BOOK "DATA ANALYSIS TECHNIQUES TO WIN KAGGLE"

11 Evaluation Metrics Data Scientists Should Be Familiar with— Lessons from A High-rank Kagglers' New Book

It is about loss function, right? No, it isn't.

Moto DEI Nov 29, 2019 · 9 min read

Photo by Andreas Wagner on Unsplash

This is a post to pick up tips introduced in a new book *"Data Analysis Techniques to Win Kaggle"*, authored by three high-rank Kagglers (not including myself thus this is not a personal promotion! :))

For the full table of contents of this book, see my other post.

. . .

Table of Contents:

. . .

Evaluation Metric vs. Loss Function

Evaluation metric, a theme of this post, is a somewhat confusing concept for ML beginners with another related but separate concept, *loss function*. They are similar in a sense they could be the same when we are lucky enough, but it will not happen every time.

Evaluation metric is a metric "we want" to minimize or maximize through the modeling process, while loss function is a metric "the model will" minimize through the model training.

Photo by AbsolutVision on Unsplash

Giving an example in simple logistic regression:

- **Loss function** is the quantity which the model will minimize over the training. It is also called as cost function or objective function. Very basic version of logistic regression uses negative log likelihood as loss function. Searching the parameters of the model to minimize the negative log likelihood is something which is done in training the model.
- **Evaluation metric** is the metric we want the model to maximize. It is separate from the model training process, and *ideally evaluation metric should reflect our business needs*. Depending on business application, we may want to maximize AUC, or we may care how good the recall is.

Loss function is tightly linked to the model and typically a model has restrictive list of loss function candidates — e.g. [*negative log likelihood, negative log likelihood with penalty term(s)]* for logistic regression — , because the selection of loss function is the part of core determination by model algorithm creator.

On the other hand, the evaluation metric can be whatever we want to set. Ultimately we can use '1' for any model, though using universal '1' for evaluation metric never makes sense. How high the evaluation metric is

usually measured by the data not used in training such as out-of-fold data or test data.

. . .

Evaluation score is widely used in the hyper parameter tuning, but for more experienced data science people, one of the most useful cases to understand the distinction of loss function and evaluation metric is **early stopping**. Early stopping is a technique typically used to determine when to stop training in order to avoid overfitting in boosting type of models or neural network type of models.

> *[Early Stopping]*
>
> *While the model is tuning the parameter based on minimization of **loss function**, check how much one iteration improve the **evaluation metric** and if not a lot improvement any more, stop learning.*

. . .

In Kaggle, the competition participants are ranked by "leaderboard score".

There are two types of leaderboard scores, public and private, but it's another story. Leaderboard score can be categorized to an evaluation metric the competition host sets to meet their business goal or needs.

> Then, understanding evaluations score and how to maximize them through the model training should be a road to win the Kaggle competition.

. . .

So, now you understand the difference of evaluation metric and loss function. Next, we will see the common evaluation metrics further, with their properties.

Photo by Tierra Mallorca on Unsplash

. . .

Evaluation Metrics Used in Regression Task

#1 — RMSE (Root Mean Squared Error)

(Used in Kaggle competition "Elo Merchant Category Recommendation")

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

Formula of RMSE

- RMSE has solutions equivalent to the solutions of maximum likelihood method when assuming the errors are normally distributed. Therefore, favorable when the distribution of y is normal around some base structure.
- RMSE is sensitive to outliers. Therefore, clipping or removing the outliers in advance is important.

```python
import numpy as np
from sklearn.metrics import mean_squared_error

y_true = [1.0, 1.5, 2.0, 2.5, 3.0]
y_pred = [0.9, 1.7, 3.0, 2.0, 2.7]

rmse = np.sqrt(mean_squared_error(y_true, y_pred))
# 0.5272570530585626
```

gistfile2019112801.py hosted with ♡ by GitHub view raw

. . .

#2 — RMSLE (Root Mean Squared Logarithmic Error)

(Used in Kaggle competition "Recruit Restaurant Visitor Forecasting")

$$RMSLE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(log(1+y_i) - log(1+\hat{y}_i))^2} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(log\frac{1+y_i}{1+\hat{y}_i})^2}$$

Formula of RMSLE

- RMSLE is used when y has long tail distribution, or we are interested in the ratio of true value and predicted value.
- 1 is added to avoid divergence when y is zero.

```python
import numpy as np
from sklearn.metrics import mean_squared_error

y_true = [1.0, 1.5, 2.0, 2.5, 3.0]
y_pred = [0.9, 1.7, 3.0, 2.0, 2.7]

rmsle = np.sqrt(mean_squared_error(np.log1p(y_true), np.log1p(y_pred)))
#0.15566336290314164
```

gistfile2019112802.py hosted with ♡ by GitHub view raw

. . .

#3 — MAE (Mean Absolute Error)

(Used in Kaggle competition "Allstate Claims Severity")

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Formula of MAE

- Robust to outliers compared to RMSE.
- Not second-order differentiable at true y = predicted y. Therefore, some algorithms such as xgboost does not allow MAE as loss function.
- Instead of MAE, the approximated functions such as "Fair function" or "Pseudo-Huber function" may be usable.

$$Fair function = c^2\left(\frac{|y-\hat{y}|}{c} - ln\left(1 + \frac{|y-\hat{y}|}{c}\right)\right)$$

Fair Function

$$PseudoHuber = \delta^2\left(\sqrt{1 + \left(\frac{y-\hat{y}}{c}\right)^2} - 1\right)$$

Pseudo-Huber Function

```python
from sklearn.metrics import mean_absolute_error

y_true = [1.0, 1.5, 2.0, 2.5, 3.0]
y_pred = [0.9, 1.7, 3.0, 2.0, 2.7]

mae = mean_absolute_error(y_true, y_pred)
#0.420
```

gistfile2019112803.py hosted with ♡ by GitHub view raw

See also this nice comparison chart MAE vs. Fair function vs. Pseudo-Huber function posted by Ioannis Dassios.

. . .

#4 — R-Squared (R^2)

(Used in Kaggle competition "Mercedes-Benz Greener Manufacturing")

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2}, \text{ where } \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$$

Formula of R-Squared

- $0 \leq R^2 \leq 1$ (commonly but some worst cases you will see negative values) and higher R^2 is better, which means the prediction is closer to the data.
- Denominator does not depend on the prediction and fixed once the data is set. Therefore, **maximizing R^2 is equivalent to minimizing the RMSE.**

```python
from sklearn.metrics import r2_score

y_true = [1.0, 1.5, 2.0, 2.5, 3.0]
y_pred = [0.9, 1.7, 3.0, 2.0, 2.7]

r2 = r2_score(y_true, y_pred)
#0.444
```

gistfile2019112804.py hosted with ♡ by GitHub view raw

Photo by Riccardo Pallaoro on Unsplash

. . .

Evaluation Metrics Used in 0/1 Prediction in Binary Classification Task

#5 — Accuracy and Error Rate

(Used in Kaggle competition "Text Normalization Challenge — English Language")

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}, \; error\ rate = 1 - accurary$$

Formula of Accuracy and Error Rate

Here, the two-lettered alphabets are of course coming from the confusion matrix.

		True Value	
		Positive	Negative
Prediction	Positive	True Positive (=TP)	False Positive (=FP)
	Negative	False Negative (=FN)	True Negative (=TN)

Confusion Matrix

- **NG to use for the imbalanced data** because very bad model which predicts everything is positive or negative can easily hack the high accuracy in imbalanced data.
- **For imbalanced data instead, use F1-score, Fbeta-score, MCC, or balanced accuracy** introduced later.

```
1  from sklearn.metrics import accuracy_score
2
3  y_true = [1, 0, 1, 1, 0, 1, 1, 0]
4  y_pred = [0, 0, 1, 1, 0, 0, 1, 1]
5
6  accuracy = accuracy_score(y_true, y_pred)
7  accuracy
```

```
8   #0.625
```

gistfile2019112805.py hosted with ♡ by GitHub view raw

. . .

#6 — Precision and Recall

$$precision = \frac{TP}{TP+FP}, \quad recall = \frac{TP}{TP+FN}$$

Formula of precision and recall

- Precision and recall represent the proportion of TP over the first horizontal row or over the first vertical column in the confusion matrix, respectively.
- Are the value between 0 and 1, higher is better.
- Precision and recall move the opposite direction when the cutoff point (=the threshold to determines the prediction is 0 or 1 from the probability) moves.
- **Are asymmetric to positive and negative swap**, which means when we change which to call as "positive" (whether y=1 or 0), precision and recall change.

```python
from sklearn.metrics import precision_score, recall_score

y_true = [1, 0, 1, 1, 0, 1, 1, 0]
y_pred = [0, 0, 1, 1, 0, 0, 1, 1]

precision, recall = precision_score(y_true, y_pred), recall_score(y_true, y_pred)
print(precision, recall)
#0.75 0.6
```

gistfile2019112806.py hosted with ♡ by GitHub view raw

. . .

#7 — F1-Score and Fbeta-Score

(Used in Kaggle competition "Quora Insincere Questions Classification")

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

$$F_1 = F_{\beta=1} = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = \frac{2\cdot recall \cdot precision}{recall + precision}$$

Formula of F1-Score and Fbeta-Score

- It is called 'harmonic mean' of precision and recall.

- Are the value between 0 and 1, higher is better.
- Fbeta-score can change the balance between recall and precision, and when **$\beta < 1$ then weights to precision, $\beta > 1$ then weights to recall**. Potentially usable when we prefer the one to the other, like the medical diagnosis usually prefers false positive over false negative because the former one brings just additional cost but the latter one brings late treatment and may lead to death.
- Again, **these are asymmetric to positive and negative swap**, just as precision and recall aren't either.

```python
from sklearn.metrics import f1_score, fbeta_score

y_true = [1, 0, 1, 1, 0, 1, 1, 0]
y_pred = [0, 0, 1, 1, 0, 0, 1, 1]

f1, fbeta = f1_score(y_true, y_pred), fbeta_score(y_true, y_pred, beta=2)
print(f1, fbeta)
#0.67 0,625
```

gistfile2019112807.py hosted with ♡ by GitHub view raw

. . .

#8 — MCC (Matthews Correlation Coefficient)

(Used in Kaggle competition "Bosch Production Line Performance")

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

Formula of MCC

- Takes value between -1 and 1, higher is better. 0 means the prediction is equivalent to random.
- Finally, **this is symmetric to positive and negative swap!**

```python
from sklearn.metrics import matthews_corrcoef

y_true = [1, 0, 1, 1, 0, 1, 1, 0]
y_pred = [0, 0, 1, 1, 0, 0, 1, 1]

y_true_swap = [0, 1, 0, 0, 1, 0, 0, 1]
y_pred_swap = [1, 1, 0, 0, 1, 1, 0, 0]

mcc = matthews_corrcoef(y_true, y_pred)
print(mcc)
# 0.2582

mcc_swap = matthews_corrcoef(y_true_swap, y_pred_swap)
print(mcc_swap)
# 0.2582
```

gistfile2019112808.py hosted with ♡ by GitHub view raw

. . .

#9 — Balanced Accuracy

Balanced accuracy is a metric usable for both of binary classification and multi-class classification.

$$balanced\ accuacy = \frac{1}{M} \sum_{m=1}^{M} \frac{r_m}{n_m}$$

Formula of balanced accuracy

Where, M: number of classes, n_m: data size belongs to class m, r_m: number of data accurately predicted belonging to class m.

Here, if the problem is binary classification,

$$balanced\ accuracy\ (binary\ classification) = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

Balanced accuracy when binary classification

- Applicable to both multi-class classification and binary class classification.

- Value between 0 and 1, higher is better.
- Put higher weight to accurate prediction of smaller classes, therefore **good to use in imbalanced data.**

```python
from sklearn.metrics import balanced_accuracy_score

y_true = [0, 1, 0, 0, 1, 0]
y_pred = [0, 1, 0, 0, 0, 1]
balanced_accuracy_score(y_true, y_pred)
# 0.625
```

gistfile2019112809.py hosted with ♡ by GitHub view raw

Photo by Denys Nevozhai on Unsplash

. . .

Evaluation Metrics Used in Probability Prediction in Binary Classification Task

#10 — Log Loss (or Cross Entropy or Negative Log-Likelihood)

(Used in Kaggle competition "Quora Question Pairs")

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log p_i + (1 - y_i)log(1 - p_i))$$

Formula of logloss

- A usual suspect in classification.
- This is *loss* and higher is worse.

- When p_i is low while y_i=1, meaning not good probability prediction, or the other way around, logloss goes high.
- Common in loss function of many models as well.

```python
from sklearn.metrics import log_loss

y_true =  [0, 1, 0, 0, 1, 0]
y_proba = [0.1, 0.4, 0.8, 0.1, 0.9, 0.4]
log_loss(y_true, y_proba)
# 0.559
```

gistfile2019112810.py hosted with ♡ by GitHub view raw

. . .

#11 — AUCROC (Area Under the Receiver Operating Characteristic Curve)

(Used in Kaggle competition "Home Credit Default Risk")

This is also called as just AUC typically but to avoid confusion with AUCPR (Area under the Precision-Recall Curve), I stick to AUCROC.

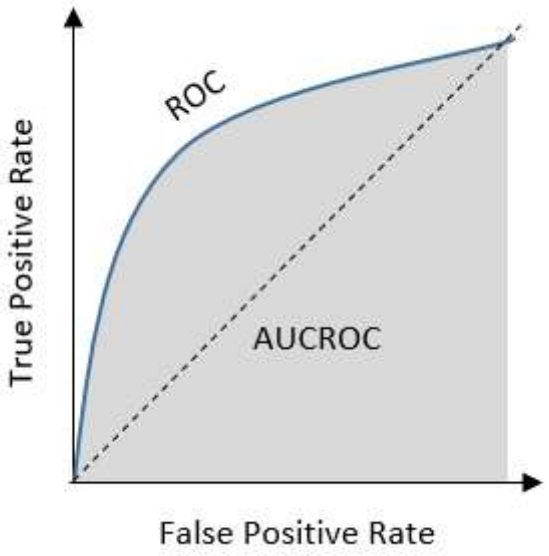


Illustration of AUCROC and ROC

- Values from 0 to 1. 0.5 means the prediction is equivalent to random.
- Another usual suspect in classification, but only for binary classification not for multi-class classification.
- The relation with gini coefficient is as follows. Maximization of AUCROC is equivalent to maximization of gini coefficient.

$$Gini = 2 \cdot AUCROC - 1$$

Relation with Gini and AUCROC

- Equivalently representable in the form:

$$AUCROC = \frac{number\ of\ combinations\ (i,j)\ satisfying\ (y_i=1, y_j=0, \hat{y}_i > \hat{y}_j)}{number\ of\ combinations\ (i,j)\ satisfying\ (y_i=1, y_j=0)}$$

Alternative representation of AUCROC

- Therefore, **only the order of the predicted probabilities matter**, which means AUCROC is the same when the probabilities for four records are [0.1, 0.3, 0.7, 0.9] or [0.01, 0.02, 0.3, 0.99], as long as the order is preserved.
- This alternative representation gives easier access to know how much model prediction improvement contributes the score improvement, than the original definition.
- In case of imbalanced data (e.g. positive data is minor to negative), having consistently high probabilities to positive minor class data is important to AUCROC, while the probabilities for negative major class data is insensitive to the noise.

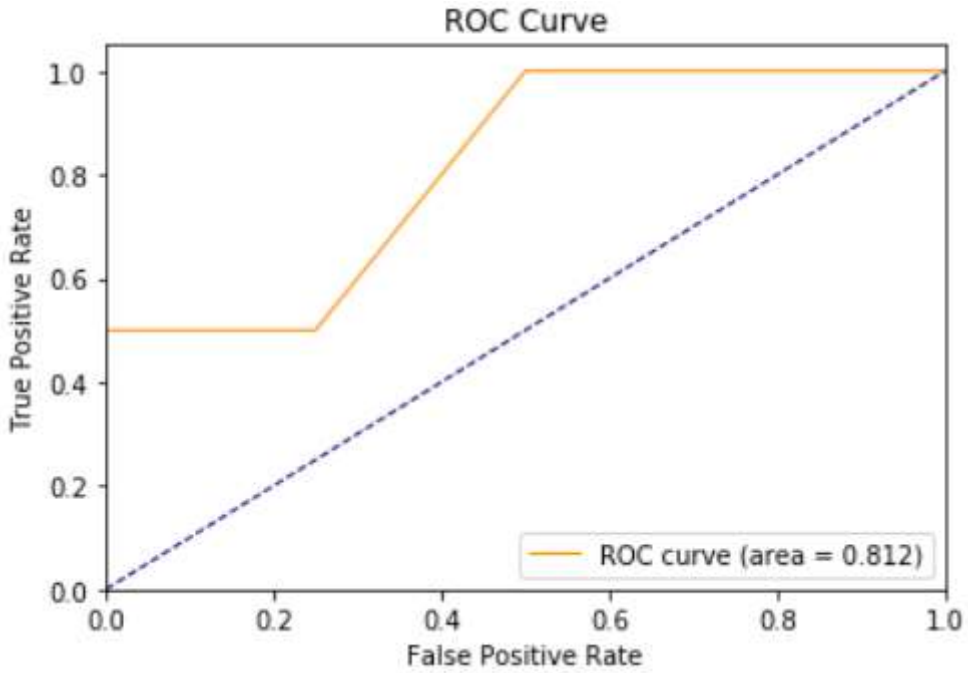
```
1  from sklearn.metrics import roc_auc_score, roc_curve, auc
2  import matplotlib.pyplot as plt
3
4
5  def plotROC(y true, y proda):
```

```
    fpr = dict()
    tpr = dict()
    threshold = dict()
    roc_auc = dict()
    for i in range(2):
        fpr[i], tpr[i], threshold[i] = roc_curve(abs(np.array(y_true)-abs(i-1)),y_proba)
        roc_auc[i] = auc(fpr[i], tpr[i])
    plt.figure()
    plt.plot(fpr[1], tpr[1], color='darkorange',
             lw=1, label='ROC curve (area = %0.3f)' % roc_auc[1])
    plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="lower right")
    plt.show()


y_true =  [0, 1, 0, 0, 1, 0]
y_proba = [0.1, 0.4, 0.8, 0.05, 0.9, 0.4]
plotROC(y_true, y_proba)
# AUCROC = 0.812

# Change probability without changing the order.
y_true =  [0, 1, 0, 0, 1, 0]
y_proba = [0.3, 0.49, 0.5, 0.01, 0.99, 0.49]
plotROC(y_true, y_proba)
# AUCROC = 0.812
```

gistfile2019112811.py hosted with ♡ by GitHub view raw



ROC curve and AUCROC visualized by code

Photo by Kelly Sikkema on Unsplash

. . .

Conclusion

There are many possible choices in evaluation metrics, inside and outside of Kaggle. Each evaluation metrics has their properties and understanding their behavior is important when we optimize them or when we choose the right evaluation metric.

In another post(s), I will talk about the evaluation metrics we use in multi-class classification and recommendation, and we will also find tips to optimize the evaluation metrics while letting the models tuned by minimizing the loss function.

Machine Learning Python Modeling Kaggle Data Science