UNIVERSITÄT
LEIPZIG

Concept learning

Faculty of mathematics und informatics
of the university of Leipzig

A thesis submitted in partial fulfillment of the requirements for the degree of B.Sc. in computer science

By
Abdulnaser Sabra

In the date …

Supervisor: Prof. Dr. Gerhard Brewka

Acknowledgements

I would like to thank Prof. Dr. Gerhard Brewka for his continued support and encouragement throughout the course of this project. His keen interest enthusiasm towards research inspired me a great deal and motivated me to resume the project and think about a career in research. In addition, I would like to thank my friends and my family for their continuous support.

Table of Contents

Table of Figures

Abstract

Concept learning algorithms play a major role in learning new concepts to predict the outcomes of future inputs. This study presents a website that can generate the version space boundaries regarding a submitted training data set. The algorithm used in the background is the candidate elimination algorithm. The algorithm iterates over the list of the training instances and by the end it produces the final border sets of the version space. The website provides the user with an interactive interface and enables him/her to either submitting the training records directly to the website by creating a training table or uploading the training table from his/her machine. It uses state-of-the-art JavaScript libraries to automatically process the data set submitted by the user. Different programming and script languages are used to achieve the functionality of the website and make it responsive on all devices regardless of what their operating system they use or what the size of their screen is. The special focus is on JavaScript to achieve the functionality of the website and on Html, CSS, Bootstrap to achieve the responsiveness of it.

1.Introduction

Learning can be observed as the process of obtaining new information and knowledge over a period of time. As humans, we are capable of gaining new knowledge with experience and identifying some features, to classify objects. The computers can only execute some lines of code given to them by the programmers. They are not capable of extracting patterns or gaining knowledge from experiences over a period of time.

In recent years, the term “machine learning” has won a lot of importance in the computer science field. “Research in machine Learning has been concerned with building computer programs able to construct new knowledge by using input information”. Machine Learning algorithms allow computers to derive general hypotheses that can be used to predict future outcomes. Those hypotheses can be thought of as the knowledge the computer gain after being trained on different training examples. The training examples can be thought of as the experiences the computer has seen during its learning phase.

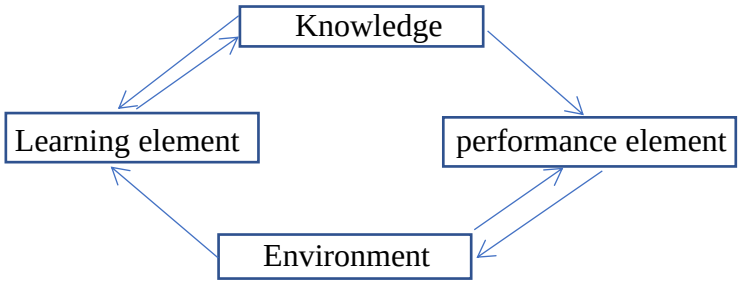


Figure 1: learning model

The above figure represents a general learning model. It can be seen that the performance element integrates with the environment and it is controlled by the knowledge that the system has gained. The better knowledge the system has gained during its learning phase, the more accurate it performs on new input data. The learn element can be thought of as the learning algorithm that takes the training data from the environment to be trained on. The learn element can be thought of as the learning algorithm that takes the training data from the environment to be trained on. The more training data is submitted to the learning algorithm, the more enhanced the gained knowledge is. The learning model can be classified according to three dimensions, 1) learn strategy 2) The knowledge the system gains and 3) the scope of the learning system.

As this work is concerned with concept learning and concept learning is assigned to the learn strategy dimension, it is important to mention the different learning strategies and it is not necessary to write deeply about the two other dimensions, as they are not covered in this work.

Different learning strategies can be taken into consideration when it comes to generating new conclusions out of the available ones or learning new concepts. Storing data into the computer’s memory can be viewed as a learning strategy, even though the computer cannot derive information from the stored data or make conclusions out of them. Just storing them into the computer and making the computer see them, is a way the computer can learn, as its memory fills up with new data. As humans, when we are given new information about history, science, or something else, we consider ourselves that we have learned something new, even though we did not involve in discovering the information or making conclusions out of them. Just the idea that we store new information in our memories, makes us learn something new. The same way the computer can be thought of. When its memory is filled with new data (e.g database), it can be claimed that the system has learned something and this can be seen as the simplest way the computers can learn by storing new information into their memory or by programming new programs.

On the other hand, the deductive learning strategy enables using and analyzing the general available knowledge to conclude new knowledge. In other words, from the general given knowledge new specific one can be excluded.

Learning through analogy is quite important, as the already available knowledge and experiences can be used to derive new knowledge and experiences in new situations, whereas there is already knowledge derived for similar situations. This type of learning might not be so efficient when it comes to deriving new knowledge in situations, where for the new situations there is no similar one situation experienced already by the system. Concept learning can be thought of as an inductive learning strategy. Learning a specific concept requires prepared training data, on which the machine can be trained, and by the end a hypothesis can be derived that can be used, to classify new future instances. [1][3][6]

2. Concept learning

2.1 Concept

A concept refers to any term that is used to define an object or an event. Concepts can be structured as small groups or categories referred to subsets which can be identified in relation to a larger set referred to as the super set [1][5].

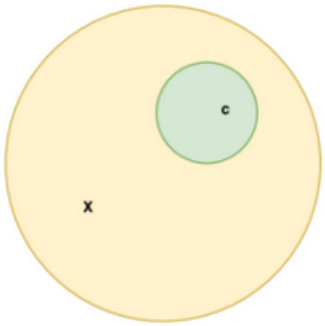


Fig 2.1: concept representation

Taking the above figure into consideration, let c be the concept of laptops that encompasses all types of laptops in the world and X be the set of all computers in the world including laptops. It can be seen from the above figure that c which is the target concept of laptops is specified over x, the set of all computers. Every instance (laptop) of the target concept should belong to the superset, but not every instance from the superset can be included in the target concept. For instance, a desktop cannot be considered a laptop, and in this way, it cannot be encompassed by the target concept. Alternatively, a concept can be regarded as a function that returns a Boolean value either 0 or 1. That is, when an input is given, the function will generate a 'yes' or 'no' indicating whether or not this input is a laptop. The returned value will be true for a laptop and false for a desktop. It is in this sense that a concept can be considered as a Boolean valued function. As the subset of objects is defined in relation to a larger set of objects, it can be assumed that there is a particular set of features that characterizes the subset and also differentiates it from the larger set. In other words, since c is a subset of the larger set X and as c is characterized in relation to x, the particular set of properties that define c – a subset of laptops – is what differentiates it from x – a set of computers. This set of features that differentiates the laptops can be called a concept [1][5].

2.2 Completeness and correctness of concepts

In this section, the features of a concept will be explored.

2.2.1 Completeness

As it is explained in the previous section, x is the set of all computers in the world and the target concept encompasses all laptops available in the world.

Completeness refers to the fact that all laptops in the world must be covered by the target concept. When this is not the case, the concept cannot be called as a complete concept. In other words, the concept should assign a given laptop to 1. However, it should not be deduced from the definition of completeness that the concept assigns the non-laptops to zero.

This is expressed by the following mathematical formula: $\forall\, e \in X\ (e\ is\ a\ laptop \Longrightarrow c(e) = 1)$. Based on this formula, it can be claimed that it does not make any difference which value a non-laptop example is assigned to, the completeness feature will remain regardless. What matters is that a given laptop gets assigned to true [6].

2.2.2 Correctness

As the name suggests, the feature of correctness means that the target concept ought not to comprise of any example that should not be covered by it in relation to the set of examples X. In other words, the concept laptop should not comprise of a desktop. In mathematical formula, this is expressed as: $\forall\, e \in X\ (e\ is\ a\ not\ a\ laptop \Longrightarrow c(e) = 0)$. Based on this formula, for a concept to be correct, it does not matter, whether it assigns a given laptop to 0 or 1. What is important is that it assigns a not-laptop (negative example) to zero [6].

2.2.3 Consistency

When a concept has the two features explained above (i.e. completeness and correctness) with respect to the set of examples E, then it can be said that the concept is consistent with the set of examples E [6].

2.3 Concept learning

The human learning and object recognition process relies most upon the gained experiences from the past. Concept learning can be interpreted as analyzing and evaluating the training data to locate a general definition that best fits the training examples and depends on that a classifier can distinguish whether an object/instance belongs to a particular category or not.
A grown human being can determine if an observed object is a laptop or not using past experiences. On the contrary, a two years old kid cannot make this distinction. Similarly, a machine cannot distinguish which category a considered example belongs to without being trained on early examples.

2.4 Representation of training examples

Training examples representation:

One of the most effective ways a human's brain can distinguish objects is through depending on the gained experiences from the past, which empowers it to recognize objects dependent on particular highlights. The key here is the accumulated experiences and how a machine can be trained on the training data to derive a general definition of the concept. The training examples structure a significant component regarding concept learning and machine learning, so the samples must be shaped so that the machine can process, analyze and understand them.

Example	Eyes	Nose	Head	FColor	Hair	Smile
1	Round	Triangle	Round	Purple	Yes	Yes
2	Square	Square	Square	Green	Yes	No
3	Square	Triangle	Round	Yellow	Yes	Yes
4	Round	Triangle	Round	Green	No	No
5	Square	Square	Round	Yellow	Yes	Yes

Figure 2.4: Training examples representation

The training dataset can be viewed as a table containing some examples to be trained on, where every row consists of a single training example. The first row consists of the attributes/features, and every attribute has its own value-domain. Every row is a training example, and it contains the values of the corresponding attributes.

The above table also contains the dependent attribute smile, which specifies if the observed person smiles or not based on the given highlights.

2.5 Representation of hypotheses

Hypotheses are made use of within concept learning to discover the target concept. A hypothesis is the communication medium through which the learner can express his thoughts about a concept. A hypothesis can be viewed as a combination of constraints.

The following are some examples of constraints:

-? it does not matter, what the value of the attribute is ("Water=?")

- A precise value (water=cold)

- empty set, no value is accepted

For example: <?,Triangle,Round,?,?>

The above vector represents the hypothesis that a person smiles when his noise is triangle and his head is round.

An example *e* is said to satisfy the hypothesis *h*, when *h(x)=1*, regardless of whether *x* is a positive or a negative example of the target concept. The example is said to be consistent with the hypothesis, if and only if, *c(x) = h(x)*. What is asked here is the relation between the target concept *c* and the hypothesis *h*. The consistency of *h* can be examined for a single example as well as for a set of examples *D*. When we take a training dataset *D* into consideration, hypothesis *h* can be said to be consistent with *D*, if and only if, for each example *x* from *D* the following applies: *h(x)=1 iff c(x)=1*, and this is equal to the condition that *h(x)= 0 if c(x)= 0*. The consistency of a hypothesis with an example can also be expressed the other way around: an example is consistent with a hypothesis [1].

2.5.1 Special and general hypotheses

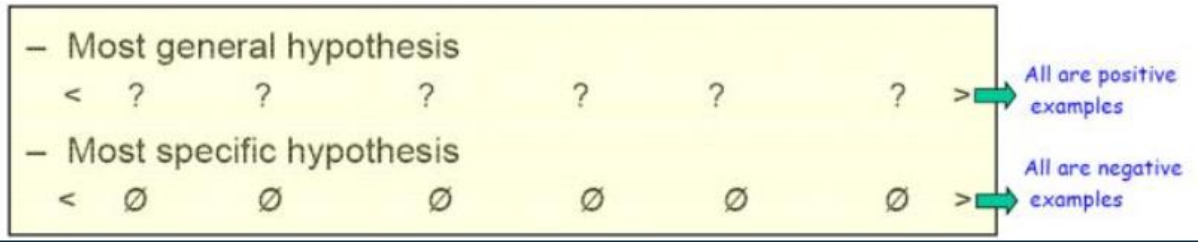


Figure 2.5.1: Most general and specific hypotheses

There are two types of special hypotheses: namely, the maximally general (or most general) and the maximally specific (or most specific) hypotheses. A hypothesis g_0 is called a maximally general hypothesis only when it encompasses all examples, whether they are classified as positive or negative. In other words, every possible example must be covered by the hypothesis (all x from dataset X fulfills g_0) and considering the example on the smile data set, it can be said that a person

smiles always. On the other hand, a hypothesis is referred to as the maximally specific hypothesis only if it does not cover any example (There isn't any x from the training set X fulfills S_0) [5].

- maximally general hypothesis: let h be the maximally general hypothesis, then $\forall\, e \in X\ (h(e) = 1)$.

- maximally specific hypothesis: let h be the maximally specific hypothesis, then $\forall\, e \in X\ (h(e) = 0)$.

2.6 Concept learning components

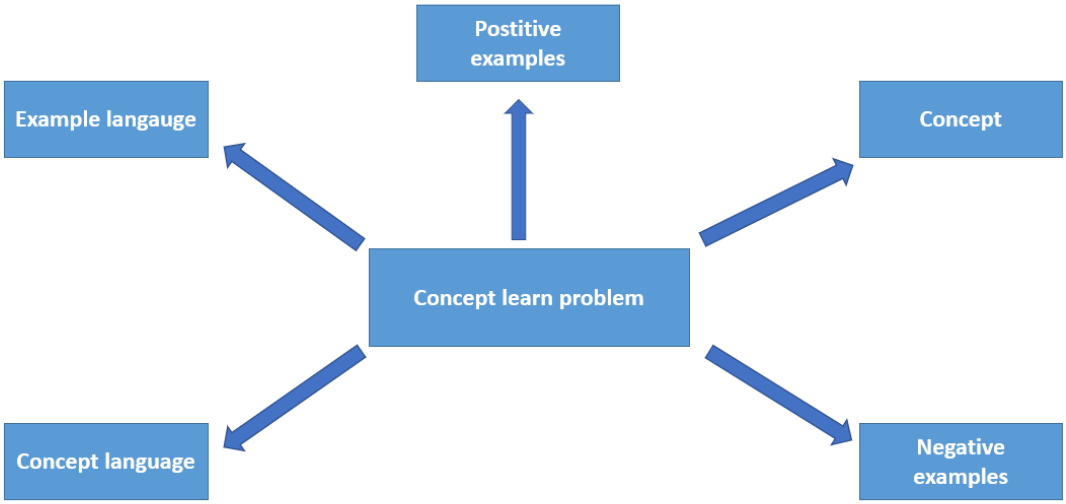


Figure 2.6: Concept learning components

A concept learning problem consists of five components:

As clarified above, there must be such a way, how the examples should be composed and consequently a language required in that specific circumstance. Consider the above figure, it can be said that the set of all possible 6-tuples attribute values provides the example language for the attributes Eyes, Nose, Head, Fcolor, Hair, so it can be applied that e=<Round, Triangle, Round, Purple, Yes> belongs to Le. On the other hand, there must be a language describing how a particular hypothesis can be composed, and this is additionally referenced above in the chapter of hypotheses (chapter 2.5). It can be said that the set of all 6-tuples of constraints over the possible attribute values provides the concept language. The most important component is the concept itself, which has to be learned and for which a hypothesis ought to be searched, which best fits the training examples. A significant component in the concept learning process is the training set, which without it, it is preposterous to achieve the learning process. The training examples are separated into two sets, the positive and the negative examples. This process's objective is to infer a hypothesis h that belongs to the set of all hypotheses, and for every given, an example should yield a similar outcome as the concept C. h(e)=c(e). [5].

2.7 Relation between hypotheses

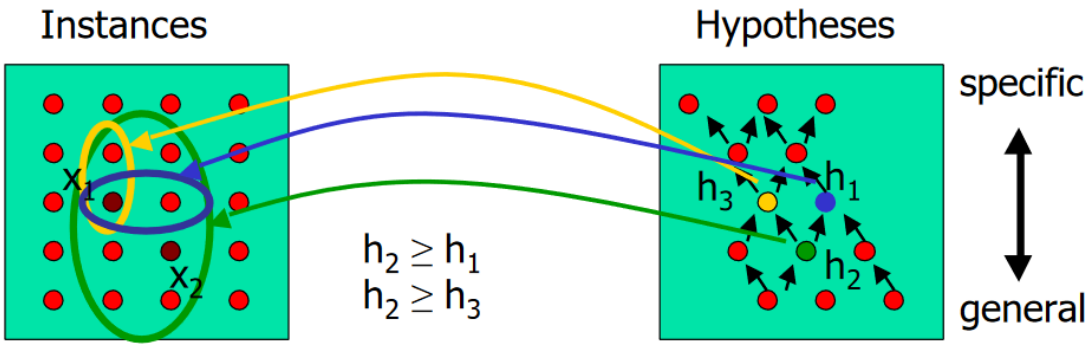


Figure 2.7: Relation between hypotheses

On the right side of the figure is the field of hypotheses that contains all possible hypotheses for an observed concept. All hypotheses in this field are in order from the maximally specific to the maximally general hypothesis. Each circular point within the field signifies a hypothesis while the arrow on each demonstrates the relation between the hypotheses. When an arrow stems from the hypothesis h2 and points towards the hypothesis h3, this means that h2 is more general than h3. On the left side of the figure is the set of all examples regarding the concept. In its relation to another hypothesis, a hypothesis can either be more or less general or more or less specific. If we assume that there are the following two hypotheses, *ha* and *hb*, it can be claimed that *ha* is more general than or equal to *hb*, On the assumption that any example that satisfies *hb* also satisfies *ha*. For the following definition to be valid, it must also be referenced that the two hypotheses are Boolean-valued functions [3][1].

$$\forall\, x \in X\ [(Hb(x) = 1\ ==>\ (Ha(x) = 1)]$$

The above graph shows three different hypotheses and the relations between them. The hypothesis h2 >= h1 signifies that h2 is more general than h1 and that refers to the fact that the number of constraints imposed by h1 is greater than the number of constraints imposed by h2. This can also be observed from the figure above on the left side, where the circle of the examples covered by h2 encompasses the circle of examples covered by h1. From this, it can be asserted that the more general a hypothesis is, the higher the number of instances covered by it will be.

H1 = <Sunny,?,?,Strong,?,?> H2 = <Sunny,?,?,?,?,?>

The second hypothesis H2 contains more question marks "? " than the first hypothesis H1. Hence, the fourth attribute in the second hypothesis takes more values as the one in the first hypothesis because it is denoted as "?". Subsequently, the number of instances covered by the second hypothesis is greater than the number covered by the first, and for this reason, it can be claimed that the second hypothesis is more general than the first one.

3. Learning algorithms

There are various learning algorithms that inspect various methods for exploring the hypothesis space that can be utilized to infer a general definition consistent with the perceived training examples. In this chapter, the distinctive learning algorithms will be investigated with their focal points and disadvantages. The learning algorithms' functionality depends on the given training examples that will be utilized to conclude a final hypothesis that will simulate the target concept on future examples. The learning process can be thought of as a searching process through the large hypothesis space characterized by the hypothesis representation language to find the hypothesis that best fits the observed training examples [1][5].

Concept learning as a search problem

Concept learning approach can be regarded as a search problem, in which the set of all possible hypotheses will be searched through. As explained above in chapter 2.7, since it is the language Lc that depicts the hypotheses, it is essential to choose it with caution. The number of instances and conceivable hypotheses can be determined based on the number of values every attribute can take. When the example on training in chapter 2.4 is taken into consideration, it can be argued that the number of all instances is 3*2*2*2*2*2= 96, where 3 is the number of values that the first attribute *Sky* can take, whereas the rest of the attributes take two possible values [2]. With regards to the number of possible hypotheses, in every hypothesis in addition to the concrete values, constraint values such as (empty set, ?) can take place. This adds to the number of possible values, 2 for every attribute, thus making the number of the possible hypothesis 5*4*4*4*4*4 = 5120. As already indicated, each attribute has a value scope of at least one; based on this, there is no attribute that does not take any value. Therefore, all hypotheses that have at least one attribute with the value (i.e., the empty constraint set) will be excluded and not contemplated. When this aspect is taken into consideration, and after all hypotheses that contain at least one (empty set) constraint are gotten eliminated, the final number of possible hypotheses that can be searched through in order to infer the general hypothesis is 1+(4*3*3*3*3*3) = 972 [1][6].

3.1 Find S algorithm

3.1.1 Introduction

One of the most fundamental and straightforward algorithms in concept learning is the Find-S algorithm. When this algorithm is used, only the positive examples are examined while the negative ones are disregarded. It begins by initializing the most specific hypothesis and then iterates through the list of all positive examples. Each time an observed positive example is iterated but not classified by the algorithm, the hypothesis is generalized in such a way that it becomes consistent with the example and this process is repeated. The hypothesis that results from this should be consistent with all positive examples in the training data [3][1].

3.1.2 Example

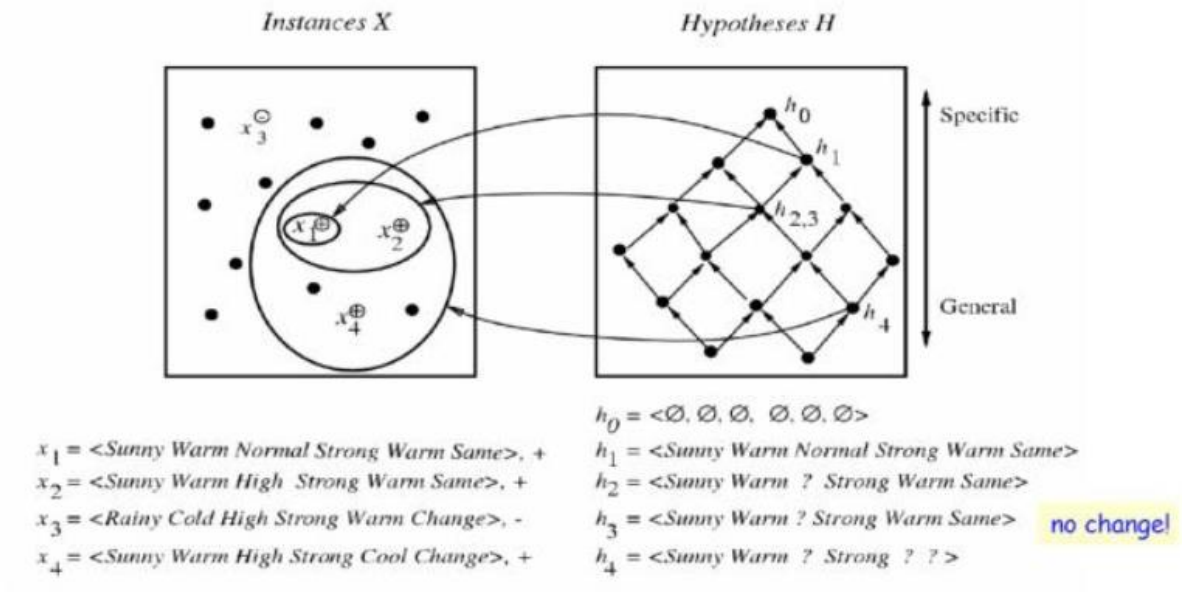


Figure 3.1.2: Find-S example figure

The above figure shows an example of how Find-S algorithm capacities. On the left side is the set of all instances and on the right side is the hypothesis space.

h0 = {Ø, Ø, Ø, Ø, Ø, Ø}

The process begins with initializing h0 that is the most specific hypothesis and afterward we iterate over the set of all training examples.

h1 = {sunny, Warm,Normal, String, Warm,

The information in example 1 is {sunny , warm, normal, string, warm ,same} and it is classified as positive example, it tends to be seen that the hypothesis h0 is more specific and thusly , it ought to be generalized to cover the example. The hypothesis becomes h1 = <Sunny, Warm, ?, String, Warm, Same > so all the values of the first example have been added to the hypothesis.

h2 = {sunny, Warm,?, String, Warm, Same}

it tends to be viewed that the second example have as well a positive outcome.

Presently, it very well may be seen that the hypothesis and the second example have different values at the third attribute , in this manner , the hypothesis should to be changed such that it fits both examples the first and second and hence the value of the third attribute should be generalized.

h3 = h2

Because the third example has a negative outcome, it ought to be disregarded and not contemplated and thusly remains the hypothesis unaltered.

h4 = {Sunny, Warm , ? , Strong , ? , ? }

The last example has a positive outcome and the values of all attributes of both the example and the hypothesis need to be compared and it can be seen that the values of the last and before last attributes are different and therefore the hypothesis should be generalized to fit the example.[1].

3.1.3 Limitations of Find-S algorithm

As explained above, since the negative examples are not taken into consideration in Find-S algorithm, there is no way of checking whether or not disregarding these examples could mislead the resulted hypothesis. In another words, it cannot be tested if the final hypothesis is consistent with the negative examples or not. Assuming that the following training example <Sunny, warm, Normal, Strong, Cool, same > were to be included into the set of all training examples in the above example and this example had a negative outcome. Within the framework of the Find-S algorithm, this example should have been discarded as its outcome is not positive. However, in spite of the fact that its outcome is negative, it is covered by the resulted hypothesis, which ought to not be the case. Another disadvantage of this algorithm is that it does not check to see if the hypothesis derived is the only possible one consistent with the data or if there are other hypotheses that fit. In other words, the algorithm stops working as soon as it reaches a hypothesis that is consistent with the training data [1][3].

3.2 Version space:

As explained in the previous section, since the negative valued examples are not taken into consideration with the Find-S algorithm, it cannot be guaranteed that the resulted hypothesis would be consistent with all training examples. The Find-s algorithm generates just one hypothesis consistent with the observed training data while there may be more than one hypothesis that fit the training data. This limitation can be addressed through the candidate elimination algorithm, which is used to generate the version space. Both positive and negative examples are examined with the candidate elimination algorithm. The version space VH,D of a hypothesis space H and a training dataset D contains all hypotheses that are consistent with the training dataset. The set of all hypotheses that are consistent with the given training dataset can be referred to as the version space of the target concept with respect to the concept language and the training data. In contrast to using the Find-S algorithm that retrieves just one hypothesis consistent with the training data, the version space learning process retrieves all hypotheses that are consistent with the data. Using the "general-as" relation is helpful in providing a concise description of the version space [3][5].

The version space incorporates the uppermost and lowermost limits and the hypotheses that lie between these. Every hypothesis in the version space is consistent with all observed training examples.

$V_B = \{ h \in Lc \mid h \; is \; correct \; and \; complete \; regarding \; D \}$

Example:

Illustration of $V_{H,D}$ for the example set D:

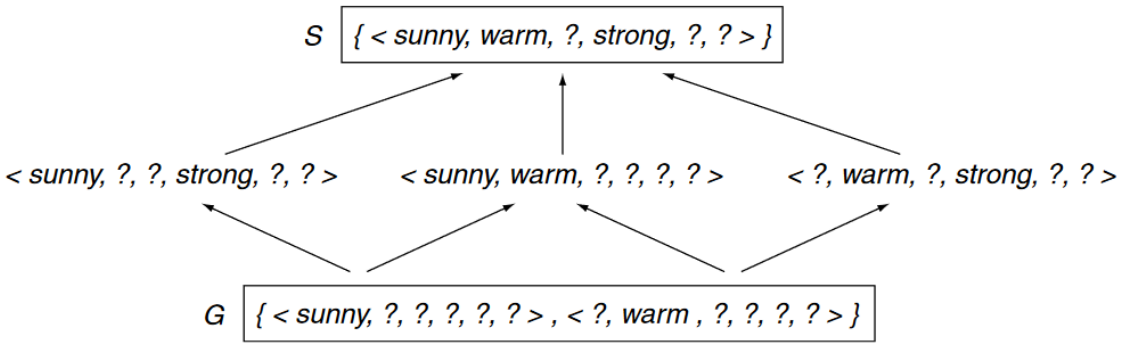


Figure 3.2: The version space of the training data

The above figure shows the version space of the training examples D in Figure 2.4. It can be seen that the boundary S contains the most special hypothesis of the version space. Whereas the boundary G contains the two most general hypotheses. The arrows indicate the relations between the above hypotheses. There are six distinct hypotheses in the above version space. Another depiction of the version space can be accomplished through the two border sets S and G, since it is conceivable to generalize and count all hypotheses that lie between these two boundaries. In order to achieve this, the "general-as" relation discussed in the chapter 2.7 can be used [6].

The boundaries S and G constitute the borders of the version space that contain all hypotheses that fit the training data. These hypotheses are the most **special** and most **general** generalizations of the example set. A hypothesis h can be referred to as the most special generalization of the dataset under the following conditions: that is it complete and correct in its relation to the training data; and that there are no other hypothesis h' that is complete and correct regarding the data, where h'<h. In other words, it can be said that S is the set of minimally general (i.e., maximally specific) hypotheses of H consistent with D.

$$S \equiv \{s \in H | Consistent(s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge Consistent(s', D)]\}$$

The same principle can also be applied to a hypothesis that is a most general generalization of the training data. In that case, in addition to completeness and correctness, the condition is that there should not exist any hypothesis that is more general than the hypothesis h'>h [6].

$$G \equiv \{g \in H | Consistent(g, D) \wedge (\neg \exists g' \in H)[(g' >_g g) \wedge Consistent(g', D)]\}$$

A hypothesis is said to be included in the version space when it is less general than or equal to one of the most general hypotheses held within the upper boundary. Likewise, a hypothesis is also included in the version space when it is less specific than or equal to one of the most specific hypotheses contained in the upper boundary. To put it differently, the hypothesis should either be contained in G or S, or lie between them [4].

Taking into account the lower boundary in the Figure 3.2.1, the hypothesis <Sunny, Warm, ?,Strong,?,?> can be said to be the most specific generalization, as it is both correct and complete, and as there exists no other hypothesis in the version space that is more specific. Based on the most general boundary in the Figure 3.2.1, the two hypotheses contained in the general boundary can be considered the most general generalizations, as they are both complete and correct and as there exists no other hypothesis in the version space that is more general than them.

3.2.1 Candidate elimination algorithm:

After the concept of the version space has been defined and explained in the preceding chapter, it is very fundamental to explore the candidate elimination algorithm that is being used to construct the version space. Given the set of the training examples E and the hypotheses space V, the candidate elimination algorithm builds incrementally the version space. The training examples will be iterated over one by one and each training example may shrink the hypothesis space by getting rid of the hypothesis that are inconsistent with the considered example. The upper and lower boundaries of the version space will be represented and updated with each training instance. the candidate elimination algorithm describes the set all hypotheses that are consistent with the observed training set. This set is a subset of the hypothesis space and it is called the version space, as it contains all versions of the hypotheses that best fit the training data [1][2

S and G, which are the boundary hypothesis, represent the version space V. For each new example S and G must be checked and if necessary be adjusted. For a considered example e, if there is a hypothesis h belonging to S or G and it is consistent with the example i.e. it applies that h(e) = 1, if e is a positive example and h(e) = 0, if e is a negative example, in this case, the boundaries should not be altered. If it is the case that h is inconsistent with the considered example, then two cases can be taken into consideration. Either e is a positive example and although e is mapped wrongly to zero by the hypothesis h(e) = 0 or e is a negative example and it is mapped wrongly to 1 by the hypothesis h(e) = 1

3.2.2 **Pseudocode**: No palagrsim

Function create_versionSpace_boundaries(E):

// E is the set of examples

// S is the lower bound of the version space and contains the most special generalizations regarding the example set E.

S = {s0}

// G is upper bound of the version space and contains the most general generalizations regarding the example set E.

G = {g0}

//The function isMoreGeneral(h,e) returns true if the hypothesis h is more general than the observed example e.

FOREACH e ∈ E DO

IF e is positive THEN

FOREACH g ∈ G DO// Checks out if the example is not covered by the hypothesis g

If not isMoreGeneral(g,e) THEN:

G = G\{g} // g should be removed from G.

FOREACH s ∈ S DO // returns a set of all minimal generalizations of g regarding e

IF not isMoreGeneral(s,e) THEN:

S_mingeneral = minGeneralizations(s,e)

S = S \ {s}

//checks out, if there is at least one hypothesis g in G, such that G is more general than s.

FOREACH s1 in S_mingeneral DO

FOREACH g1 in G DO

IF isMoreGeneral(g1,s1) THEN

S = S ∪ {s1}

//removes all hypotheses from S, that are more general than any other hypotheses in S.

FOREACH s2 in S Do

FOREACH s3 in S DO

IF isMoreGeneral(s2,s3) THEN S = S \ {s2}

```
IF e is negative THEN
    FOREACH s in S DO
        IF isMoreGeneral(s,e) THEN
            S = S \ {s} //removes s, because it covers a negative example
    FOREACH g in G DO//returns a set of all min specializations of g regarding e
        IF isMoreGeneral(g,e) THEN
            G_minspec = minSpecialization(g,e)
            G = G \ {g}
    // it should be made sure, that there is least one hypothesis s in S, such that g is more general
       than s.
    FOREACH g in G_minspec DO

        FOREACH s in S DO
            IF isMoreGeneral(g,s) THEN
                G = G ∪ {g}

    // it should also be made sure that there does not exist any hypothesis g1 in G, such that g1 is
       More special than another hypothesis g2 in G.
    FOREACH g1 in G DO
        FOREACH g2 in G DO
            IF g1 isMoreGeneral(g2,g1) THEN
                G = G \ {g1}
```

Figure 3.2.2: The pseudocode of the candidate eliminate algorithm

The above figure shows the pseudocode of the algorithm. The input for the algorithm is the set of the training examples and the algorithm spies out a representation of hypotheses that are consistent with the given training examples. The algorithm starts with initializing the version space to the set of all hypotheses in the hypothesis space H. Every hypothesis is found to be inconsistent with the training data will be removed .The examples will be iterated over once at a time and if the example is positive then it must be made sure that it is covered by all hypotheses in the upper boundary G, because the G set represents the upper boundary of the version space and therefore all hypotheses within G must cover all positive examples and if it is the case that there is a hypothesis in G that does not cover the new positive instance , then it must be removed. Furthermore, the S set, which is the lower boundary of the version space, must be also checked out to see, if it contains any hypothesis s that does not cover the positive example and once one found, it gets deleted. moreover, it is not enough to delete s, all its minimal generalizations regarding an observed positive example should be also generated. The generalized hypotheses should be as minimal as possible and they should cover

the observed positive example. As it is mentioned above s is contained in the lower boundary S of the version space, therefore the generalization of a hypothesis s ∈ S must be as minimum as possible to not lose the specialization feature. Additionally, the generalized hypothesis s must be less or equal to at least one general hypothesis. The hypotheses contained in S are considered as the most specialized hypotheses in the version space and therefore, there should not exist any hypothesis s1 ∈ S, such that s1 is more general than a hypothesis s2 ∈ S and if this the case then the hypothesis s1 should be removed. Observing a negative example e, all hypotheses contained in S must be deleted, if they cover the example. If there is a hypothesis in G that cover e, then it must be deleted and all its minimal specializations must be so generated, that they do not cover the negative example e. Additionally, all hypotheses in G that are more general than other hypotheses in G must be also deleted [1][2].

3.2.3 Example:

attributes

Sky	Temp	Humid	Wind	Water	Fore-cast	Enjoy Sport
Sunny	Warm	Norr	example	Warm	Same	Yes
Sunny	Warm	High		Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Figure 3.2.3: training data set

S0 = {Ø, Ø, Ø, Ø, Ø, Ø}

G0 = {?, ? ,? ,? ,? ,?}

The first thing that should be done is to set the most specific and most generic boundaries. G0 = <?, ? ,?,?,?,?> is the most generic boundary comprising of question marks and S0 = <Ø, Ø, Ø, Ø, Ø, Ø > is the most specific boundary comprising of the nulls.

E1 = sunny, warm, Normal, strong ,warm ,same ➔ +

Considering the first example, the first example is a positive example. The hypothesis within the most generic boundary contains all question marks and all question marks match the values of all attributes of the first example and thusly the hypothesis within the generic boundary remains unchanged and similarly , it must also be looked at the hypothesis in the most specific boundary and check out whether it is consistent or not with the example . it contains all nulls and is not consistent with the observed example and the hypothesis maps wrongly the example to zero though the example is a positive one. Since it is not consistent with the example, it ought to be generalized to S1: <Sunny, Warm,Normal,Strong,warm,Same> .

S1 = {sunny, warm, normal, strong, warm, same}

G1 = G0

E2 = sunny, warm, high, strong ,warm ,same ➔ +

The second example is also a positive example. Initially, the hypothesis within the most generic boundary must be consistent with the example and in light of the fact that it is consistent with the example, it stays unaltered G2 = G1. Checking now the hypothesis within the most specific boundary, it can be said that S1 wrongly maps the positive example to 0 and accordingly it isn't consistent with the example , as the value of the Humidity attribute at the s1 hypothesis does not match the value in the example and therefore s1 hypothesis must be generalized, so as to match the example . The value of the humidity attribute in s1 must be replaced with the question mark.

S2 = {sunny, warm, ?, strong, warm, same}

G2 = G1

E3 = rainy, cold, high, strong ,warm ,change ➔ -

The third example is a negative example. firstly, it should be ensured that s2 is consistent with the example and considering the third example it can be said that s2 is not consistent with the third example and in this manner it remains unchanged s3 = s2. The hypothesis within the most generic boundary contains all question marks and accordingly, it maps wrongly the example to zero. Since the hypothesis is not consistent with the training example, all hypothesis that are consistent with all the training examples seen till now will be written. In another words, all the minimal specializations of the hypothesis g2 regarding the observed example E3 should be written such that they are consistent with all previous examples and not consistent with the negative ones. For composing those particular hypotheses, one question mark at a time should be considered.

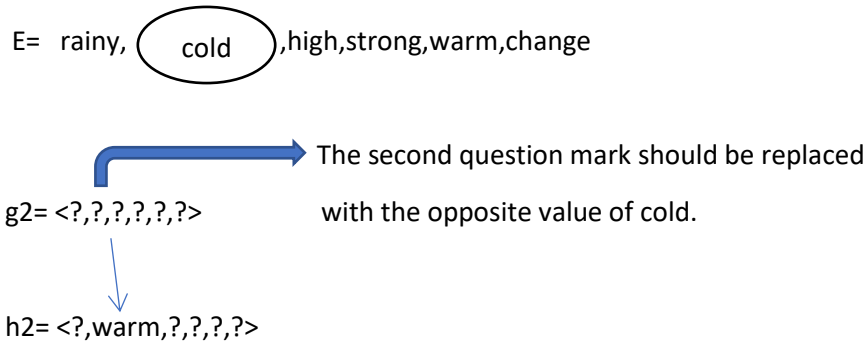
E= rainy ,cold,high,strong,warm,change

The first question mark should be replaced with the opposite value of rainy.

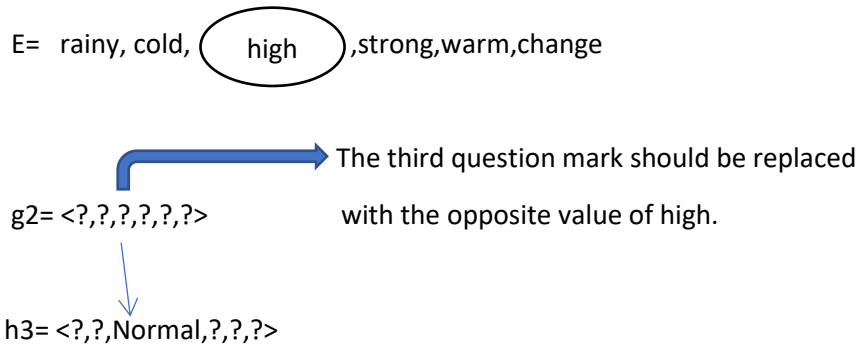
g2= <?,?,?,?,?,?>

h1 = <sunny,?,?,?,?,?>

The first question mark at g2 must be replaced with the opposite value of the first value of E3 and that is Sunny and all other question marks at g3 remains unchanged so the outcome is <sunny,?,?,?,?,?>.



The next hypothesis should contain at its second value the opposite value of the value at the second index in the E3 and that is Warm and all other values are question marks <?,Warm,?,?,?,?> .



The third hypothesis are constructed similarly through replacing the third question mark by the opposite of High that is Normal <?,?,Normal,?,?,?>. For the 4-th attribute within the negative example, there should not be any change, because there is no opposite of the value ”Strong” in the table. The last two hypothesis are constructed similarly <?,?,?,?,Cool,?> and <?,?,?,?,?,Same>.

h1 = <sunny,?,?,?,?,?>

h2 = <?,warm,?,?,?,?>

~~h3~~ = <?,?,Normal,?,?,?>

~~h4~~ = <?,?,?,?,cool,?>

h5= <?,?,?,?,?,same>

After composing all hypotheses, it should be made sure that these hypotheses are consistent with the previous examples. There are five hypotheses that have been written and now the question whether all of them are consistent or not. If they are consistent with all seen examples, they can be kept and if they are not, then they should be removed. The first and second hypothesis are consistent with the first two examples and not with the third one, therefore, they should not be deleted. The third hypothesis is not consistent with the second example, because the third value in the hypothesis (Normal) does not match the third value in the second example and therefore it should be removed. The fourth hypothesis is not consistent with the first example as the 4-th value does not match the value in the first example. Th lest hypothesis can be kept as it is consistent with all the first three examples.

S3 = s2

G3 = { <Sunny,?,?,?,?,?> , <?,warm,?,?,?,?> , <?,?,?,?,?,Same> }

E4 = sunny, warm, high, strong ,cool ,change ➔ +

The last example is a positive example. All hypotheses that are consistent with the example should be retained and if there is a hypothesis that is not consistent with the example, then it should be removed. The first and second hypothesis at the most generic boundary <Sunny,?,?,?,?,?> <?,Warm,?,?,?,?> are consistent with the example as they are more general than the example and therefore they should not be removed. The last hypothesis in the most generic boundary is not consistent with the example as the value at the last index (Same) does not match the value of the last attribute in the example and therefore it should be removed.

G4= { <Sunny,?,?,?,?,?> <?,Warm,?,?,?,?> }

The last example cannot be covered by the hypothesis at the most specific boundary S3 as the 4-th and the 5-th values (Warm ,Same) in the hypothesis does not match the values of the 4-th and 5-th attributes (Cool, Change) in the example and therefore the last two values in the hypothesis should be replaced with the ? , so that the examples can be covered by the hypothesis.

S4 = { <Sunny, Warm,?,Strong,?,?> }

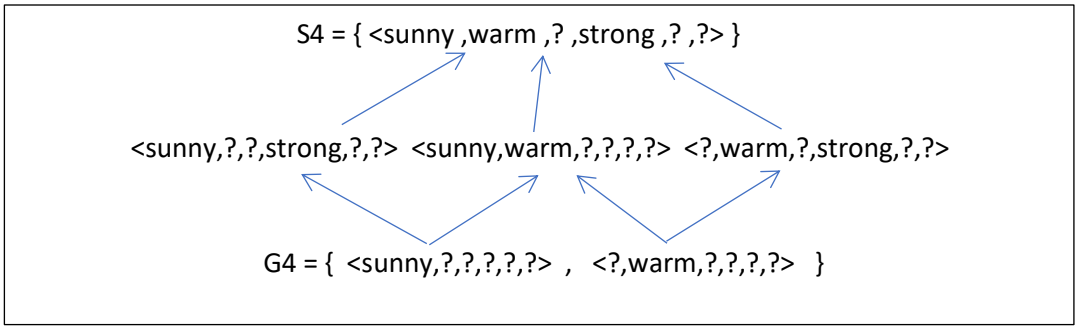
S4 = { <sunny ,warm ,? ,strong ,? ,?> }

G4 = { <sunny,?,?,?,?,?> , <?,warm,?,?,?,?> }

S4 and G4 are boundaries of the version space

```
For each g in G4:
    For each s in S4:
        writeHypothesisBetweenSandG()
```

After constructing the boundaries, the version space must be generated. This can be done through constructing all the hypothesis that lie between the two border sets. All hypotheses that lie between a hypothesis in G4 and a hypothesis in S4 must be written. in another words, each hypothesis in G4 (g) must be iterated over and then for each hypothesis in S4 (s), all hypotheses that lie between g and s must be included into the version space.



The first hypothesis in G4 is <sunny,?,?,?,?,?> and the hypothesis in S4 is { <sunny ,warm ,? ,strong ,? ,?>

The value of first attribute in g4 matches the value of the first one in S4, so there is nothing to be done in this case, the value of the second attribute in g4 is more general than the one in the second attribute in s4, therefore a new hypothesis must be constructed <sunny,warm,?,?,?,?>. it can be viewed that the new constructed hypothesis is more general than s4 and more special than g4 and because the value of the 4th attribute in g4 is more general than the one of the 4th attribute in s4 , a new hypothesis must be constructed <sunny,?,?,strong,?,?>.

The same process must be applied for the second hypothesis in G4 and the resulted hypotheses are <sunny,warm,?,?,?,?> <?,warm,?strong,?,?> .

3.2.4 Features of the version space learning process:

Prior to beginning the learning process, some prerequisites that must be met to converge against the hypotheses that best define the target concept.

3.2.4.1 The training set does not contain any errors:

Firstly, it should be ascertained that the given data set does not contain any errors – that it is correct. One way in which the data can be set to contain errors is if a positive example e is introduced to the learning process as a negative one. In such a case, all hypotheses that deal with the example e must be deleted as it is a negative example. Moreover, since the example e is also covered by it, the right target concept of the version space will also have to be eliminated. Similarly, depicting a negative example e to the learning process as a positive example could affect the search for the right hypotheses. That is, if the example e were to be represented incorrectly as a positive example, all hypotheses that do not cover it would have been deleted; as a result, the right hypothesis that define the concept will have been eliminated from the version space as well because it does not cover the example e. Such an example could expose the failure provided that there are enough training examples. When there are a sufficient number of positive and negative training examples, a pattern can be derived from each set of examples. A thorough observation of these patterns can be helpful in detecting the training examples that were incorrectly represented. In turn, an incorrectly represented example could lead to the disclosure of the error, since the version space will finally have fallen into the empty set [6].

The version space's breakdown could also be due to the target concept not being able to be described in the representation language.

3.2.4.2 Concepts that cannot be described in the hypothesis space:

Sky	Temp	Humid	Wind	Water	Forecast	Enjoy Sport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
cloudy	Warm	Normal	Strong	Warm	Same	Yes
Rainy	Warm	Normal	Strong	Warm	Change	No

Figure 3.2.4.2: training data set

It is not always conceivable to infer a hypothesis from the hypothesis space that best fits the training data. This depends on the training data and how it is structured. When the above data set is observed, it can be claimed that there exists no hypothesis that is consistent with the training data (i.e., complete and correct in relation to the training set) and that can be expressed in the hypothesis space L_C of the training data. The most special hypothesis that is consistent with the first two examples is the following hypothesis: h = <?,Warm,Normal,Strong,Warm,Same>. This hypothesis is not consistent with the third one – it is wrong because it incorrectly covers the third example, although, it is a negative example. The issue is that it is not possible to describe disjunctions such as Sky = Sunny OR Sky= Cloudy in the concept language L_C, due to the constraint that the hypothesis space can only contain conjunctions of attribute values as discussed in section 2.5 Hypothesis representation. To put it simply, an attribute in the hypothesis space cannot be set to take on specific values. The version space learning process uses inductive bias and based on the inductive bias, only conjunctive hypotheses can be taken into consideration [1][5].

3.2.5 Concept learning with feature trees:

In the previous section, it was demonstrated that despite the fact that the last instance is a negative one, the resulted hypothesis incorrectly covers it. This is because the inductive hypothesis (i.e., inductive bias) – with which the learning system functions – allows only conjunctive hypotheses to be taken into consideration, and therefore it is not possible to assign an attribute a specific number of values (Sky = sunny OR sky = cloudy).

This issue is addressed by feature trees which can be used to order attribute values hierarchically. Through this, generalization can be avoided in the case that two different values of an attribute coincide.

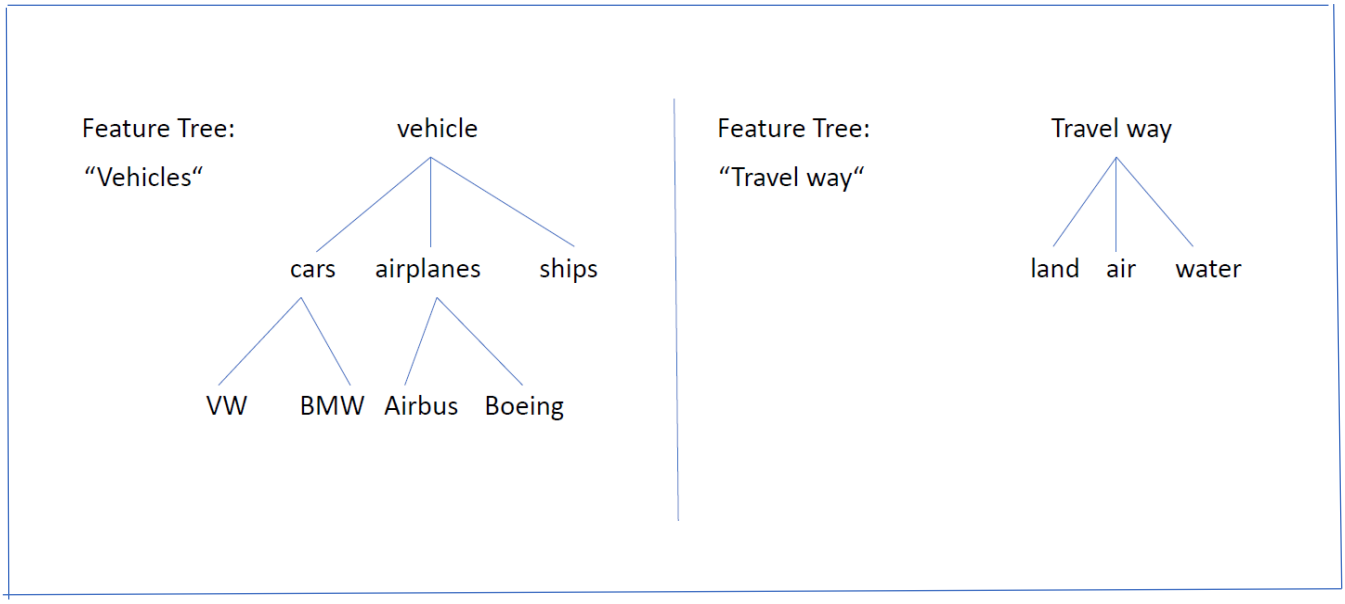


Figure 3.2.5: training data set

The figure above examines a concept learning problem about a concept that identifies specific vehicle types and travel ways. The feature tree on the left side demonstrates the vehicle type and the one on the right side depicts the travel way.

As explained in section 2.6, each concept learning problem consists of five components. The example language is the set of all pairs (a and b) with a from {VW, BMW, Airbus, Boeing, ships} and b from {land, air, water}. The potential values of a and b subsequently can be seen on the leaves of the feature trees in the above figure. Another important component is concept language which consists of all pairs [A, B], where A and B are features of the feature trees in the above figure. Both A and B should be nodes of the feature trees in the figure: "A" for vehicle and "B" for travel way. Examples of concept description are [cars, land], [Airbus, Travel way], [vehicle, Travel way]. The following component of the concept learning problem is the Boolean defined function given by the concept description: [A, B]: example language ➔ {0,1}. The input is an example (a, b) and [A, B] is applied to the example (a, b) and [A, B] generates a 1 if the node A in the feature tree for vehicle is situated above the leaf a or is equal to the leaf a. It will generate a zero if the node B in the feature tree for travel way is situated above the leaf b or is equal to the leaf b.

The concept description [airplanes, Travel way] applied on the example (Airbus, air) generates a 1, as the example is encompassed by the concept. On the other hand, the concept description [airplanes,

Travel way] applied on the example (BMW, land) returns a 0, because the example cannot be covered by the concept, as the feature “airplanes” is not situated above “BMW” in the feature tree.

3.2.5.1 Version space on feature trees:

Nr	Example	Classification
E1	(BMW, Water)	No
E2	(Airbus,Air)	Yes

Figure 3.2.5.1: Training data set (Feature tree)

Initially, s_0 and g_0 of the two boundary sets are initialized. S_0 = {[Ø, Ø]} and G_0 = {[Vehicle, Travel way]}. S_0 contains the concept description, whose existence the empty set is. The two border sets encompass the entire version space of all hypotheses. Considering the first example, the first example is a negative one and S_0 should not be altered and therefore S1 = S0. The most general hypothesis [Vehicle, Travel way] in the boundary set G0 covers wrongly the first example ,because the “vehicle” lies above “BMW” and “Travel way” lies above “Water” , and therefore it must be specialized. There are 5 minimal specializations of G_0 regarding the first example E1 that do not cover E1: h1 = [Vehicle, land], h2 = [Vehicle, Air], h3= [Airplanes, water], [Ships, water] and [VW, Water]. The hypotheses do not cover the example and they are minimal specializations , as there does not exist any other hypothesis that is more general than any of the hypotheses {h1, h2, h3, h4,h5} and does not cover the first example. Considering these two observations, it can be claimed that the 5 hypotheses structure the minimal specializations set of G_0 regarding the first example E1. The hypotheses are as well more general than the hypothesis that is contained in S_0 and therefore , the 5 hypotheses can be written in G1 , also G1 = { [Vehicle, land],[Vehicle, Air],[Airplanes, Water],[Ships, Water], [VW, Water] }.

S0 = {[Ø,Ø]}

G0 = { [Vehicle, Travel way] }

S1 = S0

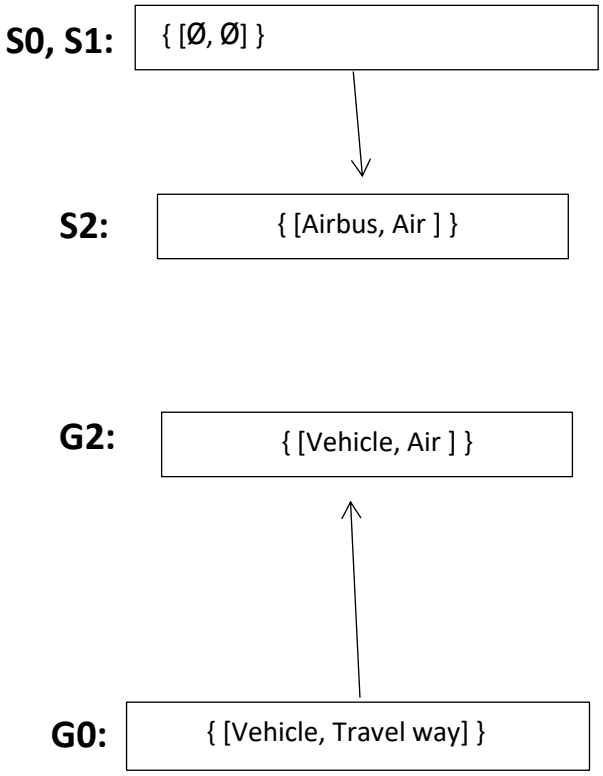
G1 = {[Vehicle, land], [Vehicle, Air], [Airplanes, Water], [Ships, Water],[VW, Water]}.

The second example is a positive one and it cannot be covered by the hypothesis in S1 and therefore S1 should be generalized into S2, whereas S2 = { [Airbus, Air] }. This generalization is minimal, as there does not exist any other hypothesis that is more special than the hypothesis in S2 and can cover the example. All hypotheses in G1 that does not cover the example should be removed from G1, as it is a positive example and should be covered by all hypotheses in G1. All hypotheses except [Vehicle, Air] don’t cover the example and therefore should be removed from G1. [Vehicle, Air] is more general than [Airbus, Air] and therefore [Vehicle, Air] can be written in G2, also G2= {[Vehicle, Air]}.

S2 = {[Airbus, Air]}

G2 = {[Vehicle, Air]}.

The resulted version space is as following:



4.Implementation

The final objective of this work is to create a software that displays the version space depending on the training data given by the user. The software should enable the user to create a table of training data or upload the data from the computer without the need of inserting each row himself. Meanwhile, the user can see how the border sets S and G change by inserting every training row. The software allows the user also to edit or delete a specific row. By editing and deleting a specific row, the borders S and G will be adjusted automatically.

4.1 Tools:

The software should be accessible by the users, and the best way of achieving this is to create a website that the users can access. The advantage of creating a website is the ability to access it from any device that has an internet connection, and it is not limited to android, iPhone , windows ,Linux, or any other operating systems. Additionally, the website should be responsive; in other words, it should be rendered correctly on any device regardless of its screen size. Significant screen devices should display the content of the site differently from devices with small screens.

4.1.1 JavaScript:

JavaScript is a programming language that can be used to implement the website's logic. It contains many pre-implemented functions and features that enable the programmer to smoothly achieve his work without creating some functions from scratch. It provides the programmer with a function list that helps him/her access the interface elements and gets their contents. In this project, JavaScript implements the candidate elimination algorithm, which is the main algorithm to create the version space's border sets. Additionally, JavaScript functions check the validation of the user's data before processing them; for instance, the training data can be checked, whether they contain any null values.

4.1.2 html:

Html is the primary tool that can be used to create the structure, and the elements of the website. it allows the programmer to create buttons, text areas, tables. When the user access the website, the html file will be loaded and its content will be displayed by the browser.

4.1.3 CSS:

CSS is a style sheet language that is used to style the elements of the website (buttons, text fields, tables). It allows the programmer to determine what width and length specific Html elements should have. CSS enables the programmer to decorate the website.

4.1.4 Bootstrap:

The structure and order of interface elements cannot be the same for all devices. Bootstrap helps the programmer by determining how the elements should be structured for each device screnn size’s (Big,Medium,Small).

4.2 2-Tier-Architecture:

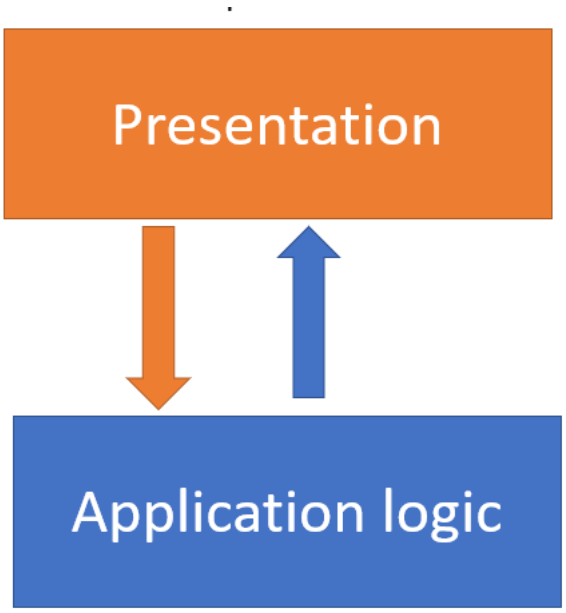


Figure 4.2: Architecture of the application

The application is divided into two tiers(layers), the presentation and application tier. The presentation tier contains all classes and modules that create and style the interface elements (button, text fields, tables). On the other hand, the application logic tier contains all classes and modules that handle the given data's processing. In the application logic, the candidate elimination algorithm, which is the main algorithm used to generate the version space, will be implemented. The presentation and logical tiers interact with each other. The presentation tier receives the training data from the user. It then sends them into the application logic tier that implements the candidate elimination algorithm using the received training data. Once the border sets S and G are generated, they will be sent to the presentation tier to present them to the user.

The changes in the presentation tier will be reflected on the application logic tier. A change in the property of an interface element should change the functions in the application logic that reference it.

The presentation tier is responsible for examing the validation of the inputted data. An input field expected to receive a numerical input should display an alert message if it gets a string from the user. Multiple application logic tier functions may reference a single interface element in the presentation tier. Therefore it is essential to check out the validation of the element in the presentation tier without checking them in every function that references the element.

4.3 Website hierarchy:

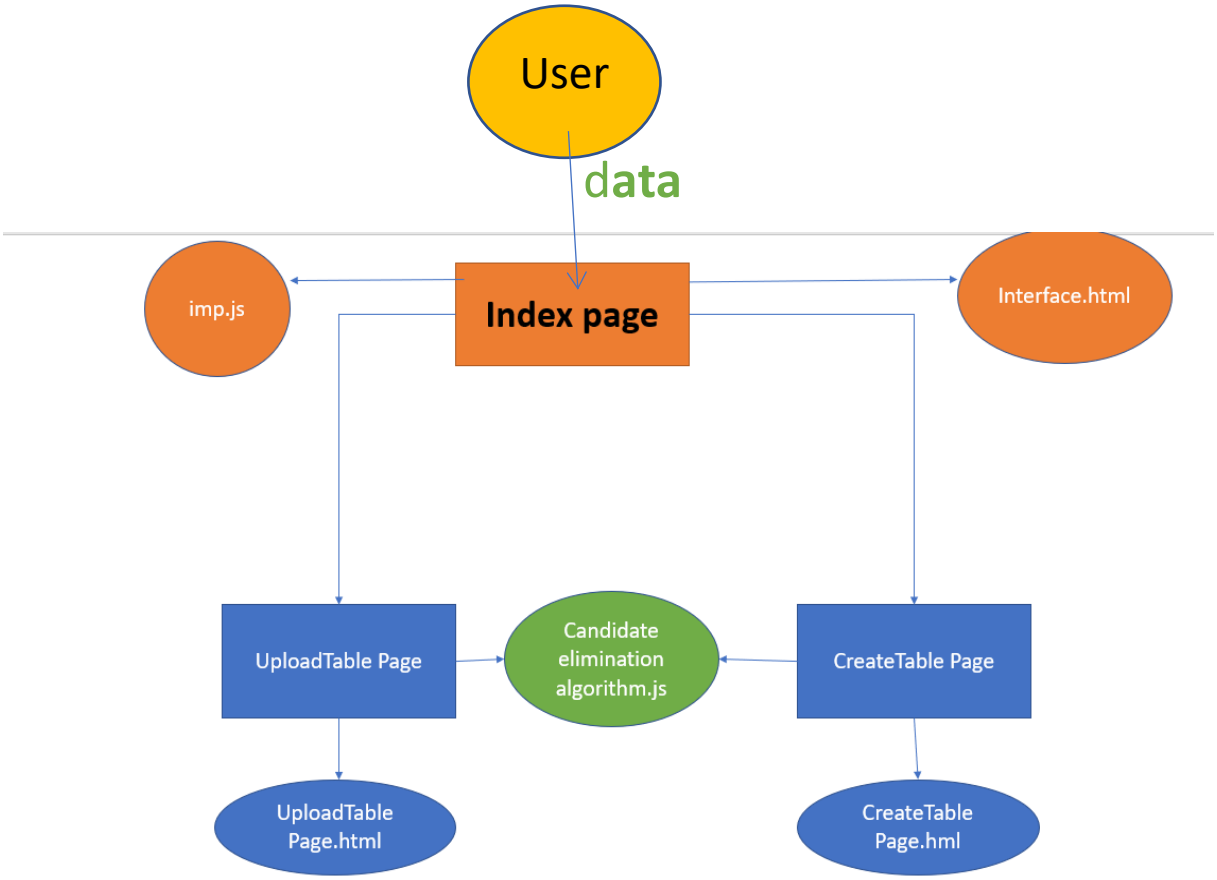


Figure 4.3: The website's hierarchy

The website consists of three pages, and every page has its classes attached to it. When the user accesses the website, the index page will be loaded and rendered. The index page is the first page in the application, and it enables the user to decide whether he /she wants to navigate to the Uploadtable Page or CreateTable page. The CreateTable Page allows the user to create a training table and specifies its header attributes. The user can also enter the training examples into the table and eventually run the candidate elimination algorithm to construct the boundaries and, finally, the version space.

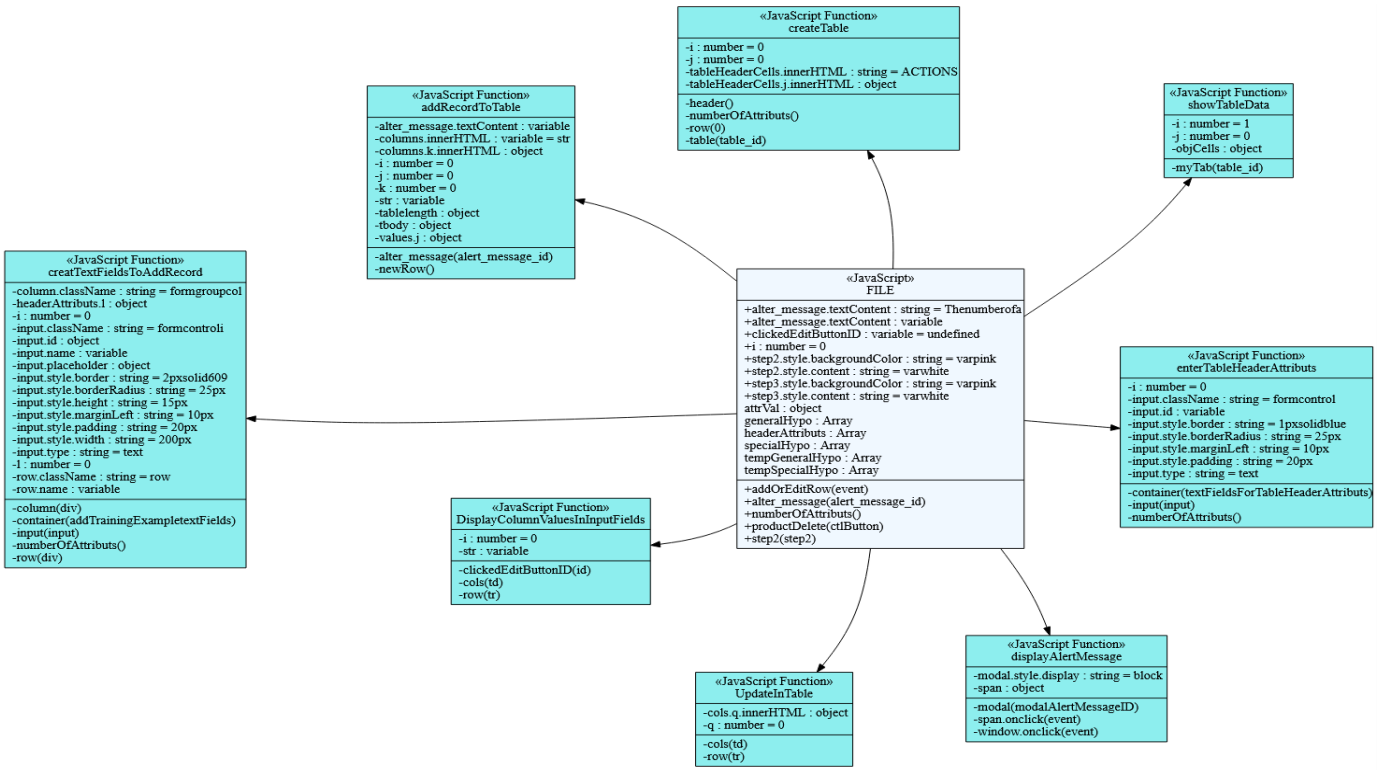
On the other hand, The UploadTable page enables the user to upload the training data in a CSV(comma-separated value) format from his/ her computer and run the version space algorithm directly without the need to create a table himself/herself and add every row to it. Every page consists of two classes. One provides an interactive user interface (extended with .html). The other class (extended with .js) processes the user's data and implements the candidate elimination algorithm to generate the version space. The first class is assigned to the presentation tier, and the other class is assigned to the application logic tier, as explained above.

4.4 UML:

As the implementation of this project is not build using the object oriented paradigm, it can be claimed that it is unnecessary to navigate deeply through the structure of the project and explain it, instead the main methods that were used to create the version space will be explored.

4.4.1 The interface file's UML (.js):

This file includes alle functions that enable the user to interact with the website. Not all functions and variables will be explored , rather just the most relevant and important methods.



1) enterTableHeaderAttributs(): allows the user to enter the header attributs of the table.
2) createTable(): creats a training data table.
3) addRecordToTable(): adds a new training data example.
4) createtextFieldsToAddRecord(): creats inputs fields to enable the user to add record to the table.
5) addOrEditRow(): enables the user to add new row into the training data or delete an already existing record form the table.
6) displayAlertMessage(): displays a message to the user , when something goes wrong.

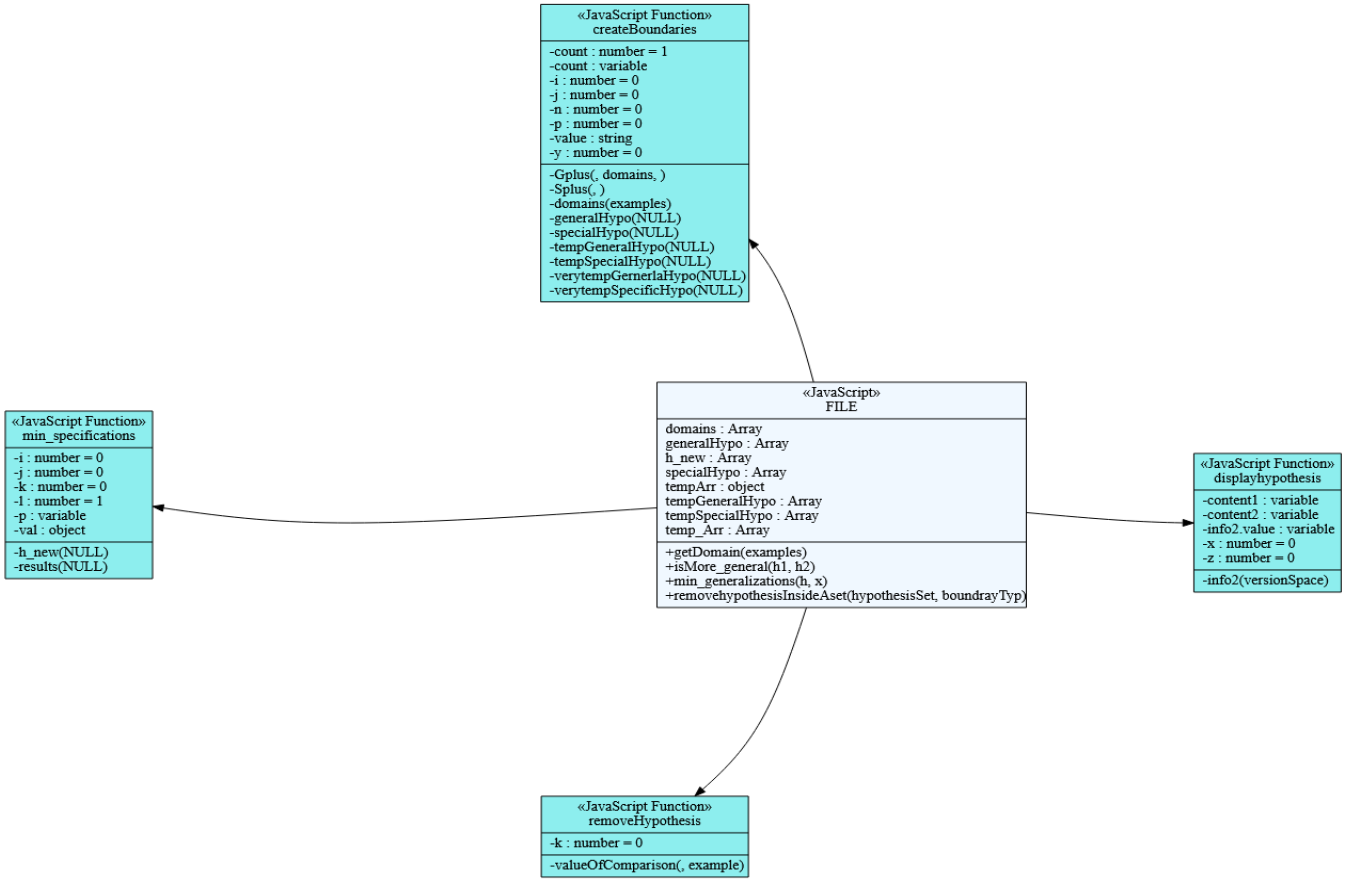
   For example: when the user enters a string in a field that is expected to receive an integer value.

7) numerOfAttributs(): returns the number of the head attributs of the training data table.

8) ProductDelete(): deletes a training example form the table.

4.4.2 The candidate elimination algorithm file's UML (.js):

This file includes all functions that are necessary to create the boundaries of the version space.



1) getDomain(examples): returns a list of domains of the training data.

   For example: [{sunny,cold,normal},{cloudy,warm,change},{sunny,cold,same}] ➔

   [{sunny,cloudy},{cold,warm},{normal,change,same}]

2) isMore_general(h1, h2): returns true , if the hypothesis h1 is more general than the hypothesis h2.

3) min_generalizations(h,x): returns a list of all hypotheses that are minimal generalizations of the hypothesis h considering a training example x.

4) min_specifications(h,x,domains): returns a list of all hypotheses that are minimal specifications of the hypothesis h and considering the training example x and the domains of the training data.

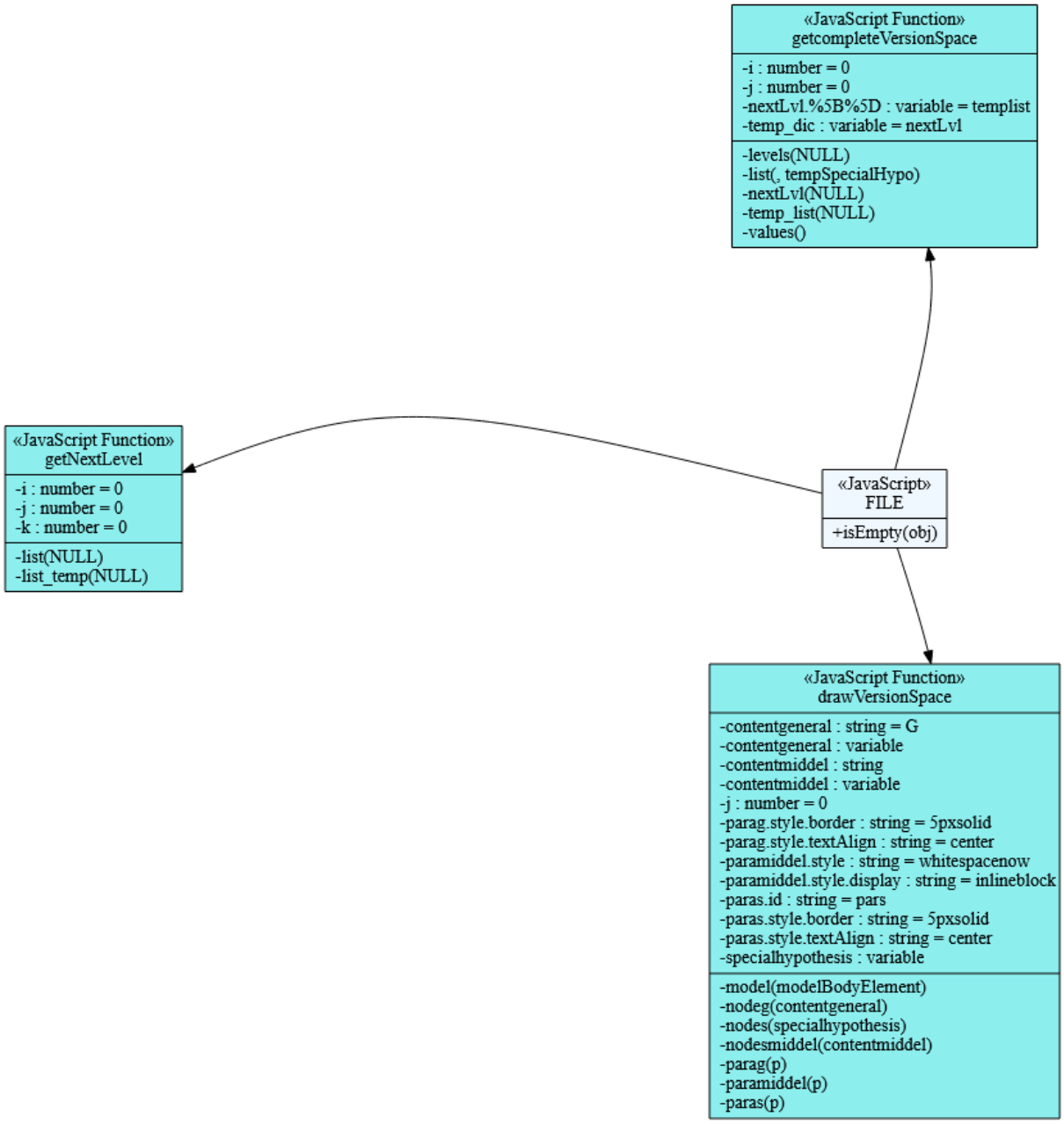
5) removehypothesis(h1,hypoList): removes hypothesis h1 from a list of hypotheses hypoList.

6) displayHypothesis(h1): displays a hypothesis in a text area in order to be seen by the user.

7) createBoundaries(): creats the boundary sets of the version space.

4.4.3 The version space file's UML (.js):

This file includes all functions that are necessary to build the version space.



1) getCompleteVersionSpace(S,G): returns a list of all hypotheses that lie between the two boundary sets S and G.

2) drawVersionSpace: draws the version space (the boundary sets with all hypotheses between them)

4.5 Pages:

4.5.1 Start page (index page):

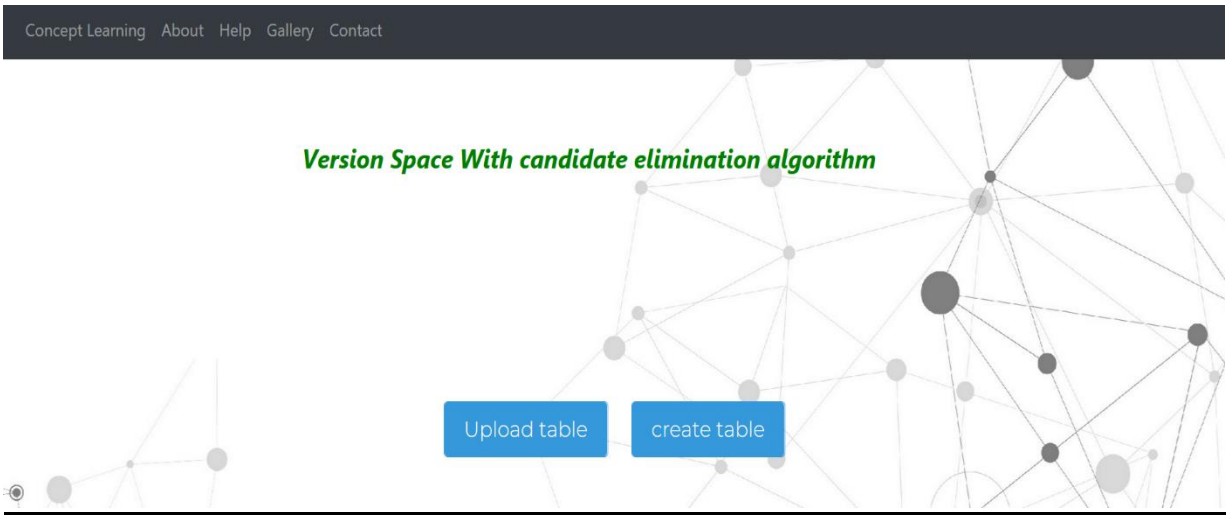
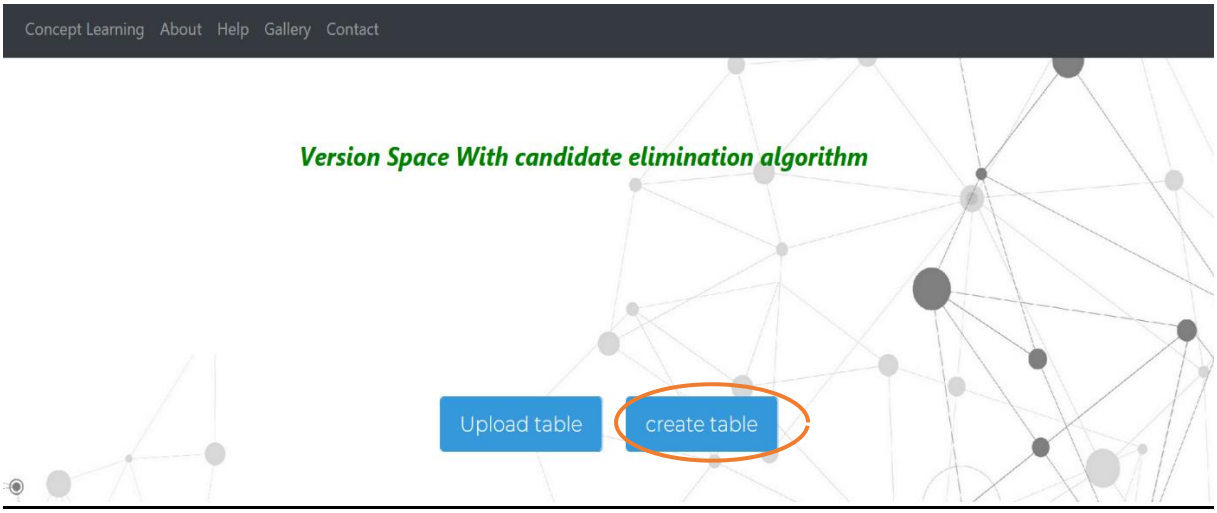


Figure 4.5.1: Start page (Index page).

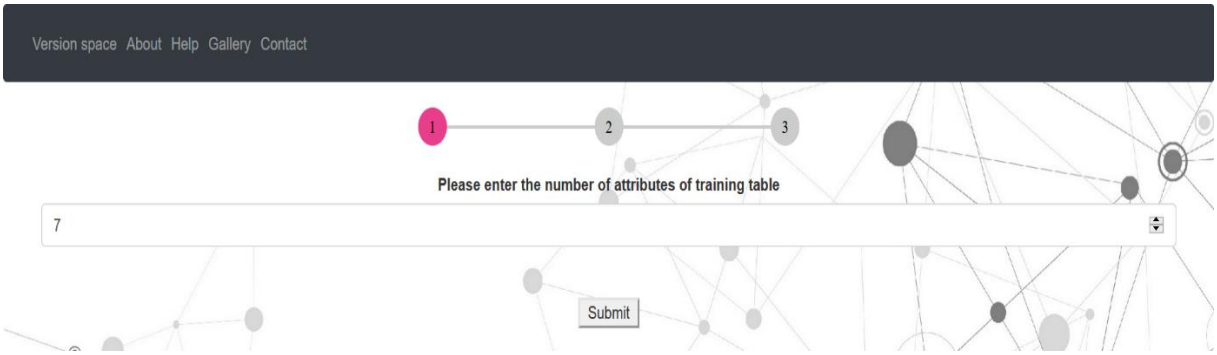
The above picture shows the first page of the website. The user can navigate to the createTable page by clicking on the button (create Table) and navigate to the UploadTable page by clicking on the UploadTable page. The navigation bar lists different buttons (Concept learning, About, Help, Contact). Every button leads to a different page.

4.5.2 CreatTable page:

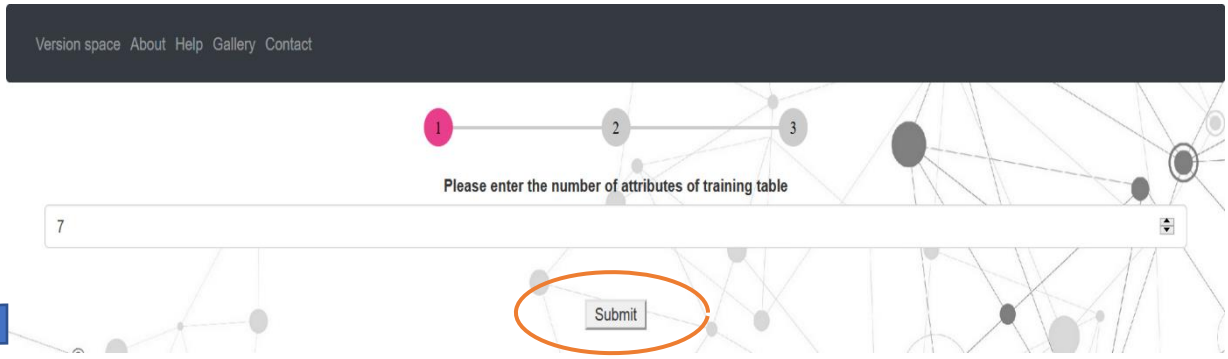
4.5.2.1 First step:



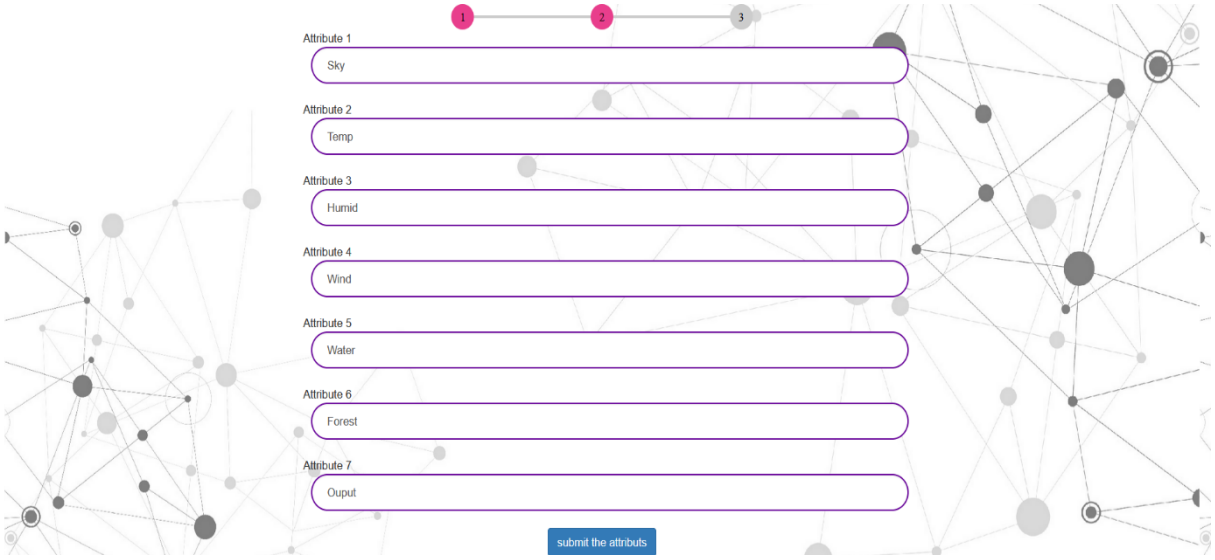
When the create Table button in the start page is clicked, the user will be navigated to the create Table Page. As explained previously, this page allows the user to create the training data table. The first step is inputting the number of attributes of the training table's header, 7, in the above example. The table header has at least one attribute, and therefore the input field takes just one positive numerical value greater than 0. If the user inputs a string value or a value less than 0, an alert message will be prompted.



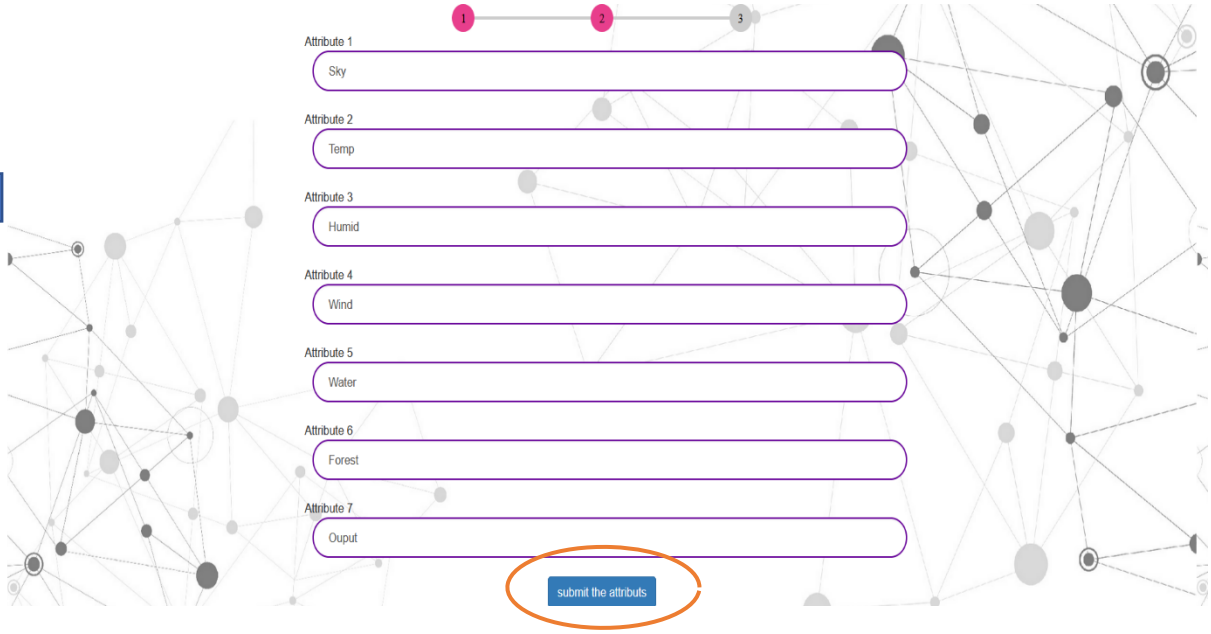
4.5.2.2 Second step:



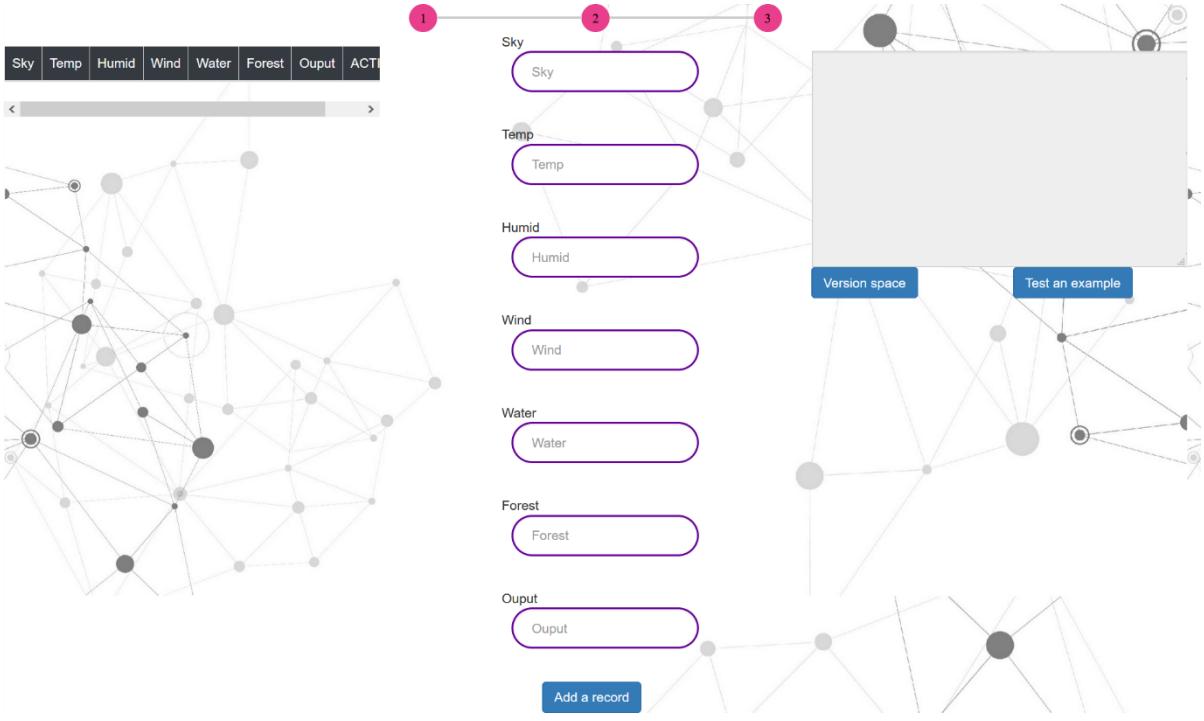
Once the number of attributes is given and submitted by the user, a form of input fields will be shown, where the user can input the header attributes of the table. The number of input fields matches the number that the user enters on the first page, which is 7.



4.5.2.3 Third Step:

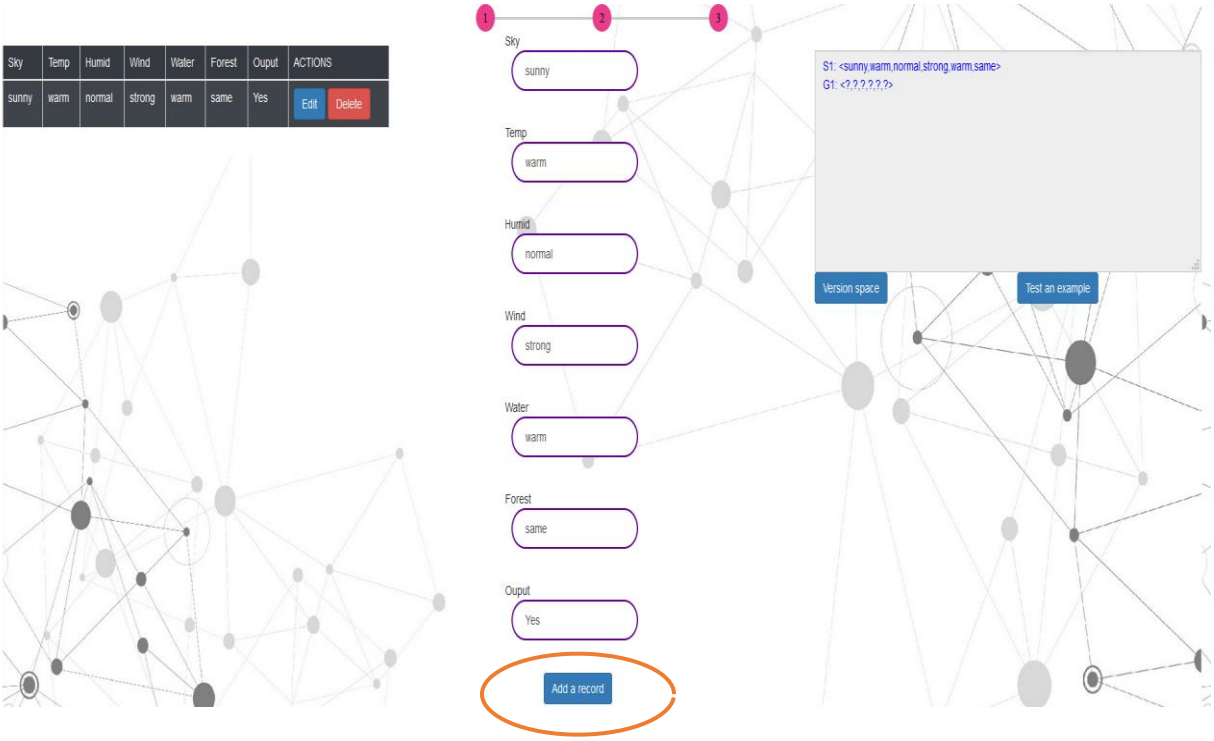


When all attributes are given and submitted by the user, the last page will be shown. It consists of three components, on the right side the training data table with its header attributes given by the user, a form of input fields that enables the user to add records to the table, and on the right side a text area, where the border sets S and G can be displayed. Whenever a training example is added to the table, the border sets S and G will be presented in the text area on the right side , so that the user can see what happens by adding an example and how the border sets change.

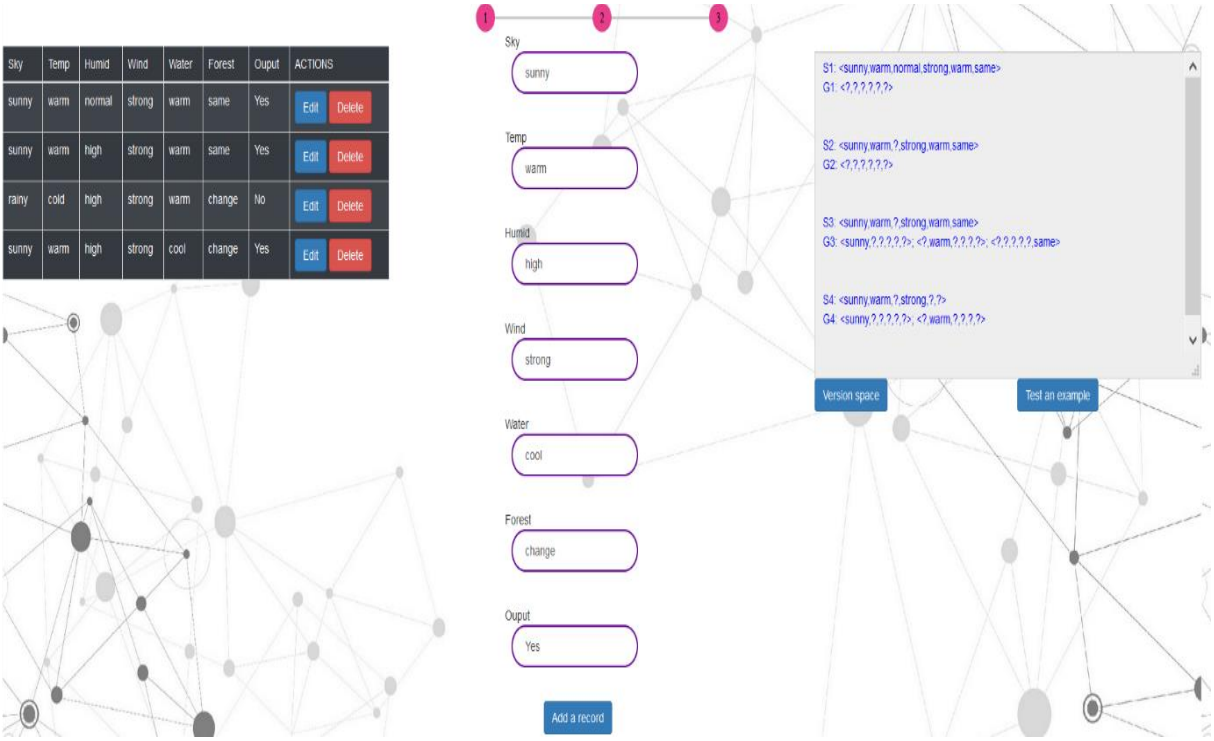


4.5.2.4 Features:

4.5.2.4.1 Add to table feature:

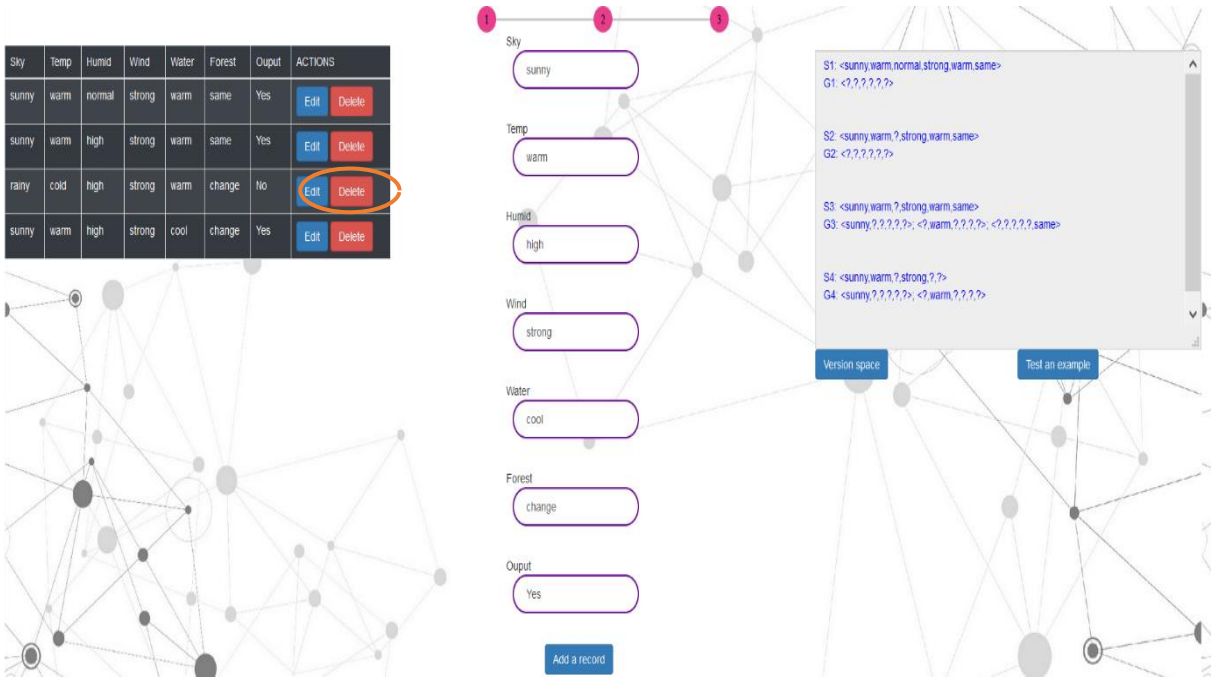


When a new instance is added to the table, the border sets S and G change accordingly. From the above picture, it can be seen that by adding the first example, S1 and G1 get initialized. Furthermore, every row in the table can be deleted or edited using the “Edit” and “Delete.” buttons.

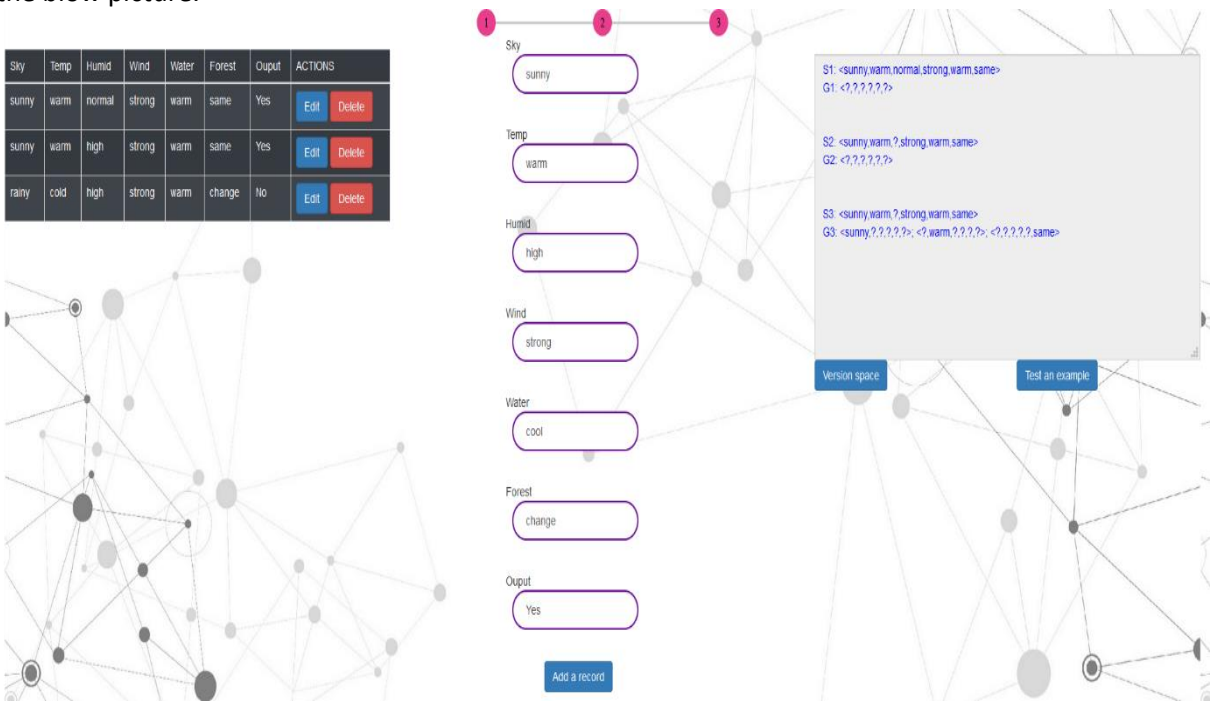


This picture shows how the page should be looking after adding all training examples to the table. The text area on the right side displays the border sets' changes by adding every new example.

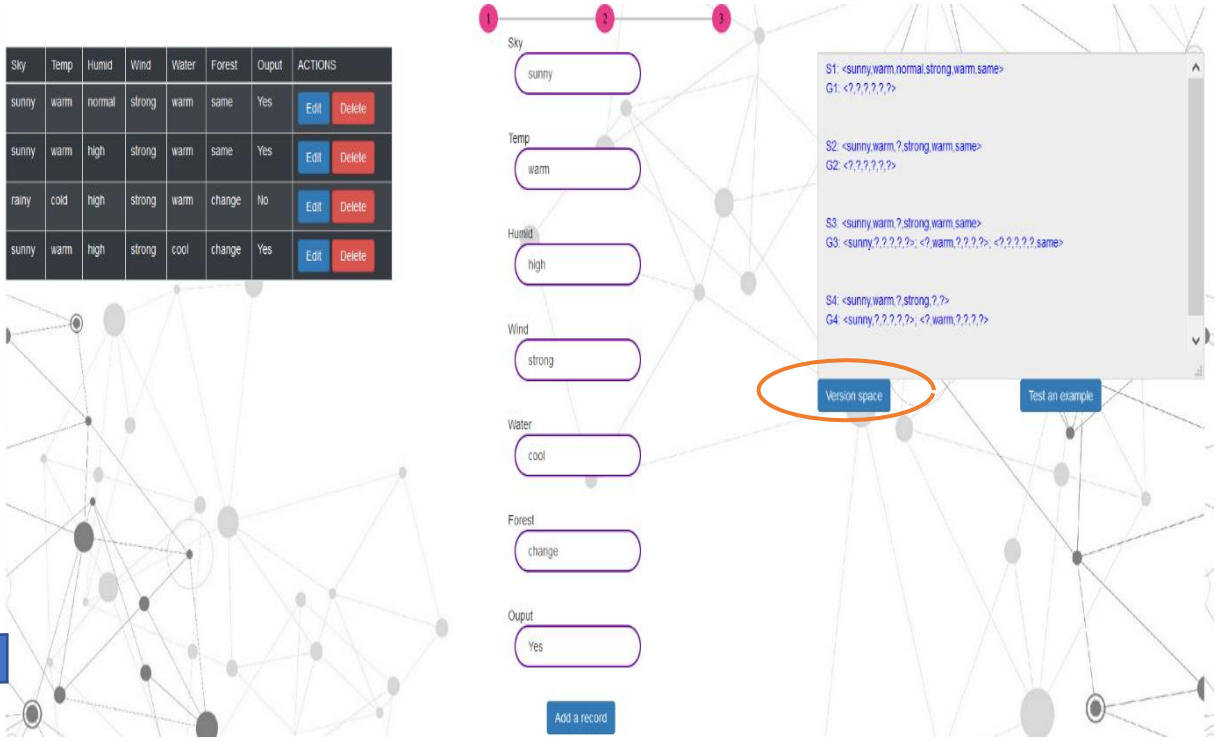
4.5.2.4.2 Delete row Feature:



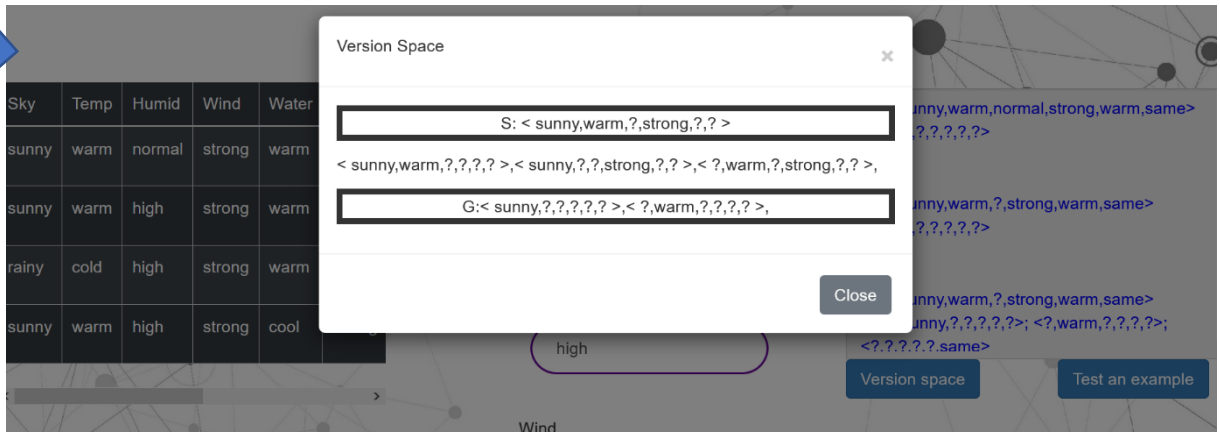
When the user deletes a training example, he/she can click on the delete button at the observed example. After deleting the example, the border sets S and G should be altered accordingly as shown in the blow picture.



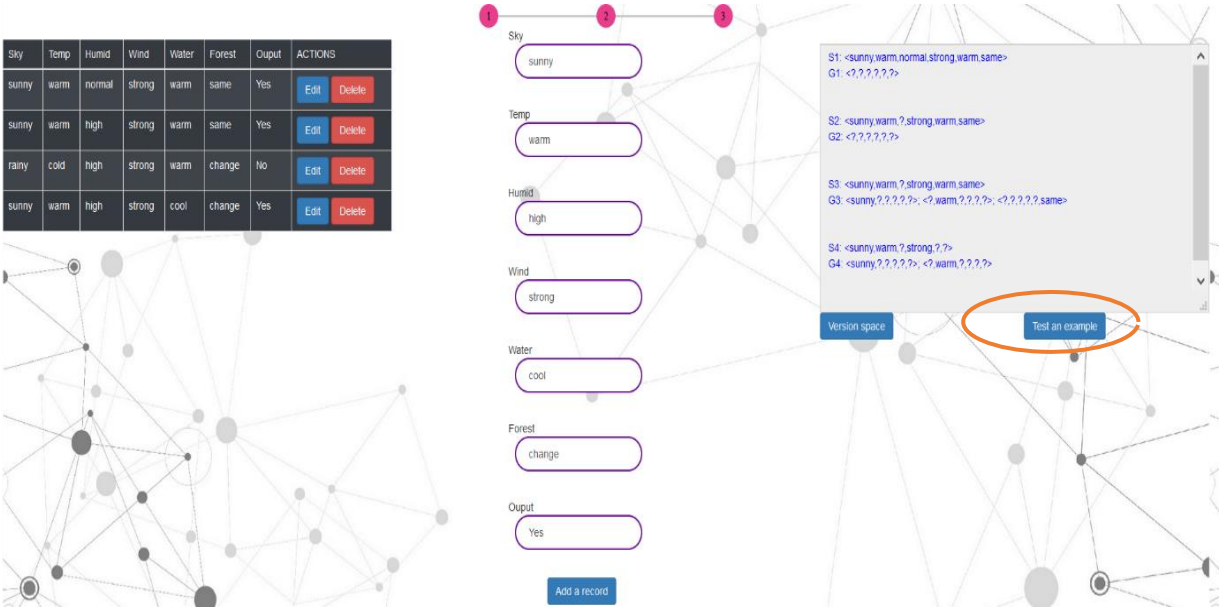
4.5.2.4.3 Display version space feature:



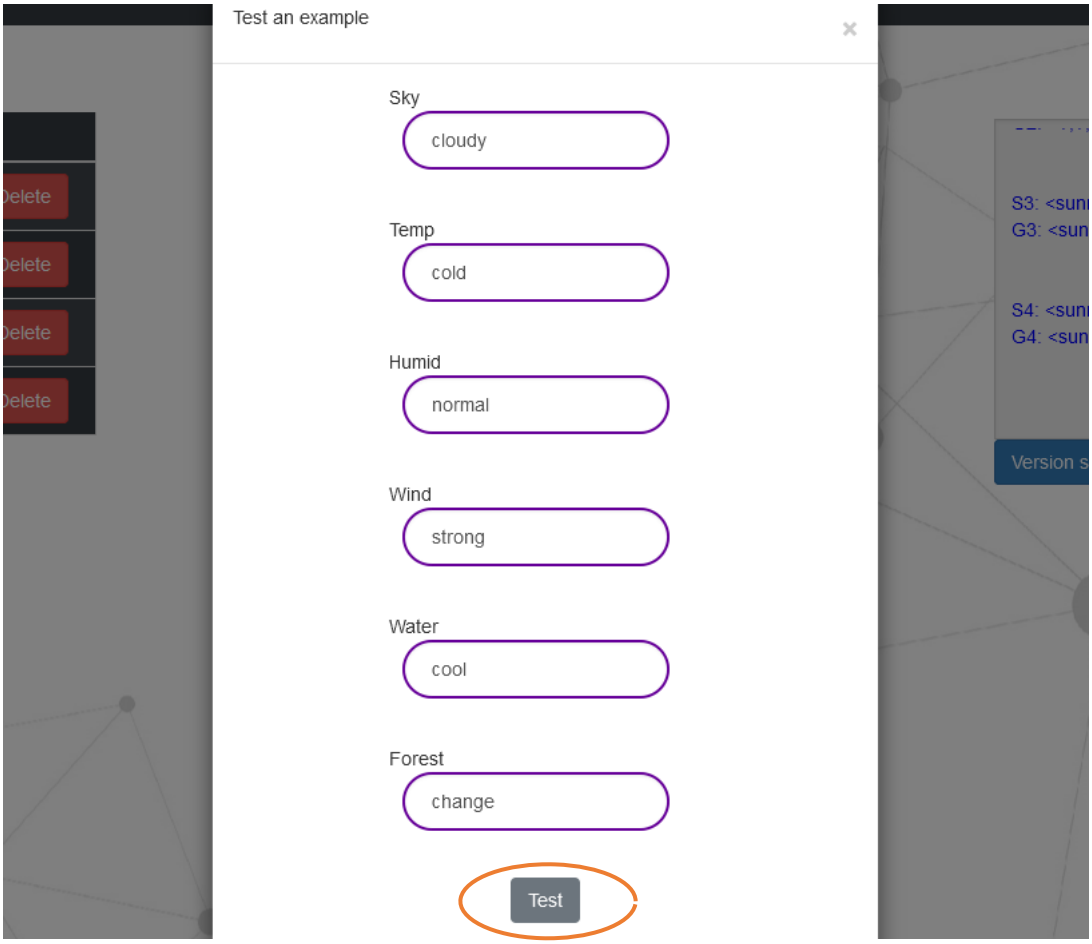
When the version space button is clicked, the version space will be displayed. The border sets S and G are bordered with bold black color. Between S and G are all hypotheses that cover the training examples.



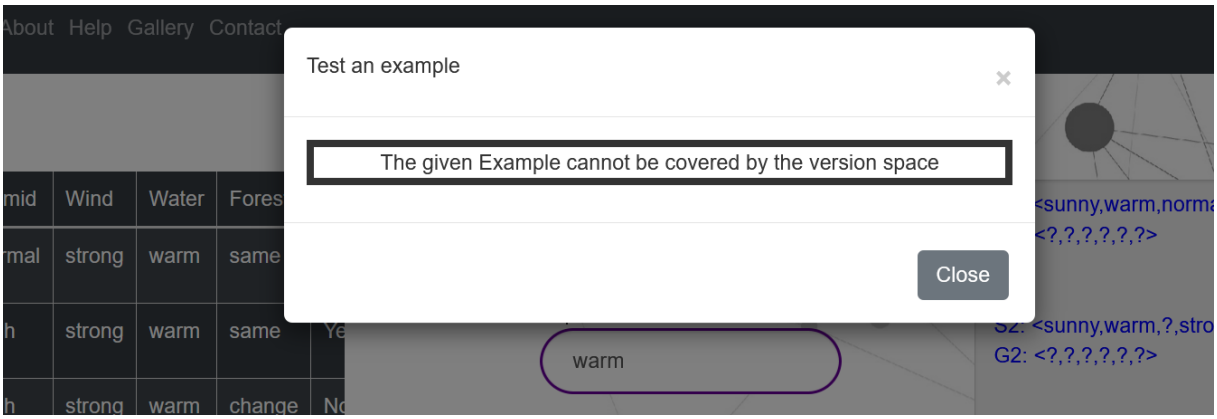
4.5.2.4.4 Test an example feature:



After the version space is generated, the user can test whether a specific example can be covered by the version space or not. When the "Test an example" button is clicked, a form of input fields will be displayed, where the user can enter the values of the test example.



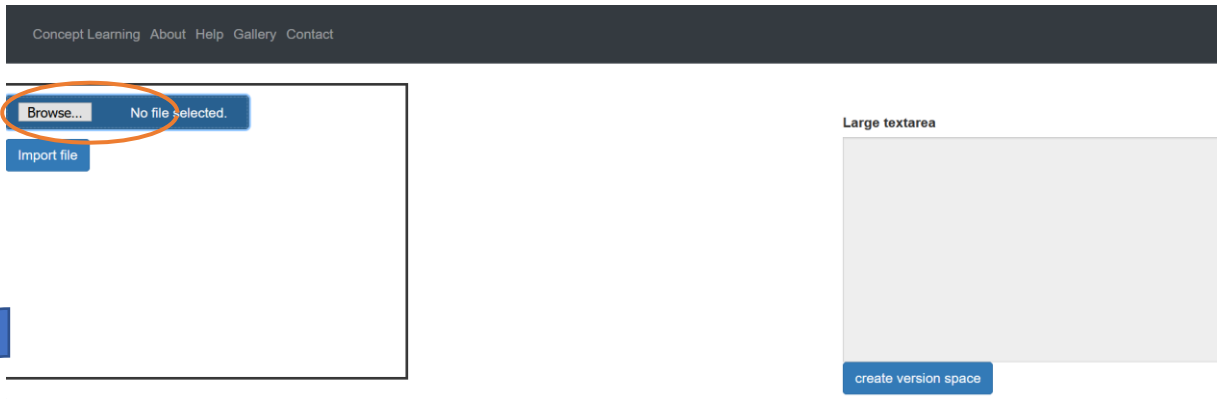
Once alle values of the test example are given, then it can be tested, whether the given example is covered by the version space or not. The given example cannot be covered by the user.



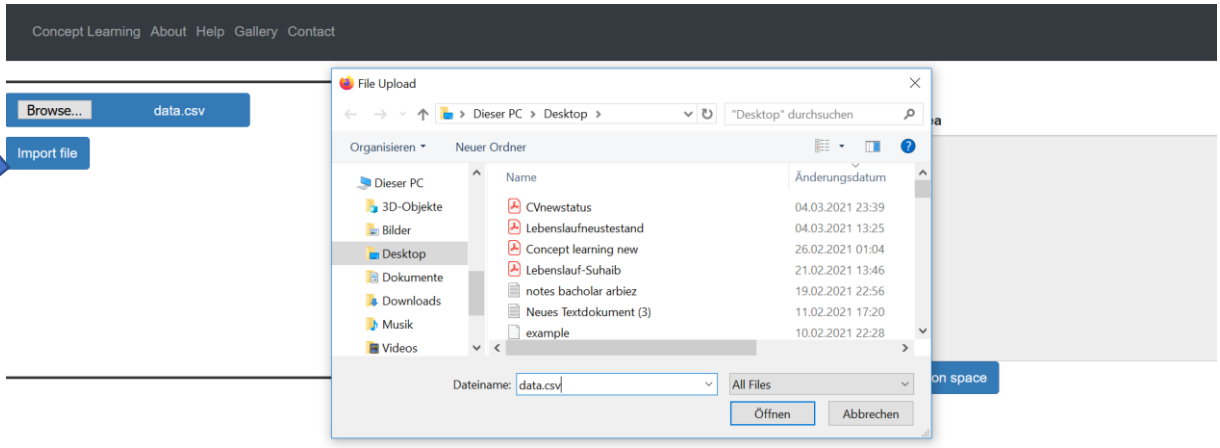
4.5.3 UploadTable page:

This page enables the user to upload a training table from his/ her computer and run the candidate elimination algorithm to generate the version space.

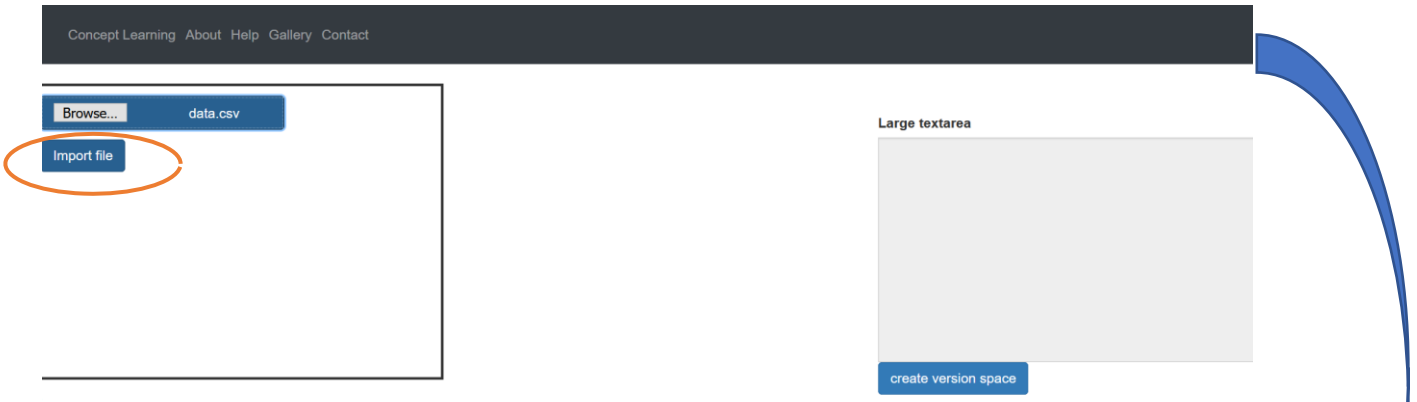
4.5.3.1 First step:



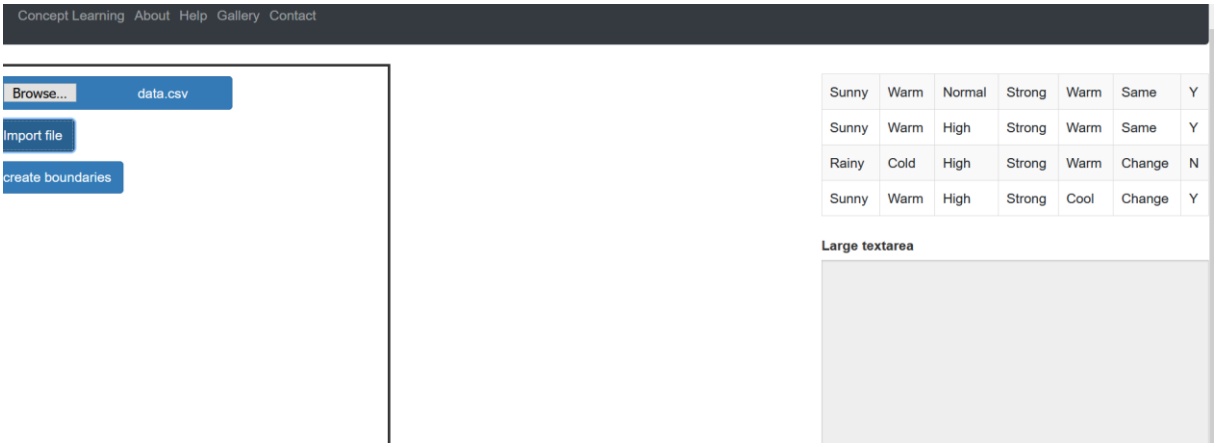
When the user clicks on the “Browse” button, he/she will have the ability to navigate through the computer and chose the file that contains the training data. It is very important that the chosen file is extended with (.csv (comma separated value)), otherwise an exception will be thrown and an alert message will be shown to the user.



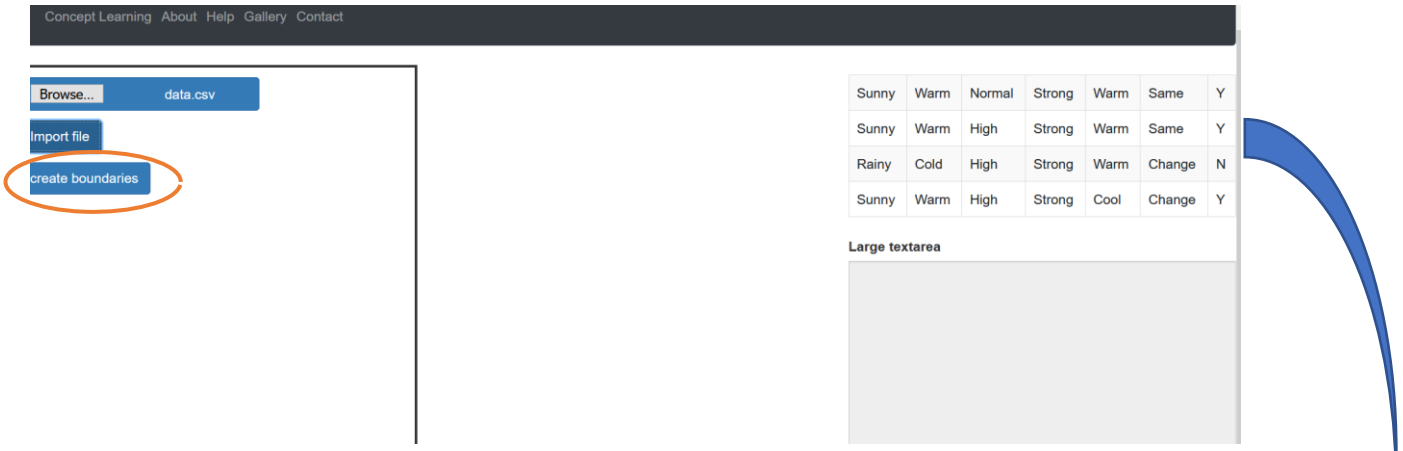
4.5.3.2 second step:



Once the user has chosen the file that contains the training table from the computer, he/she can import it into the website by clicking on the “import file” button. The table will be displayed on the right side of the website containing all data.

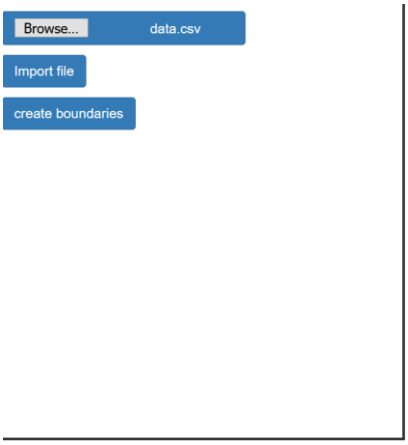


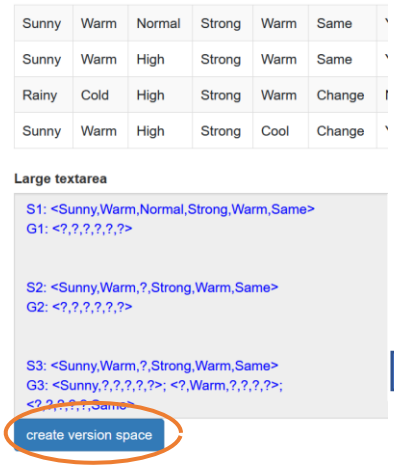
4.5.3.3 Third step:



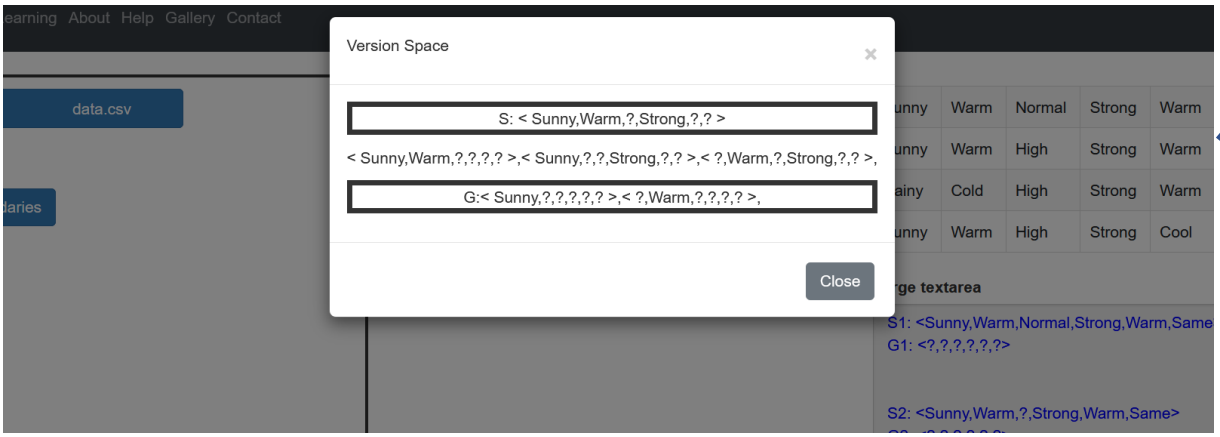
When the user clicks on the “create boundaries” button, the version space boundaries will be created and displayed in the text area.

4.5.3.4 last step:





When the “create version space” button is clicked, the version space will be displayed with its border sets and the hypotheses that lie between them.



5.Results

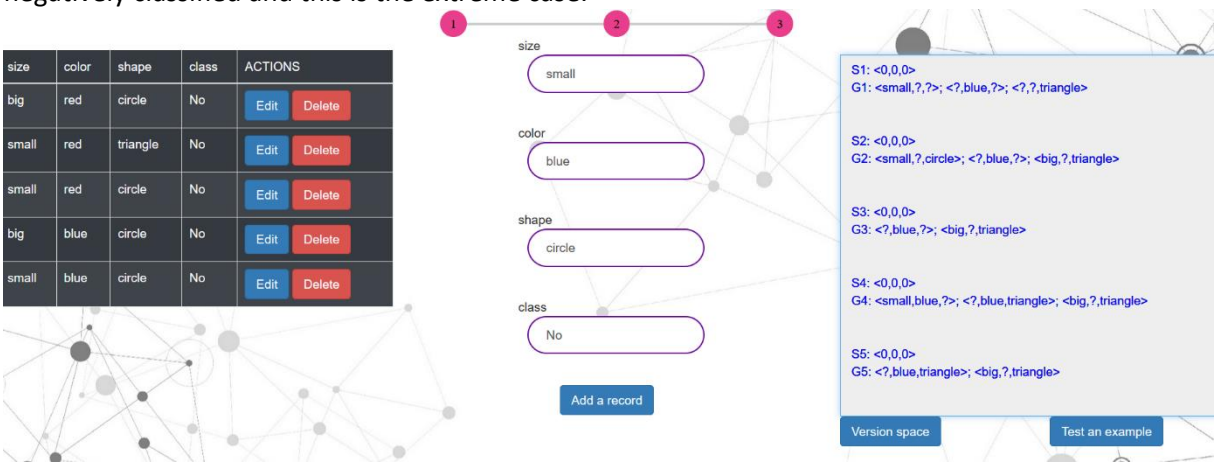
As the functionality of the software depends most upon the training data, there is not a good metric that can be used to evaluate the results. However, different training data sets can be submitted to the software, and then it can be observed how the software performs on each of the submitted training data sets. Furthermore, the submitted training data sets must be different, Particulary, it is not practical to upload multiple data sets, where they are equally structured and have the same order. 4 different data sets aimed to be submitted or uploaded to the website. The first data set has all records negatively labeled, whereas the second data set has all records positively marked. The third data set has its first training example negatively classified and then it contains mixed labeled training examples. The last data set has its first record positively labeled and the rest training examples are mixed classified. The software should be examined with the different data sets described above. It can be claimed, that there are a lot of data sets that can be used to test the software to come up with a good evaluation. Nevertheless, testing all different data sets is not possible, as this can take a lot of time and effort to collect them and then evaluate the effectiveness of the software. Moreover, Due to the drawbacks of the candidate elimination algorithm that are discussed in section 3.2.4.2, the software is not expected to be 100% accurate, as many concepts cannot be learned applying the candidate elimination algorithm.

Four different data sets will be submitted to the software to test its efficiency:

1. All records are negative classified
2. All records are positive classified
3. The data set starts with negative classified records and then mixed classified records.
4. The data set starts with positive classified records and then mixed classified records.

5.1 Case 1:

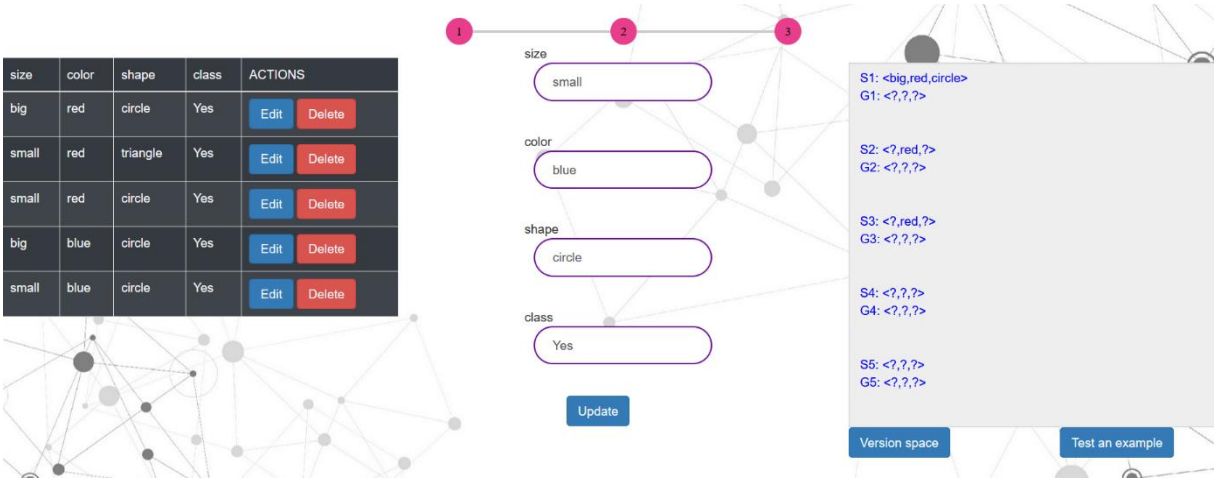
Firstly, it is very essential to evaluate, how the software runs on a data set, where all instances are negatively classified and this is the extreme case.



It can be seen from the above picture that all examples are negatively classified. The results(border sets) were correctly calculated. However, it cannot be said that the algorithm is operating 100% perfectly on all data sets where all instances are negatively classified, as we cannot run the algorithm on all data sets, but using one example, it can be made sure that the algorithm functions well when it comes to the extreme case (all examples are negatively classified).

5.2 Case 2:

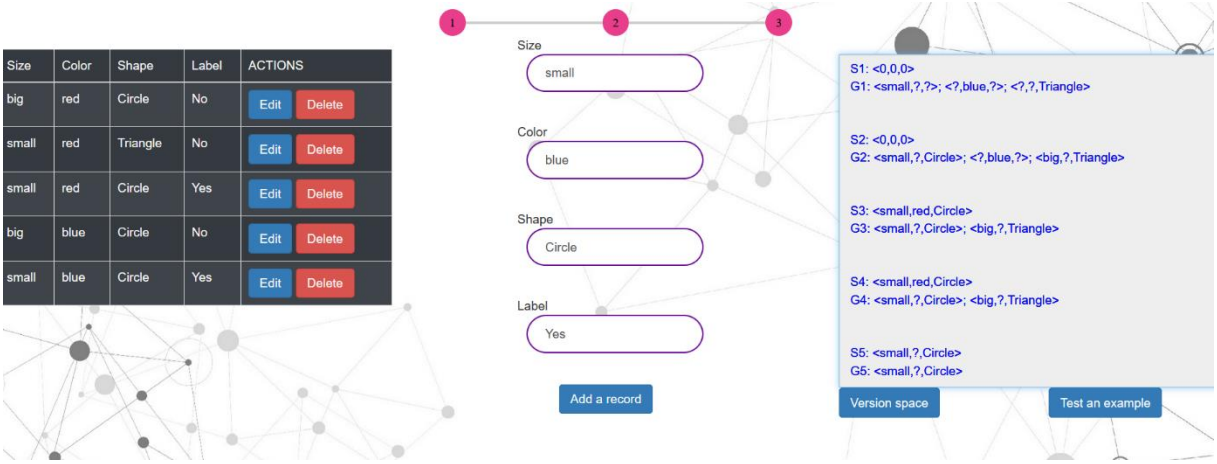
After examining the software on a data set, where all examples have a negative label, it is vital to test it on a data set, where all examples have a positive label.



It can be noticed from the above picture that all examples are positively classified. The results (border sets) were correctly determined. Though, it cannot be deserved that the software functions 100% quite on all data sets, where all instances have a positively labeled, but applying one example(case) can have an indication, how the algorithm performs on a data set, where all records are positively classified.

5.3 Case 3:

Additionally, the software must be verified on a data set, that starts with a negative example and then mixed classified examples. It must be evaluated, how the algorithm operates on such a data set.

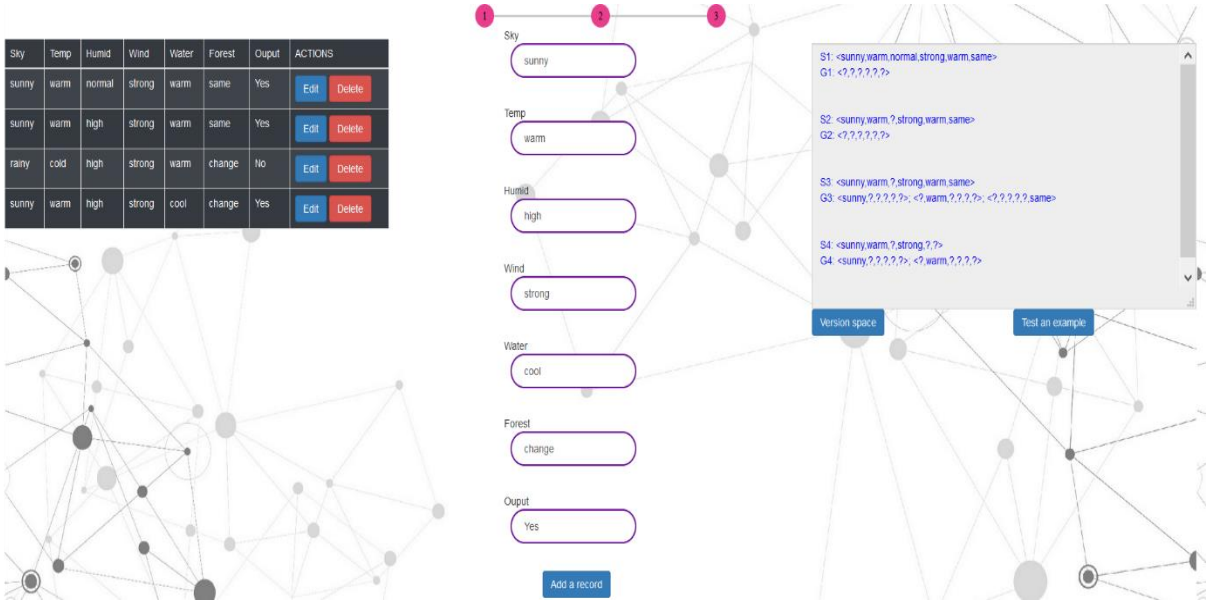


It can be inspected from the above image that the results (border sets) are calculated as expected. However, it should not be assumed that the software functions 100% accurate on all data sets that start with a negative record and then mixed classified records, but it can be claimed that for a tested

dataset that starts with a negative and then mixed labeled records the software operated as expected.

5.4 Case 4:

The last testing case is considered for a data set that starts with a positive labeled record and then with mixed labeled records.



The above picture shows the results (border sets) considering the training table of the left-hand side and it can be said that the results are calculated as expected and the results are accurate.

5.5 final table:

Case	Correct
1	Yes
2	Yes
3	Yes
4	Yes

After examining the four different scenarios, it can be declared that the software performs well sofar on the offered training tables. Due to the dependency on the training tables, it cannot be claimed that the software is 100% errorfree when it comes to future training data sets, but according to the results retrieved after testing several training data sets, it can be assumed that the software functions well on the submitted training data.

6.Conclusion and Future work

This study presents a deep analysis and investigation of concept learning. Moreover, different learning algorithms like Find-S and candidate elimination algorithms were discussed and investigated along with their advantages and drawbacks. Furthermore, an extensive discussion of the software, which was created to learn new concepts using the candidate elimination algorithm, was formulated. The experimental results show that the software performs accurately on different training data sets.

Concepts can be learned more efficiently using the different machine learning models like logistic regression that overcomes the drawbacks that candidate elimination algorithm cannot get rid of.

References

[1] Mitchell, T. M. *Machine Learning*. New York: McGraw-Hill Science/Engineering/Math, 1997.Print.

[2] Hirsh, H. (1994). Generalizing version spaces. *Machine Learning*, 17(1), 2-13.

[3] Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*. 18(2), 3-4.

[4] Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule learning. *Fifth International Joint Conference on AI* (pp. 305-310). Cambridge, MA: MIT Press.

[5] Mitchell, T. M. (1979). Version spaces: *An approach to concept learning*, (Ph.D. dissertation). Electrical Engineering Dept., Stanford University, Stanford, CA.

[6] Beierle, Christoph., & Isberner-Gabriele, K. *Methoden wissenbasierter Systeme*. Wiesbaden: Springer Fachmedien GmbH, 2019. Print.

I certify that I have prepared the present work independently and only using the sources and the given material.
In particular, literal or analogous quotations are marked as such. I am aware that an infringement can also subsequently lead to the withdrawal of the graduation. I certify that the electronic copy matches the printed copies.

place **Date**

signature