Knowledge-Based Systems 164 (2019) 68–84

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A recommender system for component-based applications using machine learning techniques

Antonio Jesús Fernández-García [a,*], Luis Iribarne [a], Antonio Corral [a], Javier Criado [a], James Z. Wang [b]

[a] Applied Computing Group, University of Almeria, Spain
[b] College of Information Sciences and Technology, The Pennsylvania State University, USA

ARTICLE INFO

Article history:
Received 19 March 2018
Received in revised form 9 October 2018
Accepted 11 October 2018
Available online 3 November 2018

Keywords:
Machine learning
Recommender systems
Feature engineering
Feature selection
Component-based interfaces
Interaction information acquisition

ABSTRACT

Software designers are striving to create software that adapts to their users' requirements. To this end, the development of component-based interfaces that users can compound and customize according to their needs is increasing. However, the success of these applications is highly dependent on the users' ability to locate the components useful for them, because there are often too many to choose from. We propose an approach to address the problem of suggesting the most suitable components for each user at each moment, by creating a recommender system using intelligent data analysis methods. Once we have gathered the interaction data and built a dataset, we address the problem of transforming an original dataset from a real component-based application to an optimized dataset to apply machine learning algorithms through the application of *feature engineering* techniques and *feature selection* methods. Moreover, many aspects, such as contextual information, the use of the application across several devices with many forms of interaction, or the passage of time (components are added or removed over time), are taken into consideration. Once the dataset is optimized, several machine learning algorithms are applied to create recommendation systems. A series of experiments that create recommendation models are conducted applying several machine learning algorithms to the optimized dataset (before and after applying feature selection methods) to determine which recommender model obtains a higher accuracy. Thus, through the deployment of the recommendation system that has better results, the likelihood of success of a component-based application is increased by allowing users to find the most suitable components for them, enhancing their user experience and the application engagement.

1. Introduction

Launching a new software application requires an in-depth study of many variables in order to successfully achieve a good position in the market. Some of these variables are related to accomplishing a level of design and usability good enough to become established in an increasingly competitive market.

Many software projects fail when going to market because of a lack of users embracing them, even when the software application could help them improve their daily task performance or offer them a service that covers theirs needs. Regardless of whether companies consume a lot of resources on designing, enhancing usability, or marketing campaigns, if users do not quickly find the features they would like to use, the software is likely to fail. This is more evident today because software applications have greatly improved the interfaces that adapt themselves to the users' requirements.

The popularity of modern component-based web applications motivates the creation of interfaces that frequently have a vast amount of components to be discovered. Examples of these interfaces are: Google Now [1], a kind of personal assistant where each component (called "card") offers information, answers questions, or helps perform actions that users may need; Netflix [2], a streaming video-on-demand platform where each component offers information on a video's content [3,4]; Flipboard [5], a news and social media aggregation system presented in a magazine format; or Fitbit Dashboard [6], an activity tracker's user interface where users can monitor their activity, exercise, food, weight and sleep patterns.

The morphology of component-based applications is illustrated in Fig. 1. Several aspects can be considered in order to make these component-based applications most attractive to users. Adapting the user interface in real time according to the users' need is a strategy in order to create "smart" component-based applications. This can be achieved by suggesting component to users, suggesting

* Corresponding author.
E-mail addresses: ajfernandez@ual.es (A.J. Fernández-García), luis.iribarne@ual.es (L. Iribarne), acorral@ual.es (A. Corral), javi.criado@ual.es (J. Criado), jwang@ist.psu.edu (J.Z. Wang).

https://doi.org/10.1016/j.knosys.2018.10.019
0950-7051/© 2018 Elsevier B.V. All rights reserved.

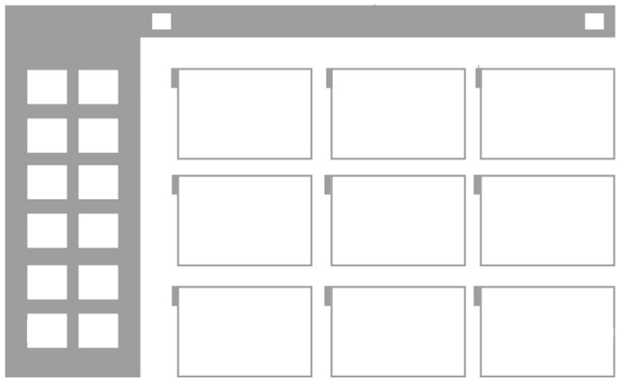


Fig. 1. Component-based applications morphology.

users the removal or replacements of components, suggesting users how they can redistribute the components they are using to make an optimal usage of the interface, as well as other adaptations more dependent of the component-based application architecture such as combining components or resizing them.

Due to the vast number of components that applications may have, the market success of a component-based web application depends not only on the ease with which the user may find components useful to them, but also on the ability of the system to identify the right components at the right time and properly organize them so they can be at the users' disposal. If users do not find useful components, they will probably discard the use of the web application, partly because of their lack of interest in the components they know. It is for that reason that we focus on recommending components, but analogously the proposed approach can also be used to suggest component removal, replacement, resizing, amongst others, or a combination of these.

The success of a web application can be partially achieved by predicting the components that are most suitable for each user at a given moment. This can be solved by using simple methods such as identifying the most frequently used components. More elaborate techniques may also be applied, for instance, through computational intelligence methods. This is an area yet to be explored because, to our knowledge, there have been no related works that explored the component-based application usage by end-users that analyzes behavior to enhance user experience and maximize the software's probability of success in the market.

The task of recommending components is nontrivial. With the passage of time, new components are added to the web application and users should be able to discover these in the event that a newly added component may be useful or of interest to them. The problem is more compounded when applications have to make suggestions to new users without prior knowledge of their activities or preferences. In these scenarios, suggestions should be made based on others, *i.e.* similar user activities, which could be difficult to identify without bias.

The problem is even more challenging today because users interact with web applications across several devices (smartphones, tablets, and potentially natural language interactive assistants) using several forms of interactions (*e.g.* touch, gesture, voice). The devices that users handle and the ways they interact with them clearly affect users' behaviors. Often it is insufficient to identify which component may be suitable for each situation or whether the usage of that component is appropriate to the form of interaction and the device that a user is handling at that specific time.

In addition, more aspects need to be considered in predicting the suitability of components to users, including the following examples: (a) The nature of the user interacting may be taken into consideration (users categorization); (b) Information may be extracted from the surrounding environment (context awareness); (c) Other relevant external information related to the web application may be identified that may enrich the prediction data.

Our work addresses the problem of creating a useful recommendation system [7] that would be able to forecast the use of components in cross-device component-based applications with multiple forms of interactions. We intend to create a recommender system that can help users discover the components most suitable for them, thereby improving their user experience of the applications. An effective recommendation system will likely improve the usability of a web application project and increase its chance of market success.

To our knowledge, there has been no previous work that deals with the creation of recommender systems specifically intended for component-based applications and their peculiarities. This fact, along with the popularity of modern component-based applications and the recent ease of access to cloud computing resources that makes the use of data analysis techniques affordable, motivated our work.

To validate the effectiveness of the recommendation system and the data analysis methods applied to suggest an example set of components to users, we have performed an empirical study on a real component-based web application with tracked data of its existing users. The application, named ENIA (Environmental Information Agent) [8], is a component-based user interface for environmental management used by the Andalusian Environmental Information Network (REDIAM, Spain) [9].

In this case study, a practical methodology is applied that in early stages, obtains a raw dataset from the ENIA application. To this dataset, *feature engineering* techniques, such as deleting features, filtering instances, transforming features data types, and merging or splitting features, among others, are applied. This is fundamental for the proper application of machine learning, because feature representation has a great impact on improving prediction models. After that, the *Mutual Information Score* and *Chi-Squared Statistics* methods are applied to obtain a subset of the dataset previously transformed by the feature engineering techniques. These *feature selection* methods automatically choose the most relevant features in the dataset and, together with the feature engineering techniques, alleviate the curse-of-dimensionality problem that appears in our case study dataset.

In summary, our main contributions are as follows:

- We describe the problem of suggesting components to users in component-based applications as a way to adapt the interface to the users' requirements at a specific time. We illustrate the process of identifying important data that need to be acquired from component-based applications to create a dataset to learn knowledge from.
- We transform the raw data directly acquired from the component-based applications through the use of feature engineering techniques to increase the predictive power of machine learning algorithms. We apply feature selection methods to avoid problems that may arise when building algorithms such as high-dimensional spaces or sparsity of data as well as to reduce computing times.
- We build models based on well-known machine learning algorithms and study their results by evaluating their efficiency and effectiveness in component-based applications environments. We provide a consistent and detailed guide about how the recommender system is deployed in a case study.

In conclusion, through these contributions, we address the problem of creating a recommender system that is able to suggest to users which are the most suitable components for them to use at a specific time, improving the user experience in the domain of component-based applications. Moreover, this recommender system reduces the limitations that these applications may have when there is a large number of components that can be offered to end users, which entails a significant increase of the possibilities to achieve a better position in the market.

The rest of the article is organized as follows. Section 2 reviews some related projects that involve creating forecasting systems or model-based recommendations models. Section 3 describes the problem of suggesting components to users of components-based user interfaces in depth and summarizes the proposed methodology. Details about creating the dataset is provided in Section 4. The main topics discussed in this section are: (a) gathering and describing the raw dataset obtained from the ENIA web application, (b) applying feature engineering techniques to transform features with better representation, and (c) applying feature selections approaches to identify the most significant features. Section 5 describes the algorithm used to build the recommendation models and analyzes the experimental results through relevant metrics. Section 6 describes how to deploy the recommender system in real component-based applications. Finally, we conclude and suggest further considerations in Section 7.

2. Related work

The evolution of data mining and big data stimulates an increasing interest in forecasting the demand for a product, the needs of a user, or the possibility of a series of events happening. Accurately forecasting, precisely identifying trends, and the discovery of behavior patterns can optimize resource usage or consumption, generate new knowledge in science and research, and enable faster and better decisions in politics, retail, weather, sport, science, research, real estate, sports or health-care, among many others fields.

For these reasons, the use of recommender models [7] to forecast the use of resources, to anticipate user needs, or to make recommendations based on their behavior (and that of other users, including a situation context) are becoming more prevalent. A compilation of some forecasting and recommender models using supervised machine learning algorithms found in the literature are featured below.

In [10], the authors presented a framework to improve user recommendation in social networks. They capture user preferences involving both interest and social factors to create a model using matrix factorization [11] techniques, based on the principle that users who have shown similar interests in the past will likely have similar interests in the future.

The authors of [12] created a multigraph ranking-based recommender system to find potential items for users. They improve previous attempts based on collaborative filtering algorithms by incorporating different types of user relationships such as friendships, neighborhood or colleagues that alleviate the sparsity of data. It is possible to obtain such relationships thanks to the growth of online social networks. By following the same line, the authors of [13] made use of hypergraph ranking to provide a solution that is able to handle complex and changing demands for e-commerce customers. Due to the nature of the recommendations, traditional single-objective recommender systems are limited, *e.g.*, a customer will not always go to the same restaurant even though it is the "best-fit", the choice may be impacted by companions and the time-location context. For that reason, authors propose a multi-objective recommendation framework capable of handle these situations.

Given the difficulty of modeling a real-world recommendation system, sometimes it is useful to create hybrid models that combine a model and knowledge of the problem addressed. In [14] the authors created a hybrid model-based job recommendation system using statistical relational learning [15]. They also took into account tuning the algorithm to satisfy the requirement to give more weight to the more desirable classes (or difficult ones).

There have also been approaches that work with similar algorithms used in this article such as decision trees [16], logistic regression [17], or artificial neural networks [18]. In [19], the authors used data analysis approaches for predicting the sales of newly published books. The authors of [20] used contextual information as a way of recognizing human activities and, subsequently, make music streaming recommendation accordingly. There are more works that focus on multiple purposes such as [21] in which the authors addressed the problem of detecting cell mitosis using deep neural networks, [22] that deals with the detection of malicious webmail attachments, or [23] that focuses on predicting intervals for solar energy forecasting with neural networks.

In our study, we also make use of context-aware information as we consider it to be valuable. Works that deal with context-aware information to create recommendation models can also be found in the literature. The authors of [24] proposed that contextual features may be considered to be organized in an understandable and intuitive hierarchy and exploited this hierarchical structure to improve the quality of the recommendation models. The exploitation of context information to improve models is frequently used [25]. For instance, in [26] the authors incorporated contextual situations (*e.g.*, geographical location, special date or activities of interest) to improve the precision of a retail recommendation system using the collaborative filtering approach. The authors of [27] made use of smartphone's context-aware capabilities to create an adaptive and personalized mobile learning system, which aims to support the semi-automatic adaptation of learning activities and helps students complete the learning activities of an educational scenario.

Often, before the creation of a recommendation model, feature selection techniques are used to improve the dataset quality (among other purposes). We also apply feature selection techniques to improve our model's accuracy, deal with high-dimensional spaces and to solve problems such as the sparsity of data. Many works use feature selection before building models. Examples in different areas include spam detection [28], fraud detection [29], and diagnosis of mechanical faults [30], among others.

In relation to component-based user interfaces, there are several works that focus on the problem of providing the best possible interface for users in concrete situations. This goal can be achieved by using different approaches. In [31], a model-driven method is proposed to generate adapted and customized UIs according to user preferences and/or capabilities based on rules. In [32], rules are used to create smart users interfaces that adapt component-based interfaces to the users' need in real time by using model transformation. Both works lack intelligent techniques to produce new rules. Although they address the adaptation problem differently, both of them are based on static predefined rules.

There exists an approach that use intelligent methods in mashup (component-based) applications to discover patterns of how users manage some task [33], but it is used to improve how certain tasks should be managed and not to adapt the interface to users. [34] studied the inter-component communication in distributed systems and how to make decisions for selecting control flow alternatives. It is an approach focused on the components instead of the users. In [35], authors focused on users by providing a ranking of user intentions. However, they did not provide how to implement it on component-based interfaces (or any other) and did not clearly explain how to obtain or process the data, or to exploit the knowledge inferred.

The study of bibliography led us to research on how intelligent methods can be applied to improve the way of enhancing the user experience in component-based interfaces. Furthermore, works such as [36,37] provide cloud infrastructure for managing component-based user interfaces through web services and deal with managing components or inter-dependencies and other peculiarities of such interfaces.

Table 1 compare the related work mentioned in this section in aspects such as:

(a) use of machine learning;
(b) use of feature selection;
(c) use of feature engineering;
(d) implementation of real-time recommendations;
(e) user interfaces adaptation;
(f) application to component-based applications; and
(g) use of context awareness.

Given the related works, it makes sense to explore the use of machine learning algorithms to create a recommendation model for suggesting the most suitable component for users on component based interfaces to enhance user experience.

3. Methodology and problem description

The steps to create the recommendation models on component-based web applications are illustrated in Fig. 2. These steps do not represent a formal methodology. Instead, they are guidelines we have followed in carrying out the work described in this paper. While they follow a typical data-driven modeling pipeline, the application of any of these steps is optional, subject to the nature of the problem presented and the data scientists' knowledge.

The first step, understanding the problem, is already briefly discussed in the introduction. It can be summarized as follows: we need to create a recommendation model that assists users in component-based web applications to discover useful components, optimizing the user engagement with the application. The aim of the recommendation model is to maximize the potential market success of a component-based web application.

In order to really understand the problem of the ENIA component-based application case study, there are important facts to be considered (which can be extrapolated to other component-based applications):

(a) The set of components offered by the web application are in continuous growth, with new components added (or removed) over time.
(b) New users are registered in the web application with a degree of frequency.
(c) Users are categorized according to how they are expected to use the application, *e.g.*, a tourist or a farmer.
(d) Users interact with the interface from different devices (smartphones, tablets, wearables, laptops, desktops PCs, etc.) with different forms of interaction (mouse and keyboard, touch screen, voice, etc.).
(e) The context that surrounds the users' interactions with the web application is relevant to the way they behave, *e.g.*, weather condition.

The second step is gathering the data from the component-based web application [38]. Fortunately, an increasing number of applications have data repositories and a growing number of organizations and companies now store data, sometimes leaving it freely available to everyone (OpenData). If this were not the case, this step would be tedious because of the need of manual data collection. In this case, the ENIA application has a service that stores all of the information related to each interaction performed over its user interface in a MySQL database [39]. A CSV file containing all the interaction can be requested and will be generated using the stored data in the ENIA database.

The third step consists of processing the raw data obtained in the previous step. Finding out the relevant attributes that describe an interaction is vital to ensure the success of the recommendation system. These attributes are called *features* and the set of features is called a *dataset*. This step includes cleaning features and transformation, as well as an important and laborious process known as feature engineering that we will be described later.

The fourth step consists of applying feature selection methods that, as we shall see later, handle many issues related to data such as the sparsity, in addition to reducing the computing resources needed by machine learning algorithms in creating the models.

The fifth step consists of applying machine learning algorithms to model the recommendation system we are looking for. A wide range of algorithms can be applied. In this work, we make use of decision trees, logistic regression, and artificial neural networks.

The last step consists of validating the recommendation model built by analyzing the accuracy of the algorithms used. A comparison of the experimental results of the applied algorithms is made to determine whether the recommendation models created are suitable to be deployed in the component-based web application and which recommendation model is expected to provide better results.

4. Creating optimal datasets

We now focus on creating the optimal datasets for applying machine learning algorithms. We shall maximize the possibilities of suggesting the proper component to users in every given moment. Thus, the recommendation system that will be built in further steps will be truly useful to end users.

First, we obtain raw data from the ENIA component-based user interface. Next, we use feature engineering techniques to help machine learning algorithms create accurate and simple prediction models, providing better results [40]. We then apply feature selection methods to create a subset with more meaningful features of the processed dataset which will imply shorter training times to build the models by removing redundant or irrelevant features. Fig. 3 conceptually illustrates all processes that are involved in creating an optimal dataset from raw data.

4.1. Gathering raw data

In order to create a recommendation system on component-based applications and suggest useful components to users, it is necessary to create datasets with rich enough attributes to model the problem in the first place. As a starting point, we have a raw dataset provided by our case study component-based application, ENIA. The data contained in the dataset has been directly extracted from the component-based web application through a data acquisition system that is incorporated in the proprietary software application [41,42] powered by `COSCore`, a modular and scalable service infrastructure approach for the management of component-based architectures [36,37].

When the dataset was obtained, it consisted of 27,702 interactions performed (instances) with a set of 18 features. The set of features describing each interaction can be seen in Table 2.

4.2. Feature engineering

Table 2 describes the data obtained from the component-based web application. It contains data too raw for direct application of data analysis techniques. In agreement with the REDIAM agency and their experts in the field, some transformations on the dataset

Table 1
Summary of related work. ML: (use of) Machine Learning, FS: (use of) Feature Selection, FE: Feature Engineering (explained), UI: (applied to) User Interfaces, CBA: (applied to) Component-Based Applications, RTR: Real-Time Recommendations, CA: Context Awareness.

Research work	Objective	ML	FS	FE	UI	CBA	RTR	CA
Alelyani et al. (2013), [35]	Users intentions detection	✓			✓			
Armentano and Amandi (2011), [34]	Inter-component communications				✓	✓		
Castillo et al. (2017), [19]	Predicting book sales	✓	✓					
Cohen et al. (2018), [22]	Spam Detection	✓						
Criado et al. (2017), [32]	Component-based apps. adaptation				✓	✓	✓	✓
Galvan et al. (2017), [23]	Solar energy forecasting	✓						
Ghaibi et al. (2017), [31]	User interfaces adaptation				✓	✓		
Gomez et al. (2014), [27]	Mobile learning	✓			✓		✓	✓
Hajek and Henriques (2017), [29]	Fraud detection	✓	✓	✓				
Lee et al. (2017), [20]	Music recommendation	✓	✓		✓		✓	
Mao et al. (2017), [12]	Items recommendation	✓						✓
Mao et al. (2017), [13]	Multi-objective recommendations	✓						✓
Portugal et al. (2018), [7]	Multiples objectives	✓	✓		✓			
Rodriguez et al. (2017), [33]	Users patterns detection	✓			✓	✓		
Sanchez et al. (2017), [26]	Retailing recommendations	✓		✓				✓
Wei et al. (2017), [30]	Mechanical faults detection	✓	✓	✓				
Xu et al. (2018), [10]	Social media recommendations	✓						
Yang et al. (2017), [14]	Job recommendations	✓						
Zhang et al. (2014), [28]	Spam detection	✓	✓	✓				
Zhou et al. (2017), [21]	Cell mitosis detection	✓						
(Our proposal)	Users components suggestions	✓	✓	✓	✓	✓	✓	✓

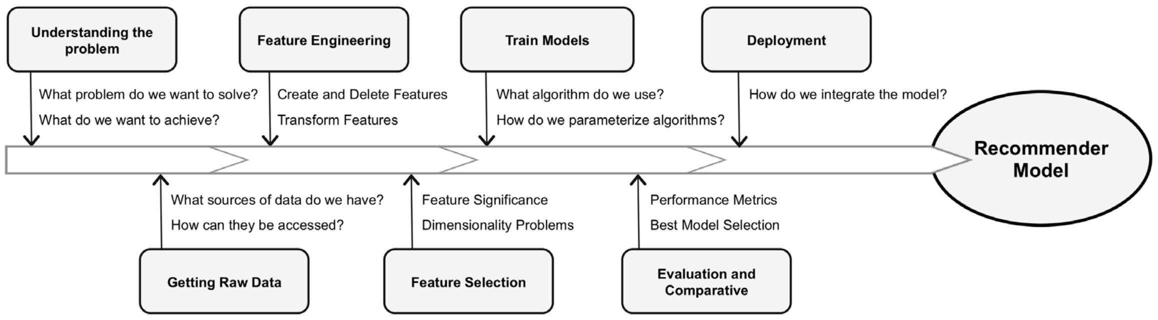

Understanding the problem
Feature Engineering
Train Models
Deployment
What problem do we want to solve?
What do we want to achieve?
Create and Delete Features
Transform Features
What algorithm do we use?
How do we parameterize algorithms?
How do we integrate the model?
Recommender Model
What sources of data do we have?
How can they be accessed?
Feature Significance
Dimensionality Problems
Performance Metrics
Best Model Selection
Getting Raw Data
Feature Selection
Evaluation and Comparative

Fig. 2. Methodology applied to create the recommendation model.

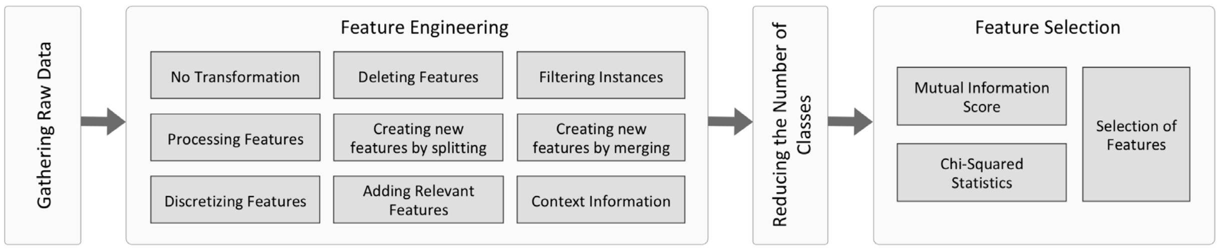


Fig. 3. Processes involved in creating an optimal dataset from raw data.

and its features have been made using feature engineering techniques.

Feature engineering is a process that transforms raw data to create features that have better representation and therefore better to create predictive models [43]. The quality of the prediction models created with machine learning algorithms depends on the feature engineering approach that has been followed, where usually better features result in better prediction models. Although feature engineering is usually treated lightly, it plays an important role in the success of a machine learning experiment. These techniques optimize the performance of the dataset, improve the data analysis algorithms's results, and create simple and reliable prediction models that perform well and accurately [40].

The following feature engineering transformations have been applied to the original dataset:

(a) **No transformation**. There are features that have good significance in the form they appear in the original dataset. For this kind of features, no transformation is necessary. For this reason, the `actionComponent`, `deviceType`, `interactionForm`, `countryOrigin`, `areaOrigin`, and `userType` features have been left unaltered. The value they add to the dataset is considered significant enough as is.

(b) **Deleting Features**. There are features that do not correlate well with the objective field (label). Some are easy to identify because they have no knowledge associated with the field. Others are more difficult to identify because they are dependent on other features or domain-specific knowledge is required. It is important to identify these features and remove them. The `idSession`, `idUserClient`, and `idInteraction` features have been discarded because they are not relevant to the accuracy of the model – they are simply auto-generated identifiers created by the ENIA database. The `idUserSubType` feature has also been discarded because it is an optional field that some users miss. According to

Table 2
Set of features describing an interaction directly obtained from the ENIA web application.

Feature	Description
`idInteraction`	Unique Interaction Identification
`dateTime`	Exact moment in which the interaction took place
`operationPerformed`	Kind of interaction performed in the application
`latitude`	Latitude in which the interaction took place
`longitude`	Longitude in which the interaction took place
`idUserClient`	Unique User Identification
`Name`	Name of the user that performs the interaction
`Surname`	Surname of the user that performs the interaction
`userType`	Type of user that performs the interaction (tourist, farmer, etc.)
`userSubType`	Subtype of user that performs the interaction (tourist, farmer, etc.)
`birthDate`	Birth date of the user that performs the interaction
`countryOrigin`	Country of origin of the user that performs the interaction
`areaOrigin`	Area/State of the Country of the user that performs the interaction
`email`	Email of the user that performs the interaction
`deviceType`	Type of device on which the interaction was performed (laptop, desktop, smartphone, etc.)
`interactionType`	Form of interaction on which the interaction was performed (mouse&keyboard, touch, voice, etc.)
`idSession`	Unique Session Identifier
`actionComponent`	Component over which the interaction has been performed

the REDIAM experts, ENIA does not provide a good categorization of users subtypes. The `areaOrigin` feature has also been removed. On this occasion, because it has many possible cases, *i.e.*, a large amount of values can be assigned to these features (17) and we already has the `countryOrigin` feature that is similar and has more homogeneous values. In the future, with a higher number of instances, the `areaOrigin` feature could be kept.

(c) **Filtering Instances**. Often, the whole set of instances are not required because the problem is circumscribed to a fraction of it. On these occasions, it is necessary to obtain a subset of the original dataset according to a rule that can isolate the required instances that are useful to create the model. In our case study dataset, the `operationPerformed` feature has 13 possible values: {`Add`, `Group`, `AddGroup`, `Ungroup`, `Delete`, `UngroupGroup`, `UngroupDelete`, `ResizeBigger`, `ResizeSmaller`, `ResizeShape`, `Move`, `Maximize`, `Minimize`}. These are the possible operations that users can perform in the ENIA component based application user interface. Because our purpose is to recommend components to users, we are interested in filtering the `operationPerformed` feature to keep the instances where the `operationPerformed` values are {`Add`, `AddGroup`}. These values of the `operationPerformed` feature bring together the operations where users begin using a component.

(d) **Processing Features**. There are features that, according to domain-specific experts, can be useful but algorithms may find them difficult to manage. These features should be processed to adapt them to more suitable datatypes, or representations that increase their significance and may be easily processed using data analysis algorithms. In our case study, the `birthDate` feature is too complex to obtain significance from it, thus it has been transformed to a new feature: the age the user was when he/she performed the interaction (`age`), which performs better.

(e) **Creating new features by splitting existing ones**. Sometimes there are features rich enough to be broken down into more than one feature because they contain a great deal of information in a specific domain. The ENIA original dataset comes with the `dateTime` feature, which contains the exact time when an interaction was performed. It has been divided into two features: `season` and `partOfTheDay`, which allows us to explore two different aspects that surround the interaction and may influence the user's behavior.

(f) **Creating new features by merging existing ones**. In contrast to the previous case, sometimes there are features that by themselves do not contribute to creating better prediction models. However, by merging them with others features, it is possible to create meaningful features that improve the overall performance. The `latitude` and `longitude` features do not add relevant information to the recommendation systems by themselves, but they can be transformed into two interesting new features: `countryInteraction` and `areaInteraction`.

(g) **Adding relevant features**. Often, there is meaningful data that is not contained in the dataset but can be obtained from existing features. Because we are talking about a component-based web application developed for an environmental agency, it is considered relevant, according to the experts, to know the weather conditions at the time that the interactions have been carried out. A piece of code has been implemented that connects to the `OpenWeatherMap` [44] service and using the `latitude`, `longitude` and `dateTime` data features as inputs can retrieve the weather conditions. Thus, two new features have been inserted in the dataset: `weatherDescription` and `Temperature`.

(h) **Discretizing Features**. Many machine learning algorithms produce better prediction models using discretized values. By discretizing, the impact that small variations have on the data is reduced, which justifies the loss of information [45]. We performed the following discretizations, assisted by the experts:

— The numeric values of the `age` feature are calculated (as described before) and categorized into the following values:

{*<18, 18–25, 25–35, 35–50, 50–65, >65*}

— The values of the `season` feature have been categorized according to the values:

{*spring, summer, fall, winter*}

— The values of the `partOfTheDay` feature have been categorized according to the values:

{*morning, afternoon, evening, night*}

Table 3 synthesizes the transformation that has been taken place in each feature.

After the application of the *feature engineering* process and observing the set of features obtained, it can be clearly seen the importance of features related with the environment of the component-based application. In many occasions, this contextual information contributes to useful data. Considering this, the access to the software application through a specific device also implies that the user is looking for certain functionalities that may or may not coincide from one device to another. This is known as context awareness, that is, the ability of a device to gather information about its context, environment, location, and other nearby resources to behave accordingly. In most cases, contextual information helps to understand the human behavior when using a user interface. This contextual information is collected by multiple sensors or by accessing external web services with related data.

Table 3
Features transformation carried out in the feature engineering process.

Before FE	After FE	Consideration
idInteraction	~~idInteraction~~	Deleted
idUserClient	~~idUser~~	Deleted
idSession	~~idSession~~	Deleted
operationPerformed	~~operationPerformed~~	Deleted. Instances filtered by add or addGroup
name	~~name~~	Deleted
surname	~~surname~~	Deleted
email	~~email~~	Deleted
userType	userType	Kept
userSubType	~~userSubType~~	Deleted
birthDate	Age	Processed
deviceType	deviceType	Kept
interactionType	interactionType	Kept
dateTime	season	New Feature (Splitting)
	partOfTheDay	New Feature (Splitting)
	temperature	New Feature (from external sources)
	weatherDescription	New Feature (from external sources)
latitude	Country Interaction	New Feature (Merging)
longitude		
countryOrigin	countryOrigin	Kept
areaOrigin	~~areaOrigin~~	Deleted
actionComponent	actionComponent	Objective field

Bearing in mind this contextual information when designing software to acquire interaction data from the user interface, we can find three possible sources for collecting the features: the component-based applications themselves, their users and devices, and external third-party components interacting with the component-based application.

In our ENIA case study, we consider it necessary to collect some contextual information alongside the interaction data because experts think it can contribute to a better understanding of the interactions produced. Some useful contextual information considered in the ENIA project has been, for instance:

- Features related to the user: `userType`, `userSubType`, and `birthDate`.
- Features related to the device: `deviceType` and `interactionType`.
- Features related to the moment and place of interaction: `date`, `time`, `latitude`, `longitude`, `countryInteraction`, and `areaInteraction`.
- Features related to the weather: `temperature`, `pressure`, `humidity`, `cloudPercentage`,`windDirection`, and `windSpeed`.

As seen in the feature engineering process some of these features have been discarded, other has been transformed, split, or merged with other features. In either case, the importance of acquiring and handle these features related to the context awareness contributes to creating useful dataset that contains feature with great significance for training predictive models.

4.3. Reducing the number of classes

Classification algorithms in machine learning can be categorized as binary or multiclass, according to the number of classes they predict. In this study, we attempt to forecast the most feasible components to suggest to users in specific situations. Thus, the algorithm must determine among the available components which one is the most suitable. This is a multiclass problem. It would be different if the model were to foresee whether a component is suitable or not, which would be a less demanding binary problem.

After the feature engineering process, our dataset consists of 27,702 instances with a set of 11 features (including the objective field). The possible values that can be assigned to each feature and its data type is described in Table 4 and summarized as follow:

```
 Dataset = {
   weatherDescription (4),
   temperature (7),
   countryInteraction (4),
   countryOrigin (4),
   userType (4),
   age (6),
   season (4),
   partOfTheDay (4),
   deviceType (3),
   interactionType (3),
   actionComponent (33)
 }
```

The number of cases in the `actionComponent` feature, that is the objective field, is too high. With 33 different components that can be suggested and 27,702 instances to train the model, it is highly probable that the forecasting would not be accurate. According to the number of cases of each feature, the total number of possible cases is more than 50 million ($4 \times 7 \times 4 \times 4 \times 4 \times 6 \times 4 \times 4 \times 3 \times 3 \times 33 = 51.093.504 \approx 5 \times 10^9$). If we had not deleted the `areaOrigin` and `areaInteraction` features the total number of possible cases would have been more than 10 billion ($4 \times 7 \times 4 \times 12 \times 4 \times 17 \times 4 \times 6 \times 4 \times 4 \times 3 \times 3 \times 33 = 10.423.074.816 \approx 10^9$). With this action the problem is drastically reduced, but still not enough.

For this reason, the number of components to be suggested has been limited. A reduction of the 33 possible classes of the objective field has been made, reducing the number of classes to eight. We have selected the components that are the most frequently used. The recommendation system is then set up to suggest only these eight components to users. It is remarkable that the appearances of these eight cases selected account for 21,882 instances ($\approx$ 78%). The possible cases of the `actionComponents` feature that can be predicted are `BioReserves`, `Clock2`, `CuttleRoads`, `GeoParks`, `Heritage`, `Twitter`, `Weather` and `Wetlands`. The number of instances of each class and its frequency can be seen in Table 5. Thus, the recommendation system is a hybrid between computational intelligence methods and a simpler method that selects the most frequently used eight components of the ENIA.

Even after the objective field classes reduction, the number of possible cases is more than 12 million ($4 \times 7 \times 4 \times 4 \times 4 \times 6 \times 4 \times 4 \times 3 \times 3 \times 8 = 12.386.304 \approx 12 \times 10^6$). It remains a large number, but given the high number of occurrences of these

Table 4
Set of features describing an Interaction after processing the original raw dataset.

Feature	Cases (Possible values)	Type
`weatherDescription`	Clouds, Clear, Fog, Mist	Categorical feature
`temperature`	<−10, −10–0, 0–10, 10–20, 20–30, 30–40, >40	Categorical feature
`countryInteraction`	Not limited possibilities. When creating the dataset: 4 cases	Categorical feature
`countryOrigin`	Not limited possibilities. When creating the dataset: 4 cases	Categorical feature
`userType`	Tourist, Farmer, Technical, Politician	Categorical feature
`age`	<18, 18–25, 26–35, 36–50, 51–65, >65	Categorical feature
`season`	Spring, Summer, Fall, Winter	Categorical feature
`partOfTheDay`	Morning, Afternoon, Evening, Night	Categorical feature
`deviceType`	Computer, SmartPhone, Tablet	Categorical feature
`interactionType`	MouseKeyboard, Touch, Voice	Categorical feature
`actionComponent`	Not limited possibilities. When creating the dataset: 33 cases. Limited to 8.	Categorical label

Table 5
Most frequently used components.

Class	Instances	Frequency
`GeoParks`	5572	0.255
`Heritage`	4760	0.218
`CuttleRoads`	3038	0.139
`Twitter`	2618	0.120
`Weather`	2282	0.104
`Wetlands`	1400	0.064
`BioReserves`	1302	0.060
`Clock2`	910	0.042
Total	21882	1

classes, and the high appearances frequency of the selected classes, it is enough. The domain-specific experts, *i.e.*, employees of the environmental agency that makes use of the application to perform their tasks, agree with this simplification. Besides, following their advice, the recommendation model will not suggest components to users in every situation, it will only suggest on the occasions that the likelihood of using a component is sufficiently high to make a good recommendation. For example, if the recommender system suggests a component with a likelihood of 37%, the recommendation is not shown to users because we have set a threshold of 50% of likelihood.

4.4. *Feature selection*

The set of attributes of a dataset should be as significant as possible to describe the nature of the reality they represent. It may seem that a large number of features can better describe a problem and (with them) better predictive models can be built, but this is not entirely true. Feature selection methods can help reduce overfitting [46] and avoid high-dimensional spaces and the sparsity of data [47] that datasets with a large set of attributes may have. Also, a feature subset of a dataset results in shorter training times in building the models by removing redundant or irrelevant features.

In this work, we make use of feature selection methods that support all data type features, specifically the *Mutual Information Score* and *Chi-Squared Statistic* methods available in the Microsoft Azure Machine Learning Studio (AzureML) [48].

4.4.1. *Mutual Information Score*

The *Mutual Information Score* method is defined as:

$$I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p(x,y)\,\log\left(\frac{p(x,y)}{p(x)\,p(y)}\right), \tag{1}$$

where $p(x,y)$ is the joint probability function of X and Y, and $p(x)$ and $p(y)$ are the marginal probability distribution functions of X and Y respectively [49].

Table 6
Mutual Information Score and *Chi-Squared Statistics* results. MR: Mutual Information Result; MO: Mutual Information Order; CR: Chi-Squared Result; and CO: Chi-Squared Order.

Feature	MR	MO	CR	CO
`userType`	0.078864	1	383.256110	1
`age`	0.036541	2	170.996097	2
`countryInteraction`	0.022832	3	106.796770	4
`weatherDescription`	0.020199	4	91.611703	5
`partOfTheDay`	0.018090	5	73.870284	6
`countryOrigin`	0.017928	6	121.178845	3
`deviceType`	0.003489	7	14.995899	7
`interactionType`	0.002720	8	11.730771	8
`season`	0.001766	9	7.233112	9
`temperature`	0	10	0	10

Table 6 shows the results of applying the *Mutual Information Score* selection method to the ENIA dataset and Fig. 4 illustrates the relevance of each feature.

The relevance of the `userType` feature is far higher than any other feature, followed by the `age`, `countryInteraction` and `weatherDescription` features, that also have considerable weight. Conversely, the `deviceType` and the `interactionType` features have little relevance, probably because the vast majority of users interactions has been performed in desktop computers by means of mouse and keyboard. The `temperature` feature has no relevance, probably because that feature can be highly correlated with the `season` and `partOfTheDay` features.

4.4.2. *Chi-Squared Statistic*

The *Chi-Squared Statistic* method is defined as:

$$X^2 = \sum_{i=1}^{n}\sum_{j=1}^{m}\frac{\left(O_{(i,j)} - E_{(i,j)}\right)^2}{E_{(i,j)}}, \tag{2}$$

where $O_{(i,j)}$ is the observed value of two nominal variables and $E_{(i,j)}$ is the expected value of two nominal values. The expected value can be calculated with the following formula:

$$E_{(i,j)} = \frac{\sum_{i=1}^{c} O_{(i,j)}\ \sum_{k=1}^{c} O_{(k,j)}}{N}, \tag{3}$$

where $\sum_{i=1}^{c} O_{(i,j)}$ is the sum of the i_{th} column and $\sum_{k=1}^{c} O_{(i,j)}$ is the sum of the k_{th} column [50].

Table 6 shows the results of applying the *Chi-Squared Statistics* selection method to the ENIA dataset and Fig. 4 illustrates the relevance of each feature.

When we observe Fig. 4 and Table 4 we appreciate that there is not very much difference. Both *Mutual Information Score* and *Chi-Squared Statistic* methods have thrown similar results. The main difference is that the weight of the `countryOrigin` feature

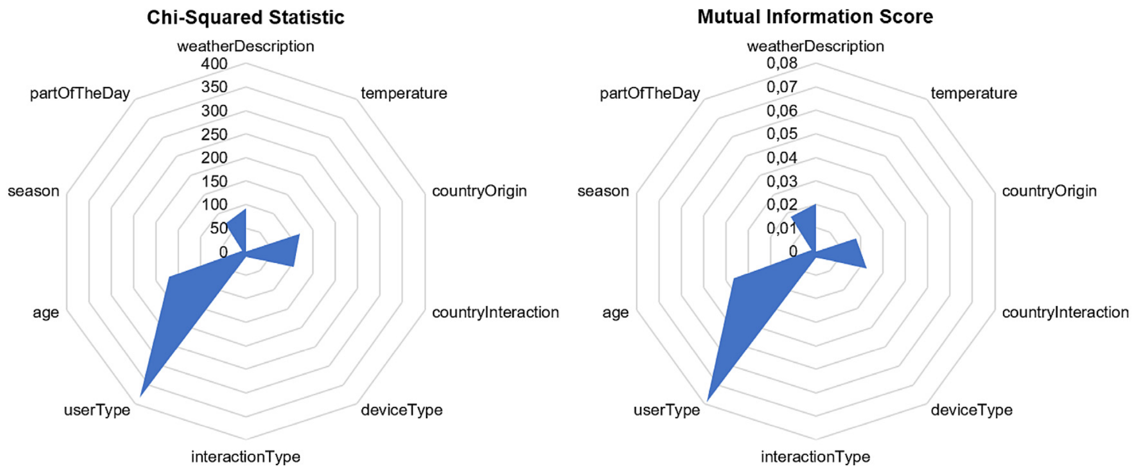


Fig. 4. *Mutual Information Score* and *Chi-Squared Statistic* features relevance.

in the *Chi-Squared Statistic* method is higher than the `country-Interaction` and `weatherDescription` features in comparison with the *Mutual Information Score* method.

4.4.3. Features selected

According to the results shown in Table 6, the best features that can be selected to create a subset of the original dataset are:

```
Dataset = {
  userType,
  age,
  countryOrigin,
  countryInteraction,
  weatherDescription,
  partOfTheDay,
  deviceType,
  interactionType,
  actionComponent (label)
}
```

Taking into account the finding of the feature selection methods, the `deviceType` and `interactionType` features would have been discarded, but we have kept them since we expect that soon enough, the ENIA web application user access from smartphones or tablets will increase and we want this recommendation system to automatically evolve over time. When creating the models, each is built using both datasets, the original and the subset of the original with the most significant features. The results are evaluated and their performance compared.

Analyzing the results of the *Mutual Information Score* and *Chi-Squared Statistics* algorithms, it can be seen that: a) both of them have a high degree of similarity in the results thrown; and b) `userType` and `age` are the features with more significance in the dataset with a great difference.

The similarity between the algorithms was expected because both of them are filter methods, independent of machine learning algorithms, based on statistical methods such as linear dependency, variance or frequency distribution among others, that measure the correlation between all the features and the objective field feature. The small correlation differences between the algorithms lies in how they work internally. The *Mutual Information Score* algorithm analyses how much it is known of the objective field feature through each of the rest dataset features and the *Chi-Squared Statistics* algorithm analyzes the discordance between the objective field feature result and each of the remaining dataset features.

In the same vein, it also makes sense that `userType` and `age` features have greater significance than any other feature in the dataset, according to the experts in the field. The `userType` feature clearly segments the components that are of interest to users in the component-based application. Tourists are likely to use a set of components while farmer are likely to use others, and accordingly, this feature has a great significance and high degree of linear dependency with the objective field. The `age` feature also has a great significance, probably due to the characteristics of the components that users in the main range areas consume have certain similarities between them.

5. Recommendation model and experiments

This section analyses the machine learning algorithms used to create the recommendation models and presents the results of the experiments conducted. We evaluate the recommendation model's accuracy and performance based on relevant metrics obtained from the empirical study, to determine which recommendation model is more suitable to deploy. By selecting the more accurate model the recommendation system will provide ENIA users with the most suitable components, thereby improving their user experience.

5.1. Recommendation model

There are plenty of algorithms available in the literature that can be applied in order to create recommendation models. Likewise, by adjusting the parameters of these algorithms, there are many more configurations possible that directly affect their performance and accuracy.

In this case, it is worth noting that we are facing a multiclass classification problem, where each instance can be classified into more than two classes, specifically, in as many classes as components are available to suggest to users. Although the possibility exists of turning binary classification algorithms into multiclass classifiers, we have decided to make use of algorithms that naturally allow decisions between more than two classes.

We make use of four multiclass classification algorithms available in Microsoft Azure Machine Learning Studio [48]. We make use of the AzureML implementation because of the ease of setting up web services over trained models to deploy them that this platforms offers. We also value positively the fact that readers can easily access these algorithms as anonymous guests on the platform for free and reproduce the experiments conducted in this paper.

The well-known classification algorithms proposed are multiclass decision forest, multiclass decision jungle, multiclass neural networks, and multiclass logistic regression.

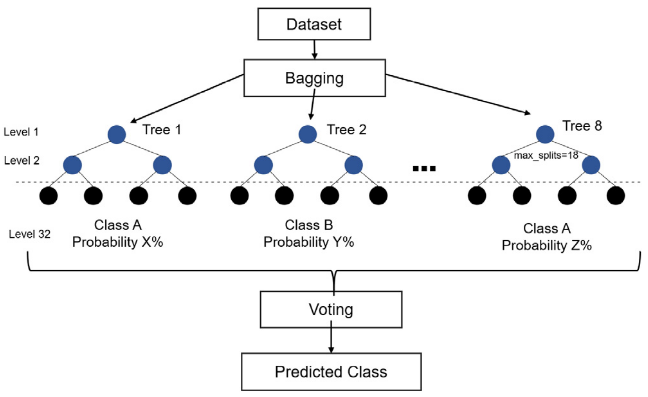


Fig. 5. Decision forest parametrization.

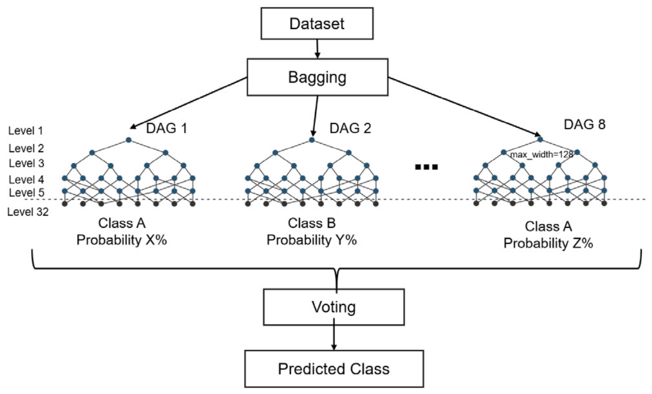


Fig. 6. DAG parametrization.

5.1.1. Multiclass Decision Forest (DF)

Decision trees algorithms build a tree-like structure where each node represents a question over an attribute. The answers to that question create new branches to expand the structure until the end of the tree is reached, being the leaf node the one that indicates the predicted class. The *Decision Forest* (DF) algorithm, graphically illustrated in Fig. 5, creates several decision trees and votes the most popular output of them. The implementation used in this paper does not directly count the output of them but sum the normalized frequency ($\hat{f}$) of each output in each tree to get the label with more "probability":

$$\hat{f} = \frac{1}{T}\sum_{t=1}^{T} f_t(x) \,, \tag{4}$$

where T is the number of trees and x the probability of each class.

The parametrization followed in this experiment built 8 decision trees with a maximum depth of 32 levels each. The amount of 128 splits are generated per node to select the optimal split. In order to generate a leaf node, just a sample is required. To resample the dataset for each decision tree the *Bagging* method (or *bootstrap aggregation*) is applied [51].

The bagging method consists of duplicating the original dataset D to a new dataset D' for each decision tree. D and D' have the same size n. When creating and evaluating models the original dataset is divided in two parts: *training* and *evaluation*. The instances used for *training* and for *evaluation* are different in each new dataset D' since they are selected randomly with replacement.

5.1.2. Multiclass Decision Jungle (DJ)

The *Decision Jungle* (DJ) algorithm is an extension of the *Decision Forest* algorithm where each tree is replaced by a DAG (directed acyclic graph). The structure of a DAG is illustrated in Fig. 6. It is more memory-efficient because it eliminates the need for repeating leaf nodes and allows branches to merge but, as a disadvantage, takes more computing time.

The parametrization followed in this experiment built 8 decision DAGs with a maximum depth of 32 levels each. The maximum width of each decision DAG is 128 and the number of steps for each level optimization of the graph is 2048. As in the case of the decision forest algorithm, to resample the dataset for each decision DAG the *Bagging* method is applied.

5.1.3. Multiclass Artificial Neural Network (ANN)

The *Artificial Neural Network* (ANN) algorithm creates a set of interconnected levels, where each level consists of a set of nodes (neurons) that receives input and produces weighted outputs. The nodes of layer 1 are the inputs, the nodes of the last layer are the output and the nodes in between are called "hidden nodes". A neural network can be seen as a weighted directed acyclic graph.

The parametrization followed in this experiment built an ANN with a fully connected structure, *i.e.*, every node of each layer is connected with every node of the previous and next layer. There is only 1 hidden layer with 100 nodes. The instances are processed a maximum of 100 times if the ANN has not converged yet. The learning rate is set to 0.1 and the momentum (weight applied to nodes from previous iterations) is set to 0, with these values we want the model to converge fast but avoid local minimums.

5.1.4. Multiclass Logistic Regression (LR)

The *Multiclass Logistic Regression* (LR) algorithm or *Multinomial Logistic Regression*), generalizes logistic regression with more than two possible classes to predict. It predicts the output probability of a class by fitting data to a logistic function. LR aims to build a function that describes the relationships between the input features and the class label.

The parametrization followed in this experiment to build the logistic regression model set the *Optimization Tolerance* 1×10^{-6}. This parameter set the threshold that each iteration has to surpass to continue fitting the model to the dataset. The $L1$ and $L2$ regularization weights that deal with the sparsity of data and with no sparse data respectively, are set to 1.

5.2. Experiment results

In this subsection, we present the results of the experiments conducted that aim to build a useful recommendation model to suggest the most feasible components to users of a component-based interface software application, thereby enhancing the user experience and improving their engagement with the interface. Due to the fact that the number of classes to predict was too high according to the number of instances of our dataset, we reduced the number of components to suggest from 33 to 8. Thus, the recommendation model will suggest only the 8 most used components to users in the ENIA application, which are: `BioReserves`, `Clock2`, `CuttleRoads`, `GeoParks`, `Heritage`, `Twitter`, `Weather` and `Wetlands`.

To build the model, we have selected the four classification algorithms previously discussed, which are *decision trees*, *decision jungle*, *artificial neural networks* and *logistic regression*. We have applied each of these algorithms to the whole dataset after applying the feature engineering techniques discussed in Section 4.2. We have also applied these algorithms to a subset of the dataset we have extracted using the *Mutual Information Score* and *Chi-Squared Statistic* feature selection methods described in Section 4.4.

Fig. 7 shows the results of the experiments by plotting the accuracy of the models created with each algorithm, with and

Table 7
Overall and average accuracy. FS: Feature Selection; OA: Overall Accuracy; AA: Average Accuracy.

Algorithm	FS	OA	AA
`Decision Forest`		0.605	0.901
`Decision Forest`	✓	0.646	0.912
`Decision Jungle`		0.399	0.850
`Decision Jungle`	✓	0.434	0.859
`Logistic Regression`		0.719	0.930
`Logistic Regression`	✓	0.741	0.935
`Neural Network`		0.788	0.947
`Neural Network`	✓	0.803	0.951

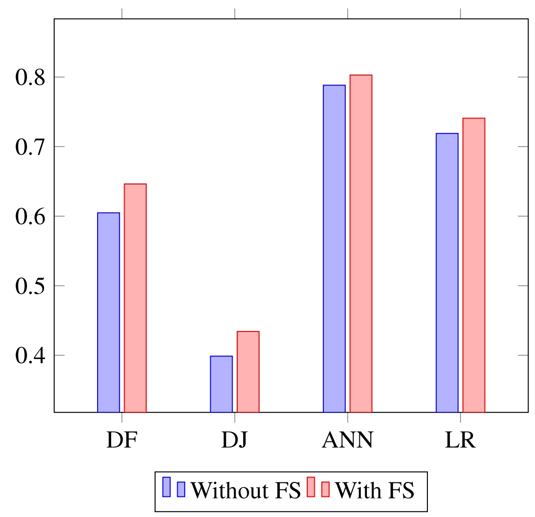


Fig. 7. Overall accuracy.

without reducing the original dataset applying features selection methods. The figure clearly indicates that the model built with artificial neural networks gets higher accuracy than the model built using the other algorithms, while the decision forest and decision jungle tree-like algorithms present the worst results, especially the decision jungle.

Fig. 7 also shows that the results are better when applying feature selection methods to the dataset than using the whole dataset. The exact numerical results can be seen in Table 7. We could foresee that a features subset selection of the original dataset could get better results, given the number of instances and the number of possible hypothesis. A high sparsity of data in high-dimensional spaces does not only computationally penalizes the creation of a model but also affects to its efficiency, decreasing the accuracy of the model.

Nevertheless, the sparsity of data itself, is not necessarily a problem if there are insights or knowledge to infer from data, as happens in this scenario. In ENIA, users' behavior clearly follows some patterns, as is borne out by the high accuracy of the models created. Inferring that patterns would be difficult in high-dimensional spaces with the lack of data that our dataset may have but, by reducing the number of features of the dataset with a subset of the most relevance features, we can avoid the problems derived from the curse of dimensionality in high-dimensional spaces. Thus, creating models using a subset of features selected by feature selection methods we can get greater accuracy. The models we have created obtain good results, reaching 80% accuracy in the case of the artificial neural network model. This happens for the models built using every algorithm, obtaining better results in all cases with a subset of the features of the original dataset.

The good results that every model offers, independently of the algorithm used to build it, is remarkable. Even the decision jungle algorithm applied to the whole dataset, which is the model that presents least accuracy, throws an accuracy of 0.398747390. If one of the 8 possible classes were to be chosen randomly, the

Table 8
The confusion matrix. Components: {BR: BioReserves; C2: Clock2; CR: CuttleRoads; GP: GeoParks; HT: Heritage; TW: Twitter; WT: Weather; WL: Wetlands} | ML Algorithms {DF: Decision Forest; DJ: Decision Jungle; ANN: Artificial Neural Network; LR: Logistic Regression}.

Without feature selection

	BR	C2	CR	GP	HT	TW	WT	WL	
BR	**0.72**	0.00	0.00	0.25	0.02	0.00	0.00	0.00	DF
	0.58	0.00	0.11	0.25	0.00	0.00	0.04	0.00	DJ
	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	ANN
	0.75	0.00	0.00	0.19	0.00	0.00	0.06	0.00	LR
C2	0.07	**0.52**	0.00	0.34	0.00	0.00	0.07	0.00	DF
	0.00	**0.22**	0.22	0.56	0.00	0.00	0.00	0.00	DJ
	0.12	**0.75**	0.00	0.00	0.00	0.12	0.00	0.00	ANN
	0.07	**0.47**	0.00	0.33	0.13	0.00	0.00	0.00	LR
CR	0.00	0.00	**0.67**	0.26	0.03	0.03	0.03	0.00	DF
	0.00	0.00	**0.67**	0.33	0.00	0.00	0.00	0.00	DJ
	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	ANN
	0.00	0.00	**0.42**	0.24	0.15	0.00	0.15	0.03	LR
GP	0.00	0.00	0.04	**0.90**	0.02	0.02	0.03	0.00	DF
	0.00	0.00	0.02	**0.97**	0.00	0.00	0.01	0.00	DJ
	0.00	0.04	0.00	**0.92**	0.04	0.00	0.00	0.00	ANN
	0.02	0.00	0.00	**0.97**	0.02	0.00	0.00	0.00	LR
HT	0.00	0.00	0.00	0.60	**0.39**	0.00	0.01	0.00	DF
	0.00	0.00	0.01	0.85	**0.11**	0.00	0.03	0.00	DJ
	0.00	0.00	0.00	0.09	**0.87**	0.00	0.04	0.00	ANN
	0.00	0.00	0.02	0.22	**0.72**	0.00	0.04	0.00	LR
TW	0.00	0.00	0.09	0.47	0.07	**0.36**	0.02	0.00	DF
	0.00	0.00	0.11	0.82	0.00	**0.08**	0.00	0.00	DJ
	0.00	0.05	0.00	0.10	0.15	**0.70**	0.00	0.00	ANN
	0.00	0.00	0.13	0.13	0.00	**0.71**	0.03	0.00	LR
WT	0.02	0.00	0.13	0.37	0.05	0.05	**0.40**	0.00	DF
	0.00	0.00	0.14	0.79	0.00	0.00	**0.07**	0.00	DJ
	0.00	0.00	0.07	0.10	0.13	0.07	**0.60**	0.03	ANN
	0.04	0.00	0.00	0.10	0.10	0.00	**0.76**	0.00	LR
WL	0.00	0.00	0.07	0.00	0.00	0.00	0.00	**0.93**	DF
	0.00	0.00	0.32	0.00	0.00	0.00	0.00	**0.68**	DJ
	0.00	0.00	0.18	0.00	0.18	0.00	0.00	**0.64**	ANN
	0.00	0.00	0.07	0.29	0.07	0.00	0.07	**0.50**	LR

(b) With feature selection

	BR	C2	CR	GP	HT	TW	WT	WL	
BR	**0.81**	0.00	0.00	0.19	0.00	0.00	0.00	0.00	DF
	0.63	0.00	0.07	0.30	0.00	0.00	0.00	0.00	DJ
	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	ANN
	0.75	0.00	0.00	0.13	0.00	0.00	0.13	0.00	LR
C2	0.00	**0.72**	0.00	0.28	0.00	0.00	0.00	0.00	DF
	0.00	**0.22**	0.25	0.50	0.00	0.00	0.00	0.03	DJ
	0.25	**0.75**	0.00	0.00	0.00	0.00	0.00	0.00	ANN
	0.07	**0.60**	0.00	0.27	0.07	0.00	0.00	0.00	LR
CR	0.00	0.00	**0.62**	0.36	0.00	0.00	0.03	0.00	DF
	0.00	0.00	**0.69**	0.27	0.00	0.00	0.00	0.04	DJ
	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	ANN
	0.00	0.00	**0.52**	0.18	0.06	0.03	0.15	0.06	LR
GP	0.00	0.00	0.00	**0.94**	0.02	0.01	0.03	0.00	DF
	0.00	0.00	0.03	**0.96**	0.01	0.00	0.00	0.00	DJ
	0.00	0.00	0.00	**0.96**	0.04	0.00	0.00	0.00	ANN
	0.02	0.00	0.00	**0.98**	0.00	0.00	0.00	0.00	LR
HT	0.00	0.00	0.00	0.55	**0.39**	0.01	0.05	0.00	DF
	0.00	0.00	0.02	0.74	**0.22**	0.00	0.01	0.01	DJ
	0.04	0.00	0.00	0.09	**0.87**	0.00	0.00	0.00	ANN
	0.00	0.00	0.04	0.28	**0.64**	0.00	0.04	0.00	LR
TW	0.00	0.00	0.07	0.51	0.00	**0.38**	0.04	0.00	DF
	0.00	0.00	0.14	0.80	0.00	**0.06**	0.00	0.00	DJ
	0.00	0.00	0.00	0.25	0.05	**0.70**	0.00	0.00	ANN
	0.00	0.00	0.08	0.08	0.00	**0.76**	0.08	0.00	LR

(*continued on next page*)

Table 8 (*continued*).

(b) With feature selection

	BR	C2	CR	GP	HT	TW	WT	WL	
WT	0.00	0.00	0.02	0.51	0.00	0.00	**0.48**	0.00	DF
	0.00	0.00	0.11	0.74	0.00	0.00	**0.14**	0.01	DJ
	0.00	0.00	0.07	0.17	0.07	0.07	**0.63**	0.00	ANN
	0.04	0.00	0.00	0.12	0.08	0.00	**0.76**	0.00	LR
WL	0.00	0.00	0.14	0.00	0.00	0.00	0.00	**0.86**	DF
	0.00	0.00	0.23	0.14	0.00	0.00	0.00	**0.64**	DJ
	0.00	0.00	0.18	0.00	0.18	0.00	0.00	**0.64**	ANN
	0.00	0.00	0.00	0.36	0.00	0.00	0.00	**0.64**	LR

probability of guessing would be 0.125. It means that the worst model built by applying machine learning techniques triplicates the possibilities of suggesting a suitable component to users when they need it, in the case that components were to be chosen randomly.

In our case study, there is a class imbalance, *i.e.*, the most frequent case GeoParks has six times the number of occurrences than the less frequent case Clock2, as previously shown in Table 5. The class imbalance is not extremely high but it needs to be considered. For this reason, the *Overall Accuracy* measure is not enough, we also need the *Average Accuracy* measure to really determine the performance of the models. The *Overall Accuracy* indicates the correctly predicted instances (number of correctly predicted items/total of items to predict). The *Average Accuracy* measures the ability to predict classes with few occurrences, difficult to predict (sum of the accuracy for each class predicted/number of classes). Table 7 presents both measures and we can see that given the performance of the *Average Accuracy* the models created using the decision forest and decision jungle algorithms have problems predicting the less frequent classes and the artificial neural networks and logistic regression models behave more consistently across all classes to predict.

Data can be analyzed in more detail by studying the *Confusion Matrix*. A *Confusion Matrix* is a table layout where each row represents the number of instances of each class and each column represents the class that has been predicted by the model. In this table, we can obtain a reliable performance of the model beyond the global accuracy. We have created a customized *Confusion Matrix* (Table 8) that has been divided into two parts. The upper part shows the results of the models created using the whole dataset and the lower part shows the results of the models created using a subset of features. Each cell shows 4 values, corresponding to each algorithm used.

The detailed values of the accuracy of each model predicting each class are shown in Table 8. We have also created a surface chart representing the Confusion Matrix, shown in Fig. 8, that helps to easily visualize the results thrown by the models created. As can be noted from this chart, the decision forest and decision jungle algorithms, regardless of using the whole dataset or a subset of it, show a high concentration of occurrences that are not correctly predicted at the bottom part of their chart. This part corresponds to the instances where the predicted class is GeoParks, the most common label. Additionally, the decision jungle algorithms have serious problems predicting the Weather, Heritage and Clock2 components. It looks like these algorithms tend to predict the most common label GeoParks. Conversely, the artificial neural networks and logistic regression algorithms present better results, specifically the model created using artificial neural networks is particularly suitable for creating the recommendation system that we aim to create in this research. The surface chart shows that this model is very consistent accurately predicting all classes as well as being highly accurate predicting the most uncommon classes. These attributes ensure that the model is reliable and we can

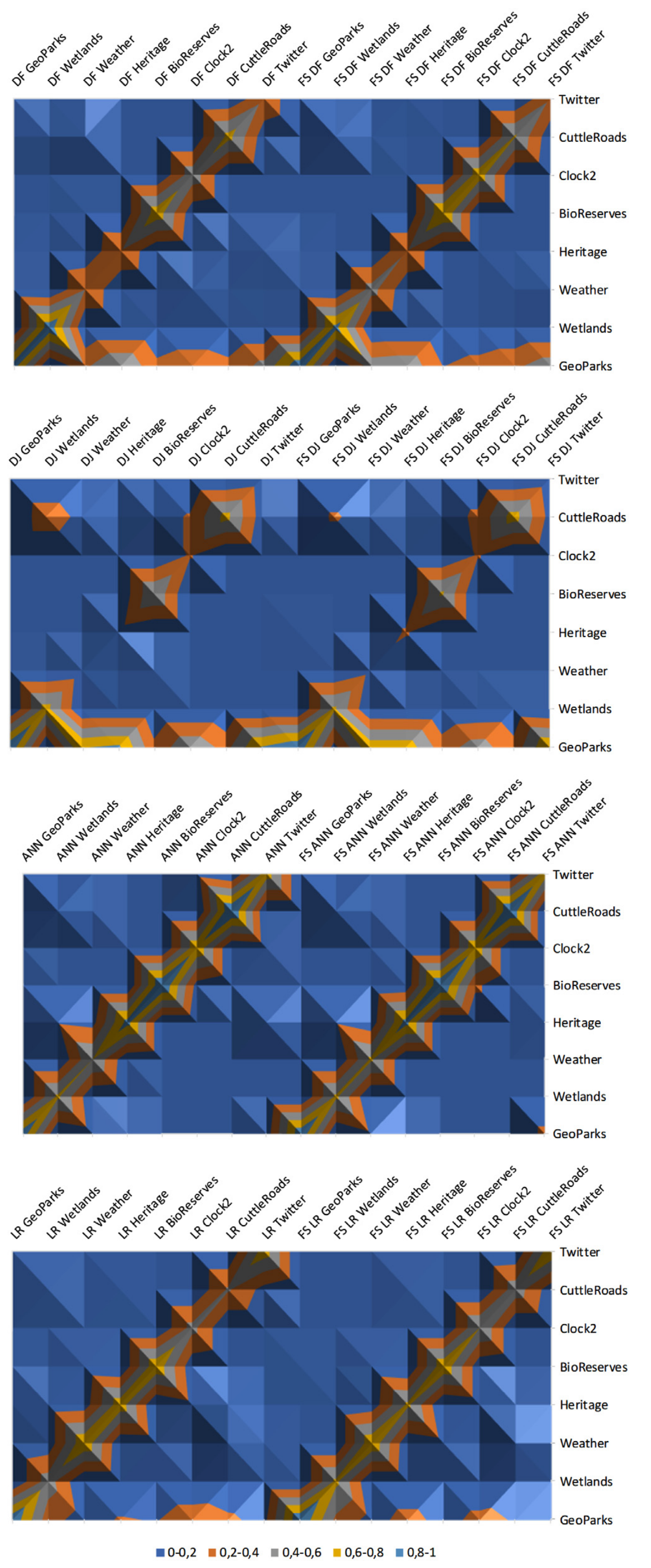


Fig. 8. Surface Chart representing the Confusion Matrix. Each part shows the accuracy of an algorithm model with and without feature selection. Colors show the accuracy. FS: Feature Selection, DF: Decision Forest, DJ: Decision Jungle, ANN: Artificial Neural Network, LR: Linear Regression.

conclude that the model created using the artificial neural networks, trained with a subset that contains the most significant features extracted using the *Mutual Information Score* and *Chi-Squared*

Statistic feature selection methods, is the best model created to suggest the most suitable component to users in component-based applications.

It is remarkable the good behavior of the artificial neural network models in this problem. It is probably because the data is not distributed according to a set of restrictions or constraints. This fact usually happens when it is intended to model human behaviors. One of the advantages of artificial neural networks is their ability in generalizing non-linear complex relationships thanks to a reasonable number of hidden layers in their structure that allow them to infer these complex relationships in data collected from human behaviors.

Following the same arguments, the bad behavior of tree-based algorithms is due to its inability to set reasonable boundaries on which the classes can be logically divided. The logistic regression algorithm performance is between the neural network and the tree-based algorithms probably because a sufficiently large number of instances can be distributed consistently in a logistic distribution. In any case, given the nature of the human behavior that needs to be modeled, the "black box" nature inside neural networks algorithms and its hidden layers allow the creation of the most accurate models for this problem.

6. Deployment

Once the best model is selected, it has to be deployed in the component-based application. The real output of the recommendation system is the list of possible components to suggest, along with the grade of certainty of each component recommended, expressed as a percentage.

The deployment of a recommender system in real component-based applications requires to analyze several aspects of the recommendation to evaluate its suitability as well as to study the potential conflicts it may provoke. In this paper, we dive into these issues through the recommender system deployment in the ENIA interface.

Once the recommender model is built, it has to be connected to the ENIA interface to exploit the knowledge inferred by the predictive model and to facilitate the suggestion of components to end users. Fig. 9 graphically illustrates the recommender system deployment process, highlighted under the gray area. The description of this process is summarized below and explained in detail in the next subsections. The main parts are:

1. **Web Service**. It consists of a web service that consumes the recommendation model and generates recommendations based on the users' interaction with the application.
2. **Decision System**. It consists of deciding whether a recommendation should be offered to end users or not.

The recommender system projected has two inputs and one output as it is described below:

– *(Input 1) Real-Time Interaction Data*. User interaction data acquired in real-time from the component-based application user interface.

– *(Input 2) Trained Model*. The most accurate predictive model is integrated as the core of a web service.

– *(Output) Recommendation*. Suggestions produced by the recommender system ready to be offered to end users of the ENIA interface.

6.1. Web service

The recommender system is hosted in a RESTful [52] web service. When the web service is called, the client sends alongside the request a JSON with data about the interaction performed by a user (input 1). A JSON Schema is available to the client, a powerful tool for validating the structure of JSON data. It contains the structure and datatype of the input and output of the web service so clients can make use of this tool to implement automated testing as well as validating the interaction data before being submitted.

The core of the web service consists of the encapsulation of the most accurate trained model created in the previous step. This model receives the set of features that better describe the user workspace immediately after an interaction is performed and, based on that features, it generates a set of suggestions. These suggestions contain the most suitable components that can be suggested to the user at that precise moment along with their scored probabilities.

The effect of the passage of time and changes in users, applications and services can make that the recommendations of the predictive model cannot always satisfy the user's requirements. In dynamic and evolving computer systems, it is intended that the interface adaptation changes intelligently with the needs of users at all times. For that purpose, in ENIA, a daemon is periodically requested to re-train the predictive models using the last data collected from the users' interaction to ensure the quality of the recommendations. Therefore, the model can accurate forecast the most suitable components in a scenario where new users are continuously registered and deleted (as well as user categories), new components are continuously registered and deleted by developers and there exists a continuous evolution in the type of devices (laptops, computers, smartphones tablets...) and forms of interaction (mouse, keyboard, gesture, voice...) that access to the component-based application user interface.

In ENIA, as commented before, the model has been built to suggest eight different components according to the users' needs. The output of the web service provides the scored probabilities for each component that is available to suggest as well as the final recommendation in the `label` field, *i.e.*, the component with the higher score. The sum of all the classes score is 1. To better understand the output of the web service, an instance of the output produced in JSON format, containing the name of each possible component to suggest and its scored probability, can be seen in Listing 1.

Listing 1: Instance of a model output in JSON

```
{ "output": {
          "idInteraction":"594231",
          "BiosphereReservesScore":"0.00113",
          "Clock2Score":"0.000067",
          "CuttleRoadsScore":"0.000278",
          "GeoParksScore":"0.001882",
          "HeritageScore":"0.000245",
          "TwitterScore":"0.000829",
          "WeatherScore":"0.992354",
          "WetlandsScore":"0.000027",
          "Label":"Weather"
} }
```

Fig. 10 shows the possible output of the web service that contains the recommender model according to a concrete workspace. The upper part of the figure shows the graphical user interface after the last interaction, from which the interaction data has been extracted. The bottom part shows the scored probabilities that the recommender model generates for each of the possible class to suggest, *i.e.*, the grade of suitability of each available component to suggest, according to the concrete situation that shows the upper part of the figure.

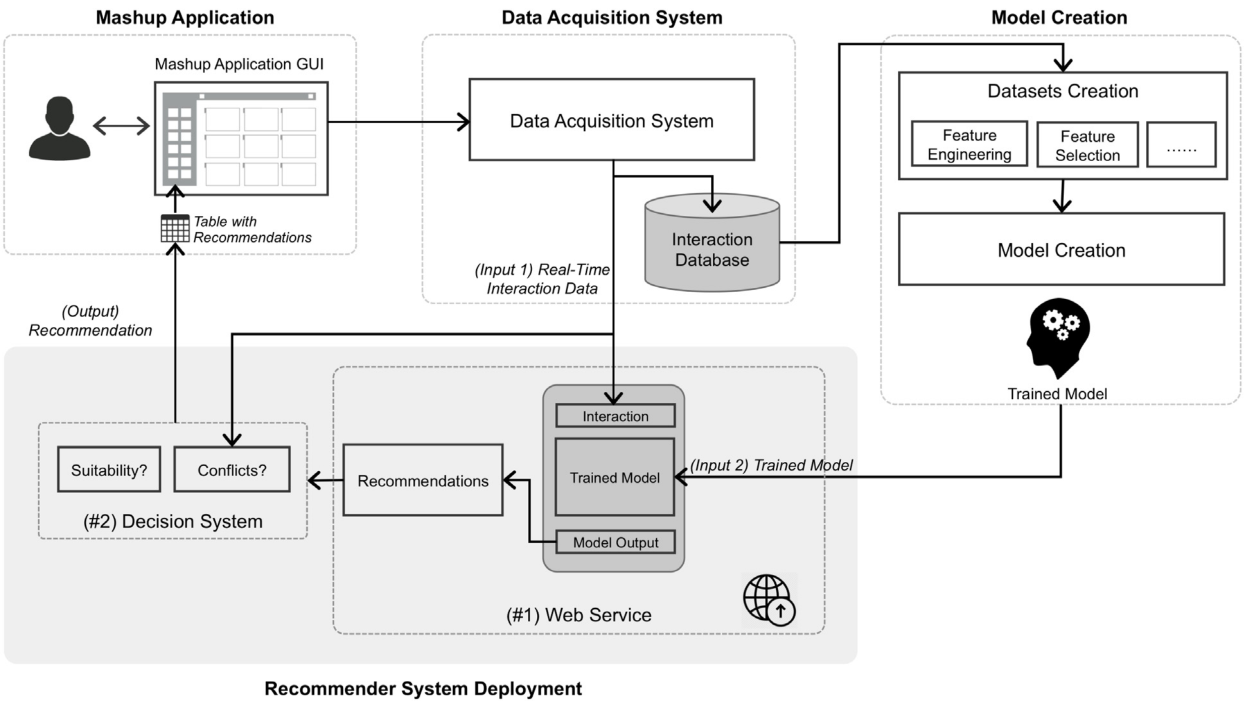
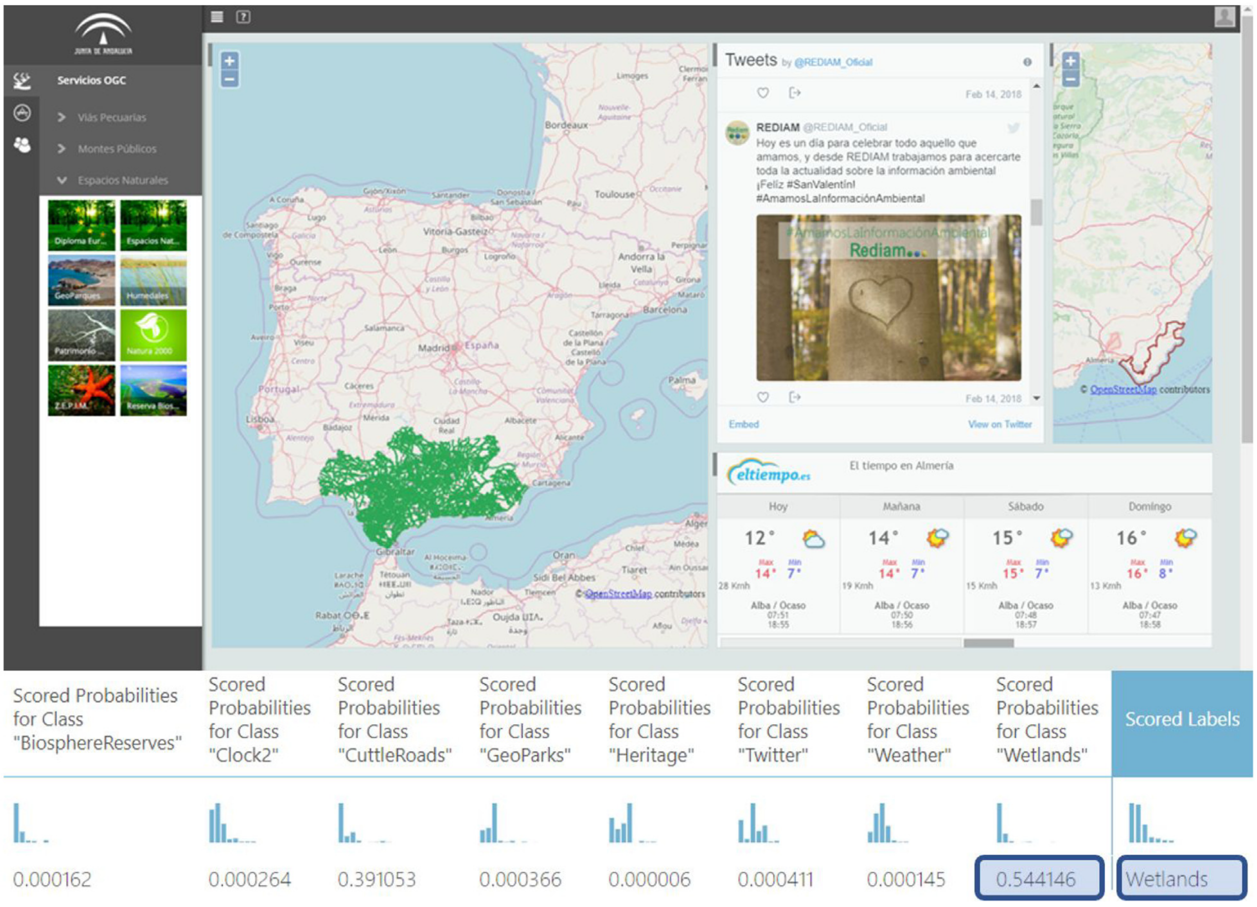


Fig. 9. Recommendation model output for a concrete scenario in ENIA.



REDIAM @REDIAM_Oficial
Hoy es un día para celebrar todo aquello que amamos, y desde REDIAM trabajamos para acercarte toda la actualidad sobre la información ambiental ¡Feliz #SanValentín! #AmamosLaInformaciónAmbiental

Scored Probabilities for Class "BiosphereReserves"	Scored Probabilities for Class "Clock2"	Scored Probabilities for Class "CuttleRoads"	Scored Probabilities for Class "GeoParks"	Scored Probabilities for Class "Heritage"	Scored Probabilities for Class "Twitter"	Scored Probabilities for Class "Weather"	Scored Probabilities for Class "Wetlands"	Scored Labels
0.000162	0.000264	0.391053	0.000366	0.000006	0.000411	0.000145	0.544146	Wetlands

Fig. 10. Recommendation model output for a concrete scenario in ENIA.

6.2. Decision system

The output of the recommender system model cannot be directly applied as a suggestion in component-based applications. Certain issues, such as the grade of suitability of the component suggested or the potential conflicts that it may arise in the interface layout, have to be considered.

In ENIA, the recommender system has eight possible components to suggest. As mentioned, the sum of all the scored probabilities for each possible class is 1. When a prediction has a scored probability of 0.95 the suitability of that component is very high and the recommender system is confident in its prediction. On the contrary, when the component suggested has a scored probability of 0.2, we may face an uncertain prediction that may not be relevant or satisfactory to the end user.

In order to ensure that predictions truly provide great value to users, a module to decide whether to make use of the prediction or not need to be added. This module can be programmed to decide whether a suggestion is suitable based on a variety of criteria. Based on these criteria, more than one class can be suggested at the same time.

In ENIA, our case study, before making a real-time suggestion, it is checked that the scored probability of the component suggested by the recommendation model is higher than 0.5. Otherwise, the suggestion will not be offered to users. We have set the threshold score to 0.5 but it may differ for different component-based interfaces depending on the number of classes they are able to predict and the nature of the components they suggest. Besides, as it is impossible that an output of the recommender model has more than a class with a scored probability higher than 0.5, we limit the number of recommendations at a maximum of one. Other applications might want to offer more than a recommendation for a concrete situation, they can do that defining other rules in this module.

Another common problem is that, on some occasions, the component that the recommender model suggests is already in the interface workspace. In ENIA, it would not be an issue, since ENIA allows that several instances of a component would coexist in the same interface at the same time. However, it can lead to a problem in others component-based applications. For example, in a temperature control system, there may exist a component that sets the temperature to a specific value and it would not be advisable that several instances of that component try to set the temperature to a different value at the same time. For such cases, a conflicts management module is needed. This module deals with the architecture constraints of the component-based application as well as other rules that may be defined by the developers that exploit the application.

The conflicts management module deals with problems like this and any other specific issues that different component-based applications may have such as incompatibilities in the simultaneous use of some components, dependencies that components may have (a component may not be added to the workspace without adding others), selection between components from same or different providers with more/less functionality, or discrepancies on how the components are positioned in the workspace of the component-based interface.

The rules which dictate the behavior of the interface to solve conflicts that may arise are defined at design time and stored in a repository. These rules are usually defined by the component-based application administrators. The providers of the components can also define their own rules. In [32], it is in-depth described how these rules make adaptations of component-based architectures using models transformation in real-time, creating smart user interfaces.

Once the suggestion of the recommender model is final, *i.e.*, the score of the component suggested is good enough and it does not raise any conflict in the interface layout (or this conflict can be solved), the suggestion is ready to be offered to users. For that purpose, a table is created in the component-based application database that contains the available suggestions to end users. New records added to this table means that there are suggestions ready to be made. The table contains two fields: the `recommendation` field contains the name of the component suggested and the `idInteraction` field contains a value that unequivocally identifies the interaction that triggered the suggestions and thus, they can be offered to the proper users.

Finally, the component-based application has to graphically indicate to the end users that a component may be useful for them in the user interface. Each interface, depending on its design, may show it (the suggestions) in different ways. Fig. 11 shows how recommendations are shown to ENIA users. There are unlimited possibilities of how suggestions can be offered to users (if they are not automatically applied). The designers of each application can show the results of the recommender model as they see fit.

7. Conclusions and future work

In this article, we address the problem of creating a recommender system that is able to suggest to users of component-based interfaces, which are the most suitable components for them to use at a specific time. By forecasting the component most closely aligned to each situation, we aim to improve the user experience in the software application and thus, optimize the possibilities of successfully achieving a good position in the increasingly competitive software development market.

We have created several models using a dataset that contains the interactions performed by users in component based applications after applying feature engineering techniques and feature selection methods. After this, we have evaluated and compared the models created in terms of overall accuracy and average accuracy, as well as analyzing the confusion matrix that offers the specifying accuracy including the deviation that the model may have predicting a class compared with the labeled class. As a conclusion, all the recommender systems get good results achieving an accuracy of up to 40%, with the neural network models being the one that gets better performance, yielding an accuracy of up to 80%. It is remarkable that feature selection methods favorably affect to the results, increasing the accuracy by an average of 3% in all cases.

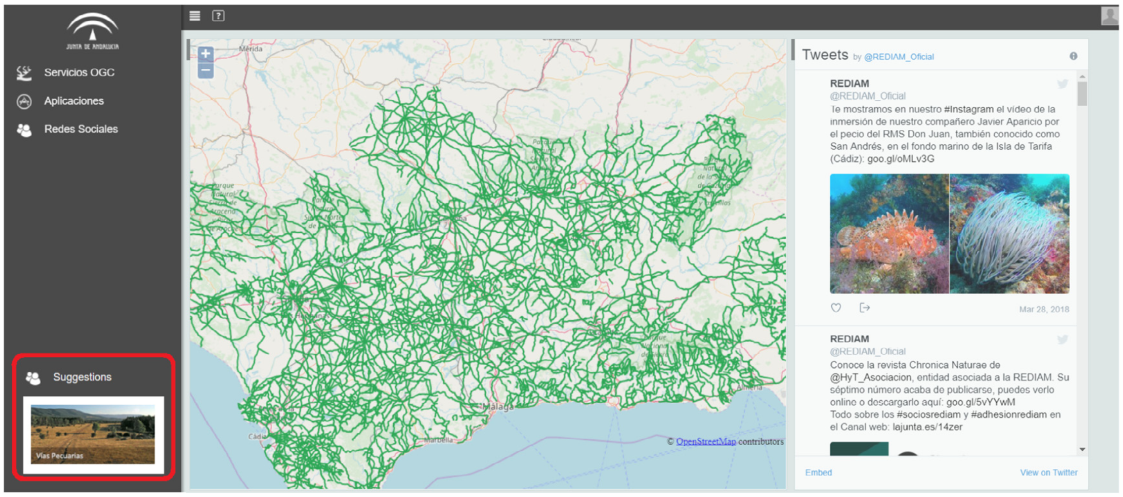
In future works, it would be interesting to improve the recommendation of components in component-based user interfaces by following these practices:

(a) Instead of using just the recommendation model with highest accuracy, the creation of a network that agglutinates all the models and votes the most popular output of each of them, considering the normalized accuracy of each, could improve the overall performance.

(b) Since there are components more frequently used than others, with a huge difference, adding weights that help to better forecast the less common classes would improve the average accuracy of the recommendation model.

(c) Given the rapid evolution of users and components in component-based user interfaces and in the software development market, a nice improvement would be the definition of a coherent strategy to continuously update the recommendation model so it can be adapted to the changes in the software application environment.

(d) Along with the adding components, the removal and replacement of components is a key factor in achieving smart user interfaces that suggest to users the most suited interface in every situation. Intelligence methods may even suggest how the components should be placed in the interface layout to



REDIAM
@REDIAM_Oficial
Te mostramos en nuestro #Instagram el vídeo de la inmersión de nuestro compañero Javier Aparicio por el pecio del RMS Don Juan, también conocido como San Andrés, en el fondo marino de la Isla de Tarifa (Cádiz): goo.gl/oMLv3G
Mar 28, 2018

REDIAM
@REDIAM_Oficial
Conoce la revista Chronica Naturae de @HyT_Asociacion, entidad asociada a la REDIAM. Su séptimo número acaba de publicarse, puedes verlo online o descargarlo aquí: goo.gl/5vYYwM
Todo sobre los #sociosrediam y #adhesionrediam en el Canal web: lajunta.es/14zer

Fig. 11. Example of a component suggestion in ENIA.

an optimal use of the components that are being used in a specific moment from a concrete user. In future works we will focus on the creation of new models to (i) suggest users the replacement of a component for another; (ii) suggest users the removal of a component and; (iii) suggest users how they can redistribute the components they are using to make an optimal usage of the interface. Several other models can be created to cover exclusive aspects of each component-based interface such as combining components or resizing them. The combination of the suggestions created by all the models contributes to the creation of smart component-based user interfaces.

Through the implementation of these improvements and the continuous optimization of the datasets, adding more instances and significant features for each purpose, the recommendation system can be properly optimized over time.

Acknowledgments

This work has been funded by the EU ERDF, Spain and the Spanish Government under AEI Projects TIN2013-41576-R and TIN2017-83964-R. A.J. Fernández-García has been funded by a FPI, United States Grant BES-2014-067974. J.Z. Wang was funded in part by the US National Science Foundation under Grant No. 1027854.

References

[1] Google, Google Now. Personal Assistant, 2018. https://www.google.com/intl/es/landing/now/.
[2] Netflix, Netflix. A component-based video-on-demand streaming platform, 2018. https://www.netflix.com/.
[3] E.P. Fernandez-Manzano, E. Neira, J. Clares-Gavilan, Data management in audiovisual business: Netflix as a case study, Profesional de la Información 25 (4) (2016) 568–576.
[4] A. Liu, JavaScript and the Netflix user interface, Commun. ACM 57 (11) (2014) 53–59, http://doi.acm.org/10.1145/2669482.
[5] Flipboard, Flipboard. News and Social Media Aggregation Platfrom, 2018. https://flipboard.com/.
[6] FitBit, FitBit. Fitness products family to monitor activity, exercise, food, weight and sleep, 2018. https://www.fitbit.com.
[7] I. Portugal, P. Alencar, D. Cowan, The use of machine learning algorithms in recommender systems: A systematic review, Expert Syst. Appl. 97 (2018) 205–227, http://www.sciencedirect.com/science/article/pii/S0957417417308333.
[8] ENIA. Environmental Information Agent Project. http://acg.ual.es/projects/enia/ui/. Online (Last accessed 30 September 2018).
[9] The Andalusian Environmental Information Network (REDIAM). http://www.juntadeandalucia.es/medioambiente/site/rediam/. Online (Last accessed 30 September 2018).
[10] K. Xu, X. Zheng, Y. Cai, H. Min, Z. Gao, B. Zhu, H. Xie, T. Wong, Improving user recommendation by extracting social topics and interest topics of users in uni-directional social networks, Knowl.-Based Syst. 140 (Supplement C) (2018) 120–133, http://www.sciencedirect.com/science/article/pii/S0950705117305002.
[11] D. Bokde, S. Girase, D. Mukhopadhyay, Matrix factorization model in collaborative filtering algorithms: A survey, Procedia Comput. Sci. 49 (Supplement C) (2015) 136–146, http://www.sciencedirect.com/science/article/pii/S1877050915007462.
[12] M. Mao, J. Lu, G. Zhang, J. Zhang, Multirelational social recommendations via multigraph ranking, IEEE Trans. Syst. Cybern. 47 (12) (2017) 4049–4061.
[13] Mingsong Mao, Jie Lu, Jialin Han, Guangquan Zhang, Multiobjective e-commerce recommendations based on hypergraph ranking, Inform. Sci. 471 (2019) 269–287, http://www.sciencedirect.com/science/article/pii/S0020025518305437.
[14] S. Yang, M. Korayem, K. AlJadda, T. Grainger, S. Natarajan, Combining content-based and collaborative filtering for job recommendation system: A cost-sensitive Statistical Relational Learning approach, Knowl.-Based Syst. 136 (Supplement C) (2017) 37–45, http://www.sciencedirect.com/science/article/pii/S095070511730374X.
[15] L. Getoor, B. Taskar, Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning), The MIT Press, 2007.
[16] J. Quinlan, Induction of decision trees, Mach. Learn. 1 (1) (1986) 81–106, http://dx.doi.org/10.1023/A:1022643204877.
[17] P. McCullagh, J.A. Nelder, Generalized linear models, in: Chapman and Hall/CRC Monographs on Statistics and Applied Probability Series, second ed., Chapman & Hall, 1989, http://books.google.com/books?id=h9kFH2_FfBkC.
[18] R. Rojas, Neural Networks: A Systematic Introduction, Springer-Verlag New York, Inc., 1996.
[19] P.A. Castillo, A.M. Mora, H. Faris, J.J. Merelo, P. García-Sánchez, A.J. Fernández-Ares, P. De las Cuevas, M.I. Garcáa-Arenas, Applying computational intelligence methods for predicting the sales of newly published books in a real editorial business management environment, Knowl.-Based Syst. 115 (Supplement C) (2017) 133–151, http://www.sciencedirect.com/science/article/pii/S0950705116304026.
[20] W. Lee, C. Chen, J. Huang, J. Liang, A smartphone-based activity-aware system for music streaming recommendation, Knowl.-Based Syst. 131 (Supplement C) (2017) 70–82, http://www.sciencedirect.com/science/article/pii/S0950705117302757.
[21] Y. Zhou, H. Mao, Z. Yi, Cell mitosis detection using deep neural networks, Knowl.-Based Syst. 137 (2017) 19–28, http://www.sciencedirect.com/science/article/pii/S0950705117303738.
[22] Y. Cohen, D. Hendler, A. Rubin, Detection of malicious webmail attachments based on propagation patterns, Knowl.-Based Syst. 141 (2018) 67–79, http://www.sciencedirect.com/science/article/pii/S0950705117305336.
[23] I.M. Galván, J.M. Valls, A. Cervantes, R. Aler, Multi-objective evolutionary optimization of prediction intervals for solar energy forecasting with neural networks, Inform. Sci. 418–419 (2017) 363–382, http://www.sciencedirect.com/science/article/pii/S0020025517308861.
[24] S. Wang, C. Li, K. Zhao, H. Chen, Learning to context-aware recommend with hierarchical factorization machines, Inform. Sci. 409–410 (2017) 121–138, http://www.sciencedirect.com/science/article/pii/S0020025516309896.

[25] N.M. Villegas, C. Sánchez, J. Dıiaz-Cely, G. Tamura, Characterizing context-aware recommender systems: A systematic literature review, Knowl.-Based Syst. 140 (2018) 173–200, http://www.sciencedirect.com/science/article/pii/S0950705117305075.

[26] C. Sánchez, N.M. Villegas, J. Díaz-Cely, Exploiting context information to improve the precision of recommendation systems in retailing, Commun. Comput. Inf. Sci. 735 (2017) 72–86.

[27] S. Gómez, P. Zervas, D.G. Sampson, R. Fabregat, Context-aware adaptive and personalized mobile learning delivery supported by UoLmP, J. King Saud Univ. - Comput. Inf. Sci. 26 (2014) 47–61, http://www.sciencedirect.com/science/article/pii/S1319157813000372.

[28] Y. Zhang, S. Wang, P. Phillips, G. Ji, Binary PSO with mutation operator for feature selection using decision tree applied to spam detection, Knowl.-Based Syst. 64 (2014) 22–31, http://www.sciencedirect.com/science/article/pii/S095070511400104X.

[29] P. Hajek, R. Henriques, Mining corporate annual reports for intelligent detection of financial statement fraud–comparative study of machine learning methods, Knowl.-Based Syst. 128 (2017) 139–152, http://www.sciencedirect.com/science/article/pii/S0950705117302022.

[30] Z. Wei, Y. Wang, S. He, J. Bao, A novel intelligent method for bearing fault diagnosis based on affinity propagation clustering and adaptive feature selection, Knowledge-Based Syst. 116 (2017) 1–12, http://www.sciencedirect.com/science/article/pii/S0950705116304014.

[31] N. Ghaibi, O. Dâassi, L. Ayed, A tool support for the adaptation of user interfaces based on a business rules management system, in: 29th Conf. on Computer-Human Interaction, ACM, 2017, pp. 162–169, http://doi.acm.org/10.1145/3152771.3152789.

[32] J. Criado, D. Rodríguez-Gracia, L. Iribarne, N. Padilla, Toward the adaptation of component-based architectures by model transformation: Behind smart user interfaces, Softw. - Pract. Exp. 45 (12) (2015) 1677–1718.

[33] C. Rodríguez, D. Florian, F. Casati, Mining and quality assessment of mashup model patterns with the crowd: A feasibility study, ACM Trans. Internet Technol. 16 (3) (2016) 17:1–17:27, http://doi.acm.org/10.1145/2903138.

[34] G.v. Bochmann, High-level design for user and component interfaces, Knowl.-Based Syst. 17 (7) (2004) 303–310, http://www.sciencedirect.com/science/article/pii/S0950705104000486, (Special issue on Legacy systems and software change).

[35] M.G. Armentano, A.A. Amandi, Personalized detection of user intentions, Knowl.-Based Syst. 24 (8) (2011) 1169–1180, http://www.sciencedirect.com/science/article/pii/S0950705111000864.

[36] J. Vallecillos, J. Criado, A.J. Fernández-García, N. Padilla, L. Iribarne, A web services infrastructure for the management of mashup interfaces, in: Service-Oriented Computing – ICSOC 2015 Workshops, Springer, 2016, pp. 64–75.

[37] J. Vallecillos, J. Criado, N. Padilla, L. Iribarne, A cloud service for COTS component-based architectures, Comput. Stand. Interfaces 48 (2016) 198–216.

[38] A.J. Fernández-García, L. Iribarne, A. Corral, J. Criado, J.Z. Wang, A flexible data acquisition system for storing the interactions on mashup user interfaces, Comput. Stand. Interfaces 59 (2018) 10–34, http://dx.doi.org/10.1016/j.csi.2018.02.002.

[39] Oracle Corporation, MySQL Database. https://www.mysql.com/. Online (Last accessed 30 September 2018).

[40] M. Kantardzic, Data Mining: Concepts, Models, Methods and Algorithms, John Wiley & Sons, Inc., 2002.

[41] A.J. Fernández-García, L. Iribarne, A. Corral, J. Criado, J.Z. Wang, Optimally storing the user interaction in mashup interfaces within a relational database, in: Current Trends in Web Engineering, Springer Int. Pub., Cham, 2016, pp. 188–195.

[42] A.J. Fernandez-Garcia, L. Iribarne, A. Corral, J.Z. Wang, Evolving mashup interfaces using a distributed machine learning and model transformation methodology, in: LNCS 9416, Springer, 2015, pp. 401–410.

[43] K. Ramasubramanian, A. Singh, Feature engineering, in: Mach. Learn. Using R, Apress, Berkeley, CA, 2017, pp. 181–217.

[44] Open Weather Map. https://openweathermap.org/. Online (Last accessed 30 September 2018).

[45] S. Garcia, J. Luengo, J.A. Saez, V. Lopez, F. Herrera, A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning, IEEE Trans. Knowl. Data Eng. 25 (4) (2013) 734–750, http://dx.doi.org/10.1109/TKDE.2012.35.

[46] S. Alelyani, J. Tang, H. Liu, Feature Selection for Clustering: A Review, Chapman&Hall, 2013.

[47] T. Hastie, R. Tibshirani, J. Friedman, The elements of statistical learning: Data mining, inference, and prediction, Internat. Statist. Rev. 77 (3) (2009) 482–482.

[48] Microsoft Corporation, Microsoft Azure Machine Learning Studios. https://studio.azureml.net. Online (Last accessed 30 September 2018).

[49] G. Chandrashekar, F. Sahin, A survey on feature selection methods, Comput. Electr. Eng. 40 (1) (2014) 16 – 28, http://www.sciencedirect.com/science/article/pii/S0045790613003066.

[50] N. Mantel, Chi-square tests with one degree of freedom; extensions of the mantel-haenszel procedure, J. Amer. Statist. Assoc. 58 (303) (1963) 690–700.

[51] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140, https://doi.org/10.1023/A:1018054314350.

[52] C. Pautasso, E. Wilde, R. Alarcon, REST: Advanced Research Topics and Practical Applications, Springer New York, 2014.