Improving conflict resolution in version spaces for precision agriculture

ADRIAN GROZA
Technical University of Cluj-Napoca
Department of Computer Science
Memorandumului , 400028 Cluj-Napoca
ROMANIA
Adrian.Groza@cs.utcluj.ro

IULIA UNGUR
Technical University of Cluj-Napoca
Department of Computer Science
Memorandumului 28, Cluj-Napoca
ROMANIA
Iulia.Ungur@isg.cs.utcluj.ro

Abstract: We developed a plant monitoring system that uses machine learning to classify environment conditions as favorable or not for plant development. The decision is taken based on six features whose values are measured from sensors: light, temperature, vibrations, soil humidity, rain quantity and vertical distance. Aiming to assure transparency in the classification decision, we used a modified version of the version space algorithm. We adapted the version space algorithm to deal with situations when hypotheses do not agree on a single decision. As a result, 20% from the instances unclassified by the version space were classified by our enhanced version space algorithm. The developed tool is available online as an open-source project.

Key–Words: Precision agriculture, Knowledge in learning, Version space algorithm, Sensor-based reasoning

1 Introduction

A version space is the set $\mathcal{L}$ of all classifiers (hypotheses) expressible in a language that correctly classify a dataset. Version space algorithm is based on candidate-elimination techniques. At each step, hypotheses inconsistent with the training examples are removed from the version space. Two boundary sets are used: $\mathcal{S}$ for specific hypotheses and $\mathcal{G}$ for general ones. Initially $\mathcal{G} \equiv \top$ and $\mathcal{S} \equiv \bot$. With each training example, $\mathcal{S}$ is generalised and $\mathcal{G}$ is specialised.

For each classifier $g_i \in \mathcal{G}$ and $s_i \in \mathcal{S}$, the new example may be a false positive or a false negative. There are four situations: First, false positive for s_i means that s_i is too general. With no consistent specializations of s_i (by definition), the hypothesis s_i is removed from $\mathcal{S}$. Second, false negative for s_i means that s_i is too specific. Hence, s_i is replaced by all its immediate generalizations, given that they are more specific than some member of $\mathcal{G}$. Third, false positive for g_i means that g_i is too general. Hence, g_i is replaced by all its immediate specializations, provided they are more general than some member of $\mathcal{S}$. Forth, false negative for g_i means g_i is too specific, but there are no consistent generalizations of g_i (by definition). Consequently, g_i is removed from $\mathcal{G}$.

These four operations are repeated for each new example, until one of three things happens: First, at the end of the training, only one hypothesis may remain in the version space $\mathcal{L}$ [2]. This hypothesis will be used to classify unlabeled instances. Second, the version space $\mathcal{L}$ collapses. That is, either $\mathcal{S}$ or $\mathcal{G}$ becomes empty, meaning that there are no consistent hypotheses for the current training set. Third, at the end of the training, several hypotheses may remain in the version space. This means the version space represents a disjunction of classifiers.

Our focus here is on this third situation when more than one learner $l \in \mathcal{L}$ remains to classify unlabeled data. For any new example, if all classifiers agree, the algorithm returns the classification. If the classifiers disagree, a conflict resolution strategy is employed. One common option is to apply the majority vote. Instead of majority voting, we propose a conflict resolution method that exploits the white-box model of the version space learning algorithm.

Version space provides a hierarchical representation of the knowledge accumulated. Its advantages include speed of computation (close to linear) and the ability to describe all possible hypotheses on the learning examples given that the data is correct and consistent [7]. Its main constraint is that the training set should be consistent. This limitation restricts the usage of version spaces in real life scenarios characterised by noisy, continuous or inconsistent data.

Hence, our goals here are twofolds: 1) to identify and validate a conflict resolution method for version spaces, respectively 2) to enhance version spaces for real-life applications characterised by noisy, continuous or inconsistent data.

2 Running Scenario

Plant survival represents an important issue on sustainability due to the constant change of environment features. This change represents a source of noisy, continuous and, in certain cases, inconsistent data.

We developed a device, which can detect and register the mean variables from its surroundings such as light, temperature, vibrations, soil humidity, rain quantity and vertical distance. This data is presented in its raw form to an online platform that classifies it in certain classes of abstraction, converts it in application models - as sensors for the environment features and predictions for learning sequences and analyzed examples, as then to proceed with the version space creation[1].

For a better understanding of the algorithm and its results, two major features have been developed: 1) manual insertion of environment data and 2) hypotheses sets viewing. These features allow for an in depth analysis of the data and an accurate measurement for the actual improvement that we brought to the algorithm.

The process begins with a classification of the raw data. Based on a set of sensors, the device registers data and creates a mean of the last n entries. The n value is set within the device through a potentiometer (between 30 seconds and 2 hours). Once the mean for each variable has been created, it is transmitted via direct connection to the interface where, each variable is classified using a fuzzy membership function. The function calculates the degrees to which the measured value belong to predifined fuzzy set. If a value is contained in two fuzzy sets max is applied between the values.

In the next step, we apply machine learning based on the version space algorithm. The analysis begins by constructing the hypotheses sets. Each hypotheses $\mathcal{S}$ and $\mathcal{G}$ is constructed following exactly the rules of the version space algorithm (further details in the next section). Since a form of quantification is needed to determine which feature is more important than the other, feature selection methods are used to compute a score. This score is based on the frequency of which the feature influences the final outcome of the prediction - if a change brought in the temperature caused the plant to die within a short period of time, the temperature will have a higher feature score on its influence.

Next, the matching degree between the sets and the prediction is computed. That is the the degree of which the current instance matches the hypotheses sets. We have a match if a feature value varies in a range of 30% from the sets feature value times the feature's influence. This operation was useful to reduce the degree to which the sets are changing during the learning phase.

[1]The system is available at https://plant-predictor.herokuapp.com/. The interested user can obtain the source code from https://github.com/IuliaUngur/plant-predictor.

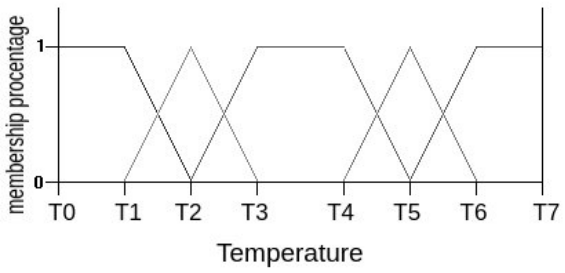


Figure 1: Data discretisation using fuzzification.

With this degree computed, the classical version space algorithm takes a decision: favorable conditions ($\oplus$), unfavorable ($\ominus$) or uncertain ($\oslash$). If the assigned label is uncertain, we apply our generalised version space method, aiming to reduce the number of instances which remain unclassified by the classical version space algorithm.

3 Improving the version space algorithm

3.1 Data discretisation

We apply a simple method for discretisation based on fuzzification [10]. We calculate the membership of sensor data to a predefined class. For example, the temperature sensor returning a value $\mathcal{X} \in [-40, 120](^{\circ}\mathcal{C})$ will be asserted to the following classes $Freeze$, $Cold$, $Cool$, $Warm$, Hot. The domain is given by the sensor specifications. That is in our case $T_0 = -40$, $T_7 = +120$ (Fig. 1). Based on the specific membership function (trapezoid or triangle), each sensor reading is attached a class based on $max(\mathcal{X}_0...\mathcal{X}_i)$, where $i \in [Freeze, Cold, Cool, Warm, Hot]$ and $\mathcal{X}_i \in [0, 1]$.

3.2 Hypotheses construction

Version space algorithm is a machine learning approach that uses inductive concepts to make assumptions based on training set. This concept represents the version space hypotheses set $\mathcal{H}$, which includes both the specific $\mathcal{S}$ and general $\mathcal{G}$ hypotheses. An hypothesis $h \in \mathcal{H}$ is a conjunction of constraints on the features of the training set. Each hypothesis needs to be consistent with all examples from the training set.

Each positive instance represents a favorable survival condition for the plant. The specific hypotheses set $\mathcal{S}$ is initialized by the first positive example in the training set. Hypothesis inconsistent with the positive example are removed from $\mathcal{G}$ (line 5 in Algorithm 1). Inconsistency occurs when a classifier in $\mathcal{G}$ returns a different classification decision with the positive example. Likewise, $\mathcal{S}$ is generalized to include the example. That is, hypothesis that classify the example as negative are removed from $\mathcal{S}$ and replaced by the minimal generalization that satisfy the condition (line 11 in Algorithm 1).

Each negative example represents an unfavorable condition for plant survival. The hypothesis from $\mathcal{S}$ that classify the example as being positive are eliminated and $\mathcal{G}$ is specialized to exclude the negative example. No hypothesis from $\mathcal{G}$ should be more specific than any classifier from $\mathcal{S}$. To ensure that the sets are consistent with one another, non-minimal hypotheses are removed from $\mathcal{G}$ (line 20 in Algorithm 1).

Algorithm 1: Hypotheses Construction.

Input: $\mathcal{P}$ - set of training examples;
Output: $\mathcal{G}$ - set of general hypotheses;
$\mathcal{S}$ - set of specific hypotheses;

```
1  G ← ⊤
2  foreach p ∈ P do
3      if p = positive example then
4          if G(p) = negative example then
5              G ← eliminate(G(p))
6          foreach s ∈ S do
7              if s(p) = negative example then
8                  gs ← generalize(s(p));
9                  if gs ∉ G then
10                     s ← gs
11         eliminate(S, minimal)
12     else
13         if S(p) = positive example then
14             S ← eliminate(S(p))
15         foreach g ∈ G do
16             if g(p) = positive example then
17                 sg ← specialize(g(p));
18                 if sg ∉ S then
19                     g ← sg
20         eliminate(G, maximal)
```

3.3 Feature selection

The relevance of each attribute in the training set is given by the influence that it has in deciding the label. If a feature value influences drastically the outcome of the plant, that attribute will have a much higher score than the rest of the attributes. For instance, in our scenario, a value of $55°\mathcal{C}$ for temperature will rise a negative label no matter the values of the other features.

Feature selection techniques are grouped roughly into three categories: wrappers, embedded methods and filters [4]. Wrapper methods perform a classification using the features selected as relevant and then evaluate the classification. Embedded methods introduce a penalty to reduce the level of variation of a model. That is, adding extra bias to regularize the overall cost. Filter methods select a subset of features based on a certain threshold. That is, filter methods rely only on the characteristics of the training set and they are not influenced by the learning algorithm used. One such filter method is the information gain of an attribute which returns the quantity of information that a feature holds considering the result. It measures the difference in information between the cases when the value of an attribute is known against the cases when it is unknown. Information gain is the method used in our experiments for feature selection.

3.4 Example analysis

We compute the levels of matching between the latest prediction received, that doesn't have a result attached, and each of the hypotheses sets $\mathcal{S} \in \mathcal{H}$ and $\mathcal{G} \in \mathcal{H}$. These levels are calculated given the features inclusion in the given hypothesis. For each hypothesis $h \in \mathcal{H}$ we compute to what degree the value of the prediction feature $\mathcal{X}_i$ diverges from the hypothesis feature h_i. This degree is calculated as the value of the prediction feature taking into consideration its influence - given the total attribute score received from the information gain feature selection, and an error of 30% (lines 4-9 in Algorithm 2).

To reduce the number of possible hypotheses created considering a consistent training set, we added the error margin to avoid situations in which very similar values are added to the version space. Otherwise, considering a number of five classes for each sensor, with a total of six sensors in the application, the number of distinct instances would have been $5^6 = 15625$, the number of distinct concepts 2^{5^6}, and thus a total of $2^{2^{15625}}$ hypotheses in the version space.

After the matching values $\mathcal{MS}$ and $\mathcal{MG}$ for each of the hypotheses sets $\mathcal{S}$ and $\mathcal{G}$ have been computed, they are analyzed together. Since both levels have to

be consistent with one another, any sign of inconsistency (between $45\% - 65\%$) will be considered as an uncertain outcome for which, an uncertain outcome analysis will be computed (lines 17-20 in Algorithm 2).

Algorithm 2: Analyzing example based on hypotheses sets.

Input: $\mathcal{P}$ - training set;
$\mathcal{S}$ - set of specific hypotheses;
$\mathcal{G}$ - set of general hypotheses;
x - example to be analyzed;
Output: $\mathcal{U}$ - set of uncertain combinations with success rate;
l - label for instance x;

1 $Scores \leftarrow FeatureSelector.(P))$
2 **foreach** $s \in \mathcal{S}$ **do**
3 **foreach** *attribute* $a \in \mathcal{X}$ **do**
4 **if** $s_a - x_a < 0.3\% * Scores(a) * x_a$ **then**
5 $matches\_S \ll add\_match(a)$;
6 $MS \leftarrow matchingPercentage(matches\_S)$
7 **foreach** $g \in \mathcal{G}$ **do**
8 **foreach** *attribute* $a \in x$ **do**
9 **if** $g_a - x_a < 0.3 * Scores(a) * x_a$ **then**
10 $matches\_G \ll add\_match(a)$;
11 $MG \leftarrow matchingPercentage(matches\_G)$
12 **if** $MS \wedge MG < 0.45$ **then**
13 $l = \ominus$
14 **else**
15 **if** $MS \wedge MG > 0.65$ **then**
16 $l = \oplus$
17 **else**
18 $analyzer \leftarrow AnalyzeUncertainOutcome$
19 $r \leftarrow analyzer.result$
20 $\mathcal{U} \leftarrow analyzer.uncertainSet$

3.5 Handle uncertain instances

Uncertainty appears when either the prediction to be analyzed matched only one of the version space hypotheses or if the degree with which it matches barely is of any significance. We combine the hypotheses with prediction, generating all valid possibilities and voting for a final result based on the degrees of truthfulness of the combinations.

The analysis starts by extracting the average values from the instances predicted as favorable for plant development. Then we construct all the valid combinations between the sets and the current instance x. If a prediction contains features that already match on the sets, they will be kept further on. Thus only different features are combined with the matching features from the hypotheses set $\mathcal{H}$ when constructing the combinations.

The difference between the sets is computed, obtaining $\mathcal{H} \setminus x$ and $x \setminus \mathcal{H}$, where x is the instance that is being analyzed, and $\mathcal{H}$ is either one of the constructed hypotheses sets. Using the intersection $x \cap \mathcal{H}$ the base of each combination in constructed. If x contains a feature that is noted as $null$ in the hypotheses set, it will be merged in the base combination construction, as seen in Table 1. Using this base and the remaining features of the prediction that have not already been included, we construct all combinations resulting in new prediction sets. For each prediction constructed we compute the levels of matching with the initial hypotheses sets, but taking into consideration the average values as well.

For instance, combination C_1 is created as follows. The common values between $\mathcal{S}$ and $\mathcal{P}$ are temperature (HOT) and humidity (33%). The value for the $light$ attribute is null in the set $\mathcal{S}$, so the value $Light = DARK$ will be kept in the construction of the combinations. The remaining three features $Distance$, $Rain$, $Vibration$ whose values do not coincide with those of hypothesis $\mathcal{S}$ are used to complete the combinations. Each occurring value is used and permuted to have full coverage of the possibilities. Then, the combination will be reintroduced in the system and analyzed. If the result is above 50%, the example x is considered favorable for plant development. Otherwise x is labeled as a negative example with a percentage of $(50 - x) * 2$. If the prediction matches exactly 50%, example x remains unlabeled.

Final phase represents the voting process, in which the maximum value is computed from each combination result from both positive and negative outcomes (line 16 in Algorihm 3). If the value of the calculated maximum is a positive outcome and the level of uncertainty is low, then the label will be positive (lines 18-19). The process for the negative result is the same: if the uncertainty level is low, the prediction will have a negative outcome (lines 21-22). If the uncertainty is high, the decision cannot be taken (line 24), and the example remains unclassified .

Finally, the uncertain set is allocated with the

Table 1: Uncertain combinations.

Op	Temperature	Light	Dist.	Rain	Humidity	Vibration
$\mathcal{P}$	Hot	Dark	Close	Condense	33%	22%
$\mathcal{S}$	Hot	null	Nearby	Dry	33%	33%
$\mathcal{S} \setminus \mathcal{P}$	null	null	Nearby	Dry	null	33%
$\mathcal{P} \setminus \mathcal{S}$	null	Dark	Close	Condense	null	22%
$\mathcal{S} \cap \mathcal{P}$	Hot	null	null	null	33%	null

#	Temperature	Light	Dist.	Rain	Humidity	Vibration	Label
C_1	Hot	Dark	Nearby	Dry	33%	33%	$\oslash$
C_2	Hot	Dark	Nearby	Dry	33%	22%	$\oslash$
C_3	Hot	Dark	Nearby	Condense	33%	33%	$\oslash$
C_4	Hot	Dark	Nearby	Condense	33%	22%	$\oslash$
C_5	Hot	Dark	Close	Dry	33%	33%	$\oslash$
C_6	Hot	Dark	Close	Dry	33%	22%	$\oslash$
C_7	Hot	Dark	Close	Condense	33%	33%	$\oslash$
C_8	Hot	Dark	Close	Condense	33%	22%	$\oslash$

valid hypotheses and example combinations that have attached the computed result.

4 System architecture

Our plant monitoring systems works in two operating modes: simulation and live. Simulation mode is used to test our modified version space algorithm. In this mode, the training set is provided from the user. Live mode continuously read measurements from the Arduino sensors. The hardware writes in a JSON file every m minutes, m being set from a potentiometer directly on the device. The minimum time is set to 30 seconds. The front end component performs a request and if the server discovers that a change has been made, it processes the set and passes it through the algorithm. Once the device is connected and functional, it waits the start signal from the user given by a remote control. Within a given time, established by the user on a potentiometer ranging from 30 seconds up to 2 hours, the device records the sensor values and computes their mean. Only this mean value is send to the server, to ensure debouncing of the signals.

The front end has been implemented in React.js using the concept of components and states to create a dynamic interface. A DOM is created every time a new rendering operation of the component is requested. React creates a backup of the previous rendered DOM called virtual DOM, compares the two and re-renders only that components that are different. The server is implemented in Ruby with the RubyOnRails framework. Each time a request from the user is made, the server retrieves from a PostgreSQL database the desired information in a response format which will be handled by the React component.

The hardware component is developed using a Arduino UNO board, six sensors each representing the environment variables and an IR Remote used for activating the process. The Arduino component has been developed using the Arduino IDE and the JSON writing for transmitting the data to the server has been implemented using Processing.

Figure 2 depicts the hardware implementation in Arduino. The temperature sensor senses the temperature between the range of $-40°C$ and $125°C$ with an offset of $0.5°C$. The raindrops component detects water that completes the leads on the board. The wetter the board the more current will be conducted. The light component uses the amount of light detected to determine how much current to pass through the circuit. For the humidity module, water in the soil means better conductivity between the pads, resulting in a lower resistance and a higher reading. To compute the distance, the system sends out a burst of ultrasound and listens for the echo when it bounces off of an object. It detects 3-400 cm in front of the sensor. For the vibration component, when the sensor moves back and forth, a certain voltage will be created by the voltage comparator inside of it.

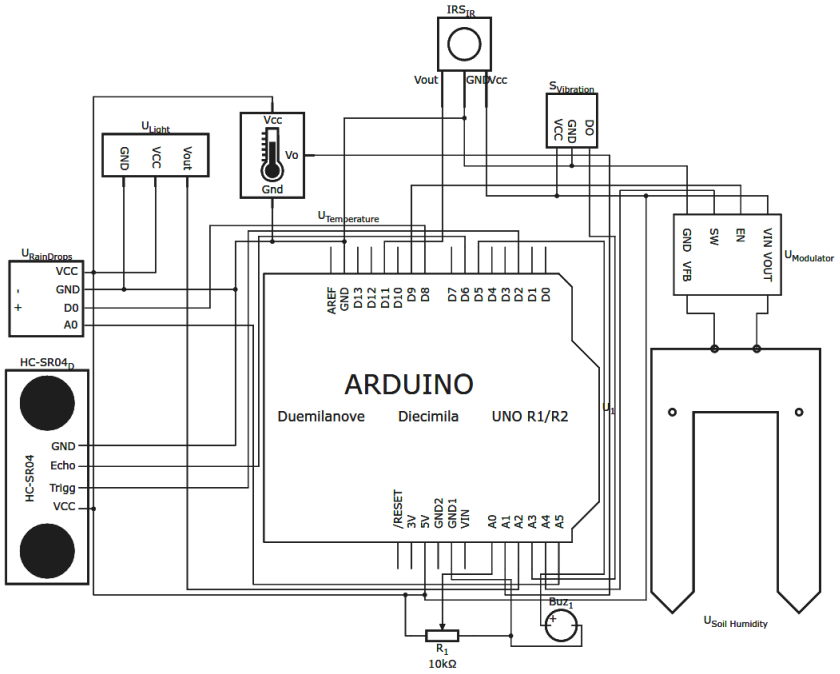
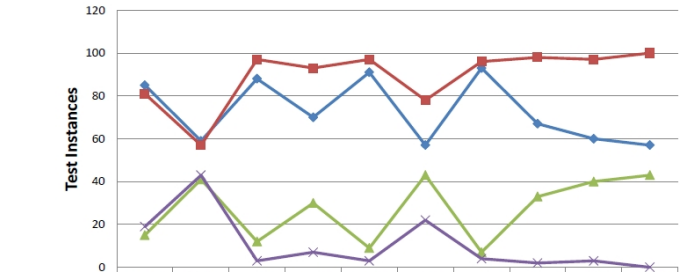


Figure 2: Electrical schema for the hardware components.

5 Running experiments

For training the device we used the szeged-weather dataset[2]. Since the data contain registering values everyday from 2006 till 2016, a sorting method was needed. We selected the first day of each month at 8:00am, while only the relevant features were kept. Analyzing average survival conditions of plants, we established that any value lower than $5°C$ or higher than $35°C$ for temperature will cause the plant to die, as well as abundant rain or constant vibrations. A sample of the learning data is depicted in Table 2.

The results in Figure 3 were obtained by varying the training set from 100 to 1000 instances. The test set contains 100 instances. The results on the tests showed that the algorithm can detect favorable conditions in more than 90% of the cases, but only 30% in case of unfavorable conditions. The improvement in reclassifying instances that the classical version space algorithm fail to label is 20%.



	100	200	300	400	500	600	700	800	900	1000
Survival Actual	85	59	88	70	91	57	93	67	60	57
Survival VS	81	57	97	93	97	78	96	98	97	100
Dies Actual	15	41	12	30	9	43	7	33	40	43
Dies VS	19	43	3	7	3	22	4	2	3	0

Figure 3: Running experiments.

6 Related work

Our conceptual problem was the classification decision in case that version spaces contained more than one classifier after all instances have been analyzed.

Sghair et al have proposed in [3] an algorithm for automatic detection of plant diseases. The classification is based on image processing to collect data for training. In our case, data is obtained from sensors.

Incremental version space [5] merging algorithm consists of each instance and all its classifiers defined as a version space. The intersection of version spaces corresponds to the union off all instances in the data set [7]. Therefore the problem is, which is the classification decision when the intersection is empty?

Generalizing version space with incremental merging allows version spaces to contain arbitrary sets of classifiers as long as they can be represented by boundary sets. Considering and keeping track of a set of candidate classifiers, not explicitly enumerated and maintained, allows 1^{st} example to be negative and

[2]The dataset is available at https://www.kaggle.com/budincsevity/szeged-weather

Algorithm 3: Computing labels for uncertain instances.

Input: $Scores$ - feature relevance;
$\mathcal{S} \subset \mathcal{H}$ - set of specific hypotheses;
$\mathcal{G} \subset \mathcal{H}$ - set of general hypotheses;
x - example to be classified;
Output: l - computed label for x;

foreach $h \in \mathcal{H}$ **do**
  foreach $set \in h$ **do**
    $\mathcal{DS}\_set \leftarrow set \setminus \mathcal{X}$
    $\mathcal{DX}\_set \leftarrow \mathcal{X} \setminus set$
    $\mathcal{I} \leftarrow set \cap \mathcal{X}$
    foreach $attr \in set$ **do**
      if $set(attr) = empty$ **then**
        $\mathcal{I} \ll \mathcal{DX}(attr)$
        $\mathcal{DX} \leftarrow eliminate(\mathcal{DX}(attr))$
    $\mathcal{C} \leftarrow generate\_combinations(\mathcal{DS}, \mathcal{I})$
foreach $c \in \mathcal{C}$ **do**
  foreach $attr \in \mathcal{X}$ **do**
    if $c(attr) \setminus X(attr) < 0.3 * Scores(attr) * \mathcal{X}(attr)$ **then**
      $matches\_c \ll$ add_match($attr$);
  $c \leftarrow matching\_procentage(matches\_c)$;
$max \leftarrow maximum(vg(C(\oplus)), avg(C(\ominus)))$;
$uncertainty \leftarrow calc(avg(C(\oplus)), avg(C(\ominus)))$;
if $max = \oplus \wedge uncertainty = low$ **then**
  $l \leftarrow \oplus$
else
  if $max = \ominus \wedge uncertainty = low$ **then**
    $l \leftarrow \ominus$
  else
    $l \leftarrow \oslash$

does not require the single-representation trick - that is for each instance there is a classifier that defines solely that instance [5]. Hong and Tseng [6] have proposed a system that handles exactly this: making trade-offs when choosing what examples to include in the learning phase. They present a similar method to the hereby present - the most consistent instances are stored in the hypotheses and a $count$ score is assigned to each instance. With a higher count in a hypothesis comes more inclusion or exclusion of a positive or negative training example. This count is called factor of including positive instances and ranges from 1 if the desire is to include positive examples and 0 to exclude them.

Herbrich et al have analysed in [8] the generalization error bound of classifiers within the version space. That is, classifiers ability to generalize an instance. In the same line, Sebag [9] controls the consistency and specificity of the classification of instances by delaying the search biases. Version Space being limited in computational aspects, they handle numerical attributes without discretisation - same aspect covered in our approach leaving vibration and humidity as percentages throughout the analyzing phase- and to eliminate the need for specific biases, this allowing noisy examples to be successfully handled.

Contiu and Groza in [1] have retrieved information from different sources to classify crops. They combine expert knowledge and supervised learning to classify crops into soybean, corn, cotton and rice. Ensemble learning is employed, where the ensemble contains a neural network, a decision tree and a support vector machine. In case of disagreement between learners, argumentation is used for conflict resolution. The features used for classification include green level moisture and $NDVi$. $NDVi$ stands for Normalized Difference Vegetation index and is calculated using $NDVi = (NIR - Red)/(NIR + Red)$, where NIR is the near infrared bands and Red the red bands. $NDVi$ is a indicator of how much land is in use and what changes appear. The conceptual research relates to conflict resolution in ensemble learning, resulting in argumentation reasoning. The system exploits both logic-based artificial intelligence and statistical learning. Expert knowledge is encapsulated within a rule-based system using Defeasible Logic Programming. Contiu and Groza have used features extracted from satellite images. Differently, our features are obtained from local sensors. After learning, Contiu and Groza have extracted knowledge in form of defeasible rules, aiming to increase transparency in the decision. Our version space algorithm also provides transparency, as the sets $\mathcal{G}$ and $\mathcal{S}$ can be translated into rules.

Table 2: Sample learning data set.

Light	Temperature	Vibration	Humidity	Rain	Distance	Label
Clear	5	21	77	Drizzle	11	$\ominus$
Clear	1	9	81	Flood	9	$\ominus$
Clear	1	16	74	Drizzle	10	$\ominus$
Clear	18	22	55	Drizzle	11	$\oplus$
Fog	16	17	64	Drizzle	11	$\ominus$
Clear	16	4	56	Drizzle	11	$\oplus$
Fog	20	12	86	Drizzle	11	$\ominus$
Clear	32	8	38	Drizzle	10	$\ominus$

7 Conclusions

We developed a plant monitoring system that uses machine learning to classify environment conditions as favorable or not for plant development. The decision is taken based on six features whose values are measured directly from sensors: light, temperature, vibrations, soil humidity, rain quantity and vertical distance. We adapted the version space algorithm to deal situations when hypotheses do not agree on a single decision. We used fuzzication such that the algorithm to handle inconsistent data or data with error measurements. These makes our method adequate for realistic scenarios.

Acknowledgements: The results presented in this paper were obtained with the support of the Technical University of Cluj-Napoca through the research grant no. 1996/12.07.2017, Internal Competition CICDI-2017.

References:

[1] Stefan Coniu and Adrian Groza. Improving remote sensing crop classification by argumentation-based conflict resolution in ensemble learning. *Expert Systems with Applications*, 64:269–286, 2016.

[2] Luc De Raedt and Stefan Kramer. The levelwise version space algorithm and its application to molecular fragment finding. *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, pages 853–859, 2001.

[3] MOSBAH EL SGHAIR, RAKA JOVANOVIC, and MILAN TUBA. An algorithm for plant diseases detection based on color features.

[4] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lotfi A. Zadeh. *Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[5] H. Haym. Generalizing version spaces. *Machine Learning*, 17(1):5–46, 1994.

[6] T. P. Hong and S. S. Tseng. A generalized version space learning algorithm for noisy and uncertain data. *IEEE Transactions on Knowledge and Data Engineering*, 9(2):336–340, 1997.

[7] T.M. Mitchell. *Version Spaces: An Approach to Concept Learning*. Ph.D. thesis, Stanford University, 1978.

[8] Herbrich Ralf, Graepel Thore, and Williamson Robert C. *The Structure of Version Space*, pages 257–273. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[9] Michle Sebag. Delaying the choice of bias: A disjunctive version space approach. In *Proceedings of the 13 th International Conference on Machine Learning*, pages 444–452. Morgan Kaufmann, 1996.

[10] C. H. Wang, T. P. Hong, and S. S. Tseng. Inductive learning from fuzzy examples. *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems*, 1:13–18, 1996.