that lie nearest to it. These $\vec{x_i}$ are called support vectors, as mentioned before [49].

- **Soft-margin**: This case is a variant of SVM, in which the data is not completely linearly separable. A key role in this case is played by the hinge loss function: $max(0, 1 - y_i(\vec{w} \cdot \vec{x_i} - b))$ [52].

  The y_i is the target class (1, -1) and the $\vec{w} \cdot \vec{x_i} - b$ is the return of the algorithm for the i-th point.

  This function returns zero if $\vec{x_i}$ lies on the correct side of the margin. For data on the wrong side of the margin, the function's value is proportional to the distance from the margin. So we come to the conclusion that we need to minimize:

$$\left[\frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i(\overrightarrow{w} \cdot \overrightarrow{x_i} - b))\right] + \lambda||\overrightarrow{w}||^2 \tag{6.5}$$

The parameter λ determines the trade-off between increasing the margin size and ensuring that the $\overrightarrow{x_i}$ lies on the correct side of the margin. Thus, for sufficiently small values of λ, the second term in the loss function will become negligible; hence, it will behave similar to the hard-margin SVM, if the input data are linearly classifiable [49].

6.2.2 RBF SVM Classification

The linear support vector machine can also be applied to datasets that have non-linear decision limits. The idea is to transform the data from the original coordinate space in x, to a new space $\Phi(x)$, so that a linear decision limit can be used to separate the records in the transformed space. After the transformation, the methodology presented in Section 6.2.1 can be applied to find a linear decision limit in the transformed space [53].

6.2.2.1 Kernel Trick

The calculations needed for this conversion are sometimes expensive and are likely to suffer from the curse of dimensionality. An innovative solution to this problem appears in the form of a method called kernel trick.

The kernel trick is a method of calculating the similarity in the transformed space by using the original set of features. The similarity function of records K, which is computed in the original features space, is known as the kernel function [54]. The kernel trick helps to address some of the concerns about implementing nonlinear support vector machines [49]:

1. The exact form of the imaging function Φ will not need to be known, as the core functions used in nonlinear support vector machines must satisfy a mathematical principle known as Mercer's theorem [55]. This principle ensures that kernel functions can always be expressed as the interior product between two input vectors in a multidimensional space. The transformed kernel space of the support vector machine is called the reproducing kernel Hilbert space - RKHS.

2. Computing internal products using kernel functions is significantly more economical than using the transformed set of characteristics $\Phi(x)$.

3. Since the calculations are made in the original space, issues related to the curse of many dimensions, are avoided.

6.2.2.2 RBF SVM in practice

As discussed earlier, the radial basis function kernel, or RBF kernel, is a popular kernel function used in various kernelized learning algorithms. The SVM algorithm is the one most commonly encountered. Having learned the basics about kernel functions and what they are, we need to see how this kernel function is applied to the SVM algorithm.

The RBF SVM algorithm (implemented by sklearn) has two hyperparameters that must be provided by the user. Naturally, the values of these hyperparameters affect how the algorithm works and, therefore, influence its results and efficiency in classifying new data. These two parameters are called **gamma** and **C**.

The **gamma** parameter determines how far the influence of a single training example reaches, with low values meaning *far* and high values meaning *close*. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. The **C** parameter trades off correct classification of training examples against maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. In other words, *C* behaves as a regularization parameter within the SVM.

The behavior of the model is *very sensitive to the gamma parameter*. If gamma is too *large*, the radius of the area of influence of the support vectors only includes the support vector itself, and no amount of regularization with C will be able to prevent overfitting. When gamma is very *small*, the model is too constrained and cannot capture the complexity or shape of the data. The region of influence of any selected support vector would include the whole training set. The resulting model will behave similarly to a linear model with a set of hyperplanes that separate the centers of high density of any pair of two classes. Smooth models (with lower gamma values) can be made more complex by increasing the importance of classifying each point correctly (through larger C values).

Finally, one can also understand that, for some intermediate values of gamma, we get equally performing models when C becomes very large: it is not necessary to regularize by enforcing a larger margin. The radius of the RBF kernel alone acts as a good structural regularizer. In practice, though, it might still be interesting to simplify the decision function with a lower value of C, so as to favor models that use less memory and are faster to predict [44].

6.3 Data splitting into categories

As noted in Section 4, the data we received from Kafka's measurements and experiments include continuous numerical data. By numerical data, we mean continuous data and not discreet data that are usually represented as categorical data. Integers and floats are the most common and widely used types of numeric data for continuous numeric data.

In this chapter, we referred to the basics of some classification algorithms. We must, therefore, examine, from this point on, whether these models and algorithms have the appropriate predictive power to provide us with satisfactory forecasts for the new test data. The purpose of this work is to create Machine Learning models that will be trained with input data—essentially, the settings of the Kafka pub/sub system—and correspond to its performance; and when they receive new data, to be able to predict the corresponding performance of the system with those settings. However, as we mentioned before, our data are numerical. This means that they do not belong to a specific category. Although the numerical data can be fed directly to Machine Learning models, in the case of classification, our model will take each value as a separate feature/category.

It is therefore understood that, in order to be able to properly apply the classification algorithms to our data, we must divide them into corresponding categories/buckets. Each bucket will be a category in which the data will be grouped. In practice, our input data have a range of values from x_0 (the smallest value) to x_n (the largest value). As such, each bucket will be a part of this value range. When new data is introduced into the model, it will be able to predict which category it belongs to and, thus, predict Kafka's system performance with the new data/settings with relative accuracy. Accuracy depends on the number of buckets; the smaller the number of buckets is, the smaller the accuracy of actual value will be because each bucket corresponds to a wider range of values. On the contrary, the greater the number of buckets is, the smaller their value range will be, so the greater the accuracy of the actual predicted value.

For this purpose we utilized the *qcut()* function of the *Python Pandas library*. This function essentially splits variables into equal-sized buckets based on rank, or based on sample quantiles. In this thesis, we tested different bin sizes [2, 3, 5, 7, 10] to see which bucket number the algorithm would handle best. When the number is small, there may be more accuracy in the correct prediction of the category. However, the range is large, so the prediction of actual value, as discussed in the previous paragraph, will be relatively small. On the other hand, as the number of buckets grows, the accuracy in forecasting the category decreases, though the accuracy in predicting the actual value practically increases due to the small value range of each category. Therefore, we will attempt to find the right balance between the number of bins and the forecast of the real values of the new data [56], [57].

7. IMPLEMENTATION AND RESULTS

In the previous chapters we took notice of the basic parts of Pub/Sub systems, observed the setup with which the experiments were carried out and the data were collected; we also explained the regression and classification algorithms we used in this thesis. In this section, we will delve deeper on how we implemented the algorithms, how we measured their performance, and will present their results. The algorithms were implemented with the Python programming language and using one of its most well-known libraries, sklearn. Its full name is Scikit-learn, and it is a free machine learning library for Python. It features various learning algorithms and performance metrics and also supports Python numerical and scientific libraries such as NumPy and SciPy. We also made use of two well-known Python libraries, NumPy and Pandas, to read and format the data.

7.1 Cross Validation

The error estimation helps the training algorithm to make the model selection, i.e. to find a model with the right complexity, which is not prone to over-fitting. The more complex the model, the greater the over-fitting. Once the model is built, it can be applied on the test set to predict the expected values or categories of records, depending on whether we are talking about a regression or classification algorithm.

It is often useful to measure the performance of the model in the test set, because such a measure provides an unbiased estimate of the generalization error of the model. The accuracy or degree of error calculated by the test set can also be used to compare the relative performance of different models in the same field. We will therefore consider the cross-validation method, which is often used to evaluate the performance of a machine learning model.

Cross-Validation (CV) is a statistical technique for estimating unknown parameters of a model or for estimating its performance. We could say that the estimation of unknown parameters is part of the performance estimation of the model, as its performance depends on its parameters and, as such, the correct estimation and prediction of the parameters gives us an overall better performance of the model. In this technique, each record is used the same number of times as the others for training, and exactly once for testing.

To illustrate this method, let us assume that the data is split into two equal subsets. Firstly, we select one of the two subsets for training and the other for testing. Next, we alternate the roles of the subsets so that the set used for training can now be used for test, and vice versa. This approach is called double cross-validation. The total error is obtained by averaging the errors of the two executions. In this example, each record is used exactly once for training and once for checking.

The **k-fold cross-validation** generalizes the above example by dividing the data into k segments of equal size. During each run, only one of the sections is selected for testing and the rest for training. This process is repeated k times, so that each section is used for test only once. That is, in each iteration, a different section is used for verification, while the rest is used for training. Again, the total error is calculated by averaging k repetitions.

The procedure and steps of the algorithm are as follows:

1. Randomly shuffle the dataset
2. Divide the dataset into *k* groups
3. Define a group as the test dataset
4. Define the remaining *k-1* as training dataset
5. Train the model with the training dataset
6. Evaluate the model with the test dataset
7. Repeat of steps (3), (4), (5) and (6) for *k-1* more repetitions with different test dataset each time (and therefore in parts different training sets)
8. Select the best model based on the evaluation scores

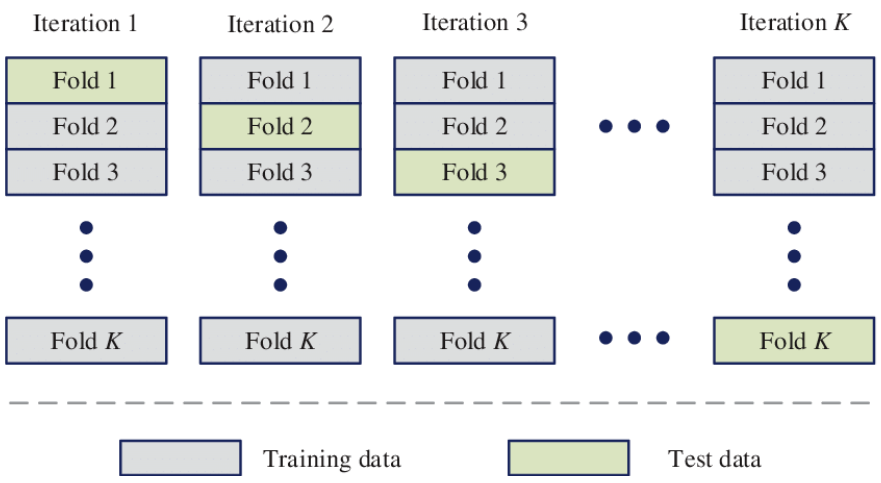


Figure 7.1: K-fold Cross Validation

A special case of k-fold cross-validation is one that sets k = N, where N is the number of records in the data set. This approach is called **leave one out**. Each test set contains only one entry and all the others (N-1) constitute the training set. This approach has the advantage of using as much data as possible for education. Moreover, test sets are mutually exclusive and virtually cover the entire data set. A drawback of this method is that it is computationally expensive due to the N repetitions. Furthermore, since each test set contains only one record, the performance variation expected to be high [44], [49], [58].

7.2 Metrics

As we as mentioned in the previous sections, we used two categories of algorithms for the purposes of this dissertation; the first category being regression algorithms to examine whether it is possible to accurately predict system performance. The second category involves classification algorithms. With this category, we aim to classify the training data into categories, according to their performance, and assess if we can predict the category of test data.

Understandably, in order to determine the reliability of each algorithm, we need to measure its performance. We can achieve this with some mathematical functions. Each function is a different metric. The variety of metrics shows us the efficacy of the algorithm from different points of view, and helps us to better understand its behavior and actual performance.

Unfortunately, the two categories of algorithms feature several differences in the way and in what they predict. As we have mentioned, regression algorithms essentially predict values, while classification algorithms predict categories. Therefore, their metrics could not be the same as they also depend on the type of algorithm. Below we note the corresponding metrics we used in each category.

7.2.1 Regression Algorithms Metrics

The metric functions we used in the regression algorithms include the following:

- **R^2 score**: R-squared (R^2) represents the proportion of variance of y that has been explained by the independent variables in the model. Whereas correlation explains the strength of the relationship between an independent and dependent variable, R-squared explains to what extent the variance of one variable explains the variance of the second one. Thus, if the R^2 of a model is 0.50, then approximately half of the observed variation can be explained by the model's input. It provides an indication of goodness of fit and, therefore, a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance. The calculation type is:

$$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{7.1}$$

  where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ and $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}\epsilon_i^2$.
  In simpler words, we could say that the formula is interpreted as follows:

$$R^2 = 1 - \frac{Unexplained\ Variation}{Total\ Variation}$$

  The best possible score is 1.0 and it can also be negative. as the model can be arbitrarily worse. A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0 .

- **Explained Variance Score**: The *explained variance score* metric computes the *explained variance regression score*. It measures the proportion to which a mathematical model accounts for the variation (dispersion) of a given data set. Dispersion is the extent to which a distribution is stretched or squeezed. If $\hat{y}$ is the estimated target output, y the corresponding (correct) target output, and Var is Variance, the square of the standard deviation, then the explained variance is estimated as follows:

$$explained\_variance(y, \hat{y}) = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}} \tag{7.2}$$

  The best possible score is 1.0, while lower values are worse [44], [59].

- **Mean Absolute Error (MAE)**: The mean absolute error function is a risk metric corresponding to the expected value of the absolute error loss or $l1$-norm loss. It is also a measure of errors between paired observations expressing the same phenomenon. If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean absolute error (MAE) estimated over $n_{samples}$ is defined as:

$$\text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \left| y_i - \hat{y}_i \right|. \tag{7.3}$$

  The mean absolute error uses the same scale as the data being measured. This is known as a scale-dependent accuracy measure and, therefore, cannot be used to make comparisons between series using different scales. Lower values are better favored [44], [60].

- **Mean Squared Error (MSE)**: The mean square error function is a risk metric corresponding to the expected value of the squared (quadratic) error or loss. The average of the squares of the errors that is measured, is the average squared difference between the estimated values and the actual value. If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the mean squared error (MSE) estimated over $n_{samples}$ is defined as:

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2. \tag{7.4}$$

  The MSE is a measure of the quality of an estimator. It is always non-negative, and values closer to zero are preferred [44], [61].

- **Median Absolute Error (MedAE)**: The median absolute error is very useful, as it essentially is insensitive to outliers (as long as there are not too many of them). This is because it is the median of all absolute values of the residuals, and the median is unaffected by values at the tails. As a result, this loss function can be used to perform robust regression. Robust regression is a form of regression analysis designed to overcome some limitations of traditional parametric and non-parametric methods. If $\hat{y}_i$ is the predicted value of the i-th sample, and y_i is the corresponding true value, then the median absolute Error (MedAE) estimated over $n_{samples}$ is defined as:

$$\text{MedAE}(y, \hat{y}) = \text{median}(\mid y_1 - \hat{y}_1 \mid, \ldots, \mid y_n - \hat{y}_n \mid). \tag{7.5}$$

Note that using the median absolute error only corrects for outliers in the response/target variable, not for outliers in the predictors/feature variables [44].

7.2.1.1 Negative values of metrics

As can be seen not only from the mathematical formulas of the metrics we presented above, but also from what the latter actually represent, the quantities they provide should be positive numbers. However, as we will later discuss in the presentation of the results, some of the metrics return negative values. This applies to the following metrics:

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Median Absolute Error (MedAE)

According to the sklearn library module used to take the metric functions to measure the effectiveness of the predictions, some functions return a positive score and others a negative one [44]. In more detail:

- functions that end with *_score* return a value for maximization; the higher, the better.
- functions ending with *_error* or *_loss* return a value for minimization; the lower, the better.

Nonetheless, there is a reasonable explanation for this choice of library developers: The unified scoring API of sklearn always maximizes the score, so scores which need to be minimized are negated in order for the unified scoring API to work correctly. The score that is returned is therefore negated when it is a score that should be minimized and left positive if it is a score that should be maximized.

The actual score of the MAE, MSE, MedAE, is simply the positive version of the number we are getting.

7.2.2 Classification Algorithms Metrics

For the classification algorithms we used the **classification_report()** function of sklearn. The classification report visualizer displays the **accuracy**, **precision**, **recall**, **F1**, and **support scores** for the classification model. The classification report shows a representation of the main classification metrics on a per-class basis. This gives a deeper intuition of the classifier behavior over global accuracy, which can mask functional weaknesses in one class of a multiclass problem.

The metrics are defined in terms of true and false positives, and true and false negatives. Positive and negative in this case are generic names for the classes of a binary classification problem. Therefore a true positive occurs when the actual class is positive, similarly to the estimated class. A false positive emerges when the actual class is negative but the estimated class is positive [62], [63].

The analysis of metrics is necessary to understand what they represent and how effective our model is:

- **Accuracy**: Classification Accuracy is what we usually point to when using the term accuracy. It features the ratio of number of correct predictions to the total number of input samples:
$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{7.6}$$

  It works well only if there is an equal number of samples belonging to each class. For example, consider that there are two classes, A and B, in our training set, and class A has many more samples than class B. Then, our model can easily get very high training accuracy simply by predicting every training sample belonging to class A. When class A has 50% of samples then accuracy is 50%, if we once again predict all the samples belonging to class A.

  We can easily understand that classification Accuracy is a great metric, although gives us a false sense of achieving high accuracy [44], [64].

- **Precision**: Precision is the ability of a classifier not to label an instance that is actually negative, as positive. For each class it is defined as the ratio of true positives to the sum of true and false positives. In other words, *"for all instances classified positive, what percent was correct?"*. The mathematical formula is:

$$Precision = \frac{true\_positive}{true\_positive\ +\ false\_positive} \tag{7.7}$$

  The best value is 1 and the worst value is 0. When $true\_positive + false\_positive == 0$, precision returns 0 [44], [62], [63].

- **Recall**: Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives. That is, *“for all instances that were actually positive, what percent was classified correctly?”*. The formula is:

$$Recall = \frac{true\_positive}{true\_positive\ +\ false\_negative} \tag{7.8}$$

The best value is 1 and the worst value is 0. When $true\_positive + false\_negative == 0$, recall returns 0 [44], [63].

- **F1**: The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. Strictly speaking, F1 scores are lower than accuracy measures as they embed precision and recall into their computation. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

$$F1 = \frac{2\ *\ (precision\ *\ recall)}{(precision\ +\ recall)} \tag{7.9}$$

  As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy [44], [63].

- **Support scores**: Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support does not change between models but instead diagnoses the evaluation process [63].

7.3 Hyperparameter optimization

7.3.1 Grid Search

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to test the learning process. Consequently, they should be adjusted properly so as to provide us with the best possible result. In terms of regression and classification algorithms, improving the result implies better predictive power of the algorithm and the model in general. Of course, the improvement is not only about the performance of the algorithm in terms of results, but also in terms of speed. Many times optimal tuning of the hyperparameters leads the algorithm towards producing results faster. However, we must remember that the hyperparameters of a model also depend on the training data. A change in the variation of data can give us other hyperparameters that are optimal for the specific data.

There are many approaches to finding the optimal hyperparameters of ML algorithms. Some of them are:

- Grid Search
- Random search
- Bayesian optimization
- Gradient-based optimization
- Evolutionary optimization

alongside many more algorithms and techniques.

In this thesis, we employed the Grid Search technique to configure the best hyperparameters. Grid Search is the most traditional way of performing hyperparameter optimization, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a hold-out validation set. Grid search is a slow technique as it can do many iterations of the algorithm execution, depending on the number and range of parameters given to it. But it is also a very effective solution for finding the optimal hyperparameters [44], [65].

We used the **GridSearchCV** function provided by the *sklearn* library. In this implementation of the library, the estimator parameters are optimized by cross-validated grid-search over the parameter grid.

7.3.2 Algorithm parameters

The parameters we examined in each algorithm are:

- **Linear Regression**: This model is the basis on which the other linear regression models are built and, as a result, it is, in essence, the simplest model and does not have any special parameter that requires adjustment. The default values of the parameters $fit\_intercept = True$ and $normalize = False$ were used.
- **Lasso Regression**: In this algorithm we did not give a value or any range of values to parameter a. With this technique, the $LassoCV()$ implementation of the sklearn decides for itself which value is better, depending on the data. Furthermore, the data were not normalized as this is done automatically by using the parameters $fit\_intercept = True$ and $normalize = True$. Finally, this implementation incorporates cross-validation to determine the splitting strategy.
- **LassoLARS Regression**: As in the previous regression algorithm, so in this one, which is a variation of the former, we did not use exhaustive search (Grid Search). Instead, we gave the algorithm a range of 10000 values of the hyperparameter a to search and decide the, itself, which value is better according to our data. Additionally, as before, the data were normalized by the model through the parameters

$fit\_intercept = True$ and $normalize = True$. Lastly, we utilized the $LassoLarsCV()$ implementation of the sklearn that performs cross-validation, so as to determine the splitting strategy.

- **CART Regression**: Since this algorithm uses decision trees to regress and predict new values, the two most basic parameters we examined with Grid Search to find their optimal combination based on the data, are the maximum depth of the tree and the minimum number of samples required to be at a leaf node. At the standard of the sklearn $DecisionTreeRegressor()$ function we used, these two parameters are referred to as max_depth and $min\_samples\_leaf$, respectively. Moreover, as a criterion for the separation of data at each level, we used the metric error measurement ”MSE” for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the $l2$ loss using the mean of each terminal node.

- **K-NN Classification**: The number of neighbors is the most important parameter of the K-NN algorithm and we used exhaustive search (grid search) to find its optimal value for each dataset, is the number of neighbors or otherwise $n\_neighbors$, as corresponding to the standard of the sklearn function that was used. There are other parameters of the classification algorithm, such as the nearest neighbor calculation algorithm which we set to auto so it will attempt to decide the most appropriate algorithm based on the data; the metric for measuring the distance from the data points where we used the Minkowski metric (p = 2), and the weight of the points which, in essence, is how important each point is (depending on its distance from the new unclassified point), where we considered uniform weight for all points.

- **Linear-SVM Classification**: In this algorithm, the parameter we set after searching in a range of values is the regularization parameter C. The regularization parameter C defines the balance between maximizing the class margin and successfully categorizing the training data. The higher the C value, the higher the penalty for incorrect training data classification. Since we are talking about linear SVM then the kernel is automatically defined as linear. In addition, we used the $l2$ function, which is the default function in the implementation of the Linear-SVM algorithm by sklearn, as a penalty function. $l2$ is, in essence, the normalization $Ridge\ Regression$ function with formula: $\lambda||w||_2^2$. Finally, the fault tolerance is small enough to be able to have good and accurate results without much error margin.

- **RBF-SVM Classification**: Since this algorithm is a variant of the linear-SVM, the parameters could not be vastly different. We have set the kernel in this variant of SVM classifier as RBF (we have explained in a previous chapter what the difference between a linear and rbf kernel is). Exhaustive search in this algorithm was used to find the best combination of the parameters C and $gamma$ for the input data. As we said before, the C parameter defines the balance between maximizing class margin and successfully categorizing training data, while the $gamma$ parameter scales the influence of each training data. The higher the value of $gamma$, the faster (close) the effect weakens. Finally, as before, we once again set the error tolerance to small, so we have good and accurate results without much error margin.

7.4 Model training and evaluation

In this section we will present the results of the models. This section shows how effective (after training) each model was, according to the metrics, in predicting the estimated performance of the Kafka system in terms of ”Records/sec”, ”MB/sec”, ”avg latency” and ”max latency”. These four ”targets” are essential measures of system performance. That is, the higher the ”Records/sec” and the ”MB/sec”, or the lower the ”avg latency” and the ”max latency”, the more efficient our system is. Therefore, we will assess how the models can predict the values of the measures we mentioned, to ”find out” which values of the parameters of the system—e.g. batch.size, message.size, linger.ms etc.—give us the desired values of these targets, so that we can properly configure the system and get high ”Records/sec” and ”MB/sec”, as well as low ”avg latency” and ”max latency”.

7.4.1 Predicted targets

In order to gain a better understanding of the diagrams, as well as the results of the algorithms in general, it is necessary to explain, in further detail, what these ”targets” are.

- **Records/sec**: the ”Records/sec”, as it is easily understood, regards how many records the producer can send to the system without considering the size of each record. The higher the number, the better throughput our system has.
- **MB/sec**: corresponding to the previous target, MB/sec indicates the amount of data (in MB) that the producer can send to the system. The message size multiplied by the Records/sec is almost this number, along with a small overhead added by the system. The higher the MB/sec, the better throughput our system has.
- **Avg Latency**: ”Avg Latency” is the average time it takes for the message to reach each consumer. This can show us the overall average performance of the system. In contrast to the previous 2 targets, the shorter the average latency is, the better performance our system has.
- **Max Latency**: corresponding to the Avg Latency, so the ”Max Latency” shows the maximum time it takes for a record/message to reach the consumer. It is important that the max latency is low, because it shows that our system is working well, even under ”pressure”.

You can find the code for the algorithms, as well as the images and results at the following link: https://github.com/GiannisKalopisis/Adjustable-pub-sub-system/tree/master/ML.

7.4.2 Regression Algorithms

In this section we will present the results of the categorization algorithms. Because the data followed a logarithmic distribution, we applied the logarithmic function to them ($f(x) =$

$\log_{10}(x)$) to convert them to linear and, thus, obtain better results, especially for metrics that measure error/distance, i.e. Mean Absolute Error, Mean Squared Error and Median Absolute Error. By using the logarithmic function on the data, a general improvement of the results of about 20% was observed in the metric R^2 and Explained Variance, and the distance/error in the metrics Mean Absolute Error, Mean Squared Error and Median Absolute Error was dramatically reduced. As we mentioned in Section 7.2.1.1, these 3 metrics return negative values, due to the unified API of the sklearn library. Though, in order to be able to represent them correctly in the diagrams, as well as compare them, the absolute value of these metrics was used.

The results are organized by the ”target” (”Records/sec”, ”MB/sec”, ”Avg latency” and ”Max latency”) we want to predict, so that we can better gauge how effectively we can predict them. For each metric there are two diagrams. The first one shows the behavior of the algorithms per datafile given to them through use of the cross-validation technique, while the second one showcases the behavior of the algorithms and their predictive ability in a dataset that has not been trained, which the algorithms ”see” for the first time. That test-set is a part (20%) of the training dataset.

Finally, I consider it necessary to point out, for a better understanding of the diagrams, that, as regards the metric R^2 and Explained Variance, the higher value is better, while for the metrics Mean Absolute Error, Mean Squared Error and Median Absolute Error the lower value is better.

7.4.2.1 Records/sec results

- $\boldsymbol{R^2}$:

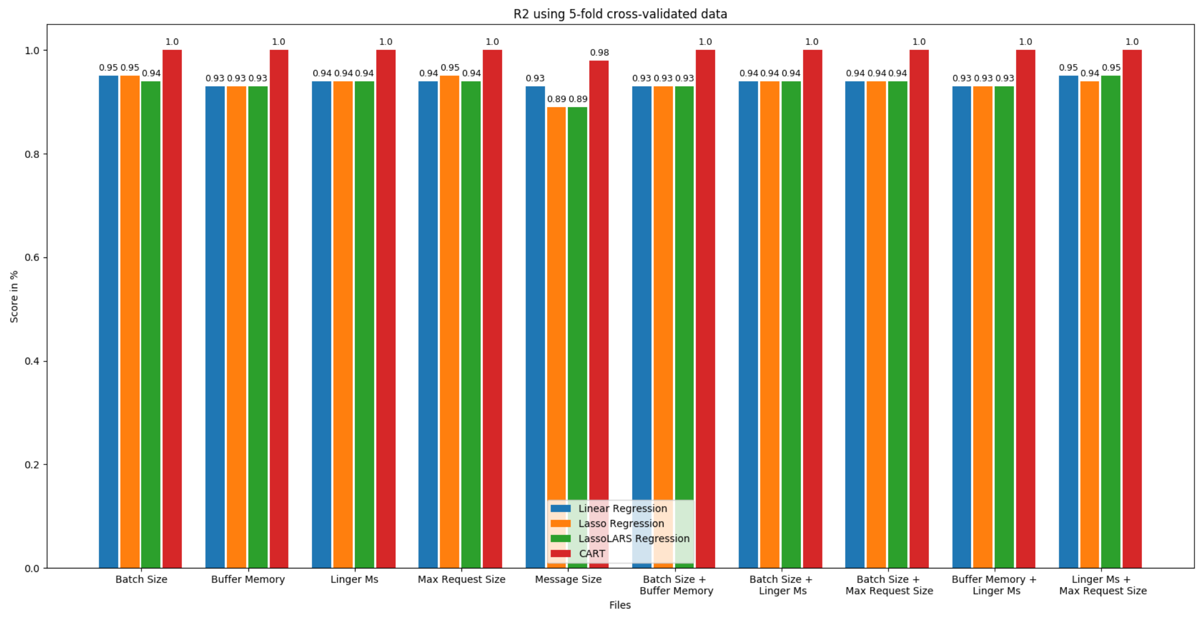


Figure 7.2: Records/sec predicting accuracy using R^2 metric with cross-validated data (higher is better)

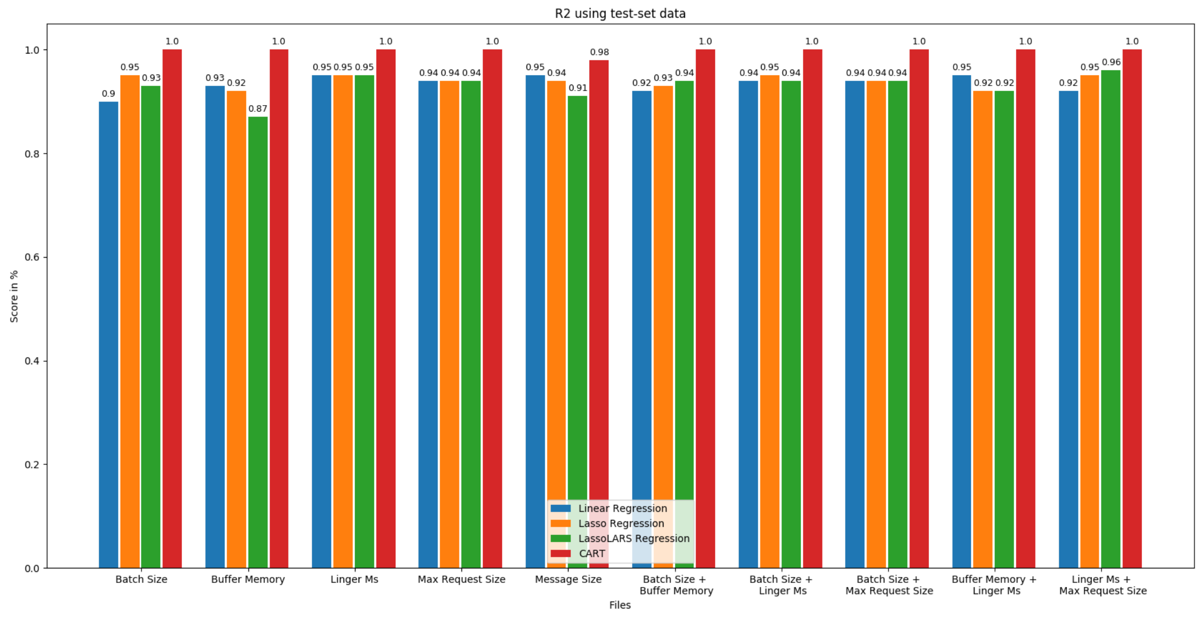


Figure 7.3: Records/sec predicting accuracy using R^2 metric with test set data (higher is better)

- **Explained Variance Score**:

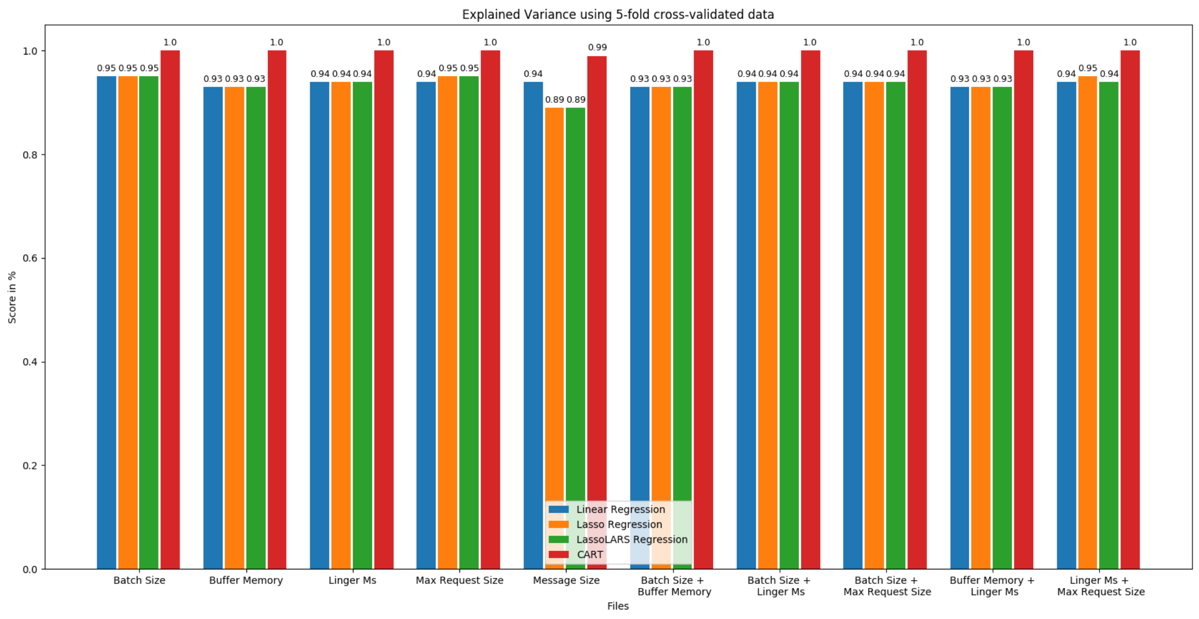


Figure 7.4: Records/sec predicting accuracy using Explained Variance metric with cross-validated data (higher is better)

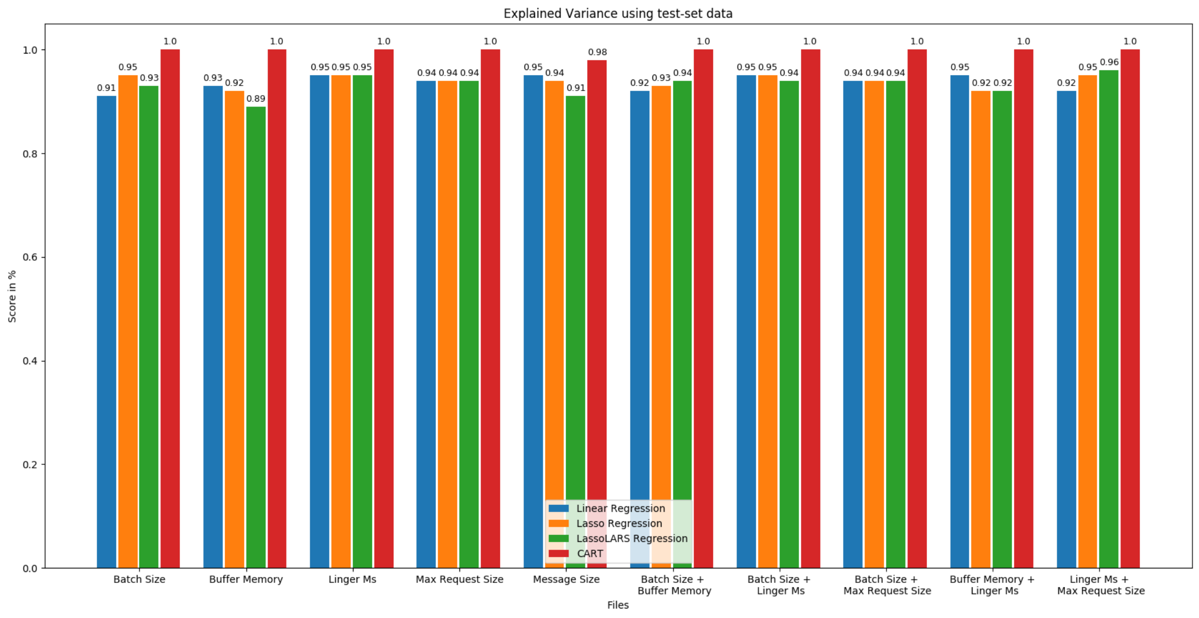


Figure 7.5: Records/sec predicting accuracy using Explained Variance metric with test set data (higher is better)

From the Figures above, we notice that, according to the above two metrics, all algorithms display very good results. Particularly with regard to the CART algorithm, we can remark that it achieves the absolute success rate in many cases. Furthermore, it is equally evident that the differences between the cross-validation and the test sets technique for predicting the values of the target ”Records/sec”, are almost negligible. This means that even with unknown data, such as the test set, the algorithms perform quite well.

- **Mean Absolute Error (MAE)**:

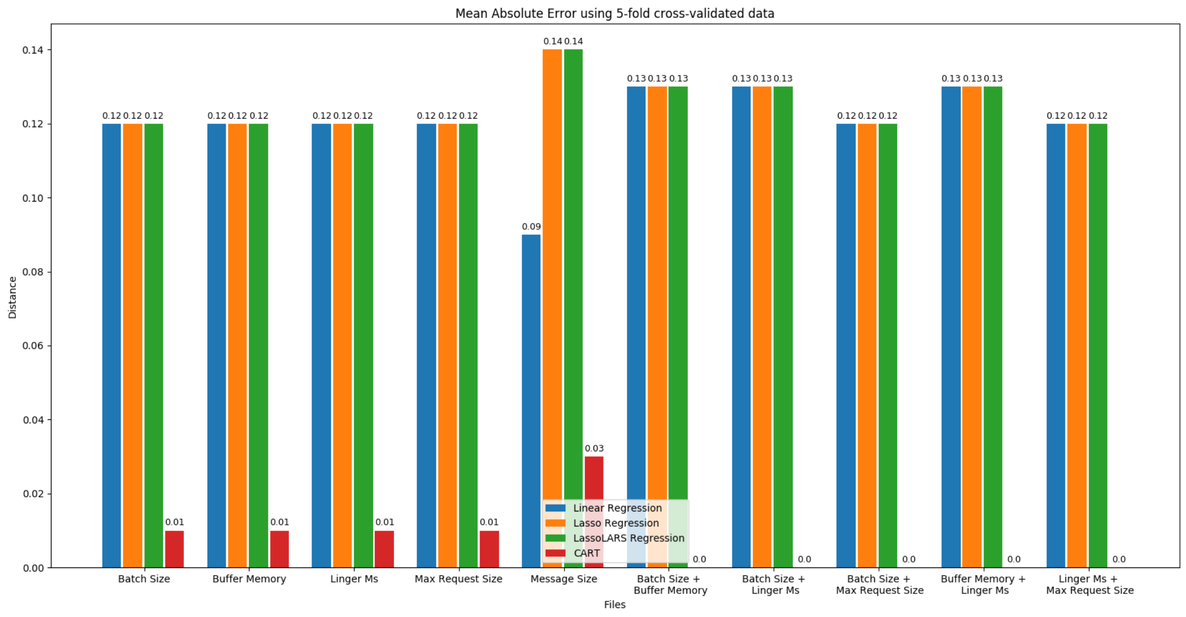


Figure 7.6: Records/sec predicting accuracy using Mean Absolute Error metric with cross-validated data (lower is better)

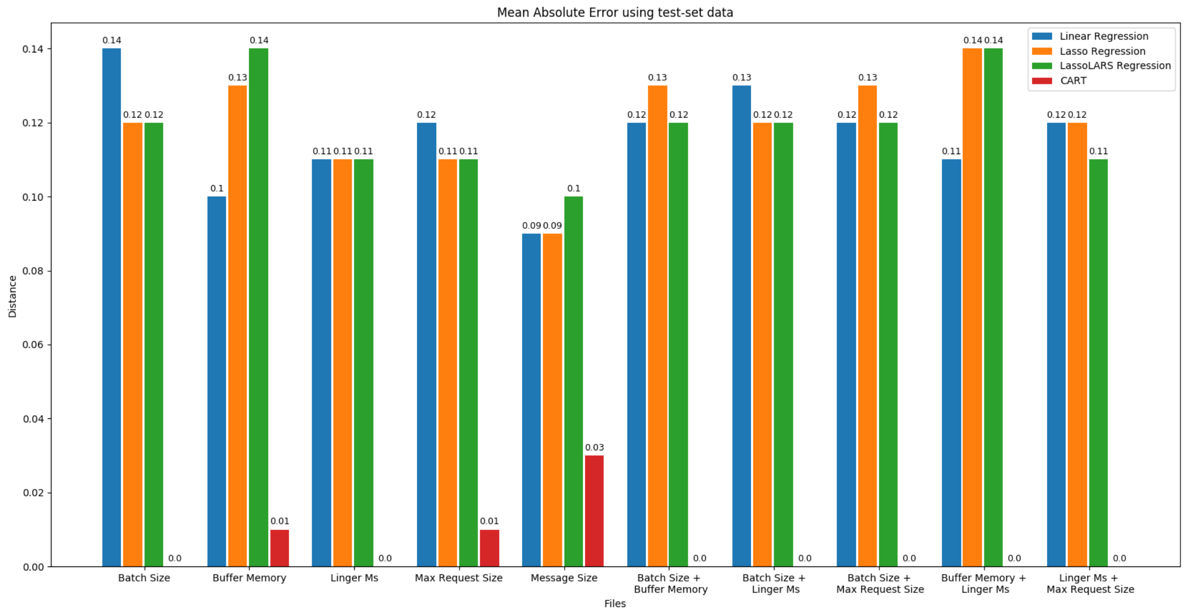


Figure 7.7: Records/sec predicting accuracy using Mean Absolute Error metric with test set data (lower is better)

- **Mean Squared Error (MSE)**:

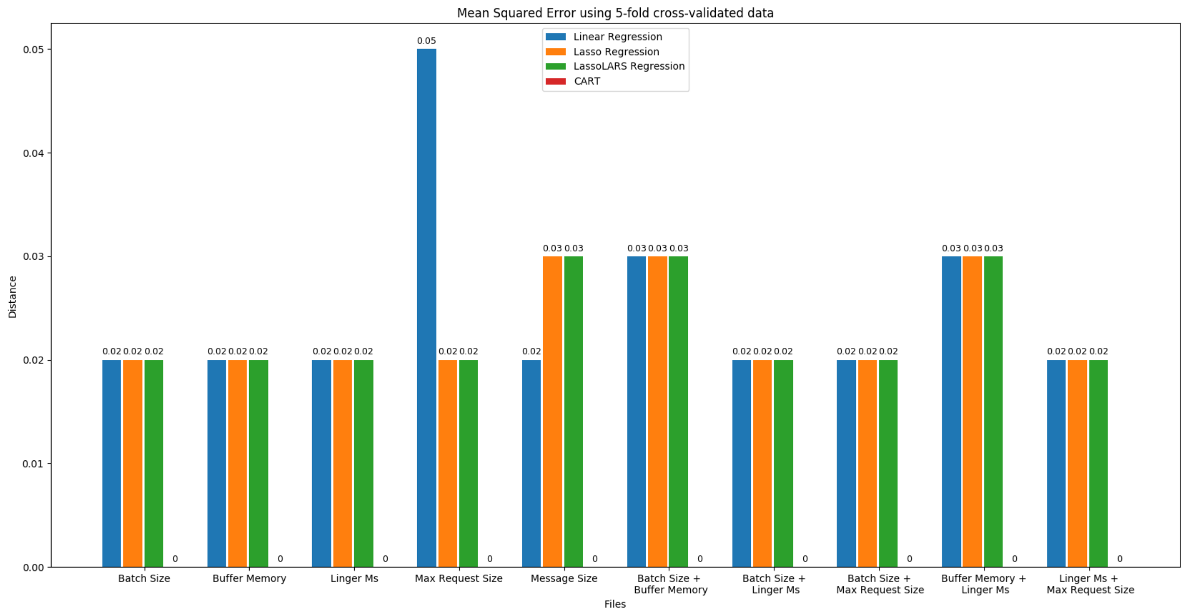


Figure 7.8: Records/sec predicting accuracy using Mean Squared Error metric with cross-validated data (lower is better)

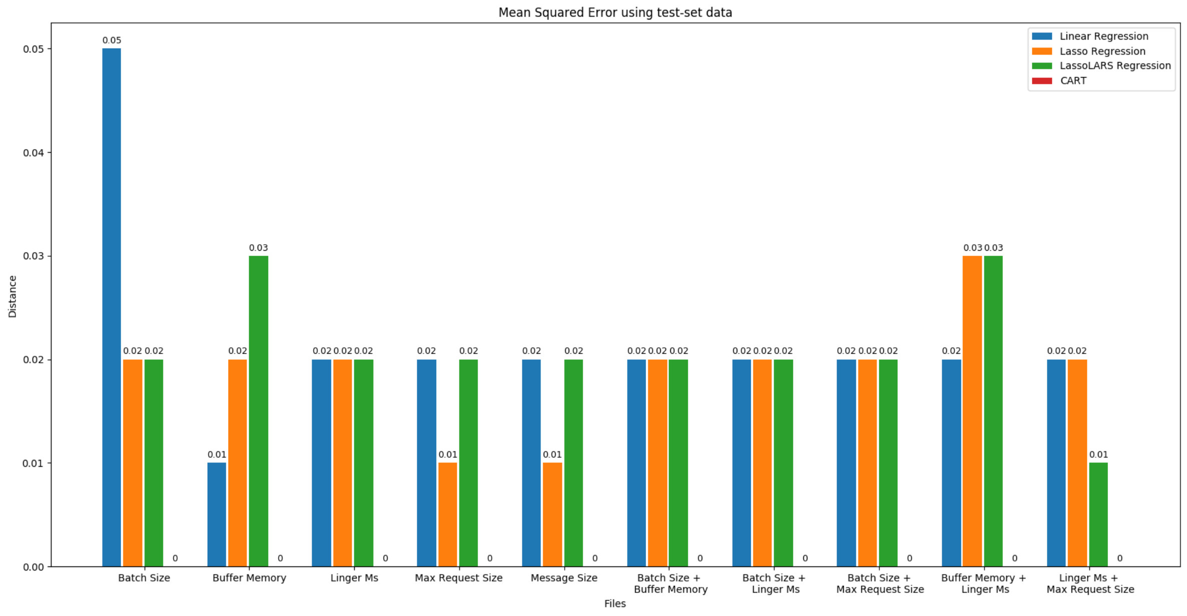


Figure 7.9: Records/sec predicting accuracy using Mean Squared Error metric with test set data (lower is better)

- **Median Absolute Error (MedAE)**:

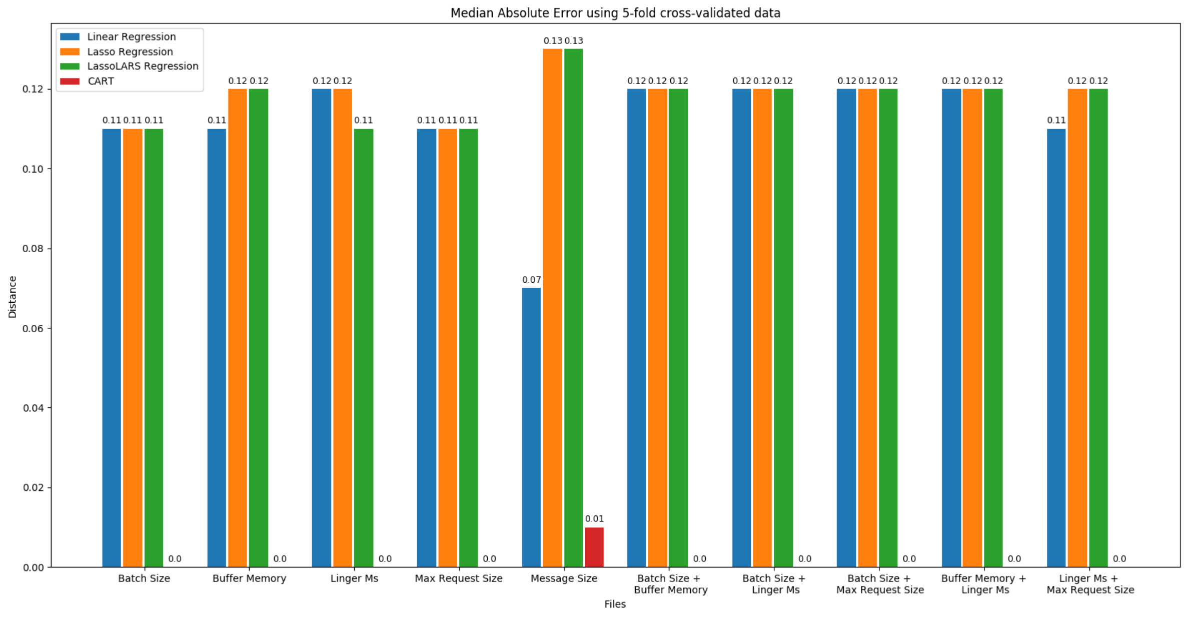


Figure 7.10: Records/sec predicting accuracy using Median Absolute Error metric with cross-validated data (lower is better)

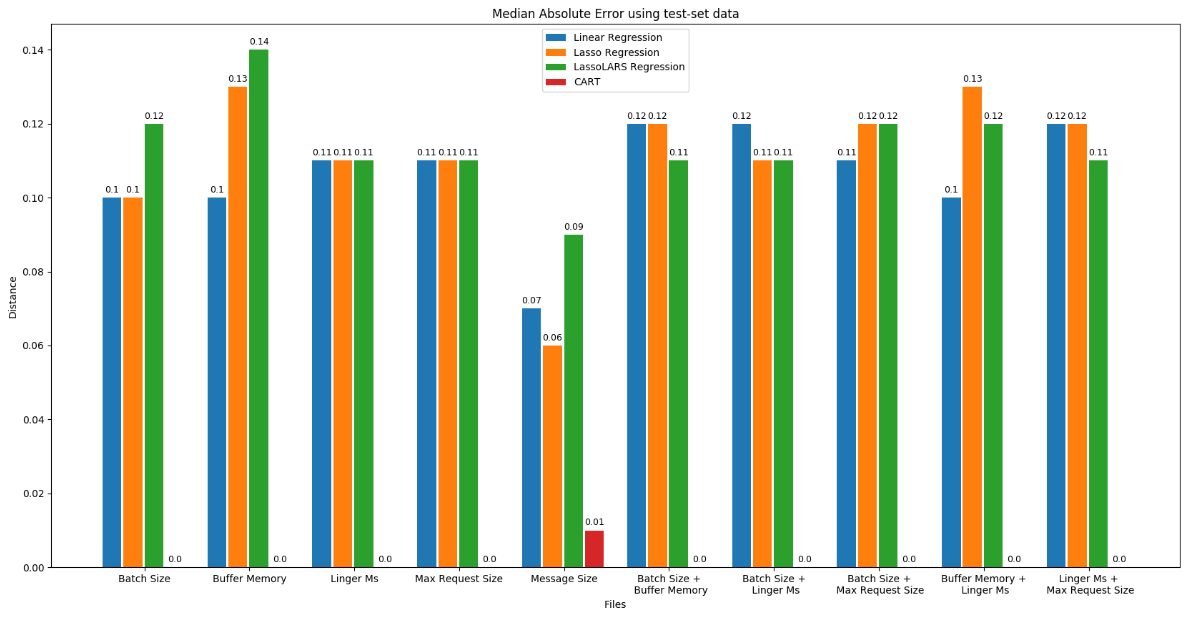


Figure 7.11: Records/sec predicting accuracy using Median Absolute Error metric with test set data (lower is better)

As observed before, the three metrics show very satisfactory results. The CART algorithm, as in the previous metrics, in particular, nearly provides the absolute result—i.e. the value zero—in some cases. Additionally, the differences between the prediction techniques—cross-validation and test set—are very small, although one could claim that the predictions are ”smoother” in the cross-validation, contrary to the test set technique.

7.4.2.2 MB/sec results

- $\boldsymbol{R^2}$:

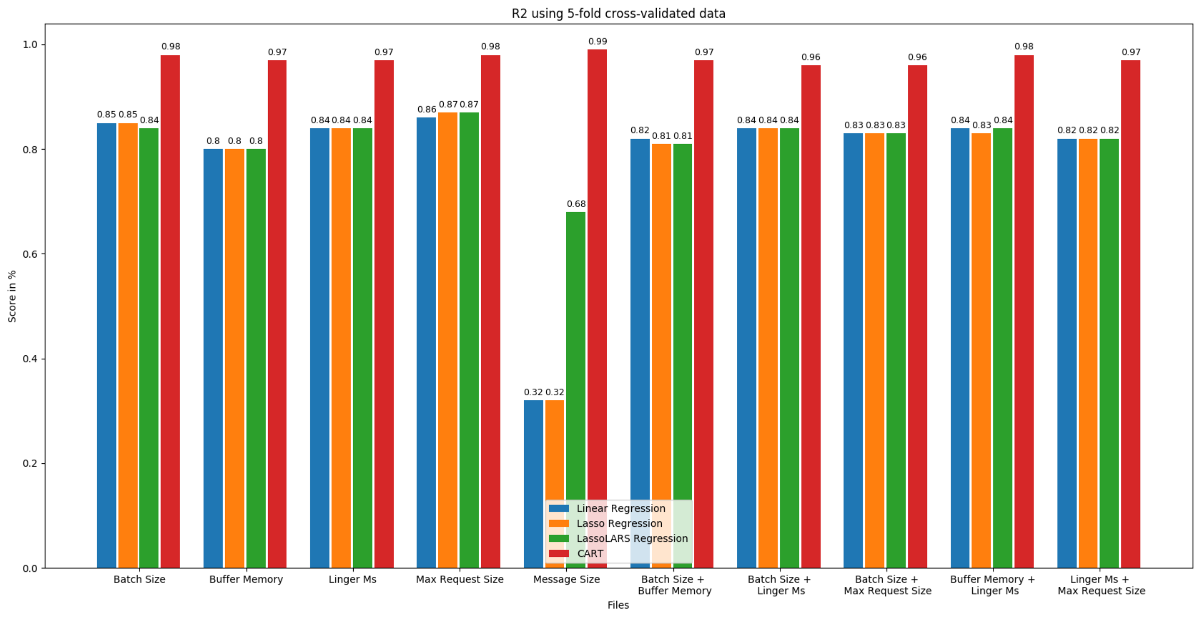


Figure 7.12: MB/sec predicting accuracy using R^2 metric with cross-validated data (higher is better)

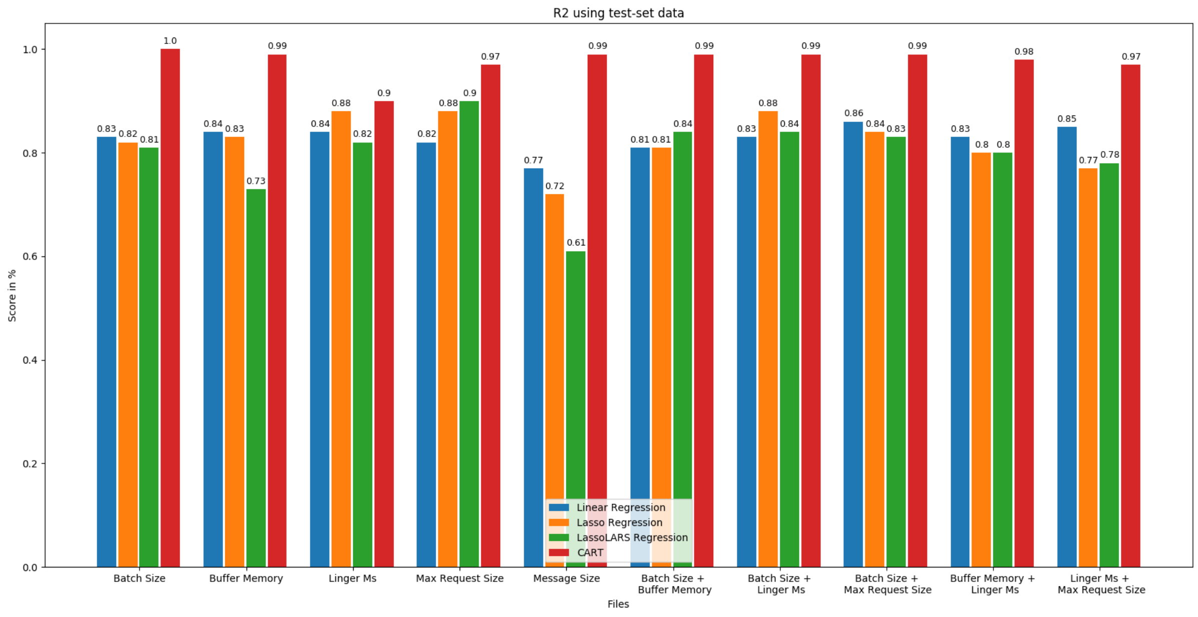


Figure 7.13: MB/sec predicting accuracy using R^2 metric with test set data (higher is better)

- **Explained Variance Score**:

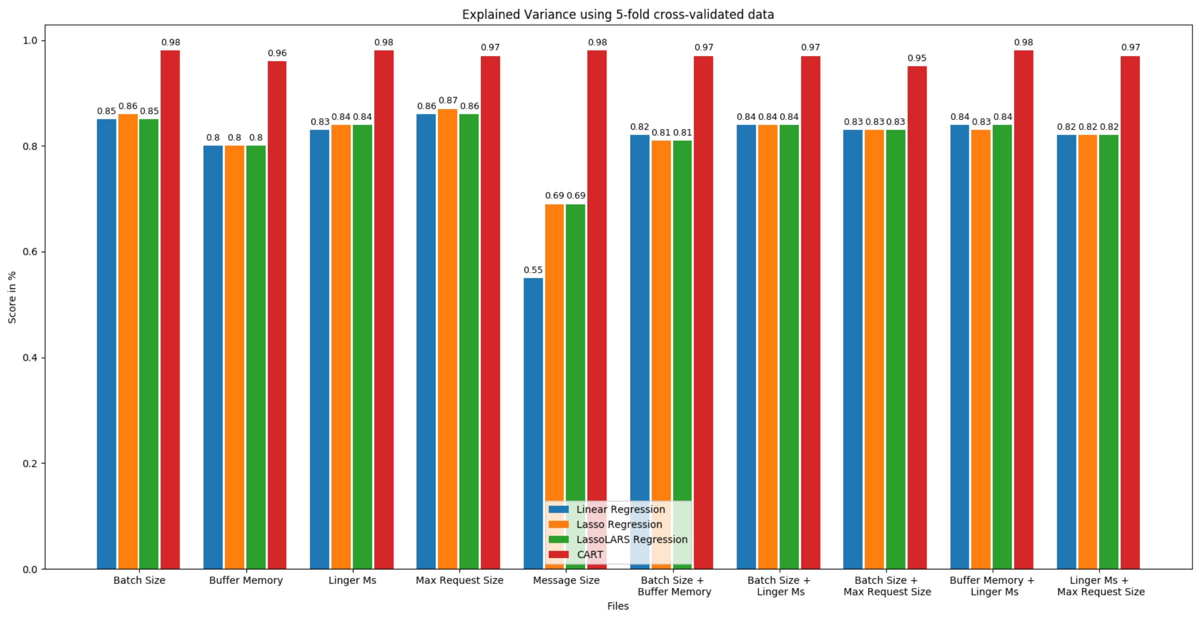


Figure 7.14: MB/sec predicting accuracy using Explained Variance metric with cross-validated data (higher is better)

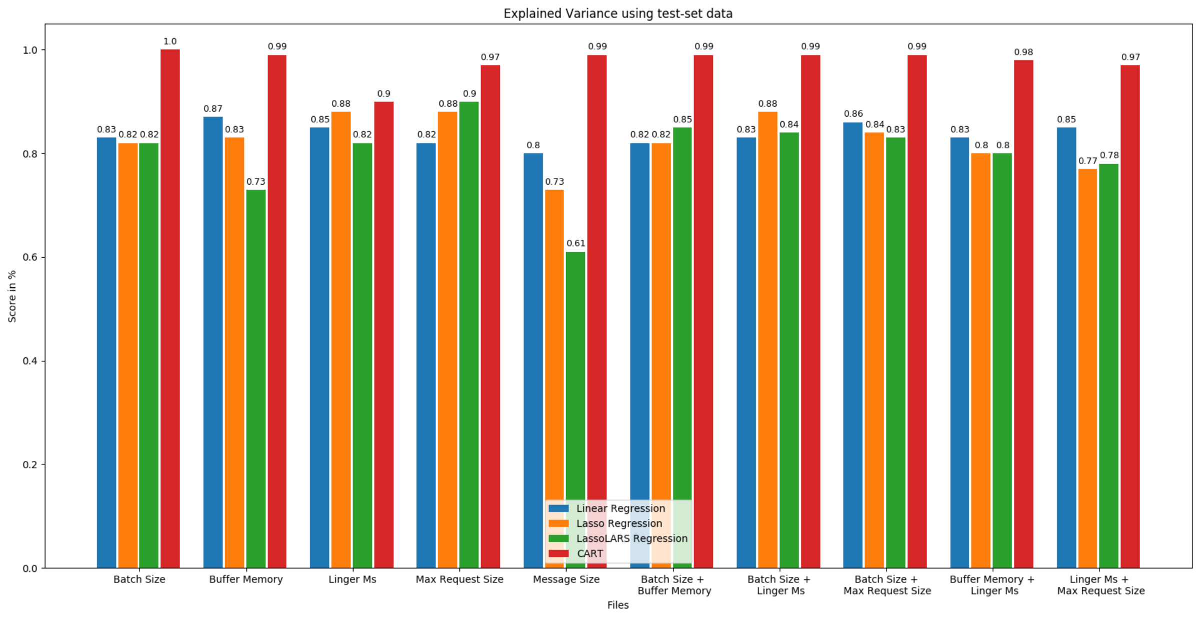


Figure 7.15: MB/sec predicting accuracy using Explained Variance metric with test set data (higher is better)

In comparison to the previous goal ”Records/sec”, we note that the results are good, though not as satisfactory as before. In detail, there is a reduction of 10-15%, yet the CART algorithm again produces excellent results that nearly reach absolute 100%. Something worth highlighting is that, as regards the file ”Message Size”, the results are significantly lower than those of the other files, which applies to all algorithms except CART.

- **Mean Absolute Error (MAE)**:

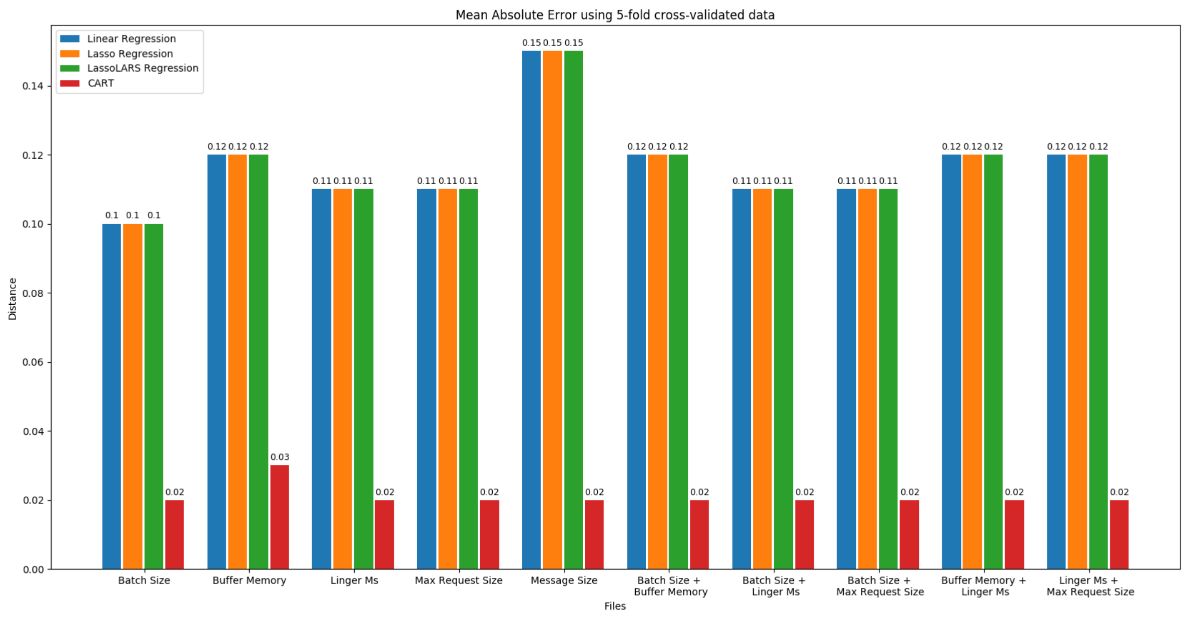


Figure 7.16: MB/sec predicting accuracy using Mean Absolute Error metric with cross-validated data (lower is better)

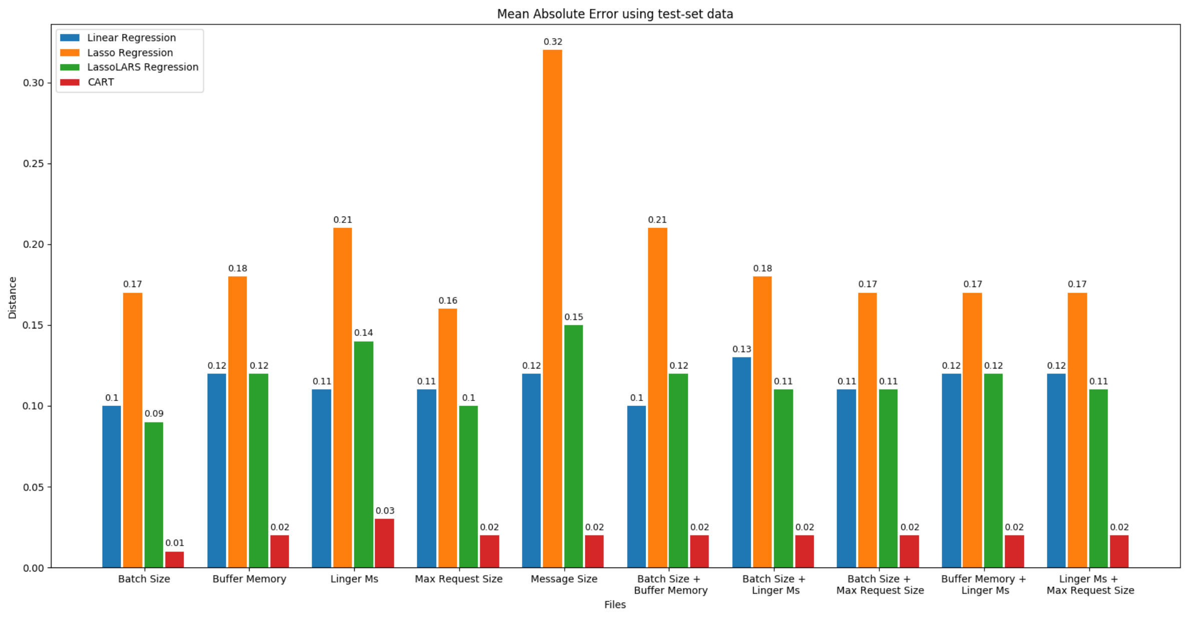


Figure 7.17: MB/sec predicting accuracy using Mean Absolute Error metric with test set data (lower is better)

- **Mean Squared Error (MSE)**:

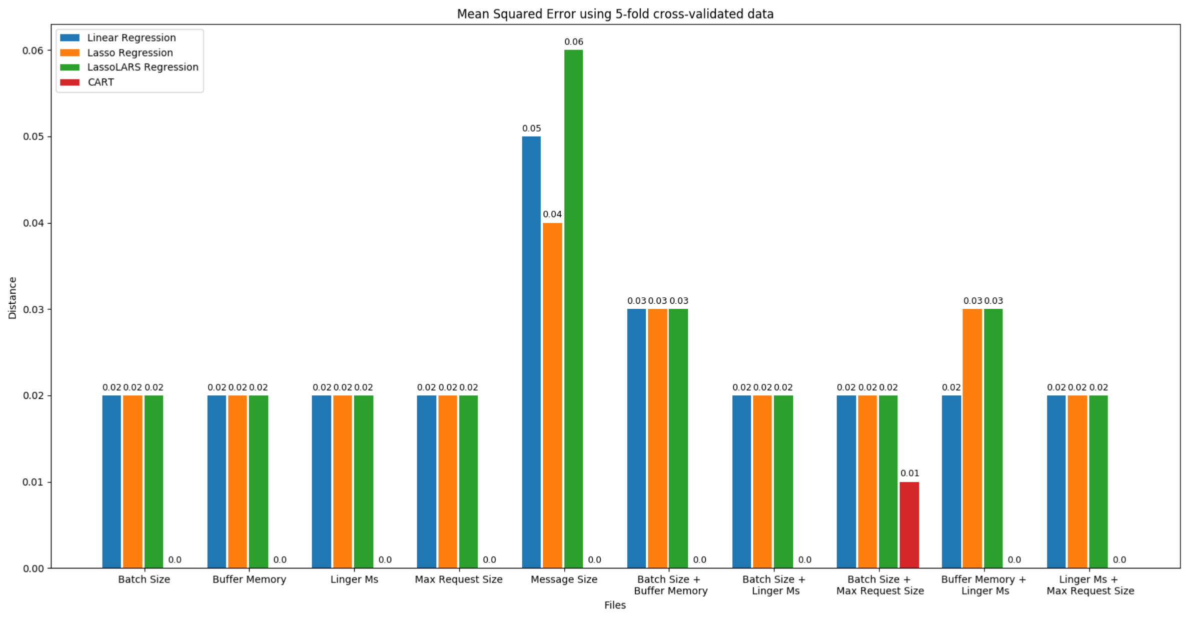


Figure 7.18: MB/sec predicting accuracy using Mean Squared Error metric with cross-validated data (lower is better)

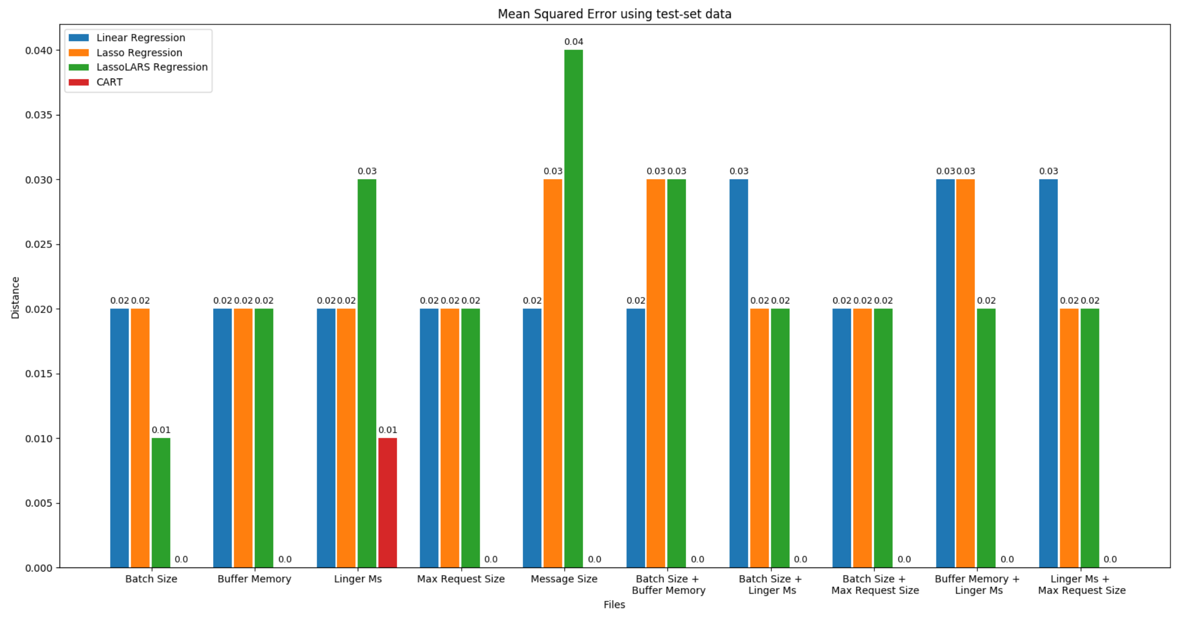


Figure 7.19: MB/sec predicting accuracy using Mean Squared Error metric with test set data (lower is better)

- **Median Absolute Error (MedAE):**

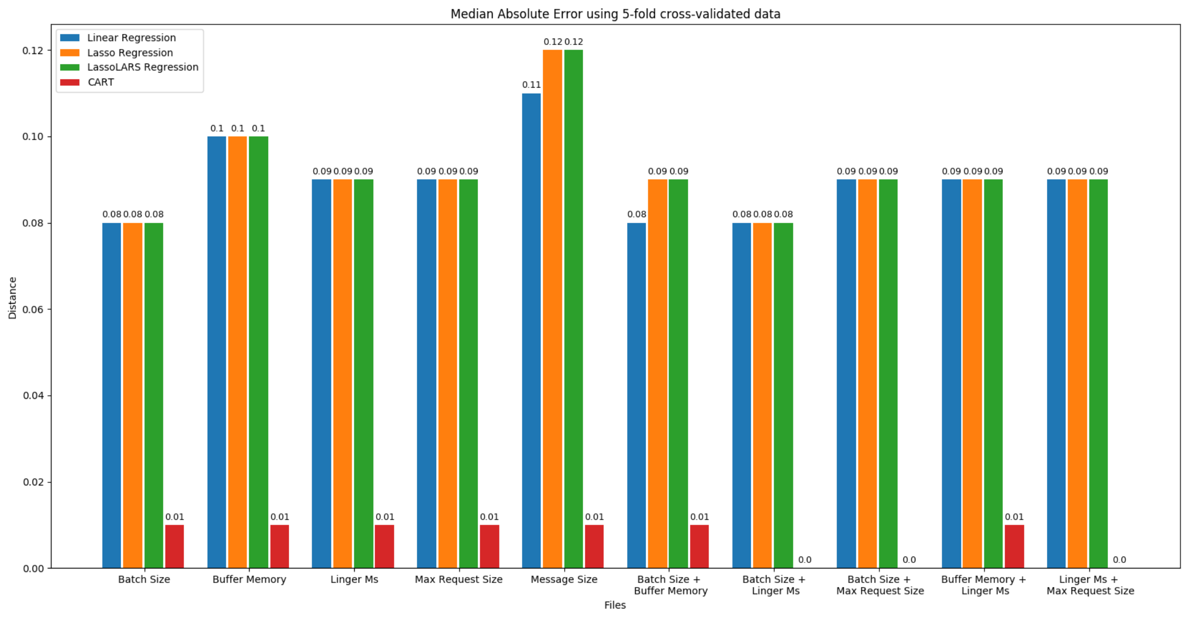


Figure 7.20: MB/sec predicting accuracy using Median Absolute Error metric with cross-validated data (lower is better)

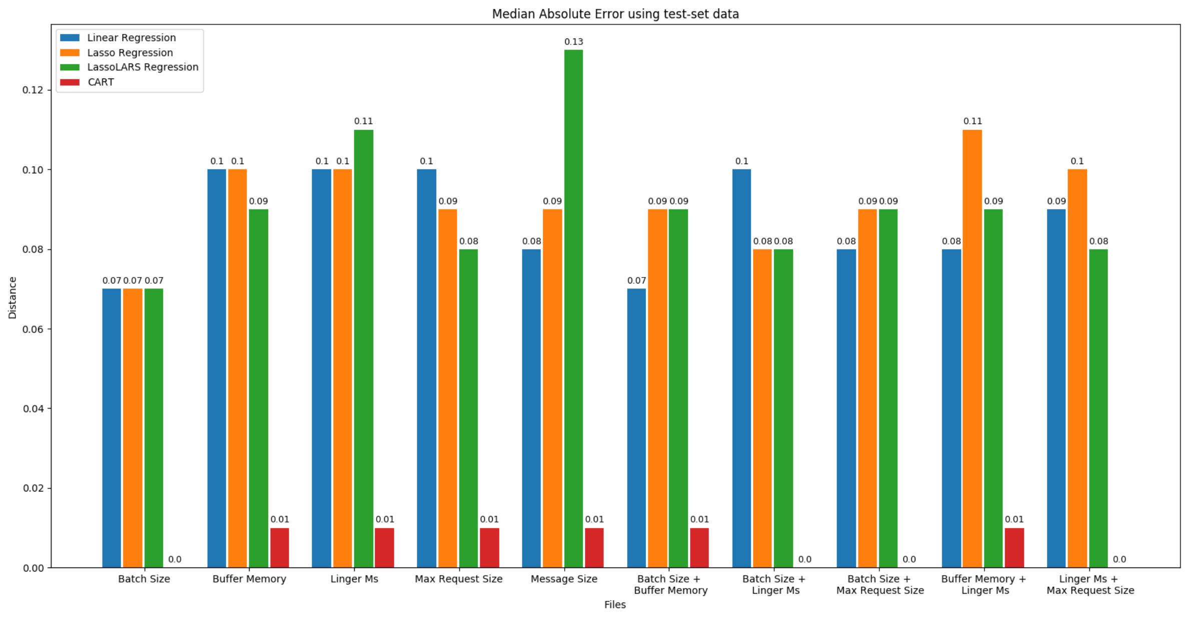
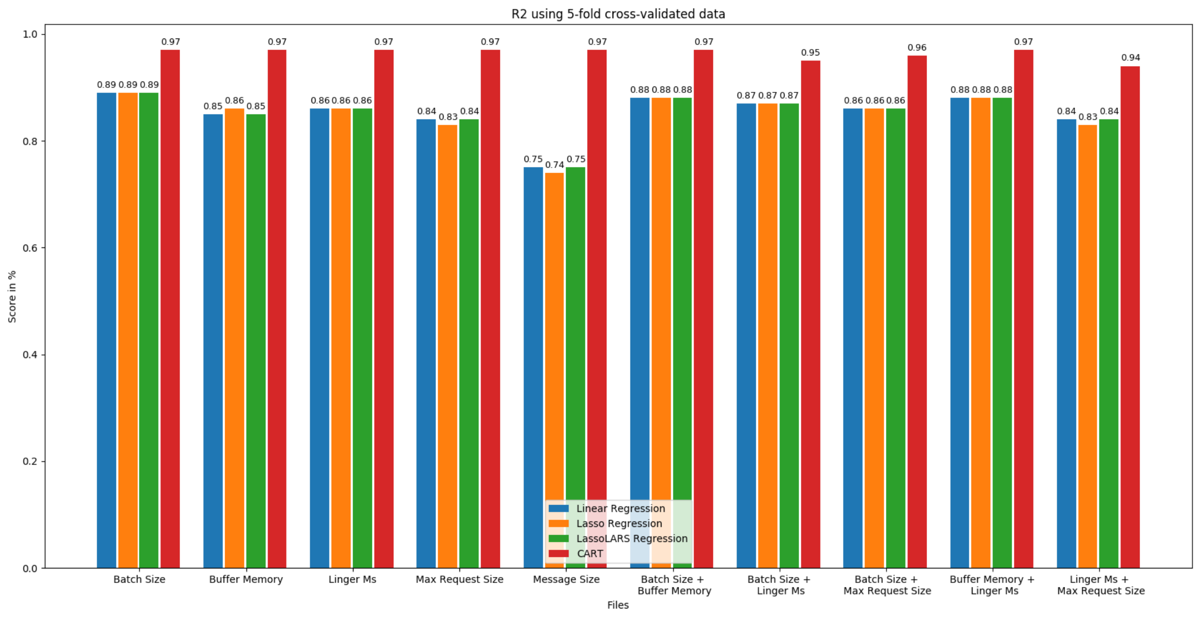


Figure 7.21: MB/sec predicting accuracy using Median Absolute Error metric with test set data (lower is better)

The algorithms yield relatively good results in each file. Once again, the CART algorithm, especially the Mean Squared Error metric, tends to generate the absolute result in the forecasts, in some cases. Moreover, the cross-validation technique in the three metrics, performs slightly better and yields ”smoother” results than the test set technique. We employ the term ”smoother” to explain that the transition, or difference, to values from algorithm to algorithm, and from metric to metric, does not feature many fluctuations in values.

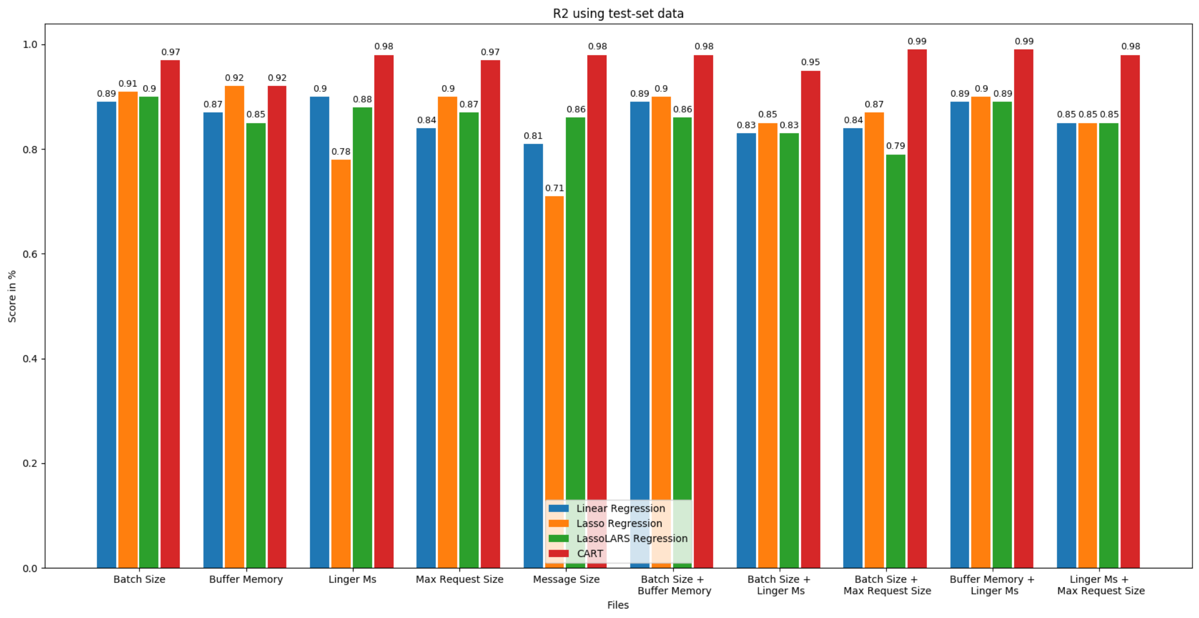
7.4.2.3 Avg Latency results

- $\boldsymbol{R^2}$:



Files	Linear Regression	Lasso Regression	LassoLARS Regression	CART
Batch Size	0.89	0.89	0.89	0.97
Buffer Memory	0.85	0.86	0.85	0.97
Linger Ms	0.86	0.86	0.86	0.97
Max Request Size	0.84	0.83	0.84	0.97
Message Size	0.75	0.74	0.75	0.97
Batch Size + Buffer Memory	0.88	0.88	0.88	0.97
Batch Size + Linger Ms	0.87	0.87	0.87	0.95
Batch Size + Max Request Size	0.86	0.86	0.86	0.96
Buffer Memory + Linger Ms	0.88	0.88	0.88	0.97
Linger Ms + Max Request Size	0.84	0.83	0.84	0.94

Figure 7.22: Avg Latency predicting accuracy using R^2 metric with cross-validated data (higher is better)



Files	Linear Regression	Lasso Regression	LassoLARS Regression	CART
Batch Size	0.89	0.91	0.9	0.97
Buffer Memory	0.87	0.92	0.85	0.92
Linger Ms	0.9	0.78	0.88	0.98
Max Request Size	0.84	0.9	0.87	0.97
Message Size	0.81	0.71	0.86	0.98
Batch Size + Buffer Memory	0.89	0.9	0.86	0.98
Batch Size + Linger Ms	0.83	0.85	0.83	0.95
Batch Size + Max Request Size	0.84	0.87	0.79	0.99
Buffer Memory + Linger Ms	0.89	0.9	0.89	0.99
Linger Ms + Max Request Size	0.85	0.85	0.85	0.98

Figure 7.23: Avg Latency predicting accuracy using R^2 metric with test set data (higher is better)

- **Explained Variance Score**:

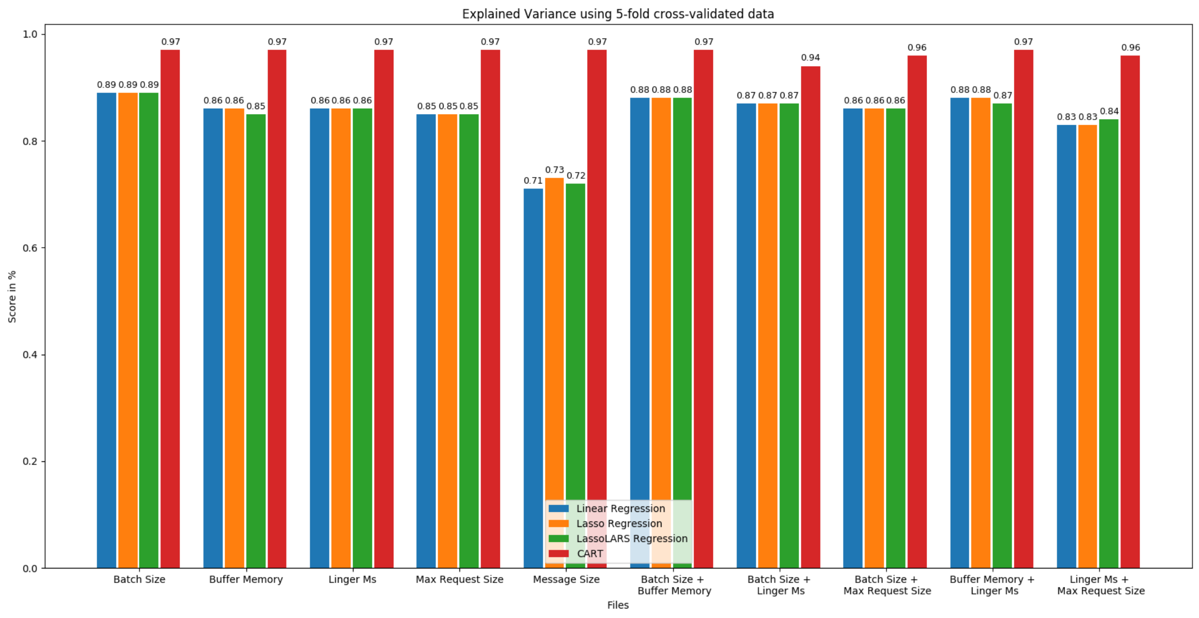


Figure 7.24: Avg Latency predicting accuracy using Explained Variance metric with cross-validated data (higher is better)

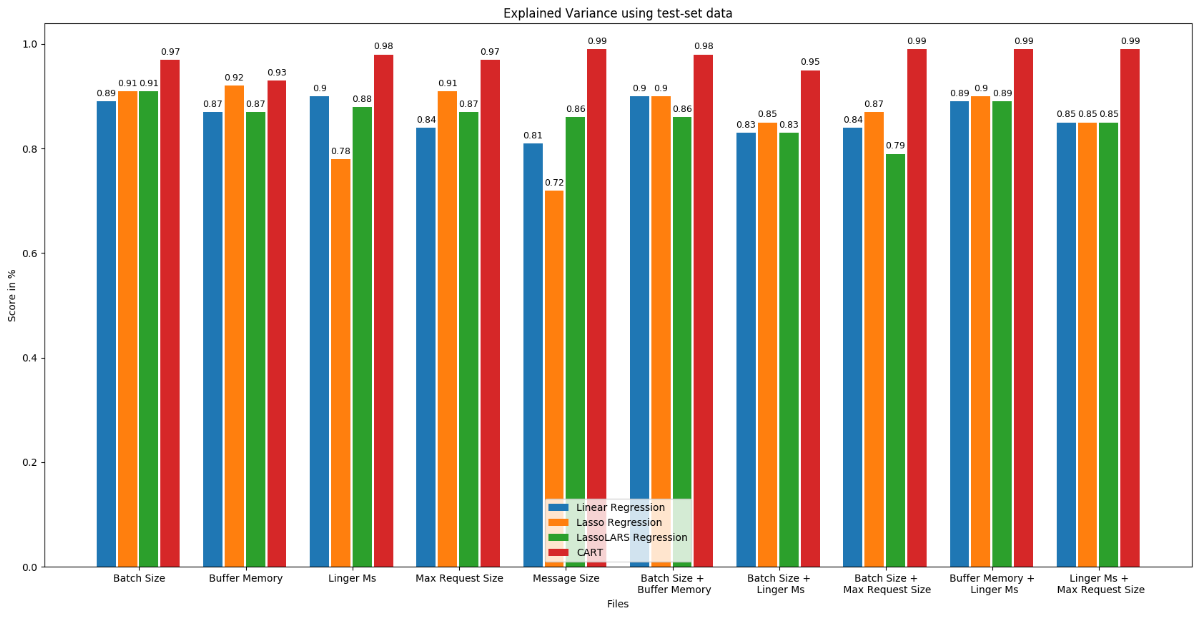


Figure 7.25: Avg Latency predicting accuracy using Explained Variance metric with test set data (higher is better)

For this target we detect that the results of the Linear Regression, Lasso Regression and LassoLARS Regression algorithms, are slightly lower than the previous two targets, ”Records/sec” and ”MB/sec”. Likewise, the results for the file ”Message Size” are, again, slightly worse than those of the other files. As mentioned in the previous targets, the cross-validation technique once again yields ”smoother” results than the use of test sets for prediction.

- **Mean Absolute Error (MAE)**:

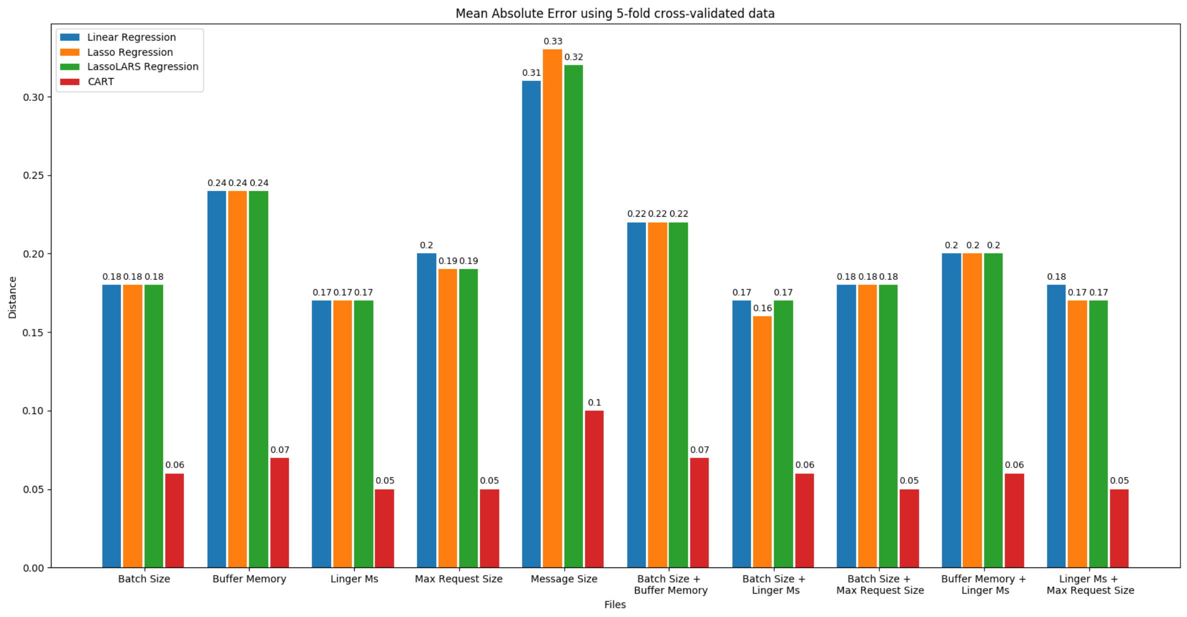


Figure 7.26: Avg Latency predicting accuracy using Mean Absolute Error metric with cross-validated data (lower is better)

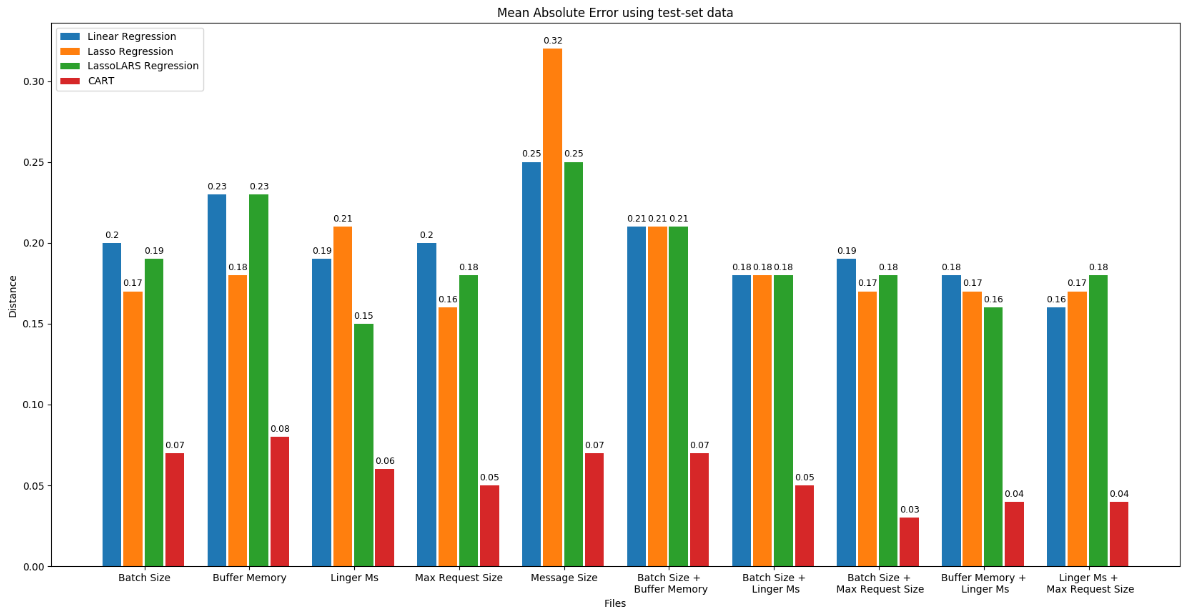


Figure 7.27: Avg Latency predicting accuracy using Mean Absolute Error metric with test set data (lower is better)

- **Mean Squared Error (MSE)**:

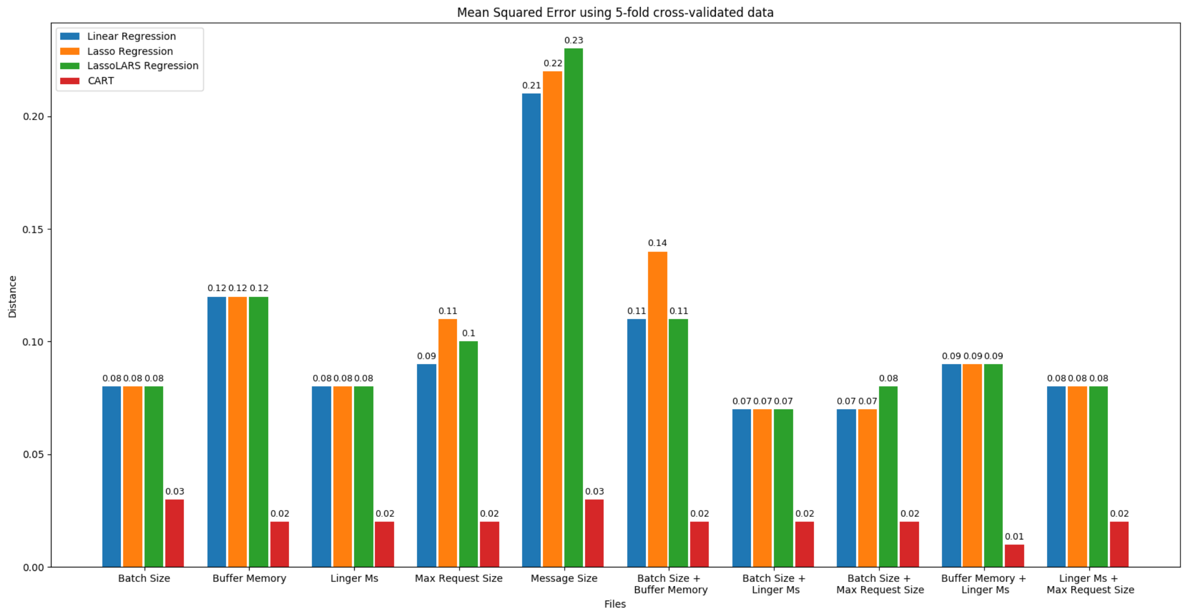


Figure 7.28: Avg Latency predicting accuracy using Mean Squared Error metric with cross-validated data (lower is better)

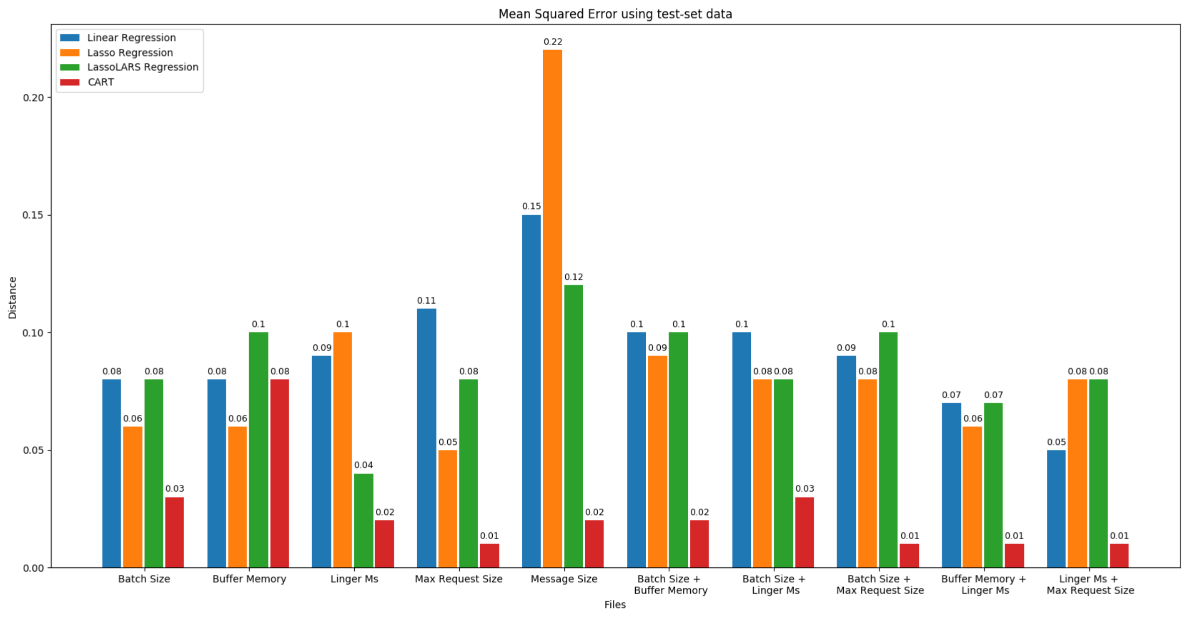


Figure 7.29: Avg Latency predicting accuracy using Mean Squared Error metric with test set data (lower is better)

- **Median Absolute Error (MedAE):**

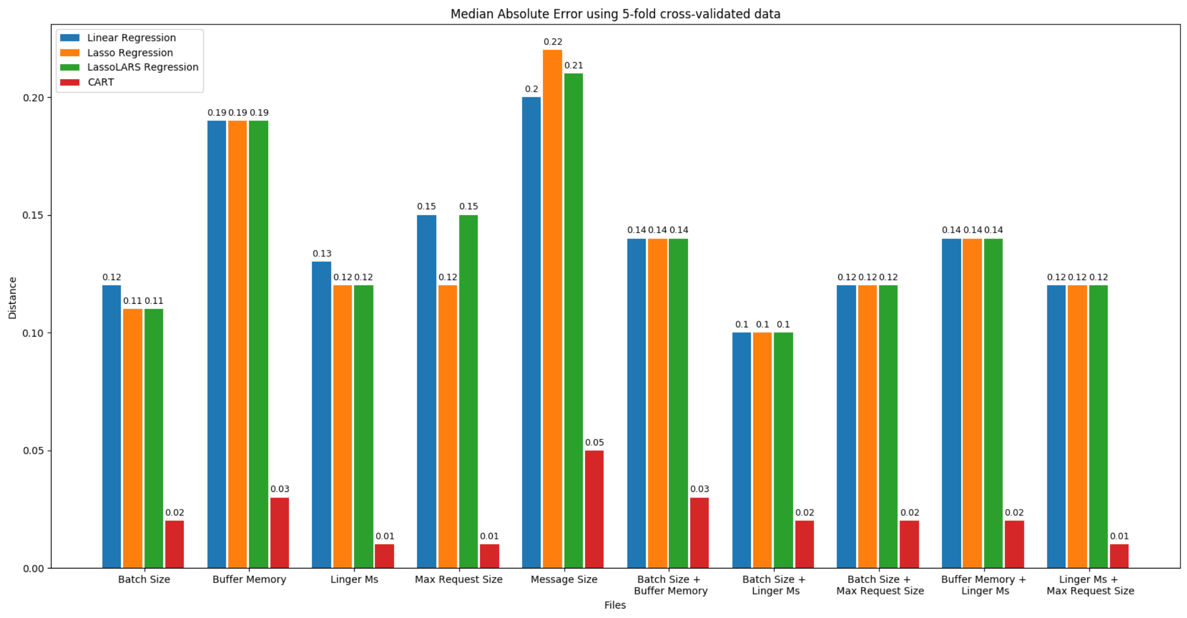


Figure 7.30: Avg Latency predicting accuracy using Median Absolute Error metric with cross-validated data (lower is better)

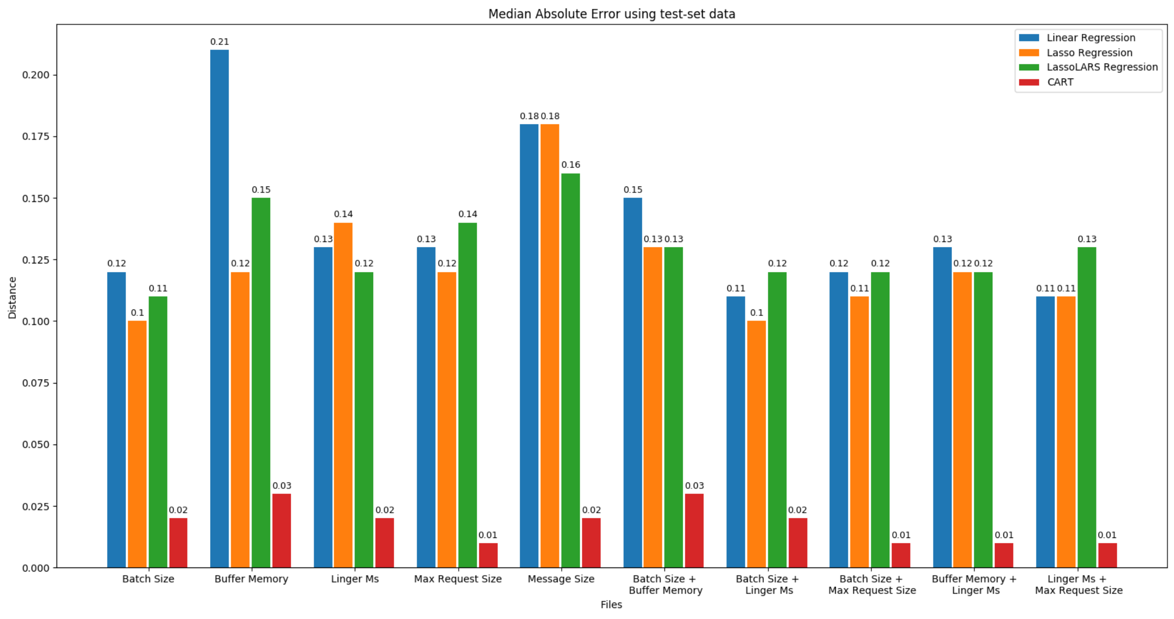


Figure 7.31: Avg Latency predicting accuracy using Median Absolute Error metric with test set data (lower is better)

The distances given by the forecasts of the Mean Absolute Error, Mean Squared Error and Median Absolute Error metrics, are slightly above the ”Avg Latency” target, in comparison to the previous two targets. This might indicate that the algorithms may not be as good at predicting the target ”Avg Latency”, at least when using these particular pieces of data.

7.4.2.4 Max Latency results

- $\boldsymbol{R^2}$:

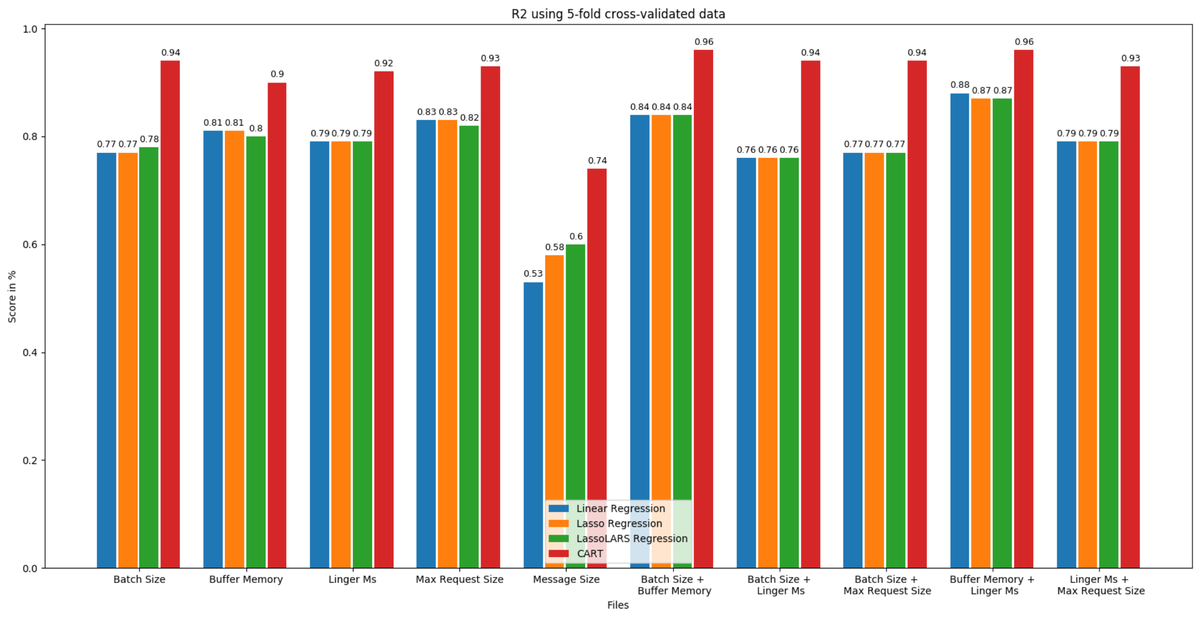


Figure 7.32: Max Latency predicting accuracy using R^2 metric with cross-validated data (higher is better)

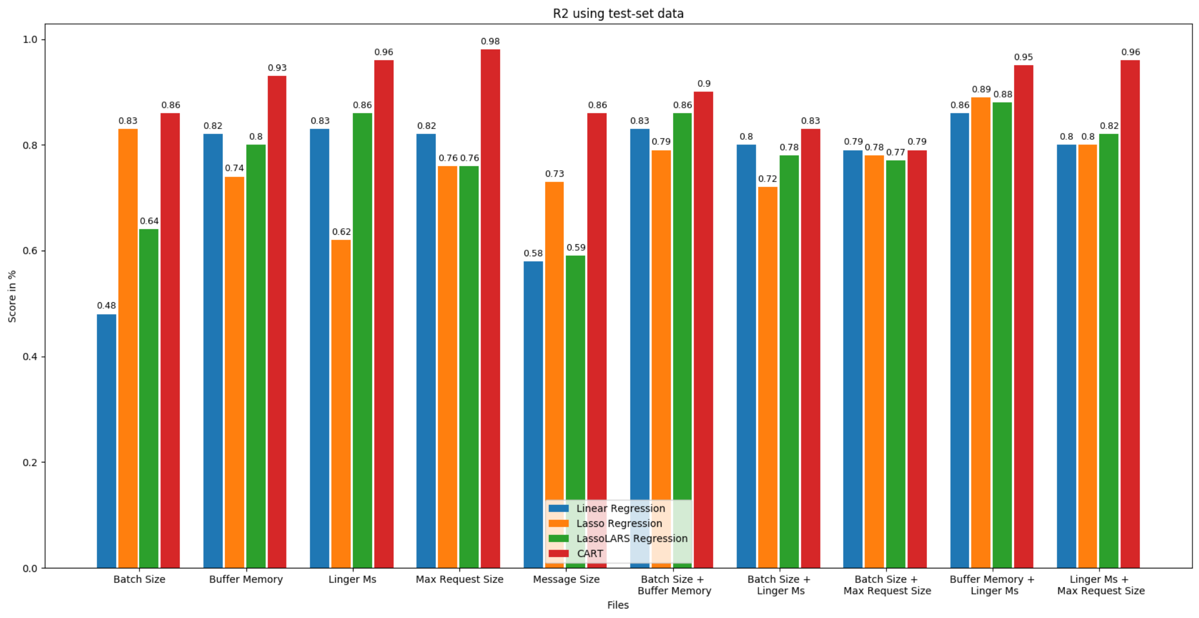


Figure 7.33: Max Latency predicting accuracy using R^2 metric with test set data (higher is better)

- **Explained Variance Score**:

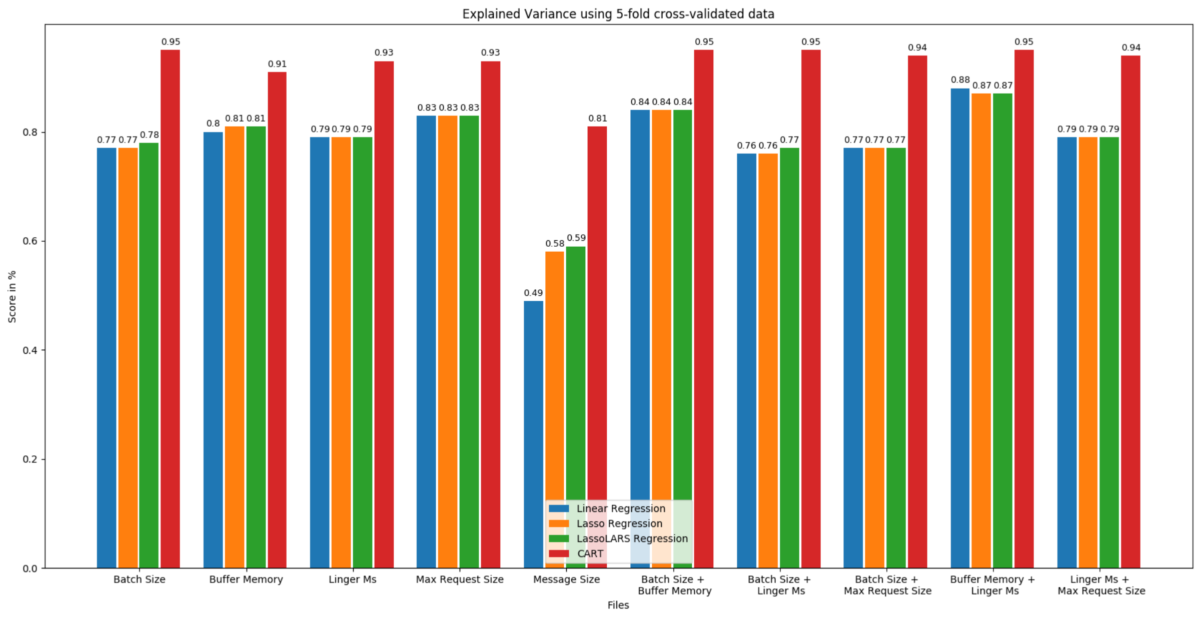


Figure 7.34: Max Latency predicting accuracy using Explained Variance metric with cross-validated data (higher is better)

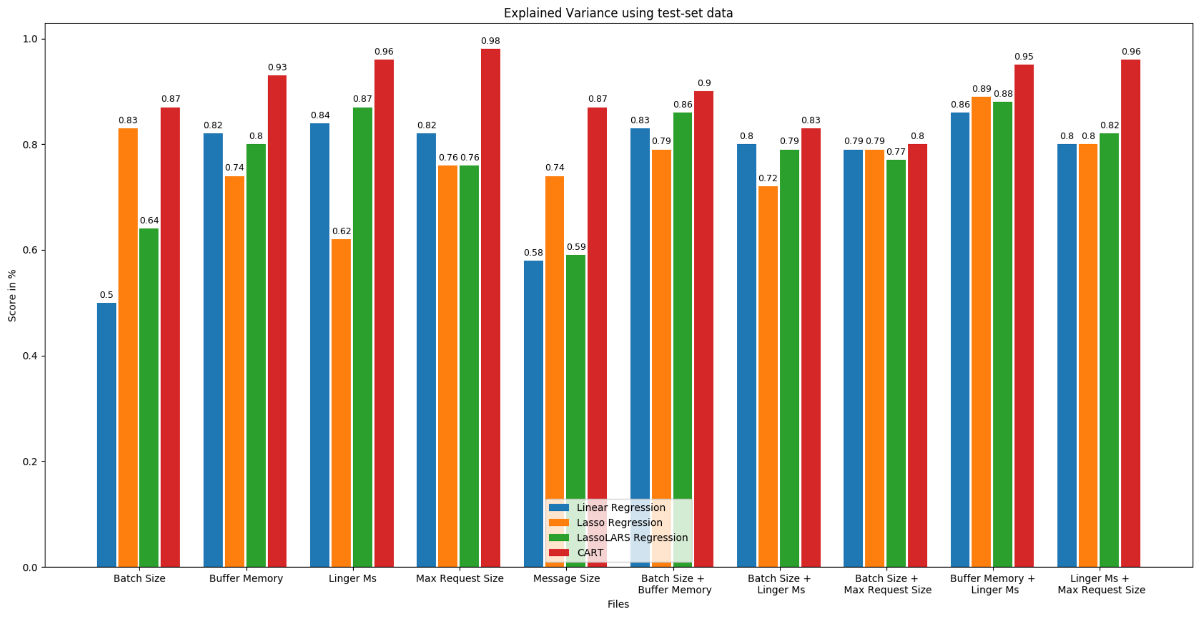


Figure 7.35: Max Latency predicting accuracy using Explained Variance metric with test set data (higher is better)

As is the case in the target ”Avg Latency”, so, too, in the ”Max Latency”, we acknowledge that the percentages of the algorithms are slightly lower than the first two targets, ”Records/sec” and ”MB/sec”. As this phenomenon occurs in the ”Avg Latency” target, it would stand to reason that this target would follow the same course as, in essence, the latter (”Max Latency”) constitutes an extreme case of the former.

- **Mean Absolute Error (MAE)**:

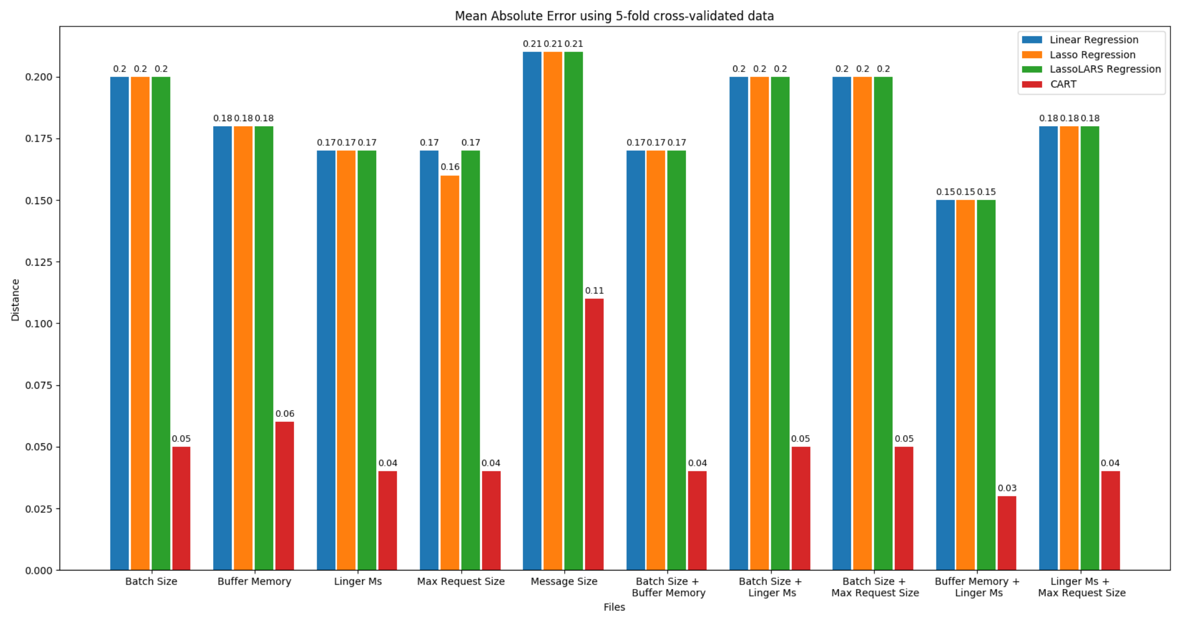


Figure 7.36: Max Latency predicting accuracy using Mean Absolute Error metric with cross-validated data (lower is better)

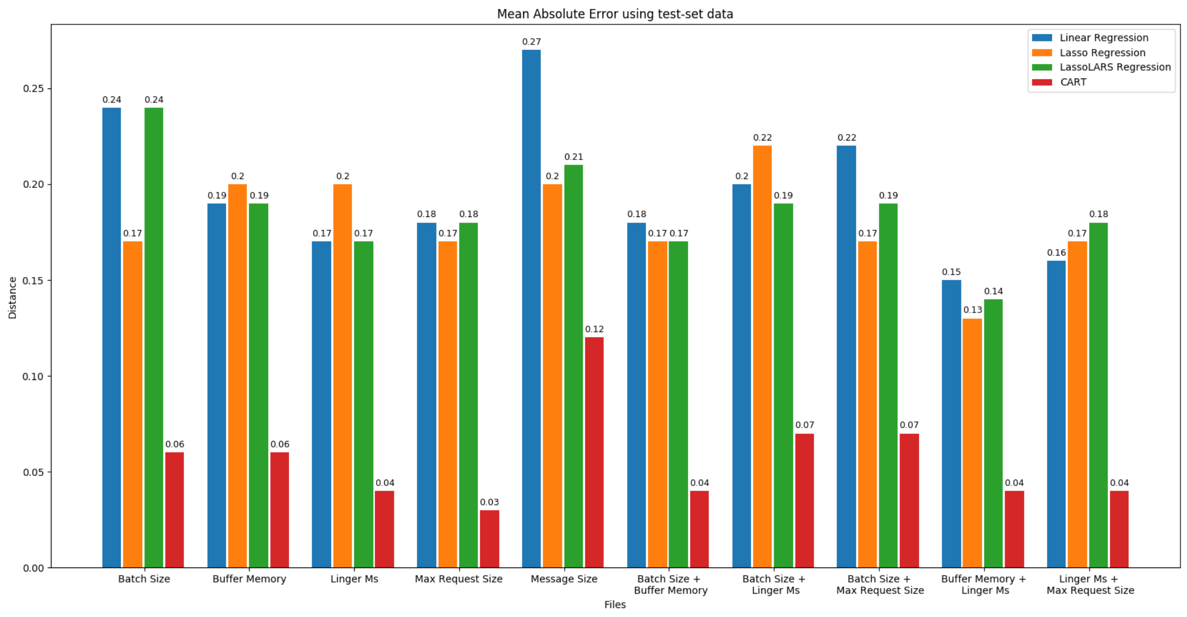


Figure 7.37: Max Latency predicting accuracy using Mean Absolute Error metric with test set data (lower is better)

- **Mean Squared Error (MSE)**:

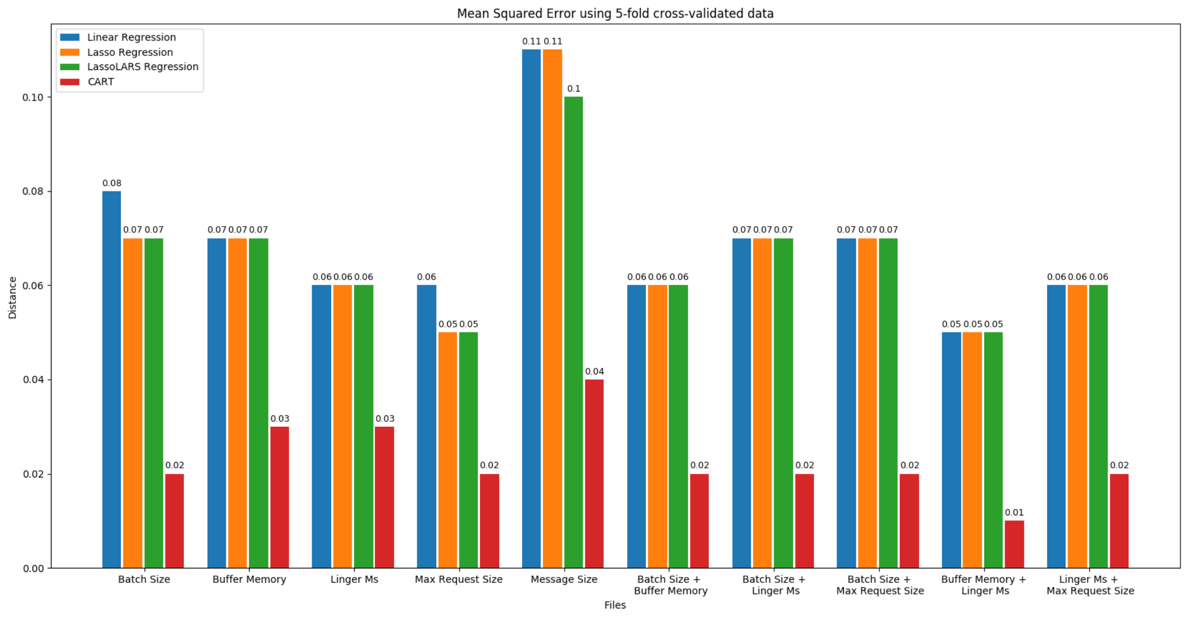


Figure 7.38: Max Latency predicting accuracy using Mean Squared Error metric with cross-validated data (lower is better)

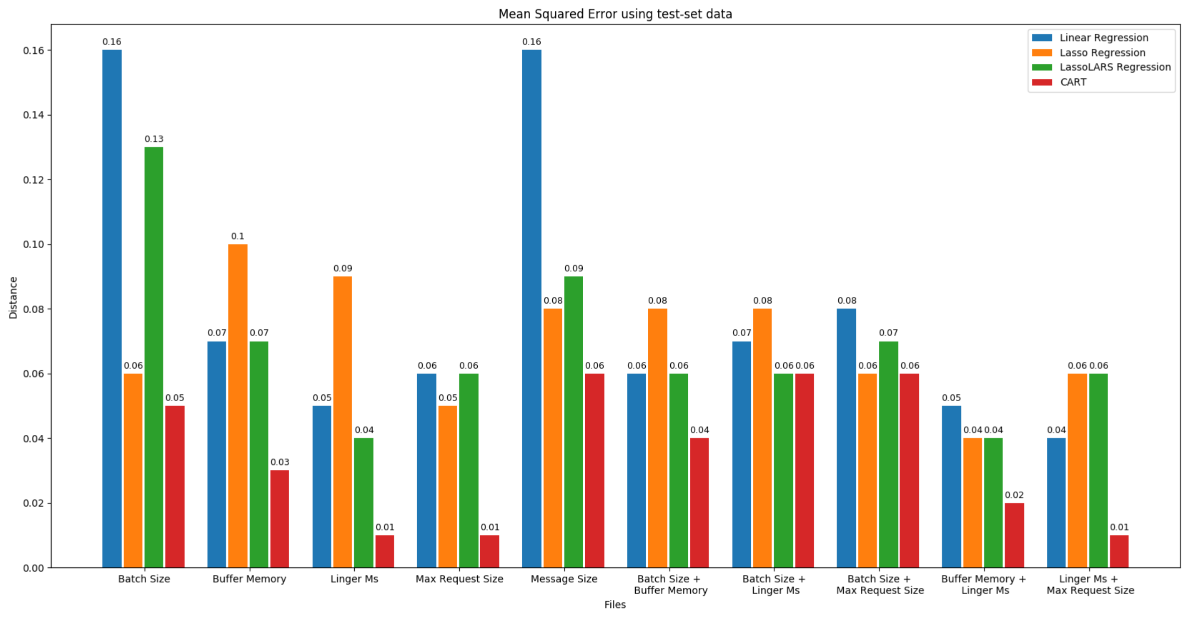


Figure 7.39: Max Latency predicting accuracy using Mean Squared Error metric with test set data (lower is better)

- **Median Absolute Error (MedAE)**:

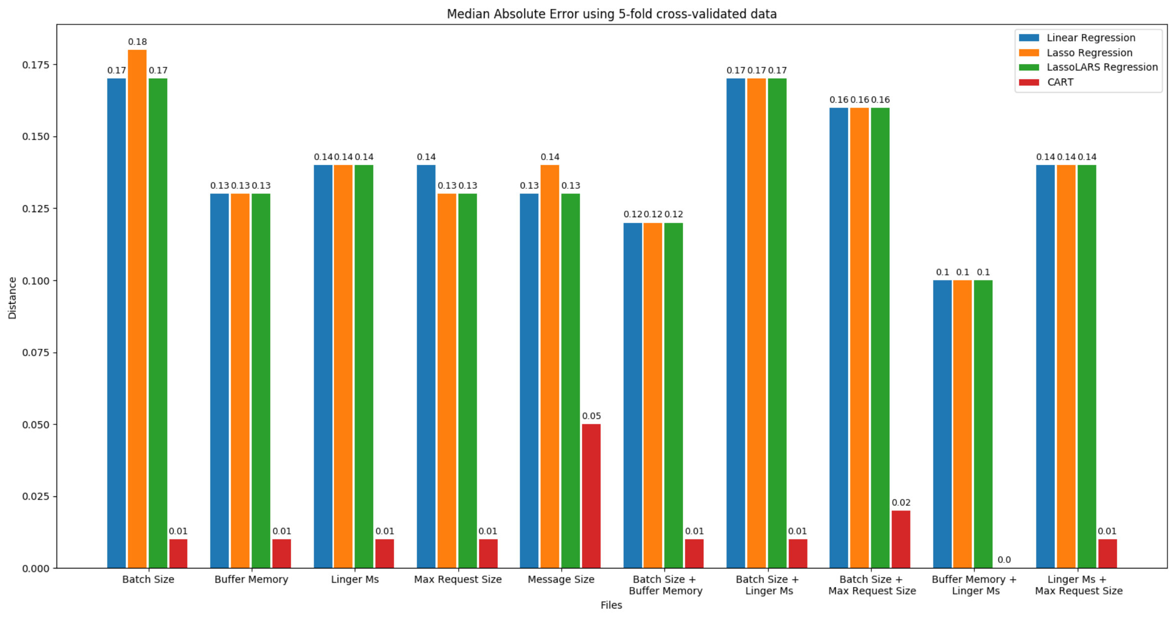


Figure 7.40: Max Latency predicting accuracy using Median Absolute Error metric with cross-validated data (lower is better)

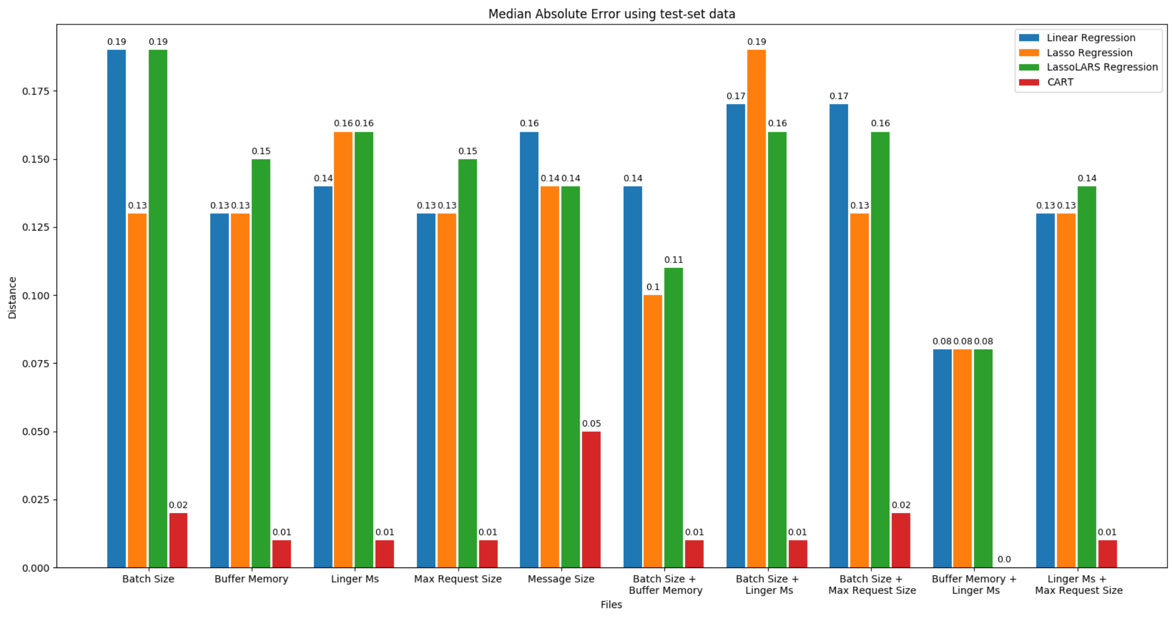


Figure 7.41: Max Latency predicting accuracy using Median Absolute Error metric with test set data (lower is better)

As observed with the metric R^2 and Explained Variance, where results are lower than the previous targets, this case moves along the same vein, with more pronounced results in the CART algorithm, which was the best in the previous targets by far. Of course, we should not forgo mentioning that the results with the use of the test set technique, display greater fluctuations per algorithm than what is the case with the cross-validation technique.

7.4.2.5 General Results for Regression algorithms

After a thorough look at the results of the Regression algorithms, we can conclude that they all produce fairly satisfactory results. The significant improvement came when the logarithmic function was applied to the data and, thus, allowed the algorithms to produce more accurate results. We note that some algorithms, like CART, closely provide the absolute value. We can observe, as mentioned previously, that the use of cross-validation provides smoother results between the algorithms, i.e. without significant differences between them.

The diagrams also showcase that the Regression algorithms are slightly better at predicting Kafka performance on the ”Records/Sec” and ”MB/sec” targets, as opposed to the ”Avg Latency” and ”Max Latency” targets. That is, they seem to be better able to predict the throughput of the system, particularly predicting the amount of information that passes through it, rather than predicting the delay resulting from transmission.

7.4.3 Classification Algorithms

As in Section 7.4.2, we will now present the results of the second type of algorithms we used, the classification algorithms. We will examine how well the algorithms we used can predict the corresponding target. In this type of algorithms, we did not need to apply the logarithmic function ($f(x) = \log_{10}(x)$) to the data, as the metrics did not measure distance, but whether the algorithm managed to classify the new data correctly.

Unlike Section 7.4.2, the results here are organized by algorithm. We considered that they can be presented better this way, due not only to the extent of the results, but also due to the nature of the algorithms. In each algorithm, the results are organized per the ”target” that we wanted to predict (”Records/sec”, ”MB/sec”, ”Avg latency” and ”Max latency”), while each ”target” is organized per metric so as to assess if each metric yields different results. We also noted that the results from the cross-validated data are almost identical to the results from the use of test datasets (20% of the whole dataset) for predictions. As such, in order to make the thesis more concise and comprehensible, we did not include the results from the use of test datasets. Furthermore, as highlighted in Section 6.3, the data had to be separated into buckets, due to their continuous nature, so that we could apply classification algorithms on them. Therefore, in each file (i.e. in each diagram), we observe how the algorithm handles a varying number of buckets.

Each algorithm we will present had hyperparameters that we learned—i.e. we found the best value for our data through exhaustive search, or, as we saw in Chapter 7.3.1, it is called *Grid Search*. That is why we will present and chart with the values of these parameters per file, as well as per number of buckets. Finally, because we noticed that the different metrics did not affect the value of the hyperparameters—i.e. they were identical for each metric—the values were different for each target. As a consequence, no diagram, but one for each target, has been added for each metric of the targets.

7.4.3.1 k-NN algorithm

Records/sec:

- **Accuracy**:

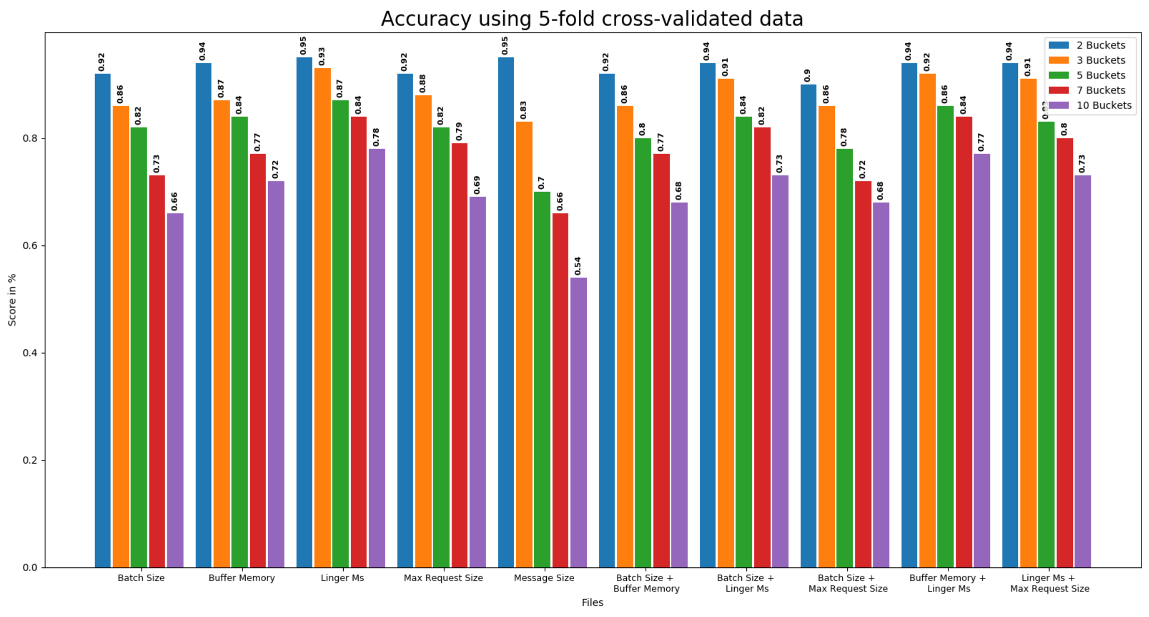


Figure 7.42: Records/sec predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

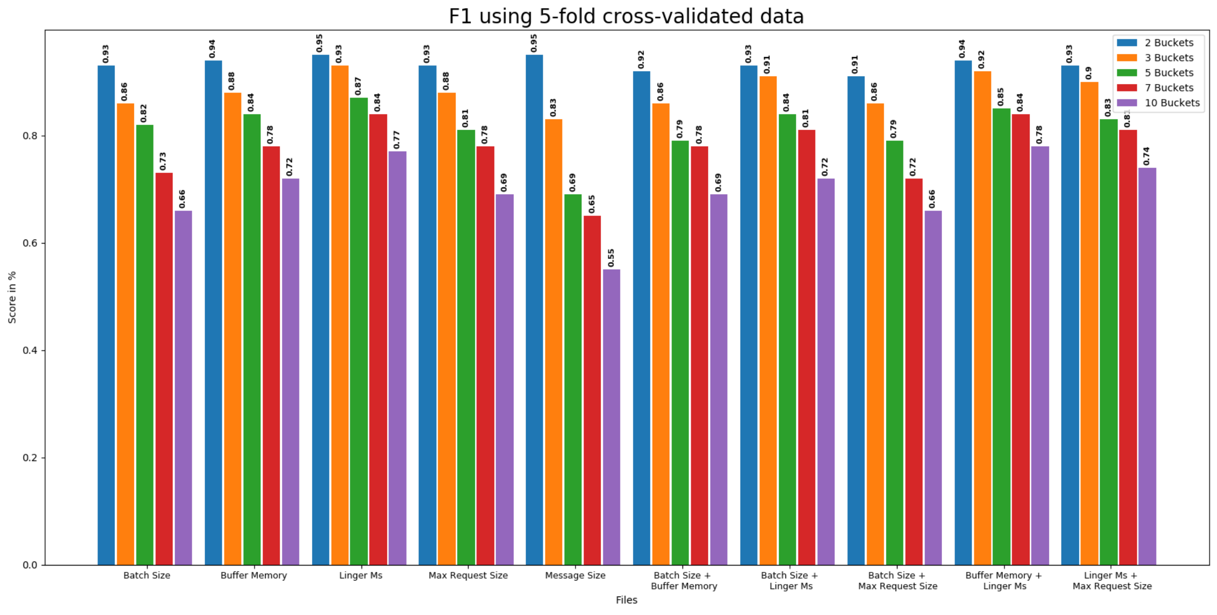


Figure 7.43: Records/sec predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

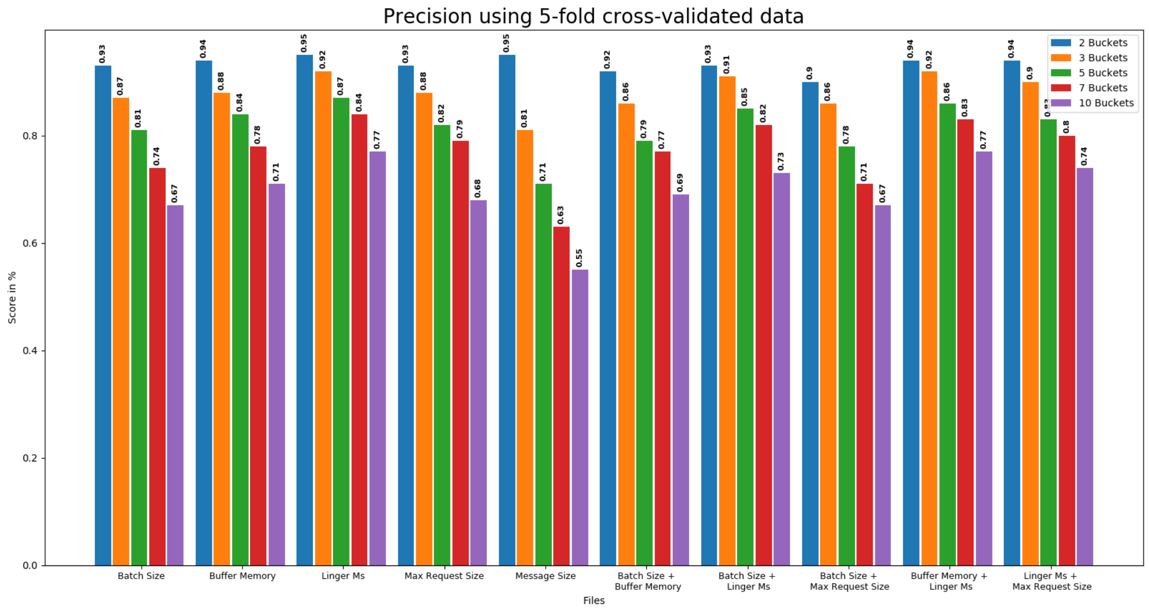


Figure 7.44: Records/sec predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

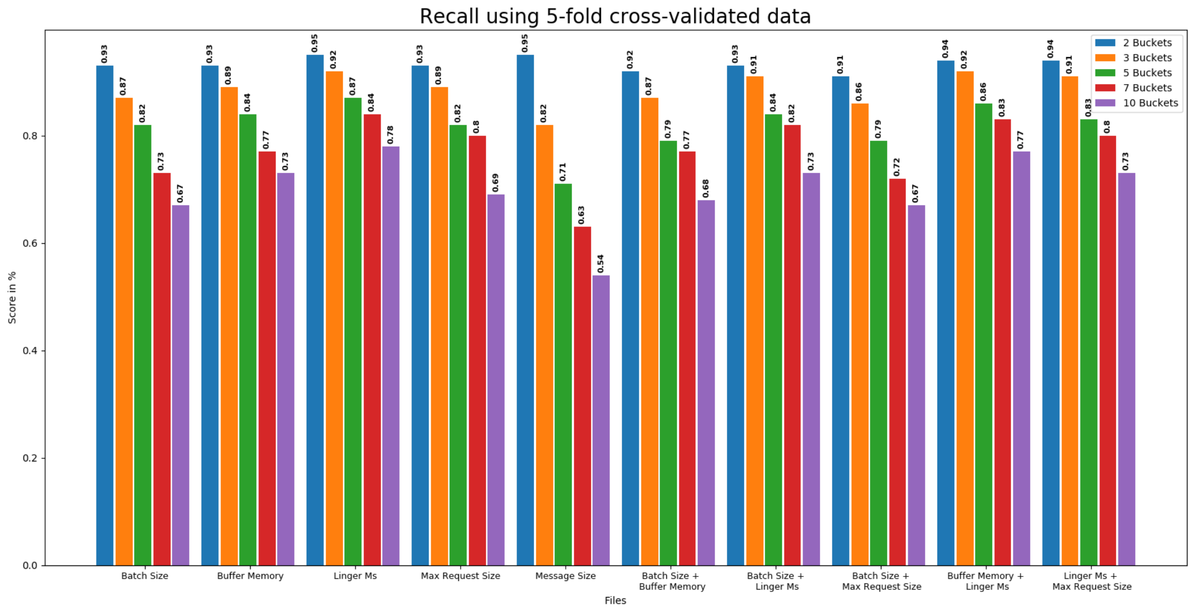


Figure 7.45: Records/sec predicting accuracy using recall metric with cross-validated data (higher is better)

In the above Figures that all metrics have similar results. Additionally, for each file there is close to no difference in each bucket, which indicates that the variables we modify in each file do not have a significant impact in the system. Moreover, as the number of buckets increases, the success rate decreases by a percentage that is almost constant. This may be interpreted as the fact that there is no good separation of data in the extreme values of the buckets and, thus, the performance of the algorithm decreases with the increase of buckets.

- **Neighbors**:

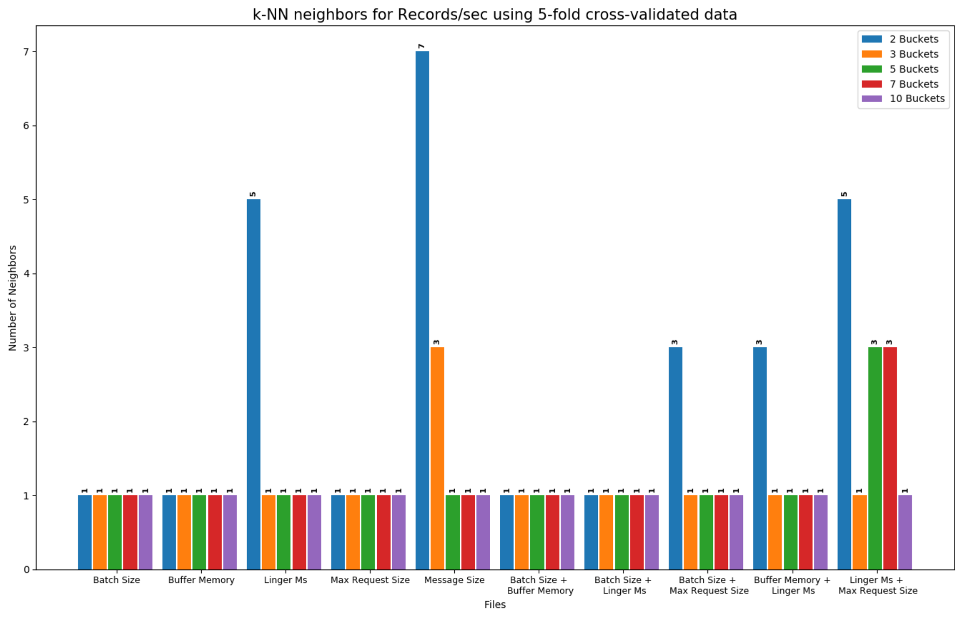


Figure 7.46: Parameter Neighbors for k-NN algorithm with ”Records/sec” predicted target

The number of neighbors was the same for each metric, which still remains stable for almost every case/file. However, we observe some steep climbs that may stem from the randomness with which the data is fed into the algorithm; due to their shuffling, in essence.

MB/sec:

- **Accuracy**:

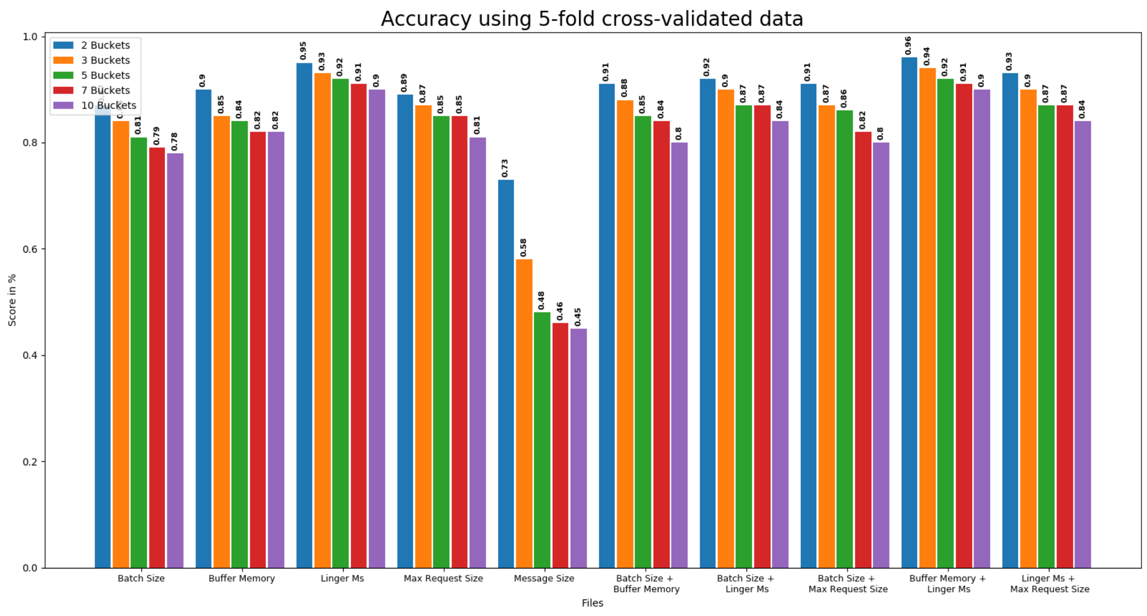


Figure 7.47: MB/sec predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

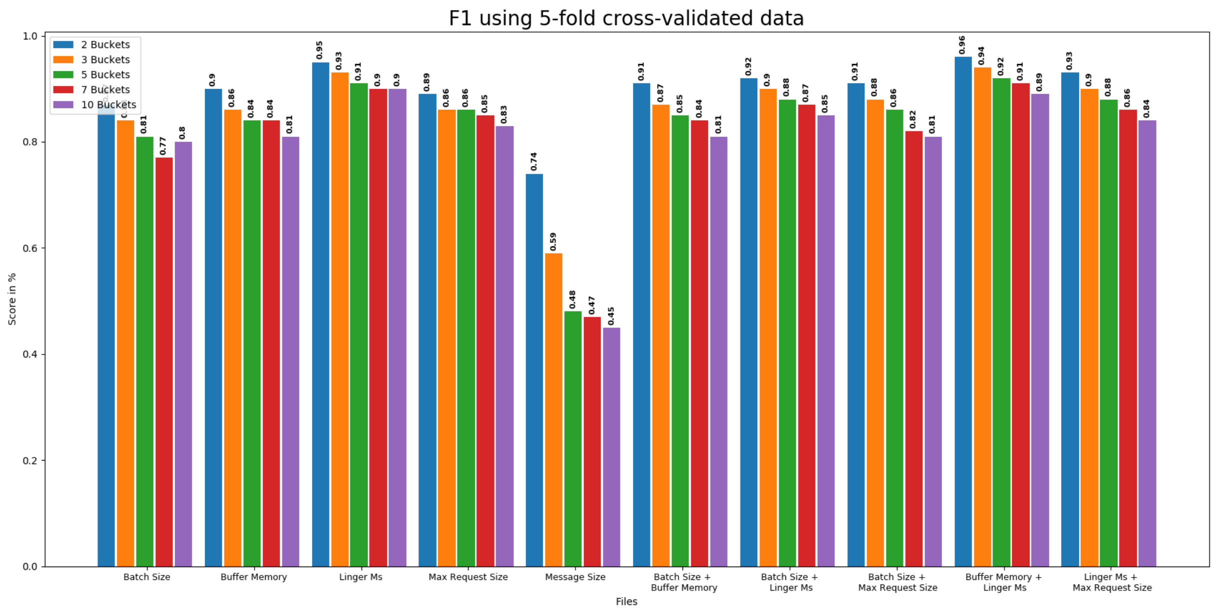


Figure 7.48: MB/sec predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

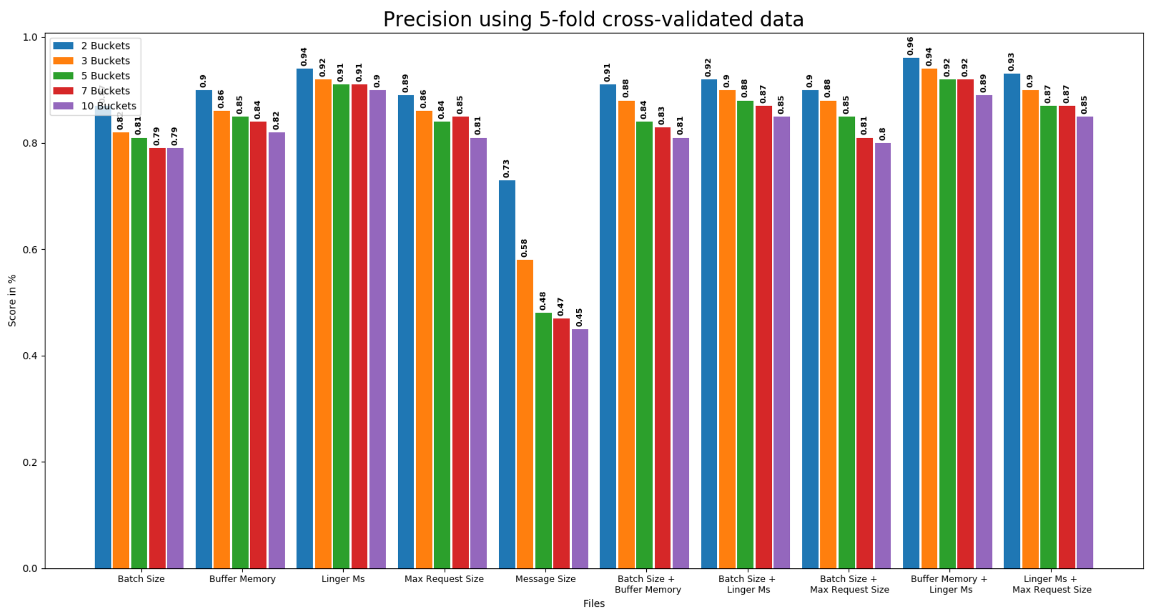


Figure 7.49: MB/sec predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

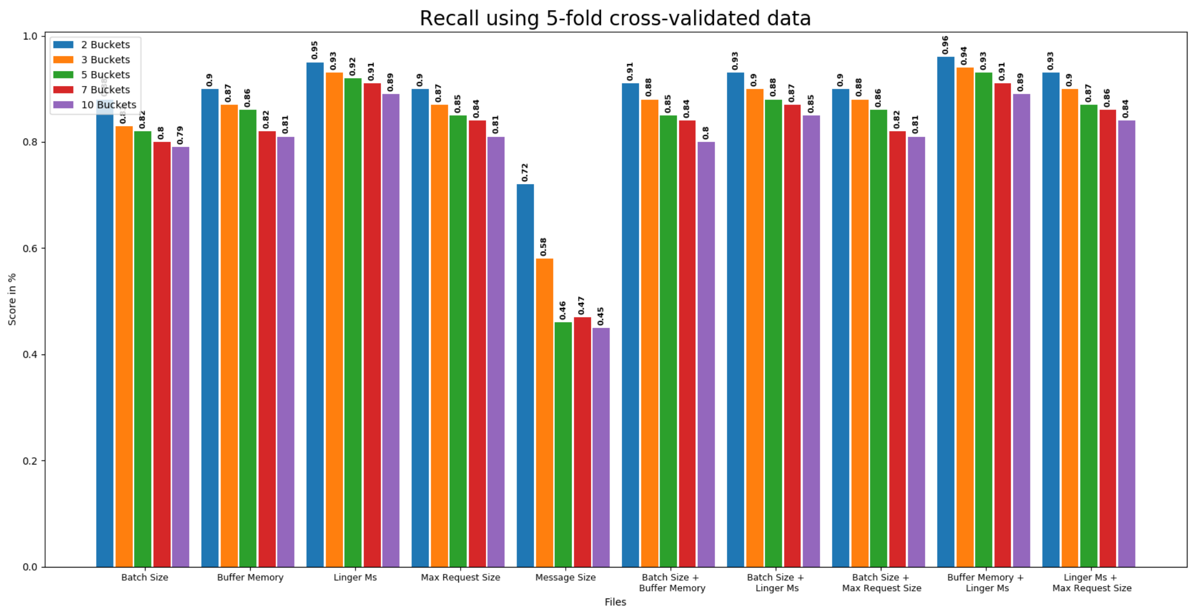


Figure 7.50: MB/sec predicting accuracy using recall metric with cross-validated data (higher is better)

As with the target ”Records/sec”, at ”MB/sec” we detect that the results in each metric and file are nearly identical. However, contrary to the target ”Records/sec”, the increase of buckets in each file does not result in a significant decrease in the performance of the algorithm. Perhaps the data in the buckets for this purpose, have clearer limits. Nevertheless, we do observe a decrease of the efficiency of the order of 45% in the file where we study the size of the message in relation to the target ”MB/sec”.

- **Neighbors**:

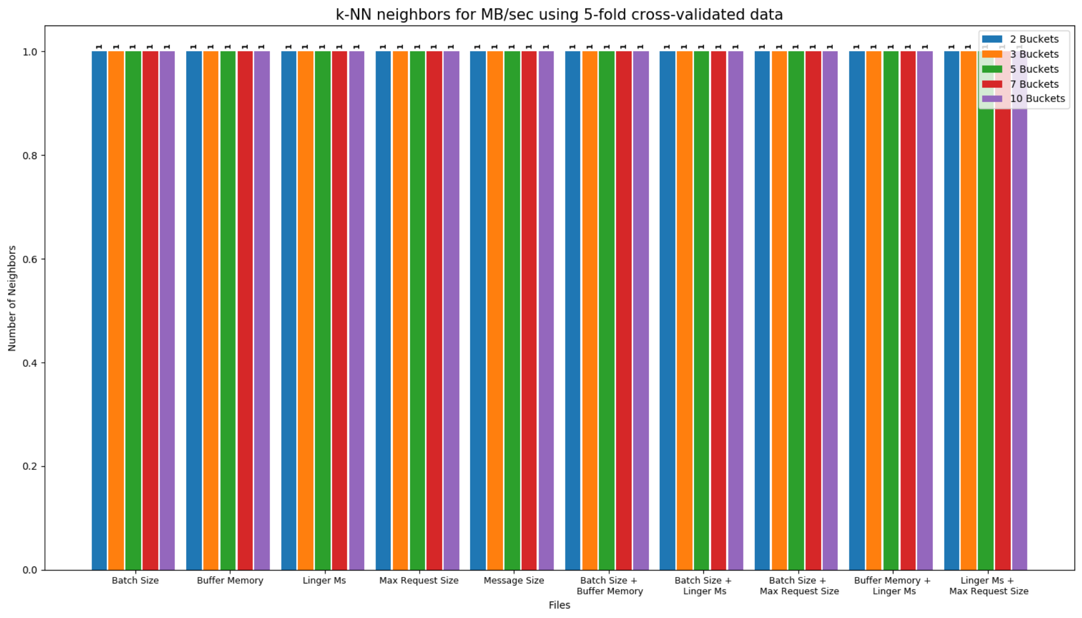


Figure 7.51: Parameter Neighbors for k-NN algorithm with ”MB/sec” predicted target

As was the case before, the number of optimal neighbors remained the same in each metric. We also acknowledge that it remains constant in each file, as well as for each number of buckets.

Avg Latency:

- **Accuracy**:

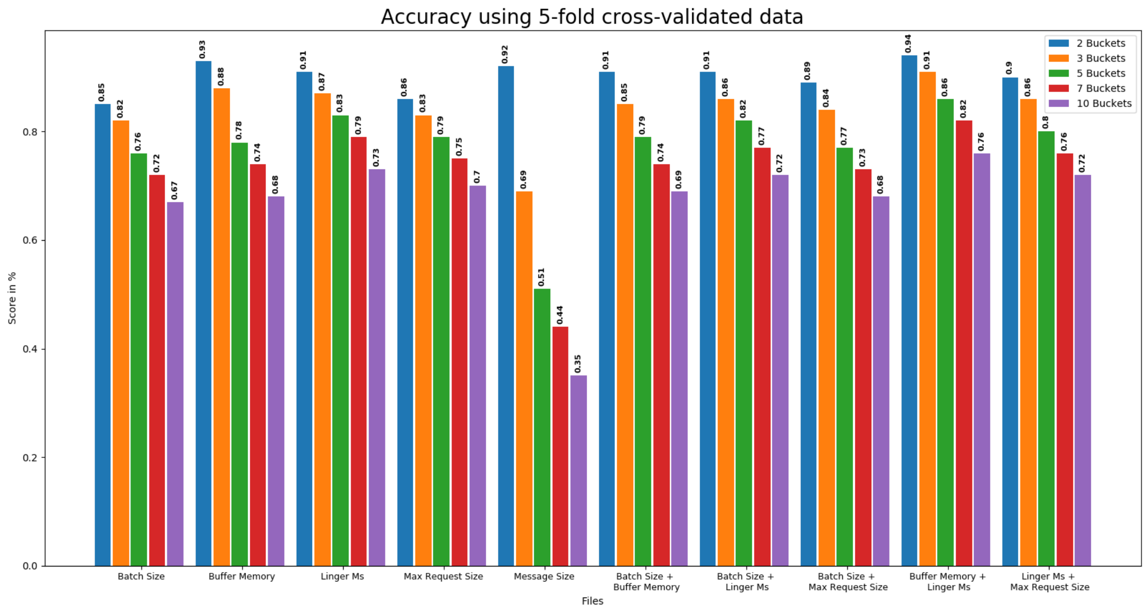


Figure 7.52: Avg Latency predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

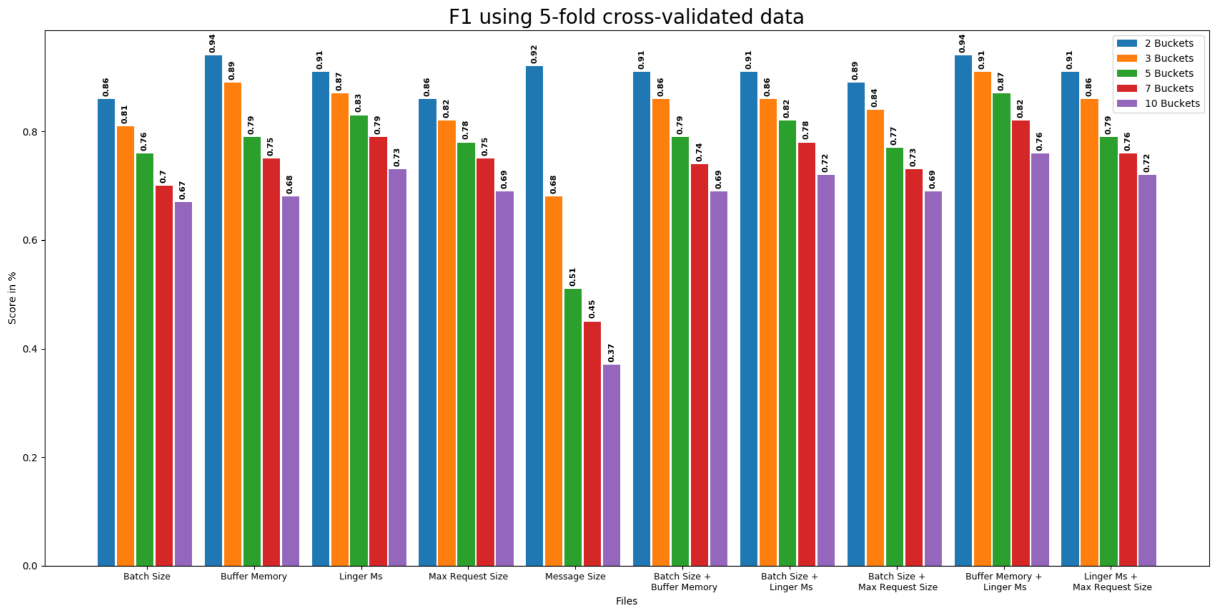


Figure 7.53: Avg Latency predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

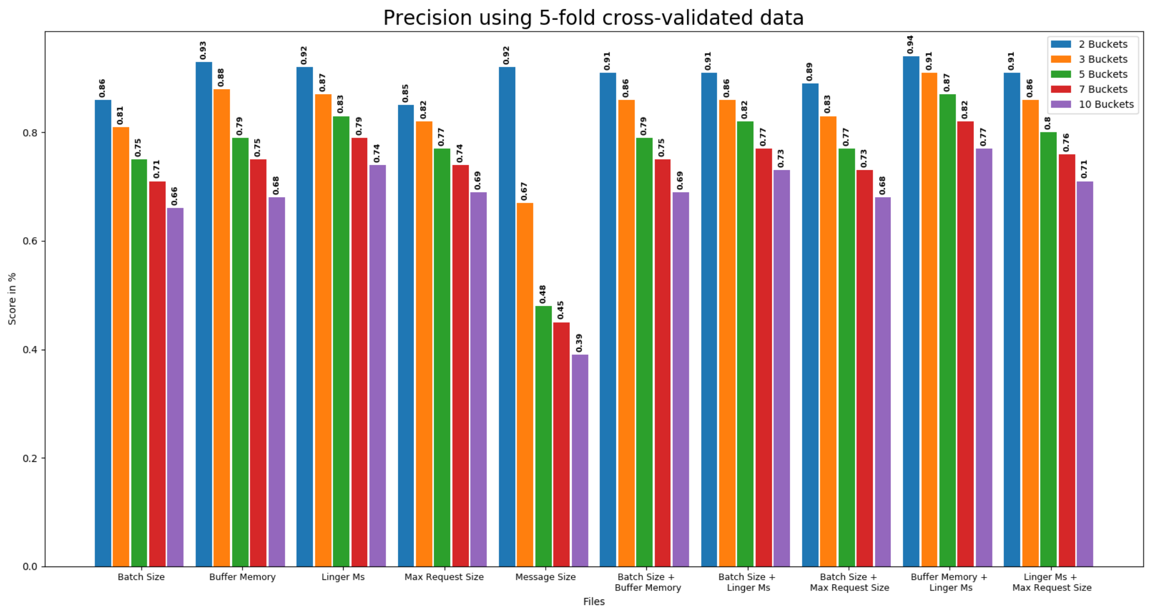


Figure 7.54: Avg Latency predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

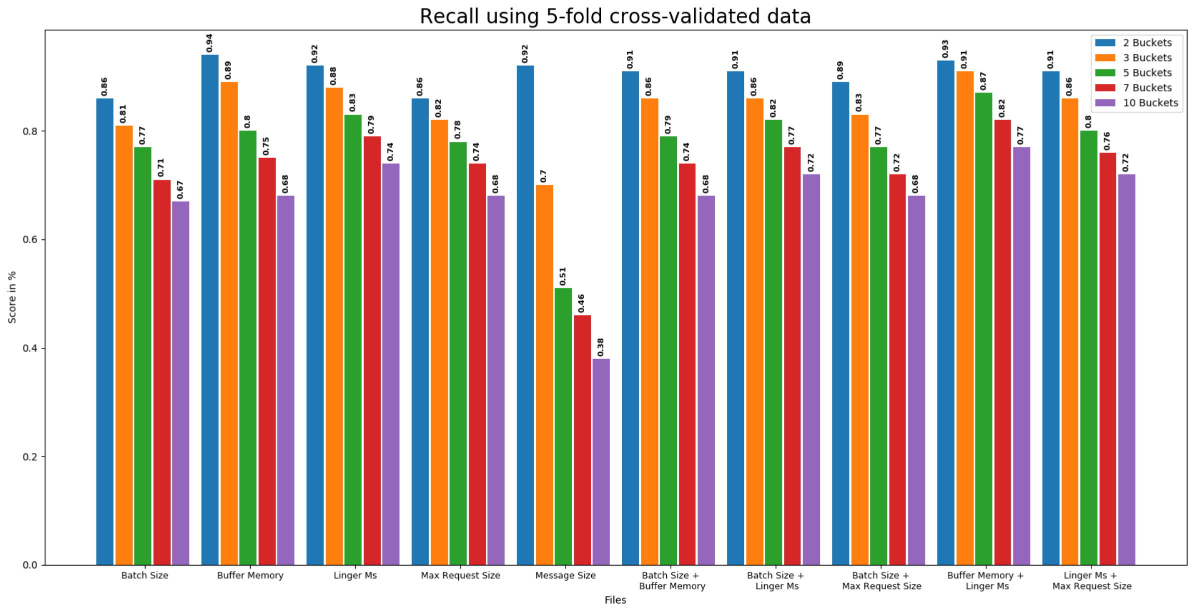


Figure 7.55: Avg Latency predicting accuracy using recall metric with cross-validated data (higher is better)

Once again, we take notice of a gradual decrease in values as the number of buckets for classification in each file increases. We further detect a lower success rate in the prediction, in terms of the file we are looking at the size of the messages for.

- **Neighbors**:

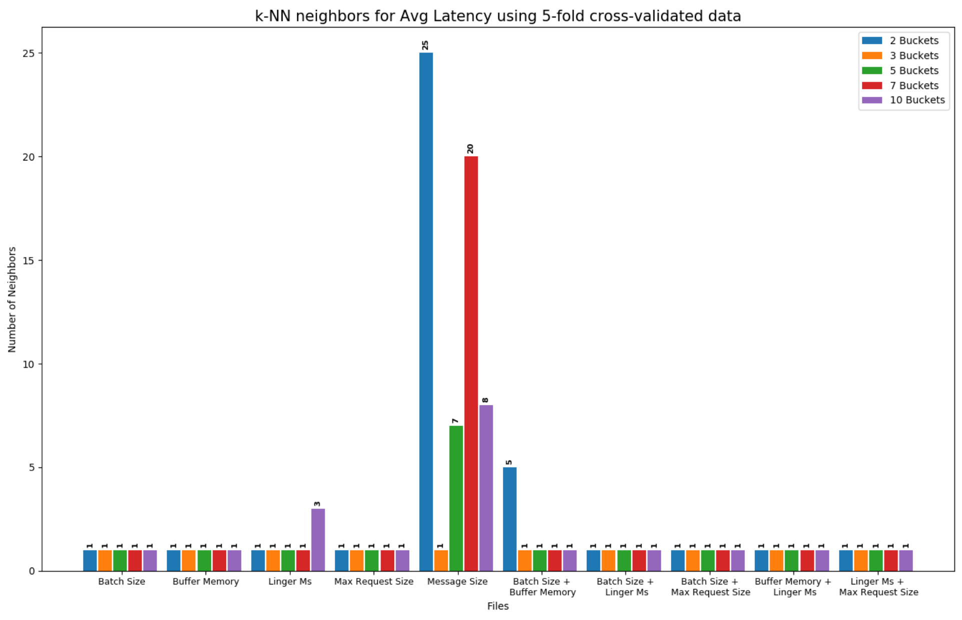


Figure 7.56: Parameter Neighbors for k-NN algorithm with ”Avg Latency” predicted target

The number of neighbors typically fluctuates at constant levels and has a general value of 1; with the ”Message Size” file being an exception to the rule due to its varying values.

Max Latency:

- **Accuracy**:

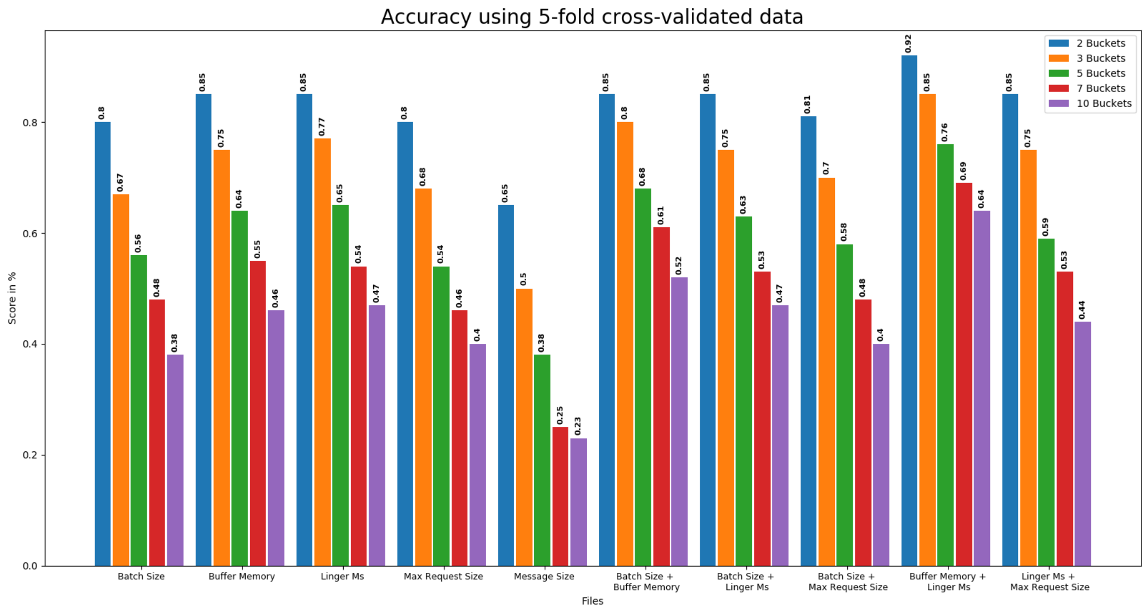


Figure 7.57: Max Latency predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

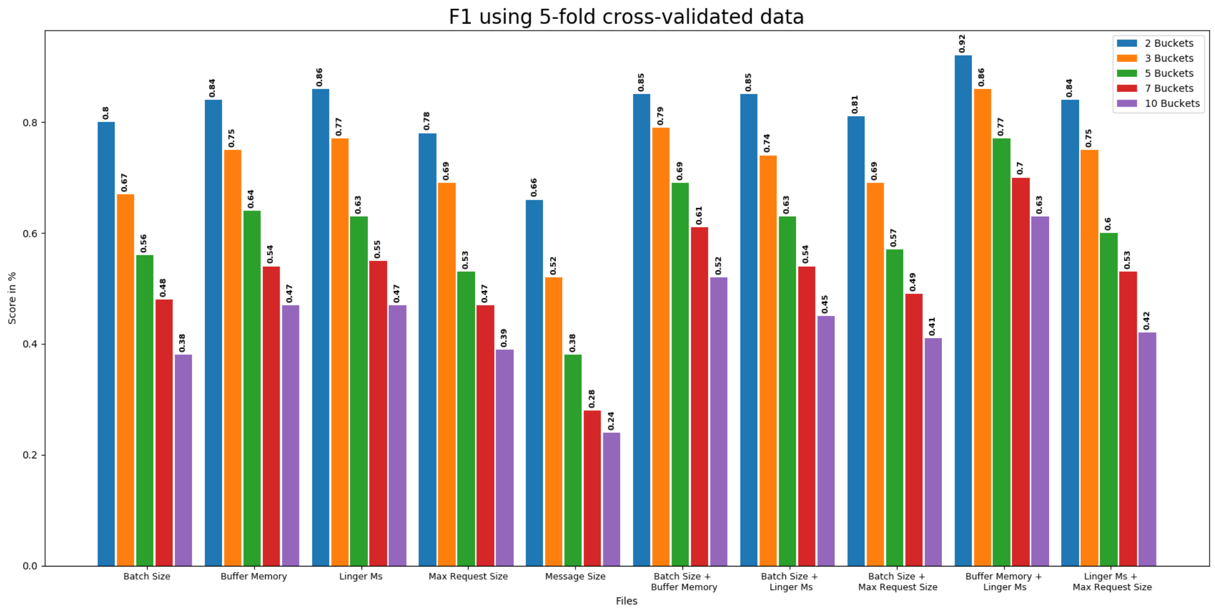


Figure 7.58: Max Latency predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

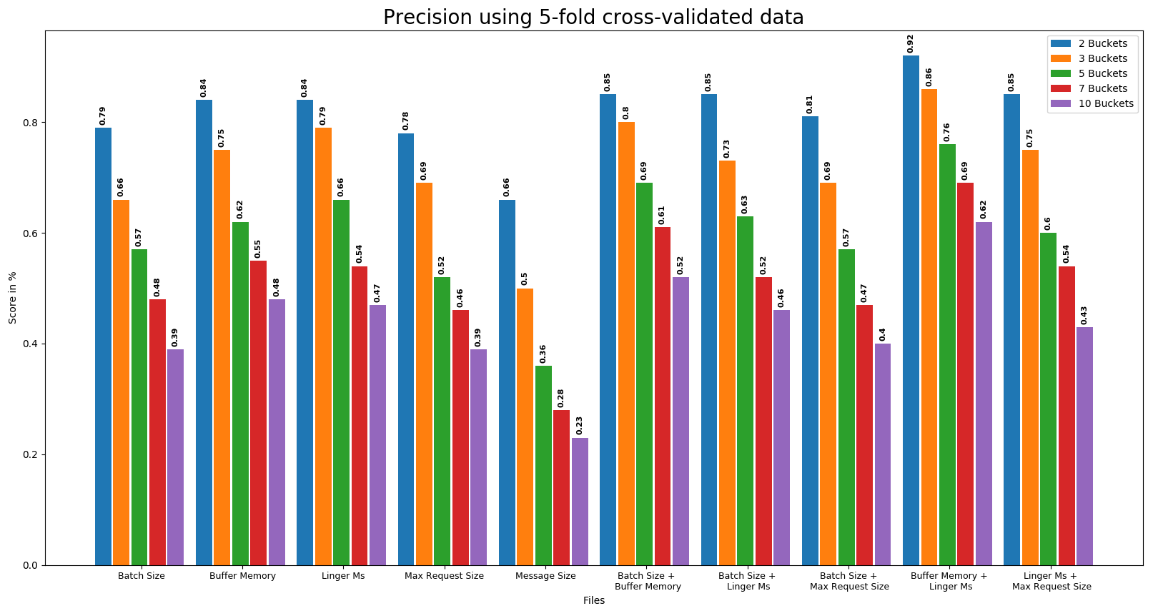


Figure 7.59: Max Latency predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

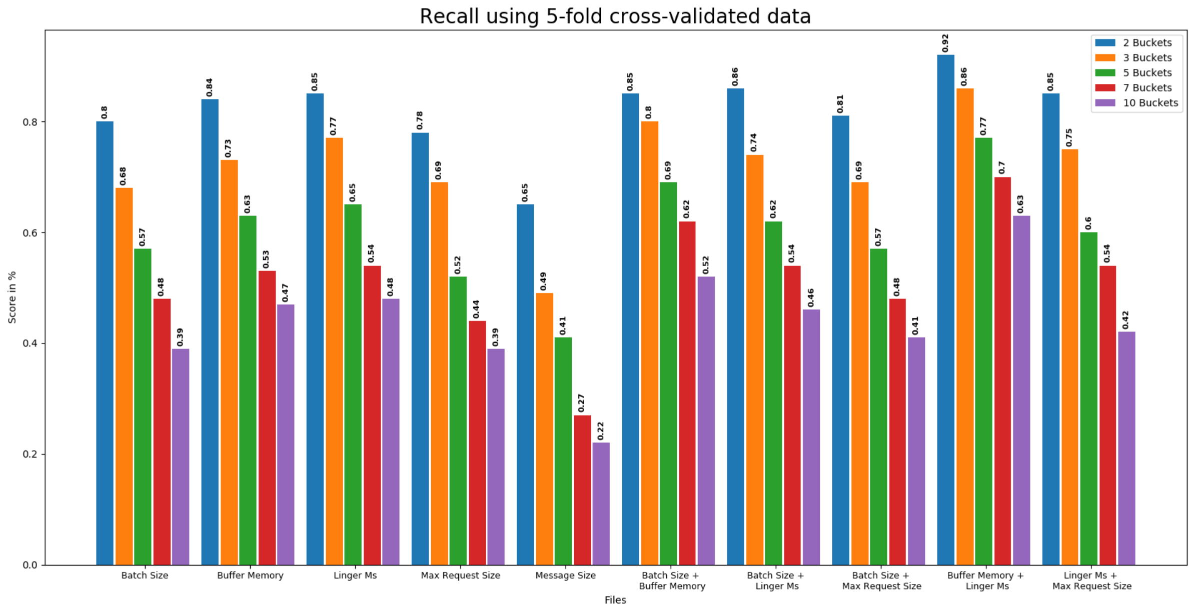


Figure 7.60: Max Latency predicting accuracy using recall metric with cross-validated data (higher is better)

As regards the gradual decrease of values, alongside the increase of the buckets—both of which we discussed in the previous targets—we note that, in the ”Max Latency” target, this decrease is more pronounced. In addition, the success rates of the algorithm are lower than those of the previous targets. However, once again, we notice that the percentages of the file ”Message Size” are the smallest, which means it may not be possible to make a correct prediction given the size of the message.

- **Neighbors**:

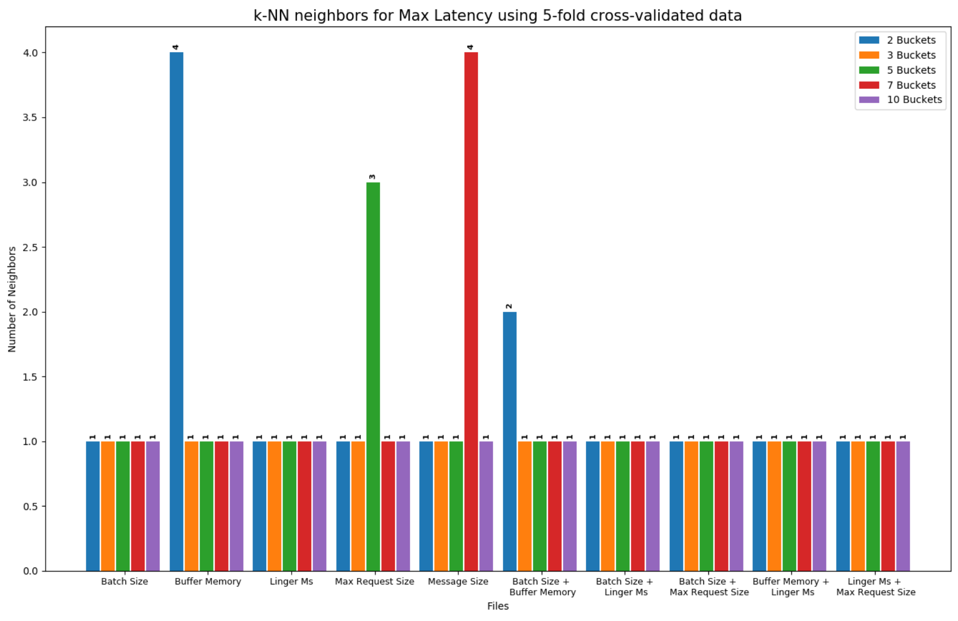


Figure 7.61: Parameter Neighbors for k-NN algorithm with ”Max Latency” predicted target

The number of neighbors that produces the best results in the classification, once again remains almost constant at value 1, as in previous targets.

7.4.3.2 Linear SVM algorithm

Records/sec:

- **Accuracy**:

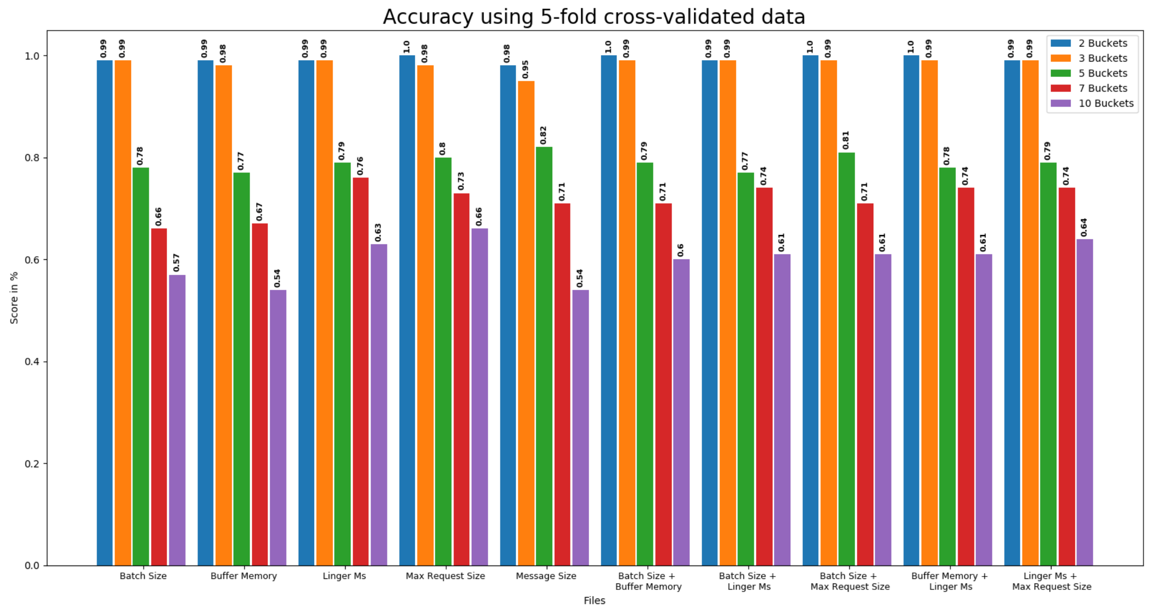


Figure 7.62: Records/sec predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

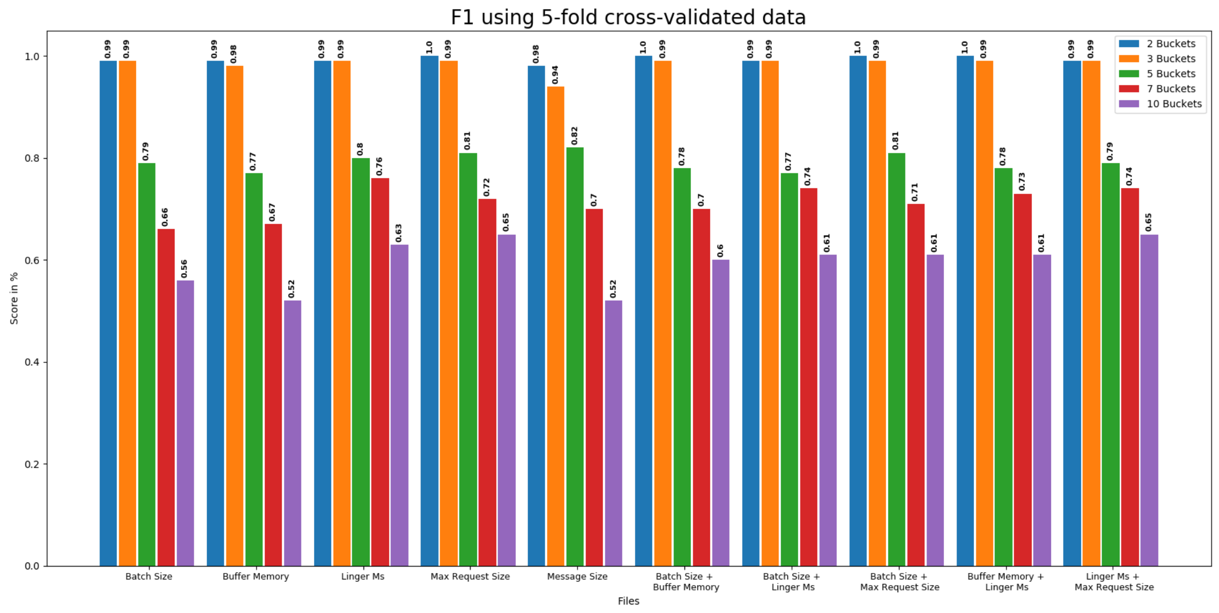


Figure 7.63: Records/sec predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

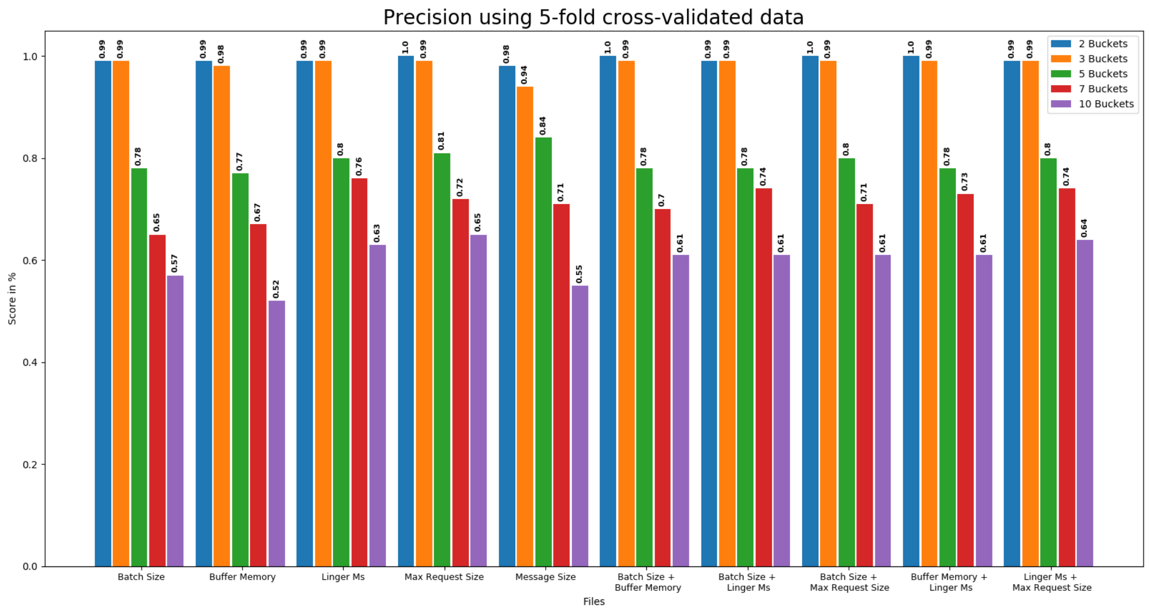


Figure 7.64: Records/sec predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

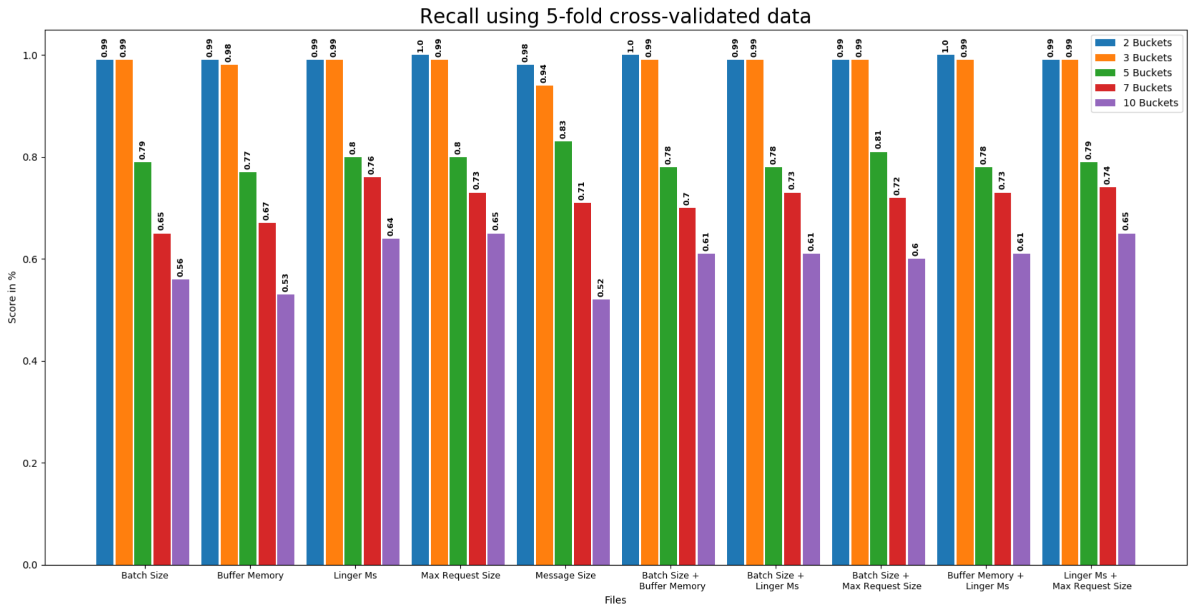


Figure 7.65: Records/sec predicting accuracy using recall metric with cross-validated data (higher is better)

Here we notice that, for the ”Records/sec” target, all files almost hold the same results. Furthermore, as the number of buckets for classification increases in each file, the success rate of the algorithm gradually decreases. However, for buckets two and three, we notice that the success rate is nearly the same and almost reaches the absolute value.

- **C**:

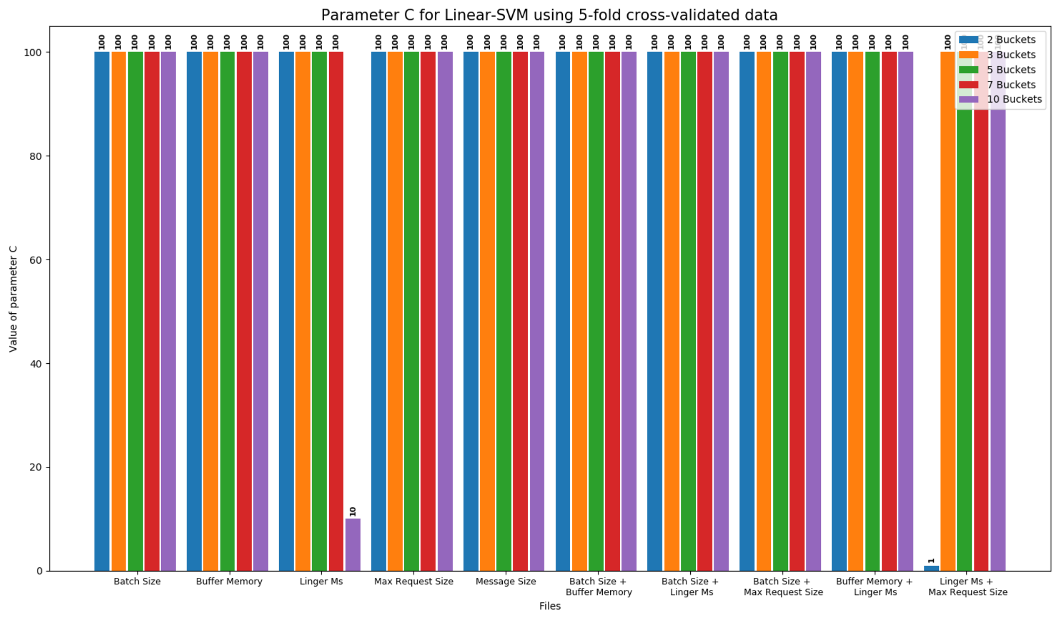


Figure 7.66: Parameter C for Linear-SVM algorithm with ”Records/sec” predicted target

In almost every case, as well as for every metric, the exhaustive search yielded the optimal value 100 for the parameter C, which is considered a reasonably high one.

MB/sec:

- **Accuracy**:

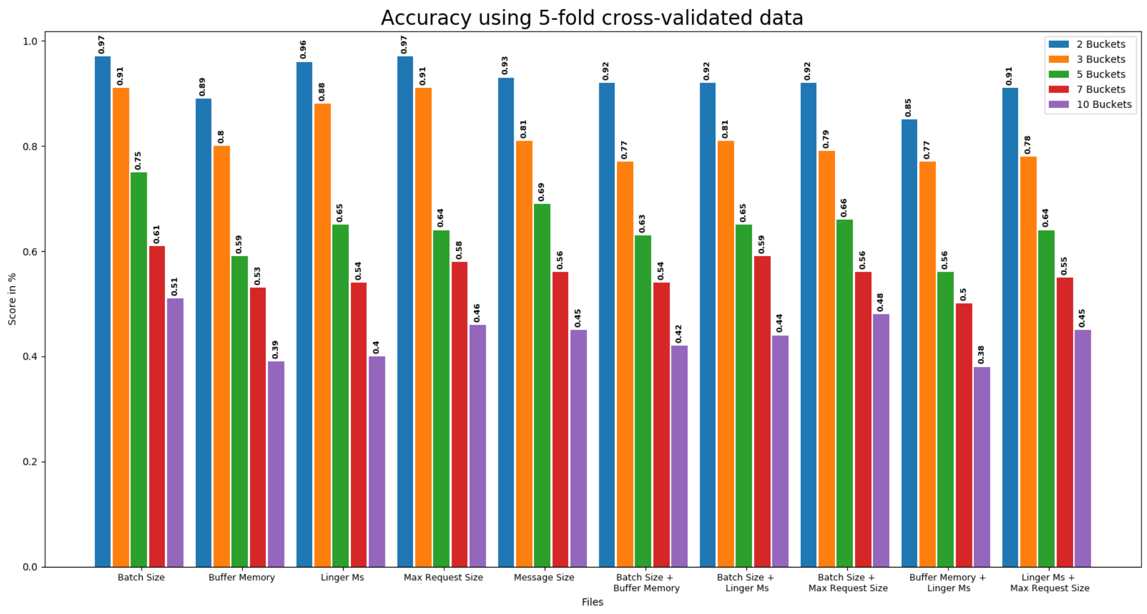


Figure 7.67: MB/sec predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

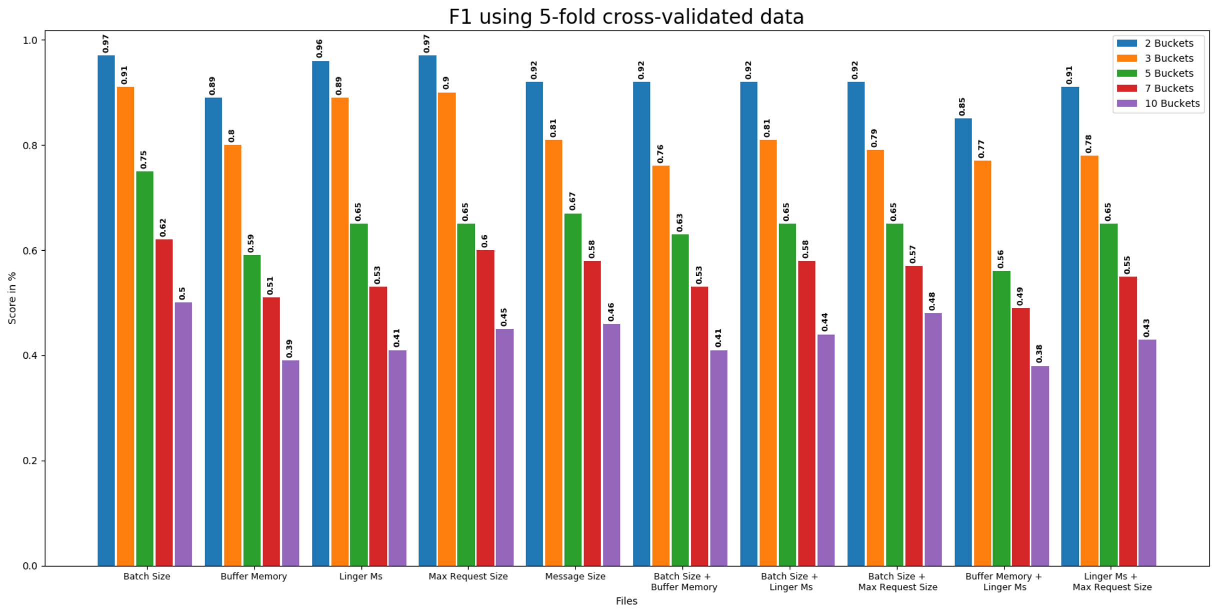


Figure 7.68: MB/sec predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

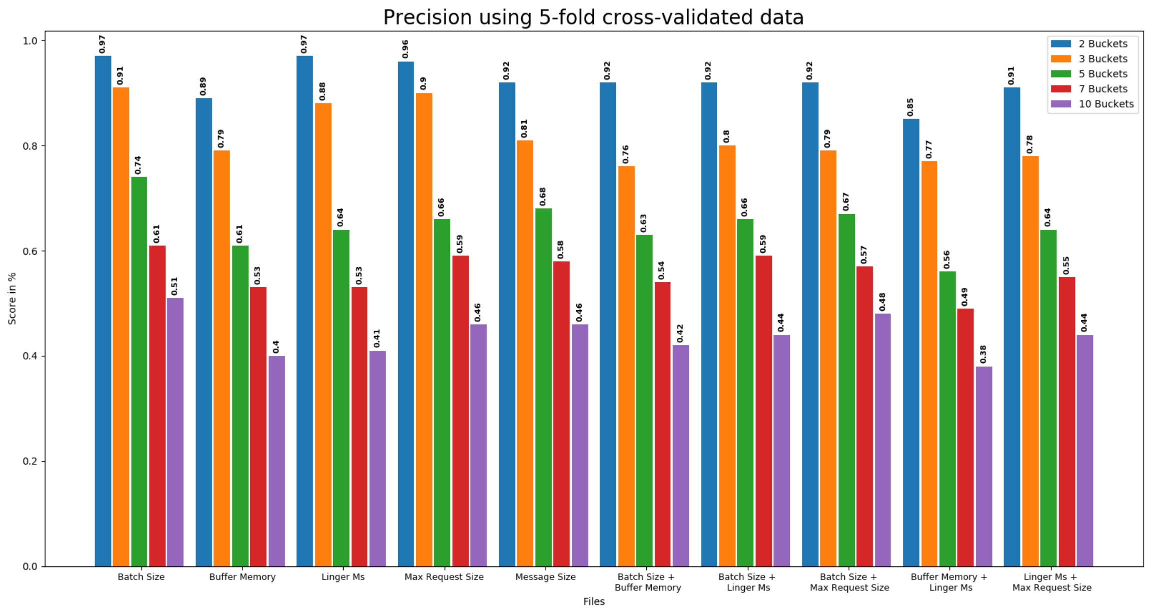


Figure 7.69: MB/sec predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

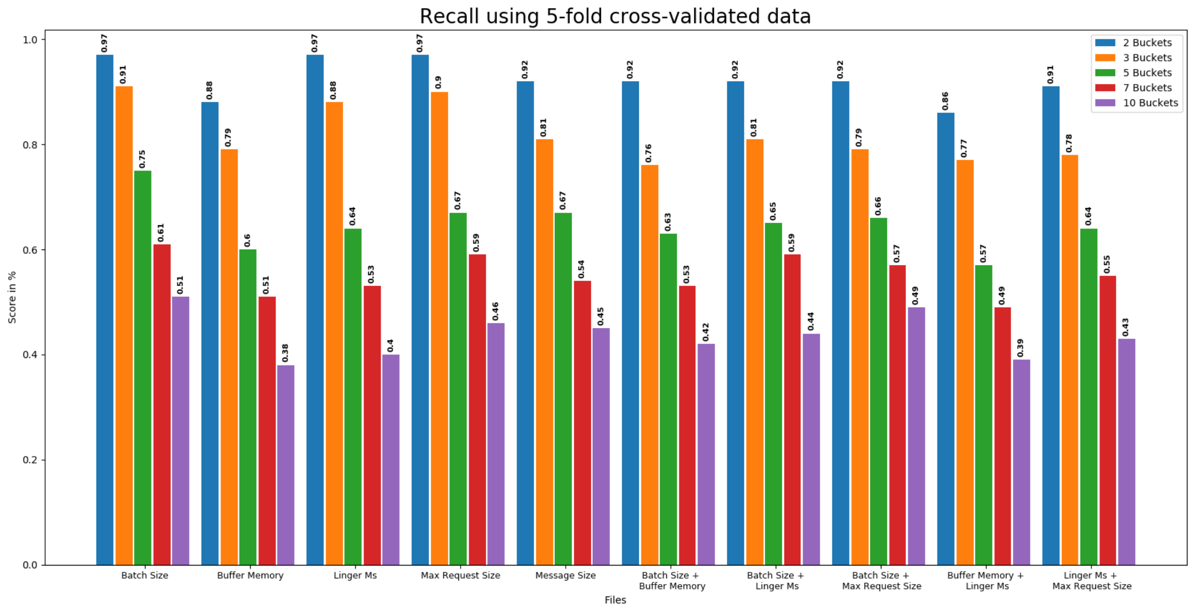


Figure 7.70: MB/sec predicting accuracy using recall metric with cross-validated data (higher is better)

We notice again that, as the buckets increase, there is a drop in the performance of the algorithm. However, contrary to the target ”Records/sec”, the success rate for three buckets also diminishes. Percentage drops less than it does from three to five buckets, or from five to seven; though it still does not remain stable, as was the case with target ”Records/sec”.

- **C**:

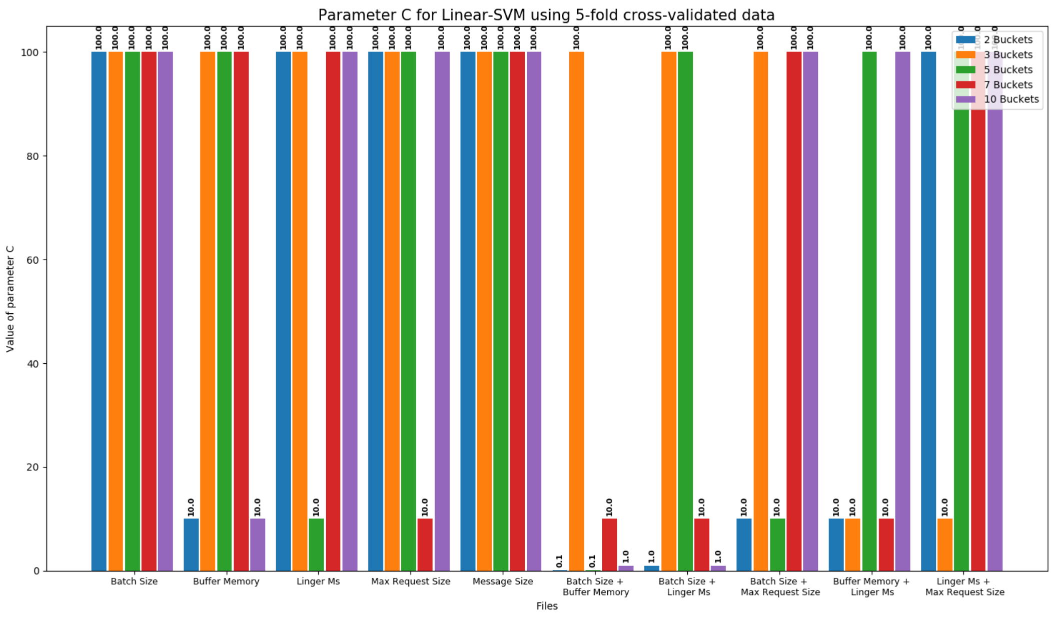


Figure 7.71: Parameter C for Linear-SVM algorithm with ”MB/sec” predicted target

Here we see that there are transitions in the value of the parameter C, though this may be due to the random order in which the data are given to the algorithm.

Avg Latency:

- **Accuracy**:

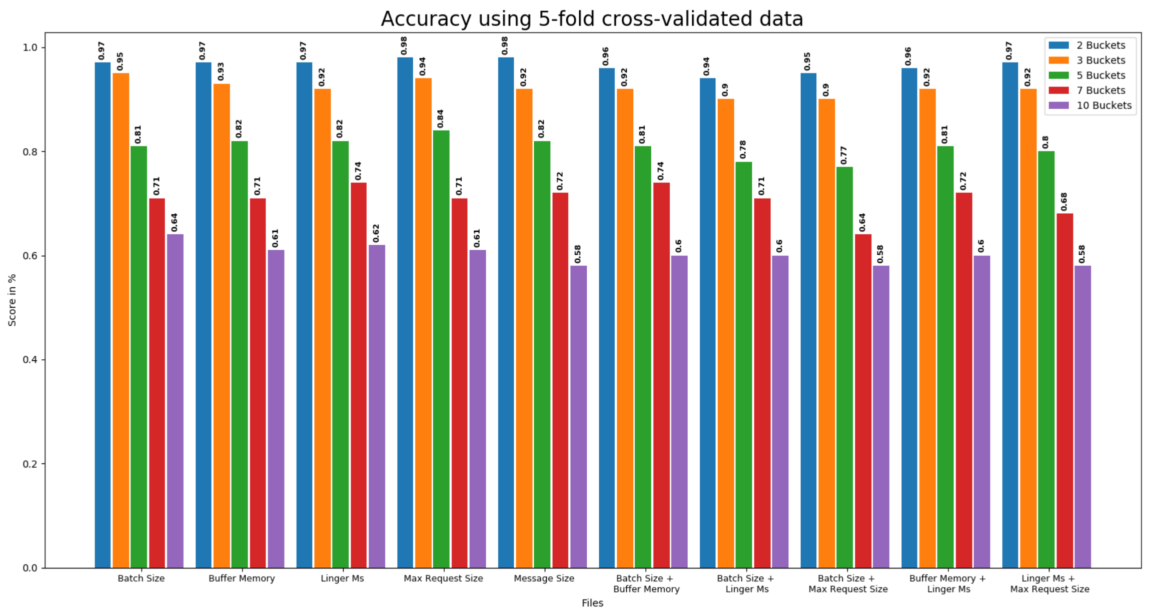


Figure 7.72: Avg Latency predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

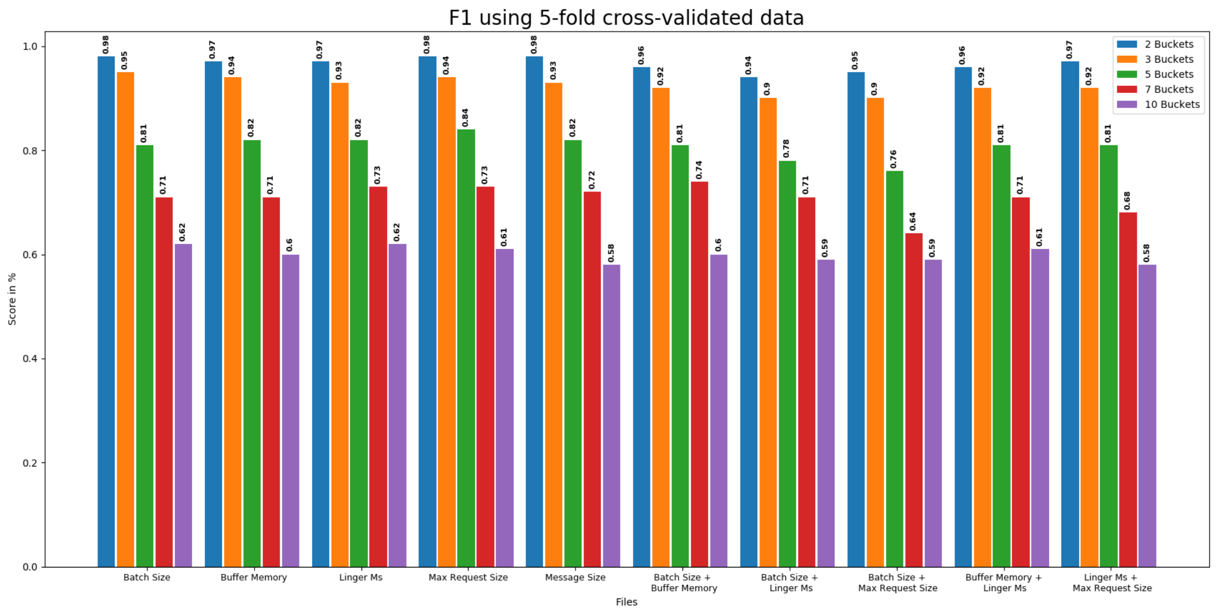


Figure 7.73: Avg Latency predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

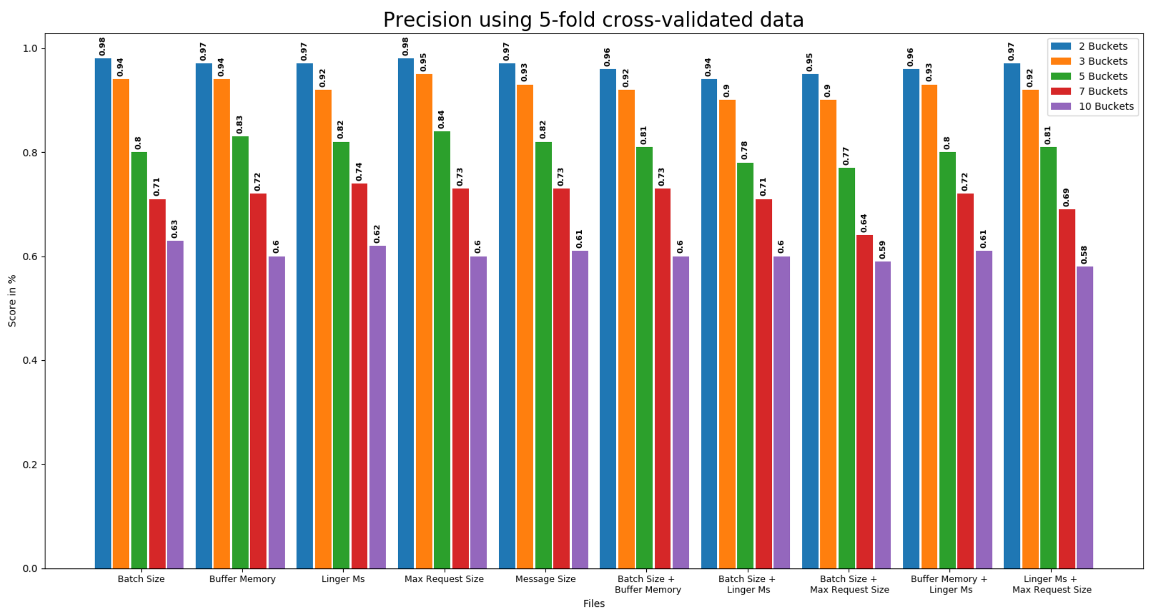


Figure 7.74: Avg Latency predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

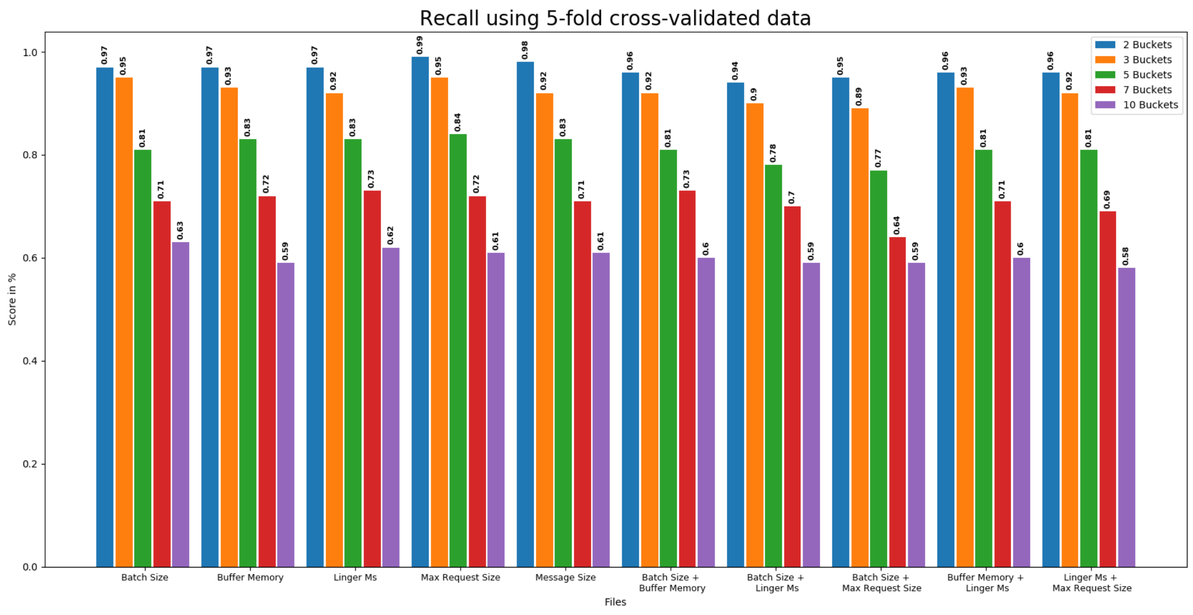


Figure 7.75: Avg Latency predicting accuracy using recall metric with cross-validated data (higher is better)

We observe almost identical results as the ”MB/sec” target; that is, a gradual decrease as the number of buckets for all files increases, but the percentage is smaller for the change from two to three buckets. Although the results for two buckets are close to absolute, the results for three buckets have a satisfactory success rate in the predictions.

- **C**:

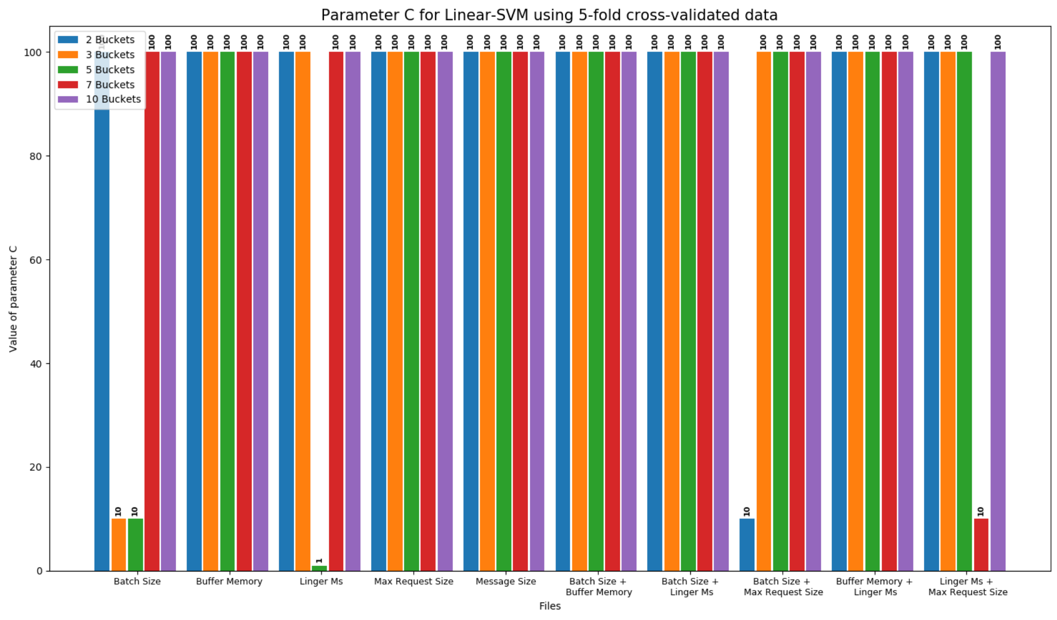


Figure 7.76: Parameter C for Linear-SVM algorithm with ”Avg Latency” predicted target

As in the previous targets, a constant value of 100 is observed for parameter C, coupled with some downward fluctuations.

Max Latency:

- **Accuracy**:

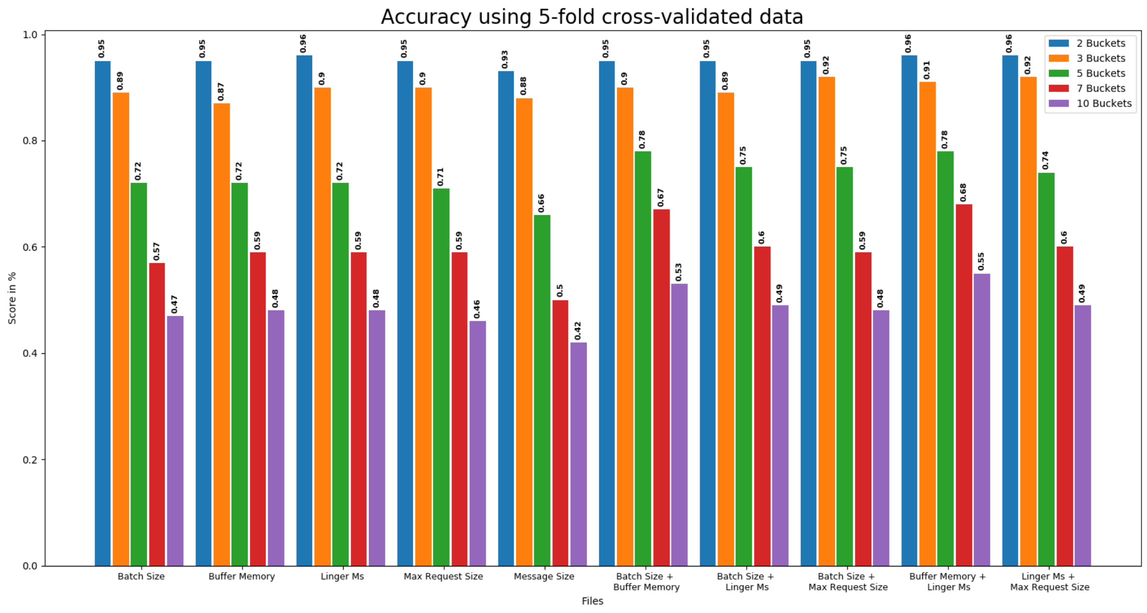


Figure 7.77: Max Latency predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

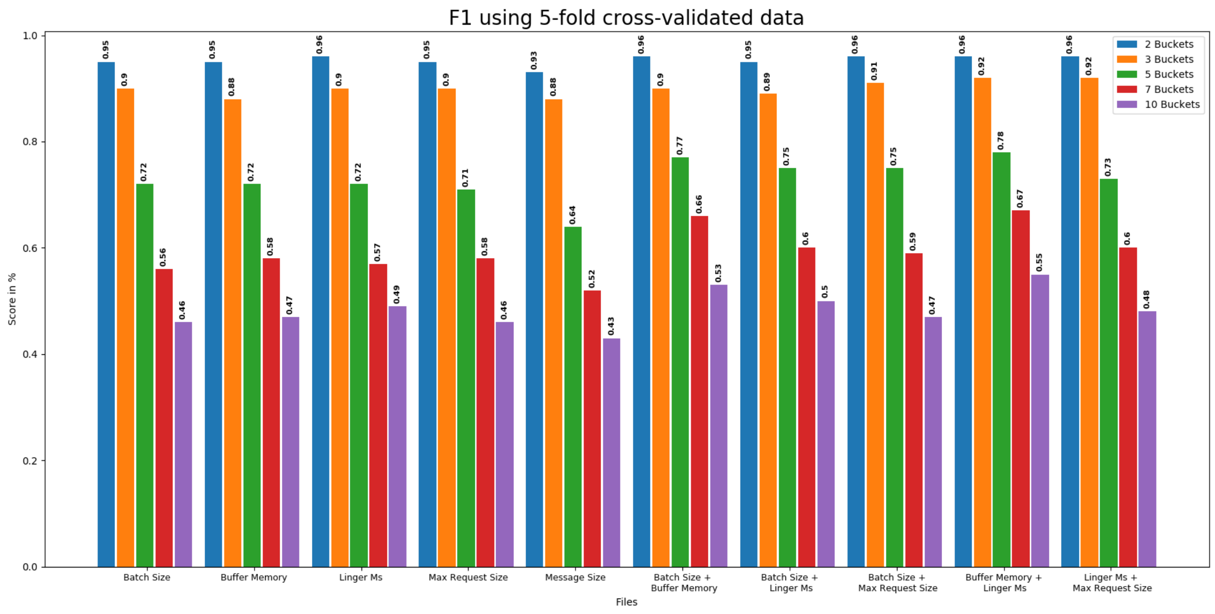


Figure 7.78: Max Latency predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

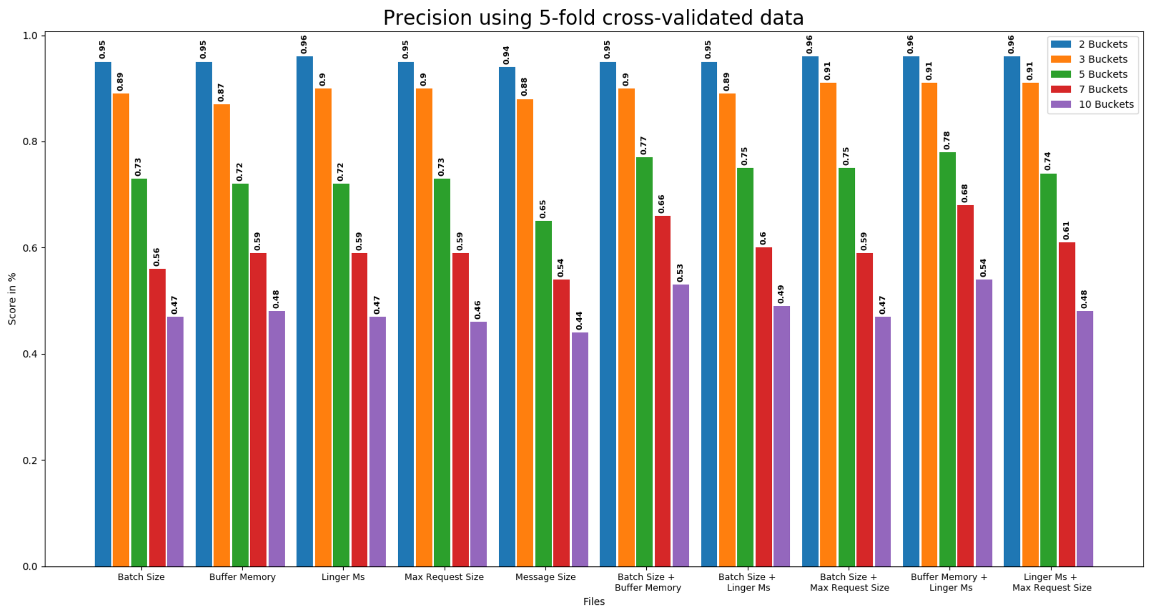


Figure 7.79: Max Latency predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

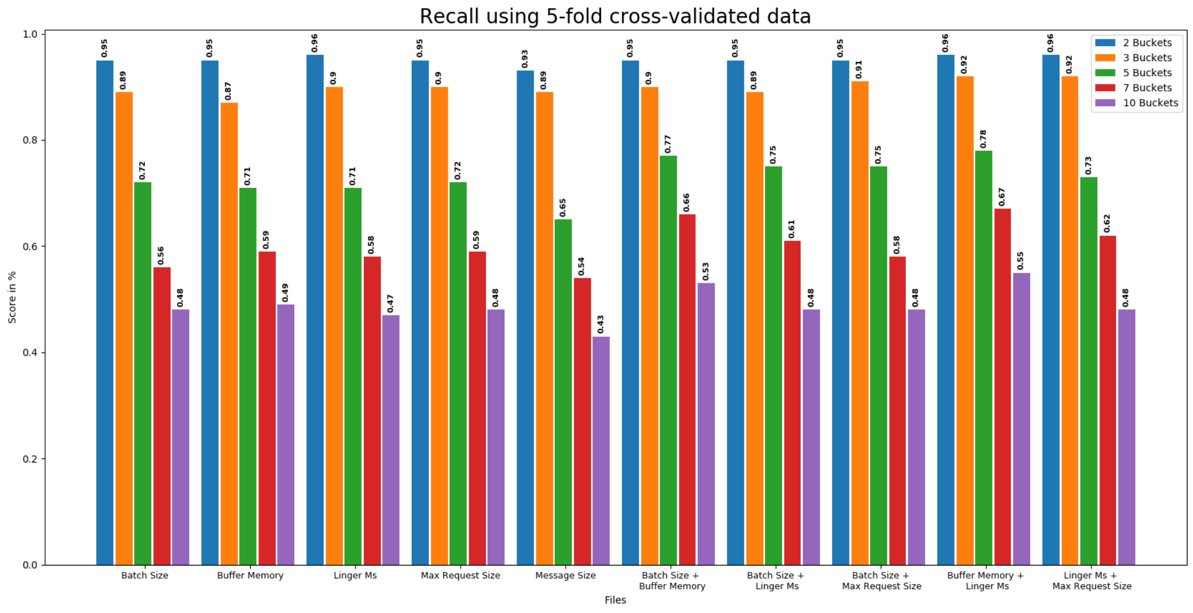


Figure 7.80: Max Latency predicting accuracy using recall metric with cross-validated data (higher is better)

We observe that after the three buckets, there is a large reduction in the performance of the algorithm. For the three buckets the efficiency percentage is around 90%, which is adequate but not perfect, while for the two buckets it reaches 96%, which is a very good percentage; however, the range of the buckets is bigger and, perhaps, as we discussed in Section 6.3, the prediction of the actual value may have a lot of noise/variance.

- **C**:

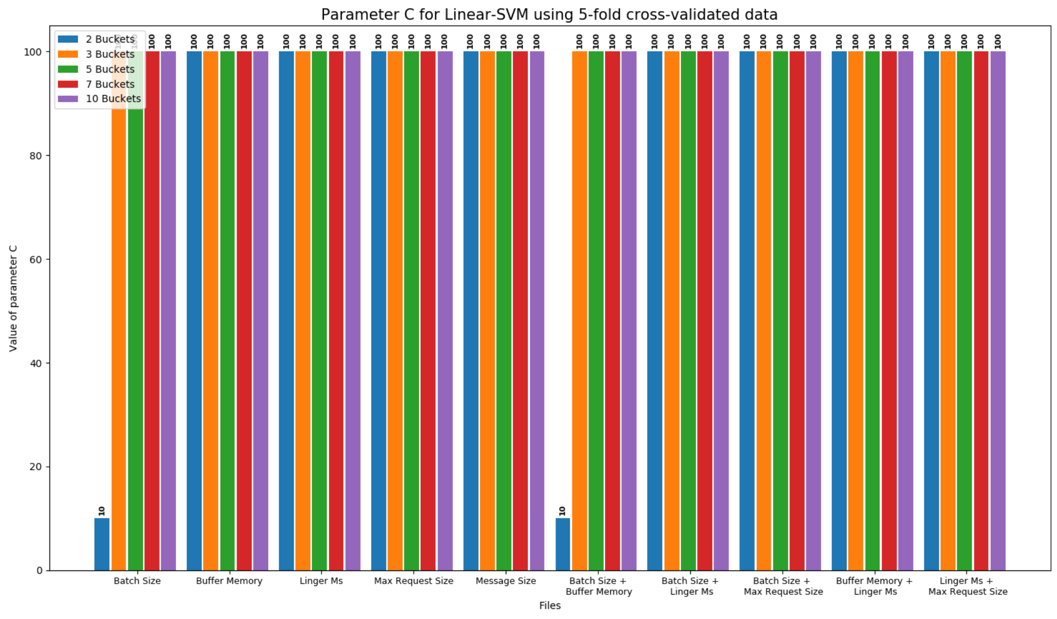


Figure 7.81: Parameter C for Linear-SVM algorithm with ”Max Latency” predicted target

Once again, the value of parameter C is 100 for almost any number of buckets per file. It is also constant for every metric.

7.4.3.3 RBF SVM algorithm

Records/sec:

- **Accuracy**:

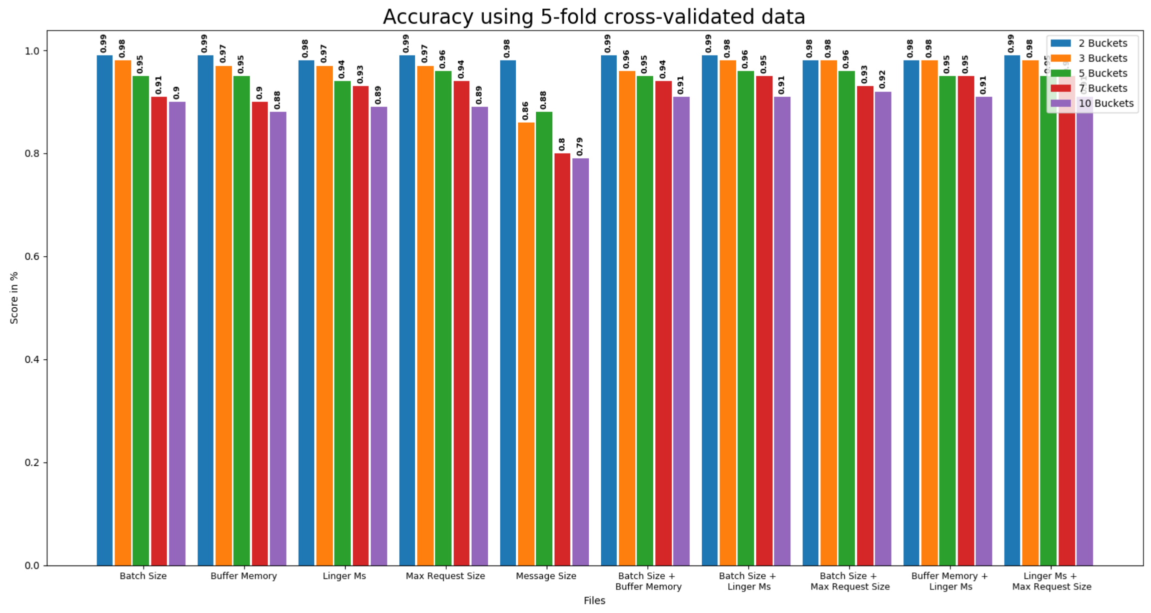


Figure 7.82: Records/sec predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

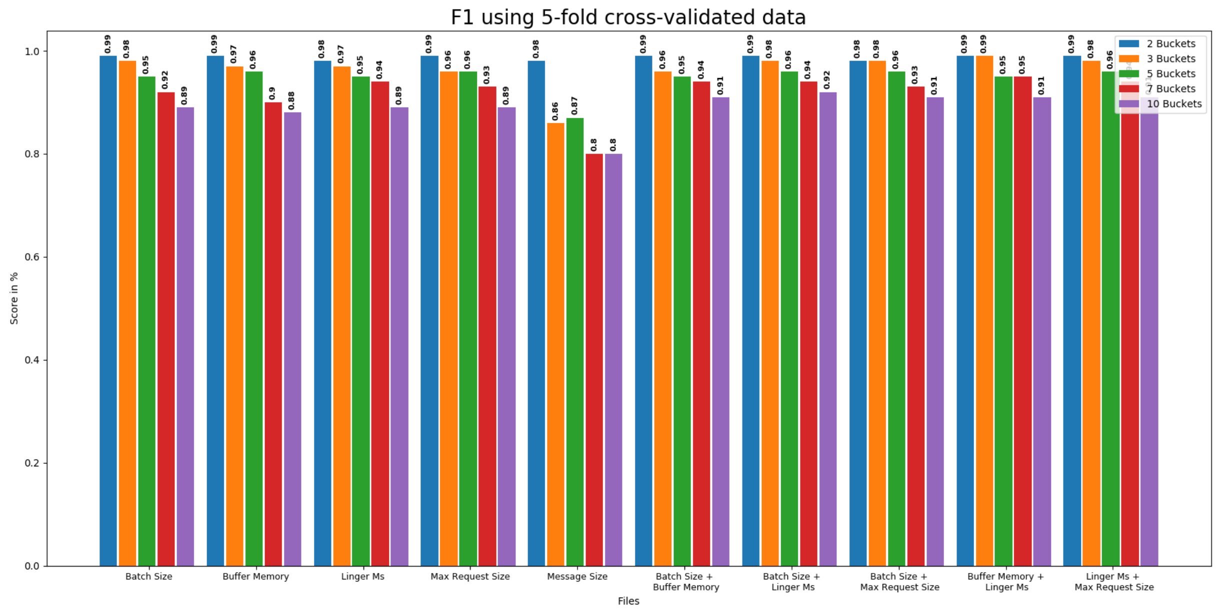


Figure 7.83: Records/sec predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

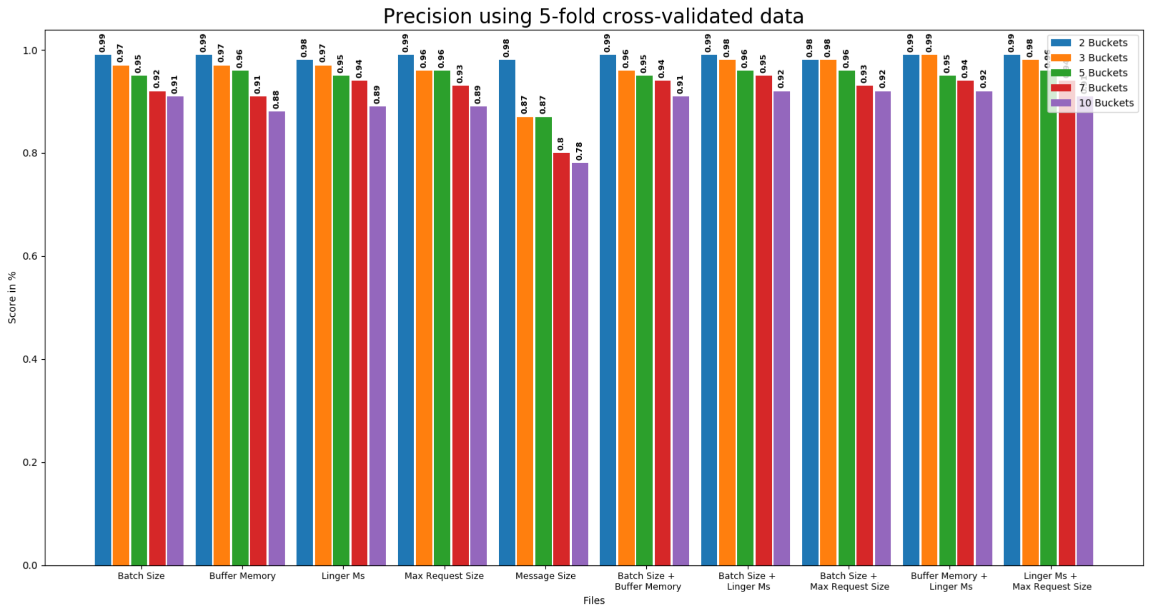


Figure 7.84: Records/sec predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

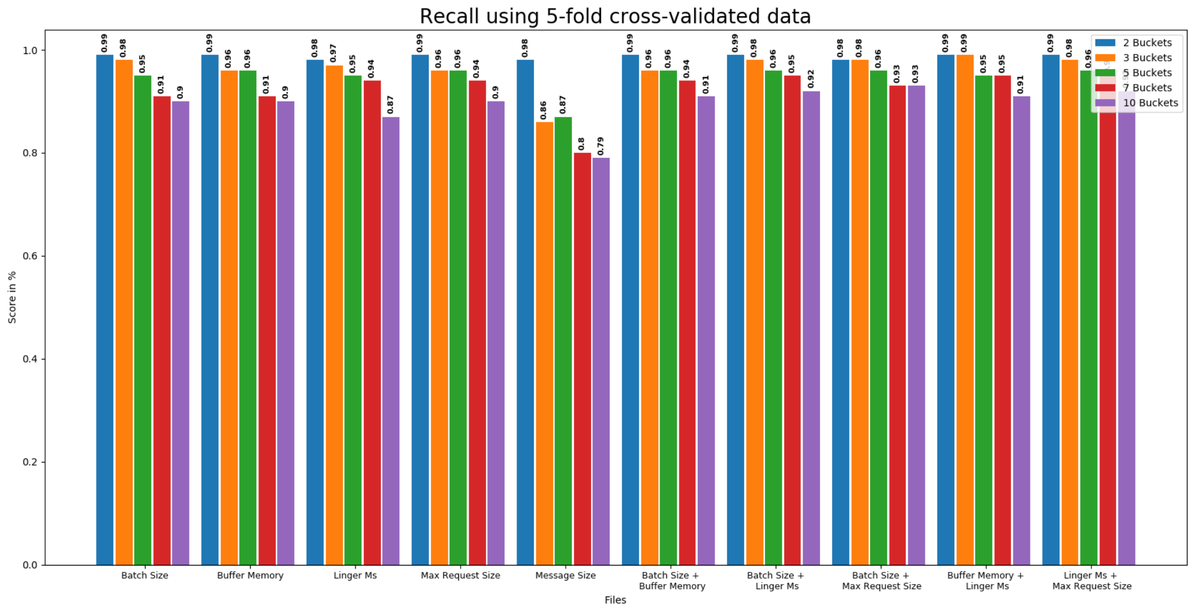


Figure 7.85: Records/sec predicting accuracy using recall metric with cross-validated data (higher is better)

Contrary to the previous algorithms, we remark that we have quite high success rates and the decrease observed alongside the increase of buckets is almost negligible. In addition, the success rates of the algorithm reach almost absolute success rates, with a slight exception occurring in the file of ”Message Size”.

- **C**:

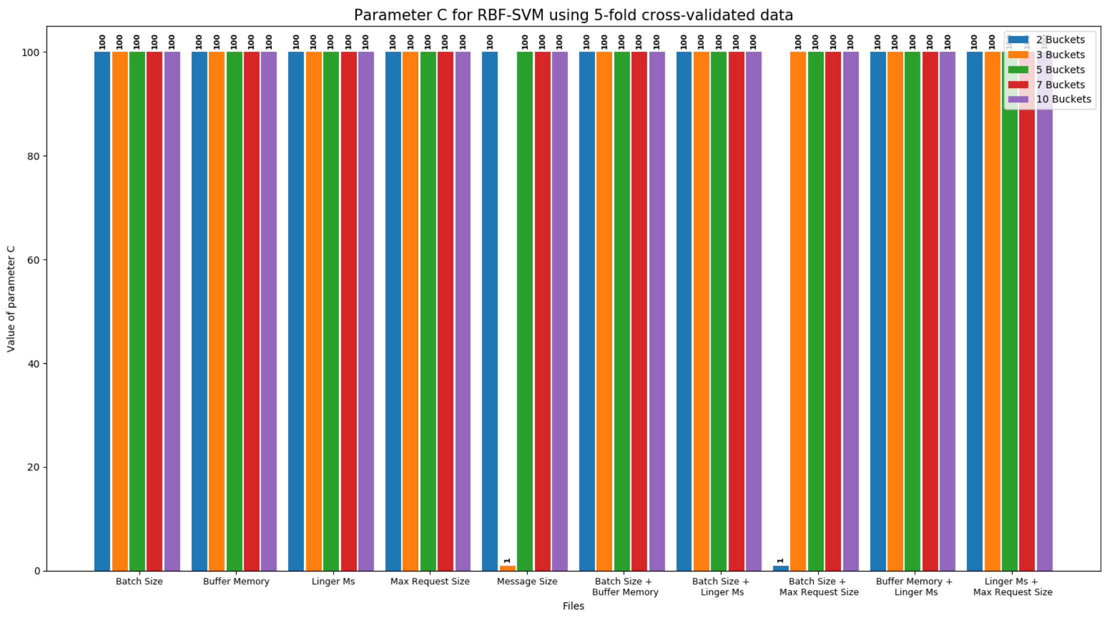


Figure 7.86: Parameter C for RBF-SVM algorithm with ”Records/sec” predicted target

- **Gamma**:

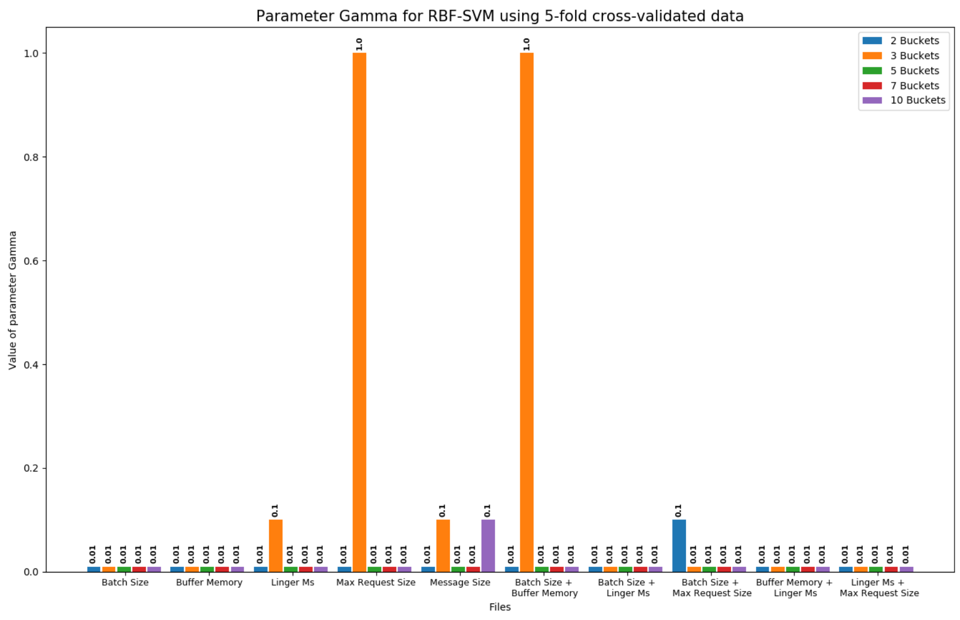


Figure 7.87: Parameter Gamma for RBF-SVM algorithm with "Records/sec" predicted target

We see that the value of hyperparameter C is consistently 100 with only two exceptions, while the value of hyperparameter Gamma is low at 0.01, though, again, with few exceptions that reach even value 1.

MB/sec:

- **Accuracy**:

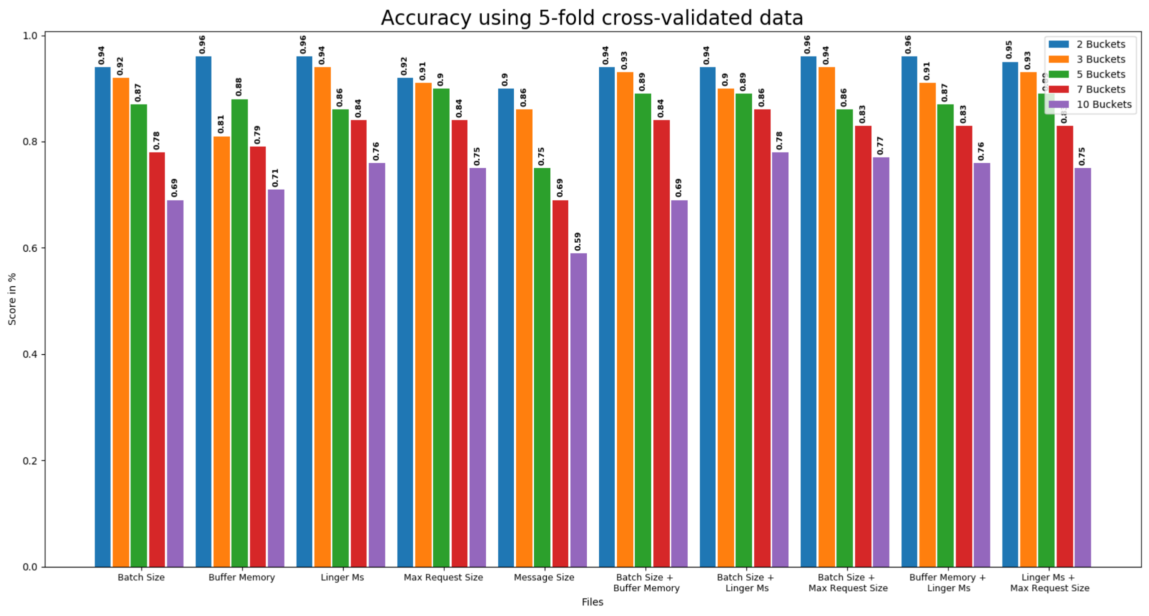


Figure 7.88: MB/sec predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

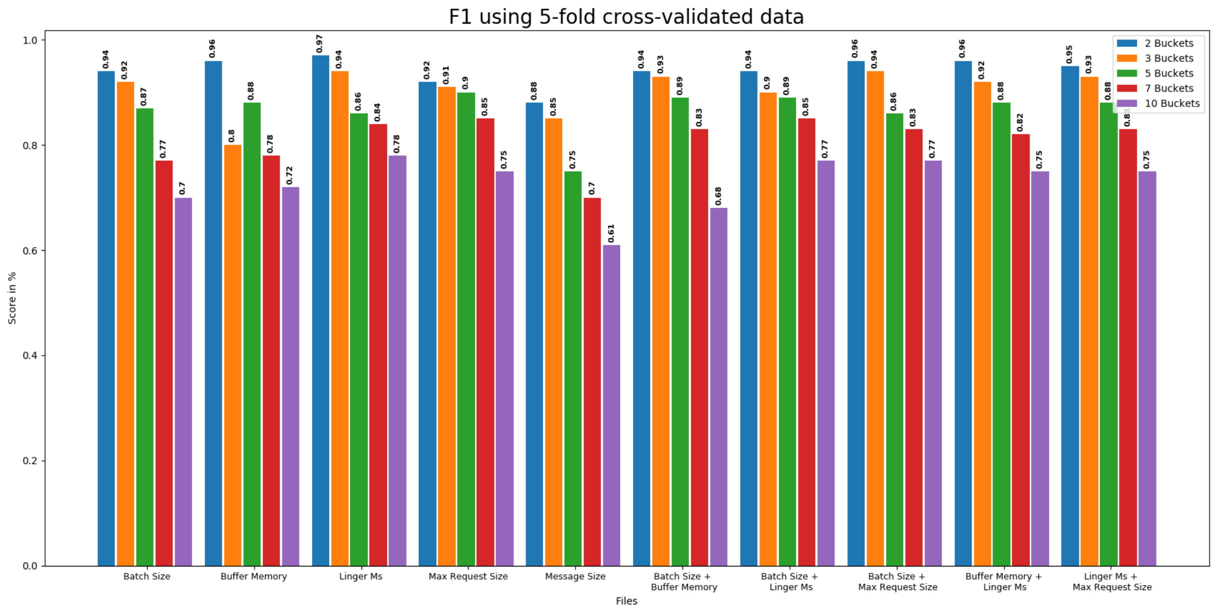


Figure 7.89: MB/sec predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

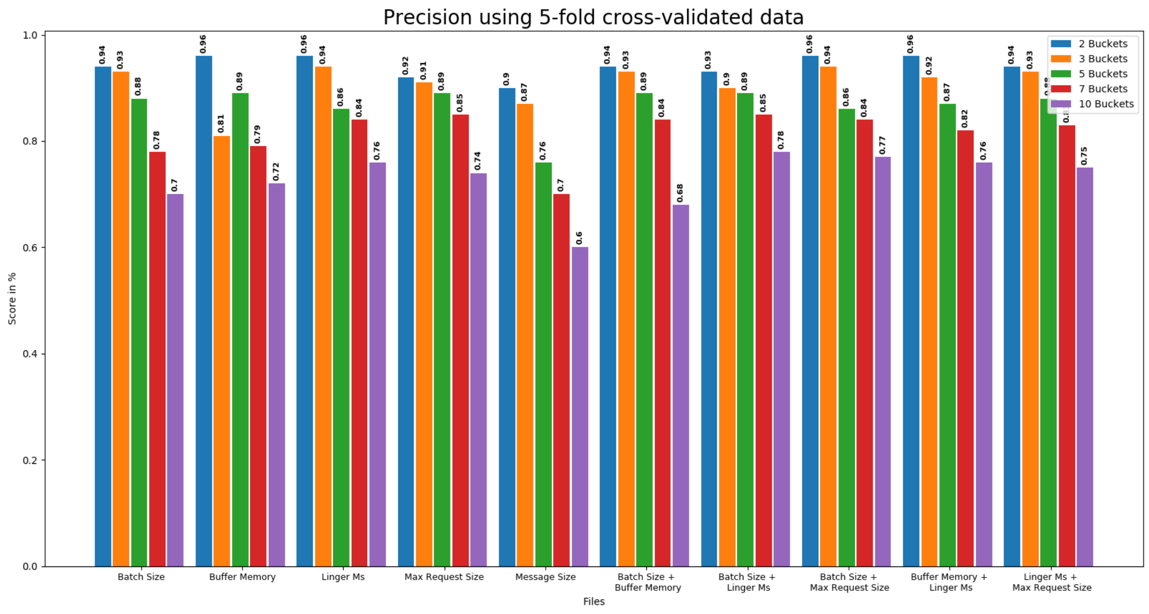


Figure 7.90: MB/sec predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

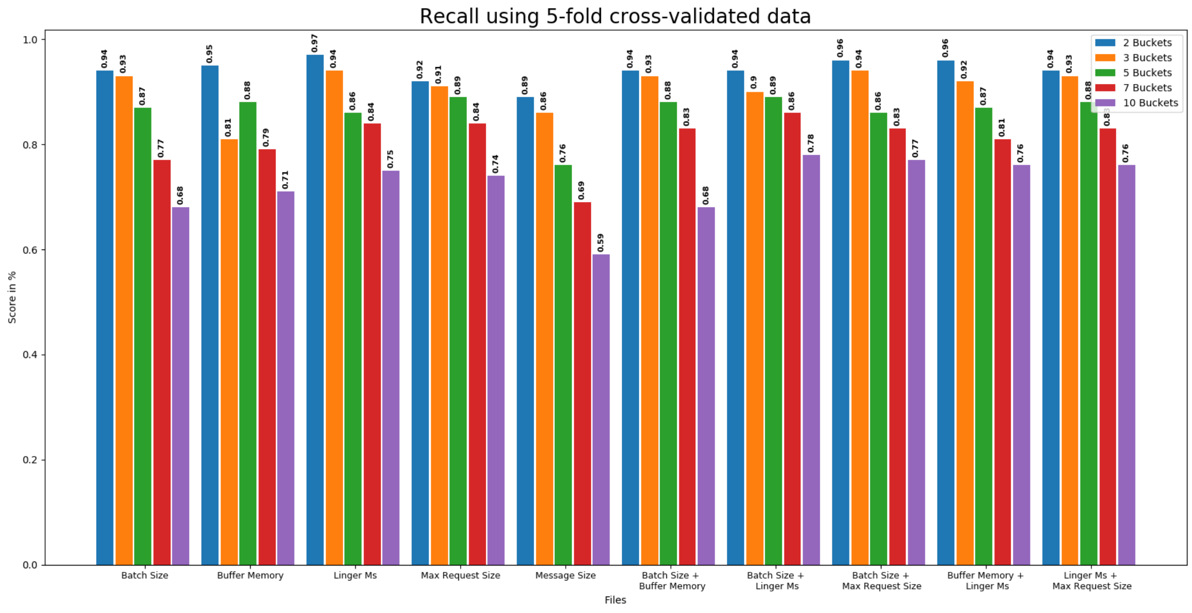


Figure 7.91: MB/sec predicting accuracy using recall metric with cross-validated data (higher is better)

In the target ”MB/sec” we notice that, in general, the performance levels of the algorithm are slightly lower than those of the target ”Records/sec”. Despite that, in addition to the general decline, we observe that there is a gradual decline as the number of buckets increases, something that was not observed in the previous target.

- **C**:

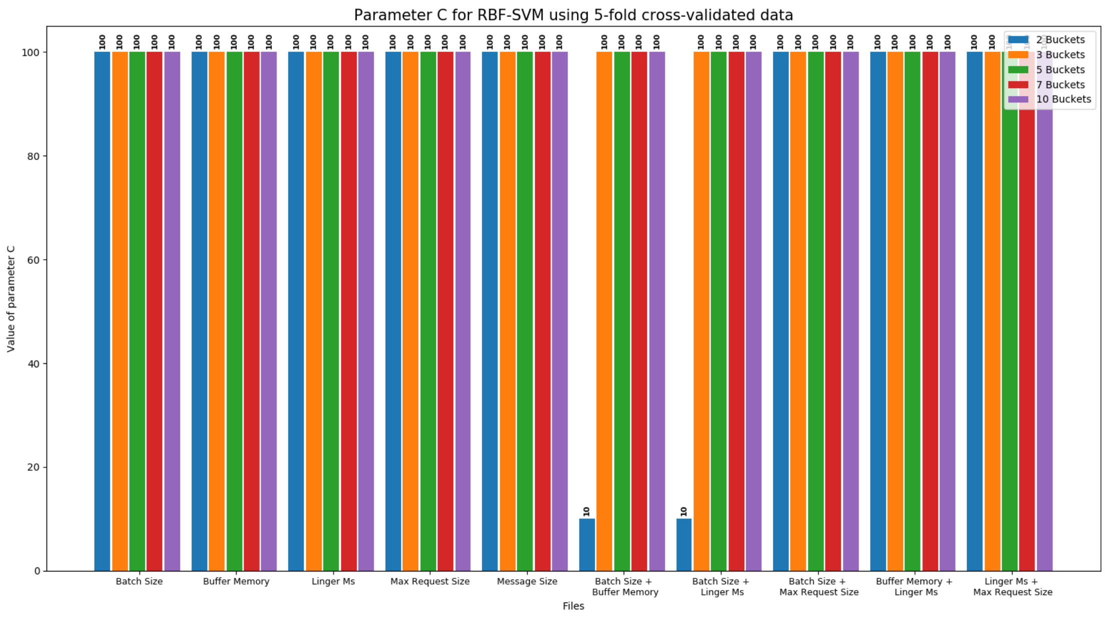


Figure 7.92: Parameter C for RBF-SVM algorithm with ”MB/sec” predicted target

- **Gamma**:

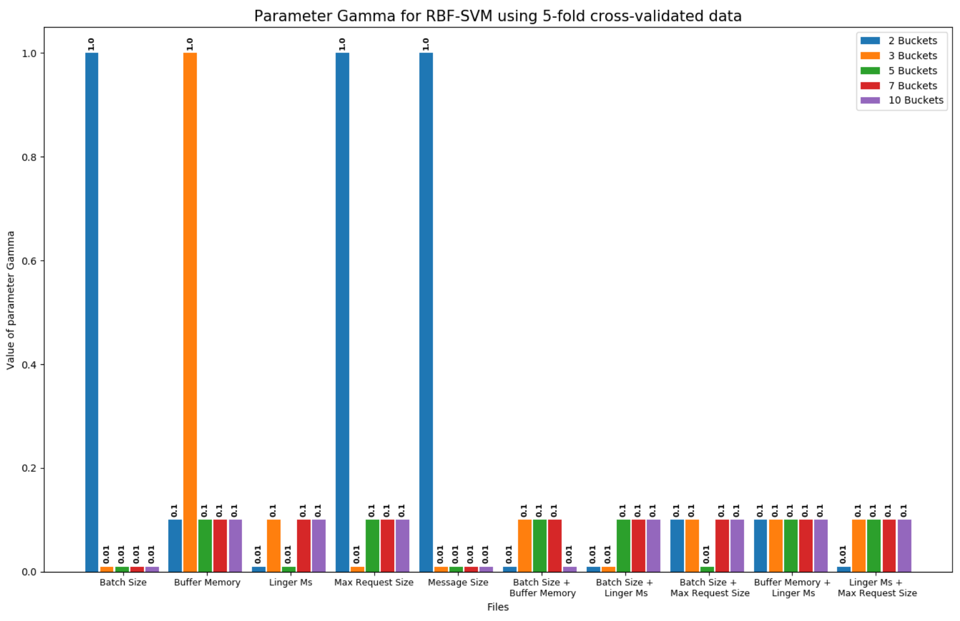


Figure 7.93: Parameter Gamma for RBF-SVM algorithm with ”MB/sec” predicted target

As for the target ”Records/sec”, here too, the value of hyperparameter C remains constant at 100, with only two exceptions. On the contrary, the value of the Gamma hyperparameter ranges mainly in two value levels, 0.01 and 0.1. There are also four points that reach the value of 1. We conclude that the value for the target ”MB/sec” is not very stable, i.e. the changes per number of buckets and per metric, are more noticeable, compared to the previous target ”Records/sec”.

Avg Latency:

- **Accuracy**:

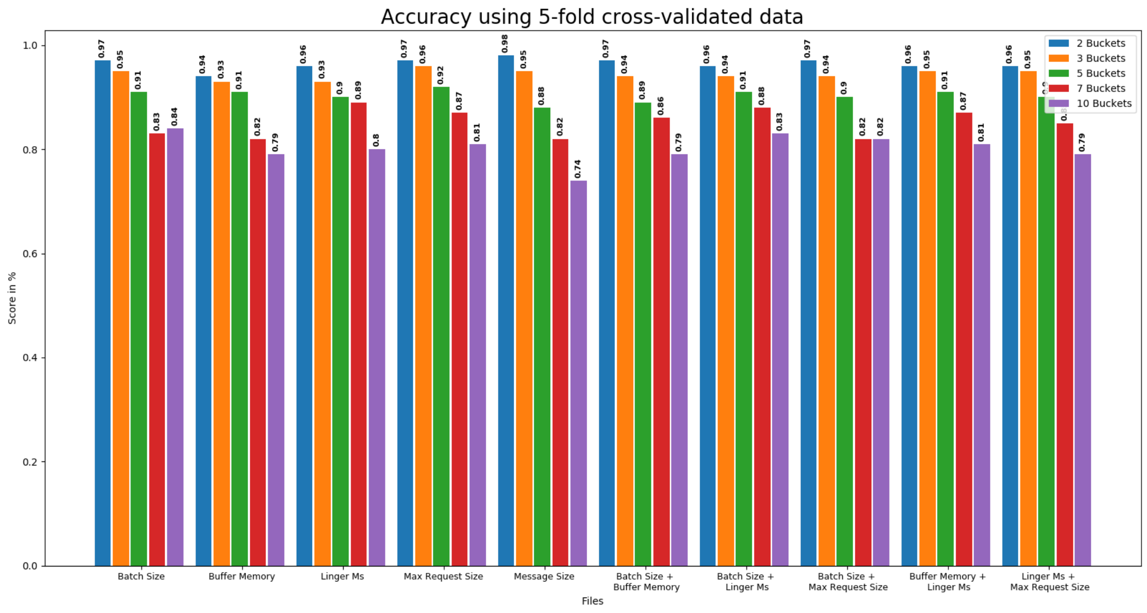


Figure 7.94: Avg Latency predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

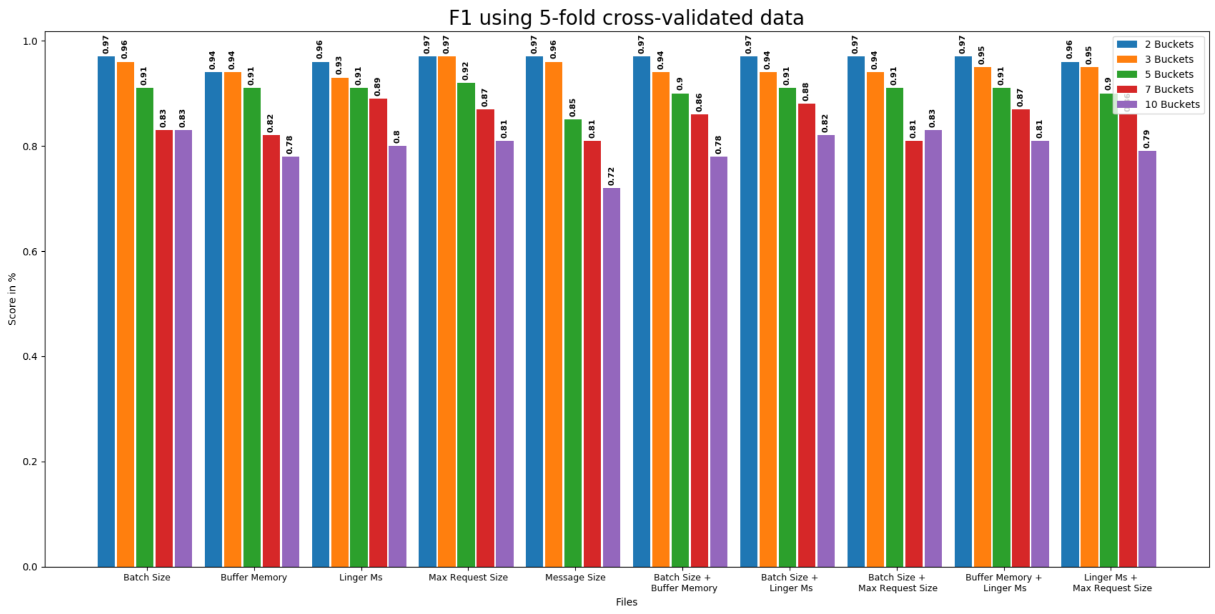


Figure 7.95: Avg Latency predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

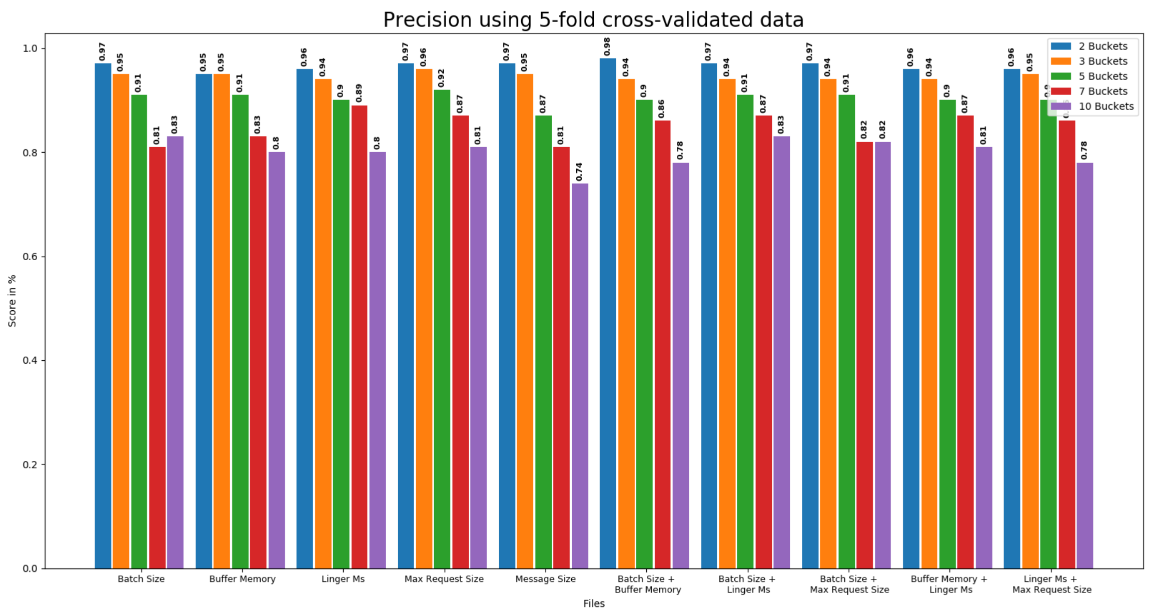


Figure 7.96: Avg Latency predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

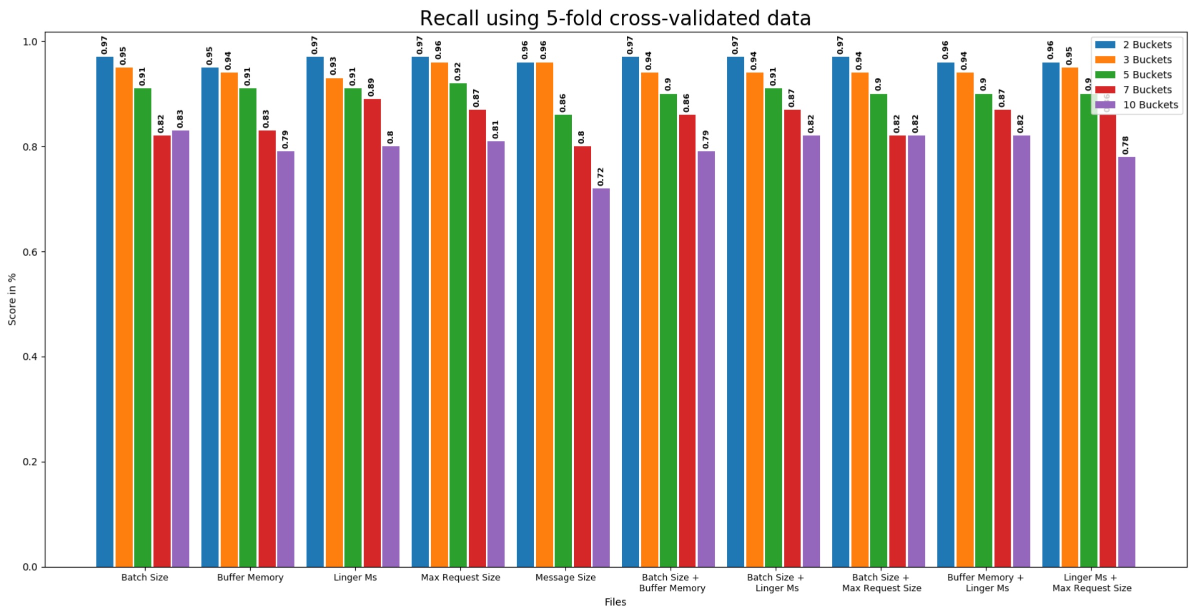


Figure 7.97: Avg Latency predicting accuracy using recall metric with cross-validated data (higher is better)

It can be seen from the diagrams that the performance of the algorithm is quite good, in terms of the ”Avg Latency” target. Of course, as the buckets increase, there is a gradual decrease. Nevertheless, especially for two and three buckets, the performance levels remain very high and almost equal to each other.

- **C**:

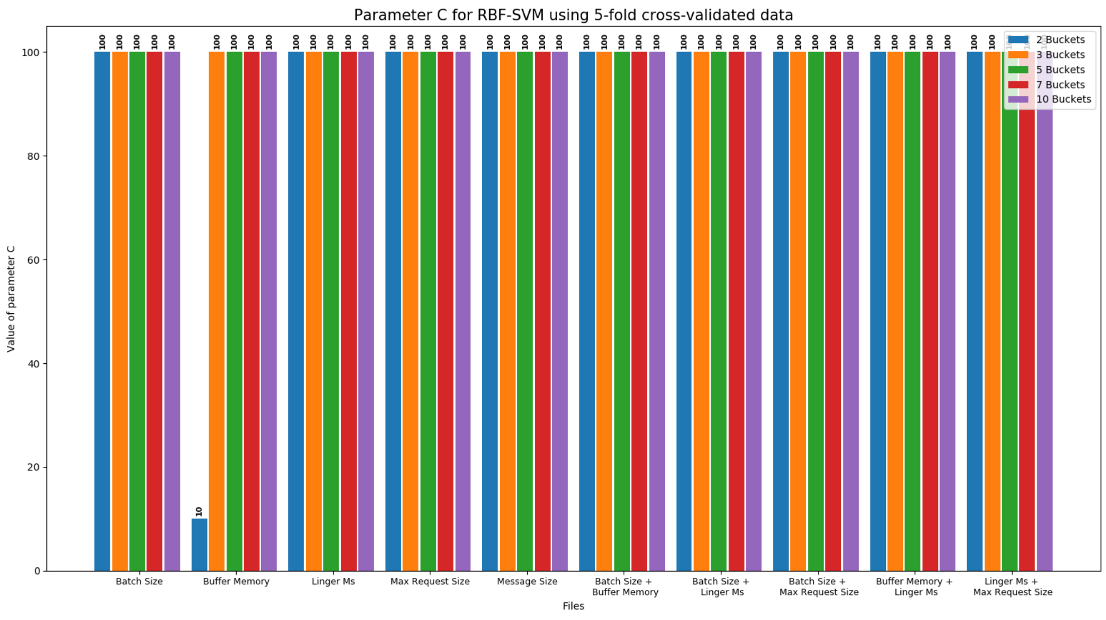


Figure 7.98: Parameter C for RBF-SVM algorithm with ”Avg Latency” predicted target

- **Gamma**:

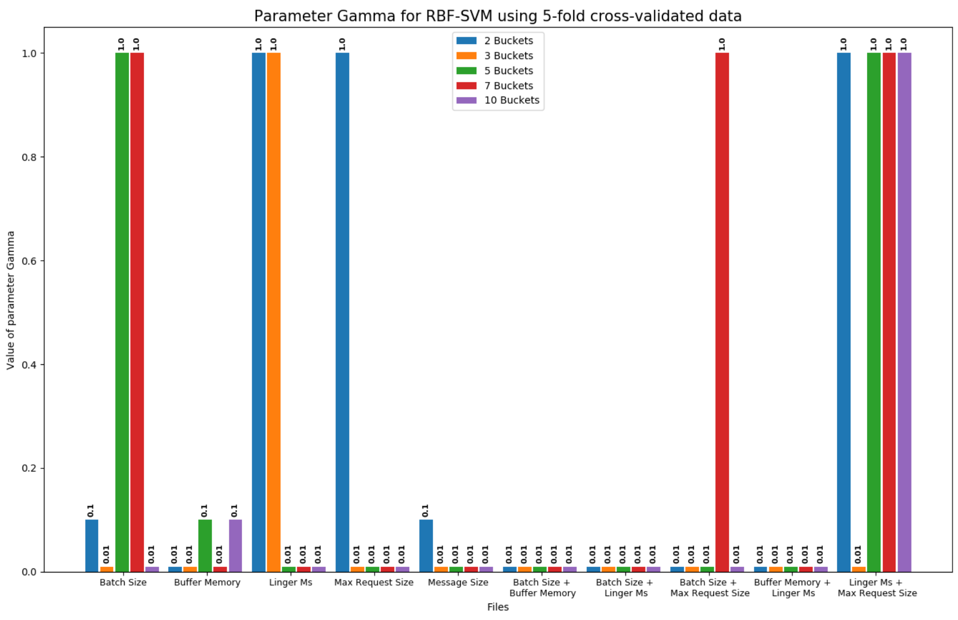


Figure 7.99: Parameter Gamma for RBF-SVM algorithm with ”Avg Latency” predicted target

As for the hyperparameter C of the algorithm, it holds a constant value of 100, except for one case. On the other hand, the gamma hyperparameter ranges in three levels: 0.01, 0.1 and 1, which differ significantly, as shown in the diagram.

Max Latency:

- **Accuracy**:

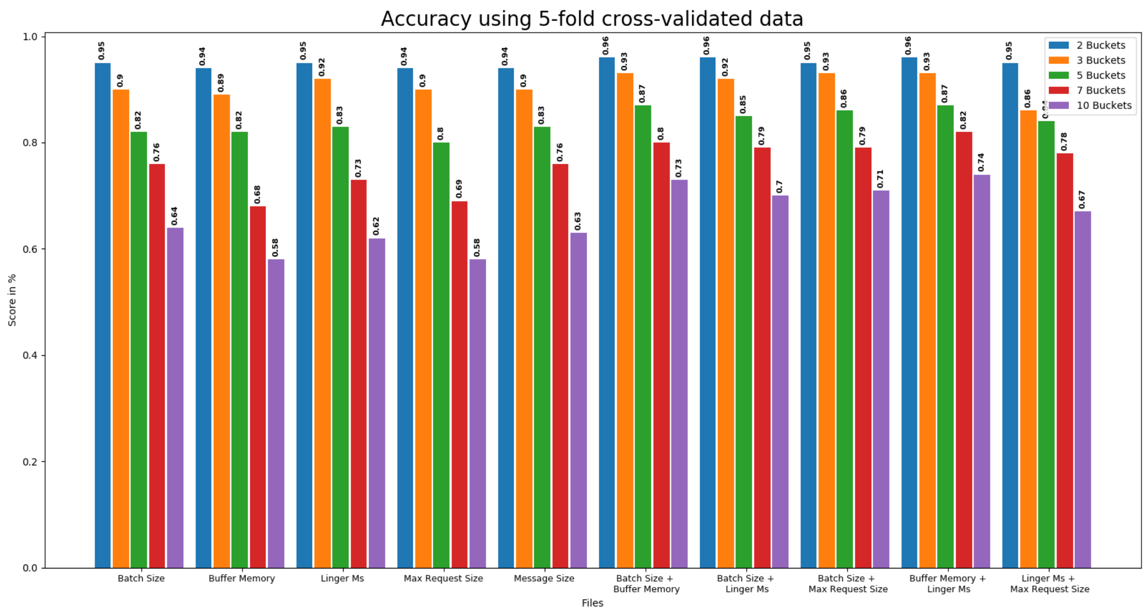


Figure 7.100: Max Latency predicting accuracy using accuracy metric with cross-validated data (higher is better)

- **F1**:

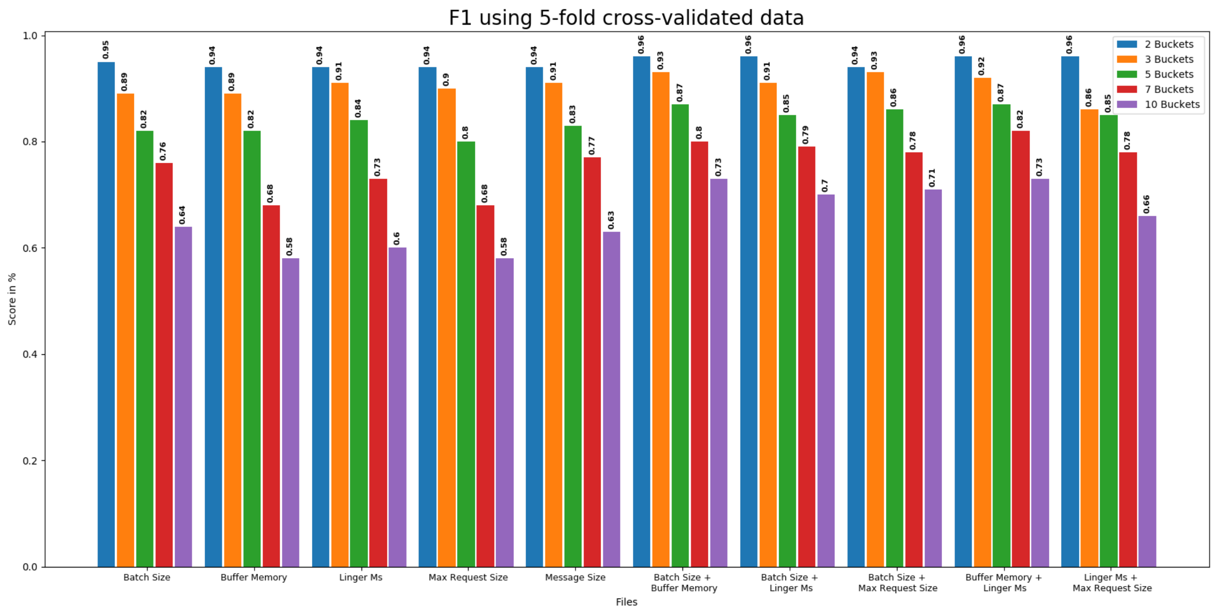


Figure 7.101: Max Latency predicting accuracy using f1 metric with cross-validated data (higher is better)

- **Precision**:

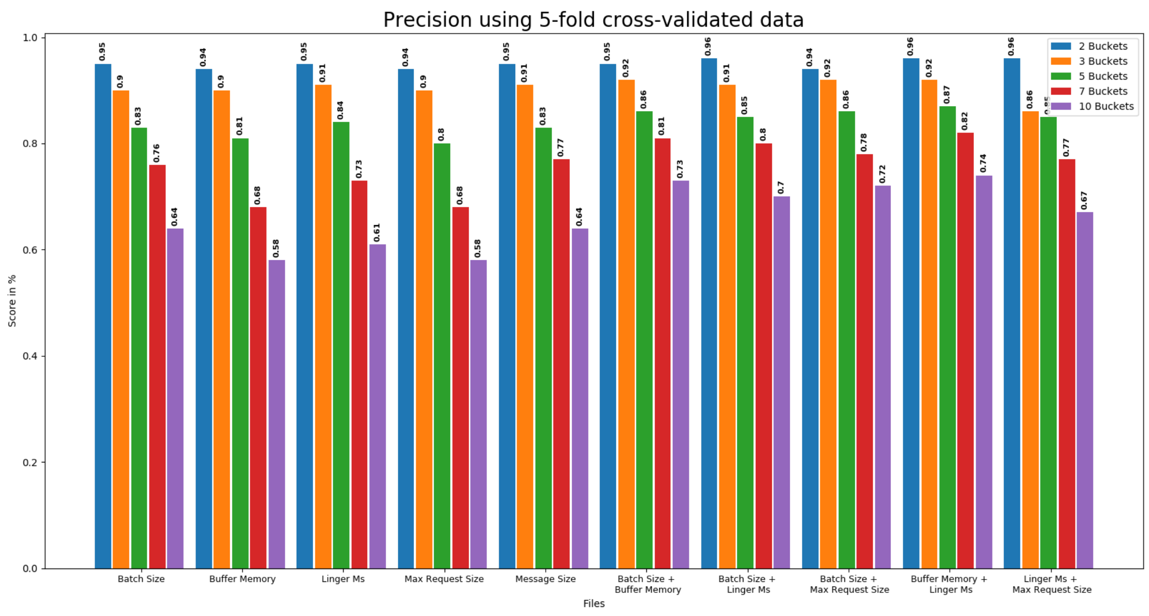


Figure 7.102: Max Latency predicting accuracy using precision metric with cross-validated data (higher is better)

- **Recall**:

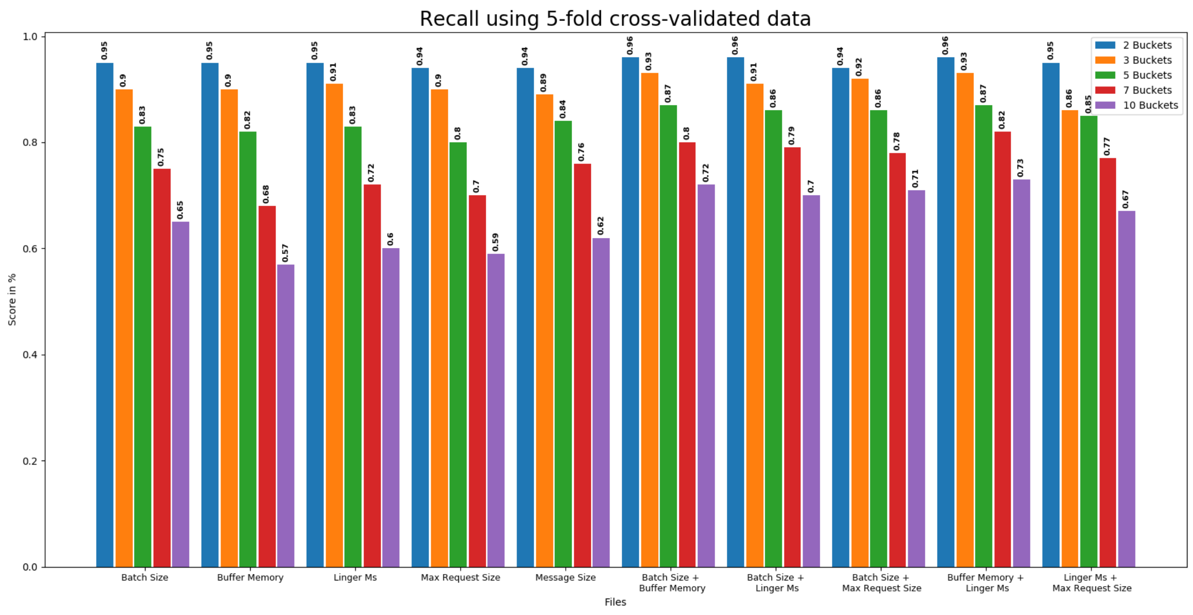


Figure 7.103: Max Latency predicting accuracy using recall metric with cross-validated data (higher is better)

We observe that the accuracy of the algorithm for the ”Max Latency” target is less than the ”Avg Latency” target, which is a similar target, as we explained in Section 7.4.1. Even the gradual decrease, as the number of buckets increases, is much steeper compared to all targets.

- **C**:

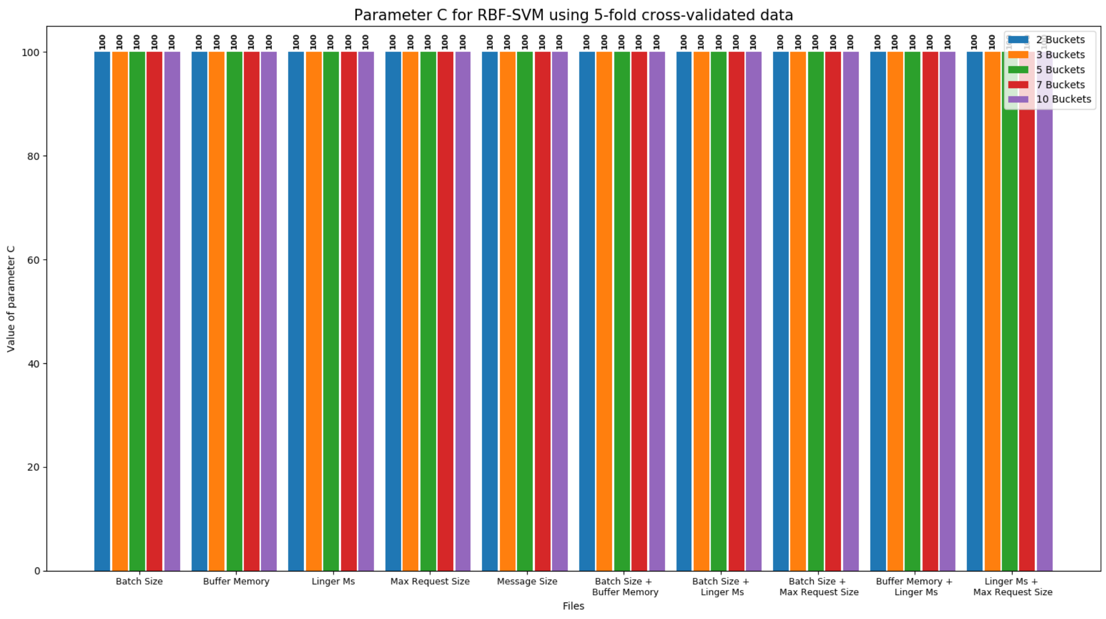


Figure 7.104: Parameter C for RBF-SVM algorithm with ”Max Latency” predicted target

- **Gamma**:

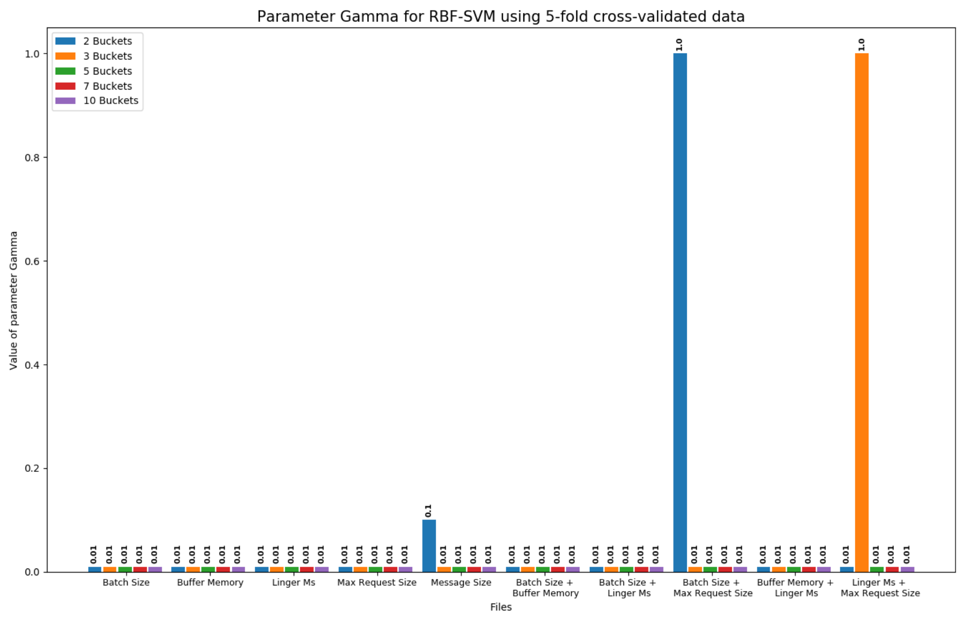


Figure 7.105: Parameter Gamma for RBF-SVM algorithm with ”Max Latency” predicted target

From the diagram we see that the hyperparameter C remains constant at the value 100, which is significantly high. With regard to the value of the hyperparameter Gamma has mainly the value 0.01 with a few upward fluctuations that reach the value of 1.

7.4.3.4 General Results for Classification algorithms

The Classification algorithms additionally provided very good results. However, it became clear that, as the number of buckets grew, so did their performance, at least most of the time. As we explained, this was due to the fact that the data at the boundaries of the bucket separation were not so clear, and the classification at those points could not be performed as efficiently. An improvement of this phenomenon was observed with the use of the RBF-SVM algorithm, which likely has to do with the nature of the algorithm and the way it creates the ”areas” within which it classifies the data.

Therefore, we conclude that the categorization algorithms are quite capable of predicting the performance of Kafka, when predicting the corresponding targets we studied. The ”Max Latency” target is perhaps an exception, as there has been a greater decline in performance as buckets increase, as opposed to other targets.

Finally, as we have already mentioned in Section 6.3, the problem of Classification algorithms involves the correct separation and assignment of data to categories, as the data themselves are continuous and do not belong to any category. The tradeoff between the highest efficiency with fewer categories and the lowest efficiency with more categories and, therefore, a more accurate of actual price forecast, is something that can be studied further and may depend on each use case of Kafka.

8. CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The purpose of this thesis was to study the Publisher/Subscriber system Apache Kafka, as well as to study whether it is possible to properly configure its parameters in order to achieve the best performance, by using Machine Learning algorithms. The two types of Machine Learning algorithms utilized are **Regression** and **Classification** algorithms. There were three stages of this thesis.

- The first step was to get acquainted with the Apache Kafka system and perform experiments so that we could take measurements. These measurements were later used as training data for the Machine Learning algorithms.
- The second step was the creation of Machine Learning models, according to the algorithms used. The models were trained with the data collected from the first stage and the predictions of the new measures/targets were made, which, in essence, showcase the performance of the system from different points of view.
- The third and final step of the work involved the visualization and interpretation of the results. The results were aggregated and plotted on the diagrams shown in Section 7.4; following that, the algorithms for their performance were then refined and tested.

The experimental results of the ML models used indicate that such techniques can be used to predict the performance of the Apache Kafka. For the Regression models and algorithms, the data would be modified according to the logarithmic function as described above, to produce better results. Respectively, for the algorithms and the Classification models, the appropriate hyperparameters were found, in order to provide the best possible result for the data. This process, however, was relatively time-consuming, depending on the case and model. Moreover, the division of data into categories was another problem that was addressed, as the data do not belong to a category by themselves. Nevertheless, the results were quite satisfactory and showcased that this family of algorithms could also be used for such purposes.

8.2 Future work

The deep study and understanding of the topics covered in this thesis, highlighted even more topics and aspects to explore. Unfortunately, the practical constraints on infrastructure and time did not allow to continue to deepen the issue even further. Nonetheless, we present below some suggestions for further work that could stem from this thesis.

The use of more computing resources, such as a computer cluster, to test the system in more realistic situations, instead of using VMs, would yield data that could be closer to actual system use; as a result, the new predictions of the Machine Learning models would be even more realistic. In addition to the ”quality” of the measurements, more computing resources could provide a larger training dataset for the Machine Learning models.

In addition, as the field of Machine Learning is so extensive, the use and testing of more models is something that could be considered in future work. A possible scenario could involve combining the different processing of the data with these new models. Even the use of Deep Learning techniques such as Neural Networks, as an alternative way to classifying the data, is something that, although it was not considered in this thesis, would be very interesting to explore and even apply to such techniques.

Finally, this entire study could further be applied to other messaging systems such as Amazon Kinesis, Microsoft Event Hubs, Google Pub/Sub, etc., with the ultimate goal being to create a system that automatically adjusts the parameters of such systems without user intervention.

ABBREVIATIONS - ACRONYMS

IoT	Internet of Things
Pub	Publisher
Sub	Subscriber
ML	Machine Learning
RPC	Remote Procedure Call
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
DSM	Distributed Shared Memory
MOM	Message-Oriented Middleware
FIFO	First In First Out
IP	Internet Protocol
QoS	Quality of Service
API	Application Programming Interface
ZAB	Zookeeper Atomic Broadcast
acks	acknowledgments
CPU	Central Processing Unit
RAM	Random-Access Memory
OS	Operating System
Lasso	Least Absolute Shrinkage and Selection Operator
LARS	Least Angle Regression
CART	Classification and Regression Trees
NP	Nondeterministic Polynomial
XOR	Exclusive Or
K-NN	K - Nearest Neighbor
SVM	Support Vector Machine
RBF	Radial Basis Function
CV	Cross Validation
MAE	Mean Absolute Error
MSE	Mean Squared Error
MedAE	Median Absolute Error
Avg	Average
MB	Megabytes
VMs	Virtual Machines

REFERENCES

[1] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *IEEE INFOCOM 2004*, volume 2, pages 918–928 vol.2, 2004.

[2] Yali Wang, Yang Zhang, and Junliang Chen. SDNPS: A load-balanced topic-based publish/subscribe system in software-defined networking. *Applied Sciences*, 6:91, 2016.

[3] Patrick Th. Eugster, Pascal Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[4] Message passing. /https://www.tutorialspoint.com/message-passing-model-of-process-communication.

[5] Wikipedia contributors. Message passing — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Message_passing&oldid=968041311, 2020. [Online; accessed 18-September-2020].

[6] Wikipedia contributors. Remote procedure call — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Remote_procedure_call&oldid=976896180, 2020. [Online; accessed 18-September-2020].

[7] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition, 2012.

[8] Margaret Rouse. TCP protocol. /https://searchnetworking.techtarget.com/definition/TCP, May 2020.

[9] Wikipedia contributors. Transmission control protocol — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Transmission_Control_Protocol&oldid=977068225, 2020. [Online; accessed 18-September-2020].

[10] Wikipedia contributors. User datagram protocol — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=User_Datagram_Protocol&oldid=978719960, 2020. [Online; accessed 18-September-2020].

[11] Distributed shared memory. /https://www.geeksforgeeks.org/architecture-of-distributed-shared-memorydsm/.

[12] Wikipedia contributors. Distributed shared memory — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Distributed_shared_memory&oldid=934636158, 2020. [Online; accessed 18-September-2020].

[13] Introduction to message queue by IBM. /https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q002620_.htm.

[14] Kai Sachs, Stefan Appel, Samuel Kounev, and Alejandro P. Buchmann. Benchmarking publish/subscribe-based messaging systems. 6193:203–214, 2010.

[15] Introduction to Apache Kafka. /https://kafka.apache.org/intro, 2020.

[16] Matthew O'Riordan. Everything You Need To Know About Publish/Subscribe. /https://www.ably.io/topic/pub-sub.

[17] AbdulFattah Popoola. Design Patterns: PubSub Explained. /https://abdulapopoola.com/2013/03/12/design-patterns-pub-sub-explained/, MARCH 12, 2013.

[18] Amazon Pub-Sub messaging. /https://aws.amazon.com/pub-sub-messaging/.

[19] Patrick Eugster. Type-based publish/subscribe: Concepts and experiences. *ACM Trans. Program. Lang. Syst.*, 29(1):6, 2007.

[20] Angelo Corsaro, Leonardo Querzoni, Sirio Scipioni, Sara Tucci Piergiovanni, and Antonino Virgillito. Quality of Service in Publish/Subscribe Middleware. /https://www.researchgate.net/publication/237100885_Quality_of_Service_in_PublishSubscribe_Middleware.

[21] How can Kafka help you? /https://www.confluent.io/what-is-apache-kafka/.

[22] Elin Vinka. What is Zookeeper and why is it needed for Apache Kafka? /https://www.cloudkarafka.com/blog/2018-07-04-cloudkarafka_what_is_zookeeper.html.

[23] Zookeeper: A Distributed Coordination Service for Distributed Applications. /https://zookeeper.apache.org/doc/current/zookeeperOver.html.

[24] Algorithmia. The importance of machine learning data. /https://algorithmia.com/blog/the-importance-of-machine-learning-data, 26 March 2020.

[25] Cogito Tech LLC. Understanding the Importance Of Training Data in Machine Learning. /https://medium.com/@cogitotech/understanding-the-importance-of-training-data-in-machine-learning-da4235332904, Aug 26, 2019.

[26] Rinu Gour. Kafka Performance Tuning — Ways for Kafka Optimization. /https://medium.com/@rinu.gour123/kafka-performance-tuning-ways-for-kafka-optimization-fdee5b19505b, Dec 17, 2018.

[27] CONFLUENT. CONFLUENT: Kafka's producer configuration. /https://docs.confluent.io/current/installation/configuration/producer-configs.html.

[28] Apache Kafka. Apache Kafka producer configuration. /https://kafka.apache.org/documentation/#producerconfigs.

[29] Apache Kafka. Apache kafka quickstart code. /https://kafka.apache.org/quickstart.

[30] Richa Bhatia. Top 6 regression algorithms used in data mining and their applications in industry. /https://analyticsindiamag.com/top-6-regression-algorithms-used-data-mining-applications-industry/, Sept 19, 2017.

[31] Apoorva Dave. Regression in machine learning. /https://medium.com/datadriveninvestor/regression-in-machine-learning-296caae933ec, Dec 4, 2018.

[32] Wikipedia contributors. Regression analysis — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Regression_analysis&oldid=977533924, 2020. [Online; accessed 18-September-2020].

[33] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[34] Nagesh Singh Chauhan. A beginner's guide to linear regression in python with scikit-learn. /https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-scikit-learn-83a8f7ae2b4f, Feb 25, 2019.

[35] Mirko Stojiljković. Linear regression in python. /https://realpython.com/linear-regression-in-python/.

[36] Nikolaos Perdikopanis. Normalization and Lasso Regression. /https://colab.research.google.com/drive/1qxN5WcI6HGz9lzMaU2bi7-5HbGwX_Zsh?usp=sharing.

[37] Wikipedia contributors. Occam's razor — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Occam%27s_razor&oldid=975663366, 2020. [Online; accessed 18-September-2020].

[38] Stephanie Glen. Lasso Regression: Simple Definition. /https://www.statisticshowto.com/lasso-regression/, Sep 24, 2015.

[39] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009.

[40] Wikipedia contributors. Lasso (statistics) — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Lasso_(statistics)&oldid=967820964, 2020. [Online; accessed 18-September-2020].

[41] Wikipedia contributors. Least-angle regression — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Least-angle_regression&oldid=941643629, 2020. [Online; accessed 18-September-2020].

[42] Efron Bradley, Hastie Trevor, Johnstone Iain, and Tibshirani Robert. LEAST ANGLE REGRESSION. *Ann. Statist.*, 32(2):407–499, 04 2004.

[43] Trisha Magdalena Adelheid Januaviani, N. Gusriani, Khafsah Joebaedi, S. Supian, and S. Subiyanto. The best model of lasso with the lars (least angle regression and shrinkage) algorithm using mallow's cp. 2019.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[45] Jason Brownlee. Classification and regression trees for machine learning. /https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/, Aug 15, 2020.

[46] Dr. Saed Sayad. Decision tree - regression. /https://www.saedsayad.com/decision_tree_reg.htm, 2020.

[47] Dheeru Dua and Casey Graff. UCI machine learning repository. /http://archive.ics.uci.edu/ml, 2017.

[48] Wikipedia contributors. K-nearest neighbors algorithm — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=K-nearest_neighbors_algorithm&oldid=977287943, 2020. [Online; accessed 17-September-2020].

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

[50] Adipta Martulandi. K-Nearest Neighbors in Python + Hyperparameters Tuning. /https://medium.com/datadriveninvestor/k-nearest-neighbors-in-python-hyperparameters-tuning-716734bc557f, Oct 22, 2019.

[51] Wikipedia contributors. Support vector machine — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=976259237, 2020. [Online; accessed 17-September-2020].

[52] Wikipedia contributors. Hinge loss — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Hinge_loss&oldid=949912210, 2020. [Online; accessed 11-October-2020].

[53] Wikipedia contributors. Radial basis function kernel — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Radial_basis_function_kernel&oldid=927526941, 2019. [Online; accessed 17-September-2020].

[54] Wikipedia contributors. Kernel method — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Kernel_method&oldid=971091524, 2020. [Online; accessed 17-September-2020].

[55] Wikipedia contributors. Mercer's theorem — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Mercer%27s_theorem&oldid=950643624, 2020. [Online; accessed 11-October-2020].

[56] Dipanjan (DJ) Sarkar. Continuous Numeric Data. /https://towardsdatascience.com/understanding-feature-engineering-part-1-continuous-numeric-data-da4e47099a7b, Jan 4, 2018.

[57] Chris Moffitt. Binning Data with Pandas qcut and cut. /https://pbpython.com/pandas-qcut-cut.html, Oct 14, 2019.

[58] Wikipedia contributors. Cross-validation (statistics) — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Cross-validation_(statistics)&oldid=975192923, 2020. [Online; accessed 17-September-2020].

[59] Wikipedia contributors. Explained variation — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Explained_variation&oldid=950924846, 2020. [Online; accessed 17-September-2020].

[60] Wikipedia contributors. Mean absolute error — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Mean_absolute_error&oldid=975662598, 2020. [Online; accessed 17-September-2020].

[61] Wikipedia contributors. Mean squared error — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=978068800, 2020. [Online; accessed 17-September-2020].

[62] Shivam Kohli. Understanding a Classification Report For Your Machine Learning Model. /https://medium.com/@kohlishivam5522/understanding-a-classification-report-for-your-machine-learning-model-88815e2ce397, Nov 18, 2019.

[63] Scikit-yb developers. Classification Report — Yellowbrick v1.1 documentation. /https://en.wikipedia.org/w/index.php?title=Mean_squared_error&oldid=978068800, Feb 26, 2020.

[64] Wikipedia contributors. Accuracy and precision — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Accuracy_and_precision&oldid=978348772, 2020. [Online; accessed 18-September-2020].

[65] Wikipedia contributors. Hyperparameter optimization — Wikipedia, the free encyclopedia. /https://en.wikipedia.org/w/index.php?title=Hyperparameter_optimization&oldid=972537954, 2020. [Online; accessed 18-September-2020].