CS578/STAT590: Introduction to Machine Learning **Fall 2014**

Problem Set 2

Gen Nishida *Handed In: October 10, 2014*

1 Questions

1. Given n boolean variables $(x_1, \cdots, x_n)$, we define our target classification function $f(\cdot)$ as $f(\cdot) = 1$ if at least 3 variables are active. For $n = 5$ show how this function can be represented as (1) Boolean function (2) Linear function.

   (1) Boolean function

   By using conjunctions for every combination of three variables, we can test if at least three variables are active. Therefore, the Boolean function for this can be represented as follows:

$$\begin{aligned} f = & (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_5) \vee (x_1 \wedge x_3 \wedge x_4) \vee (x_1 \wedge x_3 \wedge x_5) \\ & \vee (x_1 \wedge x_4 \wedge x_5) \vee (x_2 \wedge x_3 \wedge x_4) \vee (x_2 \wedge x_3 \wedge x_5) \vee (x_2 \wedge x_4 \wedge x_5) \vee (x_3 \wedge x_4 \wedge x_5) \end{aligned}$$

   (2) Linear function

   The equivalent linear function just needs to check whether the sum is more than two. Thus,

$$f = \text{sign}(x_1 + x_2 + x_3 + x_4 + x_5 - 2)$$

   where

$$\text{sign}(x) = \begin{cases} 1 & (\text{if } x > 0) \\ 0 & \textit{Otherwise} \end{cases}$$

2. Let CON_B be the set of all different monotone conjunctions defined over n boolean variables. What is the size of CON_B?

   For every variable, there are two cases, active or not active. Thus, the size of CON_B is 2^n.

3. Let CON_L be the set of all linear functions defined over n boolean variables that are consistent with the functions in CON_B. What is the size of CON_L?

   Given a monotone conjunction, one example of the equivalent linear functions is defined by the sign of the summation of all the active literals subtracted by (the number of the active literals - 1). For instance, for a monotone conjunction $f_b = x_1 \wedge x_3 \wedge x_5$, the corresponding linear function is $f_l = \text{sign}(x_1 + x_3 + x_5 - 2)$. However, the linear function can have any real nubmer as the coefficient of each literal. In the above case, for instance, another equivalent linear function would be $f_l' = \text{sign}(2x_1 + 2x_3 + 2x_5 - 4)$, and both f_l and $'_l$ are consistent with f_b. Thus, the size of CON_L is inifinite.

4. Define in one sentence: Mistake bound.

   Mistake bound is the maximum number of mistakes that can be made by an online learning algorithm, which is also used to evaluate the performance of the convergence of the algorithm.

5. Suggest a mistake bound algorithm for learning Boolean conjunctions. Show that your algorithm is a mistake bound algorithm for Boolean conjunctions.

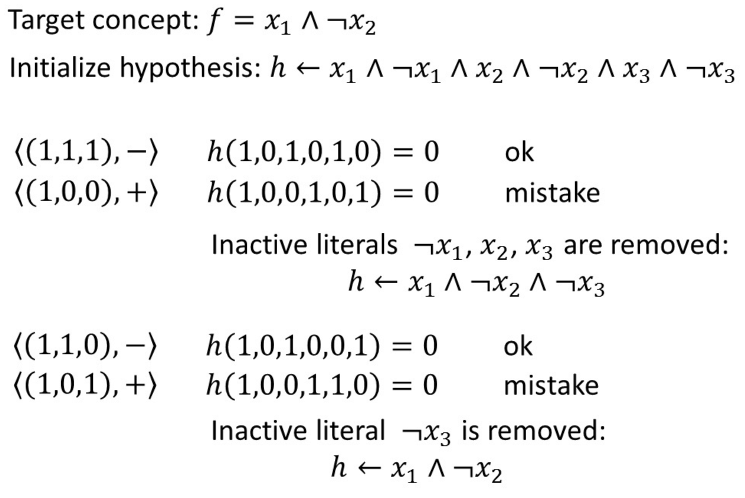
   Algorithm 1 shows the mistake bound algorithm for learning Boolean conjunctions.

Algorithm 1 Mistake bound algorithm for learnign Boolean conjunctions

```
1: procedure LearnBooleanConjunctions()
2:     Initialize the hypothesis: $h = x_1 \wedge \neg x_1 \wedge x_2 \wedge \neg x_2 \wedge \cdots \wedge x_n \wedge \neg x_n$
3:     for all examples $x \in X$ do
4:         if $h(x) \neq y$ then
5:             eliminate literals that are not active (0)        (See some examples in Figure 1)
```

An example of the learning process is shown in Figure 1. When the first mistake is made for an training data $x = (1, 0, 0)$, the literals $\neg x_1$, x_2, and x_3 are not active, and these literals are removed from the conjunctions, which results in the updated hypothesis $h = x_1 \wedge \neg x_2 \wedge \neg x_3$.

For every mistake, we remove at least one unnecessary literal from the conjunctions. Since we have $2n$ literals at the beginning, and at least one literal in the conjunctions will be remained in the end, the total number of mistakes is at most $2n - 1$, which is $O(n)$. Thus, this is a mistake bound algorithm.



Target concept: $f = x_1 \wedge \neg x_2$

Initialize hypothesis: $h \leftarrow x_1 \wedge \neg x_1 \wedge x_2 \wedge \neg x_2 \wedge x_3 \wedge \neg x_3$

$\langle(1,1,1), -\rangle$	$h(1,0,1,0,1,0) = 0$	ok
$\langle(1,0,0), +\rangle$	$h(1,0,0,1,0,1) = 0$	mistake

Inactive literals $\neg x_1, x_2, x_3$ are removed:

$h \leftarrow x_1 \wedge \neg x_2 \wedge \neg x_3$

$\langle(1,1,0), -\rangle$	$h(1,0,1,0,0,1) = 0$	ok
$\langle(1,0,1), +\rangle$	$h(1,0,0,1,1,0) = 0$	mistake

Inactive literal $\neg x_3$ is removed:

$h \leftarrow x_1 \wedge \neg x_2$

Figure 1: An example of a learning process

6. Given a linearly separable dataset consisting of 1000 positive examples and 1000 negative examples, we train two linear classifier using the perceptron algorithm. We provide the first classifier with a sorted dataset in which all the positive examples appear first, and then the negative examples appear. The second classifier is trained by randomly

selecting examples at each training iteration.

(1) Will both classifiers converge?

Yes. For the linearly separable dataset, it is proved that the perceptron algorithm will converge, and the mistake bound is R^2/γ^2, which is independent from the order of the examples.

(2) What will be the training error of each one of the classifiers?

When both classifiers converge, no more mistakes will be made. In other words, the training error will be zero for both classifiers.

7. We showed in class that using the Boolean kernel function $K(x, y) = 2^{same(x,y)}$ we can express all conjunctions over the attributes of x and y (with both the positive and negative literals). This representation can be too expressive. Suggest a kernel function $K(x, y)$ that can express all conjunctions of size at most k.

   Every active conjunctions of size at most k is a combination of positive or negative literals $l_i \in L$ that have the same value in x and y (i.e. $|L| = same(x, y)$). Then, counting the active conjunctions of size at most k is reduced to selecting a set of literals of size at most k from L. Thus,

$$K(x, y) = {}_{same(x,y)}C_0 + {}_{same(x,y)}C_1 + {}_{same(x,y)}C_2 + \cdots + {}_{same(x,y)}C_k$$

2 Programming Assignment

Regarding the continous attributes, I used thresholds as disscussed in class. Also, I ignored the examples which contain missing values during the learning. Finally, I used 0.01 as the learning rate for my implementation.

I ran my Perceptron algorithm with different $maxIterations$ from 1 to 7 in order to find the best $maxIterations$ which yields the best performance on the validation data. The learning curve in terms of the accuracy for $featrueSet = 1$ is shown in Figure 2. The best accuracy on the validation data was achieved when $maxIterations = 3$, so I chose $maxIterations = 3$ to evaluate the performance on the data (Table 1). It might be better to use F1-score to choose the best $maxIterations$, and I believe that it depends on the applications.

data	accuracy	precision	recall	F1
Training data	0.957	0.940	0.962	0.951
Validation data	0.969	0.931	1.0	0.964
Test data	0.529	0.516	0.696	0.593

Table 1: The performace result of $featureSet = 1$ when $maxIterations = 3$ is chosen.

In the similar manner, I experimented the learning curves for $featrueSet = 2$ and $featureSet = 3$, and the results are shown in Figures 3 and 4, respectively. The best accuracy on the validation data was achieved when $maxIterations = 3$ and $maxIterations = 5$,

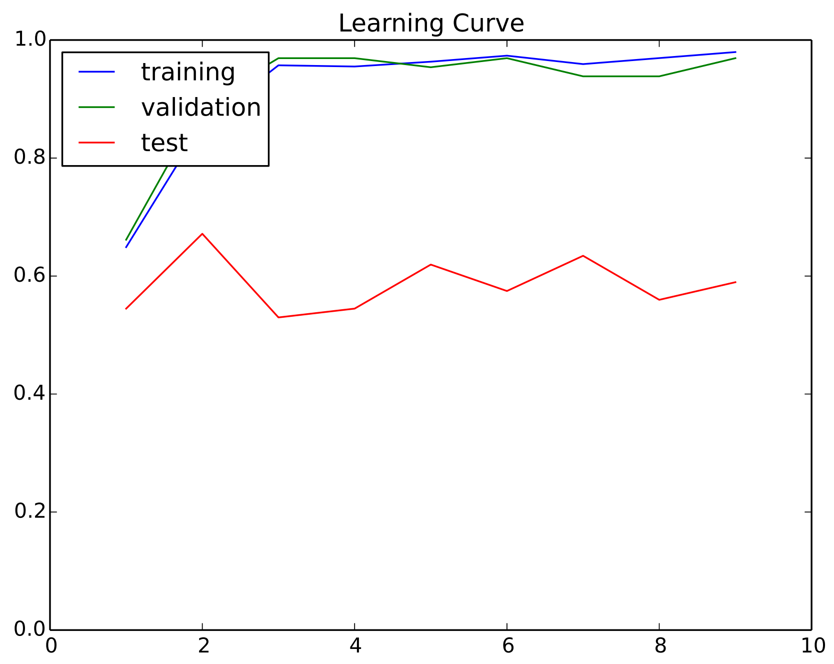


Figure 2: The performance evolution over the number of iterations for $featureSet = 1$

respectively, and I chose these values as $maxIterations$ to evaluate the performance (Tables 2 and 3).

data	accuracy	precision	recall	F1
Training data	0.970	0.953	0.980	0.966
Validation data	0.967	0.931	1.0	0.964
Test data	0.581	0.554	0.836	0.666

Table 2: The performace result of $featureSet = 2$ when $maxIterations = 3$ is chosen.

data	accuracy	precision	recall	F1
Training data	0.974	0.994	0.946	0.970
Validation data	0.967	0.962	0.962	0.962
Test data	0.606	0.580	0.770	0.661

Table 3: The performace result of $featureSet = 3$ when $maxIterations = 5$ is chosen.

The results show the significant improvement in the performance by using more complex feature set (i.e. $featureSet = 2$ or $featureSet = 3$). Notice that the performance on the test data is also improved by using complex feature set without overfitting. For instance, the accuracy on the test data improved from 0.529 to 0.581 ($featureSet = 2$) and 0.606 ($featureSet = 3$). This indicates that the higher dimensions of representation is more expressive and can reduce the error. However, the computation time significantly increases, and this approach may not be applicable when the original dimensionality is very high.

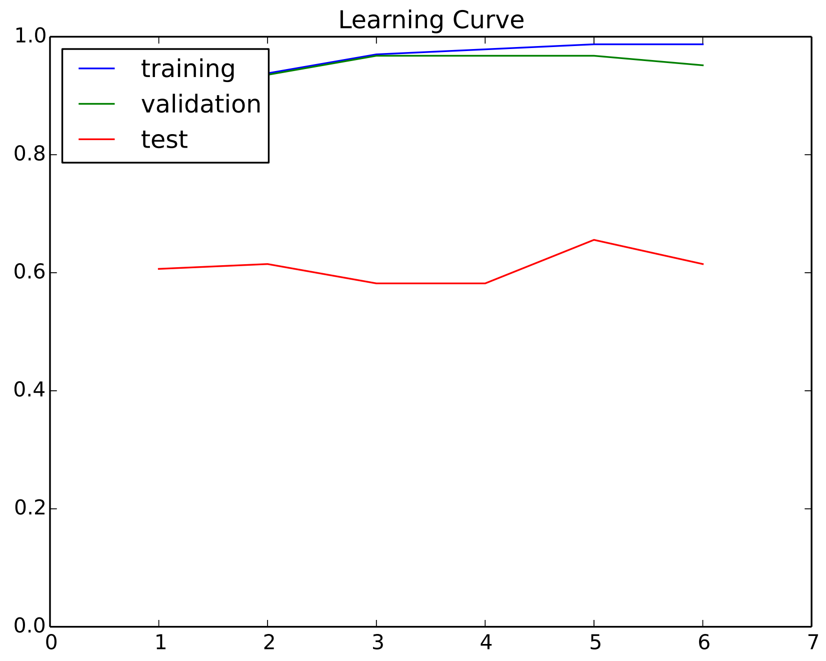


Figure 3: The performance evolution over the number of iterations for $featureSet = 2$

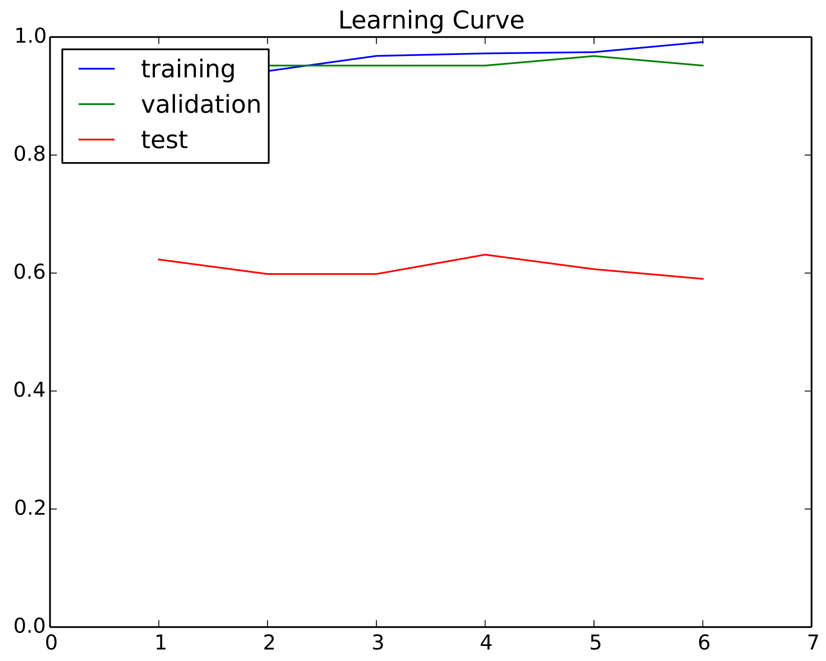


Figure 4: The performance evolution over the number of iterations for $featureSet = 3$