

Project

CS 6243 – Spring 2005
Tom Bylander, Instructor

assigned March 8, 2005
due May 2, 2005

The project is to implement learning algorithms for linear machines in Weka. This project may be done in groups, for which additional features will be required.

Notation

The training set will be a set of examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots$, where each $\mathbf{x} = (x_1, \dots, x_n)$ is a attribute value vector of length n and each $y \in \{0, 1, \dots\}$ is a category/class from a finite set.

Linear Machines

In a linear machine each class y is represented by a discriminant function with bias weight b_y and attribute weights \mathbf{w}_y :

$$d_y(\mathbf{x}) = b_y + \sum_j w_{yj}x_j$$

The linear machine predicts the class with the highest value. The perceptron algorithm you implemented in Lab 2 can easily be converted to a linear machine for 2 classes by defining $d_1(\mathbf{x})$ to be the perceptron and $d_0(\mathbf{x}) = -d_1(\mathbf{x})$.

To generalize a linear learning algorithm for 2 classes to any number of classes, we will use a technique called *Kesler's construction* [4]. Suppose t is the correct class for \mathbf{x} , but y is predicted by the linear machine. That is, $d_t(\mathbf{x}) - d_y(\mathbf{x}) \leq 0$. We then pretend that we have a positive example \mathbf{x}' :

$$\mathbf{x}' = (1, x_1, \dots, x_n, -1, -x_1, \dots, -x_n)$$

being classified by weights \mathbf{w}' :

$$\mathbf{w}' = (b_t, w_{t1}, \dots, w_{tn}, b_y, w_{y1}, \dots, w_{yn})$$

and apply the update rule of the learning algorithm.

Two modifications to these steps should be included. One is to include a margin, so that $d_t(\mathbf{x}) - d_y(\mathbf{x}) < 1$ should trigger the update rule. The other is that the update rule should be applied to each class y where $d_t(\mathbf{x}) - d_y(\mathbf{x}) < 1$.

Update Rules

Any update rule should be applied in the following manner:

```
\\ obtain predictions
for each class  $y$ 
     $p_y \leftarrow b_y + \mathbf{w}_y \cdot \mathbf{x}_y$ 
\\ do updates
for each class  $y$ 
     $z \leftarrow p_t - p_y$ 
    if  $y \neq t$  and  $z < 1$ 
        then apply update rule using Kesler's construction
\\ postprocessing for some update rules
```

Because of Kesler's construction, we only need to specify update rules for positive examples $(\mathbf{x}', 1)$ being classified by weights \mathbf{w}' without any bias weight.

Perceptron

Initialize weights to 0 and update using:

```
for each  $x'_j$  in  $\mathbf{x}'$ 
     $w'_j \leftarrow w'_j + \eta x'_j$ 
```

Widrow-Hoff

Initialize weights to 0 and update using:

```
for each  $x'_j$  in  $\mathbf{x}'$ 
     $w'_j \leftarrow w'_j + \eta(1 - z)x'_j$ 
```

Exponentiated Update

Exponentiated updating is more efficient if few attributes are relevant [1]. Initialize weights to 1 (or some other positive number). Use one of the following update rules:

For minimizing absolute error similar to perceptrons:

```
for each  $x'_j$  in  $\mathbf{x}'$ 
     $w'_j \leftarrow w'_j \exp\{\eta x'_j\}$ 
```

For minimizing squared error similar to Widrow-Hoff:

for each x'_j in \mathbf{x}'
 $w'_j \leftarrow w'_j \exp\{\eta(1 - z)x'_j\}$

As a postprocessing option, the sum of the weights after updating are normalized to be equal to the sum of the weights before updating:

$a \leftarrow$ sum of all weights of all classes before updating
 $b \leftarrow$ sum of all weights of all classes after updating
for each weight w
 $w \leftarrow aw/b$

Other Linear Learning Algorithms

Two of the papers to be presented describe the winnow [2] and weighted majority [3] algorithms. These algorithms can be adapted to linear machines.

Filtering

Similar to the Lab 2 assignment, you should apply the Normalize, NominalToBinary, and ReplaceMissingValues filters in that order. As an option, apply Discretize (weka.filters.supervised.attribute.Discretize) instead of Normalize first.

Grading and Groups

A group of size n must implement n of the following algorithms: Perceptron, Widrow-Hoff, Exponentiated Update, Gradient Descent Batch, Winnow, and Weighted Majority. These are listed in roughly easiest to hardest order (or so I think). Implementing exponentiated-update also means implementing the normalization option. Implementing gradient-descent-batch means that batch updating can be applied to all other algorithms that have been implemented (be careful not to end up with a nonpositive weight for exponentiated-update). Implementing winnow and weighted-majority will require some creativity to map them into the linear machine framework.

All projects should implement the Discretize option described above. You should be able to select any learning algorithm with options. All the options should work with Weka's explorer GUI.

Testing

Test your algorithms on the following UCI datasets: glass, iris, hypothyroid, letter, segment, vowel. You need to choose a learning rate in a consistent way (suggested $1/(10 *$

number of attributes)). You need to choose the same number of epochs for each dataset (at least 1000). You need to test every variation that you have implemented. For example, if you implement gradient-descent-batch, then every update rule can be run using either incremental or batch update.

Also compare to the following Weka algorithms: IBk, J48, and NaiveBayes. For IBk, choose 1-nearest-neighbor. For NaiveBayes, use the `weka.filters.unsupervised.attribute.Discretize` filter with equal frequency binning (this can be done using Weka's Explorer).

For each dataset, show the results of the algorithms sorted by their error rate.

Deliverables

I should be able to put your `LinearMachine.java` in the `weka/classifiers/functions` directory. It should work in the Explorer given an appropriate change in the `GenericObjectEditor.props` file.

Email me your lab with `LinearMachine.java` as one attachment. One other attachment should be a description of what was implemented and the algorithms' performance on the datasets listed above.

References

- [1] J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132:1–63, 1997.
- [2] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
- [3] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994.
- [4] N. J. Nilsson. *Learning Machines*. McGraw-Hill, New York, 1965.