华南理工大学

South China University of Technology

The Experiment Report of *Machine Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Lizhao Liu

Supervisor:
Mingkui Tan

Student ID:
201730683109

Grade:
Undergraduate

November 12, 2019

Adaboost for face image classification

***Abstract*—In this report, we solve the face/non-face classification problem by adaboost with Decision Tree Classifier (DTC) as weak learner.**
We perform experiments on three aspects:
1. Weak learner's performance effect.
2. Number of weak learner's exploration.

I. Introduction

ADABOOST short for "Adaptive Boosting", is an ensemble method that combine a bunch of weak learners into a strong learner which outperforms all the weak learners. It's wildly used in academic and industry community, which make it very important for us to fully explore it. We will explore the the effect of weak learner's performance and the performance with different number of weak learner.

II. Methods and Theory

In this part, we define Adaboost algorithm.

A. Adaboost

Given dataset $D = \{x_i, y_i\}_{i=1}^m$, where $x_i \in \mathbf{R}^n$. $y_i \in \{-1, 1\}$ and weak Learner $L(x) \in \{-1, 1\}$, Adaboost algorithm can be found in Algorithm 1.

Algorithm 1: Adaboost

Input: Training Dataset $D = \{x_i, y_i\}_{i=1}^m$, num weak learner N, weak learner L
Output: Final learner $H(x)$

$w_1(i) \leftarrow \frac{1}{m}$ where i = 1, 2, ..., m
for $idx = 1, 2, ..., N$ **do**
  $h_{idx}(x) = L(D, w_{idx})$
  $\epsilon_{idx} = \Sigma_{i=1}^m w_{idx}(i)\mathbb{1}(h_{idx}(x_i) \neq y_i)$
  $\alpha_{idx} = \frac{1}{2}\log\frac{1-\epsilon_{idx}}{\epsilon_{idx}}$
  $w_{idx+1}(i) = w_{idx}(i) * exp(-1 * \alpha_{idx} * y_i * h_{idx}(x_i))$ where i = 1, 2, .., m
  $w_{idx+1} = \frac{w_{idx+1}}{\Sigma_{i=1}^m w_{idx}(i)}$
return $H(x) = \Sigma_{idx=1}^N \alpha_{idx} h_{idx}(x)$

We can see that, in each iteration, the examples that correctly classified by previous weak learner, will get a smaller weight, so that current learner can pay more attention to the misclassified examples.

III. Experiments

A. Dataset

We conduct all the experiments on self constructed face/non-face dataset, which has 500 256x256 face images and 500 32x32 non-face images. All images have 3 channels. We preprocess all the images into 24x24 gray scale images. We then convert them by using NPD features. Finally, each example's form is (x, y), where $x \in \mathbf{R}^{165600}$ and $y \in \{-1, +1\}$, where $+1$ means face and -1 means non-face. We randomly split the dataset into three part: 700 training examples, 100 validation examples and 200 test examples. $X_{train},\ X_{val}\ and\ X_{test}$ are divided by 256 for scale purporse.

B. Implementation

We implement the Adaboost using python and mainly rely on the numpy package.

C. Weak learner's performance's effect.

In this section, we explore the influence of weak learner's performance in Adaboost. We set number of weak learner to five weak learner due to computation limitation.

In this section, we explore the influence of C in SVM. We set log_{10}^C in a coarse-grained range e.g. -2, -1, 0, 1, 2, 3, 4, and compare $loss_{val}$, $accuracy_{val}$ and l_2norm of w, where $loss = \frac{w^T w}{2} + \frac{C}{m}\Sigma_{i=1}^m \max(0, 1 - y_i(w^T x_i + b))$ and $l_2norm(w) = \sqrt{\Sigma_{i=1}^n w_i}$. We zero initilize w and b, use SGD as optimizer with learning rate 0.001, set epoch to 200 and batch size to 64.

From Fig 1, we can see that, when C is very small, e.g. smaller than 1, the gradient descent process is very unstable and the loss is higher than other. As C goes up, the loss decreases, e.g. from 1 to 10. But when C gets larger than 10, the loss become slightly bigger, but still better than small C, like 0.01 and 0.1. So, we can draw the conclusion that, bigger C usually better than small C, and the best C is in the middle, which is not very big but also not very small. As we can see from Fig 1, $C = 10$ gets the lowest loss.

From Fig 2, we can see that, when C is very small, e.g. 0.01, the accracy in both training set and validation set is much lower than others. Small C like 0.01 and 0.1 get worse performance comparing with others. In training set, the accracy is very similar when C bigger than 1. In validation set, $C = 10$ gets slightly better performance comparing with others. Best validation performance under each C setting can be found in Table I.

From Fig 3, we can see that, as C goes up, $l_2norm(w)$ goes up except $C = 100$. As said above, $\frac{1}{2C}$ can be viewed as regurilizer, so when C is bigger, the smaller of its impact on regurization, so the weight maginitude goes up.

TABLE I
Best Result in Validation set under different C's settings. Best result are in bold.

log_{10}^C	-2	-1	0	1	2	3	3
$Accuracy_{val}$	84.39	84.42	84.72	**84.75**	84.73	84.56	84.49
$Epoch$	**2**	86	62	138	173	147	145

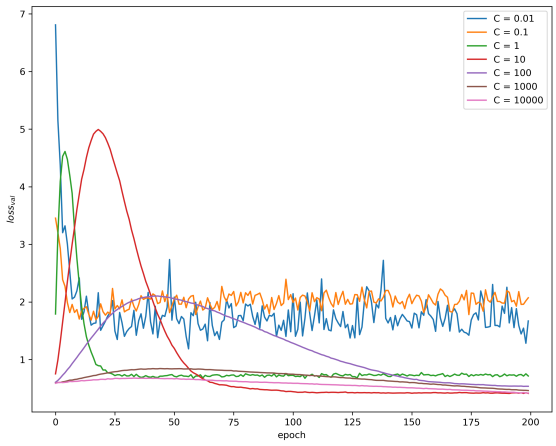

Fig. 1. $Loss_{val}$ under different C.

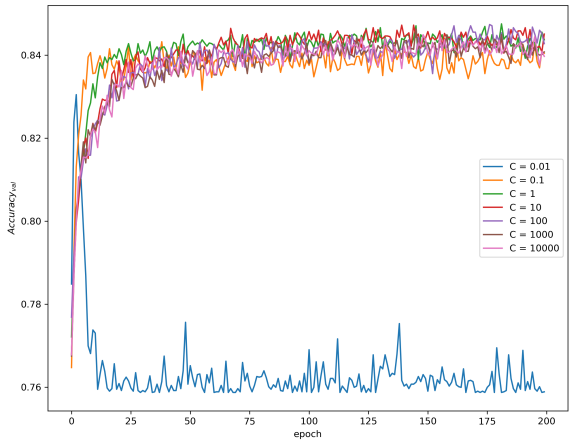

Fig. 2. $Accuracy_{val}$ under different C.

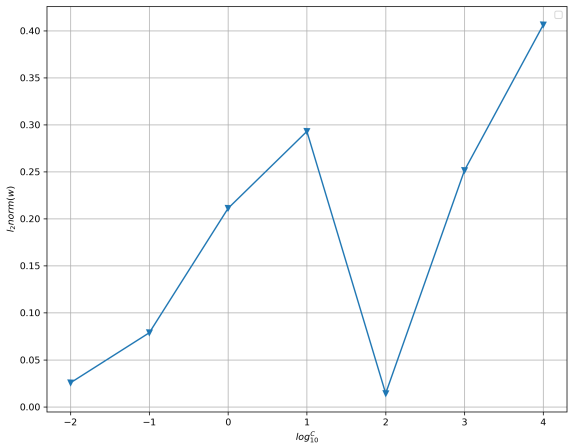

Fig. 3. The magnitude of W under different C.

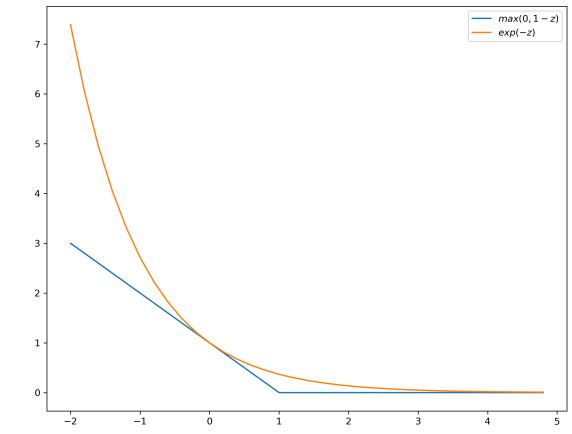

Fig. 4. The plot of HL and EL.

D. Loss function comparision in SVM

In this section, we conduct experiment on loss function comparision in SVM. Especially, we compare two loss function, hinge loss which is illustrated above And Exponential Loss (EL), where $EL = exp\{-y_i(w^T x_i + b)\}$. More formaly, $HL(z) = max(0, 1 - z)$ and $EL(z) = exp^{-z}$. Fig 4 shows that HL and EL both have one advantage and one disadvantage. HL is stable when z is very mall, however EL become very large, which will lead to numeric unstable. EL is derivable every but HL is not derivable when $z = 1$. We use SGD as optimizer with learning rate 0.001, set epoch to 200, batch size to 64 and C to 10.

From Fig 5 and Fig 6, we can see that the performance of HL is superior. EL has numeric unstablity problem during gradient descent process.

E. initializer comparision in both LR and SVM

In this section, we compare the performance of LR and SVM under different initializers. We explore over ones, zeros, xavier uniform and random initializer. Ones initializers makes the initial w to all one and Zeros initializers makes the initial w to all zero. Xavier uniform initializer uniformally initial w, but bounded to interval $[-n, n]$, where n is W's dimension. Random initializer initial w from standard normal distribution. We use SGD as optimizer with learning rate 0.001, set epoch to 200, batch size to 64 and C to 10 in SVM. We use HL as SVM's loss function and BCE as LR's loss function.

From Fig 7 and Fig 8, we can see that LR's performance is affected by the initializer. Typically Ones initializer brings the worst performance and Xavier uniform initializer bring the best performance.

From Fig 9 and Fig 10, we can see that SVM is very robust to the initializer, different initializer doesn't disturb the

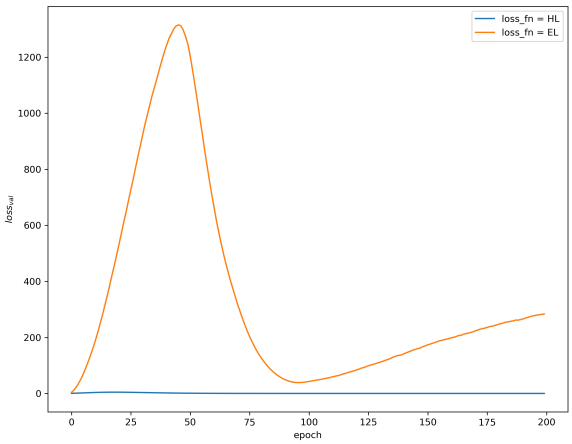

Fig. 5. $loss_{val}$ under different loss function.

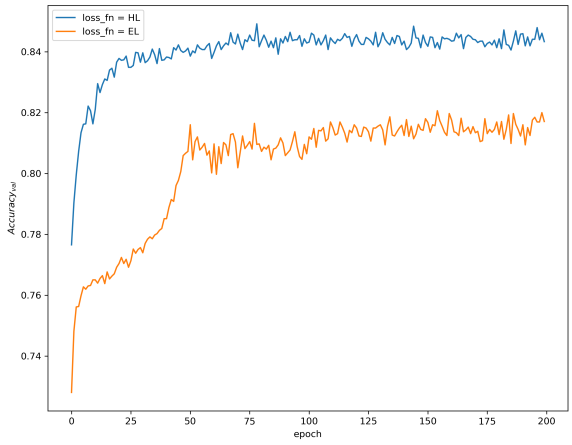

Fig. 6. $Accuracy_{val}$ under different loss function.

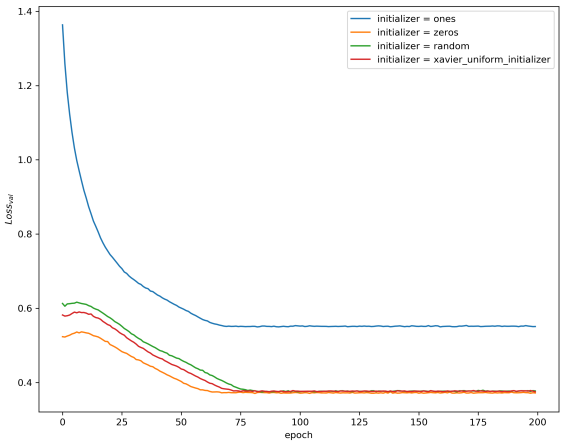

Fig. 7. LR's BCE_{val} under different initializer.

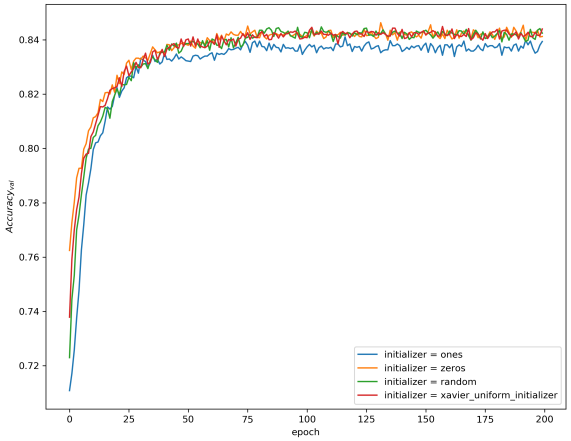

Fig. 8. LR's $Accuracy_{val}$ under different initializer.

performance.

F. Optimizer comparision in both LR and SVM

In this section, we compare the performance of LR and SVM with different optimizers. We explore over Stochastic Gradient Descent (SGD), Momentum, Adam and RMSProp. We use xavier uniform initializer, set learning rate to 0.001 epoch to 200, batch size to 64 and C to 10 in SVM. We use HL as SVM's loss function and BCE as LR's loss function.

From Fig 11 and Fig 12, we can see that LR's performance is affected by the optimizer. Typically SGD converges very slowly; Momentum converges faster than SGD, but not as good as SGD's performance; both Adam and RMSProp converge very fast and get to the best performance.

From Fig 13 and Fig 14, we can see that SVM is very robust to the optimizer too, different optimizer didn't disturb the performance except Momentum which converges slower and get worse performance comparing with other optimizers.

G. Performance on validation set and test set.

Now we report the final performance of LR and SVM on test set. For LR, we set optimizer to Adam with learning rate 0.001, batch size to 64, initializer to Xavir uniform intitializer, epoch to 200. For SVM, we set optimizer to Adam with learning rate 0.001, batch size to 64, initializer to Xavir uniform initializer, C to 10, epoch to 200.

Although LR is very sensitive to initializer and optimizer, but using the best initializer Xavir uniform initializer and optimizer Adam, it outperform SVM by a little under the same experiment settings and enjoys faster convergence.

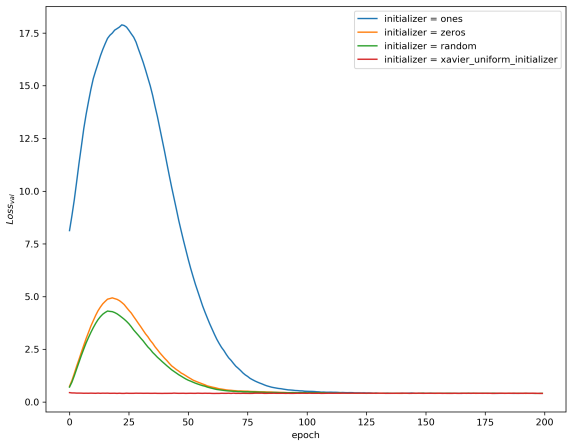

Fig. 9. SVM's HL_{val} under different initializer.

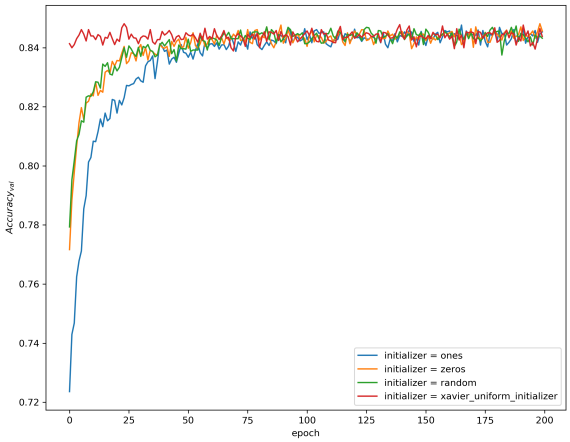

Fig. 10. SVM's $Accuracy_{val}$ under different initializer.

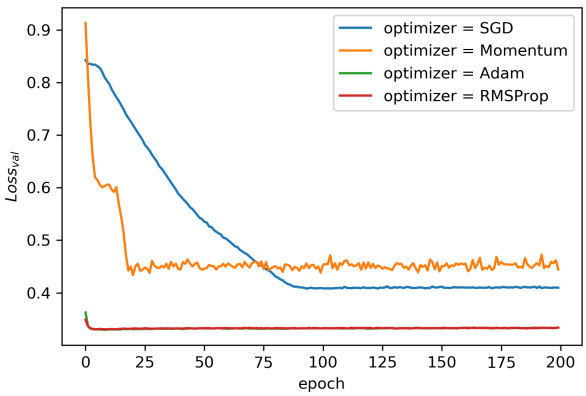

Fig. 11. LR's BCE_{val} under different optimizer.

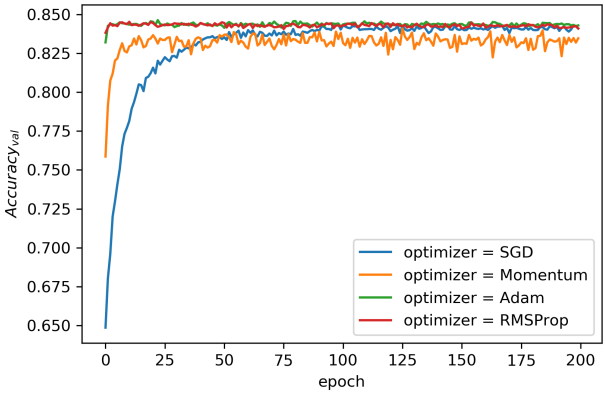

Fig. 12. LR's $Accuracy_{val}$ under different optimizer.

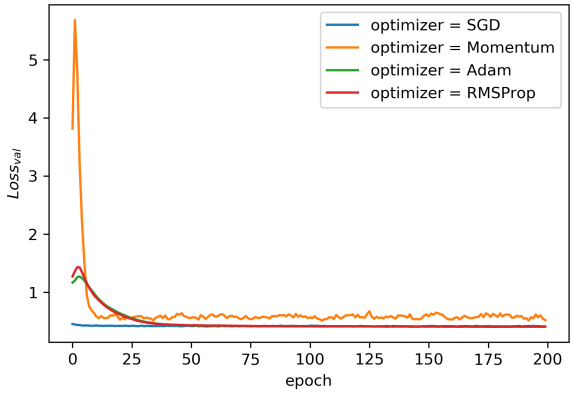

Fig. 13. SVM's HL_{val} under different optimizer.

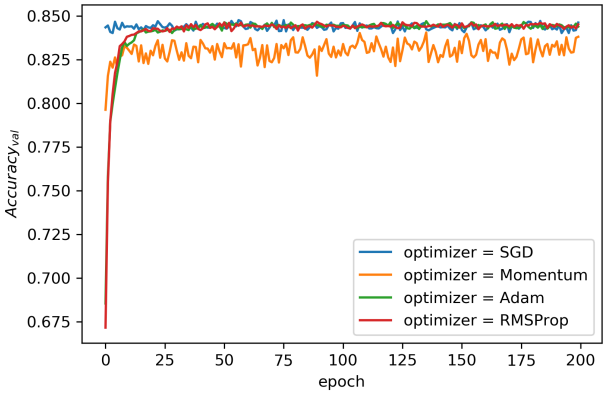

Fig. 14. SVM's $Accuracy_{val}$ under different optimizer.

IV. Conclusion

In this report, we explore SVM and logistic regression. Specifically, we explore the hyper-parameter C and loss function comparision in SVM, we draw the conclusion that best C is something in the middle (not too small and too big) and

TABLE II
RESULT ON A9A VALIDATION SET AND TEST SET. BEST RESULT ARE IN BOLD.

	LR	SVM
$Accuracy_{val}$	**84.66**	84.44
$Accuracy_{test}$	**84.98**	84.88
$Epoch$	**37**	107

hinge loss is better than exponential loss. Besides that, we also compare different initializers and optimizers in logistic regression. We oberseve that SVM is more robust than logistic regression under differen initializer and optimizer settings; Adam and RMSProp both obtain very good performance in logistic regression and SVM. Finally, we report te performance of LR and SVM under each best settings and get their performance on test set, suprisingly LR outperform SVM in a little performance.