CS57800 Statistical Machine Learning

Homework 3

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

November 10, 2015

1 Foundations

1.1

The VC dimension of C is is $2n+1$. We first show that $2n+1$ nodes on a circle can be shattered by C: we start a clockwise run to collect the consecutive same-label points until a point label is changed, and mark a point on the circle that is slightly larger than the last included point but smaller than the next different-label point. We do the same run again, and this time use a line to connect from the last mark to the current one. Repeat this procedure for a total of at most n lines, and we get our polygon by these lines. Figure 1 to Figure 3 illustrate this procedure for a case of $n = 3$.

We than proof that $2n + 2$ points cannot be shattered by C by contradiction: suppose there are $2n + 2$ points which can be shattered by C. We discuss two cases: first, if one node is in the convex hull formed by the other nodes, C can NOT shatter the points, because we cannot include exterior nodes while exclude interior nodes; second, suppose all nodes are vertices of a convex hull. To shatter these nodes, we need $n+1$ lines, since one line can at most increase two arcs on a circle. However, there are no n-polygon with $n + 1$ edges. This contradicts our initial assumption. ■

Figure 1: $2n + 1$ points can be shattered by C, when $n = 3$

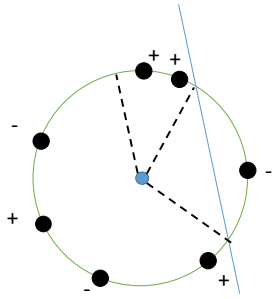


Figure 2: $2n+1$ points can be shattered by C, when $n=3$

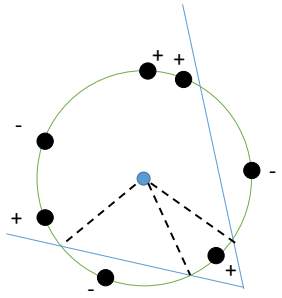
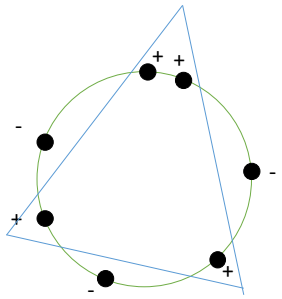


Figure 3: $2n+1$ points can be shattered by C, when $n=3$



1.2

$$
\begin{aligned}
l(y,\hat{y}) &= \frac{1}{\log 2}\log(1+e^{-y\hat{y}}) \\
&= \log_2(1+e^{-y\hat{y}}) \\
l'(y,\hat{y}) &= (\ln 2)^{-1}(1-e^{-y\hat{y}})^{-1}(e^{-y\hat{y}})(-y) \\
l''(y,\hat{y}) &= (-y)(\ln 2)^{-1}(e^{-y\hat{y}})(-y)(1-e^{-y\hat{y}})^{-1} + (-y)(\ln 2)^{-1}(e^{-y\hat{y}})(-1)(1-e^{-y\hat{y}})^{-2}(0-e^{-y\hat{y}}(-y)) \\
&= (y^2)(\ln 2)^{-1}(e^{-y\hat{y}})(1-e^{-y\hat{y}})^{-1} + (y^2)(\ln 2)^{-1}(e^{-2y\hat{y}})(1-e^{-y\hat{y}})^{-2} \\
&= \frac{y^2e^{-y\hat{y}} - y^2e^{-2y\hat{y}} + y^2e^{-2y\hat{y}}}{(\ln 2)(1-e^{-y\hat{y}})^2} \\
&= \frac{y^2e^{-y\hat{y}}}{(\ln 2)(1-e^{-y\hat{y}})^2} \geq 0
\end{aligned}
$$

Because the second derivative is no smaller than zero, the logistic loss function is convex. ■

1.3

From the class slide we know that $E_{train}(D) \leq \prod_t Z_t = \prod_t [2\sqrt{\epsilon_t(1-\epsilon_t)}] = \prod_t \sqrt{1-4\gamma_t^2} \leq e^{(-2T\gamma^2)}$. We want our training error, $E_{train}(D)$, to be zero after T iterations, and we know that

when only one training example is misclassified the training error is $\frac{1}{n}$. Therefore, $e^{(-2T\gamma^2)} < \frac{1}{n}$, i.e., $T > \frac{\ln n}{2\gamma^2}$.

1.4

First Iteration ($t = 1$): we choose $A = 5$, which minimizes the error rate. The weighted error rate $\epsilon_t = 0.2$, $\alpha_t = 0.693147$, $e^{-\alpha_t} = 0.5$, and $e^{\alpha_t} = 2.0$. Table 1 shows the result of this update:

Table 1: First Iteration

index	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
D_1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Pre-Normalized D_2	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.2	0.25
D_2 ($Z_1 = 0.8$)	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.25	0.25

current hypothesis: $f_1(x_1, x_2) = 0.693147 * I(x_1 > 5 \text{ AND } x_2 > 5)$, where $I(c)$ is the indicator function shown in Equation (1) below. The hypothesis still has 2 mistakes and the weighted error rate is 0.5.

$$I(c) = \begin{cases} 1 & \text{if } c \text{ is true} \\ -1 & \text{if } c \text{ is false} \end{cases} \tag{1}$$

Second Iteration ($t = 2$): we choose $A = 3$, which minimizes the error rate. The weighted error rate $\epsilon_t = 0.375$, $\alpha_t = 0.255413$, $e^{-\alpha_t} = 0.7746$, and $e^{\alpha_t} = 1.290995$. Table 2 shows the result of this update:

Table 2: Second Iteration

index	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
D_2	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.25	0.25
Pre-Normalized D_3	0.0484	0.0807	0.0484	0.0807	0.0484	0.04841	0.0484	0.0484	0.1937	0.3227
D_3 ($Z_2 = 0.9682$)	0.05	0.0833	0.05	0.0833	0.05	0.05	0.05	0.05	0.2	0.3333

current hypothesis: $f_2(x_1, x_2) = 0.693147 * I(x_1 > 5 \text{ AND } x_2 > 5) + 0.255413 * I(x_1 > 3 \text{ AND } x_2 > 3)$, where $I(.)$ is the indicator function. The hypothesis still has 2 mistakes and the weighted error rate is 0.533338.

1.5

Since K_1 and K_2 are valid kernels, $\int\int f(\vec{x})K_1(\vec{x},\vec{y})f(\vec{y})\mathrm{d}\vec{x}\mathrm{d}\vec{y} \geq 0$ and $\int\int f(\vec{x})K_2(\vec{x},\vec{y})f(\vec{y})\mathrm{d}\vec{x}\mathrm{d}\vec{y} \geq 0$. To show the positive-semi-definite of K:

$$\begin{aligned}
&\int\int f(\vec{x})K(\vec{x},\vec{y})f(\vec{y})\mathrm{d}\vec{x}\mathrm{d}\vec{y} \\
&= \int\int f(\vec{x})[c_1K_1(\vec{x},\vec{y}) + c_2K_2(\vec{x},\vec{y})]f(\vec{y})\mathrm{d}\vec{x}\mathrm{d}\vec{y} \\
&= c_1\int\int f(\vec{x})K_1(\vec{x},\vec{y})f(\vec{y})\mathrm{d}\vec{x}\mathrm{d}\vec{y} + c_2\int\int f(\vec{x})K_3(\vec{x},\vec{y})f(\vec{y})\mathrm{d}\vec{x}\mathrm{d}\vec{y} \\
&\geq 0 + 0
\end{aligned}$$

Therefore, K is also a valid kernel. ■

1.6

We can relax the constrain, $\sum_{t=1}^{M}\epsilon_i$ and $\epsilon_i \geq 0$, to be the hinge loss, $max(0, 1 - y_i w \cdot x_i + b)$. That is to say, when an error occurs on i, $\epsilon_i \geq 1$. Our worst-case training error is $\sum_{i=1}^{M} 1 = M$, when all M examples are misclassified. This is no larger than the $\sum_{t=1}^{M}\epsilon_i$, because $\epsilon_i \geq 1$ when an error occurs. Therefore, $\sum_{i=1}^{M} 1 = M \leq \sum_{t=1}^{M}\epsilon_i$.

2 Programming Report

This program aims to build a SVM classifier that supports L1 and L2 regularization. We have three hyperparameters needed to be tuned: (1) maxIterations, (2) stepSize, and (3) lmbd, and the feature sets we use are: (1) unigram, (2) bigram, (3) and both.

2.1 Feature Extraction

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 3: 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

2.2 Experiment Settings

We have three hyperparameters needed to be tuned: (1) maxIterations, (2) stepSize, and (3) lmbd. However, tuning the maxIterations is somehow similar to tuning the stepSize. The maxIterations determine the number of steps our learner takes and the stepSize determines the magnitudes of each step. Increasing the maxIteration while fixing the stepSize is similar to fixing the maxIteration while increasing the stepSize. To save time for finding the best hyperparameters, my idea is that we can first fix the maxIterations to be a number large enough, say 1000, and try to tune the stepSize and lmbd. When we get the best result that is achievable by a given range of lmbd, we then come back to find a better maxIterations and stepSize by fixing the lmbd. In addition, the SVM.LinearSvc classifier in *scikit-learn* scikit-learn developers [2014] has default maxIterations of 1000 in its setting, so I believe fastening the maxIterations to be 1000 is a decent start point for tuning other two parameters. Here are the sets of values of the hyperparameters we use in the validating phase:

- stepSize = [0.1, 0.3, 0.5, 0.7, 0.9]
- lmdb = [0.1, 0.3, 0.5, 0.01, 0.03, 0.001, 0.003, 0.0001, 0.0003, 0.00001, 0.00003]

For simplifying explanation, I do not put all detailed data here but in the appendix section. To see more detail about training/validating/testing results, please refer the appendix section.

2.3 Experiment 1: Tuning lmbd and stepSize by fastening maxIteration = 1000

Here comes with six figures (Figure 4-9), which shows the validation results of the combination of L1/L2 regulation and Unigram/Bigram/Both features. From these six figures, we can observe that for L1 regulation, the validation accuracy is relatively unstable, in terms of lmdb. I think this is because that L1 regulation forms a more sparse (low-complexity) hypothesis that underfits the data, compared to L2 regulation. In L2 regulation, the validation accuracy is relatively stable. The three L2-regulation results are in the same trend that begin with a very low accuracy, followed by a big jump to a very high accuracy, and then gradually decrease. For another observation, bigram seems not providing too much useful information, compared to unigram. It can reach only around 61% accuracy while the unigram can reach around 74%.

Table 4 shows the complete results of the experiment 1. We can see that our hypothesis is almost consistent with the training data (For L1 regulation, both feature, we get 97% of training accuracy). I think that this might result in overfit of the training data; however, the testing accuracy is still pretty good in this setting (For L2 regulation, unigram feature, we get 74% testing accuracy). Moreover, the overall results of these six settings are quite similar, regardless the bigram does not provide useful information.

Figure 4: Validation Accuracy with L1 Regulation and Unigram Feature Settings. The best validation accuracy is 0.7397 occurring at lmdb=0.0001 and stepSize=0.1

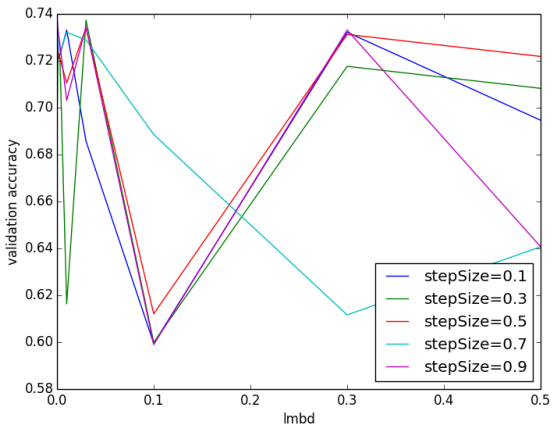


Figure 5: Validation Accuracy with L1 Regulation and Bigram Feature Settings. The best validation accuracy is 0.6130 occurring at lmdb=0.03 and stepSize=0.7

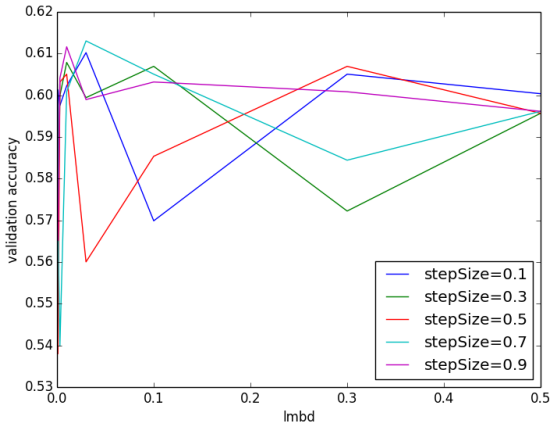


Figure 6: Validation Accuracy with L1 Regulation and Unigram+Bigram Feature Settings. The best validation accuracy is 0.7542 occurring at lmdb=0.00003 and stepSize=0.9

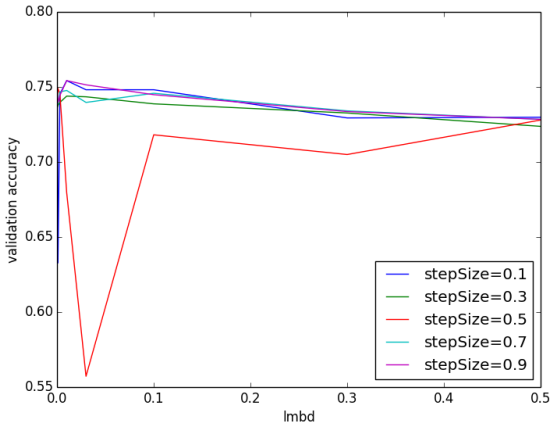


Figure 7: Validation Accuracy with L2 Regulation and Unigram Feature Settings. The best validation accuracy is 0.7387 occurring at lmdb=0.0003 and stepSize=0.9

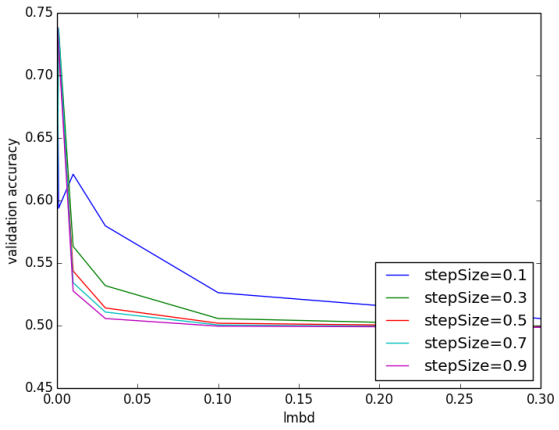


Figure 8: Validation Accuracy with L2 Regulation and Bigram Feature Settings. The best validation accuracy is 0.6135 occurring at lmdb=0.00001 and stepSize=0.5

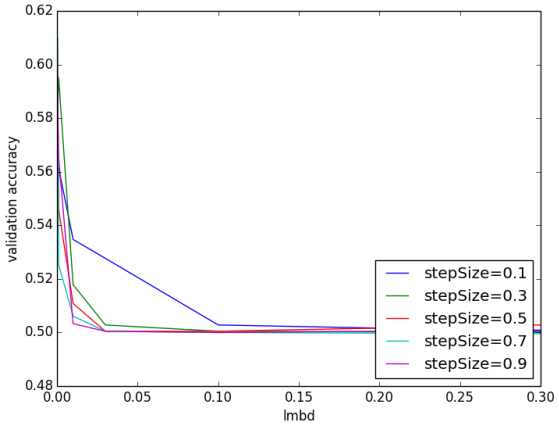


Figure 9: Validation Accuracy with L2 Regulation and Unigram+Bigram Feature Settings. The best validation accuracy is 0.7538 occurring at lmdb=0.00001 and stepSize=0.1

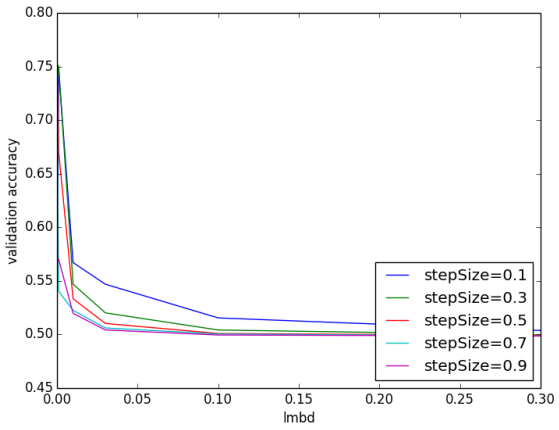


Table 4: Complete best results for the Experiment 1by using the fine-tuned hyperparameters. (L1/L2 stands for L1/L2 regulation. F1/F2/F3 stands for Unigram/Bigram/Both feature)

Parameters	**L1,F1**	**L1,F2**	**L1,F3**	**L2,F1**	**L2,F2**	**L2,F3**
Training Accuracy	0.9095	0.9018	0.9678	0.9072	0.9023	0.9703
Training Precision	0.9519	0.9722	0.9694	0.9331	0.9726	0.9681
Training Recall	0.8632	0.8280	0.9664	0.8779	0.8287	0.9729
Training F-1	0.9054	0.8943	0.9679	0.9047	0.8949	0.9705
Validating Accuracy	0.7397	0.6130	0.7547	0.7387	0.6135	0.7538
Validating Precision	0.7773	0.6901	0.7525	0.7594	0.6918	0.7498
Validating Recall	0.6732	0.4129	0.7603	0.7004	0.4120	0.7631
Validating F-1	0.7215	0.5167	0.7564	0.7287	0.5164	0.7564
Testing Accuracy	0.7364	0.6098	0.7317	0.7378	0.6093	0.7289
Testing Precision	0.7626	0.6627	0.7200	0.7480	0.6647	0.7153
Testing Recall	0.6771	0.4274	0.7474	0.7075	0.4217	0.7493
Testing F-1	0.7173	0.5196	0.7335	0.7272	0.5160	0.7319

2.4 Experiment 2: Find overall best results by tuning maxIterations

We conduct the Experiment 2 by using the fine-tuned stepSize and lmbd, and try to find the best maxIterations. I experimented with maxIterations [100, 500, 1000, 1500, 2000, 2500]. From Figure 10, we can see that the overall best result occurs when using L1 regulation and both features at maxIterations=1000. This matches our initial assumption that 1000 is a quality start point for tuning other hyperparameters. The overall best result and its hyperparameters are shown in Table 5.

Figure 10: MaxIterations vs Validation Accuracy

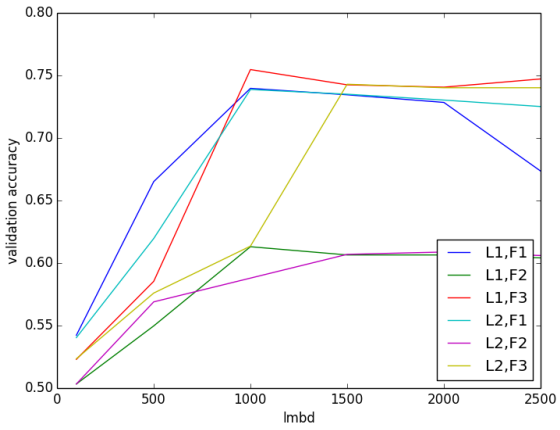


Table 5: Overall Best Result

Parameters	**1000 l1 0.9 3e-05 3**
Training Accuracy	0.9678
Training Precision	0.9694
Training Recall	0.9664
Training F-1	0.9679
Validating Accuracy	0.7547
Validating Precision	0.7525
Validating Recall	0.7603
Validating F-1	0.7564
Testing Accuracy	0.7317
Testing Precision	0.7200
Testing Recall	0.7474
Testing F-1	0.7173

2.5 Program Efficiency

The time needed for each part of this program is listed below.

Table 6: Time required for each part of this program

Part of Program	**Average Time Needed**
Unigram Feature Extraction	98 s
Bigram Feature Extraction	160 s
Load Unigram/Bigram from Feature File	1 s
10-Iteration Training	0.81 s
100-Iteration Training	7.15 s
1000-Iteration Training	60 s

We can observe that our training algorithm is very efficient compared to the feature extraction

part. Plus, the features do not need to be re-calculated for each run. Therefore, I implemented a mechanism to dump the produced features into a feature file (which is a pickle file), and there is an argument to load the features from that file. By doing so, our time for loading the features only require about 1 second. This largely saves my time for the experiments. To know how to configure dumping/loading such file, please see the ReadMe file.

2.6 Conclusions

In this project, I learned how to setup a machine learning experiment to fine-tune the hyper-parameters for different kinds of models. And I know that different hyperparameters will largely affect the training results, especially for the hyperparameter like lmdb, which locates in the cost function. For some possible improvements of this work, I think I can do the crossvalidation like KFold to make sure we get the general hyperparameters. However, because the validation data is provided in this project, I didn't do the KFold. Moreover, our current results show that my method works well and can be applied to other machine learning problems. That is the most import lesson I learned from this project.

References

scikit-learn developers. Scikit-learn svm.linearsvc, 2014. URL `http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`.

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