SNoW User Manual

Andrew J. Carlson, Chad M. Cumby, Nicholas D. Rizzolo,
Jeff L. Rosen, Dan Roth

Cognitive Computation Group
Computer Science Department
University of Illinois, Urbana-Champaign

Urbana, Illinois
Current Version: August, 2004
This research has been supported by NSF grants IIS-9801638, IIS-9984168, ITR-IIS-0085836, ITR-IIS-0085980 and an ONR MURI Award.
Contents

1 Introduction 6

2 License terms 8

3 Installation 10

4 The SNoW Architecture 11
   4.1 The Basic System ......................................................... 12
   4.2 Basic Learning Rules ....................................................... 13
      4.2.1 Winnow ............................................................... 13
      4.2.2 Perceptron .......................................................... 15
      4.2.3 Naive Bayes ......................................................... 15
   4.3 Extensions to the Basic Learning Rules ................................. 16
      4.3.1 True Multi-Class Classification (Constraint Classification) ..... 16
      4.3.2 Regularization ...................................................... 17
      4.3.3 Function Approximation (Regression) .............................. 17
      4.3.4 Sequential Model ................................................... 18
      4.3.5 Voting: The Clouds Architecture .................................. 18
      4.3.6 Threshold-Relative Updating ...................................... 19
   4.4 Prediction ................................................................. 20

5 Using SNoW 21
   5.1 Execution Modes ................................................................ 21
   5.2 Command line usage ......................................................... 22
      5.2.1 Mode Selection ......................................................... 22
      5.2.2 Command Line Parameters .......................................... 23

6 File Formats 35
   6.1 Example Files .................................................................... 35
      6.1.1 The Sequential Model ................................................. 36
      6.1.2 Interactive Mode ...................................................... 36
      6.1.3 Example .................................................................... 37
   6.2 Network Files .................................................................... 37
   6.3 Error Files ....................................................................... 38

7 Tutorial 40
   7.1 Training ........................................................................... 40
   7.2 Testing ............................................................................ 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.3 Server</td>
<td>42</td>
</tr>
<tr>
<td>7.4 Interactive Mode</td>
<td>43</td>
</tr>
<tr>
<td>7.5 Other Options</td>
<td>43</td>
</tr>
<tr>
<td>A Command-Line Parameter Index</td>
<td>45</td>
</tr>
<tr>
<td>B Glossary of Terms</td>
<td>46</td>
</tr>
</tbody>
</table>
List of Tables

5.1 Default algorithmic parameters for SNoW’s algorithms. Perceptron uses its “learning rate” for promotion, and the opposite thereof for demotion. Winnow uses \( \alpha \) for promotion and \( \beta \) for demotion. .......................................................... 25

6.1 Partial truth table and SNoW examples for the concept \( x_2 \lor x_5 \). SNoW (with default parameters) will create two target nodes, and each can be said to learn a separate concept. One learns to build a bigger activation for positive examples (label 1), and the other learns to build a bigger activation for negative examples (label 0). ........... 37
Chapter 1

Introduction

SNoW (Sparse Network of Winnows\textsuperscript{1}) is a learning architecture framework that is specifically tailored for learning in the presence of a very large number of features and can be used as a general purpose multi-class classifier.

The current release of the SNoW architectural framework is the third generation of the original SNoW learning system developed by Dan Roth. The learning framework is a sparse network of sparse linear functions over a predefined or incrementally acquired feature space. Several update rules\textsuperscript{2} may be used: classical Winnow and Perceptron, variations of a regularized Winnow and a regularized Perception, regression algorithms based on Gradient Descent, and the naive Bayes algorithm.

SNoW is a multi-class learner, where each class label is represented as a linear function over the feature space. Both one-vs-all and true multi-class training policies\textsuperscript{3} are supported. Predictions are done via a winner-take-all policy or via a voted combination of several learners.

SNoW takes input examples of variable size. Only those features that are active need be mentioned in each example. The expressivity of the linear learner can then be increased by automatically generating new features as combination of primitive features. Input features can be either Boolean or real valued. Decisions made by SNoW are either binary, indicating which of the labels is predicted for a given example, or continuous (in (0, 1)), indicating a prediction confidence. Several other output modes are available.

Training SNoW is very efficient – order(s) of magnitude more efficient than other linear learners – and is competitive in performance to the best learners\textsuperscript{4}.

SNoW has been used successfully in several applications in the natural language and visual processing domains. You are welcome to experiment with it. This release is meant to be used only for research purposes, with the hope that it can be a useful research tool for studying learning in these domains. Feedback of any sort is welcome.

Dan Roth


\textsuperscript{1}To winnow: to separate chaff from grain.
\textsuperscript{2}All implemented algorithms are described in Section 4.2.
\textsuperscript{3}See Section 4.3.1.
\textsuperscript{4}Scripts for parameter tuning and running experiments are supplied with the package.
The document is organized as follows. Chapter 2 contains the software license under the University of Illinois terms. Users need to agree to it and register on-line in order to use the software. Chapter 3 describes how to install the SNoW system. Chapter 4 gives a brief overview of the learning architecture framework and the technical approach including a description of the algorithms and some references. Next, in Chapter 6, the formats of the various files used by the system are described. A detailed description of how to use the system follows in Chapter 5, where all command line parameters and execution modes are described.

Finally, Chapter 7 is a tutorial showing how to use the system with various parameters. A glossary of terms and notations is also supplied.

A new user is encouraged to read all of this document, but the best starting place for learning to use the system is the tutorial. The tutorial gives a good sense of the required steps for using the system. Once a user is comfortable with the default method of using the system, the more detailed description of the command line parameters given in Chapter 5 may be more useful.
Chapter 2

License terms

UNIVERSITY OF ILLINOIS
“SNoW” SOFTWARE LICENSE AGREEMENT
(Office of Technology Management (OTM) #UIUC-DCS-R-99-210 )
(Research and Academic Use License)
Cognitive Computation Group
University of Illinois at Urbana-Champaign

Downloading and using the SNoW software implies that you accept the following license:

Under this Agreement, The Board of Trustees of the University of Illinois (“University”), a body
corporate and politic of the State of Illinois with its principal offices at 506 South Wright Street,
Urbana, Illinois 61801, U.S.A., on behalf of its Department of Computer Science on the Urbana-
Champaign Campus, provides the software (“Software”) described in Appendix A, attached hereto
and incorporated herein, to the Licensee identified below (“Licensee”) subject to the following
conditions:

• 1. Upon execution of this Agreement by Licensee, the University grants, and Licensee accepts,
a royalty-free, non-exclusive license:

  – A. To use unlimited copies of the Software for its own academic and research purposes.
  
  – B. To make derivative works. However, if Licensee distributes any derivative work
  based on or derived from the Software (with such distribution limited to binary form
  only), then Licensee will (1) notify the University (c/o Professor Dan Roth, e-mail:
danr@cs.uiuc.edu) regarding its distribution of the derivative work and provide a copy
  if requested, and (2) clearly notify users that such derivative work is a modified version
  and not the original Software distributed by the University.
  
  – C. To redistribute (sublicense) derivative works based on the Software in binary form only
  to third parties provided that (1) the copyright notice and any accompanying legends
  or proprietary notices are reproduced on all copies, (2) no royalty is charged for such
  copies, and (3) third parties are restricted to using the derivative work for academic and
  research purposes only, without further sublicensing rights.

• No license is granted herein that would permit Licensee to incorporate the Software into a
  commercial product, or to otherwise commercially exploit the Software. Should Licensee wish
to make commercial use of the Software, Licensee should contact the University, c/o the Office of Technology Management (OTM) to negotiate an appropriate license for such commercial use. To contact the OTM: otmailaccount@ad.uiuc.edu; telephone: (217)333-3781; fax: (217) 265-5530.

**THE UNIVERSITY GIVES NO WARRANTIES, EITHER EXPRESSED OR IMPLIED, FOR THE SOFTWARE AND/OR ASSOCIATED MATERIALS PROVIDED UNDER THIS AGREEMENT, INCLUDING, WITHOUT LIMITATION, WARRANTY OF MERCHANTABILITY AND WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTY AGAINST INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS.**

- Licensee understands the Software is a research tool for which no warranties as to capabilities or accuracy are made, and Licensee accepts the Software on an “as is, with all defects” basis, without maintenance, debugging, support or improvement. Licensee assumes the entire risk as to the results and performance of the Software and/or associated materials. Licensee agrees that University shall not be held liable for any direct, indirect, consequential, or incidental damages with respect to any claim by Licensee or any third party on account of or arising from this Agreement or use of the Software and/or associated materials.

- Licensee understands the Software is proprietary to the University. Licensee will take all reasonable steps to insure that the source code is protected and secured from unauthorized disclosure, use, or release and will treat it with at least the same level of care as Licensee would use to protect and secure its own proprietary computer programs and/or information, but using no less than reasonable care.

- In the event that Licensee shall be in default in the performance of any material obligations under this Agreement, and if the default has not been remedied within sixty (60) days after the date of notice in writing of such default, University may terminate this Agreement by written notice. In the event of termination, Licensee shall promptly return to University the original and any copies of licensed Software in Licensee’s possession. In the event of any termination of this Agreement, any and all sublicenses granted by Licensee to third parties pursuant to this Agreement (as permitted by this Agreement) prior to the date of such termination shall nevertheless remain in full force and effect.

- The Software was developed, in part, with support from the National Science Foundation, and the Federal Government has certain license rights in the Software.

- This Agreement shall be construed and interpreted in accordance with the laws of the State of Illinois, U.S.A..

- This Agreement shall be subject to all United States Government laws and regulations now and hereafter applicable to the subject matter of this Agreement, including specifically the Export Law provisions of the Departments of Commerce and State. Licensee will not export or re-export the Software without the appropriate United States or foreign government license.

- The license is only valid when you register as a user. If you have obtained a copy without registration, you must immediately register by sending an e-mail to danr@cs.uiuc.edu.

By its registration and use of the Software, Licensee confirms that it understands the terms and conditions of this Agreement, and agrees to be bound by them.
Chapter 3

Installation

The SNoW system is available for download as gzipped tar archive from:

http://L2R.cs.uiuc.edu/~cogcomp/

Before downloading the archive, you must register as a user and accept the license agreement. You can then download the file snow-unix.tar.gz or snow-win32.zip, which contains complete source code (C++) for the SNoW program, the tutorial data files, a sample script, and the documentation and license. The program can be easily compiled on most UNIX systems as well as on Windows machines as a console application¹.

To install the system on a UNIX machine, unpack the downloaded file:

> gunzip --stdout snow-unix.tar.gz | tar xvf -

Equivalently, if you have GNU tar:

> gtar zxfv snow-unix.tar.gz

This will make a directory Snow#, where # is replaced by SNoW’s major revision number under your current directory. Now change to the newly created directory:

> cd Snow_v3.1

and compile the binary by typing make. If compilation is successful, you should now have an executable named snow. If not, take a look in the Makefile for platform-specific compilation instructions.

The Windows version of SNoW has been successfully compiled and tested using Microsoft Visual C++ (but see the footnote below). Unzip the file snow-win32.zip using any standard compression utility, and load the workspace file snow.dsw included in the archive into Visual C++. Select “Build Snow.exe” from the Build menu and the executable should be compiled successfully.

¹Please note that the Windows version of the software has not been kept up to date after version 2.1.2, since one of the most prominent features of the next major revision (3.0) was the incorporation of performance enhancements centered around the hash_map data structure, which Visual C++ does not support. If you are working on a Win32 platform, we encourage you to use Cygwin to compile our software.
Chapter 4

The SNoW Architecture

The SNoW (Sparse Network of Winnows) learning architecture framework is a sparse network of linear functions over a predefined or incrementally learned feature space and is specifically tailored for learning in domains in which the potential number of features taking part in decisions is very large, but may be unknown a priori. Some of the characteristics of this learning framework are its sparsely connected units, the allocation of features and links in a data driven way, its computational dependence on the number of active features rather than the total number of features and the utilization of a feature efficient update rule.

SNoW has been used successfully in a variety of large scale learning tasks in domains such as natural language [Roth, 1998, Golding and Roth, 1999, Roth and Zelenko, 1998, Munoz et al., 1999, Punyakanok and Roth, 2001, Shen and Joshi, 2003], bioinformatics [Chuang and Roth, 2001], and visual processing [Roth et al., 2000, Yang et al., 2000, Agarwal and Roth, 2002].

Several update rules may be used within SNoW: classical Winnow and Perceptron, variations of a regularized Winnow and a regularized Perception, regression algorithms based on Gradient Descent, and the naive Bayes algorithm.

SNoW can be thought of as a general purpose multi-class classifier, and in this release we have also included a true multi-class capability, in addition to the standard one-vs-all training policy. In addition to the predicted class label, SNoW can assign a prediction confidence value to each label which can be calculated as a function of the distance between the target node’s activation and the threshold.

SNoW should also be thought of as a learning architecture framework. The user designs an architecture within that framework. This means, at a minimum, defining the number of class representations to be learned, but may also include defining many more parameters of the architecture, including update rules and their parameters, regularization parameters, training policies, etc.

In SNoW’s documentation, the user-defined architecture and all data accumulated therein are referred to collectively as the network. In the network, class labels are called targets, and they are learned as sparse linear functions over the input features. By sparse in this context we mean that each target may be learned as a function of a (small) subset of all features in the feature space in a data driven way that is partially controlled by parameters set by the user.

When viewing SNoW simply as a classification system, the typical input would be a collection of labeled examples, consisting of Boolean or real valued features, in a format specified in Chapter 6. The following section provides a slightly more abstract view that may be useful for people in the stage of modeling their problem as a learning problem.
4.1 The Basic System

The SNoW learning architecture framework is a sparse network of linear units over a Boolean or real valued feature space.

In SNoW’s most basic architectural instantiation, a two layer network is maintained. The input layer is the features layer. Nodes in this layer are allocated to features observed in training examples. The second layer consists of target nodes. Each target node corresponds to a concept (a class label) one wants to represent as a function of the input features. For example, when learning a Boolean function, it is most common to use two target nodes; one representing the positive examples and one the negative. Architectural instantiations can also redefine the number of targets, target nodes, and their types, choice of algorithms and algorithmic parameters, and more.

SNoW expects a set of examples as input. During training, each target node uses all examples provided to learn a representation of its concept as a function of (some of) the features. During evaluation/testing, these learned representations are evaluated on a given example to yield a prediction. An incremental mode in which training continues along with testing is also available. SNoW makes use of the infinite attribute domain [Blum, 1992] as opposed to the standard example representation as an array of Boolean or real values, one per feature in the feature space, where element $i$ in the array specifies the strength of feature $i$. As such, examples are represented as a list of active features. Features that would have a strength of 0 in the classical definition are not listed. Each example is thus a list of feature indices (possibly associated with a real valued strength). This is done in anticipation of a very large feature space and sparse examples; that is, only a small percentage of all possible features are active in each example.

Target nodes are linked via weighted edges to (some of the) input features. Edges are allocated dynamically; a feature $i$ is allocated and linked to target node $t$ if and only if $i$ is present in an example labeled $t$. It is also possible to make more complex decisions about which features to include in the network. Since such decisions are made throughout the first pass through the training set, the full benefit of the information in each example may not be reaped after that first pass. Therefore, SNoW processes the training set twice by default, and the number of passes or cycles can be modified on the command line (see the `-r` command line parameter).

Let $A_t = \{i_1, \ldots, i_m\}$ be the set of features that are active in an example and are linked to target node $t$. Let $s_i$ be the real valued strength associated with feature $i$ (default: 1) in the example. Then we say that $t$ predicts positive if and only if

$$\sum_{i \in A_t} w_{i,t} s_i \geq \theta_t$$

(4.1)

where $w_{i,t}$ is the weight on the edge connecting the $i^{th}$ feature to target node $t$, and $\theta_t$ is $t$’s threshold.

The real-valued result of the summation in equation 4.1 is also referred to as the target node’s activation. Target node activations drive weight vector updates during training as well as predictions during testing. The default testing policy for multiple target networks is a winner-take-all rule. Let $T$ be the set of all targets defined in the current architecture instantiation. The predicted target $t^*$ for example $e$ with a set of active features $\bigcup_{t \in T} A_t$ is:

1SNoW also supports voting several target nodes to represent a single target via the cloud architecture - an additional level defined over the target nodes’ layer.
2See the `-d` and `-e` command line parameters.
3This assumes that each target is represented by a single target node. See Section 4.3.5 for situations where this assumption does not hold.
\[ t^*(e) = \arg\max_{t \in T} \sigma(\theta_t, \Omega_t(e)) \]

where \( \Omega_t(e) \) is the activation calculated by the summation in equation 4.1 for target node \( t \) given \( e \), and \( \sigma(\theta, \Omega(e)) \) is a learning algorithm specific sigmoid function\(^4\) whose transition from an output close to 0 to an output close to 1 centers around \( \theta \). Specific sigmoid functions employed by SNoW's various learning algorithms are discussed in the next section.

The only other testing policy currently implemented in SNoW is automatically enabled for networks which contain exactly one target node. This single-target testing policy makes predictions similarly to the classical versions of the linear threshold algorithms SNoW supports by simply returning the prediction of the single target node (either positive or negative as determined by the equation discussed above) as SNoW's prediction.

SNoW can then display for the user its prediction for each example, its accuracy across an entire labeled testing set, the activations of each target node for each example, those same activations after softmax normalization, or the results obtained from applying the sigmoid function to the activations, which will hereafter be referred to as the sigmoid activations. In simple architecture instantiations, sigmoid activations are equivalent to prediction confidences, but see Section 4.3.5 for a more in depth discussion.

### 4.2 Basic Learning Rules

The learning policy is on-line and mistake-driven (except when naive Bayes or Gradient Descent is enabled), and several update rules can be used. With the exception of naive Bayes, all update rules are variations of Winnow and Perceptron as implemented within the infinite attribute model. Below, we briefly describe the basic learning rules implemented within SNoW and point to some relevant papers. The specific ways these rules are selected and used within SNoW are described in Chapter 5.

In the default architecture instantiation, SNoW treats target nodes independently; each is updated individually depending on its own activation and threshold and independent of the activations of other target nodes (but see Section 4.3.1). Target node \( t \) considers a training example positively labeled if \( t \) is active in it and negatively labeled otherwise. In addition, if the algorithm associated with target node \( t \) is either Winnow or Perceptron and an initial feature weight \( f_0 \) is not explicitly specified, it will be calculated with the following formula:

\[ f_0 = \frac{\theta_t}{\rho} \]

where \( \theta_t \) is the threshold setting for \( t \)'s algorithm and \( \rho \) is the average number of active features per example\(^5\). The justification for this formula is purely empirical; it has been found to work well in practice.

#### 4.2.1 Winnow

The Winnow update rule has, in addition to the threshold \( \theta_t \) at target node \( t \), two update parameters: a promotion parameter \( \alpha_t > 1 \) and a demotion parameter \( 0 < \beta_t < 1 \). These are used to

---


\(^5\)Note that SNoW will decide to abbreviate the calculation of \( \rho \) if it finds it cannot process the entire dataset quickly. In that case, only part of the dataset is used to calculate average example size.
update the current hypothesis in \( t \) (the set of weights \( w_{i,j} \)) only when a mistake in prediction is made.

A Winnow update proceeds as follows:

- Let \( A_t = \{i_1, \ldots, i_m\} \) be the set of active features in a given example that are linked to target node \( t \), and let \( s_i \) be the strength associated with feature \( i \) in the example.

- If the algorithm predicts negative (that is, \( \sum_{i \in A_t} w_{t,i} s_i < \theta \)), and the specified label is positive, the weights of features active in the current example are promoted in a multiplicative fashion:

\[
\forall i \in A_t, w_{t,i} \leftarrow w_{t,i} \cdot \alpha_i^{s_i}
\]

- If the algorithm predicts positive (\( \sum_{i \in A_t} w_{t,i} s_i \geq \theta \)) and the specified label is negative, the weights of features active in the current example are demoted:

\[
\forall i \in A_t, w_{t,i} \leftarrow w_{t,i} \cdot \beta_i^{s_i}
\]

- All other weights are unchanged.

In SNoW, Winnow’s sigmoid activation is calculated with the following formula:

\[
\sigma(\theta, \Omega) = \frac{1}{1 + e^{\theta - \Omega}}
\]

where \( \theta \) is a target’s threshold and \( \Omega \) is a target’s activation with respect to an example.

The key feature of the Winnow update rule [Littlestone, 1988] is that the number of examples required to learn a linear function grows linearly with the number of relevant features and only logarithmically with the total number of features\(^6\). This property seems crucial in domains in which the feature space is vast, but a relatively small number of features is relevant (this does not mean that only those will be active, or have non-zero weights). Winnow is known to learn any linear threshold function efficiently, to be robust in the presence of various kinds of noise and in cases where no linear threshold function can make perfect classification, and to still maintain its above-mentioned dependence on the number of total and relevant attributes [Littlestone, 1991, Kivinen and Warmuth, 1997].

We note that the original Winnow algorithm is a positive weight algorithm. Therefore, it is typically not expressive enough for applications. Using the “duplication trick” [Littlestone, 1988] is not feasible when the number of features is very large but only a small number of them is active in each example. The default SNoW architecture instantiation, using one target node for each class label (two target nodes for Boolean functions), resolves this issue. Note also that this is different from the balanced version of Winnow. This version can be run in SNoW as a special case of the true multi-class training policy (see below).

---

\(^6\)In the on-line setting this is usually phrased in terms of a mistake-bound but is known to imply convergence in the PAC sense [Valiant, 1984, Helmbold and Warmuth, 1995].
4.2.2 Perceptron

The Perceptron update rule is implemented similarly. It takes only two parameters, a threshold $\theta_t$ and a learning rate $\alpha_t$. As in Winnow, whenever a mistake is made, the weights of active features are updated. In this case, they are updated via either addition or subtraction, depending on whether the mistake was made on a positive or negative example respectively.

A Perceptron update proceeds as follows:

- Let $A_t = \{i_1, \ldots, i_m\}$ be the set of active features in a given example that are linked to target node $t$, and let $s_i$ be the strength associated with feature $i$ in the example.
- If the algorithm predicts negative (that is, $\sum_{i \in A_t} w_{t,i}s_i < \theta_t$), and the specified label is positive, the weights of features active in the current example are promoted in an additive fashion:

  \[ \forall i \in A_t, w_{t,i} \leftarrow w_{t,i} + \alpha_t s_i \]

- If the algorithm predicts positive ($\sum_{i \in A_t} w_{t,i}s_i \geq \theta_t$) and the specified label is negative, the weights of features active in the current example are demoted:

  \[ \forall i \in A_t, w_{t,i} \leftarrow w_{t,i} - \alpha_t s_i \]

- All other weights are unchanged.

Perceptron’s sigmoid activation is calculated with the following formula:

\[ \sigma(\theta, \Omega) = \frac{1}{1 + e^{\theta - \Omega}} \]

This is exactly the formula used by [Mitchell, 1997] as a sigmoid function in the context of neural networks.

4.2.3 Naive Bayes

In the case of naive Bayes, a feature’s weight in a target node is simply the logarithm of the fraction of positive examples (positive with respect to the target) in which the feature is also active (see, e.g., [Roth, 1999, Roth, 1998]). In addition, the relative weight of the target is used as “prior”, and a fixed smoothing weight is added for evaluation of active features that were never observed in training\(^7\).

Notice that when using Perceptron and Winnow with the default training policy, all examples are presented to every target node, and an update may occur whether that target is active in the example or not. This is not the case for naive Bayes. Each target node takes into account only the examples labeled with it (i.e., the target’s hypothesis is learned only from positive examples, and training is not mistake driven).

Naive Bayes does not apply a sigmoid function; its activation is equivalent to its sigmoid activation.

---

\(^7\)See the `-b` command line parameter for more information on smoothing in naive Bayes.
4.3 Extensions to the Basic Learning Rules

SNoW contains a large number of options that serve to modify the behavior of the basic update rules. These include eligibility of features, options for discarding features, conditional prediction based on a prediction threshold, and others. They are all selected via the command line, and all command line parameters are described in Chapter 5.

Below we describe only the main algorithmic extensions over the basic update rules. These training policies and update rules are implemented independently of each other. As a consequence, for example, the true multi-class (Constraint Classification) [Har-Peled et al., 2002] training policy can be invoked with either of the Perceptron or Winnow update rules. Similarly, one can enable a function approximation (regression) training policy with Perceptron to perform a stochastic approximation to Gradient Descent [Mitchell, 1997] or with Winnow to perform Exponentiated Gradient Descent [Kivinen and Warmuth, 1997]. In addition, each example can be applied only to selected targets when using Winnow or Perceptron as in the sequential model [Even-Zohar and Roth, 2001]. For one final example, it is also possible for nodes’ output to be cached and processed along with the output of other nodes to produce a more complicated decision support mechanism\(^8\) (as in, e.g., [Golding and Roth, 1999]).

4.3.1 True Multi-Class Classification (Constraint Classification)

SNoW is a multi-class classifier. The default architecture instantiation implements a one-vs-all multi-class training policy: each target node \( t \) is learned individually, by considering all examples labeled \( t \) as positive and all other examples as negative. The testing policy then simply evaluates all target representations on a given example and applies a winner-take-all gate to the targets’ prediction confidences.

The winner-take-all testing policy is expressive and allows for representing a decomposition of the domain into labeled regions that are separated by hyperplanes (a Voronoi diagram). However, learning independently does not support this expressivity, since each labeled region needs to be linearly separable from the union of all others.

The true Multi-Class classification training policy buys back this expressivity and allows SNoW to learn Voronoi diagrams [Har-Peled et al., 2002]. In addition to supporting true multi-class, it supports ranking of labels, and allows examples to be labeled with several labels, with a specified order between them. In this document, we refer to this training policy as Constraint Classification. See the \(-0\) command line parameter for usage information.

Specifically, assume there are \( k > 1 \) possible class labels in a learning experiment, and assume that a given training example contains \( n, 1 \leq n \leq k \), class labels (target IDs) with IDs \( c_1, c_2, ..., c_n \) which appear in that order in the example. With \(-0 + \) set, SNoW will interpret the labels in this example as asserting an activation lattice. The activation of target node \( c_1 \) should be higher than the activation of target node \( c_2 \), the activation of target node \( c_2 \) should be higher than the activation of target node \( c_3 \), and so on, and the activation of target node \( c_n \) should be higher than the individual activations of all the remaining \( k - n \) target nodes.

Let \( c_i \) and \( c_{i+1} \) be the first two labels in the example’s lattice for which SNoW evaluates a higher activation for target node \( c_{i+1} \) than for target node \( c_i \). Target node \( c_i \) will be promoted and target node \( c_{i+1} \) will be demoted. The activation of target node \( c_{i+1} \) is then recalculated before comparing with target node \( c_{i+2} \). When target node \( c_n \) is compared to the remaining \( k - n \) target nodes, target nodes with smaller IDs are inspected first. Again, if target node \( c_n \) needs to

\(^8\)See 4.3.5 for a more concrete description of this mechanism.
be promoted after comparison with one of the remaining $k - n$ target nodes, its activation will be recalculated after promotion and before comparing with the target node with the next higher ID.

This implementation is a special case of the Constraint Classification algorithm described in [Har-Peled et al., 2002]. Notice that when $k = 2$ and the update rule used is Winnow, this algorithm reduces to the balanced version of Winnow described in [Littlestone, 1988].

SNoW also implements a more conservative version of this algorithm. In the conservative Constraint Classification algorithm, only the target node with highest activation according to the current network with respect to a given example and the target node that corresponds to that example’s label are compared. If they are the same target node (in other words, if the network predicted correctly), no change is made. Otherwise, the target node that ended up with the highest activation is demoted, and the target node corresponding to the example’s label is promoted.

### 4.3.2 Regularization

Regularization in SNoW is implemented as a small modification to the Winnow and Perceptron update rules so that they try to fit a “thick hyperplane” in between positive and negative examples [Dagan et al., 1997, Grove and Roth, 1998, Li et al., 2002]. In fact, SNoW allows for a different thickness for positive and negative examples which can be used to incorporate a non-symmetric loss function. It is not difficult to show that the modified update rules still have a mistake bound that depends on the margin of the data (with the additional thickness parameter).

Specifically, if the floating point thickness parameters are set to $p$, $n$ and target node $t$ encounters a positive example, its activation will have to be greater than or equal to $\text{threshold} + p$ for SNoW to interpret the node’s prediction as correct. Otherwise, it will be promoted. Similarly, if $t$ encounters a negative example, SNoW will interpret $t$’s prediction as correct if its activation is less than $\text{threshold} - n$. Otherwise, it will be demoted.

See option $-S$ for usage details.

### 4.3.3 Function Approximation (Regression)

SNoW implements a function approximation version of Perceptron’s additive update rule - a stochastic approximation to the Gradient Descent algorithm (GD) – and of Winnow’s multiplicative update rule - an exponentiated Gradient Descent algorithm (EGD) [Kivinen and Warmuth, 1997]. See option $-G$ for usage details.

Both use the same training policy. They are non-mistake-driven, meaning that each target node an example is presented to will perform an update on that example whether the network can be said to have made a mistake on that example or not. They are also on-line, as opposed to the true Gradient Descent algorithm in which the total error incurred by a weight vector with respect to an entire training set and the total update to that weight vector with respect to that error are calculated before any weights are updated. Mitchell calls the on-line version a stochastic approximation. A disadvantage to this approximation algorithm is that it requires a smaller learning rate to guarantee convergence. An advantage is that it is less susceptible to “getting stuck” in a local minimum that is not the global minimum [Mitchell, 1997].

Specifically, let $A_t = \{i_1, \ldots, i_m\}$ be the set of active features in a given example that are linked to target node $t$, let $w_i^t$ be the weight of feature $i$ in $t$, let $\alpha$ be the learning rate, let $s_i$ be the strength of feature $i$ in the given example, and let $s_t$ be the strength of target $t$ in that same example$^9$. Then $\Omega_t = \sum_{i \in A_t} w_i^t s_{i_t}$ is the predicted activation of target node $t$ before updating, and $s_i$ will represent the correct activation for this example while updating.

---

$^9$See Section 6.1 for more information on feature (and target) strengths.
A regression update in SNoW proceeds as follows:

- **GD:** When enabled in conjunction with Perceptron, the GD update rule becomes:

  \[ w_{t,j} = w_{t,j} + \alpha \cdot (s_t - \Omega_t) \cdot s_i \]

- **EGD:** When enabled in conjunction with Winnow, the corresponding algorithm is referred to as Exponentiated Gradient Descent in the literature:

  \[ w_{t,i} = w_{t,i} \cdot \alpha^{(s_t - \Omega_t) \cdot s_i} \]

(Note that \( \beta \) is never used in this case.)

Naive Bayes does not have a Gradient Descent update rule. In both Winnow and Perceptron, each example is presented only to targets active in that example.

### 4.3.4 Sequential Model

The number of class labels considered as possible predictions for a given example, also called the size of the “confusion set”, plays an important role in the performance of multi-class learners. Typically, the larger the confusion set is, the lower the average prediction accuracy is\(^{10}\).

SNoW supports a dynamic decision, per example, on which targets should be included in the confusion set. Assume that \( c_1, \ldots, c_k \) are the class labels in a learning scenario. SNoW allows the user to include, along with the example, a subset of targets in \( c_1, \ldots, c_k \). That subset is viewed as a specification of which target nodes this example should be presented to. This can be done both in training and testing.

The determination of this subset can be done by an external process if it is capable of suggesting that a given example, say, will clearly not be labeled \( c_1 \). See [Even-Zohar and Roth, 2001] for a study of the sequential model and [Li and Roth, 2002] for an application.

The Sequential Model can be thought of as a training policy, although no command line parameter need be set to enable it. Examples simply need to have the appropriate information added to them. See Chapter 6 for details on an example’s format both with and without the sequential model.

### 4.3.5 Voting: The Clouds Architecture

Each target node in SNoW is learned as a linear function of the features found active with its target ID in the training set. One way to increase the expressivity of this representation is to allow each target (class label) to be represented as a weighted combination of target nodes’ output. SNoW supports this by allowing more than one target node, potentially learned by different learning algorithms, to represent the same class label. Those that do are grouped into *clouds*, in which they collectively provide the output for a single target. For example, target 1 could be the combination of two target nodes, one learned using Perceptron and one learned using Winnow, while target 2 is a combination of a naive Bayes learner and a Winnow learner.

Each linear function that participates in this combination is trained individually according to its own parameters. Only at decision time is the target’s prediction confidence determined as a weighted sum of the individual target nodes’ sigmoid activations with coefficients that are a function

---

\(^{10}\)See [Even-Zohar and Roth, 2000] for a detailed discussion
of the cumulative number of mistakes made by each target node in the cloud during training. In light of this, the target $t^*$ predicted by SNoW’s winner-take-all testing policy for an example $e$ can be stated more generally than it was in Section 4.1 as:

$$ t^*(e) = \arg \max_{t_n \in T} \sum_{t_n \in T} \sigma_{t_n}(\theta_{t_n}, \Omega_{t_n}(e)) \cdot c_{t_n} $$

where $T$ is the set of all possible predictions (class labels), each $t_n$ is a target node, $\theta_{t_n}$ is the threshold of the algorithm associated with $t_n$, $\Omega_{t_n}(e)$ is the activation calculated by $t_n$ for $e$, $\sigma_{t_n}(\theta, \Omega(e))$ is the algorithm specific sigmoid function discussed in Sections 4.1 and 4.2, and $c_{t_n}$ is $t_n$’s cloud confidence (discussed next).

Cloud confidence is a floating point value stored in each target node and initialized to 1. When $t_n$ makes a mistake during training, its cloud confidence $c^{t_n}$ is updated with the following formula:

$$ c_{t_n} = \frac{c_{t_n}}{1 + \frac{2}{100 + m_{t_n}}} $$

where $m_{t_n}$ is the total number of mistakes made by target node $t_n$ so far. This formula was designed to decrease cloud confidence at a decreasing rate so that a cloud confidence value is still representable by a double-precision floating point variable after many mistakes. Note that the value that $c_{t_n}$ is divided by approaches 1 as the number of mistakes approaches $\infty$. For instance, after the first mistake, the cloud confidence of $t_n$ is roughly 0.98. After 100 mistakes, the cloud confidence is roughly 0.25, and after 1,000,000 mistakes, the cloud confidence is roughly $1.03 \cdot 10^{-8}$.

Cloud confidence values are only updated in SNoW’s -train mode. After SNoW has finished training on all examples, cloud confidences are then normalized so that they sum to 1 within each cloud. When incremental learning is enabled (see the -i option) and SNoW is in -test mode, cloud confidence values are read from the network file and used in example evaluation, but they are not updated.

### 4.3.6 Threshold-Relative Updating

In addition to the function approximation mode in Section 4.3.3, SNoW implements another update rule in which the weight update rate is not constant (as in $\alpha$ or $\beta$ for Winnow, or learning_rate for Perceptron) but rather depends on how far the activation value is from the threshold. With Threshold-Relative Updating (option -t), an update always causes the activation of an example on which the network has made a mistake to jump directly to the immediate vicinity of the threshold instead of taking a small step towards it.

Let $\mathcal{A}_t = \{i_1, \ldots, i_m\}$ be the set of active features in a given example that are linked to target node $t$, let $w_{i,t}$ be the weight of feature $i$ in target $t$, let $s_i$ be the strength of feature $i$ in the example, and let $\theta_t$ be the threshold at target node $t$. Then $\Omega_t = \sum_{i \in \mathcal{A}_t} w_{i,t} s_i$ is the activation of target $t$ before updating.

The Winnow threshold relative updating rule is:

$$ w_i = w_i \left( \frac{\text{rate} \cdot \theta_t}{\Omega_t} \right) $$

where rate is $\alpha$ if the example is positive and $\beta$ if it’s negative. Notice that in this case, following the update, we get:

$$ \text{new } \Omega_t = \sum_{i \in \mathcal{A}_t} \text{new } w_{i,t} s_i = \text{rate } \theta_t $$

19
Similarly, the update rule for Perceptron becomes:

\[ w_i = w_i + \frac{\text{rate} + (\theta_i - \Omega_i)}{\text{active_features}} \]

where \text{rate} is the Perceptron algorithm parameter \text{learning rate} if the example is positive and \text{learning rate} if it’s negative, and \text{active features} is the total number of active features in the example. Again, it’s easy to see that in this case, following the update, we get:

\[ \text{new } \Omega_i = \sum_{i \in A_t} \text{new } w_{i,s_i} = \text{rate} + \theta_i \]

That is, the example’s new activation will be roughly equal to the threshold with a small buffer between the updated activation and the threshold. See option \text{--t} for details.

4.4 Prediction

SNoW currently implements two different testing policies.

- **Winner-take-all**: The target with the highest prediction confidence when testing a given example is selected as the final prediction for that example. Note that (in light of Section 4.3.5) this prediction confidence is actually a weighted sum of sigmoid activations of target nodes in a cloud that may contain multiple target nodes. Note also that if the user instantiates only one algorithm for each target, the prediction confidence is equal to the sigmoid activation of the single target node in the cloud.

- **Single-target**: SNoW predicts either positive or negative if the single target node’s activation is higher (actually, \( \geq \)) or lower than its threshold respectively. In this case, there is no weighted sum or sigmoid function involved.

Those are the policies that SNoW can handle automatically. The user may also choose to score examples externally, making use of the results of testing calculations that SNoW performs. For instance, target node activations and sigmoid activations can be reported. In addition, SNoW can automatically normalize target node activations with softmax and report the results. The softmax normalized activation for target \( t \) on example \( e \) is

\[ \frac{e^{\Omega_t(e)}}{\sum_{t' \in T} e^{\Omega_{t'}(e)}} \]

where \( \Omega_t(e) \) is the target node activation of \( t \) on \( e \). See the \text{--o} command line parameter for more information.
Chapter 5

Using SNoW

5.1 Execution Modes

SNoW may be run in five major modes: Training, Testing, Interactive, Evaluation, and Server.

Training consists of presenting labeled examples (see Chapter 6 for a description of file formats) to the network and learning a weight vector representation for each target concept. At the completion of training, the resultant network, which can be used to make predictions on previously unseen examples, is written to the network file. Thus, training must always be performed before testing or evaluations can be performed.

Testing consists of presenting examples to the system and predicting a label. This is done by evaluating the representation learned for each of the targets – computing the target’s prediction confidence given the new example – and choosing the target with the highest prediction confidence. The results of testing can be output in a number of ways.

Interactive mode allows the user to (1) evaluate specific examples, and (2) control the training process by specifying the targets to be promoted or demoted on an example by example basis. In this mode, the updating decision is usually made based on previous results of evaluation. For example, one may want to evaluate one example first, look at the activation value of each class, and then decide which classes to promote and which to demote. Therefore, to make the interactive mode truly interactive, the input file shouldn’t be a regular file, but STDIN or a pipe instead.

Evaluation is used to make a prediction based on a labeled or unlabeled example supplied on the command line.

Server mode is a limited version of Testing mode in which after loading the network, SNoW waits for examples to be sent over a user specified port. SNoW then returns its output over the same port. This mode can make testing much more efficient when the network is large and there are many example files that are best or necessarily kept separate when tested. Time will not be spent reloading the network for each file.

Several other modes of operation initiated with optional command line parameters are supported, including, but not limited to, incremental learning (see option -i), regularization (Section 4.3.2), and function approximation (Section 4.3.3).

A recommendation: SNoW has several options and input parameters that are required and others that are commonly used. We recommend saving commands and options in a simple shell script that records the typical way you use SNoW, and running this script. This also allows you to automate parameter tuning in a simple way. In SNoW’s distribution, you will find a booleanexample/ directory corresponding to the discussion in Section 6.1.3 and a tutorial/ directory corresponding
to the discussion in Chapter 7, both of which contain examples of SNoW’s usage in easy to use shell scripts.

5.2 Command line usage

5.2.1 Mode Selection

SNoW’s basic usage is as follows:

```
snow -train -I inputfile -F networkfile [-AaBbcdEefGgiLMmOpPrRsSstvWw]
snow -test -I inputfile -F networkfile [-abEefGgiLMmOpRsSstvWw]
snow -interactive -I inputfile -F networkfile [-AEfLoPpRvWw]
snow -evaluate -x example -F networkfile [-befmpRvW]
snow -server port -F networkfile [-befllmopvw]
```

Parameters in the bracketed section are optional, and the others are required. All parameters require an argument, and thus must be listed separately on the command line. Their order does not matter except for the first parameter, which must be one of `-train`, `-test`, `-interactive`, `-evaluate`, or `-server`. This parameter sets one of the following modes:

- `train`: The system is run in training mode. Command line parameters describing the desired SNoW architecture are used to create a new network. The input file is considered to be a set of labeled training examples. Each example in the file is considered a positive example for all targets which are included (active) in the example and a negative example for all other targets (absent from the example). At the end of training, the resulting network (i.e., weight vectors and other such parameters) is written to the network file. If the file already exists, it is overwritten.

- `test`: The system is run in a batch test mode. First, the system is initialized with the data in the network file. Next, each example in the input file is presented to the system and classified. The input file can consist of labeled or unlabeled testing examples. If the examples are labeled, the result of the classification can be compared with the label of the example and scored, with a final accuracy reported after all examples are presented. The result of each prediction can also be output to a file in a number of ways and scored externally.

- `interactive`: The system is run in an interactive training mode. First, the network is initialized with the data in the network file, just as in test mode. If the network file doesn’t exist, a new network is created according to any specified architecture parameters as in training mode. The input file contains unlabeled examples with individual handling instructions. At the end of processing, the resulting network (i.e., weight vectors and other such parameters) is written to the network file. Note that when running in this mode, the following additional command line parameters are implied and cannot be overridden: `e count:1 -g -l -r 1 -s f`. Also, the Naïve Bayes, Constraint Classification, threshold relative updating, Gradient Descent, and thick separator algorithms are not available in this mode.

- `evaluate`: The system is run in an “online” test mode. A single labeled or unlabeled example is supplied on the command line. SNoW assumes it is unlabeled. The process terminates after making a prediction for this single example. The predicted target ID is output and also returned to the operating system in the process’s exit status. Note that running the system
in Evaluation mode loads the network once for each example, and thus is not the best way to process large sets of examples.

-server: The system is run in batch test mode. First, a client program may send a string of command line parameters to the server. Any of the parameters that are legal when instantiating SNoW in -server mode may be passed by a client to the server except the -e parameter. Input to the server can consist of labeled or unlabeled testing examples and is expected to be received over a user specified port. Each example is presented to the system and classified. If the examples are labeled, the result of the classification can be compared with the label of the example and scored. SNoW’s output is then sent back over the same port. When instantiating SNoW as a server, the network is loaded once for any number of examples from any number of files to be tested.

See Chapter 7 for more detailed examples of SNoW’s usage in each of these modes. Also, see Section 6.1 for precise descriptions of accepted example file formats.

5.2.2 Command Line Parameters

Some parameters are required, some are optional and some may be required, optional, or superfluous depending on the mode in which the system is run. Refer to Chapter 7 for simple usage examples. Note that some training parameter settings are not stored in the network. Therefore, when invoking SNoW in a mode other than -train or interactive, it is good practice to specify the same optional command line parameter settings that were used when training the specified network. Alternatively, reading the descriptions of each parameter below will give insight as to which parameters are written to the network file. The user can then decide precisely which parameters need to be set in each mode for his particular purposes.

There are four kinds of SNoW command line parameters: (1) architecture definition, (2) training, (3) testing, and (4) I/O parameters.

Architecture definition parameters instantiate target nodes and algorithms, tweak algorithm parameters, and modify the way algorithms behave while training. They are also training parameters, and they conform to all the rules of training parameters. Unless otherwise specified, their settings are stored in the network file at the end of -train or -interactive mode. You may find it useful to define an architecture file and mention it in the command line or in a shell script (using the -A option) instead of describing the entire architecture directly in the command line. See Chapter 7 for examples.

Training parameters can always be specified in -train mode and may or may not be written to the network file at the end of execution. They are also available in -interactive mode unless otherwise specified. If a training parameter is written to the network file at the end of -train or -interactive mode, its setting cannot be overridden by specifying a new setting on the command line in another mode. Otherwise, it can (and often should be, since its setting during the initial training was lost when it wasn’t written to the network file). In -test mode, training parameters that aren’t written to the network after -train or -interactive mode can be specified on the command line and will take effect when incrementally learning with the -i + testing parameter. Settings that were stored in the network will also take effect in this scenario. Training parameters never have an effect on -evaluate or -server modes.

Testing parameters are available in any mode other than -train unless otherwise specified and are never written to the network. In -train mode, they can also take effect when testing after training is complete with the -T I/O parameter.
I/O parameters are for specifying the names of SNoW's input and output files and for changing the type and amount of information presented to the user. Unless otherwise specified, they are available in every execution mode.

**Architecture Definition Parameters**

SNoW is prepared with a useful default architecture that performs well in many situations. Many alternative architectures can also be specified either directly in the command line or using an architecture file as described below.

The main component of an architecture definition is the target node description. Each target node can be trained using a different update rule (learning algorithm) although it’s more common to use the same update rule for all targets. For each target, at least one update rule used to learn its representation must be specified. If Winnow (-W) or Perceptron (-P) are used, it is also possible to specify their parameters; otherwise, default parameters are used. Target nodes are declared by listing those nodes which will be associated with each algorithm. Either single IDs or ranges of targets can be given.

For example, this is probably the simplest architecture that can be specified:

-W :0-1

It declares two target nodes, for the target IDs 0 and 1, to be learned using the Winnow update rule with default parameters. This architecture is suitable for a two-class learning scenario.

Here’s an example of a more complicated architecture:

-W 1.5,0.8,4.0,0.5:0-2,5,9  
-P 0.1,4.0,0.2:1-3,4,8

Here, Winnow will be used to learn a representation for targets 0, 1, 2, 5, and 9, and Perceptron will be used for targets 1, 2, 3, 4, and 8. Note that when more than one algorithm is specified for a single target ID, the outputs of those algorithms will be combined into a single prediction confidence for that target (see Section 4.3.5). In the above case, all target nodes trained with Winnow will be trained with the same parameters, and all those trained with Perceptron will be trained with the same parameters. Note that one can also train different target nodes with different parameter settings of the same algorithm.

Command line parameters for designing architectures with SNoW’s various supported algorithms either on the command line or in architecture files are described below.

-A `<architecture file>`: Specifies the name of a file from which to read the desired architecture definition and parameters. The file may look, for example, like:

-W 1.5,0.8,4.0,0.5:0-1  
-P 0.1,4.0,0.20:0-1  
-e count:1  
-r 4

This parameter has no default.

-B -P -W: These three parameters instantiate naive Bayes, Perceptron, and Winnow learners respectively. If none of them are specified, a default Winnow architecture is instantiated. They are `-train` mode parameters that have no effect in any other mode.
They also must each be followed by arguments, which are described in detail below. The arguments that follow each of these parameters may not have any whitespace between them; they are instead separated by commas and colons. Any `<targets>` argument may make use of commas and hyphens to specify ranges of target IDs. When using `-interactive` mode, none of the arguments to `-P` or `-W` may be omitted. Otherwise, some of them can be omitted, and when they are, they take the defaults listed in table 5.1. See below for more details.

- `B :<targets>` : Specifies the targets to train with naive Bayes. Targets must always be specified. This parameter is not available in `-interactive` mode.

- `P <learning_rate>,<threshold>,<initial_weight>:<targets>` : Specifies the targets to train with the single layer Perceptron algorithm, along with explicit algorithm parameters. Here all parameters are stated explicitly.

- `P <learning_rate>,<threshold>:<targets>` : Specifies the targets to train with the single layer Perceptron algorithm, along with explicit algorithm parameters. Here, the initial weight is omitted, and is thus calculated as a function of the threshold and the average number of active features per example\(^1\).

- `P <learning_rate>:<targets>` : Specifies the targets to train with the single layer Perceptron algorithm, along with the `learning_rate`. Here a default threshold is used, and the initial weight is calculated as a function of the threshold and the average number of active features per example.

- `P :<targets>` : Specifies the targets to train with the single layer Perceptron algorithm, and uses default algorithm parameters. Targets must always be specified.

- `W <promotion>,<demotion>,<threshold>,<initial_weight>:<targets>` : Specifies which targets to train with the Winnow algorithm along with explicit algorithm parameters. All parameters are stated explicitly.

- `W <promotion>,<demotion>,<threshold>:<targets>` : Specifies the targets to train with the Winnow algorithm along with explicit algorithm parameters. Here, the initial weight is omitted, and is thus calculated as a function of the threshold and the average number of active features per example.

- `W <promotion>,<demotion>:<targets>` : Specifies the targets to train with the Winnow algorithm along with explicit algorithm parameters. Here, a default threshold is used, and the

\(^1\)See Section 4.2 for more details

Table 5.1: Default algorithmic parameters for SNoW’s algorithms. Perceptron uses its “learning rate” for promotion, and the opposite thereof for demotion. Winnow uses \(\alpha\) for promotion and \(\beta\) for demotion.
initial weight is calculated as a function of the threshold and the average number of active features per example.

-\(W\) :\(<\text{targets}\>\) : Specifies the targets to train with the Winnow algorithm and default algorithm parameters. Targets must always be specified.

-\(G\) :\(<+ \mid ->\) : Setting this parameter to \(+\) enables the function approximation (regression) algorithm. This algorithm is non-mistake-driven, meaning that each target node an example is presented to will perform an update on that example whether the network can be said to have made a mistake on that example or not. This option can be enabled in conjunction with Perceptron, giving rise to a stochastic approximation to the Gradient Descent algorithm, or with Winnow, resulting in an Exponentiated Gradient Descent algorithm. Please see Section 4.3.3 for details on these algorithms.

Naive Bayes does not have a Gradient Descent update rule. In both Winnow and Perceptron, each example is presented only to targets active in that example. Gradient Descent cannot be enabled at the same time as either Constraint Classification (Section 4.3.1) or threshold relative updating (Section 4.3.6). If \(-O\) \(+\) and \(-G\) \(+\) are specified, \(-G\) is automatically set to \(-\). Otherwise, if \(+t\) \(+\) and \(-G\) \(+\) are specified, \(-t\) is automatically set to \(-\). This parameter is not available in \(-\text{interactive}\) mode. Default \(-\).

-\(O\) :\(<+ \mid ->\>\) : This parameter is used to enable the true multi-class (a.k.a. Constraint Classification or "Ordered Targets") training algorithm. When set to \(-\), each target node learns a simple relation between the features and a class label without any interaction with the other target nodes. Setting this parameter to \(+\) enables the Constraint Classification training algorithm, in which target nodes compare their activations with each other to determine which will be promoted or demoted. The optional second argument to this parameter, which is only recognized when the first argument is set to \(+\), enables the conservative version of this algorithm when set to \(+\). Omitting the second argument is the same as setting \(+,\)\(-\). See a discussion of the algorithmic details in Section 4.3.1.

Note that with \(-O\) \(+\), the threshold parameter specified on the command line for Winnow and Perceptron learners is never involved in training unless \(-t\) \(+\) is also specified. The setting of this parameter is not stored in the network file. This parameter is not available in \(-\text{interactive}\) mode. Default \(-,\)\(-\).

-\(S\) :\(<p>,<n>\) : Specifies the floating point thickness of the separator between positive and negative examples for training Perceptron and Winnow learners. This option is the regularization option of SNoW and it is described in detail in Section 4.3.2. Generally, the goal of this option is to modify the given algorithm such that it tries to fit a thicker hyperplane between positive and negative examples. We have found that any setting of this parameter from \(-S\) 1.0 to \(-S\) 2.5 most often results in better performance of the learned network (but, of course, this finding is intrinsically dataset dependent).

If the \(<n>\) is not specified, \(n\) will take the same value as \(p\). This parameter will not have any effect if \(-O\) \(+\) is also specified. The setting of this parameter is not stored in the network file. This parameter is not available in \(-\text{interactive}\) mode. Default 0.0.

-\(t\) :\(<+ \mid ->\) : This parameter is for enabling threshold relative updating in either of the Winnow or Perceptron update rules. Setting it to \(+\) enables the algorithm. In threshold relative updating, the respective learning rate parameters for each update rule are intended to synchronize the
example’s updated activation with the algorithm’s threshold setting, save for a small buffer. A discussion and the algorithmic details of this option are described in Section 4.3.6.

This parameter has no effect when \(-g +\) is also specified. See the description of \(-g\) for more details. This parameter is not available in \(-interactive\) mode. Default \(-\).

**Training Parameters**

The following parameters are all optional. As in all other cases, they can be defined as parts of the architecture file or on the command line. Some change the bookkeeping that SNoW does as it keeps track of the features linked to its target nodes. Some are used to tailor SNoW’s efficiency to a particular learning scenario and dataset. None of the settings of these parameters are written to the network file unless otherwise stated.

\(-a \leftrightarrow \rightarrow\) : Setting this parameter to \(+\) forces all non-discarded features to be written to the network file. If set to \(-\), features that have not yet reached the eligibility threshold (i.e., \(pending\) features) are not written to the network. When set to \(+\) and used in conjunction with a non-default setting of \(-e\) in training, it is best to specify the same setting of \(-e\) during testing. This parameter is not available in \(-interactive\) mode. Default \(-\).

\(-d <none | abs:<k> | rel>\) : Specifies the discarding method, if any. Once a feature is discarded, it can never again contribute to a target node’s activation. Features are discarded on a per target node basis. Discarding is done only when training and only in \(-train\) mode, every 1000 examples. One more application of discarding is then performed when SNoW finishes training on all examples. Note that features cannot be discarded until they have become eligible, and they will never be eligible again in the same invocation of SNoW after they have been discarded.

**Absolute** discarding discards features in a target node’s weight vector with weight less than some threshold (specified as a fraction of the default weight). For example, specifying \(-d \abs:0.1\) on the command line means that when the feature’s weight drops below 10% of the initial weight, the feature is discarded.

**Relative** discarding compares feature weights across the whole network. The feature with the smallest weight is discarded, and its weight is subtracted from the weight of every other feature in the network. This method was developed specifically for Winnow networks and is probably best used only with Winnow networks.

This parameter is not available in \(-interactive\) mode. The default is \(none\).

\(-e <count | percent>:\langle i\rangle\) : This parameter controls when the weights of linked features become eligible to participate in target node activation calculation and updates during training. Features that are more frequently encountered in the presented examples are more likely to be considered eligible. There are two types of eligibility, \(count\) and \(percent\).

We note that although the default method is \(count\), if you are interested in generating small hypotheses we are recommending using the \(relative\) method which we call \(percent\) here. See [Carlson et al., 2001] for experimental evidence.

Using the \(count\) method, an integer threshold is specified, and features become eligible as soon as their active counts become equal to that threshold. Before then, they are \(pending\) (not eligible, but not discarded either). See the \(-s\) parameter for a description of what makes
a feature’s active count within a given target node go up. Experiments with a small number of examples may benefit from setting this parameter to \texttt{count:1}.

For example, if \texttt{-e count:3} is specified and feature 12836 appears twice in the training file, then this feature would never be included in activation calculations during training, and it would not be written to the network file after training is complete (see the \texttt{-a} parameter for the only exception to this rule).

Using the \texttt{percent} method, the specified floating point threshold represents a fraction of total feature occurrences. During first cycle of training, all features are considered eligible. After the first cycle, a histogram of feature frequencies (active counts) is created for each target node. Starting with the highest frequencies, features are declared eligible until the total number of eligible feature occurrences within a given target node meets the specified percentage. All other features become pending. Again, see the \texttt{-s} parameter for a description of what makes a feature’s active count within a given target node go up.

For example, let \( n \) be the number of features observed with a specific target node \( t \) (one of which is feature 12836), and let \( i_1 \ldots i_k, k < n \), be the features that are active more times than feature 12836 in examples in which \( t \) is also active. Also, let \( a_{i,t} \) be the number of times feature \( i \) is active with \( t \), and let \( \gamma \) represent the user specified percentage eligibility (as in \texttt{-e percent:}\( \gamma \)). Then, if the condition

\[
a_{t,12836} + \sum_{f=1}^{k} a_{i,f} \leq \gamma \sum_{f=1}^{n} a_{i,f}
\]

holds true, then feature 12836 will remain eligible for participation in updates and activation calculations (as will all features \( i_1 \ldots i_k \)). Otherwise, feature 12836 will become pending.

The \texttt{-e} parameter also affects \texttt{-test} mode. When the network is read in, each feature’s eligibility (either “eligible” or “pending”) is assigned according this parameter’s setting and the active count found for the feature in the network file. This parameter is not available in \texttt{-interactive} mode. The default setting of this parameter in both training and testing is \texttt{count:2}.

\texttt{-f} \( \leftrightarrow \mid \rightarrow \) : This parameter controls the automatic insertion of the “fixed” feature into every example. When set to \( + \), a feature with ID equal to the highest integer that can be represented by an \texttt{unsigned int} in C++ is inserted into every example as it is read from disk. When set to \( - \), no extra feature is added.

The fixed feature acts as a dynamic threshold in Winnow and Perceptron learners. The total activation produced by each example (including the fixed feature) is still compared to the constant threshold specified on the command line to determine if a mistake has been made (unless \texttt{-D +} has also been specified), but with the fixed feature enabled, this threshold can be thought of as merely the constant component of a dynamic threshold. This parameter is available in every execution mode. Default \( + \).

\texttt{-g} \( \leftrightarrow \mid \rightarrow \) : If \texttt{-g +} is specified, SNoW automatically generates conjunctions of features active in each example. For each pair of active features, a new feature is generated using a set mapping, where the pair of feature IDs \( (i,j) \) is mapped to the new ID \( 10000j + i \), where \( i \) is the smaller of the pair. Note that in order for this mapping to work, all feature IDs used must be less than 10000. Also, using this parameter with many active features can significantly increase computation time and memory usage.
If -g - is specified, no conjunctions will be generated. If neither is specified, SNoW will examine the training data to decide if conjunctions would be useful. SNoW decides to generate conjunctions automatically if fewer than 100 unique features are present in the training data. Users also have the option of writing the new examples to disk by specifying an additional argument. If -g +,+ is specified, input examples will be written to disk with conjunctions added. They will be output into a file whose name is the original filename concatenated with “.conjunctions”. If no conjunctions are generated, the file will be left empty.

The setting of the first argument to this parameter is written to the network, but the second argument is not. This parameter is not available in -interactive mode. The default is -g <unset>,-, when SNoW will examine the training data to decide if conjunctions should be generated and no examples are written to disk.

-M <+ | -> : This parameter controls the storage of examples in memory and only has an effect on -train mode. Setting it to - makes SNoW parse and train one example at a time. At the beginning of the next cycle, the input stream is rewound, and the parsing and training process begins again. Setting it to + makes SNoW parse and store every training example in memory before training begins. This uses much more memory, but runs quicker and quicker in comparison to the alternate setting as the number of training cycles increases. This parameter is not available in -interactive mode. Default -.

-m <+ | -> : This parameter specifies whether examples should be treated as having multiple labels. If -m + is specified, the targets that appear in an example will not be treated as features by other targets, and thus a target will not be learned as a function of other targets in training. In testing, since all targets that appear in the example are treated as labels when this option is set to +, SNoW will count the example as correct if any of the targets that appear in the example are predicted.

If -m - is specified, a given target will treat other targets as features, and will therefore use those other target’s strengths in the example for activation calculations in training. In testing, the example is only counted as correct if the predicted target is the first target found in the example.

This parameter can be set in every execution mode, but will not make any difference in any mode unless there exist examples with more than one target active. This parameter is not available in -interactive mode. Default +.

-r <i> : Specifies the number of cycles (rounds) through the training data. Multiple passes through the training data can sometimes improve the resulting network. Only Perceptron and Winnnow are affected by this parameter. This parameter is not available in -interactive mode. Default 2.

-s <s | f> : Specifies whether to use the sparse or full network option. This setting only affects Perceptron and Winnnow learners. In a sparse network (-s s), features are only linked to target nodes if both the feature and the target have appeared active together in an example. In contrast, a full network (-s f) has each target node linked to the same set of features. That is, if a feature is linked to any target node, then it is linked to all target nodes in the network. Also, in a sparse network, the active count of a given feature in a given target node is incremented only when that target ID and feature ID are active in the same example. In a full network, the active count of a given feature is incremented in every target node whenever that feature ID is active in any example.
In either kind of network, active counts are only incremented during the first cycle of -train mode and when -i + is specified in -test mode. This is important for eligibility (see -e), the rules of which are applicable whether the network is full or sparse. This parameter is not available in -interactive mode. Default s.

-u <+ | -> : This parameter is for disabling first cycle updates. Setting it to + means that mistakes will trigger updates during the first round of training with Perceptron and Winnow learners. Setting it to - means that mistakes will not trigger updates (in fact, nothing will) during the first round of training. This can be useful if it is important that all eligibility decisions be made before any updates take place (see the -e parameter). This parameter can only be specified in -train mode. This parameter is not available in -interactive mode. Default +.

-z <+ | -> : This parameter is for enabling “raw” mode, a mode of operation in which all of SNoW’s bells and whistles are turned off so that SNoW’s output is more easily compared with hand calculated results. Setting this parameter to + is exactly equivalent to setting Perceptron’s initial feature weight to 0, Winnow’s initial feature weight to 1, and setting the following combination of command line parameters with every other optional parameter taking its default value:

-e count:1 -f -g -r 1 -s f

If any of these parameters are specified in addition to -z + on the command line, the extra command line settings will override those that -z + imposes no matter where in the command line they occur. This parameter is not available in -evaluate or -server mode. This parameter is not available in -interactive mode. Default -.

Testing Parameters

The following parameters are all optional. As in all other cases, they can be defined as parts of the architecture file or in the command line. None of the settings of these parameters are ever written to the network file.

-b <k> : Specifies a floating point smoothing parameter to be used when testing with naive Bayes. When a feature that was never encountered during training is encountered in a testing example, its strength is multiplied by the smoothing parameter (instead of a weight from the weight vector) and then the product is subtracted from the total activation of the example instead of being added to it. This parameter is not available in -interactive mode. The default is 15, which essentially makes features that were never encountered in the same example as target node t contribute to t’s activation as they would have if they were active in roughly $3.06 \cdot 10^{-7}$ of the examples in which t was also active.

-i <+ | -> : This parameter specifies whether to use incremental learning. If -i + is specified, then mistakes made during testing are used to update the network immediately after the example is tested. The network is written out after testing all examples in the input with “.new” appended to its original filename. If a feature was originally not written to the network due to discarding or eligibility, it can still be added in incremental learning during a different invocation of SNoW, due to the fact that only information on eligible features is written to the network file. This parameter is not available in -interactive, -evaluate, or -server modes. Note that in this mode the examples need to be labeled. Default -.

\[\text{See the -a parameter for a way to include pending features in the network}\]
-L \texttt{<k>}: Specifies a long value used to limit the amount of target IDs displayed during output with the various \texttt{-o} settings. More specifically, when an output mode that lists all targets and information about them is enabled and \texttt{-L \texttt{<k>}} is enabled as well, only the first \textit{k} targets will appear in the list. Note that these are the targets with the \textit{k} highest sigmoid activations.

-1 \texttt{<+ | ->} : This parameter specifies whether test examples are labeled or not. When set to +, SNoW checks every feature ID in every example in the input to determine if it's a target ID. When set to -, SNoW doesn't check to see if a feature is a target, which can make example parsing faster but also disables the output of evaluation statistics (just the information about how accurate SNoW's predictions were). This parameter is not available in \texttt{-interactive} or \texttt{-evaluate} mode. Default '+'.

-p \texttt{<k>}: This parameter specifies a floating point prediction threshold which must be met in order for SNoW to make a prediction. This parameter can be used as a prediction confidence filtering for cases in which SNoW is not confident enough in its prediction. When testing, if the prediction confidences of the targets with the two highest prediction confidences differ by less than the prediction threshold, no prediction is made. A target ID of -1 is then output if the output mode is \texttt{accuracy} or \texttt{winners}. [Carlson et al., 2001] discusses this option and describes some experiments exhibiting its usefulness. Default 0.

-w \texttt{<k>}: Specifies a smoothing value for Winnow and Perceptron learners. When a feature that was never encountered during training is encountered in a testing example, its strength is multiplied by the smoothing parameter (instead of a weight from the weight vector) and then the product is subtracted from the total activation of the example instead of being added to it. Default 0.

\textbf{Input/Output Parameters}

The following parameters are optional unless otherwise stated. As in all other cases, they can be defined as parts of the architecture file or in the command line. None of the settings of these parameters are ever written to the network file.

-\texttt{c \texttt{<i>}}: This setting controls the generation of results that can be used to create a learning curve. It can only be used in \texttt{-train} mode, and it must be accompanied by the \texttt{-T} parameter. When its integer argument is positive, it denotes an interval of training examples. After each interval of the specified length within a given cycle (see \texttt{-r}), the network is tested with the examples in the file specified by the \texttt{-T} parameter and results are output. If \texttt{-u} is specified, no testing or results output will occur during the first cycle. The presence of the \texttt{-T} parameter (both with and without the \texttt{-c} parameter) also causes the network to be tested when training has entirely completed. We suggest setting \texttt{-u -r 2} when using a positive argument for this parameter. This parameter is not available in \texttt{-interactive} mode. The default is 0, when the network is never tested during training.

-\texttt{E \texttt{<errorfile>}}: Specifies the name of a file in which to write information about mistakes during testing. If the file already exists it is overwritten. This option is only supported by the \texttt{-o accuracy} output mode (which is the default). If \texttt{-o} is set otherwise, no error file will be created. No default.

-\texttt{F \texttt{<networkfile>}}: Specifies the name of a file which the resulting network is written to (after training) or read from (for testing). This parameter is required in every mode of execution. No default.
-I <inputfile> : Specifies the input file from which examples are read. During training and
testing the input file specifies the training examples and testing examples, respectively. This
parameter is required in -train and -test modes and unavailable in -evaluate and -server
modes. No default.

-o <accuracy | winners | softmax | allpredictions | allactivations | allboth> :
Specifies which output mode to use when reporting results during test mode. All output
modes output information on the algorithms used as well as the overall accuracy of the test.
If this information is irrelevant to your application, consider the -v off verbosity mode.
Several output modes are available, and are described below. The default is accuracy.

accuracy : This output mode requires labeled examples. SNoW compares each prediction
to each example's label and keeps track of correct and incorrect predictions, outputting
an accuracy report at the end of testing.

Algorithm information:
Winnow: (1.35, 0.741, 4, 0.2) Targets: 0-2
52 test examples presented
Overall Accuracy - 96.15% (50 / 52)

winners : This mode outputs the target ID with the highest prediction confidence for each
example in the test set. For example, if we had targets with ID's 0, 1, and 2, the output
appearing in between the algorithm information and the overall accuracy report might
appear as:

1
2
0
2
1

On the other hand, if we had a network containing only target node with ID 2, the
single-target testing policy would take effect\(^3\), and the output appearing in between the
algorithm information and the overall accuracy report might appear as:

negative
positive
negative
negative
positive

softmax : For each example, this mode outputs the softmax normalized activations of each
target. The output is sorted by activation. If the test data is labeled, the correct label
for each example is output and the correct target's output is marked with an asterisk.

Example 47 Label: 0
0: 0.77582*
1: 0.14876

\(^3\)See Section 4.4 for more information on testing policies.
2:  0.075420

Example 48 Label:  1
2:  0.41475
1:  0.38784*
0:  0.19741

allpredictions: This mode outputs, for every example, a list of all targets and their predictions (1 or 0, indicating if it was the chosen target for this example or not). The target with the highest prediction confidence is predicted as true (1), and the rest are false (0). The output is sorted by ID. If the test data is labeled, the correct label for each example is output and the correct target’s output is marked with an asterisk.

Example 47 Label:  0
0:  1*
1:  0
2:  0

Example 48 Label:  1
0:  0
1:  0*
2:  1

allactivations: For each example, this mode outputs the activations of each target. The first activation is the prediction confidence (see Section 4.3.5 for more details on prediction confidence). The second is the target’s raw activation (in other words, it is simply the dot product of the target’s weights and the example\(^4\)). The third is the softmax normalized activation. The output is sorted by activation. If the test data is labeled, the correct label for each example is output and the correct target’s output is marked with an asterisk.

Example 47 Label:  0
0:  0.70005  4.8475  0.77582*
1:  0.30915  3.1959  0.14876
2:  0.18493  2.5167  0.075420

Example 48 Label:  1
2:  0.67585  4.7348  0.41475
1:  0.66099  4.6677  0.38784*
0:  0.49811  3.9924  0.19741

allboth: This mode outputs, for every example, a list of all targets and their activations and predictions as described in the above modes. If the test data is labeled, the correct label for each example is output and the correct target’s output is marked with an asterisk.

Example 47 Label:  0
0:  1  0.70005  4.8475  0.77582*

\(^4\)In the case of clouds containing more than one target node, this activation is the unweighted summation over all target nodes that learn the class label of those dot products.
1: 0 0.30915 3.1959 0.14876
2: 0 0.18493 2.5167 0.075420

Example 48 Label: 1
2: 1 0.67585 4.7348 0.41475
1: 0 0.66099 4.6677 0.38784*
0: 0 0.49811 3.9924 0.19741

-R <results_file> : Specifies the name of a file in which testing results and any extra information generated by a non-default verbosity setting are output (see the -v parameter). If this parameter is unspecified, the output will be directed to the console. No default.

-T <testing_file> : Using this parameter, a network can be trained and tested in the same invocation of snow. After training the network, test examples are read from the file specified using this parameter, and output is given just as if SNoW was run in -test mode. No default.

-v <off | min | med | max> : Specifies the verbosity, or the level of detail at which output is given. The default is min.

  off : This mode turns off the output.
  min : This mode outputs minimal informational messages.
  med : This mode outputs a moderate information, such as when target nodes are updated for mistakes during training, and outputs a list of features at the end of training.
  max : This mode outputs lots of information, such as when target nodes are updated for mistakes during training, showing each example before the network begins to process it, and outputting a list of features at the end of training. All predictions with activations are output during testing.

-x <example> : This parameter is only legal in -evaluate mode, and it must be specified in that mode. The example can take any form that is legal in an example file. It is usually necessary to surround the example in quotes in most shells. No default.
Chapter 6

File Formats

SNoW uses two main types of files. Example files store the training and testing examples and are treated as read-only by SNoW. Network\(^1\) files are written by SNoW during training and are read during testing. They store the representation of each of the target nodes, as well as information about the structure of the SNoW network. In addition SNoW creates error files and files for reporting results when requested.

6.1 Example Files

Example files are ASCII text files. In their basic format, SNoW represents Boolean examples. Each example consists of a list of non-negative integer IDs, separated by a single comma and any amount of whitespace, and terminated by a colon. The largest legal ID (either feature or target) is \(2^{32} - 4\). It is common practice to place a single example on each line, but it is not required. SNoW ignores all whitespace. Each number uniquely represents a feature or a class label (target ID), and they can appear in any order within the example. The appearance of an ID within an example is an assertion stating that the feature or target is active within the example.

During training, SNoW will treat each given example as positive for each target node whose ID is active in the example and negative for each target node whose ID does not appear in the example. All examples are presented to all target nodes except those target nodes that learn with naive Bayes for which each target node is only trained on positive examples. Here are two examples:

7, 5, 1, 13:
0, 3, 1234, 123456, 12, 987, 234, 556:

Labels are not required in testing unless the user wants SNoW to keep performance statistics.

SNoW can also deal with continuous features. This is done by associating a *strength* with each feature index (ID). Each occurrence of an ID may be followed by a floating point strength surrounded by parentheses. For instance:

7(1.5), 5, 10(0.6), 13(-3.2):

If no parentheses appear after the feature ID, it is equivalent to giving the feature a strength of 1.

\(^{1}\)Network is the term we use for the learned hypothesis; a network file thus contains this hypothesis as well as some of the parameters relevant for evaluating it.
In testing, the Winnow, Perceptron, and naive Bayes learning algorithms multiply each feature's weight within a target node by its strength in the example before adding it to that target node's activation. Winnow and Perceptron also do it during training. (Naive Bayes does not calculate activations during training.) When a mistake is made and weights are updated, Perceptron multiplies its learning rate parameter by the strength of the feature being updated before adding or subtracting the total to that feature's weight. Winnow raises the appropriate multiplier (either alpha or beta) to a power equal to the strength of the feature being updated before multiplying that feature's weight into the total.

Features IDs that represent targets may also be given a strength that is different than 1. Doing so will not affect SNoW's behavior at all unless the multiple labels option is disabled (see −m) or a function approximation algorithm is enabled (see −G or Section 4.3.3). See sections 4.1 and 4.2 for more details on the role feature strengths play in learning.

6.1.1 The Sequential Model

As described in Section 4.3.4, the Sequential Model is supported by a slight modification to the example format, which allows examples to specify which target nodes they should be presented to. After the last active feature in the example but before the colon, insert a semicolon followed by a list of target IDs. Examples with this structure added to it will only be presented to the target nodes whose IDs are listed after the semicolon. Examples without this structure will be presented to all target nodes. IDs found after the colon are not assertions stating that ID is active in the example, and they cannot have strengths attached to them. For instance, in a learning scenario with target IDs 3, 5, 10, and 16, the following example will only be presented to target nodes 5 and 10:

5, 9, 12, 25(0.32), 45(0.87), 103, 104; 5, 10:

SNoW is always capable of reading examples of this form. No extra command line parameter need be set; SNoW will automatically handle examples of this form in the manner described above. Note that omitting all target IDs between the semi-colon and colon at the end of an example is equivalent to also omitting the semi-colon.

6.1.2 Interactive Mode

SNoW's interactive mode has a slightly different example file format. Associated with each example in this mode is a single character which instructs SNoW to take a particular action on this example. Currently, there are three possible actions:

- Evaluate:
  Syntax: e,f_1,f_2,...,f_n :

  For example, the following line requests SNoW to evaluate the example with active features 1001, 1005, 1009, 1010, and 1200.

  e, 1001, 1005, 1009, 1010, 1200:

  The output format depends on the value of parameter −a. If the value is accuracy, which is the default, then no output will be given.

36
<table>
<thead>
<tr>
<th>label</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>SNoW Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0, 2:</td>
</tr>
<tr>
<td>true</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1, 6:</td>
</tr>
<tr>
<td>false</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0, 4, 5:</td>
</tr>
<tr>
<td>false</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0, 2, 4, 5, 7:</td>
</tr>
<tr>
<td>true</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1, 3, 5:</td>
</tr>
<tr>
<td>false</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0, 2, 5, 7:</td>
</tr>
<tr>
<td>true</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1, 2, 4, 5, 6:</td>
</tr>
<tr>
<td>true</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1, 3, 5, 6:</td>
</tr>
<tr>
<td>false</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0, 2, 4, 7:</td>
</tr>
</tbody>
</table>

Table 6.1: Partial truth table and SNoW examples for the concept $x_2 \lor x_5$. SNoW (with default parameters) will create two target nodes, and each can be said to learn a separate concept. One learns to build a bigger activation for positive examples (label 1), and the other learns to build a bigger activation for negative examples (label 0).

- Promote / Demote specific targets:
  Syntax: $[pd], f_1, f_2, \ldots, f_n; t_1, t_2, \ldots, t_m$:
  For example, the following line asks SNoW to promote targets 1, 3, and 5 for the example with active features 1002, 1013, 1018, and 1011.
  ```plaintext
  p, 1002, 1013, 1018, 1011; 1, 3, 5:
  ```
  Similarly, the following line demotes targets 2 and 4 for the example with active features 1004, 1007, 1212, 1311, and 1312.
  ```plaintext
  d, 1004, 1007, 1212, 1311, 1312; 2, 4:
  ```
  These two training control commands won’t generate any output.

6.1.3 Example
For example, consider a learning problem where a boolean concept over 6 boolean variables is to be learned. The value 1 might be used as a target ID to represent the label True, and the value 0 the label False. The values 2 through 7 could then be features representing the variables $x_1$ through $x_6$ respectively. Let’s say that, unbeknownst to SNoW, we have training data that represents the concept $x_2 \lor x_5$. A partial truth table for this concept and the corresponding SNoW examples are given in table 6.1. Look in the `booleanexample/` subdirectory of the software distribution to see scripts that show how SNoW might be used to train a network over these examples.

6.2 Network Files
Network files (specified by the `-F` command line parameter) contain the representation learned by SNoW for each of the target nodes as well as all the information required for SNoW to recreate (in testing) the structure learned during training. Each target node has a section in the file containing information on its algorithm’s parameters and features (most notably, the features’ weights). The header line for each target node takes the following form:
target ID priorProbability cloudConfidence activeCount nonActiveCount algorithm learnerType parameters

For example:

target 2 0.4 0.473593433165 42 63 winnow 1 1.35 0.8 4 0.2

This specifies a target node with ID 2, prior probability 0.4 (the fraction of training examples labeled with the target ID), and cloud confidence 0.4736 (see Section 4.3.5), which appeared active in 42 examples, and was not found active in 63 examples. It uses the Winnow algorithm found at index 1 of an internal data structure that maintains information on every algorithm used by the network and their parameters. The Winnow algorithm used by this target node has an $\alpha$ of 1.35, a $\beta$ of 0.8, a threshold of 4, and a default feature weight of 0.2.

Following the target node header, the target node’s features are enumerated. Each line corresponds to a single feature, in the format:

$ID : learnerType : featureID : activeCount updates weight$

For example:

1 : 2 : 34 : 13 6 0.3645

This specifies a feature which is linked to target node 1 using algorithm 2, and its ID is 34. It had an active count of 13, and its weight was updated 6 times during training. After those 6 updates, its weight is now 0.3645.

For those users who are concerned about a feature with very large ID found in their network files, take a look at the -f command line parameter.

6.3 Error Files

Error files (generated with the -E command line parameter) can sometimes provide insight into why the network failed to learn the target concept exactly. Error files can only be created when testing using labeled examples and using the default output mode (-o accuracy). The created error file will contain a list of examples for which SNoW made an incorrect prediction. Each example’s listing shows the example number and each target’s prediction confidence and activation.

For example, in the following error file, we see there were mistakes on the 3rd and 4th examples in the input file. We also see the activations calculated by each target. The first activation is the prediction confidence (see Section 4.3.5 for more details on prediction confidence). The second is the target’s raw activation (in other words, it is simply the dot product of the target’s weights and the example$^2$). The correct target’s output is marked with an asterisk. Note that this output is the same as that generated by the -o allactivations output mode.

Ex: 3 Prediction: 1 Label: 0
1: 0.1824 4.5

$^2$In the case of clouds containing more than one target node, this activation is the unweighted summation over all target nodes that learn the class label of those dot products.
0:  0.1192 4*
Ex:  4 Prediction:  1 Label:  0
1:  0.19193 4.5625
0:  0.18243 4.5*
Chapter 7

Tutorial

This tutorial is meant to demonstrate how to use many of the basic options of SNoW. We show how to start with training examples, train a classifier, and then test the classifier with some more examples. The example task given is context-sensitive spelling correction [Golding and Roth, 1999]. The task is to train a classifier with many examples of the correct usage of the words “their” and “there” so that, given a test context, the classifier can decide which word best fits the context.

A script demonstrating some of the topics discussed here is included in the tutorial/ subdirectory of the software distribution called tutorial.sh. It can easily be modified and used as a general purpose SNoW script.

7.1 Training

We start with a file containing labeled examples. Our target concepts have IDs 0 and 1, and all other numbers appearing in the examples represent other features present (word collocations, parts of speech, etc.). The first example in the training file is:

0,96,116,119,120,128,138,157,212,230,328,451,454,601,636,641,
646,773,774,815,872,897,937,1134,1160,1197,1231,1267,1461,1503,
2387,2612,2958,3211,3221,3222,3233,3242,3308,3315,3318,3487,
3524,3526,3897,4037,4136,4404,6933,6991,7269,7298,7398,7488,
7539,7562,7755,7794,8032,8377,9336:

Here, the example has a label of 0, meaning that it will be a positive example for target 0 (the word “their”) and a negative example for all other targets (in our case, just target 1, the word “there”). In training, labels are always present, but their location in the list is not restricted. An example with no active target IDs has a negative label for all targets. All examples are terminated with a colon.

The original sentence from which the above example was generated is:

In the interim between now and next year, we trust the House and Senate will put << their >> minds to studying Georgia’s very real economic, fiscal and social problems and come up with answers without all the political heroics.

The angled brackets were added for emphasis. The above example was generated from this sentence
using Fex (Feature Extractor)\textsuperscript{1}, a program which generates features based on specified relational definitions. In this example features include words around the target word, parts of speech in close proximity to the target word, and simple conjunctions of those.

Given our training data (provided in the file \texttt{traindata.snow}), we can now train a classifier which will be able to classify new examples from outside the training set based on what the system learned about the features present in the training data. In order to train our network, we must invoke SNoW in training mode with our training examples as the input file. We do this as follows:

\texttt{\textgreater{} snow -train -I tutorial/traindata.snow -F tutorial/test.net \textasciitilde{}w :0-1}

This gives the output:

\texttt{SNoW+ - Sparse Network of Winnows Plus}
\texttt{Cognitive Computations Group - University of Illinois at Urbana-Champaign}
\texttt{Version 3.2.0}
\texttt{Input file: 'tutorial/traindata.snow'}
\texttt{Network file: 'tutorial/test.net'}
\texttt{Training with 2 cycles over training data.}
\texttt{Directing output to console.}
\texttt{Network Spec \rightarrow w(:0-1),}

The output from SNoW lets us know if there were any errors in the parameters we entered, and also gives information on the learning algorithm used. Here, we used a Winnow learning algorithm with default parameters by specifying \texttt{-w :0-1} on the command line. This tells SNoW to use a default set of parameters (which work quite well for many experiments) and that our target concepts have IDs 0 and 1. Different algorithms and parameters can be specified on the command line or in an architecture file, as will be shown later in the tutorial.

The training made two cycles through our training data, which is the default. The number of cycles can be specified on the command line, and generally, the more cycles used, the closer the classifier comes to completely learning the training data.

\section*{7.2 Testing}

We now have our network file, \texttt{test.net}, which contains the parameters of our Winnow algorithm as well as weights for all of the features which appeared in our training examples. Now that we've trained our network, we can test it on more examples.

\texttt{\textgreater{} snow -test -I tutorial/testdata.snow -F tutorial/test.net}

Our test file contains labeled examples of exactly the same format as those used in training, and we used the default output mode to simply let SNoW score its accuracy on those examples. In this mode, each example is given to the system and the resulting prediction output by the classifier is compared to the example's label. A mistake is scored if the two do not match. Here are our results:

\texttt{SNoW+ - Sparse Network of Winnows Plus}
\texttt{Cognitive Computations Group - University of Illinois at Urbana-Champaign}

\footnote{Fex can be downloaded from our website (http://L2R.cs.uiuc.edu/~cogcomp).}
Version 3.2.0
Input file: 'tutorial/testdata.snow'
Network file: 'tutorial/test.net'
Directing output to console.
Algorithm information:
Winnow: (1.35, 0.8, 4, 0.3245) Targets: 0-1
850 test examples presented
Overall Accuracy - 96.71% (822 / 850)

We can also verify how well the training data was learned by testing with our training set (that
is, use the file tutorial/traindata.snow after the -I above).

To receive output on a more detailed level, we can specify different output modes on the com-
mand line with the -o outputmode parameter. For example, try executing SNoW as follows:

> snow -test -I tutorial/testdata.snow -F tutorial/test.net -o allactivations

This gives the output:

SNoW+ - Sparse Network of Winnows Plus
Cognitive Computations Group - University of Illinois at Urbana-Champaign
Version 3.2.0
Input file: 'tutorial/testdata.snow'
Network file: 'tutorial/test.net'
Directing output to console.
Algorithm information:
Winnow: (1.35, 0.8, 4, 0.3159) Targets: 0-1
Example 1 Label: 1
1: 0.98732 8.3553*
0: 0.352 3.3897

Example 2 Label: 0
0: 0.85602 5.7826*
1: 0.082134 1.5863

Example 3 Label: 1
1: 0.73296 5.0097*
0: 0.066876 1.3643
...

See the description of the -o command line parameter for more details about the information it
produces.

7.3 Server

In this tutorial, we only have one training file and one testing file. If, however, we have many testing
files, we can use SNoW's -server mode to save time. SNoW's -server mode loads the network
file and then waits for client programs to send examples to it. It classifies the examples and sends
the results back to the client. This way, the network need not be reloaded for each testing file.

To instantiate SNoW in server mode, type the following:

```bash
> snow -server 4444 -F tutorial/test.net &
```

The port number has been set to 4444. Any valid port number can be substituted. This command line will run SNoW in the background in -server mode. Next, using the example client script also provided in the tutorial directory, we can send the examples in `tutorial/testdata.snow` to the server.

```bash
> cd tutorial
> ./example_client.pl -s 4444 -I testdata.snow > output.txt
```

Results sent back from the server are stored in the file `output.txt`.

The example client can also be instantiated with:

```bash
> ./example_client.pl --help
```

to display usage notes including a list of command line parameters that correspond to SNoW command line parameters. Specifying any of those command line parameters to the `example_client.pl` script will make the script send them to the server. Take a look inside the `example_client.pl` script for detailed comments explaining how to write a client program that works with a SNoW server. Currently, the only way to stop the server is to kill it manually.

### 7.4 Interactive Mode

Interactive mode is useful for users who want complete control over all training decisions and the ability to make those decisions based on the results from testing other examples. The user can run this mode with a prepared input example file, but its full functionality is utilized only when an external program hooks into SNoW’s `STDIN` and `STDOUT`. For instance:

```bash
> snow -interactive -I /dev/stdin -F snow.net -W 1.05,0.95,4,0.03:1-44 \ 
-o winners
```

Recall that it is also necessary to specify every algorithm parameter.

### 7.5 Other Options

If we want to use algorithms other than the default, we can specify them with the `-W`, `-P`, and `-B` parameters and their associated options. Each algorithm we define can be assigned to any number of targets. To run the tutorial experiment with algorithms we specify, we can run SNoW as follows:

```bash
> snow -train -I tutorial/traindata.snow -F tutorial/test.net \ 
-W 1.5,0.8,4,0.05:0 \ 
```

---

2The top of the `example_client.pl` script contains a couple of machine specific paths that will need to be modified before the script will execute on your machine. Make sure to read the first two comments in that script before attempting to execute it.
-P 0.1,4.0,0.20:1

Here, a Winnow algorithm was used for target 0 and a Perceptron was used for target 1.

We could have also executed SNoW like this:

```shell
> snow -train -I tutorial/traindata.snow -F tutorial/test.net \ 
-W 1.5,0.8,4.0,0.5:0-1 \ 
-P 0.1,4.0,0.20:0-1
```

This will use both a Winnow and Perceptron on each target, combining the results of the algorithms to calculate a single activation for each target.

Command line parameters can also be specified in an *architecture file*. This file is specified with the -A parameter, and contains parameters which don’t need to be changed frequently. For example, when running an experiment over many datasets, the only parameters which change from dataset to dataset will usually be the -I inputfile and the -F network file parameters. Therefore, we can use an architecture file to save typing most of our command line parameters. Our architecture file might be named “archfile”, and it might read:

```plaintext
-W 1.5,0.8,4.0,0.5:0-1
-P 0.1,4.0,0.20:0-1
-e count:1
-r 4
```

Using this architecture file, we have defined two algorithms, set the *eligibility threshold* to 1, and set SNoW to run through the training data four times. Now we can use these parameters with any data files. We can execute SNoW with the architecture file as follows:

```shell
> snow -train -I tutorial/traindata.snow -F tutorial/test.net -A archfile
```
Appendix A

Command-Line Parameter Index

Each listing displays the parameter’s name, syntax, a description of acceptable arguments, its default setting (if any), and a page reference to a more detailed description in Chapter 5.

<table>
<thead>
<tr>
<th>Name</th>
<th>Syntax</th>
<th>Argument Description</th>
<th>Default</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture file</td>
<td>-A &lt;file&gt;</td>
<td>Input filename.</td>
<td>none</td>
<td>24</td>
</tr>
<tr>
<td>Also write pending features</td>
<td>-a ±</td>
<td>‘+’ or ‘-’.</td>
<td>-</td>
<td>27</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>-B ...</td>
<td>See Chapter 5.</td>
<td>none</td>
<td>15, 24</td>
</tr>
<tr>
<td>Naive Bayes smoothing</td>
<td>-b &lt;k&gt;</td>
<td>A real number.</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>Learning curve generation</td>
<td>-c &lt;i&gt;</td>
<td>A non-negative integer.</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>Discard method</td>
<td>-d ...</td>
<td>See Chapter 5.</td>
<td>none</td>
<td>27</td>
</tr>
<tr>
<td>Error file</td>
<td>-E &lt;file&gt;</td>
<td>Output filename.</td>
<td>none</td>
<td>31</td>
</tr>
<tr>
<td>Eligibility policy</td>
<td>-e ...</td>
<td>See Chapter 5.</td>
<td>count:2</td>
<td>27</td>
</tr>
<tr>
<td>“Fixed” feature</td>
<td>-f ±</td>
<td>‘+’ or ‘-’.</td>
<td>+</td>
<td>28</td>
</tr>
<tr>
<td>Gradient Descent</td>
<td>-G ±</td>
<td>‘+’ or ‘-’.</td>
<td>-</td>
<td>17, 26</td>
</tr>
<tr>
<td>Generate conjunctions</td>
<td>-g ±[,±]</td>
<td>‘+’ or ‘-’ once or twice.</td>
<td>&lt;unset&gt;,</td>
<td>28</td>
</tr>
<tr>
<td>Incremental learning</td>
<td>-i ±</td>
<td>‘+’ or ‘-’.</td>
<td>-</td>
<td>30</td>
</tr>
<tr>
<td>Target output limit</td>
<td>-L &lt;k&gt;</td>
<td>A positive integer.</td>
<td>ULONG_MAX</td>
<td>31</td>
</tr>
<tr>
<td>Labeled examples</td>
<td>-l ±</td>
<td>‘+’ or ‘-’.</td>
<td>+</td>
<td>31</td>
</tr>
<tr>
<td>Store examples in memory</td>
<td>-M ±</td>
<td>‘+’ or ‘-’.</td>
<td>-</td>
<td>29</td>
</tr>
<tr>
<td>Multiple labels</td>
<td>-m ±</td>
<td>‘+’ or ‘-’.</td>
<td>+</td>
<td>29</td>
</tr>
<tr>
<td>Constraint Classification</td>
<td>-O ±[,±]</td>
<td>‘+’ or ‘-’ once or twice.</td>
<td>-,-</td>
<td>16, 26</td>
</tr>
<tr>
<td>Output mode</td>
<td>-o ...</td>
<td>See Chapter 5.</td>
<td>accuracy</td>
<td>32</td>
</tr>
<tr>
<td>Perceptron</td>
<td>-P ...</td>
<td>See Chapter 5.</td>
<td>none</td>
<td>15, 24</td>
</tr>
<tr>
<td>Prediction threshold</td>
<td>-p &lt;k&gt;</td>
<td>A real number.</td>
<td>-1</td>
<td>31</td>
</tr>
<tr>
<td>Results file</td>
<td>-R &lt;file&gt;</td>
<td>Output filename.</td>
<td>none</td>
<td>34</td>
</tr>
<tr>
<td>Training rounds (cycles)</td>
<td>-r &lt;i&gt;</td>
<td>A positive integer.</td>
<td>2</td>
<td>29</td>
</tr>
<tr>
<td>Thick separator</td>
<td>-S &lt;p&gt;[,&lt;n&gt;]</td>
<td>One or two real numbers.</td>
<td>0</td>
<td>17, 26</td>
</tr>
<tr>
<td>Sparse network</td>
<td>-s &lt;s</td>
<td>f&gt;</td>
<td>‘s’ or ‘f’.</td>
<td>s</td>
</tr>
<tr>
<td>Testing file</td>
<td>-T &lt;file&gt;</td>
<td>Input filename.</td>
<td>none</td>
<td>34</td>
</tr>
<tr>
<td>Threshold relative updating</td>
<td>-t ±</td>
<td>‘+’ or ‘-’.</td>
<td>-</td>
<td>19, 26</td>
</tr>
<tr>
<td>First cycle updating</td>
<td>-u ±</td>
<td>‘+’ or ‘-’.</td>
<td>+</td>
<td>30</td>
</tr>
<tr>
<td>Verbosity</td>
<td>-v ...</td>
<td>See Chapter 5.</td>
<td>min</td>
<td>34</td>
</tr>
<tr>
<td>Winnow</td>
<td>-W ...</td>
<td>See Chapter 5.</td>
<td>none</td>
<td>13, 24</td>
</tr>
<tr>
<td>Winnow smoothing</td>
<td>-w &lt;k&gt;</td>
<td>A real number.</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>“Raw” mode</td>
<td>-z ±</td>
<td>‘+’ or ‘-’.</td>
<td>-</td>
<td>30</td>
</tr>
</tbody>
</table>
Appendix B

Glossary of Terms

**activation**  See either target node activation or sigmoid activation.

**cloud**  A collection of target nodes that all learn the concept represented by the same target ID $t$ in the training set. While learning, they maintain a cloud confidence measure that decreases whenever a mistake is made. During testing, the sum of their sigmoid activations is weighted by their cloud confidences, and the result is the prediction confidence that represents $t$ in the winner-take-all competition for deciding on a prediction. The term “target” (but never “target node”) is sometimes used as a synonym of “cloud”.

**cloud confidence**  A monotonically decreasing function of the number of mistakes a learning algorithm has made with respect to a particular target node. Cloud confidence values are in the range $(0, 1]$.

**confidence**  See either cloud confidence or prediction confidence.

**conjunctions of features**  See option `-g`. SNoW learns to represent each target node as a linear function over its active features. This may not be expressive enough to represent the true concept. You may want to generate more expressive features as input to SNoW, using a tool like FeX¹ or use this option available within SNoW. Note that this is recommended only if the number of active features in each example is small.

**cycle**  See training cycle.

**demotion**  The update performed on a target node’s weight vector when that target node has mistakenly classified an example labeled negative as positive is called demotion. In general this means that weights, and in turn, the target node’s activation, will be decreased.

**discarding features**  SNoW supports a mechanism for breaking the links between target nodes and features if user-specified conditions are not met. For example, the user may wish to sort features within a target node by weight and keep only the top $n\%$. A feature cannot be discarded until it has become eligible. Once a feature is discarded by a target node, it can never again become eligible within that target node during that invocation of SNoW. See the `-d` option for more details.

**eligible**  A feature status allowing that feature to contribute its weight to activation calculations and to be included in updates. SNoW is capable of complicated decisions regarding which

---

¹Follow the Software link from: http://l2r.cs.uiuc.edu/~cogcomp/
features are eligible and when they become eligible. Those decisions are made separately from the decisions that result in discarding features. See the \(-e\) and \(-d\) options for more details.

**feature** The most basic unit of data processed by a machine learning algorithm. In SNoW, features are associated with strengths in examples and with weights in target nodes. They are combined linearly to form a classification function.

**feature space** The set of all possible features that may appear active in a training example.

**feature strength** The optional floating point value that may be associated with a feature in an example. Feature strengths specify the fraction of that feature’s weight in a target node that should be added to the target node’s activation during both training and testing.

**full network** By default, SNoW generates a sparse network (see below). Option \(-s\) allows the user to override this default. The result is a network in which all features encountered during training are linked to all target nodes.

**incremental learning** SNoW’s terminology for training on testing examples. Normally, a network is written to disk after training, and that network is used but not modified during testing. The \(-i\) parameter is used to enable network updates when mistakes are made during testing.

**learning rate** An algorithmic parameter used by both Perceptron and Winnow controlling the magnitude of weight updates. Perceptron uses \textit{learning rate} and \textit{–learning rate} to update after mistakes on positive and negative examples respectively. Winnow uses \(\alpha\) and \(\beta\) in the same way.

**link** The dynamically allocated space in a target node’s representation which stores information about how a particular feature relates to that target node. SNoW’s learning architecture framework is based on the infinite attribute model in which the number of features active in a training set is potentially much smaller than the total number of features in the feature space. Therefore, instead of maintaining a weight vector in each target node with an index for every possible feature, links are allocated between target node and feature when they are seen active in the same example.

**multi-class classifier** A classification function capable of distinguishing between two or more classes. While SNoW can still be considered a multi-class classifier even when training its targets independently, it is also capable of making training decisions based on comparisons between targets. See Section 4.3.1 for more details.

**multiple labels** See option \(-m\). SNoW allows examples to have multiple labels. When training a specific target \(t\), it is learned as a function of all other features that are active with it, which may or may not include (depending on the option \(-m\)) those features that are also targets.

**naive Bayes** An off-line, probabilistic update rule. In SNoW, the statistics used to calculate the conditional probabilities naive Bayes works with are collected in an on-line fashion during training. When training is complete, those statistics are then converted to feature weights within the target nodes.

**network** An instantiation of the SNoW learning architecture framework. In addition to describing the structure of the instantiated architecture, the network contains the hypothesis (learned features’ weights), training statistics, and algorithmic parameters. For research purposes you may find it useful to look at the network or even post process it to learn more about the learned hypothesis. See Chapter 6 for its structure.
pending A feature status that is neither eligible nor discarded. Pending features do not participate in activation calculations or updates and are not written to the network file (but see the -a option for a counter-example). Unlike discarded features, pending features can become eligible. Depending on the eligibility mechanism chosen (see the -e option), features may start as eligible and then become either pending or discarded, or they may start as pending and then become eligible. A pending feature is never directly discarded; it must become eligible first.

Perceptron An on-line, mistake driven, additive update rule. Perceptron updates the weights in a target node by adding to them a learning rate that is a function of the type of mistake made (either positive or negative) and the strengths of features in the example.

policy See either testing policy or training policy.

prediction confidence A target’s prediction confidence is the sum over all its target nodes of their sigmoid activations multiplied by their cloud confidences. Note that when a target has been assigned a single algorithm to learn it, its prediction confidence is equal to its target node’s sigmoid activation. Target nodes’ prediction confidences are compared to determine if a prediction will be made (see the -p option) and to decide on a prediction in the winner-take-all testing policy.

prediction threshold The smallest difference between highest and second highest prediction confidence that SNoW will allow when deciding whether or not to make a prediction on a testing example. See option -p for more details.

promotion The update performed on a target node’s weight vector when that target node has mistakenly classified an example labeled positive as negative is called promotion. In general this means that weights, and in turn, the target node’s activation, will be increased.

sparse example An example whose active features comprise only a small fraction of all features in the feature space.

sparse function A function that has been learned over features that comprise only a small fraction of all features in the feature space.

sparse network A network in which only those features that are active in the same example as a target node $t$ are linked to $t$. This is significant for computational efficiency, size of representation, and performance. Option -s allows the user to override this default and use a full network.

sequential model A multi-class classification model in which a subset of targets in the network is selected to process data on a per example basis. For example, targets 1 through 5 may all be represented in a given network. Under the sequential model, a given example may request that it only be used to update the weights in targets nodes with IDs 2 and 3. See Section 4.3.4 for more details.

sigmoid activation The result obtained from applying a learning algorithm specific sigmoid function to a target node’s activation, having a range of $(0, 1)$. Sigmoid activations of target nodes representing the same target are combined to form prediction confidences.

single-target A testing policy which only operates on networks that contain a single target node. This policy simply compares the activation of the lone target node to the threshold of the algorithm used by that node to make a prediction.
smoothing In SNoW, smoothing is the special treatment of features in a testing example that were never encountered during training. The naive Bayes algorithm in particular requires weights of unseen features to be smoothed. Specifically, if feature \( i \) is observed in a testing example and has never been observed with target \( t \) in training, the corresponding weight \( w_i^t \) is substituted with a fixed value. That value is then multiplied by the feature’s strength in the testing example, and the result is subtracted from the target node’s activation. See option \(-b\).

strength See feature strength.

target The representation of a class label in a network. By default, each target learns to predict its own presence in the training examples independently. In SNoW’s most common usage, a target’s representation is simply a single target node. Clouds (see Section 4.3.5) allow a target to be represented by multiple target nodes. When used in that sense, the term “target” can be used interchangeably with the term “cloud”. The term “target” can also be used as short-hand for the target ID that appears in an example.

target node The data structure on which an update rule operates. Target nodes keep their own set of weighted links to features. The user then associates a (possibly different) algorithm with each target node to update those weights.

target node activation The result of the dot product of a target node’s weight vector and an example containing feature strengths. Each target node calculates a new activation for each example. See also sigmoid activation.

testing policy An algorithm used to arrive at a prediction for an example given the prediction confidences of all targets in the network. Winner-take-all and single-target are the only testing policies currently implemented in SNoW, but the various supported output modes allow the user to take advantage of the data SNoW calculates during testing however he desires.

threshold An algorithmic parameter representing the lowest activation a target node can have with respect to a given example in order to predict that its label in that example is positive.

training cycle One pass of processing over the training examples. The default of two training cycles can be overridden with the \(-r\) option.

training policy An algorithm that decides when update rules should be applied.

update rule An algorithm, usually triggered by a mistake in prediction, that modifies the weights in a target node based on the strengths in an example.

winner-take-all A testing policy in which the target with the highest prediction confidence becomes SNoW’s prediction for the example.

weight The floating-point value associated with the link between target node and feature. It plays the same role as weights in the weight vector of a classical Perceptron or Winnow implementation.

Winnow An on-line, mistake driven, multiplicative update rule. Winnow updates the weights in a target node by multiplying them by a learning rate that is a function of the type of mistake made (either positive or negative) and the strengths of features in the example.
Bibliography


