CS57800 Statistical Machine Learning

Homework 2

I-Ta Lee
Department of Computer Science
lee2226@purdue.edu

October 6, 2015

1 Foundations

1.1

(1) $(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4) \vee (x_3 \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_4) \vee (x_1 \wedge x_3 \wedge x_4) \vee (x_2 \wedge x_3 \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_3 \wedge x_4)$

(2) $x_1 + x_2 + x_3 + x_4 \geq 2$

1.2

The answer is 2^n, because each boolean variable can be either chose or ignored.

1.3

Suppose we have a vector β such that $y_i \beta^T x_i^* > 0, \forall i = 1, 2, \ldots, N$. This means β correctly classify all points. This doesn't change if we divide both side by $\|x_i^*\|$:

$$\begin{aligned} y_i \beta^T x_i^* &> 0 \\ y_i \beta^T (\frac{x_i^*}{\|x_i^*\|}) &> 0 \\ y_i \beta^T u_i^* &> 0 \end{aligned}$$

Let $m > 0$ be the minimum value of $y_i \beta^T u_i^*, \forall i$ in the training examples. So

$$y_i \beta^T u_i^* \geq m$$

If let $\beta^* = \frac{1}{m}\beta$, we have

$$y_i (\frac{1}{m}\beta)^T u_i^* = y_i (\beta^*)^T u_i^* \geq 1 \tag{1}$$

We keep this equation (1) in mind and move on our primary proof:

$$\begin{aligned}
\|\beta_n - \beta^*\|^2 &= \|\beta_n\|^2 - 2\beta_n\beta^* + \|\beta^*\|^2 \\
&= \|\beta_o + y_i u_i\|^2 - 2(\beta_o + y_i u_i)\beta^* + \|\beta^*\|^2 \\
&= \|\beta_o\|^2 + 2y_i\beta_o^T u_i + \|y_i u_i\|^2 - 2\beta_o^T\beta^* - 2y_i(\beta^*)^T u_i + \|\beta^*\|^2 \\
&= \|\beta_o - \beta^*\|^2 + \|y_i u_i\|^2 + 2y_i\beta_o^T u_i - 2y_i(\beta^*)^T u_i \\
&\leq \|\beta_o - \beta^*\|^2 + 1 + 2\cdot 0 - 2\cdot 1 \qquad (2) \\
&= \|\beta_o - \beta^*\|^2 - 1
\end{aligned}$$

The equation (2) holds because of the following reasons: first, $\|y_i u_i\|^2 = 1$; second, $y_i\beta_o^T u_i < 0$ because the misclassification of β_o; lastly, by equation (1), $y_i(\beta^*)^T u_i^* \geq 1$. ■

1.4

Here is our proposed algorithm:

Algorithm 1 Proposed Mistake Bound Algorithm for Learning Boolean Conjunction

Require: m training examples: $x_i, i = 1, 2, \ldots, m.$ each has n boolean features $f_1^i, f_2^i, \ldots, f_n^i$, and a corresponding label y_i

Ensure: Hypothesis h

1: // Initialize h to be the most specific hypothesis in the current hypothesis space
2: // which is conjunctions of the boolean features and their negations.
3: $h = f_1 \wedge \bar{f}_1 \wedge f_2 \wedge \bar{f}_2 \wedge \cdots \wedge f_n \wedge \bar{f}_n$
4: **for** each training example x_i **do**
5: **if** (y_i is possitive) AND (h makes mistake) **then**
6: remove from h any boolean item that is not satisfied by x_i
7: **end if**
8: **end for**
9: **return** h

The algorithm start from the most specific hypothesis h, which always generate negative results. For each positive training example, we would make a mistake. Then we eliminate from h the feature items that make the result become negative. In the end the hypothesis h satisfies all the training examples if the training data is noise-free. To assure the correctness, we need to make sure that our target concept c can be found in the hypothesis space H. Since our algorithm starts from the initial hypothesis h that is the most specific hypothesis and the h is generalized by iteratively eliminating undesired items (f_i or $\bar{f}_i$), $c \in H$.

Proof of Mistake Bound:

We then count the number of mistakes in the worst case of our algorithm. We observe that the first positive training example will eliminate half of the $2n$ items, because each f_i item will retain either itself or its negation to satisfy the boolean conjunction. After the first elimination, there will be n items left. Then each positive example on which our classifier makes mistake removes at least one item from the hypothesis h. Since the number of remaining items is n, the worst case of the number of mistakes will be n mistakes. To sum up, the mistake bound is $n + 1$ in our algorithm.

1.5

(1) Yes, they will converge ultimately but takes different times. Since they use the same dataset, if the dataset is linear separable, it will converge in the end. Even if the dataset is not linear separable, it will converge to a stable training error rate.

(2) The training error of both classifiers will firstly decrease, because of getting convergent, and then the error will increase when the overfitting occurs. However, the training error of the first classifier will decrease slower than that of the second classifier. If the training takes long, the second classifier will become overfitting earlier than the first classifier, which results in the training error arises.

The main reason of this phenomenon is that the fixed-order examples lead to less mistakes and thus the classifier learn less. For example, after making mistakes on the first 20 positive examples, the first classifier would always predict positive. When the learning process moves to the negative examples, the classifier would start predicting everything negative. After all, the classifier would only learn from few examples located at the beginning of positive/negative groups of data. On the other hand, randomly picked examples break this rule.

1.6

Let w_i be the weight updated by the $(i-1)^{th}$ example (will not be changed if no mistake) and $w_0 = 0$. We can do telescoping on (1), and then derive (2) from $w_0 = 0$.

$$
\begin{aligned}
\|\sum_{i\in N} y_i x_i\| &= \|\sum_{i\in N}(w_{i+1} - w_i)\| \\
&= \|(w_1 - w_0) + (w_2 - w_1) + (w_3 - w_2) + \cdots + (w_n - w_{n-1}) + (w_{n+1} - w_n)\| && (1) \\
&= \|w_{n+1}\| && (2) \\
&= \sqrt{(\|w_1\|^2 - \|w_0\|^2) + (\|w_2\|^2 - \|w_1\|^2) + \cdots + (\|w_{n+1}\|^2 - \|w_n\|^2)} \\
&= \sqrt{\sum_{i\in N}(\|w_{i+1}\|^2 - \|w_i\|^2)} \\
&= \sqrt{\sum_{i\in N}(\|w_i + y_i x_i\|^2 - \|w_i\|^2)} && (w_{i+1} = w_i + y_i x_i) \\
&= \sqrt{\sum_{i\in N}(\|w_i\|^2 + 2y_i w_i \cdot x_i + \|x_i\|^2 - \|w_i\|^2)} \\
&= \sqrt{\sum_{i\in N}(2y_i w_i \cdot x_i + \|x_i\|^2)} \\
&\leq \sqrt{\sum_{i\in N}\|x_i\|^2} && (y_i w_i \cdot x_i \leq 0)
\end{aligned}
$$

■

2 Programming Report

2.1 Implementation Notes

For the requirement of implementing the three functions (winnow(), perceptron(), and predict_one()), since I prefer to write Python in object oriented style, I put my implementation of these functions in the following places:

- *winnow()*: in the function of *Winnow.fit()* located in the file of *mylib/Learners.py*.
- *perceptron()*: in the function of *Perceptron.fit()* located in the file of *mylib/Learners.py*.
- *predict_one()*: in the functions of *Winnow.predict_one()* and *Perceptron.predict_one()* located in the file of *mylib/Learners.py*.

To run this program, in addition to the required command-line parameters (-a, -i, -f), we also needs to specify the data files. A simple example would be: "python hw2.py -a 1 -i 100 -f 1 data/train.csv data/validation.csv data/test.csv". Please see *ReadMe.txt* for more details.

2.2 Experiment Settings

In this project, I experimented with the Winnow, Perceptron, Averaged Perceptron, and Dual Perceptron algorithms. The hyperparameter being tuned by validating data is

- max iteration for the four learning algorithms

And the feature sets we support are unigram, bigram, and the combination of both.

The entire program can be divided into two phases: feature extraction and learning phases. During the feature extraction phase, we do some preprocessing on both unigram and bigram. I first removed the stop words recommended by XP06 [2014], but I did not use them all. I discard the stop words related to negation, like not and cannot, since I believe these words are important to sentiment analysis. Since removing stop words already eliminates the high-frequency terms, we do not filter out other high-frequency terms. For low-frequency terms, I filtered out the words with term frequency less than a parameter α. The resulting numbers of words with respect to α are listed in Table 1. Empirically, considering the tradeoff between learning time and the information lost, I select the thresholds $\alpha = 3$ for unigram and $\alpha = 2$ for bigram.

Table 1: Term Frequency Threshold α vs. Number of Features

α	1	2	3	4	5
#unigram	16159	7290	**4795**	3529	2793
#bigram	62479	**6253**	2693	1534	998

During the learning phase, we run the algorithm that is selected by users using command-line parameters (see ReadMe.txt). Since the learning algorithms are online learning algorithms, we can learn from data one epoch at a time and predict results in each epoch. The results output in the stdout. We can save these results by piping them to a file. Feeding this file to the program

"parse_and_plot_result.py", we will find the best validating accuracy among all epochs and see the corresponding testing accuracy. I have tried the hyperparameter max_iteration with range [1:1000], but there is no too much improvement after 100 iterations. Therefore, for clarity, in this report I only show the results with the max_iteration range [1:100]. We pick the best result among these epochs by using validating accuracy..

2.3 Results - Winnow

For Winnow algorithm, Figure 1, Figure 2, and Figure 3 show the training accuracy and validating accuracy v.s. number of epochs (iterations). I have two main observations here. First, we can see that in each feature set there is an epoch making the accuracy increased dramatically . Let's call this epoch "jumping point." I think this is a characteristic of Winnow, since Winnow takes several iterations (or number of mistakes) to make the classifier being able to predict positive label. Second, in bigram (Figure 2), the training and validating accuracy is relatively stable after the jumping point, while the accuracy for unigram+bigram is unstable. I think this might be because the bigram does not catch too much meaningful features to the learning algorithm.

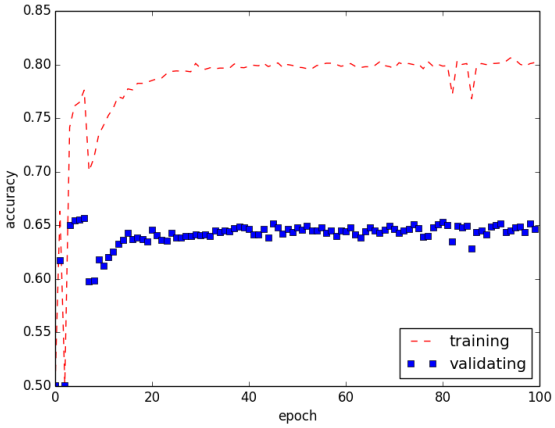

Figure 1: Winnow Algorithm with unigram

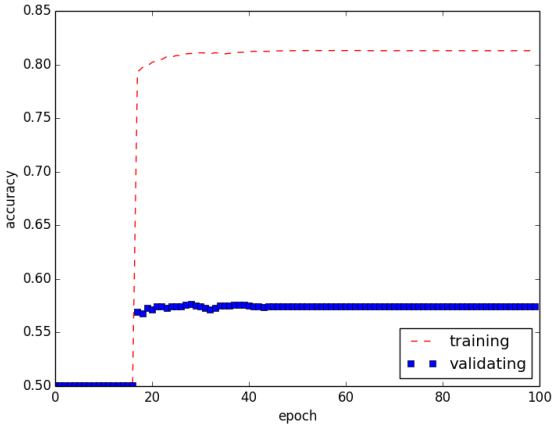

Figure 2: Winnow Algorithm with bigram

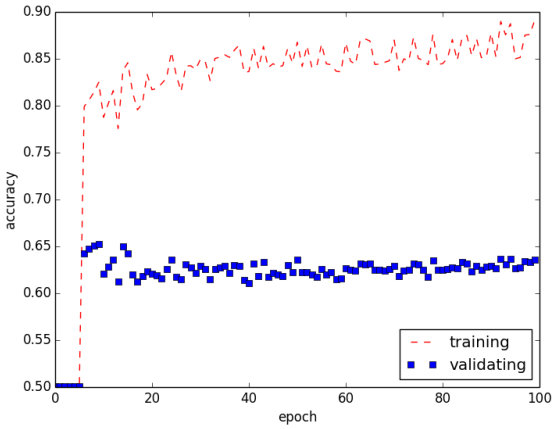

Figure 3: Winnow Algorithm with unigram+bigram

In Table 2, we show the final development result by using the validating data (as mentioned in the Experiment Settings, the range of max_iteration is from 1 to 100. This is same for the later experiments). The overall rank of accuracies of the features in Winnow is unigram $\simeq$ unigram+bigram $>$ bigram. The unigram feature gets the best accuracy and F-1 score, which is around 65%, while the bigram only gets 57% accuracy and 41% F-1 score. It seems that the bigram does not provide too much useful feature representations.

Table 2: Results of Winnow Algorithm

Winnow	unigram	bigram	unigram+bigram
Best #Iteration	6	28	9
Training Accuracy	0.776024	0.810253	0.824945
Validating Accuracy	0.65666	0.576454	0.652908
Testing Accuracy	**0.657598**	**0.577392**	**0.647749**
Training F-1	0.77896	0.788428	0.8302
Validating F-1	0.656338	0.423003	0.6512772
Testing F-1	**0.650383**	**0.407627**	**0.637373**
Training Precision	0.771463	0.894778	0.808624
Validating Precision	0.658192	0.665996	0.655598
Testing Precision	0.656039	0.662393	0.64833
Training Recall	0.786604	0.704673	0.85296
Validating Recall	0.654494	0.309925	0.647004
Testing Recall	0.644824	0.294397	0.626781

2.4 Results - Perceptron

Figure 4, Figure 5 and Figure 6 shows the training and validating accuracy trend in terms of numbers of epoch. We can see that Perceptron does not have the obvious "jumping point" like Winnow, which means it converges more gradually. It is more like a "real" learning algorithm that learns things gradually. Moreover, Perceptron can achieve really high training accuracy in all three feature sets. It is about 99%-100% in the end of 100 epochs. The validating accuracy however only has slight increase compared to the training accuracy. Other points are similar to Winnow's

For the best development result, which shows in Table 3, Perceptron has better results compared to Winnow. We can have about 72% accuracy and 71$ F-1 score in unigram+bigram by using Perceptron. However, it seems that Perceptron needs more number of epoch to get the best result. It takes 85/51/45 number of iterations to get the best result in unigram/bigram/unigram+bigram feature sets, while the Winnow only needs 6/28/9. This is an important observation that I did not expect before completion of this assignment. Moreover, the overall rank of accuracies of the features in Perceptron is unigram+bigram $>$ unigram $>$ bigram.

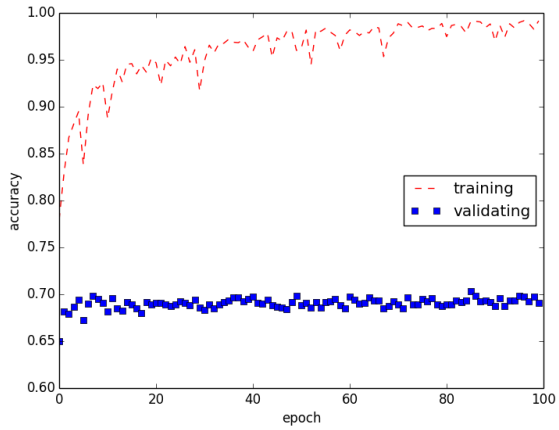

Figure 4: Perceptron Algorithm with unigram

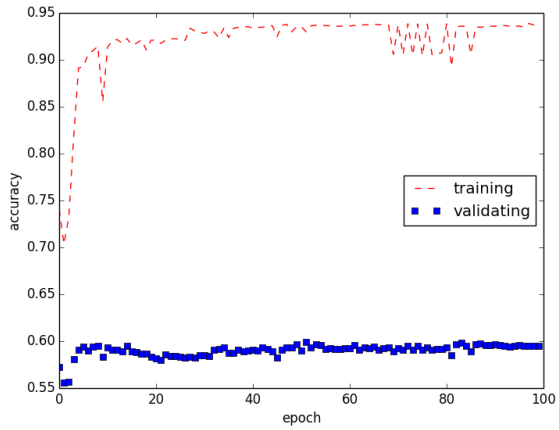

Figure 5: Perceptron Algorithm with bigram

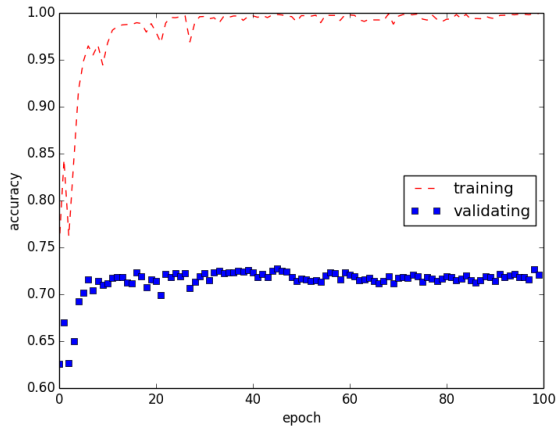

Figure 6: Perceptron Algorithm with unigram+bigram

Table 3: Results of Perceptron Algorithm

Perceptron	unigram	bigram	unigram+bigram
Best #Iteration	85	51	45
Training Accuracy	0.990935	0.929822	0.998281
Validating Accuracy	0.703096	0.598968	0.727486
Testing Accuracy	**0.695591**	**0.595685**	**0.720919**
Training F-1	0.990923	0.933766	0.998284
Validating F-1	0.70517	0.638783	0.723202
Testing F-1	**0.691686**	**0.633191**	**0.709898**
Training Precision	0.995597	0.886803	1.0
Validating Precision	0.701576	0.581986	0.736178
Testing Precision	0.692015	0.573631	0.729459
Training Recall	0.986293	0.985981	0.996573
Validating Recall	0.708801	0.707865	0.710674
Testing Recall	0.691358	0.706553	0.691358

2.5 Results - Averaged Perceptron

I also implemented the Averaged Perceptron to see its improvement compared to Perceptron. For simplicity, I only show the final development results of Averaged Perceptron. In Table 4 we can see two important things. First, the Averaged Perceptron gets a bit higher accuracy than Perceptron, which is about 73% accuracy in unigram+bigram feature set. Second, the Averaged Perceptron only needs 5/4/48 iterations to get the best result, which is far lower than the Perceptron. This means that the Averaged Perceptron not only gets the improvement in accuracy but also in the learning efficiency.

Table 4: Results of Averaged Perceptron Algorithm

AvgPerceptron	unigram	bigram	unigram+bigram
Best #Iteration	5	4	48
Training Accuracy	0.915286	0.90247	0.998593
Validating Accuracy	0.718574	0.606004	0.728424
Testing Accuracy	**0.719981**	**0.613508**	**0.725141**
Training F-1	0.915312	0.90051	0.998597
Validating F-1	0.72041	0.578736	0.731572
Testing F-1	**0.71985**	**0.585513**	**0.721747**
Training Precision	0.918182	0.922273	0.999065
Validating Precision	0.717069	0.62311	0.724518
Testing Precision	0.711503	0.62246	0.721747
Training Recall	0.912461	0.879751	0.998131
Validating Recall	0.723783	0.540262	0.738764
Testing Recall	0.728395	0.552707	0.721747

2.6 Results - Dual Form Perceptron

For the implementation of Dual Form Perceptron, we observed that it takes long time to compute the kernel, even I just use the linear kernel. The results are shown in Table 5, which are far worse than the original Perceptron. Only 63% accuracy and 70% F-1 score can be achieved in my implementation. This is not expected. I think there might be something wrong in my implementation, since I have surveyed the related topics about Kernel Perceptron, and It shows that we still need to collect support vectors in this algorithm to get better results. However, because this part was not taught in the class yet and I do not have enough time to finish it, I can only provide the results of my current version which does not use the support vectors and simply place the linear kernel into the Perceptron. I hope I can finish this part in the near future.

Table 5: Results of Dual Form Perceptron Algorithm

DualPerceptron	unigram	bigram	unigram+bigram
Training Accuracy	0.77962	0.741013	0.751016
Validating Accuracy	0.649625	0.572233	0.626173
Testing Accuracy	**0.65666**	**0.573171**	**0.632739**
Training F-1	0.771276	0.787755	0.793144
Validating F-1	0.634003	0.670043	0.700489
Testing F-1	**0.633267**	**0.668609**	**0.703072**
Training Precision	0.803988	0.668915	0.680027
Validating Precision	0.664954	0.545991	0.58506
Testing Precision	0.670201	0.542233	0.585227
Training Recall	0.741121	0.957944	0.951402
Validating Recall	0.605805	0.867041	0.872659
Testing Recall	0.60019	0.871795	0.880342

References

XP06. List of english stop words, 2014. URL `http://xpo6.com/list-of-english-stop-words/`.