

*People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Mohamed Khider University - Biskra*



*Faculty of Exact Sciences and Sciences of Nature and Life
Computer Science department*

Option: Artificial Intelligence

by

THAMEUR Halima

TITLE

Selecting SOA design patterns using machine learning techniques

*Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master in computer science*

Defended the 06/07/2019, in front of the jury composed from:

HATTAB Dalila

President

MOUSSAOUI Manel

Member

Dr ZENADJI Tarek

Supervisor

year 2018-2019

Abstract

with the continuously increasing amount of textual informations, there is a pressing need to structure them. Text classification (TC) is a technique which classifies textual information into a predefined set of categories where documents can be automatically classified based on their contents. Automatic classification of data is one of the main applications of machine learning algorithms.

This master's thesis explores a way in which text classification is used to categorize user (developer) problems using machine learning algorithms so we can help him to find the appropriate design pattern known, that finding and selecting a suitable SOA design pattern has always been a challenging task especially for young developer during the designing phase because usually, a designer has to consult and search extensively to find a suitable pattern.

Among many patterns, few are considered to be relevant to solve a problem We can define a 'suitable SOA pattern' as a solution, which is aligned with the developer's problem and has a good effect.

With the help of already manually classified data that's the set of SOA Patterns a model can be learned using learning techniques like (naïve-Bayesian, k-nearest-neighbor, svm, decision-Tree). So that we can find the class of design patterns that our problem belongs to. then we use the patterns of this class as dataset to find the most appropriate one or in other words the closest design pattern to the user problem using cosine similarity.

Keywords: *classification; cosine similarity; SOA Patterns.*

Résumé

Avec le nombre croissant d'informations textuelles, il est urgent de les structurer. La classification de texte (TC) est une technique qui classe les informations textuelles dans un ensemble prédéfini de catégories dans lesquelles les documents peuvent être automatiquement classés en fonction de leur contenu. La classification automatique des données est l'une des principales applications des algorithmes d'apprentissage automatique.

Cette thèse de master explore une manière d'utiliser la classification de texte pour classer les problèmes d'utilisateurs (développeurs) à l'aide d'algorithmes d'apprentissage automatique afin de l'aider à trouver le modèle de conception approprié, sachant que trouver et choisir un modèle de conception soa approprié a toujours été un défi tâche particulièrement réservée aux jeunes développeurs au cours de la phase de conception car généralement, un concepteur doit consulter et rechercher de manière approfondie pour trouver un modèle approprié.

Parmi de nombreux modèles, peu sont considérés comme pertinents pour résoudre un problème. Nous pouvons définir un « modèle Soa approprié » comme une solution, alignée sur le problème du développeur et ayant un impact positif.

À l'aide de données déjà classées manuellement, qui constituent l'ensemble des modèles SOA, un modèle peut être appris à l'aide de techniques d'apprentissage telles que (naïve-bayésien, k-voisin le plus proche, svm, arbre de décision). Pour que nous puissions trouver la classe de modèles de conception à laquelle notre problème appartient. Nous utilisons ensuite les modèles de cette classe comme jeu de données pour trouver le modèle le plus approprié ou, en d'autres termes, le modèle de conception le plus proche du problème de l'utilisateur, à l'aide de la similarité en cosinus.

Mots-clés : classification ; similitude cosinus ; Modèles SOA.

ملخص

مع الزيادة المستمرة في المعلومات النصية ، هناك حاجة ملحة لهيكلية هذه المعلومات. تصنيف النص (TC) هو الأسلوب الذي يصنف المعلومات النصية إلى مجموعة محددة مسبقًا من الفئات حيث يمكن تصنيف المستندات تلقائيًا بناءً على محتوياتها. يعد التصنيف التلقائي للبيانات أحد التطبيقات الرئيسية لخوارزميات التعلم الآلي.

تستكشف رسالة الماجستير هذه الطريقة التي يتم بها استخدام تصنيف النص لتصنيف مشاكل المستخدم (المطور) باستخدام خوارزميات التعلم الآلي حتى يتمكن من مساعدته في العثور على نمط تصميم مناسب مع المعرفة ، أن العثور على واختيار نمط تصميم مناسب كان دائمًا يمثل تحديًا خاصة للمطور الشاب أثناء مرحلة التصميم لأنه عادةً ما يتوجب على المصمم التشاور والبحث على نطاق واسع للعثور على نمط مناسب.

من بين العديد من الأنماط ، يعتبر القليل منها ذا صلة بحل مشكلة ما. يمكننا تعريف "نمط Soa المناسب" كحل ، والذي يتوافق مع مشكلة المطور وله تأثير جيد.

بمساعدة البيانات المصنفة يدويًا والتي هي مجموعة من أنماط SOA ، يمكن تعلم نموذج باستخدام أدوات تعليمية مثل (svm ، naïve-Bayésien ، knn ، شجرة القرار). حتى يتمكن من العثور على فئة أنماط التصميم التي تنتمي إليها مشكلتنا. ثم نستخدم أنماط هذه الفئة كمجموعة بيانات لإيجاد أنسب واحد أو بعبارة أخرى أقرب نمط تصميم لمشكلة المستخدم باستخدام تشابه جيب التمام.

الكلمات الرئيسية: تصنيف؛ جيب التماثل التشابه؛ نماذج الخدمية.

Dedication

This work is dedicated to my beloved parents.

Acknowledgements

Praise to Allah, the Compassionate, the Merciful, Peace and blessing on the Messenger of Allah Muhamed the prophet (Peace Be Upon Him).

A research is something that cannot be materialized without the co-operation of many people involved in making it a reality. I wish to express our heartfelt gratitude to all those who have helped me in making this research work a success.

I wish to express our deep sense of gratitude to my Guide, Dr. TAREK ZERNADJI, Computer Department, for his able guidance and useful suggestions, which helped me in completing the research work, in time. Without his co-operation, it would have been extremely difficult for me to complete the research work.

*Finally, yet importantly, I would like to express my heartfelt gratitude and respect to my Almighty beloved parents **THAMEUR Aide** and **DJAHRA Messouda** for their blessings, my brothers **Mostapha, Boubaker Essiddik, Ayoub**, my sister **Karima**, my friends **Ahlem, Khaira, Abir** for their help and wishes for the successful completion of this thesis work.*

Table of Contents

Abstract	i
Dedication	iv
Acknowledgements	v
Table of Contents	vi
List of Tables	viii
List of Figures	ix
List of Acronyms	xi
<i>Chapter 1: Text-classification</i>	1
1.1. Definition	1
1.2. Text Classification Process	1
1.2.1. Document Collection.....	2
1.2.2. Document Pre-Processing &Representation	2
1.2.2.2. Removing unuseful stop words	2
1.2.3. Feature Generation/Text Transformation	3
1.2.4. Feature Selection	3
1.2.5. Text Classification.....	4
1.2.6. Evaluation of Text classifier.....	4
1.3. Feature construction.....	4
1.3.1. Feature identification.....	5
1.3.2. Feature weighting.....	6
1.4. Constructing a Vector Space Model	7
1.4.1. Text representation models.....	7
1.4.2. The whole processes from scratch.....	9
1.5. Machine learning algorithms	10
1.5.1. Types of machine learning	10
1.5.2. Machine learning algorithms	11
1.6. Similarity Mathematical Metrics	25
1.6.1. Types of similarity measures.....	25
1.7. Conclusion	29
<i>Chapter 2: Service Oriented Architecture</i>	30
2.1. Definition	30
2.1.1. SOA is built with services	30
2.1.2. Service orientation.....	31
2.2. Service-orientation design paradigm	31
2.2.1. Principles of Service-Orientation	31
2.2.1.2. Service Loose Coupling	32
2.2.1.3. Service Abstraction	32
2.2.1.5. Service Autonomy	33

2.3. Design patterns.....	33
2.3.1. Definition.....	33
2.3.2. Why design patterns?	34
2.3.3. Categories of SOA design patterns.....	34
2.3.4. Design pattern catalog	35
2.3.5. Pattern Profile	35
2.4. Design Patterns and Design Principles	36
2.4.1. Measures of Application.....	36
2.5. Conclusion	37
Chapter 3: Pattern selection	38
3.1. Our design pattern selection approach.....	38
3.1.1. First phase.....	38
3.1.2. Second phase	44
3.1.3. The whole processes.....	46
3.2. Conclusion	48
Chapter 4: Implementation, results and comparaisn	49
4.1. Implementation	49
4.1.1. Visual studio code platform.....	49
4.1.2. Scientific Python	49
4.1.3. Bulma	50
4.1.4. Electron	50
4.1.5. SQLite	51
4.1.6. Python-shell.....	51
4.2. Dataset.....	51
4.3. Results and comparaisn.....	52
4.3.1. Performance measures.....	52
4.3.2. Classification results.....	55
4.3.3. Conclusion.....	55
4.3.4. Four examples of real-world problems.....	55
4.4. Interfaces of our system	56
4.4.1. drop-down menu.....	57
4.4.2. The search button	58
Conclusion and future work	62
Bibliography	64

List of Tables

Table 1 types of kernels	23
Table 2 Document Vector or Term-Frequency Vector.....	26
Table 3 SOA patterns dataset.....	39
Table 4 raw content.....	52
Table 5 Classification of a document.....	53
Table 6 Confusion matrix	53
Table 7 Results across classifiers.....	54
Table 8 real world examples	56

List of Figures

Figure 1:Text Classification process[4]	2
Figure 2 stemming process	3
Figure 3 bag of words	8
Figure 4 Representing document as Vector	9
Figure 5 document-term matrix	9
Figure 6 decision Tree exemple	16
Figure 7 K-Nearest-Neighbors example	17
Figure 8 Non-linear and linear separation	19
Figure 9 Diagram Small and Large margin separation	20
Figure 10 cosine similarity between vectors.....	26
Figure 11 tight coupling	32
Figure 12 loose coupling.....	32
Figure 13 Preprocessing stages of each design pattern description and problem scenario	40
Figure 14 indexing	40
Figure 15 numerizing the pattern classes.....	41
Figure 16 indexing Rules	41
Figure 17 turning a pattern into a Vector.....	42
Figure 18 turning the user problem into a vector.....	42
Figure 19 training and testing the model	43
Figure 20 predicting the problem class.....	44
Figure 21 dataset for second phase	44
Figure 22 indexing labels.....	45
Figure 23 cosine similarity measure	46
Figure 24 getting the appropriat design pattern	46
Figure 25 the processes of selecting design pattern.....	47
Figure 26 Application Home	56
Figure 27drop-down menu.....	57
Figure 28 text area message warning.....	58
Figure 29 algorithm selection warning	58
Figure 30 text and algorithm are set	59
Figure 31solution using naive bayesian.....	59
Figure 32 solution using decision tree	60
Figure 33 solution using knn.....	60
Figure 34 solution using svm.....	60

List of Acronyms

<i>Soa</i>	<i>Service oriented architecture</i>
<i>Svm</i>	<i>Support vector machine</i>
<i>Vsm</i>	<i>Vector space model</i>
<i>Knn</i>	<i>K nearest neighbor</i>
<i>Idf</i>	<i>Inverse document frequency</i>
<i>Tf</i>	<i>Term frequency</i>

General introduction

In today's world, software engineering has become very important for the technology. It includes the study and application of the engineering in designing, developing and maintaining the software. The aim of software engineering is to produce quality software to be delivered on time keeping it within budget which satisfies the requirement.

Software Design Process in software engineering deals with the activities that are to be performed for the successful completion of a software. Design Patterns are a way to build higher software quality, make the software maintenance easier and represent recommended solutions to design problems. Their extensive use in every day programming weaves valuable architectural information into software systems.

Four authors namely Erich Gama, Richard Helm, Ralph Johnson and John Vlissides in 1994 published a book, Gamma et al. on Design Patterns Elements of Reusable Object-Oriented Software, which was beginning to the concept of software design patterns. The authors collectively called themselves Gang of Four (GOF).

Softwares have a complexity problem. There appears to be a geometric relationship between the complexity of a monolithic system and its stability and maintainability. So, a system that is twice as complex as another one will be maybe four times more expensive to maintain and a quarter as stable. There is also the human and organisational side of software. Individual teams tend to build or buy their own solutions. The organisation then needs to find ways for these disparate systems to share information and workflow. Small single purpose systems are always a better choice than large monolithic ones. If we build our systems as components, we can build and maintain them independently. SOA is a set of design patterns that guide us in building and integrating these mini-application pieces. But To be able to use design patterns and to attain their intended benefits, designers are expected to have a good understanding and experience with them in order to be able to select the right pattern for their context.

Software design pattern selection is an important part in the software designing process. Pattern selection can be made by various methods using manual and automatic approaches but since it is not feasible for human to process high volume of textual data and classify and retrieve patterns and with the omnipresence of Machine learning in several

areas including classification. We can forget about old manual approaches and start looking for Automatic approaches that represent a better way.

where automatic method includes Machine Learning algorithms. Supervised learning is being implemented for selecting software design patterns. At the end, the design pattern selection model will be proposed. Some studies are done on the design patterns selection as in that article [1]. where it uses machine learning to find object oriented design patterns. But in this manuscript, we propose an approach for selecting SOA design patterns using machine learning techniques to predict the class of the design patterns that our problem belongs to and then using cosine similarity we can retrieve the most appropriate or in other words the most suitable pattern to our problem.

Thesis structure

This thesis is organized as follows:

Chapter 1 describes the architecture of the text classification model, details the feature construction processes and the representation in the vector space model then provides a brief introduction into the algorithms of machine learning and gives an overview about some similarity measures.

Chapter 2 discusses all the concepts related with soa technology, fundamentals of SOA design patterns, types of design patterns and general definitions.

Chapter 3 details the pattern selection problematic and our proposed solution.

Chapter 4 introduces the results of Implementation of the studied algorithm.

Finally, we will finish this thesis by a general conclusion and future work and a bibliography.

Chapter 1: Text-classification

Introduction:

Text classification (text categorization) is one of the most prominent application of machine Learning. It is used to automatically assign predefined categories (labels) to free-text documents. The purpose of text classification is to give conceptual organization to a large collection of documents. It has become more relevant with exponential growth of the data, and the wide applicability in real world applications.

1.1. Definition

*Text Classification [2] involves assigning a text document to a set of pre-defined classes automatically, using a machine learning technique. Text classification is a supervised learning technique that uses labeled training data to derive a classification system and then automatically classifies unlabelled text data using the derived classifier. The most data for text classification is collected from the web, through newsgroups, bulletin boards, and broadcast or printed news... many classification methods have been developed with the aid of learning algorithms such as **Naïve Bayesian**, **K-Nearest Neighbor (KNN)**, **Support Vector Machine** and **Decision Tree**.*

[3]More formally, if d_i is a document of the entire set of documents D and $\{c_1, c_2, \dots, c_n\}$ is the set of all the categories, then text classification assigns one category c_j to a document d_i .

1.2. Text Classification Process

*[4]The task of Text Classification is carried out in several sub phases, see (**Figure1**). They are: **Data Collection & Representation**, **Document Pre-processing phase**, **Feature selection or Transformation**, **Applying a text classifier** and **performance evaluation**.*

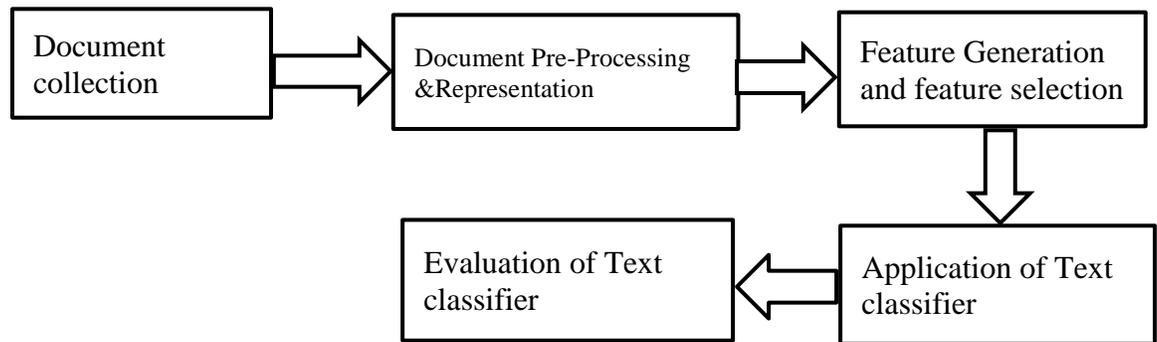


Figure 1:Text Classification process[4]

1.2.1. Document Collection

This is first step of classification process. The text documents from various sources are collected in different formats of document like html,.doc, .pdf, web content etc... These documents are used during training and testing the classifier.

1.2.2. Document Pre-Processing &Representation

A document is considered as collection of words or Bag of words. Only few of these words are used for the classification. The words which are insignificant seems to degrade the process of classification. Hence these documents need to be preprocessed. The steps of Pre –processing are:

1.2.2.1. Tokenization

A document is defined as collection of strings of sentences, which are then isolated into set of tokens by removing spaces and commas.

1.2.2.2. Removing unuseful stop words

Certain words that are not useful for classification. These are called Stop words. This step involves removal of frequently occurring Stop words like “the”, “a”, “and”, ‘of’, etc. from the list of tokens.

1.2.2.3. Stemming

Another very important step to reduce the number of words is to use stemming which converts different word form into similar canonical form. This technique is used to find the root or stem of a word. Stemming converts words to their root words. For example, the words like waiting, rained converted to wait, rain respectively.

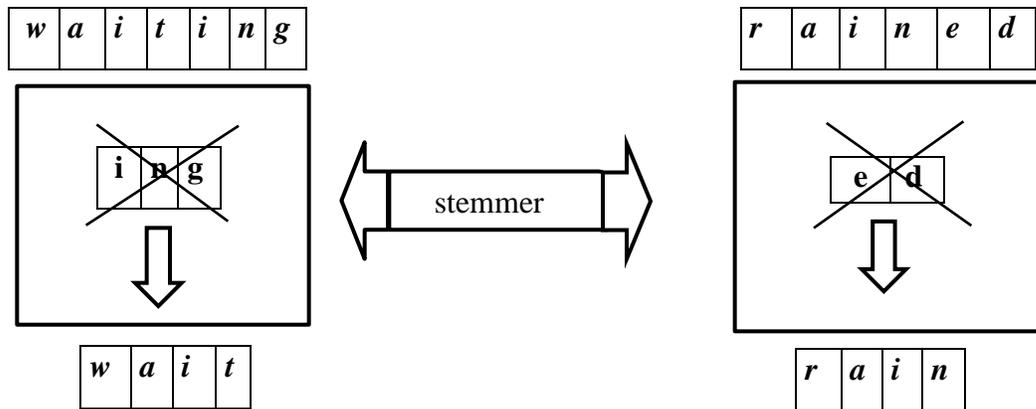


Figure 2 stemming process

1.2.3. Feature Generation/Text Transformation

Text Transformation or Feature generation is one of the important steps of pre-processing, which reduces the complexity of documents. Documents have to be transformed from the full text version to a vector of words using the vector Space model see (Constructing a Vector Space Model **section 1.4** page 7).

1.2.4. Feature Selection

Feature Selection is the dimension reduction step. The main aim of Feature selection is to select subset of features from the original Documents. Feature Selection is performed by retaining the most significant words according to predetermined measure of the significance of the word. Most important feature evaluation metric are information gain (IG), Chi-square, expected cross entropy, term frequency, Odds Ratio, the weight of evidence of text, mutual information, Gini index. Advantage of Feature selection is that it

improves the accuracy and efficiency of the classifier. Also feature selection reduces the size of dataset and provides minimum search space for more information see(Feature construction section 1.3 page 4).

1.2.5. Text Classification

*The objective of the classification is to classify the pre processed documents into predefined categories by using the training data set. The documents can be classified by three ways: supervised, unsupervised and semi supervised methods. In Supervised: All data is labelled and the algorithms learn to predict the output from the input data. Unsupervised: All data is unlabeled and the algorithms learn to inherent structure from the input data. Semi-supervised: Some data is labelled but most of it is unlabeled and a mixture of supervised and unsupervised techniques can be used. Classification is the problem of Supervised Learning. A number of Classifiers such as Bayesian classifier, Decision Tree, K-nearest neighbour (KNN), Support Vector Machines (SVMs), Neural Networks, are available see (Machine learning algorithms **section 1.5 page 10**).*

1.2.6. Evaluation of Text classifier

Another major process that needs to be performed for the text classification process is the evaluation of classifier. the process is being performed in order to judge the accuracy and the reliability of the text classifier.

1.3. Feature construction

Introduction

Feature construction is one of the key steps in the data analysis process, largely conditioning the success of any subsequent statistics or machine learning endeavor. In particular, one should beware of not losing information at the feature construction stage. Feature construction consists of various techniques and approaches which convert textual data into a feature-based representation. Since traditional machine learning and data mining techniques are generally not designed to deal directly with textual data, feature

construction is an important preliminary step, converting source documents into a representation that a data mining algorithm can then work with.

Some of the limitations are: high dimensionality of the representation, loss of correlation with adjacent words and loss of semantic relationship that exist among the terms in a document. To overcome these problems, feature identification, term weighting methods can be used.

1.3.1. Feature identification

[5]in the document representation One particularity of the text categorization problem is that the number of features (unique words or phrases) can easily reach orders of tens of thousands. This raises big hurdles in applying many sophisticated learning algorithms to the text categorization.

Thus, dimension reduction methods are called for. Two possibilities exist, either selecting a subset of the original features, or transforming the features into new ones.

1.3.1.1. Feature-selection

Choose the best features (= representation words) for your task, remove the rest.

[6]Feature selection is the process of determining the terms to be used in classification. is applied almost in all text classification studies.

excluding very frequent and/or very rare words – excluding stop words ('of', 'the', 'and', 'or, ...).

[5]The aim of feature-selection methods is the reduction of the dimensionality of the dataset by removing features that are considered irrelevant for the classification. This transformation procedure has been shown to present a number of advantages, including smaller dataset size, smaller computational requirements for the text categorization algorithms (especially those that do not scale well with the feature set size) and considerable shrinking of the search space. The goal is the reduction of the curse of dimensionality to yield improved classification accuracy and decreases process time. Another benefit of feature selection is its tendency to reduce overfitting, i.e. the phenomenon by which a classifier is tuned also to the contingent characteristics of the

training data rather than the constitutive characteristics of the categories, and therefore, to increase generalization.

1.3.1.2. Feature extraction(reparameterization)

[7]In this feature reduction technique, the original vector space is transformed to form a new minimalistic feature vector space. Unlike Feature selection technique, in this method, all the features are used. The original features are transformed into a smaller set of transformed features which might not be meaningful to humans but are composed of original human understandable features for example (Stemming and lemmatizing).

1.3.2. Feature weighting

[8]Term weighting is one of the known concepts in TC, which can be defined as a factor given to a term in order to reflect the importance of that term.

In text representation, terms are words, phrases, or any other indexing units used to identify the contents of a text. However, no matter which indexing unit in use, each term in a document vector must be associated with a value (weight) which measures the importance of this term and denotes how much this term contributes to the categorization task of the document.

There are many term weighting approaches:[9]

1.3.2.1. TF weighting

Term Frequency (TF) measures how many times a term occurs in a document.

1.3.2.2. IDF weighting

Inverse Document Frequency (IDF), helps in determining the importance of a term. When we compute term frequency, we give equal importance to all the terms. But certain terms, such as “the”, “that”, and “is”, may appear very frequently which are not important. So, we need to bring down the weights of frequent terms and increase the weights of the rare terms, by calculating the following:

$$IDF = \frac{\log_2(\text{Number of document with term } t \text{ in them})}{\text{Total number of documents}} \quad (1.1)$$

1.3.2.3. TF-IDF weighting

is the most widely used and considered as one of the most appropriate term weighting schemes. This TF.IDF is employed to get rid of terms with lower weights from documents and helps to increase the retrieval effectiveness. Term frequency–inverse document frequency, is a numerical statistic that tells us how important a word is to a document in a collection or corpus. It is mostly used as a weighting factor in various processes used for information retrieval and text mining. The increase in the TF.IDF value of a word is directly proportional to the number of times that word occurs in the document, but is neutralized by the frequency of the word in the corpus, which helps to balance off those words which appear more frequently in general.

$$TF.IDF = (\text{Term Frequency} * \text{Inverse Document Frequency}) \quad (1.2)$$

1.4. Constructing a Vector Space Model

Introduction:

[10]*The documents representation is one of the pre-processing techniques that is used to reduce the complexity of the documents and make them easier to handle, the document has to be transformed from the full text version to a document vector. The most commonly used document representation is called vector space model (VSM), documents are represented by vectors of words.*

1.4.1. Text representation models

[11]*How can texts be represented? The most popular text representation model is the vector space model. In this model, each document is represented by a vector whose dimensions correspond to features found in the corpus. When features are single words, the text representation is called bag-of-words.*

[5]*A document is a sequence of words. So, each document is usually represented by an array of words. The set of all the words of a training set is called vocabulary, or feature set.*

1.4.1.1. Bag of words

[12]Bag-of-words depiction form is used with the variety of instances of Vector Space Model. In the representation of bag- of- word, it is assumed that the content of a document could be determined by the set of terms it has. Within the vocabulary area documents can be represented as points, i.e., a document is represented by a numerical vector of length equal to the numeral of different terms within the vocabulary (the set of all different terms within the document collection) see(**figure 3 below**).

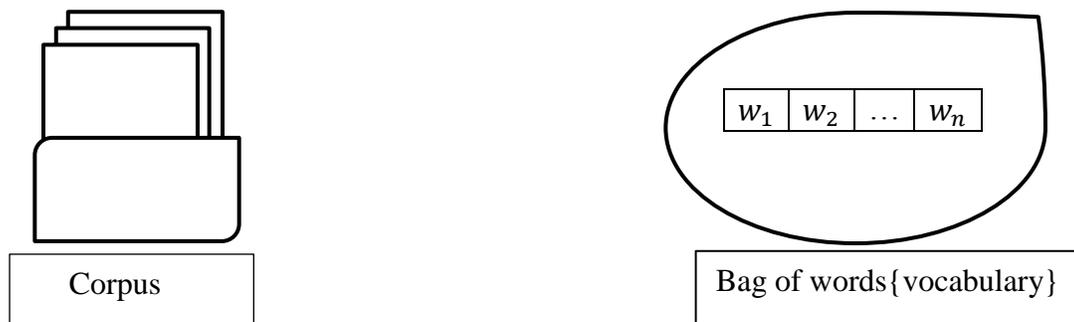


Figure 3 bag of words

1.4.1.2. Document representation

[12]The main idea consists of representing a document as a vector, in particular as a bag of words. This set contains only the words that belong to the document and their frequency. This means that a document is represented by the words that it contains. In this representation, punctuation is ignored, and a sentence is broken into elementary elements (words) losing the order and the grammar information. These two observations are crucial, because they show that it is impossible to reconstruct the original document given its bag of words; it means that the mapping is not one to one. Consider a corpus as a set of documents, and a dictionary as the set of words that appear into the corpus. This can view a document as a bag of terms or bag of words. This bag can be seen as a vector, where each component is associated with one term from the dictionary see(**Figure 4**) below.

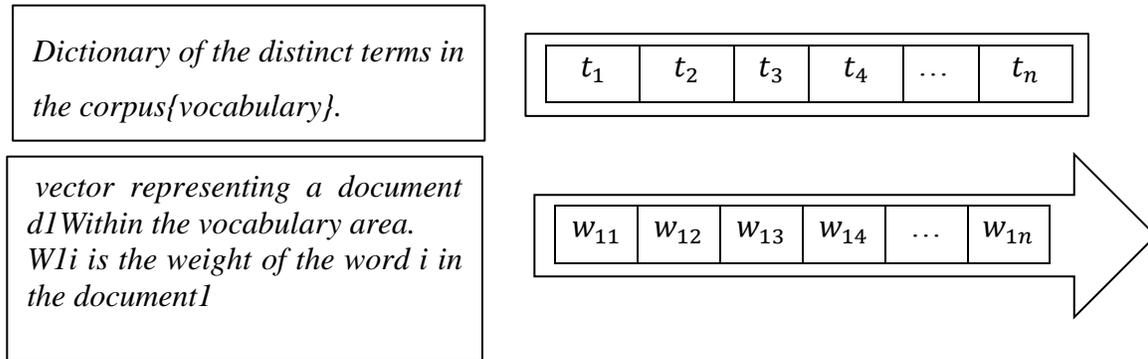


Figure 4 Representing document as Vector

1.4.1.3. Document-term Matrix

If we have a large collection of documents, and hence a large number of document vectors, it is convenient to organize the vectors into a matrix.

[12]A corpus of n documents can be represented as a document-term matrix whose rows are indexed by the documents and whose columns are indexed by the terms. Each entry at position (i, j) is the weight of the term j in document i .

as shown in following matrix:

$$\begin{pmatrix} D/T & t_1 & t_j & \dots & t_m \\ d_1 & w_{11} & w_{1j} & \dots & w_{1m} \\ d_i & w_{i1} & w_{ij} & \dots & w_{im} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ d_n & w_{n1} & w_{nj} & \dots & w_{nm} \end{pmatrix}$$

Figure 5 document-term matrix

1.4.2. The whole processes from scratch

- ✓ First after the preprocessing step and getting the important terms we make a dictionary out of this set.
- ✓ Second: we use this dictionary to index each document.
- ✓ Third: organize the vectors into a matrix. Where the rows are the documents and The columns are the terms.

1.5. Machine learning algorithms

Introduction:

After feature selection and transformation, the documents can be easily represented in a vsm form that can be used by a ML algorithm. Many text classifiers have been proposed in the literature using machine learning techniques, probabilistic models, etc. They often differ in the approach adopted: decision trees, naïve-Bayes, rule induction, neural networks, nearest neighbors, and lately, support vector machines. Although many approaches have been proposed, automated text classification is still a major area of research primarily because of the effectiveness of current automated text classifiers is not faultless and still needs improvement.

1.5.1. Types of machine learning

[13]The documents can be classified by three ways, unsupervised, supervised and semi supervised methods. Many techniques and algorithms are proposed recently for the classification of electronic documents. Machine Learning algorithms are classified as

1.5.1.1. Supervised Machine Learning

Machine learning algorithms that make predictions on given set of samples. Supervised machine learning algorithm searches for patterns within the value labels assigned to data points.

1.5.1.2. Unsupervised Machine Learning Algorithms

There are no labels associated with data points. These machine learning algorithms organize the data into a group of clusters to describe its structure and make complex data look simple and organized for analysis.

1.5.1.3. Reinforcement Machine Learning Algorithms

These algorithms choose an action, based on each data point and later learn how good the decision was. Over time, the algorithm changes its strategy to learn better and achieve the best reward.

1.5.2. Machine learning algorithms

Common algorithms for performing classification:

1.5.2.1. Naïve Bayesian

[14] Naïve bayes is a probabilistic classifier which do not work on any single algorithm rather it works on a family of algorithm that all work on a single principle of classification. All of the extracted features using this classifier are independent of each other. The advantage of using this classifier is that it works good on both numeric as well as textual data and moreover it is easier to implement. The disadvantage of this classifier is that its performance gets poorer when the extracted features are correlated to each other.

1.5.2.1.1. Definitions

1.5.2.1.1.1. The posterior probability

is defined as the probability after observing the specific characteristics of the test instance.

1.5.2.1.1.2. The prior probability

is simply the fraction of training records belonging to each particular class, with no knowledge of the test instance.

1.5.2.1.2. Principle

[15] *the posterior probability of a particular class is estimated by determining the class-conditional probability and the prior class separately and then applying Bayes theorem to find the parameters. Consider a test instance with d different features, which have values $X = \langle x_1 \dots x_d \rangle$ respectively. Its is desirable to determine the posterior probability that the class $Y(T)$ of the test instance T is i . In other words, we wish to determine the posterior probability $P(Y(T) = i | x_1 \dots x_d)$. Then, the Bayes rule can be used in order to derive the following:*

$$P(Y(T) = i | x_1 \dots x_d) = \frac{P(Y(T) = i) \cdot P(x_1 \dots x_d / Y(T) = i)}{P(x_1 \dots x_d)} \quad (2.1)$$

Since the denominator is constant across all classes, and one only needs to determine the class with the maximum posterior probability, one can approximate the aforementioned expression as follows:

$$P(Y(T) = i | x_1 \dots x_d) \propto P(Y(T) = i) \cdot P(x_1 \dots x_d / Y(T) = i) \quad (2.2)$$

The key here is that the expression on the right can be evaluated more easily in a data-driven way, as long as the naive Bayes assumption is used for simplification. Specifically, in **Equation(2.2)**, the expression $P(x_1 \dots x_d / Y(T) = i)$ can be expressed as the product of the feature-wise conditional probabilities.

$$P(x_1 \dots x_d / Y(T) = i) = \prod_{j=1}^d P(x_j / Y(T) = i) \quad (2.3)$$

This is referred to as conditional independence, and therefore the Bayes method is referred to as “naive”.

1.5.2.1.3. Multinomial Naive Bayes Classifier

[16]Multinomial Naive Bayes is a Bayesian Classifier where the number of occurrences of each token is also accounted for. We would expect this to work better than the Naive Bayes model that builds on a binary ‘Bernoulli’ value for each attribute (token) indicating occurrence or non-occurrence of the token. Multinomial Naive Bayes has traditionally been used for document classification.

multinomial model outperforms the Bernoulli model when the size of the text document is large and incorporates a large vocabulary.

1.5.2.2. Decision tree

1.5.2.2.1. Definition

[17]A decision tree is a flowchart-like tree structure see(**Figure 6**), where each internal node (nonleaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The topmost node in a tree is the root node.

1.5.2.2.2. Decision tree induction

[17]is the learning of decision trees from class-labeled training tuples.

1.5.2.2.3. The split criterion

[18]The decision at a particular node of the tree, which is referred to as the split criterion, is typically a condition on one or more feature variables in the training data. The split criterion divides the training data into two or more parts.

For example, consider the case where Age is an attribute, and the split criterion is $Age \leq 30$. In this case, the left branch of the decision tree contains all training examples with age at most 30, whereas the right branch contains all examples with age greater than 30

The goal is to identify a split criterion so that the level of “mixing” of the class variables in each branch of the tree is reduced as much as possible.

The design of the split criterion depends on the nature of the underlying attribute:

1.5.2.2.3.1. Binary attribute

Only one type of split is possible, and the tree is always binary. Each branch corresponds to one of the binary values.

1.5.2.2.3.2. Categorical attribute

If a categorical attribute has r different values, there are multiple ways to split it. One possibility is to use an r -way split, in which each branch of the split corresponds to a particular attribute value. The other possibility is to use a binary split by testing each of the $2^r - 1$ combinations (or groupings) of categorical attributes, and selecting the best one. This is obviously not a feasible option when the value of r is large. A simple approach that is sometimes used is to convert categorical

1.5.2.2.3.3. Numeric attribute

If the numeric attribute contains a small number r of ordered values (e.g., integers in a small range $[1, r]$), it is possible to create an r -way split for each distinct value. However, for continuous numeric attributes, the split is typically performed by using a binary condition, such as $x \leq \alpha$, for attribute value x and constant a . Consider the case where a node contains m data points. Therefore, there are m possible split points for the attribute, and the corresponding values of a may be determined by sorting the data in the node along this attribute. One possibility is to test all the possible values of α for a split and select the

best one. A faster alternative is to test only a smaller set of possibilities for α , based on equi-depth division of the range.

Many of the aforementioned methods requires the determination of the “best” split from a set of choices. Specifically, it is needed to choose from multiple attributes and from the various alternatives available for splitting each attribute.

1.5.2.2.4. Nodes

Each node in the decision tree logically represents a subset of the data space defined by the combination of split criteria in the nodes above it. The decision tree is typically constructed as a hierarchical partitioning of the training examples.

1.5.2.2.5. Stopping Criterion and Pruning

[18]The stopping criterion for the growth of the decision tree is intimately related to the underlying pruning strategy. When the decision tree is grown to the very end until every leaf node contains only instances belonging to a particular class, the resulting decision tree exhibits 100 % accuracy on instances belonging to the training data. However, it often generalizes poorly to unseen test instances because the decision tree has now overfit even to the random characteristics in the training instances. Most of this noise is contributed by the lower level nodes, which contain a smaller number of data points.

To reduce the level of overfitting, one possibility is to stop the growth of the tree early. Unfortunately, there is no way of knowing the correct point at which to stop the growth of the tree.

1.5.2.2.6. How are decision trees used for classification?

[17]Given a tuple X for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple

1.5.2.2.7. Why are decision tree classifiers so popular?

[17]The construction of decision tree classifiers does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery.

1.5.2.2.8. Algorithm

[17] *Most algorithms for decision tree induction also follow a top-down approach, which starts with a training set of tuples and their associated class labels. The training set is recursively partitioned into smaller subsets as the tree is being built. A basic decision tree algorithm is summarized below*

[18] **Algorithm** *GenericDecisionTree* (Data Set: *D*)

```
begin  
    Create root node containing D;  
    repeat  
        Select an eligible node in the tree;  
        Split the selected node into two or more nodes based on a pre-defined  
        split criterion;  
    until no more eligible nodes for split;  
    Prune overfitting nodes from tree;  
    Label each leaf node with its dominant class;  
End
```

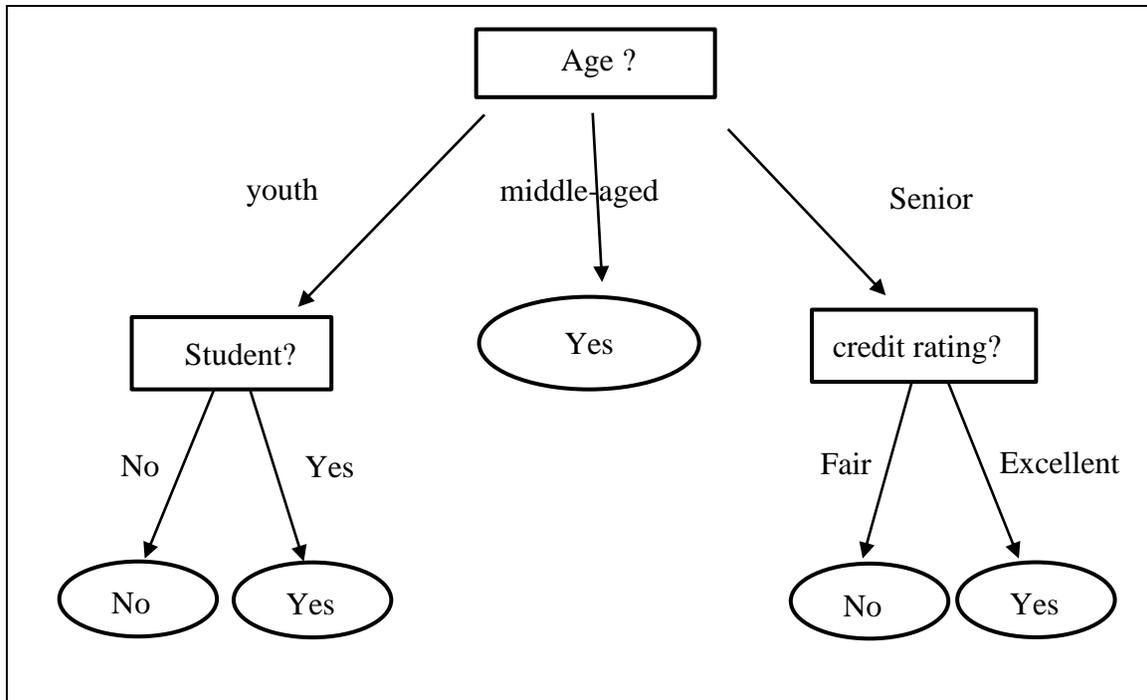


Figure 6 decision Tree exemple

A decision tree for the concept buys computer, indicating whether a customer at All Electronics is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either buys computer = yes or buys computer = no).

1.5.2.3. KNN (k-nearest-neighbor)

1.5.2.3.1. Principle

[19]Given an unlabeled sample, Knearest neighbor classifier will search the pattern space for k-objects that are closest to it and will delegate the class by identifying the class label which is frequently used.

1.5.2.3.2. Detailed:

[20]A K-Nearest Neighbor algorithm is also called as instance-based learning algorithm Nearest Neighbor classifier are based on learning by analogy, that is by comparing a given test tuple with training tuples that are similar to it. Each tuple represents a point in an n-dimension pattern space. When given an unseen tuple, a K-nearest neighbor classifier searches the pattern space for different values of K (which can take any value 1 through some arbitrary number) the training tuples that are closest to the unseen tuple. Depending on the value of K, K training tuples are used which are near to

the unseen tuple. The Euclidean distance between two points or tuples say X_1 and X_2 which have 'n' component elements is used to find the similarity (closeness) between the tuples using the Euclidean distances. the distance between the training and a particular test document is measured, the class with the smallest distance in the training data is taken as the class of the test data.

as in in (**Figure 7**)below ,K value in K-NN is 3, we take 3 smallest distances, and if both belong to same class than the test tuple is considered to belong to the same class as of the training document class.

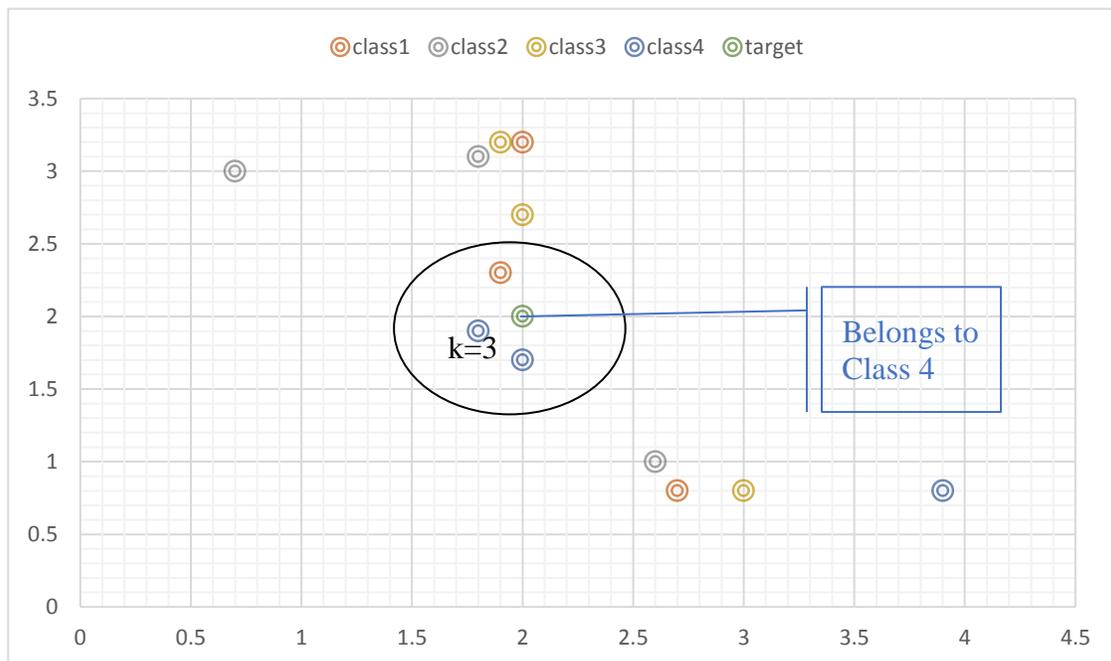


Figure 7 K-Nearest-Neighbors example

1.5.2.4. Svm

[21]Support vector machines have strong theoretical foundations and excellent empirical successes. They have been applied to tasks such as handwritten digit recognition, object recognition, and text classification.

1.5.2.4.1. Why Should SVMs Work Well for Text Categorization?

[22]To Find out what methods are promising for learning text classifiers, we should find out more about the properties of text.

- ✓ **High dimensional input space**

When learning text classifiers, one has to deal with very many (more than 10000) features. Since SVMs use overfitting protection which does not necessarily depend on the number of features, they have the potential to handle these large feature spaces.

✓ **Few irrelevant features**

One way to avoid these high dimensional input spaces is to assume that most of the features are irrelevant. Feature selection tries to determine those. Unfortunately, in text categorization there are only very few irrelevant features. even features ranked lowest still contain considerable information and are somewhat relevant. this leads to the conjecture that a good classifier should combine many features (learn a "dense" concept) and that feature selection is likely to hurt performance due to a loss of information.

✓ **Document vectors are sparse**

For each document d_i , the corresponding document vector \vec{d}_i contains only few entries which are not zero. for the mistake bound model that "additive" algorithms, which have a similar inductive bias like SVMs, are well suited for problems with dense concepts and sparse instances.

✓ **Most text categorization problems are linearly separable**

There's plenty of instances that are linearly separable.

1.5.2.4.2. Binary classification using Svm

[23]For binary classification problems, the idea behind SVM is to split the data in finest method. Binary classification is used when we need to classify the two data sets. There are numerous examples of Binary classification like try-outs (one either makes or fails to make the team), claim size (large claims are above some threshold and small claims below), and fingerprint identification (matched or unmatched). Support vector machines are primarily designed for 2-class classification problems.

Support Vector Machine consider 2 approaches:

- ✓ Case when the data are linearly separable.
- ✓ Case when the data are non-linearly separable.

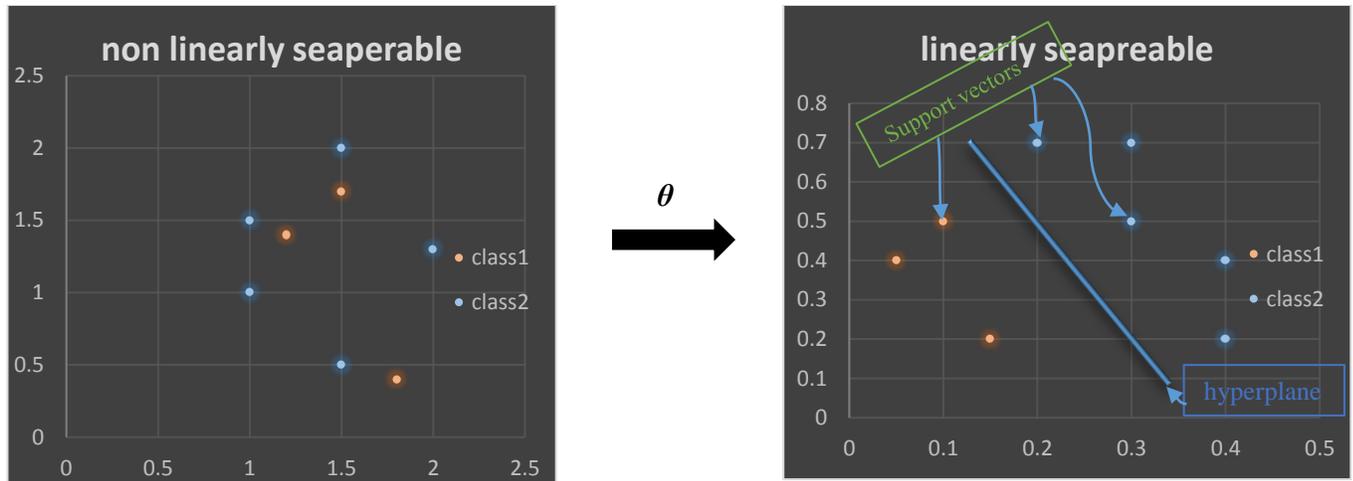


Figure 8 Non-linear and linear separation

1.5.2.4.2.1. Data are linearly-separable

there are many linear decision boundaries that divide the data. But only one of these achieves maximum division. The main purpose we need it is because if we use a decision boundary to classify, it may end up nearer to one set of datasets compared to others and we do not want this to happen and thus concept of maximum margin classifier or hyperplane as an apparent solution. Support vectors are the data points that lie closest to the decision surface see (Figure 8)above. Support Vectors can be described as those data points that the margins push up against. They are the most difficult to classify. The major problem here is to find the only optimal margin of the separating hyperplane $w^t x + b$ the one that provides maximum margin between the classes Figure 9. This margin guaranties the lowest rate of misclassification. The further advantage of margin would be avoiding local minima and better classification.

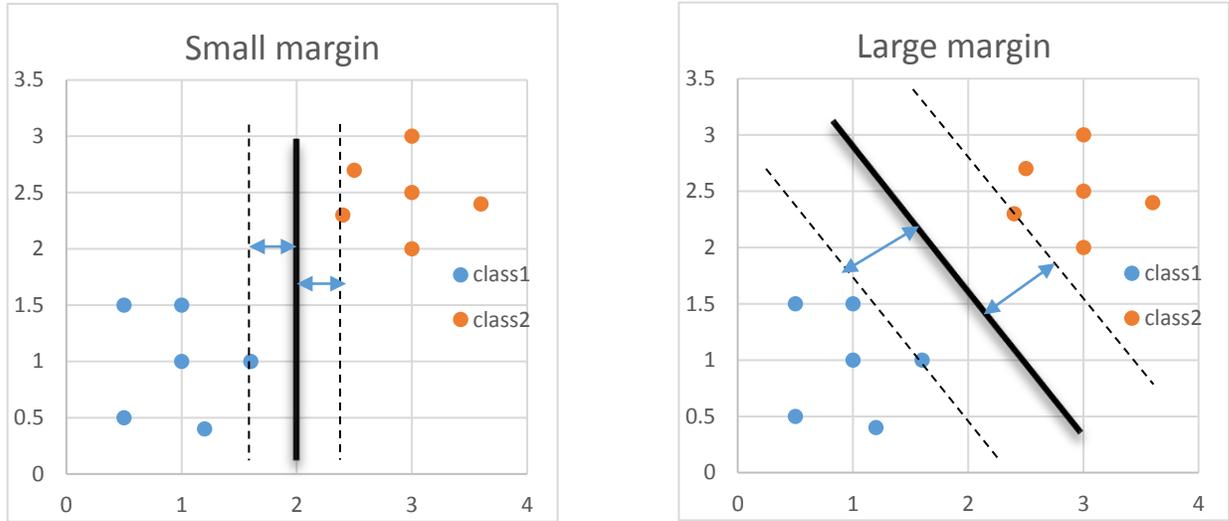


Figure 9 Diagram Small and Large margin separation

1.5.2.4.2.1.1. How do we determine the maximum margin hyperplane?

[18]The way to do this is to set up a nonlinear programming optimization formulation that maximizes the margin by expressing it as a function of the coefficients of the separating hyperplane. The optimal coefficients can be determined by solving this optimization problem. Let the n data points in the training set D be denoted by $(\bar{X}_1, y_1) \dots (\bar{X}_n, y_n)$, where \bar{X}_i is a d -dimensional row vector corresponding to the i th data point, and $y_i \in \{-1, +1\}$ is the binary class variable of the i th data point. Then, the separating hyperplane is of the following form:

$$W \cdot \bar{X} + b \quad (3.1)$$

Here, $W = (w_1 \dots w_d)$ is the d -dimensional row vector representing the normal direction to the hyperplane, and b is a scalar, also known as the bias. The vector W regulates the orientation of the hyperplane and the bias b regulates the distance of the hyperplane from the origin.

W and b need to be learned from the training data to maximize the margin of separation between the two classes. Because it is assumed that the classes are linearly separable, such a hyperplane can also be assumed to exist.

All data points \bar{X}_i with $y_i = +1$ will lie on one side of the hyperplane satisfying

$\overline{W} \cdot \overline{X}_i + b \geq 0$ Similarly, all points with $y_i = -1$ will lie on the other side of the hyperplane satisfying $\overline{W} \cdot \overline{X}_i + b \leq 0$.

$$\overline{W} \cdot \overline{X}_i + b \geq 0 \quad \forall i: y_i = +1 \quad (3.2)$$

$$\overline{W} \cdot \overline{X}_i + b \leq 0 \quad \forall i: y_i = -1 \quad (3.3)$$

These constraints do not yet incorporate the margin requirements on the data points. A stronger set of constraints are defined using these margin requirements. It may be assumed that the separating hyperplane $\overline{W} \cdot \overline{X} + b = 0$ is located in the center of the two margin-defining hyperplanes. Therefore, the two symmetric hyperplanes touching the support vectors can be expressed by introducing another parameter c that regulates the distance between them.

$$\overline{W} \cdot \overline{X} + b = +c \quad (3.4)$$

$$\overline{W} \cdot \overline{X} + b = -c \quad (3.5)$$

It is possible to assume, without loss of generality, that the variables \overline{W} and b are appropriately scaled, so that the value of c can be set to 1. Therefore, the two separating hyperplanes can be expressed in the following form:

$$\overline{W} \cdot \overline{X} + b = +1 \quad (3.6)$$

$$\overline{W} \cdot \overline{X} + b = -1 \quad (3.7)$$

These constraints are referred to as margin constraints. The two hyperplanes segment the data space into three regions. It is assumed that no training data points lie in the uncertain decision boundary region between these two hyperplanes, and all training data points for each class are mapped to one of the two remaining (extreme) regions. This can be expressed as pointwise constraints on the training data points as follows:

$$\overline{W} \cdot \overline{X}_i + b \geq 1 \quad \forall i: y_i = +1 \quad (3.8)$$

$$\overline{W} \cdot \overline{X}_i + b \leq -1 \quad \forall i: y_i = -1 \quad (3.9)$$

1.5.2.4.2.1.2. Test instance

In expression for W in terms of the Lagrangian multipliers and the training data points:

$$\sum_{i=1}^n \lambda_i y_i \overline{X}_i \quad (3.10)$$

For a test instance Z , its class label $F(Z)$ is defined by the decision boundary obtained by substituting for W in terms of the Lagrangian multipliers.

$$F(Z) = \text{sign}\{\overline{W} \cdot \overline{Z} + b\} = \text{sign}\left\{\left(\sum_{i=1}^n \lambda_i y_i \overline{X}_i \cdot \overline{Z}\right) + b\right\} \quad (4.1)$$

1.5.2.4.2.2. Case when the data are non-linearly separable

data are not linearly separable i.e. in such cases no straight line can be found that would divide the classes. Linear svm's can be extended to generate nonlinear SVM'S for classification of linearly inseparable data. Such svm are capable of finding nonlinear decision boundaries like using:

1.5.2.4.2.2.1. the Kernel Trick

The kernel trick leverages the important observation that the SVM formulation can be fully solved in terms of dot products (or similarities) between pairs of data points. One does not need to know the feature values. Therefore, the key is to define the pairwise dot product (or similarity function) directly in the d -dimensional transformed representation $\Phi(X)$, with the use of a kernel function $K(\overline{X}_i, \overline{X}_j)$.

$$K(\overline{X}_i, \overline{X}_j) = \Phi(\overline{X}_i) \cdot \Phi(\overline{X}_j) \quad (4.2)$$

To effectively solve the SVM, recall that the transformed feature values $\Phi(\bar{X})$ need not be explicitly computed, as long as the dot product (or kernel similarity) $K(\bar{X}_i, \bar{X}_j)$ is known.

the term $\bar{X}_i \cdot \bar{Z}$ in **Equation (4.1)** can be replaced by $K(\bar{X}_i, \bar{Z})$ to perform SVM classification.

$$F(Z) = \text{sign} \left\{ \left(\sum_{i=1}^n \lambda_i y_i \cdot K(\bar{X}_i, \bar{Z}) \right) + b \right\} \quad (4.3)$$

all computations are performed in the original space, and the actual transformation $\Phi(\cdot)$ does not need to be known as long as the kernel similarity function $K(\cdot, \cdot)$ is known. By using kernel-based similarity with carefully chosen kernels, arbitrary nonlinear decision boundaries can be approximated. There are different ways of modeling similarity between \bar{X}_i and \bar{X}_j . Some common choices of the kernel function are as follows:

Function	Form
Gaussian radial basis kernel	$K(\bar{X}_i, \bar{X}_j) = e^{-\ \bar{X}_i - \bar{X}_j\ ^2 / 2\sigma^2}$
Polynomial kernel	$K(\bar{X}_i, \bar{X}_j) = (\bar{X}_i \cdot \bar{X}_j + c)^h$
Sigmoid kernel	$K(\bar{X}_i, \bar{X}_j) = \tanh(k\bar{X}_i \cdot \bar{X}_j - \delta)$

Table 1 types of kernels

Many of these kernel functions have parameters associated with them. In general, these parameters may need to be tuned by holding out a portion of the training data, and using it to test the accuracy of different choices of parameters.

1.5.2.4.3. Multi class classification methods

[24] Although SVMs were originally designed as binary classifiers, approaches that address a multi-class problem as a single “all-together” optimization problem exist, but are computationally much more expensive than solving several binary problems. A variety of techniques for decomposition of the multi-class problem into several binary problems using Support Vector Machines as binary classifiers have been proposed, and several widely used are:

1.5.2.4.3.1. One-against-all

For the N -class problems ($N > 2$), N 2-class SVM classifiers are constructed. The i th SVM is trained while labeling all the samples in the i th class as positive examples and the rest as negative examples. In the recognition phase, a test example is presented to all N SVMs and is labeled according to the maximum output among the N classifiers. The disadvantage of this method is that the number of training samples is too large, so it is difficult to train.

1.5.2.4.3.2. One-against-one

This algorithm constructs $N(N-1)/2$ 2-class classifiers, using all the binary pair-wise combinations of the N classes. Each classifier is trained using the samples of the first class as positive examples and the samples of the second class as negative examples. To combine these classifiers, it naturally adopts Max Wins algorithm that finds the resultant class by first voting the classes according to the results of each classifier and then choosing the class that is voted most. The disadvantage of this method is that every test sample has to be presented to large number of classifiers ($N(N-1)/2$). This results in faster training but slower testing, especially when the number of the classes in the problem is big.

1.6. Similarity Mathematical Metrics

Introduction:

Measuring similarity or distance between two information entities is a core requirement for all information discovery tasks (whether IR or Data mining).

Use of appropriate measures not only improves the quality of information selection but it also helps reduce the time and processing costs.

1.6.1. Types of similarity measures

1.6.1.1. Cosine-Based Similarity

[25] *A document can be represented by thousands of attributes, each recording the frequency of a particular word (such as a keyword) or phrase in the document. Thus, each document is an object represented by what is called a term-frequency vector. For example, in (Table 2) below, we see that Document1 contains five instances of the word team, while hockey occurs three times. The word coach is absent from the entire document, as indicated by a count value of 0. Such data can be highly asymmetric. Term-frequency vectors are typically very long and sparse (i.e., they have many 0 values). Applications using such structures include information retrieval, text document clustering, biological taxonomy, and gene feature mapping.*

Documents	Terms	team	Coach	hockey	Baseball	soccer	penalty	score	win	loss	season
DOCUMENT1		5	0	3	0	2	0	0	2	0	0
DOCUMENT2		3	0	2	0	1	1	0	1	0	1
DOCUMENT3		0	7	0	2	1	0	0	3	0	0
DOCUMENT4		0	1	0	0	1	2	2	0	3	0

Table 2 Document Vector or Term-Frequency Vector

1.6.1.1.1. Why cosine similarity

[25]The traditional distance measures do not work well for such sparse numeric data. For example, two term-frequency vectors may have many 0 values in common, meaning that the corresponding documents do not share many words, but this does not make them similar. We need a measure that will focus on the words that the two documents do have in common, and the occurrence frequency of such words. In other words, we need a measure for numeric data that ignores zero-matches.

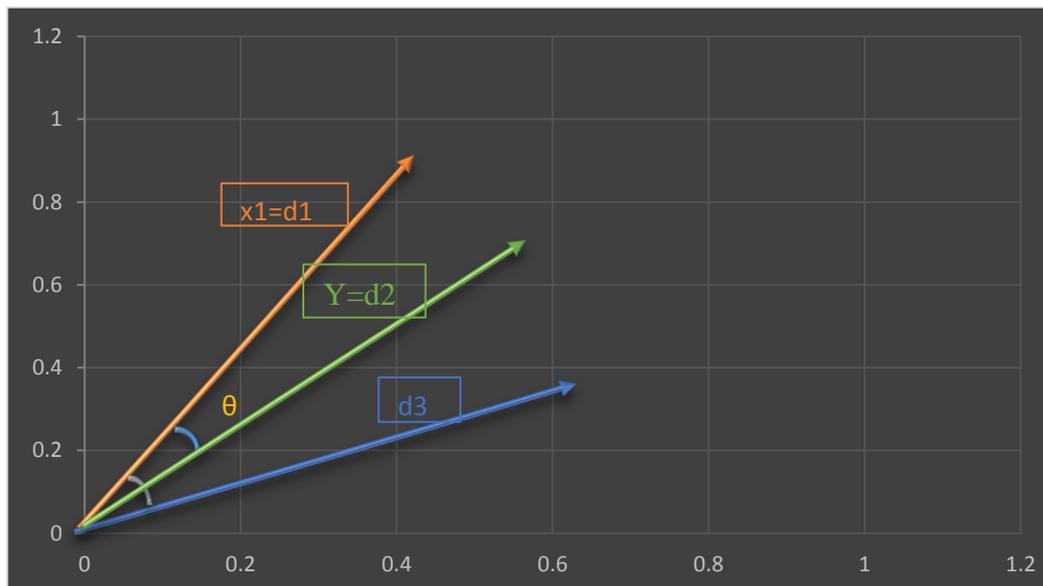


Figure 10 cosine similarity between vectors

1.6.1.1.2. Principle

[25] Cosine similarity is a measure of similarity that can be used to compare documents or, say, give a ranking of documents with respect to a given vector of query words. Let x and y be two vectors for comparison see (**Figure 10**) above. Using the cosine measure as a

$$\text{sim}(x, y) = \frac{x \cdot y}{\|x\| \|y\|} \quad (5.1)$$

where $\|x\|$ is the Euclidean norm of vector

$$x = (x_1, x_2, \dots, x_p) \quad (5.2)$$

defined as

$$\sqrt{x_1^2 + x_2^2 + \dots + x_n^2} \quad (5.3)$$

Conceptually, it is the length of the vector. Similarly, $\|y\|$ is the Euclidean norm of vector y . The measure computes the cosine of the angle (θ) between vectors x and y .

1.6.1.1.3. All the cases

- ✓ A cosine value of 0 means that the two vectors are at 90 degrees to each other (orthogonal) and have no match.
- ✓ The closer the cosine value to 1, the smaller the angle and the greater the match between vectors.

1.6.1.2. Euclidean distance

[26] The Euclidean distance between $X = (x_1 \dots x_d)$ and $Y = (y_1 \dots y_d)$ is defined as follows:

$$\text{distance}(X, Y) = \sqrt{\sum_{i=1}^d (x_i - y_i)^2} \quad (6.1)$$

It would seem at first sight that one should simply use the Euclidean distances to compute distances between pairs of points, since text is a special case of the multidimensional representation. However, the Euclidean distance is not good in computing distances in multidimensional representations that are very sparse and the number of zero values vary significantly over different points. This occurs frequently in the case of text because of the varying lengths of different documents.

1.6.1.3. Jaccard similarity

[26] *Let S_x and S_y be the set of words in a pair of documents that are represented in boolean form. Then, the Jaccard similarity is defined as follows:*

$$\text{Jaccard}(S_x, S_y) = \frac{|S_x \cap S_y|}{|S_x \cup S_y|} \quad (7.1)$$

$$= \frac{\text{Common terms in } S_x \text{ and } S_y}{\text{Distinct terms in union of } S_x \text{ and } S_y} \quad (7.2)$$

The Jaccard coefficient always lies in the range (0, 1) just like the cosine coefficient. It is also possible to define the Jaccard coefficient for the case where the documents

$X = (x_1 \dots x_d)$ and $Y = (y_1 \dots y_d)$ are represented in tf-idf form:

$$Jaccard(X, Y) = \frac{\sum_{i=1}^d x_i \cdot y_i}{\sum_{i=1}^d x_i^2 + \sum_{i=1}^d y_i^2 - \sum_{i=1}^d x_i \cdot y_i} \quad (7.3)$$

The Jaccard coefficient is especially useful for the case where the boolean representation of text is used. For the tf-idf representation, it is more common to use the cosine measure.

1.7. Conclusion

In this chapter, we presented text classification using machine learning techniques and some very known similarity measures.

In the next chapter we will introduce the SOA architecture and its concepts, as well as a detailed explanation about it and design patterns and their types.

Chapter 2: Service Oriented Architecture

Introduction

Service Oriented Architecture (SOA) is an architectural style increasingly adopted because it offers system architects a high-level solution to software design. SOA systems are based upon loosely coupled, autonomous and reusable coarse-grained components called services. Each service provides a domain specific behavior, and services can be composed as composite to fulfill high level business processes requirements. Various technologies have emerged to implement this style, among them, Web Services and SCA. Google, Amazon, Microsoft are well-known businesses that have successfully based their information systems on SOA.

2.1. Definition

[27]SOA is not a concrete architecture: it is something that leads to a concrete architecture. You might call it a style, paradigm, concept, perspective, philosophy, or representation. That is, SOA is not a concrete tool or framework you can purchase. It is an approach, a way of thinking, a value system that leads to certain concrete decisions when designing a concrete software architecture.

2.1.1. SOA is built with services

[28]This is the “service” part of “service-oriented architecture” It’s not an SOA if it’s not an architecture, or if that architecture doesn’t use the service as the common unit of measure (just as an object-oriented system uses the object or class as its common unit).

2.1.1.1. Service

[29]a service is a self-contained, reusable, and well-defined piece of business functionality encapsulated in code.

Services are indeed the holy grail of SOA. If properly designed, publicized, and self-describing, services become assets that can be widely reused in a variety of applications. This maximizes the investment in these assets and enables creation of a more agile enterprise since every business process doesn't have to be re-created from scratch.

2.1.1.2. Software Architecture

[30]*Definition of Software Architecture A software architecture is a set of statements that describe software components and assigns the functionality of the system to these components. It describes the technical structure, constraints, and characteristics of the components and the interfaces between them. The architecture is the blueprint for the system and therefore the implicit high-level plan for its construction.*

2.1.2. Service orientation

a design paradigm intended for the creation of solution logic units that are individually shaped so that they can be collectively and repeatedly utilized in support of the realization of the specific strategic goals and benefits associated with SOA and service oriented computing[31].

2.2. Service-orientation design paradigm

is comprised of a set of design principles that are applied together to achieve the goals of service-oriented computing[31].

2.2.1. Principles of Service-Orientation

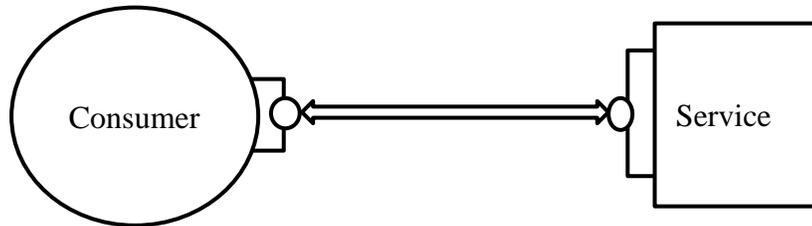
[31]*There are eight distinct design principles that are part of the service-orientation design paradigm. Each addresses a key aspect of service design by ensuring that specific design characteristics are consistently realized within every service. service-orientation design principles shape solution logic into something we can legitimately refer to as “service-oriented.” Below are the eight service-orientation design principles together with their official definitions:*

2.2.1.1. Standardized Service Contract

Services within the same service inventory are in compliance with the same contract design standards.

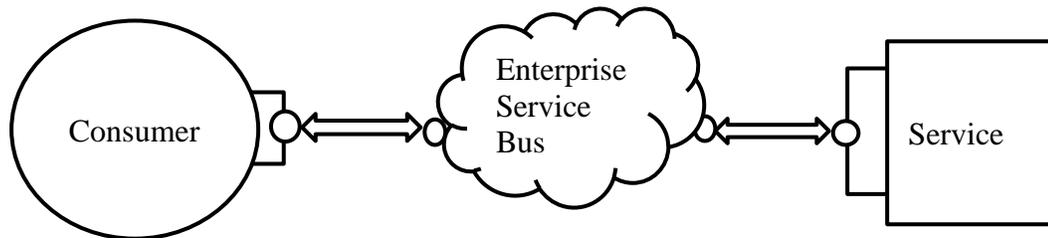
2.2.1.2. Service Loose Coupling

Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment.



Tightly Coupled
(Consumer interact directly with a service)

Figure 11 tight coupling



Loosely Coupled
(consumer and service interact via messaging and the ESB)

Figure 12 loose coupling

2.2.1.3. Service Abstraction

Service contracts only contain essential information and information about services is limited to what is published in service contracts.

2.2.1.4. Service Reusability

Services contain and express agnostic logic and can be positioned as reusable enterprise resources.

2.2.1.5. Service Autonomy

Services exercise a high level of control over their underlying runtime execution environment.

2.2.1.6. Service Statelessness

Services minimize resource consumption by deferring the management of state information when necessary.

2.2.1.7. Service Discoverability

Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.

2.2.1.8. Service Composability

Services are effective composition participants, regardless of the size and complexity of the composition.

2.3. Design patterns

Here in this section we are going to get through some of the important things about patterns[31]

2.3.1. Definition

The simplest way to describe a pattern is that it provides a proven solution to a common problem individually documented in a consistent format and usually as part of a larger collection.

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice".

2.3.2. Why design patterns?

- *represent field-tested solutions to common design problems.*
- *organize design intelligence into a standardized and easily “referencable” format.*
- *are generally repeatable by most IT professionals involved with design.*
- *can be used to ensure consistency in how systems are designed and built.*
- *can become the basis for design standards.*
- *are usually flexible and optional (and openly document the impacts of their application and even suggest alternative approaches).*
- *can be used as educational aids by documenting specific aspects of system design (regardless of whether they are applied)*
- *can sometimes be applied prior and subsequent to the implementation of a system*
- *can be supported via the application of other design patterns that are part of the same collection.*
- *Furthermore, because the solutions provided by design patterns are proven, their consistent application tends to naturally improve the quality of system designs.*

2.3.3. Categories of SOA design patterns

2.3.3.1. Service inventory

Service inventory” is a term used to represent a collection on independently standardized and governed services. Design patterns associated with the design of the service inventory technology architecture.

2.3.3.2. Service Design

a set of patterns specific to the design of services and service architecture.

2.3.3.3. Service Composition

Service composition design and runtime interaction are addressed by these patterns.

2.3.4. Design pattern catalog

A design pattern catalog is a collection of related design patterns documented together.

2.3.5. Pattern Profile

Each of the patterns is described using the same profile format and structure based on the following parts:

2.3.5.1. Pattern name

the pattern name is a handle we can use to describe a design problem, its solutions, and consequences in a word or two. Naming a pattern immediately increases our design vocabulary. It lets us design at a higher level of abstraction. Having a vocabulary for patterns lets us talk about them with our colleagues, in our documentation, and even to ourselves.

2.3.5.2. Problem

The issue causing a problem and the effects of the problem are described in this section, typically accompanied by a figure that further illustrates the “problem state.” It is this problem for which the pattern provides a solution. Problem descriptions may also include common circumstances that can lead to the problem (also known as “forces”).

2.3.5.3. Solution

This represents the design solution proposed by the pattern to solve the problem and fulfill the requirement. Often the solution is a short statement followed by a diagram that concisely communicates the final solution state. “How-to” details are not provided in this section but are instead located in the Application section.

2.3.5.4. Application

This part is dedicated to describing how the pattern can be applied. It can include guidelines, implementation details, and sometimes even a suggested process.

2.3.5.5. Impacts

Most patterns come with trade-offs. This section highlights common consequences, costs, and requirements associated with the application of a pattern. Note that these consequences are common but not necessarily predictable. For example, issues related to typical performance requirements are often raised; however, these issues may not impact an environment with an already highly scalable infrastructure.

2.4. Design Patterns and Design Principles

Specifically, the relationship between service-orientation design principles and patterns can be defined as follows:

- *Design principles are applied collectively to solution logic in order to shape it in such a manner that it fosters key design characteristics that support the strategic goals associated with service-oriented computing.*
- *Design patterns provide solutions to common problems encountered when applying design principles—and—when establishing an environment suitable for implementing logic designed in accordance with service-orientation principles.*

2.4.1. Measures of Application

It is important to acknowledge that most patterns do not propose a black or white design option. Design patterns can often be applied at different levels. Although the effectiveness of a given pattern will generally be equivalent to the extent to which it is realized, there may be practical considerations that simply limit the degree to which a pattern can be applied in the real world.

This consideration affects both design patterns and design principles. For example, individual service-orientation design principles can rarely be applied to their maximum

potential. The point is to pursue the design goals of a design pattern or principle to whatever extent feasible and to strive for an end-result that realizes the pattern or principle to a meaningful extent.

2.5. Conclusion

*In this chapter we have presented the soa architecture and design patterns
In the next chapter we will introduce our system, its main components as well as a detailed explanation about each phase of the proposed method and the steps followed to reach our goal.*

Chapter 3: Pattern selection

Introduction

design patterns are solutions to programming problems that automatically implement good design techniques. Someone has already faced the issues you're facing, solved them, and is willing to show you what the best techniques are. All without a lot of memorization on your part; all you have to do is recognize which design pattern fits which situation and lock it into place. However, the existence of a large number of design patterns makes the selection of a fit design pattern for a given design problem a difficult task to the experienced developer, and makes it a challenging task for the inexperienced one who is not familiar with design patterns. To overcome this difficulty, a supporting tool that automatically suggests to the developer the right design pattern for a given design problem during the design phase becomes a necessity.

3.1. Our design pattern selection approach

*we propose a two-phase method to select a right SOA design pattern. The proposed method is based on a:
text classification approach that aims to show an appropriate way to suggest the right design pattern(s) to developers for solving each given design problem.*

3.1.1. First phase

In this first phase we are going to predict only the class of the problem through some machine learning technique.

3.1.1.1. Dataset

*the input of our proposed method is the **problem definitions** of some SOA design **patterns** and their classes.*

only **Problem Domain** of each **design pattern** document including **problem**, and **Applicability** (called *Problem Definition*) sections is used to select automatically the right design pattern.

Our data set consists of all the Soa design patterns profiles and from these we need only the problem definition of each design pattern and the corresponding class.

See(**Table 3**)below.

Problem definition	Pattern class
(Application + problem) _i	The class _i
⋮	⋮
(Application + problem) _n	The class _n

Table 3 SOA patterns dataset

Our proposed approach is based on analyzing the corpus of pattern descriptions, then transferring each one of them (we mean here the problem definition) into a vector of features. The next step is vectorizing the classes as well, because we need to deal with numbers instead of text data.

3.1.1.2. Preprocessing

Each design pattern description and each design problem scenario are pre-processed through three activities which are: Tokenization then Noise Removal then Stemming as depicted by (**Figure 13**) below . Tokenization process splits each document at the delimiters into unigrams (tokens). Afterwards, all the tokens are transferred into the lower case such that words like “Object”, “object” and “OBJECT” are treated the same. Noise removal stage disregards the non-descriptive words like linking verbs and pronouns. These non-descriptive words are considered noise as they increase the size Finally, the words are normalized to their root forms through Stemming. For example, a stemmer can reduce each of the words “creating” and “created” to the word “create”. See (Document Pre-Processing &Representation in page 2).

Pattern selection

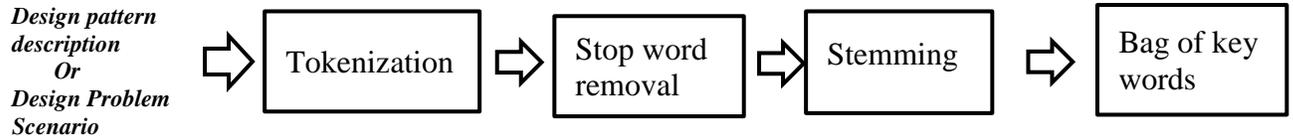


Figure 13 Preprocessing stages of each design pattern description and problem scenario

3.1.1.3. VSM (Vector Space Model)

In this stage each pattern (DP) is represented as a vector of unigrams. All the vectors have the same size which is equal to the number of unique words in the corpus of pattern descriptions see (Constructing a Vector Space Model in p 7).

To build the Unigram VSM:

3.1.1.3.1. Indexing

After the preprocessing of the pattern’s corpus all the unique words(also called vocabulary) are collected as a bag of words(see Bag of words in page 8) and each word is given an index see (Figure 14) below.

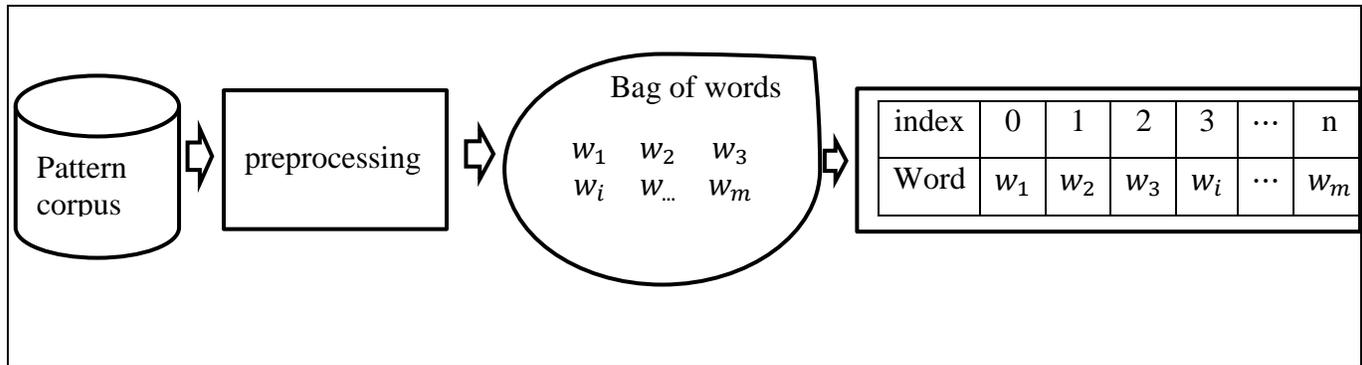


Figure 14 indexing

And then we give each class of the patterns a number to be represented in a numeric format. Since we have 3 types of classes that’s mentioned in (Categories of SOA design patterns page number 34) so a number is given to each of them. See (Figure 15) below.

<i>Class of design patterns</i>	<i>Corresponding numbers</i>
<i>Service inventory</i>	<i>0</i>
<i>Service Composition</i>	<i>1</i>
<i>Service Design</i>	<i>2</i>

Figure 15 numerizing the pattern classes

3.1.1.3.2. Vectorizing

This vectorizing processes is applied to both the design problem of the user and to each design pattern of the corpus using the vocabulary and applying the rules in (Figure 16)below.

The pattern vector will have the weight in the cells that correspond to each word of the vocabulary it's to mean that the cell that has the same index as the word will have the weight of the word in that pattern see(Feature weighting in page 6). for instance the word w_1 that exist in the index 0 has the score (w_1) in the vector at the same index 0.

see(Figure 17) below .

the same thing applies to the user problem so it will be vectorized using that same vocabulary see(Figure 18) below where each cell will contain the weight of the matching word in the vocabulary.

Here we used term frequency as weighting method($weight(w)=tf(w)$).

$$\{index_{vocabulary}(w_i) == index_{vector}(score(w_i)) \} \quad 1 \leq i \leq m$$

$$\{index_{user\ problem}(w_i) == index_{vector}(score(w_i)) \} \quad 1 \leq i \leq m$$

Figure 16 indexing Rules

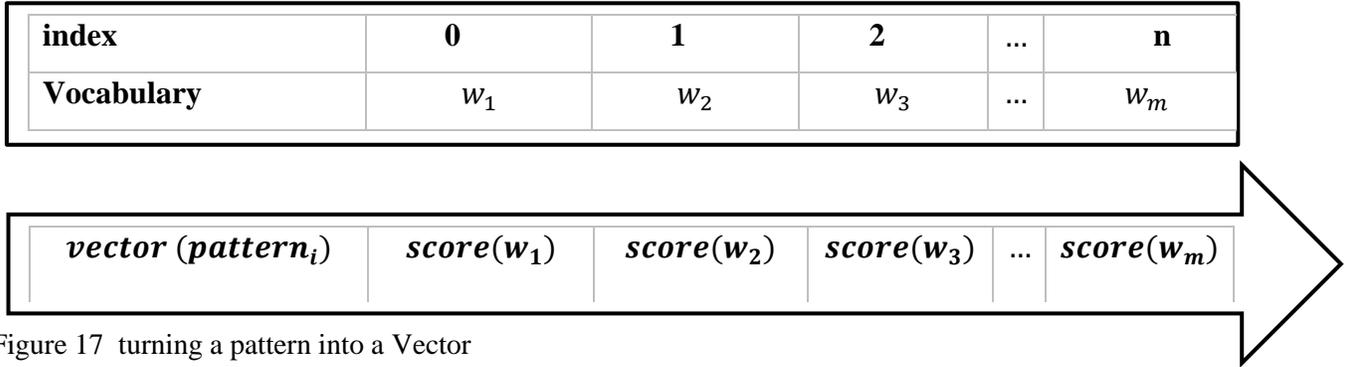


Figure 17 turning a pattern into a Vector

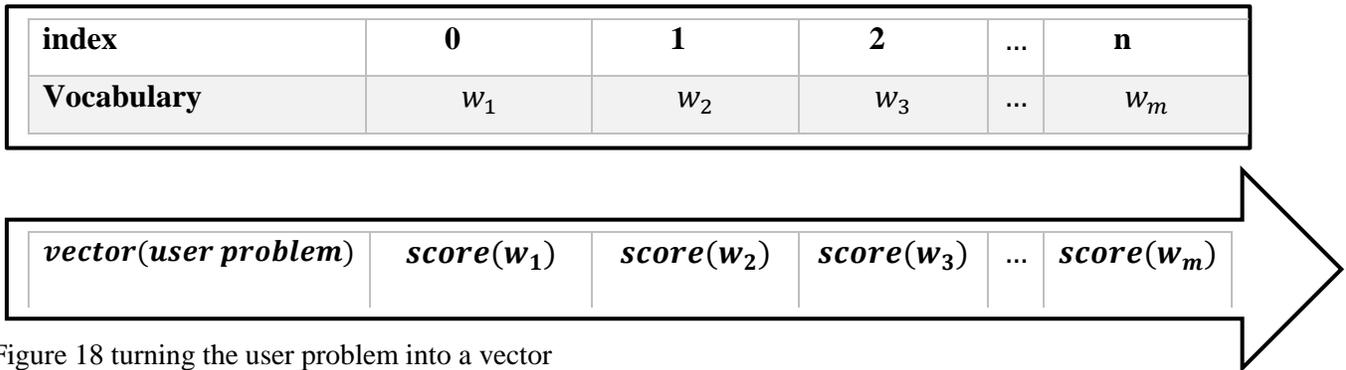


Figure 18 turning the user problem into a vector

The **Equation (8.1)** of design patterns in a space of m unique words(vocabulary) where each row corresponds to a pattern vector dP_i and each cell corresponds to the weight of a term t_j from the vocabulary.

$$VSM \text{ for patterns} = \begin{pmatrix} D/T & t_1 & t_j & \dots & t_m \\ dP_1 & w_{11} & w_{1j} & \dots & w_{1m} \\ dP_i & w_{i1} & w_{ij} & \dots & w_{im} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ dP_n & w_{n1} & w_{nj} & \dots & w_{nm} \end{pmatrix} \quad (8.1)$$

Below the **Equation (8.2)** for the user problem in the vsm (vector space model) where the row represents the user problem vector and the cells the weight w_j of each word t_j from the vocabulary in the user problem.

$$\text{user problem vector} = \begin{pmatrix} \text{problem/terms} & t_1 & t_j & \dots & t_m \\ \text{user problem} & w_1 & w_j & \dots & w_m \end{pmatrix} \quad (8.2)$$

Now both the user problem and the pattern corpus are represented in the vector space model.

3.1.1.4. Learning the model

Using machine learning techniques for text classification see (Text-classification in page 1)and (Machine learning algorithms in page 11).what we are aiming here is to classify a design pattern problem using one of these techniques (naïve Bayesian,svm,knn and decision tree).

we have our dataset that's the vector space model of the corpus of design patterns see Equation (8.1),And the class of each design pattern that has been vectorized as mentioned in (Figure 15) above, so our dataset will be as in Equation (8.3).

$$\text{dataset in the vsm} = \begin{pmatrix} \text{class} \\ C_1 \\ C_i \\ \vdots \\ C_n \end{pmatrix} \begin{pmatrix} DP/T & t_1 & t_j & \dots & t_m \\ dP_1 & w_{11} & w_{1j} & \dots & w_{1m} \\ dP_i & w_{i1} & w_{ij} & \dots & w_{im} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ dP_n & w_{n1} & w_{nj} & \dots & w_{nm} \end{pmatrix} \quad (8.3)$$

Before we feed our dataset to the learning model we have to split it up into a training and testing dataset the first set is for training the model and the second for testing the efficiency of the trained model see (Figure 19) below.

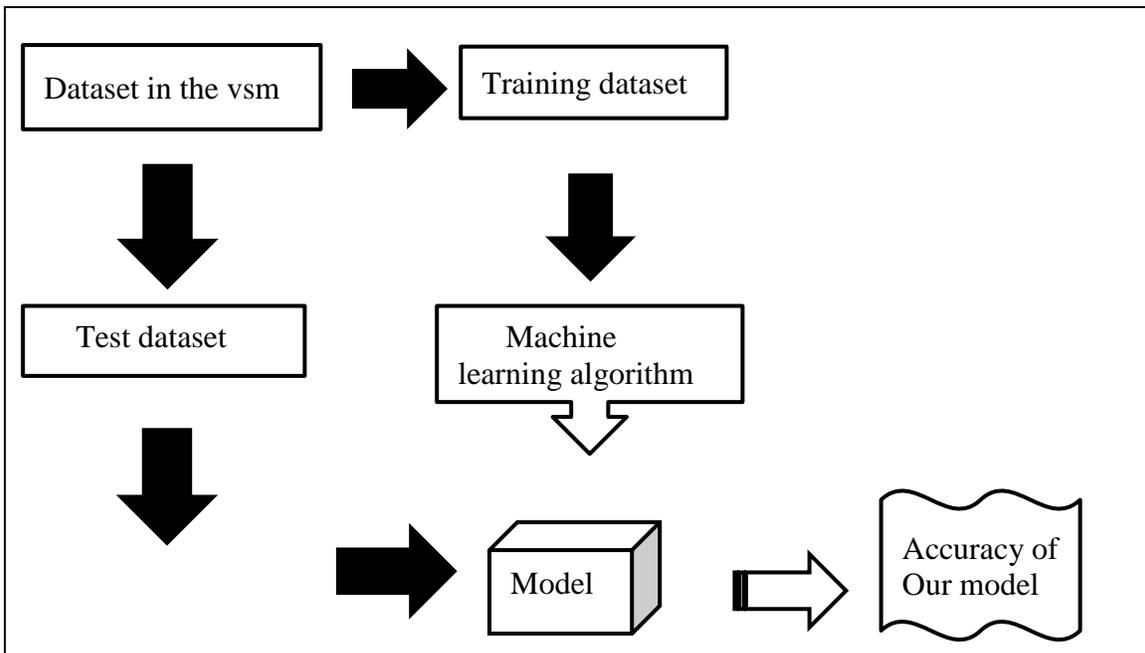


Figure 19 training and testing the model

3.1.1.5. Prediction

After learning the model, we are going to use it to predict the class of patterns that the user problem belongs to see (Figure 20) below.

The user problem is vectorized as we have mentioned before in (Vectorizing in page 41)

It's to mean that it is represented in the vector space model as vector see **Equation (8.2)**.

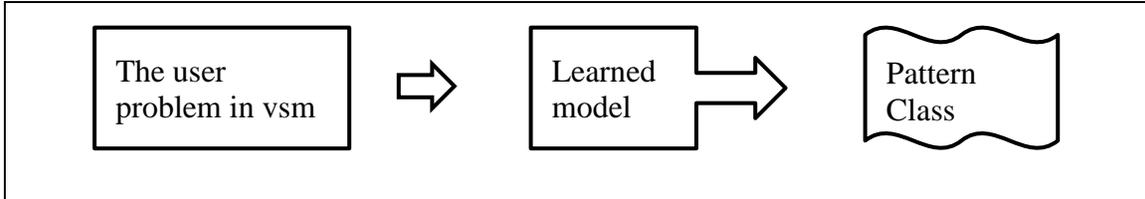


Figure 20 predicting the problem class

After knowing the class of patterns that the user problem belongs to. now we have to get the specific pattern label so we can retrieve the solution (design pattern) to that problem, for that we use cosine similarity.

3.1.2. Second phase

3.1.2.1. Dataset

Our dataset in that second phase will contains only the patterns that belong to the class that we have predicted before and their labels see (**Figure 21**)below.

Class	Pattern problem	Label
Predicted class	(Application + problem) ₁	label _i
Predicted class	⋮	⋮
Predicted class	(Application + problem) _n	label _n

Figure 21 dataset for second phase

First we are going to use the vsm to represent our dataset as mentioned in(VSM (Vector Space Model) page 40) but instead of the pattern class here we have the pattern labels .

Since we are dealing with numbers the labels as well are going to be indexed by numbers as mentioned in (**Figure 22**) below.

Pattern label	Index
l_1	0
l_2	1
\vdots	\vdots
l_n	n-1

Figure 22 indexing labels

So, in the end our dataset is going to be represented in the vector space model as in **Equation (8.4)** where $(t_1 \ t_j \ \dots \ t_m)$ are the patterns vocabulary and w_{ij} is the weight of the word j in the pattern i . as weighting method we are going to use *tf-idf* see (TF-IDF weighting in page 7).

$$\text{dataset in the vsm} = \begin{pmatrix} \text{labels} \\ l_1 \\ l_i \\ \vdots \\ l_n \end{pmatrix} \begin{pmatrix} \frac{D}{T} & t_1 & t_j & \dots & t_m \\ dP_1 & w_{11} & w_{1j} & \dots & w_{1m} \\ dP_i & w_{i1} & w_{ij} & \dots & w_{im} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ dP_n & w_{n1} & w_{nj} & \dots & w_{nm} \end{pmatrix} \quad (8.4)$$

the user problem also will be represented in the vector space model using that same vocabulary used in the data set vector space model as in **Equation (8.5)**.

$$\text{user problem vector} = \begin{pmatrix} \text{problem/terms} & t_1 & t_j & \dots & t_m \\ \text{user problem} & w_1 & w_j & \dots & w_m \end{pmatrix} \quad (8.5)$$

Here we are going to use cosine similarity as mentioned in (Cosine-Based Similarity page 25). As in(**Figure 23**)below, we have our dataset represented in the vector space model as

$$\begin{pmatrix} dp_1 \\ \vdots \\ dp_i \\ dp_n \end{pmatrix} \text{ along with the user problem that's } (p) \text{ and now the cosine similarity is calculated}$$

using **Equation (8.6)**.

$$\begin{aligned} \text{sim}(dp_i, p) &= \frac{dp_i \cdot p}{\|dp_i\| \|p\|} \\ &= \cos \theta_i \end{aligned} \quad (8.6)$$

That's for each and every design pattern vector dp_i with the vector of the user problem p and the vector with the least difference is going to have the smallest cosine it's to mean it's more similar to the user problem than the others so that pattern will be retrieved as the appropriate solution to the user problem.

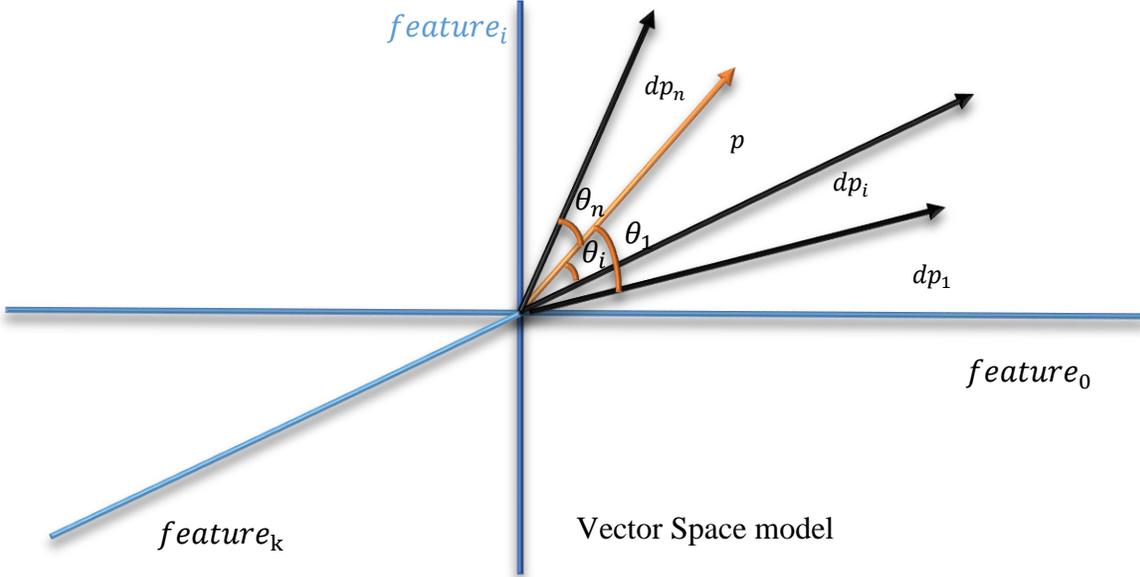


Figure 23 cosine similarity measure

All these are demonstrated in the figure (Figure 24) below

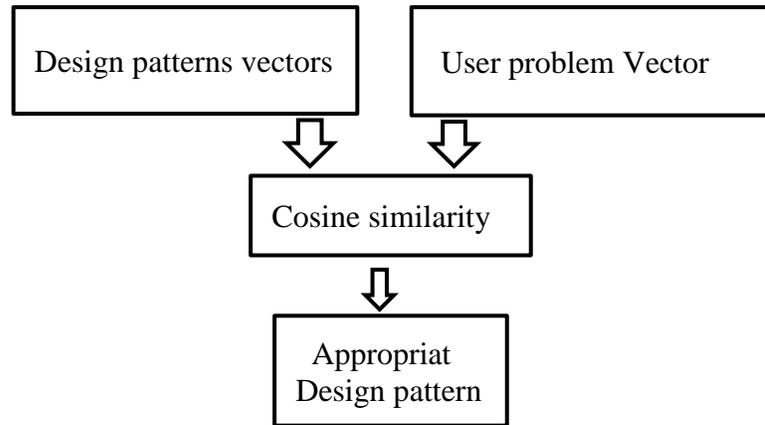


Figure 24 getting the appropriate design pattern

3.1.3. The whole processes

In the end the whole process of selecting the right design pattern is demonstrated in the following diagram.

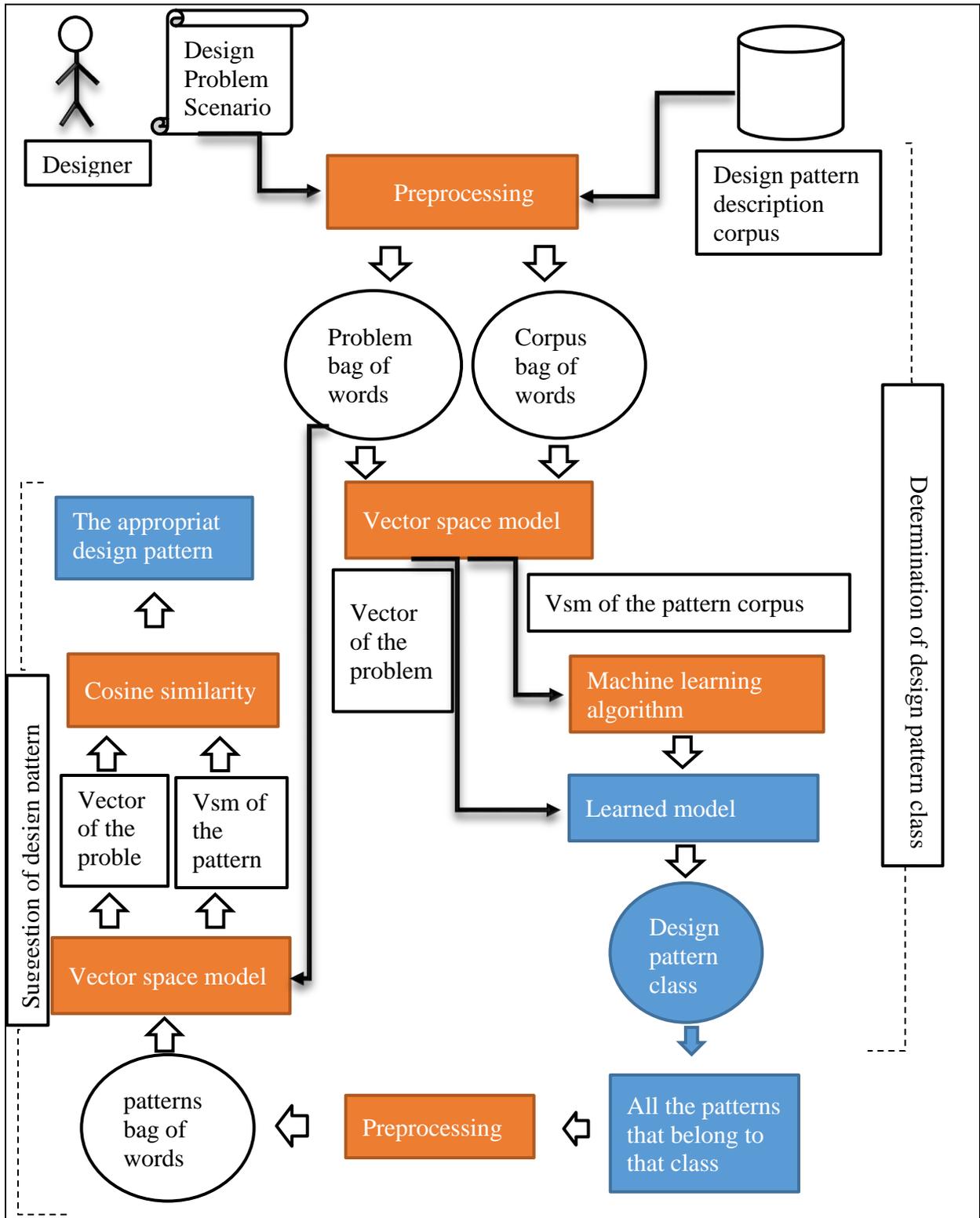


Figure 25 the processes of selecting design pattern

3.2. Conclusion

In this chapter we have presented the proposed method in details which means the way to predict the appropriate design pattern by classifying the user problem using learning techniques and cosine similarity and we have talked briefly about each of the phases and have also explained all the steps followed to achieve our objective. In the next chapter, we will see all the implementation details then the evaluation and results which are briefly discussed and the interfaces of our system.

Chapter 4: Implementation, results and comparison

Introduction

This chapter is structured in four main parts. In the first part, we will present the programming language Python, the libraries, and the dataset in the second part then in the third part we are going to show the results and compare the accuracy of the 4 models and test some real world examples that is in experimentation part, and Next, we are going to show the execution of the application by showing some snapshots of the interface during this process.

4.1. Implementation

The implementation of proposed models is constructed based on using Python programming language working on visual studio code platform. Python has a large number of scientific libraries for data processing and machine learning approaches.

4.1.1. Visual studio code platform

Visual Studio Code combines the simplicity of a code editor with what developers need for their core edit-build-debug cycle. It provides comprehensive code editing, navigation, and understanding support along with lightweight debugging, a rich extensibility model, and lightweight integration with existing tools.

4.1.2. Scientific Python



One of the advantages of implementing with Python is that the simplicity of Python is matched with the speed of compiled programming languages. Therefore, highly complex algorithms can be applied in a very short period of time.

4.1.2.1. Scikit-learn

Is a free software machine learning library for the python programming language.

One of the features that contributes to the popularity of Scikit-learn is that a uniform interface is available for all algorithm classes. All classification algorithms have the functions: fit and predict. The fit function trains the algorithm and predict executes the predictions after training.

4.1.2.2. Nltk



NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming...

4.1.3. Bulma



Bulma is a very popular Beautiful, free and open source, lightweight and stylish CSS framework made by Jeremy Thomas.

that allow developers to quickly build web interfaces easy. The modern design and layout features Bulma offer were the main reasons we chose to use it.

4.1.4. Electron



The electron as a framework is used for creating desktop applications in collaboration with other popular technologies such as JavaScript and CSS. developed and maintained by GitHub. It does the dirty work so you can focus on the significant areas of your application.

4.1.5. SQLite



SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day.

4.1.6. Python-shell

You can use python-shell to communicate between Python and Node.js/Electron. python-shell provides an easy way to run Python scripts from Node.js with basic and efficient inter-process communication and better error handling.

4.2. Dataset

I used an SQLite database that I made of the 78 soa pattern profiles, these profiles in the database are distributed over 3 genres.

Each event in the database contains at least the following informations: pattern class, problem, solution and lable. (Table 4) below shows an example of the raw content of an event from the database.

<i>Pattern component</i>	<i>Content</i>
<i>Id</i>	56

<i>Pattern class</i>	<i>Service composition</i>
<i>Problem</i>	<i>The intermediary processing layers generally required by service compositions can expose sensitive data when security is limited to point-to-point protocols, such as those used with transport layer security. A digital signature algorithm is applied to the message to provide “proof of origin,” allowing sensitive message contents to be protected from tampering. This technology must be supported by both consumer and service.</i>
<i>Solution</i>	<i>A message can be digitally signed so that the recipient services can verify that it originated from the expected consumer and that it has not been tampered with during transit.</i>
<i>Label</i>	<i>Data Origin Authentication</i>

Table 4 raw content

4.3. Results and comparison

*In this section, we will first discuss how the results are presented by explaining the performance measures. After that, the final results in(**Table 7**) below will be explained in (Section 4.3.2.).*

4.3.1. Performance measures

There are various methods to determine effectiveness; however, precision, recall, confusion matrix and accuracy are most often used. To determine these, one must first begin by understanding if the classification of a document was a true positive (TP), false positive (FP), true negative (TN), or false negative (FN).

<i>TP</i>	<i>Determined as a document being classified correctly as relating to a category.</i>
-----------	---

<i>FP</i>	<i>Determined as a document that is said to be related to the category incorrectly.</i>
<i>FN</i>	<i>Determined as a document that is not marked as related to a category but should be.</i>
<i>TN</i>	<i>Documents that should not be marked as being in a particular category and are not.</i>

Table 5 Classification of a document

4.3.1.1. Confusion Matrix

A confusion matrix can be used as a way to visualize the results of a classification algorithm. For the binary case where 1 and 0 is the two possible outcomes, the algorithm can be used to predict whether a test sample is either 0, or 1. As a way to measure how well the algorithm performs, we can count four different metrics, here 1 defined as positive and 0 defined as negative:

The confusion matrix is simply these four values visualized in one table

		Predicted class	
		0	1
Actual class	1	True positive	False positive
	0	False negative	True negative

Table 6 Confusion matrix

4.3.1.2. Precision and Recall

As a way to evaluate the performance of a machine learning algorithm, one can use precision and recall. Precision is defined as:

$$precision = \frac{TP}{TP + FP} \quad (9.1)$$

$$recall = \frac{TP}{TP + FN} \quad (9.2)$$

A very high precision means that the algorithm classifies almost no inputs as positive unless they are positive. A high recall would mean that the algorithm misses almost no positive values.

4.3.1.3. F-score

The *f*-score (or *F1*-Score) is the harmonic mean of precision and recall. The *f*-score is defined as:

$$F_1 = 2 \frac{precision \cdot recall}{precision + recall} \quad (9.3)$$

Classifier	Recall	Precision	F1-score
Naïve Bayesian			
Service Design	0.67	0.40	0.50
Service Composition	0.86	0.86	0.86
Service inventory	0.67	1.00	0.80
Decision tree			
Service Design	0.50	0.33	0.40
Service Composition	0.50	0.80	0.62
Service inventory	0.75	0.60	0.67
K-nearest neighbor			
Service Design	0.40	0.67	0.50
Service Composition	0.50	0.25	0.33
Service inventory	0.50	0.33	0.40
Support-vector-machine			
Service Design	0.41	0.33	0.37
Service Composition	0.51	0.47	0.49
Service inventory	0.54	0.37	0.44

Table 7 Results across classifiers

4.3.2. Classification results

*A comparison between the results obtained with models using Naïve Bayesian, SVM, Knn and Decision Tree is shown in (**Table 7**) above.*

Looking at the F-1 score of these models, we can observe that the F-1 yield by Service Design is =0.50 higher for the Naive Bayes and K-nearest neighbor.

while for the Service Composition the F-1 score is =0.86 for the Naive Bayes and equal to 0.62 for the decision tree classifier which are the best results for this category.

for Service inventory the F-1 score is 0.80 for Naive Bayes and 0.67 for Decision tree these are the biggest for service inventory.

4.3.3. Conclusion

From the comparison, Naïve Bayesian classifier produces the best results, significantly., the decision tree classifier produces good results but Support vector machine and knn give the worst results among them.

4.3.4. Four examples of real-world problems

*Here we have 4 real worlds example with the result of using each one of the four algorithms See (**Table 8**) below.*

<i>Real world Design problem</i>	<i>Algorithms</i>	<i>Naïve Bayesian</i>	<i>Knn</i>	<i>Svm</i>	<i>Decision tree</i>	<i>Real design pattern</i>
How can business rules be abstracted and centrally governed?		+	-	-	+	Rules Centralization
How can a service inventory overcome the limitations of its canonical protocol while still remaining standardized ?		+	-	+	+	Dual protocol
How can a large business problem be solved without having to build a standalone body of solution logic?		+	+	+	-	Functional Decomposition

Table 8 real world examples

4.4. Interfaces of our system

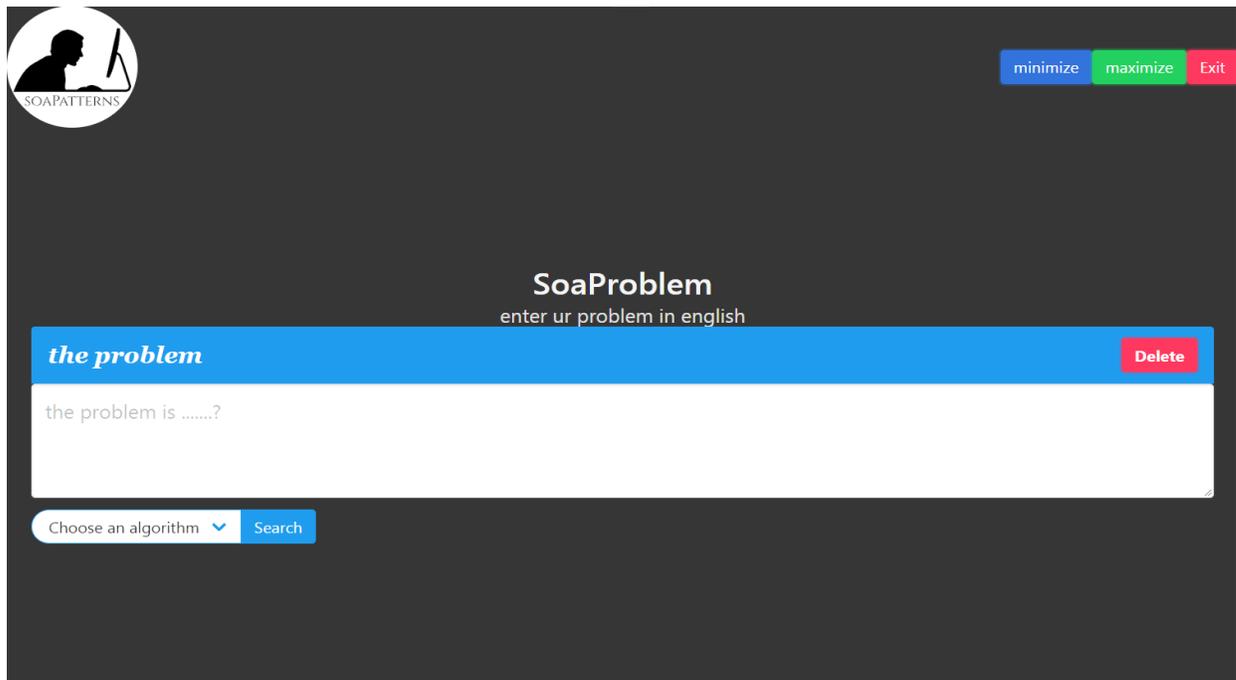


Figure 26 Application Home

We developed GUI (Graphical User Interface) to facilitate the use of the application.

The application contains text area, drop-down menu, buttons (search, delete, resizing and exit button).

The Text area is for the user to enter the problem and we have the delete button  to clear it just in case.

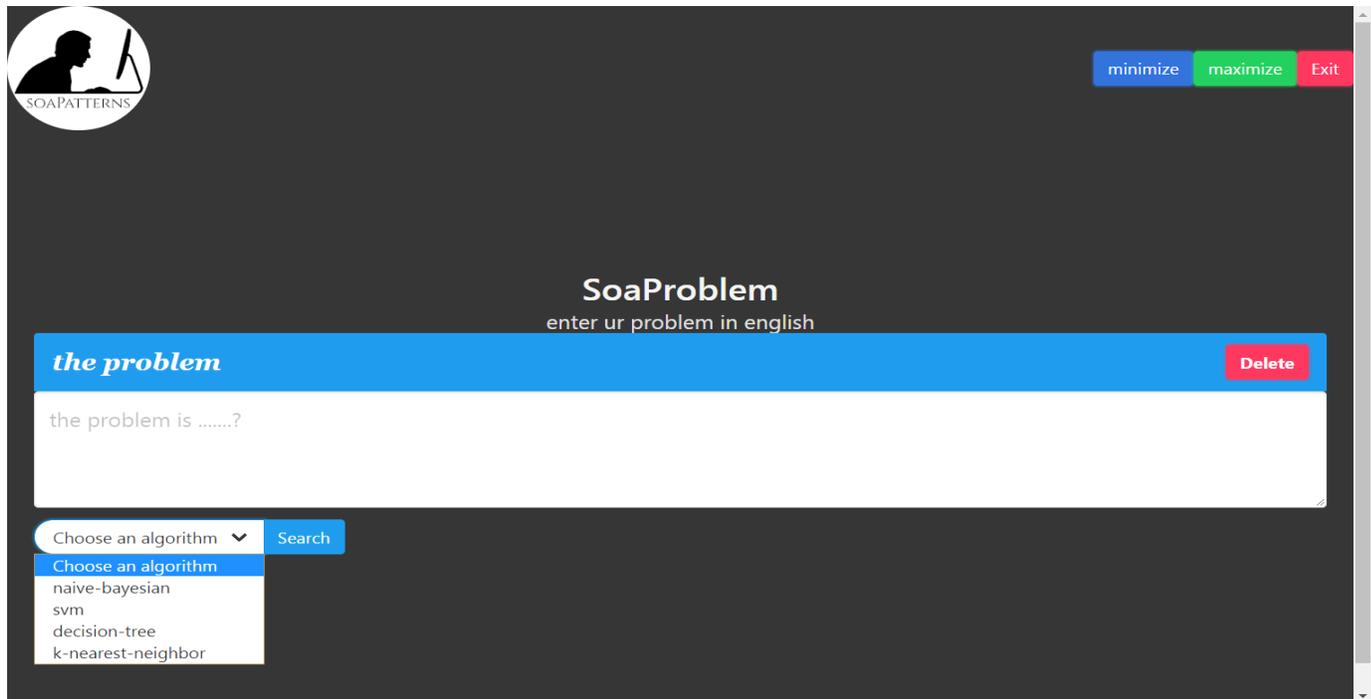
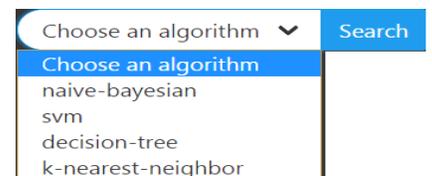


Figure 27 drop-down menu

4.4.1. drop-down menu

offers to the user four algorithms (naïve Bayesian, svm, decision tree, k-nearest-neighbor) to choose from. so the model will be created using the user choice see (Figure 27) above.



4.4.2. The search button

Once the user clicks the search button:

- ☒ If the text area is empty then a warning message box will be displayed see (**Figure 28**) below.

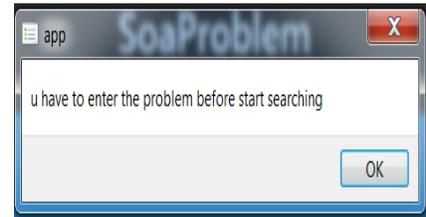


Figure 28 text area message warning

- ☒ If there no algorithm has been selected a warning message box will be displayed see. (**Figure 29**) below.

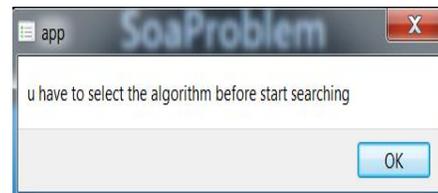


Figure 29 algorithm selection warning

- ✓ Otherwise

The text is ready and the algorithm is selected (Figure 30) below once the search button is clicked  the model will be created and the pattern will be displayed with the used algorithm and the accuracy of the model see (**Figure 31**) below.

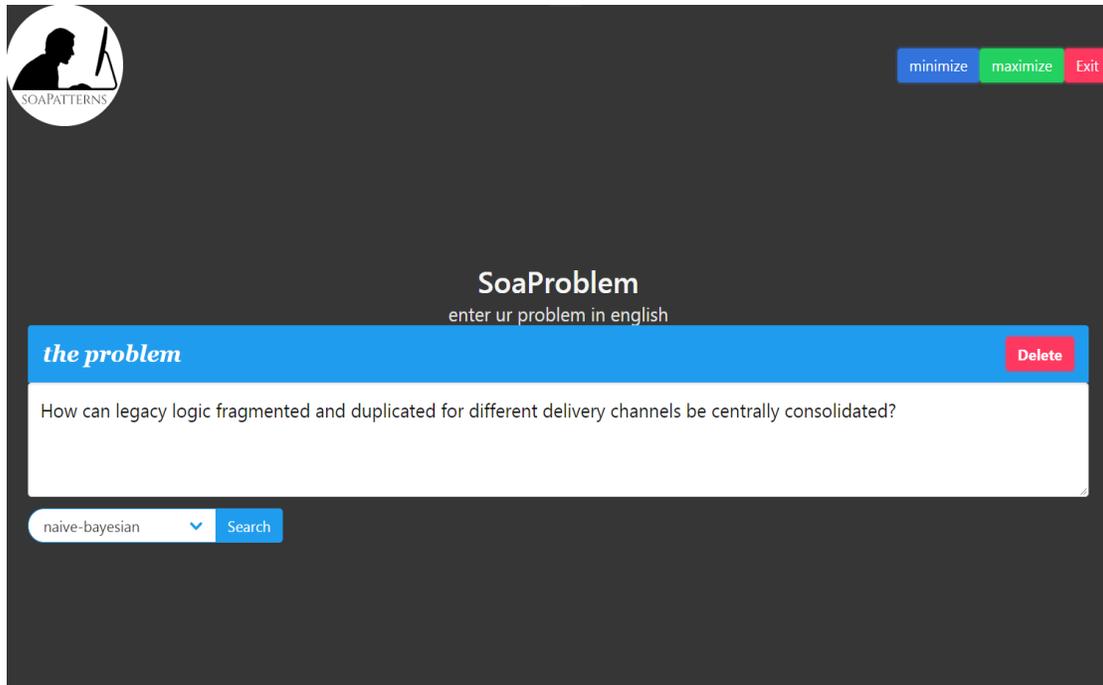


Figure 30 text and algorithm are set

4.4.2.1. Solution using naïve Bayesian

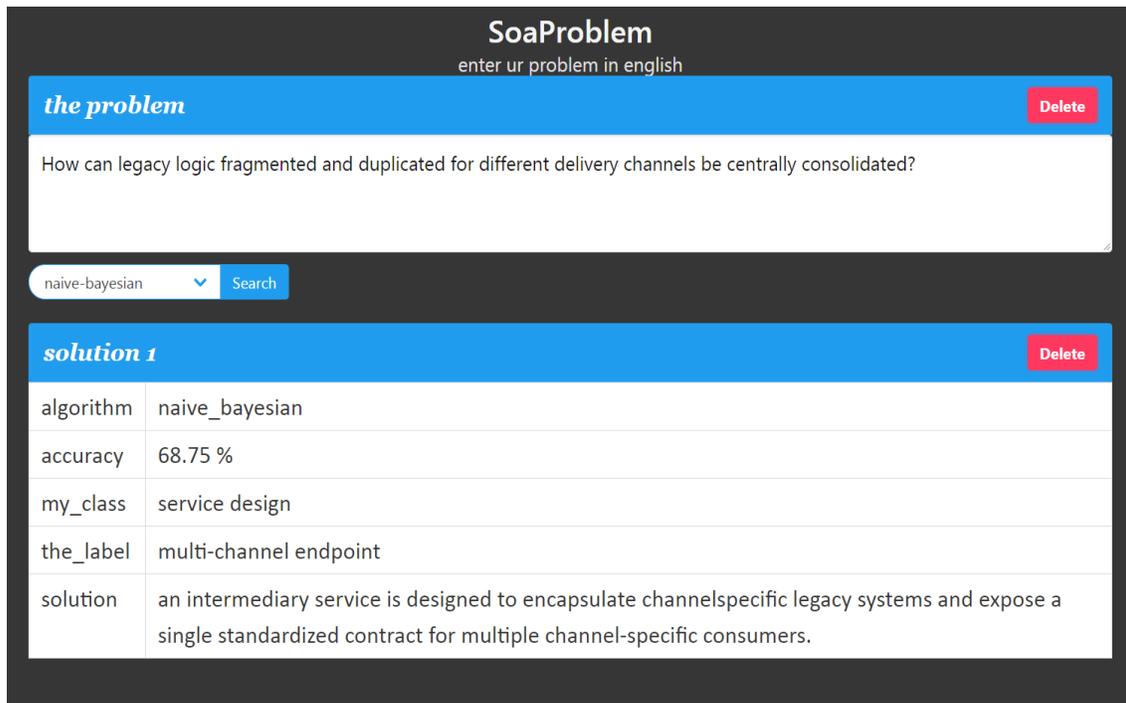


Figure 31 solution using naive bayesian

And we can use other algorithms like svm, knn and decision tree

4.4.2.2. Using decision tree

solution 2		Delete
algorithm	decision tree	
accuracy	62.5 %	
my_class	service design	
the_label	multi-channel endpoint	
solution	an intermediary service is designed to encapsulate channelspecific legacy systems and expose a single standardized contract for multiple channel-specific consumers.	

Figure 32 solution using decision tree

4.4.2.3. Solution using knn

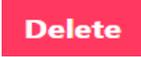
solution 3		Delete
algorithm	k nearest neighbor	
accuracy	56.25 %	
my_class	service design	
the_label	multi-channel endpoint	
solution	an intermediary service is designed to encapsulate channelspecific legacy systems and expose a single standardized contract for multiple channel-specific consumers.	

Figure 33 solution using knn

4.4.2.4. Using svm

solution 4		Delete
algorithm	svm	
accuracy	37.5 %	
my_class	service design	
the_label	multi-channel endpoint	
solution	an intermediary service is designed to encapsulate channelspecific legacy systems and expose a single standardized contract for multiple channel-specific consumers.	

Figure 34 solution using svm

There's this delete buttons  that appears with the solution and we can use it to delete it.

Conclusion and future work

In our work, we find the appropriate design pattern using several machine learning and natural language processing techniques. We collected SOA design patterns from Thomas Erl book [31], where there is a notable presence of design pattern profiles. After extracting structured data from the book content that's about patterns. We saved the retrieved data in a relational database that reflects the natural data structure presented in the profile of the design pattern. Then we used several database queries to retrieve information to build design pattern profile dataset. We extracted text features and we built 4 classifiers using this dataset.

Once all our experiments have been exposed and their results analyzed, and due mainly to the large variety of them, it is time to summarize key points found during this research, underlining major issues covered by present work. It is, of course, agreed that many open issues remain in the horizon, Text Categorization research is a richness area where several computer-based techniques meet to propose a wide variety of solutions. We believe that the success of a system resides not only on the excellence of proven learning algorithms, or specific natural language-based methods, but also on a closer observation of the matter under study: the collection of documents of SOA design pattern profile.

From our entire study, we observe that the standard precision and recall values of naïve Bayesian are better than KNN, SVM and decision tree so we conclude that naïve Bayesian works better for this type of classification due to the type of our dataset known that in that case our dataset is relatively small and the features are relatively independent.

In this dissertation, we presented SOA design pattern selection approach using learning techniques. We illustrated our approach through a pattern example. In order to reach the generality and the validity of our approach, we have applied it to more real design patterns examples within the "Service inventory design patterns" category, service design and "service composition design patterns" categories.

In real applications, problems are complex and their solutions can be represented by compound or composite patterns that require the combination and reuse of other design patterns. So, as future work, we are working on specifying pattern language or set of

Conclusion and future work

patterns that can work together to increase some quality measurement instead of just a pattern as a solution.

Bibliography

- [1] “Design patterns selection: An automatic two-phase method,” *J. Syst. Softw.*, vol. 85, no. 2, pp. 408–424, Feb. 2012.
- [2] C. Kanakslakshmi and M. chezian, “Analysis on Text Mining and Text Classification techniques,” in *Proceedings of the National Conference on Information and Image Processing*, 2015, vol. 1, pp. 132–135.
- [3] M. chezian and K. C., “Performance Evaluation of Machine Learning Techniques for Text Classification,” presented at the *Proceedings of the UGC Sponsored National Conference on Advanced Networking and Applications*, 2015, p. 53.
- [4] R. R.P, K. Juliet, and A. hana, “Text Classification for Student Data Set using Naive Bayes Classifier and KNN Classifier,” *International Journal of Computer Trends and Technology*, vol. 43, no. 1, pp. 8–9, Jan. 2017.
- [5] E. Ikonmakis, S. Kotsiantis, and V. Tampakas, “Text Classification Using Machine Learning Techniques,” *WSEAS transactions on computers*, vol. 4, no. 8, p. section2,section 3, Aug. 2005.
- [6] M. Karaca and S. BAYIR, “Examining the Impact of Feature Selection Methods on Text Classification,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, p. 381 |, Jan. 2017.
- [7] A. Basarkar, “DOCUMENT CLASSIFICATION USING MACHINE LEARNING,” Master’s Theses and Graduate Research, San Jose State University, SAN JOSE STATE ,pp.22, 2017.
- [8] J. Ababneh, O. Almanmomani, W. Hadi, N. El-Omari, and A. Alibrahim, “Vector Space Models to Classify Arabic Text,” *Int. J. Comput. Trends Technol. IJCTT*, vol. 7, p. 219 introduction, Feb. 2014.
- [9] M. Kumari, A. Jain, and A. Bhatia, “Synonyms Based Term Weighting Scheme: An Extension to TF.IDF,” *Procedia Comput. Sci.*, vol. 89, p. 556, Jan. 2016.
- [10] J. Kaur and J. Saini, “A Study of Text Classification Natural Language Processing Algorithms for Indian Languages,” *VNSGU J. Sci. Technol.*, vol. 4, no. 1, pp. 163–164, Jul. 2015.
- [11] R. A. Sinoara, J. Antunes, and S. O. Rezende, “Text mining and semantics: a systematic mapping study,” *Journal of the Brazilian Computer Society*, vol. 23, no. 1, p. 13, Jun. 2017.
- [12] R. Malviya and P. Jain, “A Novel Text Categorization Approach based on K-means and Support Vector Machine,” *International Journal of Computer Applications*, vol. 130, pp. 1–3, Nov. 2015.
- [13] M. R and S. R, “Machine learning algorithms for text-documents classification: A review,” *International Journal of Academic Research and Development*, vol. 3, no. 2, p. 384, Mar. 2018.
- [14] N. Rani, A. Sharma, and D. S. Pathak, “Text Classification Using Machine Learning Techniques: A Comparative Study,” *International Journal on Future Revolution in Computer Science & Communication Engineering*, vol. 4, no. 3, p. 553, Mar. 2018.
- [15] C. C. Aggarwal, *Data Classification: Algorithms and Applications*, 1st ed. Chapman & Hall/CRC, 2015.
- [16] P. Perner, Ed., *Machine Learning and Data Mining in Pattern Recognition: 10th International Conference, MLDM 2014, St. Petersburg, Russia, July 21-24, 2014, Proceedings*, vol. 8556. p. 516: Springer International Publishing, 2014.

- [17] R. Changala, A. Gummadi, G. Yedukondalu, and U. S. N. Raju, "Classification by Decision Tree Induction Algorithm to Learn Decision Trees from the class-Labeled Training Tuples," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. 4, pp. 427–428, Apr. 2012.
- [18] C. C. Aggarwal, *Data Mining: The Textbook*, 1st ed. New York USA: Chapman & Hall/CRC, 2014.
- [19] O. Ardhapure, L. S. Patil, D. L. Udani, and K. Jetha, "COMPARATIVE STUDY OF CLASSIFICATION ALGORITHM FOR TEXT BASED CATEGORIZATION," *International Journal of Research in Engineering and Technology*, vol. 5, no. 2, p. 218, Feb. 2016.
- [20] M. A. Wajeed and T. Adilakshmi, "Using KNN Algorithm for Text Categorization," in *Computational Intelligence and Information Technology*, 2011, p. 798.
- [21] S. Tong and D. Koller, "Support Vector Machine Active Learning with Applications to Text Classification," p. 47.
- [22] T. Joachims, "Text Categorization with Support Vector Machines," *Proc Eur. Conf Mach. Learn. ECML98*, pp. 3–4, Jan. 1998.
- [23] Y. Ahuja and S. K. Yadav, "Multiclass Classification and Support Vector Machine," *Global Journal of Computer Science and Technology Interdisciplinary*, vol. 12, no. 11, pp. 16–17, 2012.
- [24] G. Madzarov and D. Gjorgjevikj, "Multi-Class Classification Using Support Vector Machines In Decision Tree Architecture," presented at the IEEE EUROCON 2009, EUROCON 2009, 2009, pp. 414–415.
- [25] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. San Francisco, CA, USA, p77-78: Morgan Kaufmann Publishers Inc., 2011.
- [26] C. C. Aggarwal, *Machine Learning for Text*. Cham: Springer International Publishing, 2018.
- [27] N. M. Josuttis, *SOA in practice*, 1st ed. Beijing ; Sebastopol: O'Reilly, 2007pp12.
- [28] E. Hewitt, *Java SOA Cookbook*, 1 edition. Beijing ; Farnham ; Sebastopol, Calif,pp10: O'Reilly Media, 2009.
- [29] J. Davis, *Open source SOA*. Greenwich, Conn,pp 61-62: Manning, 2009.
- [30] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. pp 76: Prentice Hall PTR, 2004.
- [31] T. Erl, *SOA design patterns*. 2008.

Bibliography