Analysis of Model Evaluation Metrics

David Wang

March 30 2023

1 Precision and Recall

Precision and recall are common metrics used to evaluate the performance of a binary classification model.

We are given a dataset and a model which predicts, for each sample in the dataset, True or False. We define this as the postive class (True) and the negative class (False).

We also define:

- The true positives (TP) as the number of "True" predictions that are actually correct.
- The false positives (FP) as the number of "True" predictions that are not correct.
- The true negatives (TN) as the number of "False" predictions that are correct.
- The false negatives (FN) as the number of "False" predictions that are incorrect.

Now we define

- Precision, as the fraction of "True" predictions made by the model that should actually be "True"

$$\text{precision} = \frac{TP}{TP + FP}$$

- Recall, as the fraction of "True" examples which was correctly identified as "True" by the model

$$\text{recall} = \frac{TP}{TP + FN}$$

However, if we have an unbalanced dataset, then these metrics may not be the most informative, as will be demonstrated below.

2 Matthew's Correlation Coefficient

From some preliminary research, another metric which could be utilized is the Matthews's correlation coefficient (MCC), which is defined as

$$MCC = \frac{TN \cdot TP - FN \cdot FP}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The score produced by MCC provides a good evaluation on the performance of the model, even with unbalanced priors.

3 Tests

In the case where we care about the performance of the model on both the "True" and "False" labels, it is more intuitive to think of the two labels as "class A" and "class B". If we use precision and recall, the metrics do not take into account the number of elements in each class, and can produce very skewed results in the case of unbalanced priors. A Python script (A.1) is written to test and illustrate a more intuitive understanding of these metrics. For each test, we construct a confusion matrix of the form

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix}$$

and compare the efficacy of accuracy, precision, recall, and MCC on evaluating the performance of the model. We also swap the labels (ie change the negative class into the positive class) to show the potential problems with evalution using precision and recall.

3.1 Test 1

```
>>>>> test 1 >>>>>>
[[999   1]
 [  9   1]]
1000 in class A, 999 correctly predicted,
10 in class B, 1 correctly predicted
accuracy: 0.990, precision: 0.991, recall: 0.999
mcc: 0.220

------swapping labels--------
10 in class A, 1 correctly predicted,
1000 in class B, 999 correctly predicted
accuracy: 0.990, precision: 0.500, recall: 0.100
mcc: 0.220
```

Initially, we see that accuracy, precision, and recall are all very high. However, when we switch the labels between class A and class B, we notice that the precision and recall goes down significantly. So even though the model had a

9/10 error rate in predicting class B, because the number of samples for class B was so low, this did not reflect with precision and recall. We see, however, that MCC is invariant to the switching of labels and accurately reflects the bad performance of our model in predictions of class B.

3.2 Test 2

```
>>>>> test 2 >>>>>>
[[999   1]
 [  1   9]]
1000 in class A, 999 correctly predicted,
10 in class B, 9 correctly predicted
accuracy: 0.998, precision: 0.999, recall: 0.999
mcc: 0.899

------swapping labels--------
10 in class A, 9 correctly predicted,
1000 in class B, 999 correctly predicted
accuracy: 0.998, precision: 0.900, recall: 0.900
mcc: 0.899
```

Here, per class accuracy is high for A, but only 90% for B. However, the number of samples in B is again very small, and thus, neither precision, accuracy, nor recall reflects the bad performance seen for that class. When the labels are swapped, we now see the drop in precision and recall. Meanwhile, MCC accounts for this without the need of label swapping and re-evaluating.

3.3 Test 3

```
>>>>> test 3 >>>>>>
[[999   1]
 [998   2]]
1000 in class A, 999 correctly predicted,
1000 in class B, 2 correctly predicted
accuracy: 0.500, precision: 0.500, recall: 0.999
mcc: 0.013

------swapping labels--------
1000 in class A, 2 correctly predicted,
1000 in class B, 999 correctly predicted
accuracy: 0.500, precision: 0.667, recall: 0.002
mcc: 0.013
```

Here, the dataset contains equal distributions of samples from class A and class B. However, model performance on class B is extremely poor. We note that accuracy is affected since the number of samples in each class is similar. However, the extremely bad performance on class B is only obvious when we

evaluate the recall after switching labels. Meanwhile, MCC is able to accurately reflect the poor performance of the model.

3.4 Test 4

```
>>>>> test 4 >>>>>>
[[  1 999]
 [  9   1]]
1000 in class A, 1 correctly predicted,
10 in class B, 1 correctly predicted
accuracy: 0.002, precision: 0.100, recall: 0.001
mcc: -0.899

------swapping labels--------
10 in class A, 1 correctly predicted,
1000 in class B, 1 correctly predicted
accuracy: 0.002, precision: 0.001, recall: 0.100
mcc: -0.899
```

Here, we see that the model has bad performance on both classes, and MCC becomes negative in this case.

3.5 Test 5

```
>>>>> test 5 >>>>>>
[[999   1]
 [  2 998]]
1000 in class A, 999 correctly predicted,
1000 in class B, 998 correctly predicted
accuracy: 0.999, precision: 0.998, recall: 0.999
mcc: 0.997

------swapping labels--------
1000 in class A, 998 correctly predicted,
1000 in class B, 999 correctly predicted
accuracy: 0.999, precision: 0.999, recall: 0.998
mcc: 0.997
```

Here, we see that the model has good performance on both classes, and there is an equal number of samples for each class. MCC is high with no surprise here, and precision and recall also correctly evaluate the good-performance of the model.

4 Relation to True and False Positive Rates

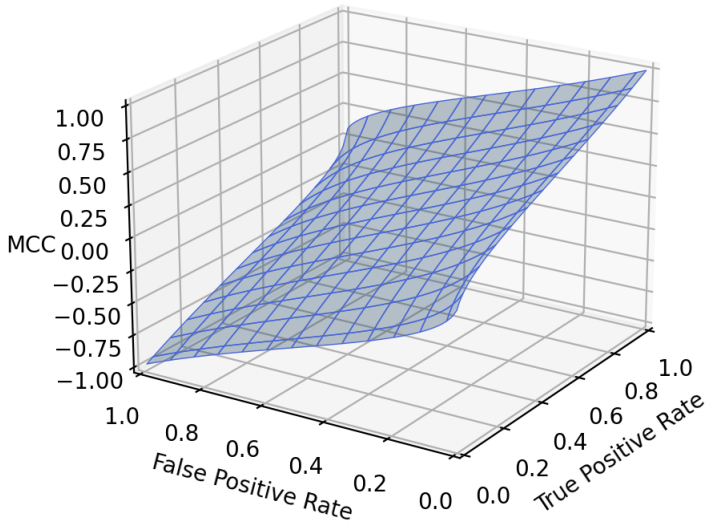
It is interesting to see the relationship between the Matthew's Correlation Coefficient to the true and false positive rates. The true positive rate is the recall

$$\text{TPR} = \frac{TP}{TP + FN}$$

and the false positive rate is $1 -$ recall after the labels have been switched

$$\text{FPR} = \frac{FP}{FP + TN}$$

Writing a script in python (A.2) to visualize the relation, the following plot is obtained



This confirms that MCC can be used as a good metric for evaluating our model, as at a max value of MCC = 1.0, the model performs perfectly with a TPR of 1.0 and FPR of 0.0.

5 Conclusion

From this initial exploration, we determine that the Matthew's Correlation Coefficient provides a more robust method to evaluting the performance of a binary classification model, which is insensitive to the prior distributions. Interesting further explorations could investigate a potential generalization of this metric to multi-class classification.

A Appendix

The following Python scripts were written and utilized to explore the effectiveness of MCC. The codebase can also be found at

`https://github.com/davidw0311/eval_metrics`

A.1 Code for Test Cases

```python
import numpy as np

def get_metrics(CM):
    tp = CM[0,0]
    tn = CM[1,1]
    fp = CM[1,0]
    fn = CM[0,1]

    # precision, recall, and accuracy
    precision = tp/(tp + fp)
    recall = tp/(tp+fn)
    acc = (tp+tn)/(tp+tn+fp+fn)
    print(f'accuracy: {acc:.3f}, precision: {precision:.3f}, recall: {recall:.3f}')

    # matthew's correlation coefficient
    mcc = (tn*tp - fn*fp)/np.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))
    print(f'mcc: {mcc:.3f}')

def evaluate(CM):
    print(f'{CM[0,0]+CM[0,1]} in class A, {CM[0,0]} correctly predicted,')
    print(f'{CM[1,0]+CM[1,1]} in class B, {CM[1,1]} correctly predicted')
    get_metrics(CM)


def test(CM, num):
    print(f'>>>>> test {num} >>>>>>')
    print(CM)
    evaluate(CM)

    print('\n------swapping labels--------')
    # switch labels by swapping rows, then swapping columns
    CM = CM[[1,0]][:,[1,0]]
    evaluate(CM)
    print('\n\n')


# create test cases of confusions matrices
cms = [
    np.array([[999, 1],
              [9,1]]),
    np.array([[999, 1],
              [1,9]]),
    np.array([[999, 1],
              [998, 2]]),
    np.array([[999, 1],
              [2, 998]])
]

for i, cm in enumerate(cms):
    test(cm, i+1)
```

A.2 Code for Plots of TPR and FPR to MCC

```python
import numpy as np
import matplotlib.pyplot as plt

def get_metrics(CM):
    tp = CM[0,0]
    tn = CM[1,1]
    fp = CM[1,0]
    fn = CM[0,1]

    tpr = tp/(tp+fn)
    fpr = fp/(fp + tn)
    mcc = (tn*tp - fn*fp)/np.sqrt((tp+fp)*(tp+fn)*(tn+fp)*(tn+fn))

    return tpr, fpr, mcc


pAs = []
grid = 100
tprs = np.zeros((grid,grid))
fprs = np.zeros((grid,grid))
mccs = np.zeros((grid,grid))

for i in range(grid):
    for j in range(grid):
        CM = np.array([[grid-i, i],[grid-j,j]])
        tpr, fpr, mcc = get_metrics(CM)
        tprs[i, :] = tpr
        fprs[:, j] = fpr
        mccs[i,j] = mcc


ax = plt.figure().add_subplot(projection='3d')

ax.plot_surface(tprs, fprs, mccs, edgecolor='royalblue', lw=0.5, rstride=8, cstride=8,
                alpha=0.3)
ax.set(xlim=(0, 1), ylim=(0, 1), zlim=(-1, 1),
       xlabel='True Positive Rate', ylabel='False Positive Rate', zlabel='MCC')

plt.show()
```