E-Doctor: A Web Based SVMs for Automatic Identification of Health Problems

Vamsi Krishna P#1, Phani Kumar Damodara#2

#1 M.Tech Student, Dept of C.S.E, RISE Prakasam Group of Institutions, Ongole.
#2 Associate Professor, Dept of C.S.E, RISE Prakasam Group of Institutions, Ongole.

Abstract

This paper proposes E-doctor; a web-based application that makes automatically identifies the nature of illness by examining the symptoms about health problems. Here we are using a supervised learning model that is Support Vector Machines (SVMs), which analyze data and proceed to decisions, based on their knowledge. The role of system administrator is to define specific characteristics of each health problem and educate the SVM by entering sample files of statistical data.

After entering the sample files, medical staff can enter the patient’s reports information and w-doctor is automatically identify the nature of problem/prediction by means of answering if the patient has a specific health problem or may have in future. Here the statistical information plays an important role on deciding about a patient’s condition. The experiments are done by heart symptoms and Thyroid symptoms and the result was satisfactory.

Keywords: Automatic diagnosis, Medical, SVM

1. Introduction

In industries across the world, automated diagnostics becomes more and more prevalent, leveraging continually advancing algorithms that become increasingly intelligent in identifying solutions to known problems. Yet, in the health care industry, doctors have outdated and limited access to potential solutions and details from a patient’s case are seldom fully available to be investigated holistically [1]. There has been significant research on automated medical diagnosis, like work on [2]-[6]. However, most of them focus on research exclusively; so, there is limited practical application and integration of systems.

Driven by this context, the scope of this paper is to provide a web-based architecture for implementing automatic diagnosis techniques in a way transparent to the user. Several methods have been proposed as a base for an automatic diagnosis engine, like decision trees, machine learning, neural networks, Markov analysis e.t.c. For our case, we chose Support Vector Machines (SVMs) that find a wide range of applications in biomedicine these days, like analysis, prediction and classification of various diseases [7]. Through the proposed application, users have the ability to define a new disease, train the SVM using statistical data, and use it for automatic
diagnosis of this disease to patients. Note that, what we provide is the overall platform where users can implement techniques already tested. Accuracy of the final results is on the user’s responsibility.

The rest of the paper is organized as follows: Section 2 is a brief introduction to SVMs, along with a description of how these can be used for medical diagnosis. Section 3 describes the proposed design architecture. Section 4 is an example of a use case, where the process of automatic diagnosis is shown step by step. Finally, Section 5 discusses issues for future work.

2. Overview of SVMs

Support Vector Machines, also known as support vector classifiers, are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification. Support Vector Machines are based on recent advances in statistical learning theory. They use a hypothesis space of linear functions in a high-dimensional feature space, trained with a learning algorithm from optimization theory that implements a learning bias derived from statistical learning theory. The learning machine is given a training set of examples (or inputs), belonging to two classes, with associated labels (or output values).

Let vector \( x \in \mathbb{R}^n \) denote a pattern to be classified and let scalar \( y \) denote its class \( (y \in \{-1, 1\}) \). Also let \( \{ x_i, y_i \mid i = 1, \ldots, l \} \) denote a set of \( l \) training examples. The problem is how to construct a decision function \( f(x) \) that correctly classifies an input pattern that is not necessarily in the training set. The examples are in form of attribute vectors and the SVM finds the hyperplane separating the input data and being furthest from both convex hulls.

The original optimal hyperplane algorithm proposed was a linear classifier. However, in 1992, scientists suggested a way to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes. The resulting algorithm is formally similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. The transformation may be nonlinear and the transformed space high dimensional; thus though the classifier is a hyperplane in the high-dimensional feature space.

Support vector learning strategy is a principled and very powerful method that has outperformed most other systems in a wide variety of applications [8]. In the last decade, SVM learning has found a wide range of applications [6], including image segmentation and classification, object recognition, image fusion, and stereo correspondence. More recently, SVMs have been employed in several applications in biomedicine.

3. Design characteristics

This section defines the functional requirements of e-doctor and describes its design characteristics.

3.1. Objective

The purpose of e-doctor is to provide an interface where users can enter statistical medical information about a disease, and get predictive results about patients’ condition. What we expect from e-doctor is to provide a true / false answer to the question if a patient has (or may have) a disease. The answer is based on information provided by the user. The information is consisted of a number of arithmetical or logical parameters related to the disease (usually, medical exam data). After entering the information, e-doctor provides a diagnosis, based on its experience on the specific disease. Of course, the user must train e-doctor before using it, in order for the latter to obtain experience. The training is performed by providing several records of parameters about the disease, along with the answer expected for each case. Yet, note that the correlation of the statistical parameters to the disease existence, and the sufficiency of the statistical data are in the user’s responsibility.
The concept described above constitutes the basis for the further definition of functional requirements and design.

3.2. Functional requirements

The actors of our application are the user and the administrator. The first one can enter information to the system about a patient, and get a diagnosis. The second one can additionally define new diseases, find the appropriate database of training sets and train the system.

The use cases that the system supports (see Figure 1) are the diagnosis, those related to the definition and training for new diseases, and those related to user authentication. Their detailed description is the subject of the rest of this subsection. The description of the use cases related to user authentication is outside of the scope of this paper.

![Figure 1: User case diagram](image)

3.2.1. Diagnose

The use case starts when the user chooses the ‘diagnose’ link from the application’s main menu. A list is shown where the user chooses the disease for which he wants a diagnosis. A form is shown where the user enters patient information (name, medical insurance ID, date of birth, gender) and parameter values related to the patient. These parameters may concern medical exam results or other information related to the patient’s health.

Upon completion, the user presses the ‘next’ button. The system responds with an answer saying if the patient has the specific disease. The user presses ‘main menu’ in order to return to the menu with the main actions.

3.2.2. New Disease

The use case starts when the administrator chooses the ‘Define new disease’ link from the main menu. A form is shown where the administrator enters the disease name along with a brief description and the disease-related parameters. This is a repeating procedure. The user presses the ‘insert new parameter’ button and fills the parameter definition window that appears. Then, he can choose the ‘add new’ button, or the ‘done’ button. In the parameter definition window, the administrator enters the parameter name, the default value and the range of values the parameter can get. If it concerns a logical parameter, the ‘logical’ check box is ticked. Finally, a complete list with all the parameters is shown, along with an ‘update’ and ‘delete’ button next to each one. When the administrator finishes defining parameters, he presses the ‘done’ button and the Train use case is executed.
3.2.3. Train

This use case is an extension of the New Disease, Update Disease, and Re-train Doctor use cases. It is started when the administrator chooses to train the system for a specific disease. A form is shown with a list of all the disease-related parameters for which the administrator must enter data, along with the format and range of each one. The system indicates the administrator to upload the data in a spreadsheet. The order of values is indicated, along with the following note: ‘The last column of the spreadsheet must be the diagnosis outcome for the specific patient in a numerical format (1 indicates positive and vice-versa)’. A file chooser exists below the note, where the administrator uploads a spreadsheet containing the values. Then, the administrator presses ‘next’. After the file upload and the completion of the training procedure, the system responds with a success message. The administrator presses the ‘main menu’ button, in order to return to the menu with the main actions.

3.2.4. Other use cases

Concerning the rest of the use cases, their flow is similar to those described above. Hence, for simplicity reasons, its detailed explanation is omitted, and a brief description is given instead.

Delete Disease deletes all the information about a disease. Diagnoses performed for this disease are not deleted.

Update Disease updates information for a disease (e.g. name), giving the option to re-train the system. Re-train Doctor enables the administrator to re-train the system for a specific disease. The training can be from scratch, or it can be complementary, by entering more training data. Again, the Train use case is called in order training data to be uploaded.

View History enables the user to view information about diagnoses performed by e-doctor for specific patients. Several filters and sorting keys are supported, like patient, disease, date e.t.c.

3.3. Application structure

Our application design is based on the MVC [9] model. The functional blocks are depicted in Figure 2. e-doctor is implemented in Java, JSP and HTML. GlassFish and MySQL are chosen as application and database servers respectively. In the figure below, S stands for servlet, B stands for Java Bean and F stand for HTML/JSP form.

![Diagram of the proposed architecture](image-url)

Fig. 2: The proposed architecture
The SVM engine itself requires complicated math equations, which cannot be easily implemented in Java. For this purpose, we implemented the engine in Matlab. An adapter was developed, which is the interconnecting interface between our Java-based application and the Matlab-based SVM engine. In the Figure above, M stands for Matlab. Additional information for using Matlab functions with Java can be found in [10] and [11].

Web forms constitute the interface to the user / administrator. The coordinator is the servlet that organizes the whole process and calls the corresponding entities. The matlab adapter is a servlet implementing the necessary functions for calling the matlab-based SVM. The ORM [12] engine is a set of Java Beans that interact with the MySQL database to provide or store data (user information, exam results from e-doctor or training data). Finally, the SVM engine is a set of Matlab functions that implement the SVM.

4. A working example

In this Section, a working example of an automatic diagnosis from e-doctor is described step by step. Furthermore, the procedure of defining the disease (preceded to diagnosis) is also explained. The example concerns the implementation of a scenario described in [6]. In this paper, the authors implemented an SVM that makes diagnosis about coronary artery disease. Ten parameters related to the disease were defined. Statistical results from heart beat rate variability (HRV) signals were taken, which were further used to train the system. With the use of our application, the whole process described above is performed in a way transparent to the user, without the need of an experienced developer. Figure 3 shows the form that the user fills in order for e-doctor to make a diagnosis.

Fig. 3: Automatic diagnosis form
Pressing the ‘perform diagnosis’ button, the system answers if there is a strong probability that the patient will present coronary artery disease in the future (see Figure 4).

![Fig. 4: Automatic diagnosis form](image)

Last, Figure 5 shows the definition of a new disease, where the administrator already defined three parameters.

![Fig. 5: New disease definition](image)
5. Experimental Results

This Project was implemented an SVM that makes diagnosis about coronary artery disease. Ten parameters related to the disease were defined. Statistical results from heart beat rate variability (HRV) signals were taken, which were further used to train the system. With the use of our application, the whole process described above is performed in a way transparent to the user, without the need of an experienced developer. The following figure 6 and figure 7 shows that the user fills in order for E-Doctor to make a health problem.

Fig. 6: Automatic Daises Form

Fig. 7: Automatic Diagnosis Form
6. Future work

Primary future work concerns the design and implementation of alternative use cases, which were not considered in the current version, and the support of training file formats other than spreadsheets.

Furthermore, we are planning to isolate the SVM engine to a separate application, which can be called through web services. This way, developers will be able to implement their own interfaces and define their own use cases, while they will have the ability to use the SVM engine in a geographically independent environment. Moreover, this will constitute an architectural pattern, where each scientist can develop his own automatic diagnosis engine as a web service. Standard APIs can be defined, in order a distributed platform of interfaces and diagnosis engines to be developed.

7. References


