Controlling $\frac{FP}{FN}$ with an adaptive loss function

George Lee

November 17, 2024

1 Overview of imbalanced binary classification

1.1 Introducton

In this work we study binary classification. This is an important problem in its own right, but also a useful "toy" problem in learning where associated methods may be useful in very general situations. Given an observation in some state space $\mathcal{X}$, the problem is to decide whether x belongs to one of two populations, which we may denote by $+$ and $-$ or by 1 and 0. unfortunately 0 isn't negative, but the latter convention is useful when writing mathematical expressions. One important situation which we are interested in handling is that of **imbalanced classification**, in which most of the observations belong to $-$ and a minority to $+$. Simplifying further, we investigate the supervised learning of such a model: the data consists of the training set

$$(x, y) \in \mathcal{X}^m \times \{\pm\}^m$$

and testing set

$$(a, b) \in \mathcal{X}^n \times \{\pm\}^n.$$

The bulk of popular models use neural networks and we largely focus on this approach here. One working conjecture is that almost regardless of choice of loss function in such a situation, the gradient updates are likely to fall in a narrow range of possibilities for a fixed batch of training data. We identify a surface on which we can expect such an update to lie, and propose an algorithm that selects a direction on this surface that forces a particular ratio of false positive and false negatives.

1.2 A reminder of some performance metrics

When evaluating the model's performance, each binary prediction and label $(\widetilde{v}_i, v_i)$ is either a true positive $(+,+)$, a false positive $(+,-)$, a false negative $(-,+)$ or a true negative $(-,-)$.

When a batch of these observations are made, denote the counts of these corresponding cases by TP, FP, FN and TN respectively. Standard associated statistics are

- the **sample(/batch) precision** $= \frac{TP}{TP+FP}$,
- the **sample recall** $= \frac{TP}{TP+FN}$ and
- the **sample accuracy** $= \frac{TP+TN}{TP+FP+TN+FN}$.

These quantities are easily understood by someone working on a wide range of possible problems. In the context of spam filtering, when differentiating between ham and spam, we don't necessarily mind if we sometimes classify ham as spam, but we really should try to insist that spam never makes it through.

You could achieve this by blocking everything, so we can't merely focus on preventing false negatives. We should try and insist instead on some combination of accuracy and minimising false negatives.

One may also worry about this sort of problem when testing for rare diseases. Not running any test at all may be highly accurate and possibly preferable to a test with too many false positives.

If we assume that the data is drawn from fixed distributions, then these sample statistics approximate true rates associated to some model with parameter $\boldsymbol{\theta}$. Write $\mathbb{P}_{\boldsymbol{\theta}}$ for the probabilities of events for such a fixed model.

- The **precision** $= \mathbb{P}_{\boldsymbol{\theta}}(+|\text{We predicted } +)$, and
- the **recall** $= \mathbb{P}_{\boldsymbol{\theta}}(\text{we predicted } +\,|+)$.

1.3 Passing to a continuous-valued classifier

For any $\lambda \in [0,1]$ the quantity $(\lambda FP + (1-\lambda)FN$ is analogous to a least squares loss where the cost for a false negative versus a false positive is controlled by λ. However, this function takes discrete values so doesn't have a useful gradient and it is impossible to apply backpropogation directly.

The ubiquitous solution here is to work with a model $f_\theta : \mathcal{X} \to (0,1)$ that outputs a number that may be interpreted as a likelihood of an observation being in the + class. With this approach, the practical use of such an f_θ is to fix a threshold $\alpha \in (0,1)$ such that the actual classification is given by + if $f_\theta(x) > \alpha$ and − otherwise. Correspondingly, a trained neural network provides a family of different possible models indexed by α which is monotonic in the proportion of data that is classified as −. As we approach the extreme cases $\alpha \to 0$ and $\alpha \to 1$ the model tends towards interpreting all points as + and − respectively. Commonly such models are then evaluated by their performance over a range of these alpha variables, resulting in metrics that give an overview of the performance of the family of generated models, and the guarantee of monotonicity as a threshold is fixed allows for a particular choice of $FP : FN$ that suits the training dataset.. In the case of imbalanced data the precision-recall curve or PRC is pertinent for imbalanced classification: one traces out the function $(\mathbf{Precision}(\alpha), \mathbf{Recall}(\alpha))$ and computes the area under the curve in the unit square. Similar metrics in the literature include the "AUC" and "ROC". We will be able to generate an analogous metric when varying a threshold in out loss function, so can produce comparable performances of the families of models that we can generate. One should however note that for classification on big datasets where we are only interested in $\frac{FP}{FN}$ very big or small, such an area isn't the relevant statistic.

2 Controlling $\frac{FP}{FN}$ with parameterised loss functions

In this work we are focused on influencing the behaviour of the model at the training stage - we will largely fix our threshold at $\alpha = \frac{1}{2}$ and make the loss function do the work of forcing a desired $\frac{FP}{FN}$ tradeoff.

We consider families of loss functions depending upon a parameter that constrains the model at the training stage. This approach ensures that only the problem of interest is being solved, at the cost of needing to retrain a model if constraints are changed. For the sake of comparison we may generate similar AUC metrics but note that these methods are only useful if we care about all possible $\frac{FP}{FN}$ ratios, which is unrealistic in highly imbalanced classification.

2.1 Smoothed statistics

Fix a batch of observations $\mathcal{F} = \{(x_i, y_i)\}_{i=1}^T \subseteq X \times \{0,1\}$ and write $\hat{y}_i = f_{\boldsymbol{\theta}}(x_i)$ for $1 \leq i \leq T$. We can approximate the counts of the four different outcomes algebraically by

- $\mathtt{tp}_{\mathcal{F}}(\boldsymbol{\theta}) = \sum_{i=1}^T y_i \hat{y}_i$,
- $\mathtt{tn}_{\mathcal{F}}(\boldsymbol{\theta}) = \sum_{i=1}^T (1-y_i)(1-\hat{y}_i)$,
- $\mathtt{fp}_{\mathcal{F}}(\boldsymbol{\theta}) = \sum_{i=1}^T (1-y_i)\hat{y}_i$ and
- $\mathtt{fn}_{\mathcal{F}}(\boldsymbol{\theta}) = \sum_{i=1}^T y_i(1-\hat{y}_i)$.

Then the estimates for precision and recall are given by

- The **surrogate precision** $= \frac{\mathtt{tp}_{\mathcal{F}}(\boldsymbol{\theta})}{\mathtt{tp}_{\mathcal{F}}(\boldsymbol{\theta}) + \mathtt{fp}_{\mathcal{F}}(\boldsymbol{\theta})}$ and
- the **surrogate recall** $= \frac{\mathtt{tp}_{\mathcal{F}}(\boldsymbol{\theta})}{\mathtt{tp}_{\mathcal{F}}(\boldsymbol{\theta}) + \mathtt{fn}_{\mathcal{F}}(\boldsymbol{\theta})}$.

Where clear we may suppress $\boldsymbol{\theta}$ and $\mathcal{F}$ dependence. Analogously to the binary statistics, writing $|y| = \#\{i : y_i = 1\}$ we also have $\mathtt{tp} = |y| - \mathtt{fn}$ and $\mathtt{tn} = |1-y| - \mathtt{fp}$.

2.2 A general form for candidate loss functions

Give na batch at some point in training, we may seek to update the weights of the network to decrease $\mathtt{fn}$ and $\mathtt{fp}$ while increasing $\mathtt{tn}$ and $\mathtt{tp}$. Write $\partial_{\boldsymbol{\theta}}$ for derivatives with respect to the parameters. One may

readily verify that

$$\partial_{\boldsymbol{\theta}}\mathtt{tp}_{\mathcal{F}}(\boldsymbol{\theta}) = \sum_{i:y_i=1} \partial_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(x_i) = -\partial_{\boldsymbol{\theta}}\mathtt{fn}_{\mathcal{F}}(\boldsymbol{\theta}) \text{ and}$$
$$\partial_{\boldsymbol{\theta}}\mathtt{tn}_{\mathcal{F}}(\boldsymbol{\theta}) = -\sum_{i:y_i=0} \partial_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(x_i) = -\partial_{\boldsymbol{\theta}}\mathtt{fp}_{\mathcal{F}}(\boldsymbol{\theta}).$$

We observe that for a fixed batch, we generally expect that we will make an update to the parameters in the direction spanned by $\partial_{\boldsymbol{\theta}}\mathtt{tp}_{\mathcal{F}}(\boldsymbol{\theta})$ and $\partial_{\boldsymbol{\theta}}\mathtt{tn}_{\mathcal{F}}(\boldsymbol{\theta})$. We may write such an update for the batch at time t as

$$\varphi(t) = -U_t\partial_{\boldsymbol{\theta}}\mathtt{fp}_{\mathcal{F}_t}(\boldsymbol{\theta}_t) - V_t\partial_{\boldsymbol{\theta}}\mathtt{fn}_{\mathcal{F}_t}(\boldsymbol{\theta}_t) = \partial_{\boldsymbol{\theta}}(-U_t\mathtt{fp}_t - V_t\mathtt{fn}_t) = \partial_{\boldsymbol{\theta}}\Phi(\mathtt{fp}_t, \mathtt{fn}_t, U_t, V_t).$$

Here U_t and $V_t > 0$ are weights corresponding to the relative amount that the model needs to improve on the rates of false positives and false negatives.

Listing 1: A dynamic choice of gradient weights

```
def d_l(self,X,y,U,V,params): #(1-y)U-yV = U or -V when y=0 or 1 respectively
  ret=dict()
  dfp=self.d_fp(X,y,params)
  dfn=self.d_fn(X,y,params)
  dfp_frob_sq=0
  dfn_frob_sq=0
  for k in dfp:
    dfp_frob_sq+=jsum(dfp[k]**2)
    dfn_frob_sq+=jsum(dfn[k]**2)
  for k in dfp:
    ret[k]=U*dfp[k]/dfp_frob_sq+V*dfn[k]/dfn_frob_sq
  return ret
```

The loss function Ψ or equivalently the gradient $\varphi(t)$ may then be taken as the input to stochastic gradient descent or a related method, such as Adam.

In the next subsection we consider an existing loss function designed to control $\frac{FP}{FN}$, and relate it to this observation about the form of the gradient update.

2.3 Comparison with an existing loss function based approach

In order to control both false positives and false negatives at a specific relative desired rate, one may consider the family of quantities discussed by Rijsbergen in [2], though here are looking for quantities to minimise so we we modify conventions and use $1-$ the original figure:

$$F_\beta = 1 - (1+\beta^2)\frac{\mathbf{precision}\cdot\mathbf{recall}}{\beta^2\cdot\mathbf{precision}+\mathbf{recall}}.$$

This quantity can't be found directly but may be approximated based on a sample, but this won't yield a function to which we can apply gradient descent methods.

swapping out for the surrogates defined above we arrive at wwidetilde is referred to in [1] as the macro soft F_β score, up to the same modification of convention as before:

$$\begin{aligned}
\mathtt{F}_\beta =& 1 - (1+\beta^2)\frac{\mathbf{surrogate\ precision}\cdot\mathbf{surrogate\ recall}}{\beta^2\cdot\mathbf{surrogate\ precision}+\mathbf{surrogate\ recall}}\\
=& 1 - (1+\beta^2)\frac{\mathtt{tp}}{\beta^2(\mathtt{tp}+\mathtt{fn})+\mathtt{tp}+\mathtt{fp}}\\
=& \frac{\left(\frac{\beta^2}{1+\beta^2}\right)\mathtt{fn}+\left(\frac{1}{1+\beta^2}\right)\mathtt{fp}}{\mathtt{tp}+\left(\frac{\beta^2}{1+\beta^2}\right)\mathtt{fn}+\left(\frac{1}{1+\beta^2}\right)\mathtt{fp}}\\
=& \frac{\lambda\mathtt{fn}+(1-\lambda)\mathtt{fp}}{\mathtt{tp}+(\lambda\mathtt{fn}+(1-\lambda)\mathtt{fp})} = \frac{\lambda\mathtt{fn}+(1-\lambda)\mathtt{fp}}{|y|+((\lambda-1)\mathtt{fn}+(1-\lambda)\mathtt{fp})} = \Psi(\mathtt{fp},\mathtt{fn},\lambda,|y|),
\end{aligned}$$

where $\lambda = \frac{\beta^2}{1+\beta^2}$. One can see fromm this that the role of λ or equivalently β is to control the relative cost of a false positive versus a false negative, just as Rijsbergen explains the role of the original F_β

function in [2]. Write ∂_i for the partial derivative of a function with respect to its ith argument. The gradient of this loss then is given by

$$\partial_{\boldsymbol{\theta}} \mathtt{F}_\beta = (\partial_1 \Psi)\partial_{\boldsymbol{\theta}} \mathtt{fp} + (\partial_2 \Psi)\partial_{\boldsymbol{\theta}} \mathtt{fn}.$$

If we assume that the performance of a model is already good, so that $|y| \gg \mathtt{fp}$ and $\mathtt{fn}$, then it is easy to verify that we roughly have

$$\partial_1 \Psi, \partial_2 \Psi > 0 \text{ and } \frac{\partial_2 \Psi}{\partial_1 \Psi} \approx \tfrac{\lambda}{1-\lambda} = \beta^2,$$

in other words, the relative weighting reflects the target ratio of false positives to false negatives. This then is an example of a loss that perturbs in a direction in the positive cone spanned by $\partial_{\boldsymbol{\theta}} \mathtt{fp}$ and $\partial_{\boldsymbol{\theta}} \mathtt{fn}$.

In the present work we instead seek to dynamically choose the relative weighting U_t and V_t depending on the performance of the model. The magnitude of the gradient is relatively immaterial when passed to Adam, so we take $U_t \in (0,1)$ and $V_t = 1 - U_t$ for simplicity.

2.4 An adaptive weighting scheme

Here we give a possible method for choice of weights U_t and V_t. When the model has $\frac{FP}{FN}$ much larger than intended, we want $1 \approx U_t \gg V_t \approx 0$ and vice versa when $\frac{FP}{FN}$ is much smaller than intended. Fix hyperparameters κ and ν which represent the timescale of the moving average over which $\frac{FP}{FN}$ is evaluated and the degree to which the weighting scales accoridng to how off-target the model is. Let $\sigma(t) = \frac{1}{1+e^{-t}}$. Write $\widetilde{FP} = \frac{FP}{\#\mathcal{F}}$ and similarly for other quantities to denote the respective rates within a batch. Let $u_0 = 0$ and at batch t define U and V via the rule

$$u_{t+1} = \kappa u_t + (1-\kappa)(\frac{\widetilde{FP}}{\beta} - \beta\widetilde{FN}) \text{ and } U_t = \sigma(\nu u_t), V_t = 1 - U_t.$$

This choice allows the loss function to reflect the binary predictions that the model has been making. When there are too many false positives, $u_t > 0$ and so $U_t > \frac{1}{2}$ causing the loss to depend more upon the false positive rate, and vice versa in the case of too many false negatives.

Listing 2: Loss weight updating subroutine

```
def update_fp_w(self,y,y_pred):
  #Check which way the skew is.
  #We are aiming for fp/fn=beta**2.
  #u>0 -> too many fp
  #u<0 -> too many fn
  u=self.b_fp(y,y_pred)/self.beta-self.b_fn(y,y_pred)*self.beta
  #self.fp_weight=self.kappa*self.fp_weight+self.one_minus_kappa*u
  self.fp_weight=self.fp_weight+self.one_minus_kappa*u

  self.U=sigmoid(self.nu*self.fp_weight)
  self.V=1-self.U
```

This is implemented via a modified adam type gradient descent step. In fact, dl needn't be computed by computing a single gradient, but may also take weighted sums of $d\mathtt{fp}$ and $d\mathtt{fn}$ computed by other means.

Listing 3: A stochastic gradient routine with binary prediction derived weighting updates

```
def adam_step(self,X,y):
  y_pred=self.infer(X)
  self.update_fp_w(y,y_pred)
  upd=self.d_l(X,y,self.U,self.V,self.params)

  self.v*=self.beta2
  for k in upd:
    self.m[k]*=self.beta1
    self.m[k]+=self.one_minus_beta1*upd[k]
    self.v+=self.one_minus_beta2*jsum(square(upd[k]))

  mult=self.lr/(self.eps+sqrt(self.v))
  for k in upd:
    self.params[k]-=mult*self.m[k]
```

References

[1] Namgil Lee, Heejung Yang, and Hojin Yoo. A surrogate loss function for optimization of F_β score in binary classification with imbalanced data. *arXiv preprint arXiv:2104.01459*, 2021.

[2] C. J. Van Rijsbergen. *Information Retrieval*. 2nd edition, 1979.