Explicit AUC maximization

How to explicitly optimize for maximum area under ROC

Michael Larionov, PhD Nov 13, 2019 · 5 min read ★

Photo by André Sanano on Unsplash

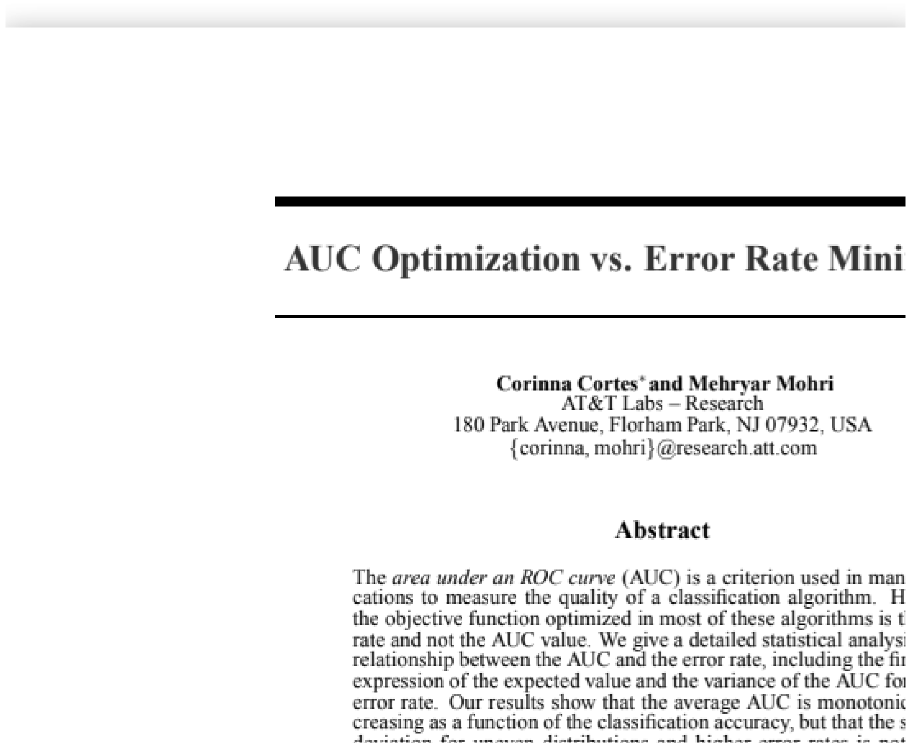
Submissions are evaluated on area under the ROC curve between the predicted probability and the observed target.

The fact that the results are evaluated based on AUC makes sense for fraud detection tasks for several reasons:

1. The data sets are often unbalanced, which makes it difficult to optimize for the recall or other simple metrics, unless you use oversampling or undersampling of data.
2. In reality, the cost of false negative and false positive is different, and the actual task should include this information and optimize the actual cost, not mathematical quantities like recall or F1 score.
3. High AUC increases the chance that we will be able to find an optimal threshold that will satisfy these requirements.

Usually, AUC is optimized only during hyperparameters tuning, whereas during training they use cross-entropy loss. Why not optimize AUC in the first place instead of cross-entropy?

The same question was raised in this paper:

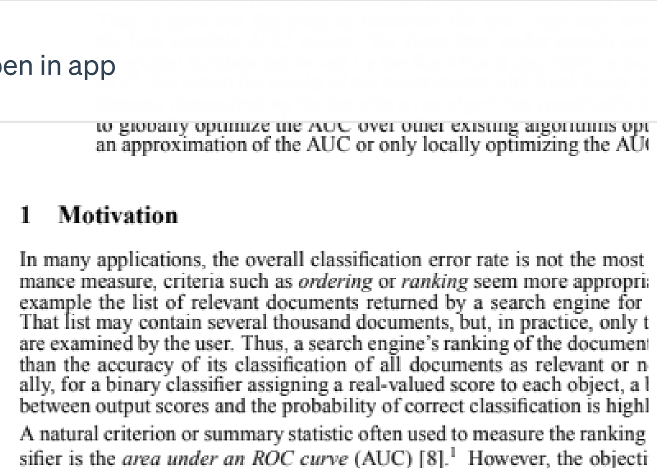


AUC Optimization vs. Error Rate Mini

Corinna Cortes* and Mehryar Mohri
AT&T Labs – Research
180 Park Avenue, Florham Park, NJ 07932, USA
{corinna, mohri}@research.att.com

Abstract

The *area under an ROC curve* (AUC) is a criterion used in man
cations to measure the quality of a classification algorithm. H
the objective function optimized in most of these algorithms is t
rate and not the AUC value. We give a detailed statistical analysi
relationship between the AUC and the error rate, including the fi
expression of the expected value and the variance of the AUC fo
error rate. Our results show that the average AUC is monotonic
creasing as a function of the classification accuracy, but that the s
deviation for uneven distributions and higher error rates is not

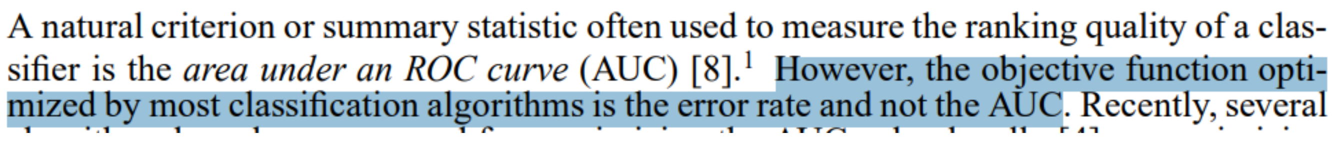

to globally optimize the AUC over other existing algorithms opt
an approximation of the AUC or only locally optimizing the AU

1 Motivation

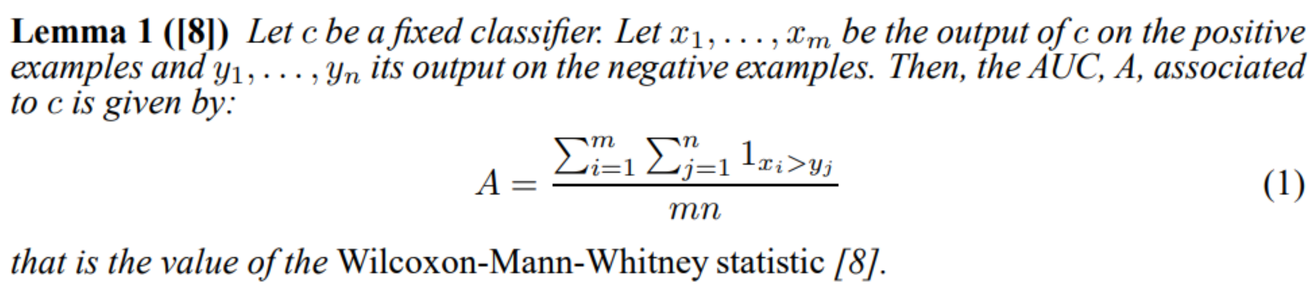
In many applications, the overall classification error rate is not the most
mance measure, criteria such as *ordering* or *ranking* seem more appropri
example the list of relevant documents returned by a search engine for
That list may contain several thousand documents, but, in practice, only t
are examined by the user. Thus, a search engine's ranking of the documen
than the accuracy of its classification of all documents as relevant or n
ally, for a binary classifier assigning a real-valued score to each object, a l
between output scores and the probability of correct classification is highl

A natural criterion or summary statistic often used to measure the ranking
sifier is the *area under an ROC curve* (AUC) [8].[1] However, the objecti

The problem is summarized well in the paper's intro:


A natural criterion or summary statistic often used to measure the ranking quality of a classifier is the *area under an ROC curve* (AUC) [8].[1] However, the objective function optimized by most classification algorithms is the error rate and not the AUC. Recently, several

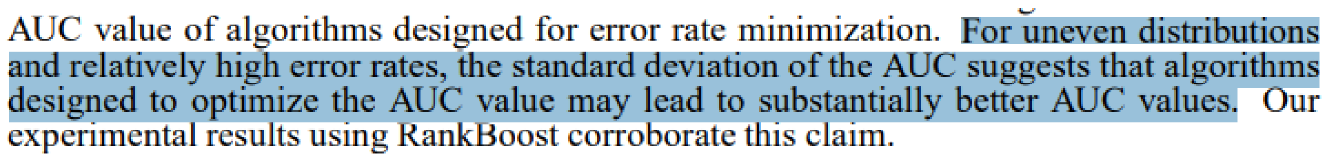
What if we did use the AUC as the optimization function instead of cross-entropy? The AUC curve can be computed using Wilcoxon-Mann-Whitney statistic:


Lemma 1 ([8]) *Let c be a fixed classifier. Let $x_1, \dots, x_m$ be the output of c on the positive examples and $y_1, \dots, y_n$ its output on the negative examples. Then, the AUC, A, associated to c is given by:*

$$A = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} 1_{x_i > y_j}}{mn} \tag{1}$$

that is the value of the Wilcoxon-Mann-Whitney statistic *[8].*

If x and y are probabilities, the metric penalizes any occurrence where the probability of the actual positive case is less than the probability of the negative case. This is not a


AUC value of algorithms designed for error rate minimization. For uneven distributions and relatively high error rates, the standard deviation of the AUC suggests that algorithms designed to optimize the AUC value may lead to substantially better AUC values. Our experimental results using RankBoost corroborate this claim.

The AUC in the equation above is not a smooth function of probabilities, because it is constructed using step functions. We can try to smooth it out, to make sure the function is always differentiable, so that we can use the conventional optimization algorithms. One of the ways to get around of this, as demonstrated in the paper, is the RankBoost loss function:

$$L = \frac{1}{mn}\sum_{i,j} exp(-\alpha(x_i - y_i))$$

Here x and y are probabilities for the positive and negative class respectively. For example, in the logistic regression model both x and y can be expressed as sigmoid functions:

$$x = \frac{1}{1+e^{-(b+w\xi)}}$$

Here ξ is a predictor variable(s).

I have to admit that these models are nothing new. They are called *pairwise comparison models* and pioneered by Thurstone in the 1920s. For more information see this book. The foundation of this analysis is Bradley-Terry-Luce model, but the most well-known variation is Elo algorithm. The model itself is very simple and is derived by maximizing the likelihood function:

$n \quad ij \qquad\qquad ij$

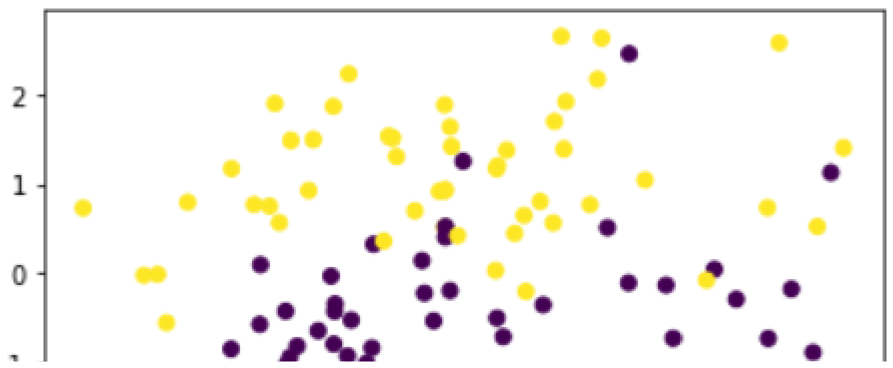
In our context, M_ij is 1 if i is from the positive class and j is from the negative class. In all other cases, it is 0. The ability values α here are logits, not the probabilities. This makes the model simpler than RankBoost, and also allows us to use it as the final layer of a neural network (in which case the values α will be just the activations of the previous layer). The loss function is:

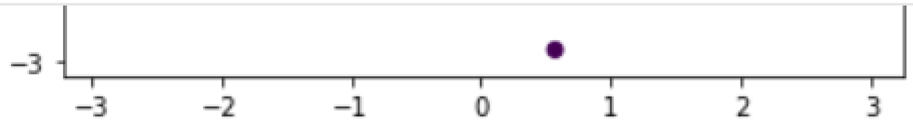
$$L \propto -\sum_{i,j} \ln \sigma(\alpha_i - \alpha_j)$$

Where i and j are positive and negative examples respectively.

We will demonstrate this approach using a simple classification problem. We will use TensorFlow to take advantage of automatic differentiation.

```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline

from sklearn.datasets import make_classification
X, y = make_classification(100, 2, 2, 0, 0)
plt.scatter(X[:,0], X[:,1], c=y);
```





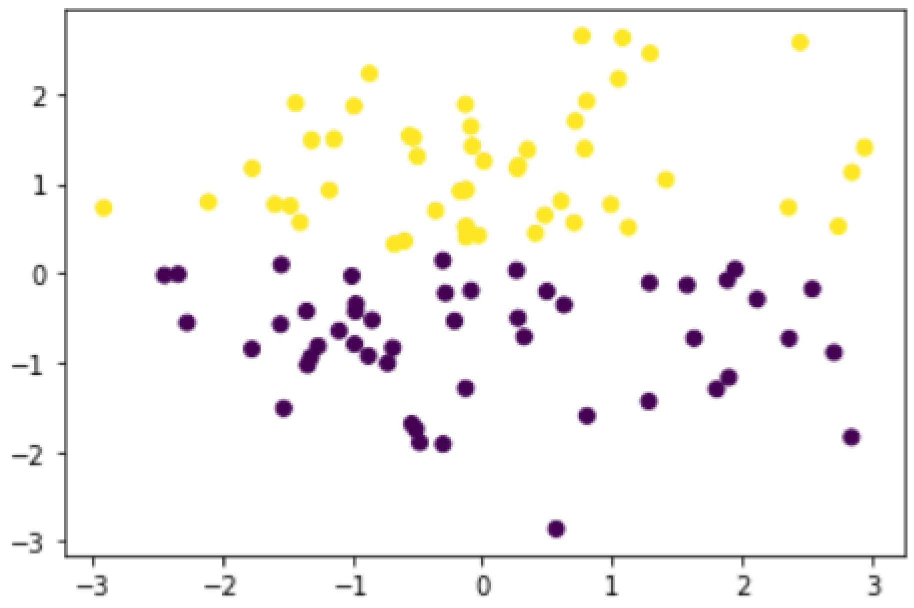
As you see, this is a binary classification problem using synthetic data. Instead of trying to correctly classify the results, we will try to maximize AUC.

```
y = y.reshape(-1,1)

import tensorflow as tf
import keras.layers as L

learning_rate = 1.0
input_layer = L.Input((2,))
dense = L.Dense(units=1)
activations = dense(input_layer)
predictions = tf.sigmoid(activations)
# This is the new cost function.
cost = - tf.reduce_mean(tf.sigmoid(activations @ tf.transpose(activations)) * np.maximum(y @ np
optimizer = tf.train.GradientDescentOptimizer(learning_rate)
training_op = optimizer.minimize(cost)

init = tf.global_variables_initializer()
n_epochs = 500

with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        if epoch % 100 == 0:
          print("Epoch", epoch, "Cost =", cost.eval(feed_dict={input_layer: X}))
        sess.run(training_op, feed_dict={input_layer: X})

    prediction_values = predictions.eval(feed_dict={input_layer: X})
```

auc02.py hosted with ♡ by GitHub view raw

```
Epoch 0 Cost = -0.12107406
Epoch 100 Cost = -0.24906836
Epoch 200 Cost = -0.2497198
Epoch 300 Cost = -0.24984348
Epoch 400 Cost = -0.24989356
```

```python
prediction_results = prediction_values.reshape((-1)) > 0.945
plt.scatter(X[:,0], X[:,1], c=prediction_results.reshape(-1));
```



Discussion

The plot below is based on the threshold 0.945. If I were to choose the conventional threshold 0.5 we would get all observations in one class. Why is that? The reason is that AUC is not sensitive to the bias value. Indeed, since in this model we subtracting pairwise activations, the bias value cancels out. We can add additional term to the cost function to bring the bias closer to 0.5. Or choose to do what we have done above, by hand-picking the threshold that works for us. I want to remind you, that we are optimizing for AUC, so if all observations are predicted to fall into a single class, it is cool with us as long as the AUC has the optimal value.

Conclusion

One can wonder if there is any usefulness in this approach for any real machine learning projects, other than data science competitions that use AUC as the evaluation criterion. But I would argue that it can be useful in case of unbalanced data set as the first step in machine learning. The second step would be to learn the bias parameter and generally fine-tuning the model by using a business-based optimization metric.

number of observations, which could become a performance bottleneck as the number of observations grows. Also one has to come up with a good batching strategy that correctly estimates AUC based on a number of smaller samples (batches).

All code is available in my github repository. Also find more information about RankBoost below.

Boosting

We consider next how to learn to rank a set of objects, a problem that arises naturally in a variety of domains. For…

mitpress.mit.edu

Machine Learning Data Science Optimization TensorFlow Kaggle