COMP/EECE 7/8745 Machine Learning

Topics:

Performance metrics

- Introduction
- Regression metrics
- Metrices for classification tasks
- others

Md Zahangir Alom
Department of Computer Science
University of Memphis, TN

Introduction

- Performance metrics are a part of every machine learning pipeline. They tell you if you're making progress, and put a number on it.
- All machine learning models, whether it's linear regression, or a SOTA technique like BERT, need a metric to judge performance.
- **Metrics are different from loss functions**. Loss functions show a measure of model performance. They're used to train a machine learning model (using some kind of optimization like Gradient Descent), and they're usually differentiable in the model's parameters.
- **Metrics are used to monitor and measure the performance of a model** (during validation and testing), and don't need to be differentiable.

https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide

Algorithm preference

- Criteria (Application-dependent):
  - Misclassification error, or risk (loss functions)
  - Training time/space complexity
  - Testing time/space complexity
  - Interpretability
  - Easy programmability
- Cost-sensitive learning

Resampling and K-Fold Cross-Validation

- The need for multiple training/validation sets
  $\{\mathcal{X}_i, \mathcal{V}_i\}_i$: Training/validation sets of fold i
- K-fold cross-validation: Divide $\mathcal{X}$ into k, $\mathcal{X}_i$,i=1,...,K

$$\mathcal{V}_1 = \mathcal{X}_1 \quad \mathcal{T}_1 = \mathcal{X}_2 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K$$
$$\mathcal{V}_2 = \mathcal{X}_2 \quad \mathcal{T}_2 = \mathcal{X}_1 \cup \mathcal{X}_3 \cup \cdots \cup \mathcal{X}_K$$

- $\mathcal{T}_i$ share K-2 parts

$$\mathcal{V}_K = \mathcal{X}_K \quad \mathcal{T}_K = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \cdots \cup \mathcal{X}_{K-1}$$

5×2 Cross-Validation

- 5 times 2 fold cross-validation (Dietterich, 1998)

$$\mathcal{T}_1 = \mathcal{X}_1^{(1)} \quad \mathcal{V}_1 = \mathcal{X}_1^{(2)}$$
$$\mathcal{T}_2 = \mathcal{X}_1^{(2)} \quad \mathcal{V}_2 = \mathcal{X}_1^{(1)}$$
$$\mathcal{T}_3 = \mathcal{X}_2^{(1)} \quad \mathcal{V}_3 = \mathcal{X}_2^{(2)}$$
$$\mathcal{T}_4 = \mathcal{X}_2^{(2)} \quad \mathcal{V}_4 = \mathcal{X}_2^{(1)}$$
$$\vdots$$
$$\mathcal{T}_9 = \mathcal{X}_5^{(1)} \quad \mathcal{V}_9 = \mathcal{X}_5^{(2)}$$
$$\mathcal{T}_{10} = \mathcal{X}_5^{(2)} \quad \mathcal{V}_{10} = \mathcal{X}_5^{(1)}$$

Bootstrapping

- Draw instances from a dataset *with replacement*
- Probability that we do not pick an instance after N draws

$$\left(1-\frac{1}{N}\right)^{N} \approx e^{-1} = 0.368$$

that is, only 36.8% is new!

Regression metrics

- Regression models have **continuous output**. So, we need a metric based on calculating **some sort of distance between *predicted* and *ground truth***.

- In order to evaluate Regression models, we'll discuss these metrics in detail:
    - Mean Squared Error (MSE),
    - Mean Absolute Error (MAE),
    - Root Mean Squared Error (RMSE),
    - R^2 (R-Squared).

https://www.kaggle.com/datasets/vikrishnan/boston-house-prices

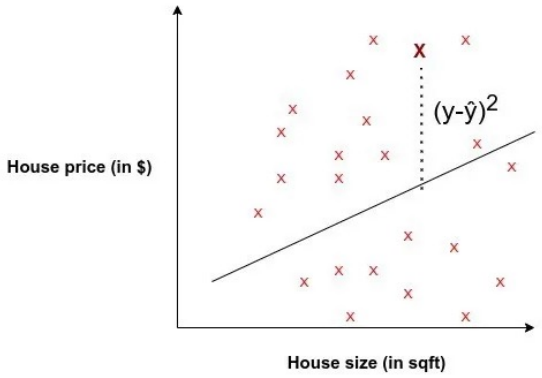
Mean Squared Error (MSE)

- Mean squared error is perhaps the **most popular metric** used for regression problems.
- It essentially finds the average of the squared difference between the target value and the value predicted by the regression model.

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(y_j - \check{y}_j)^2$$

Where:

- y_j: ground-truth value
- y_hat: predicted value from the regression model
- N: number of datums




```
mse = (y-y_hat)**2
print(f"MSE: {mse.mean():0.2f} (+/- {mse.std():0.2f})")
```

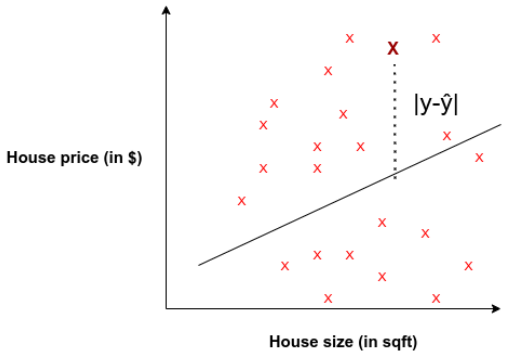
Mean Absolute Error (MAE)

- **Mean Absolute Error is the average** of the difference between the ground truth and the predicted values. Mathematically, its represented as :

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|y_j - \check{y}_j|$$

Where:

- y_j: ground-truth value
- y_hat: predicted value from the regression model
- N: number of datums




```
mae = np.abs(y-y_hat)
print(f"MAE: {mae.mean():0.2f} (+/- {mae.std():0.2f})")
```

Root Mean Squared Error (RMSE)

- Root Mean Squared Error corresponds to the square root of the average of the squared difference between the target value and the value predicted by the regression model. Basically, sqrt(MSE). Mathematically it can be represented as

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(y_j - \check{y}_j)^2}$$

Few key points related to RMSE:

- It retains the differentiable property of MSE.
- It handles the penalization of smaller errors done by MSE by square rooting it.
- Error interpretation can be done smoothly, since the scale is now the same as the random variable.
- Since scale factors are essentially normalized, it's less prone to struggle in the case of outliers.

```
mse = (y-y_hat)**2
rmse = np.sqrt(mse.mean())
print(f"RMSE: {rmse:0.2f}")
```

R² Coefficient of determination

- R² Coefficient of determination actually **works as a post metric**, meaning it's a metric that's calculated using other metrics.

$$SE(\bar{Y}) = \sum_{i=1}^{N} (y_i - \bar{y})^2$$

Percentage of variation described the regression line:

$$\frac{SE(line)}{SE(\bar{Y})}$$

Subsequently, the percentage of variation described the regression line:

$$1 - \frac{SE(line)}{SE(\bar{Y})}$$

Finally, we have our formula for the coefficient of determination, which can tell us how good or bad the fit of the regression line is:

$$coeff(R^2) = 1 - \frac{SE(line)}{SE(\bar{Y})}$$

- The point of even calculating this coefficient is to answer the question **"How much (what %) of the total variation in Y(target) is explained by the variation in X(regression line)"**
- This is calculated using the sum of squared errors. Total variation in Y (Variance of Y):

```
# R^2 coefficient of determination
SE_line = sum((y-y_hat)**2)
SE_mean = sum((y-y.mean())**2)
r2 = 1-(SE_line/SE_mean)
print(f"R^2 coefficient of determination: {r2*100:0.2f}%")
```

Adjusted R^2

- The **Vanilla R^2 method suffers from some demons**, **like misleading the researcher** into believing that the model is improving when the score is increasing but in reality, the learning is not happening.

$$R_a^2 = 1 - [(\frac{n-1}{n-k-1}).(1-R^2)]$$

Where:

- n = number of observations
- k = number of independent variables
- Ra² = adjusted R²

- This can happen when a model overfits the data, in that case the variance explained will be 100% but the learning hasn't happened.
- To rectify this, R^2 is adjusted with the number of independent variables.
- Adjusted R^2 is always lower than R^2, as it adjusts for the increasing predictors and only shows improvement if there is a real improvement.

Classification metrics

- Classification problems are **one of the world's most widely researched areas**. Use cases are present in almost all production and industrial environments. Speech recognition, face recognition, text classification – the list is endless
- *Classification Metrics* evaluate a model's performance and **tell you how good or bad the classification is**, however, each of them evaluates it in a different way.
- We generally use the following metrics :
  - Accuracy
  - Confusion Matrix (not a metric but fundamental to others)
  - Precision and Recall
  - F1-score
  - AU-ROC

Accuracy

- Classification accuracy is perhaps the simplest metric to use and implement and is defined as the **number of correct predictions divided by the total number of predictions,** multiplied by 100**.**

- **Ref. codes:**

```
from sklearn.metrics import accuracy_score
print('Accuracy Score is ‘, {accuracy_score(y_test,y_hat)})
```

Confusion Matrix

- Confusion Matrix is a tabular visualization of the **ground-truth labels versus model predictions**. Each row of the confusion matrix represents the instances in a predicted class and each column represents the instances in an actual class.

		Predicted	
		Has Cancer	Doesn't Have Cancer
Ground Truth	Has Cancer	TP	FP
	Doesn't Have Cancer	FN	TN

Confusion Matrix for H^0

- Confusion matrix is **not exactly a performance metric but sort of a basis on which other metrics evaluate the results**.

Confusion Matrix

- **True Positive(TP)** : signifies how many positive class samples your model predicted correctly.

- **True Negative(TN)** : signifies how many negative class samples your model predicted correctly.

- **False Positive(FP) :** you predicted Yes, but they were labeled as NO (This factor represents **Type-I error** in statistical nomenclature). This error positioning in the confusion matrix depends on the choice of the null hypothesis.

- **False Negative(FN)** : You predicted NO, but they were labeled as Yes (This factor represents **Type-II** error in statistical nomenclature). This error positioning in the confusion matrix also depends on the choice of the null hypothesis.

Confusion Matrix

Ref. codes:

```
def find_TP(y, y_hat): # counts the number of true positives (y = 1, y_hat = 1)
    return sum((y == 1) & (y_hat == 1))

def find_FP(y, y_hat): # counts the number of false positives (y = 0, y_hat = 1) Type-I error
    return sum((y == 0) & (y_hat == 1))

def find_FN(y, y_hat): # counts the number of false negatives (y = 1, y_hat = 0) Type-II error
    return sum((y == 1) & (y_hat == 0))

def find_TN(y, y_hat): # counts the number of true negatives (y = 0, y_hat = 0)
    return sum((y == 0) & (y_hat == 0))
```

Precision

$$P = \frac{TP}{TP+FP} = \frac{\text{Cancer patients correctly identified}}{\text{Cancer patients correctly identified+incorrectly labelled cancer patients as non-cancerous}}$$

- Precision is the ratio of true positives and total positives predicted: $0<P<1$
- The precision metric focuses on **Type-I errors**(FP). A **Type-I error occurs when we reject a true null Hypothesis(H^0)**. So, in this case, Type-I error is incorrectly labeling cancer patients as non-cancerous.
- A precision score towards **1 will signify that your model didn't miss any true positives**, and is able to classify well between correct and incorrect labeling of cancer patients.
- What it cannot measure is the existence of Type-II error, which is false negatives – cases **when a non-cancerous patient is identified as cancerous.**
- A low precision score (<0.5) means your classifier has **a high number of false positives** which can be an outcome of imbalanced class or untuned model hyperparameters. In an imbalanced class problem, you have to prepare your data beforehand with **over/under-sampling or focal loss** in order to curb FP/FN.

Precision

```
TP = find_TP(y, y_hat)
FN = find_FN(y, y_hat)
FP = find_FP(y, y_hat)
TN = find_TN(y, y_hat)

print('TP:',TP)
print('FN:',FN)
print('FP:',FP)
print('TN:',TN)
precision = TP/(TP+FP)
print('Precision:', precision)
```

```
TP: 119
FN: 120
FP: 0
TN: 443
```

Precision : 1.0

```
TP: 239
FN: 0
FP: 443
TN: 0
```

Precision : 0.350

Recall/Sensitivity/Hit-Rate

- A **Recall** is essentially the ratio of true positives to all the positives in ground truth 0<R<1
- The recall metric focuses on **type-II errors**(FN). A type-II error occurs when we **accept a false null hypothesis(H^0)**. So, in this case, type-II error is incorrectly labeling non-cancerous patients as cancerous.
- Recall towards 1 will signify that your model **didn't miss any true positives**, and is able to classify well between correctly and incorrectly labeling of cancer patients.
- What it cannot measure is the existence of type-I error which is false positives i.e the cases when a **cancerous patient is identified as non-cancerous.**
- A low recall score (<0.5) means your classifier has **a high number of false negatives** which can be an outcome of imbalanced class or untuned model hyperparameters. In an imbalanced class problem, you have to prepare your data beforehand with over/under-sampling or focal loss in order to curb FP/FN.

$$R = \frac{TP}{TP+FN} = \frac{\text{Cancer patients correctly identified}}{\text{Cancer patients correctly identified+incorrectly labelled non-cancer patients as cancerous}}$$

Recall

```
TP = find_TP(y, y_hat)
FN = find_FN(y, y_hat)
FP = find_FP(y, y_hat)
TN = find_TN(y, y_hat)
print('TP:',TP)
print('FN:',FN)
print('FP:',FP)
print('TN:',TN)
precision = TP/(TP+FP)
print('Precision:',precision)
```

```
TP: 119
FN: 120
FP: 0
TN: 443
```

Recall : 0.497

```
TP: 239
FN: 0
FP: 443
TN: 0
```

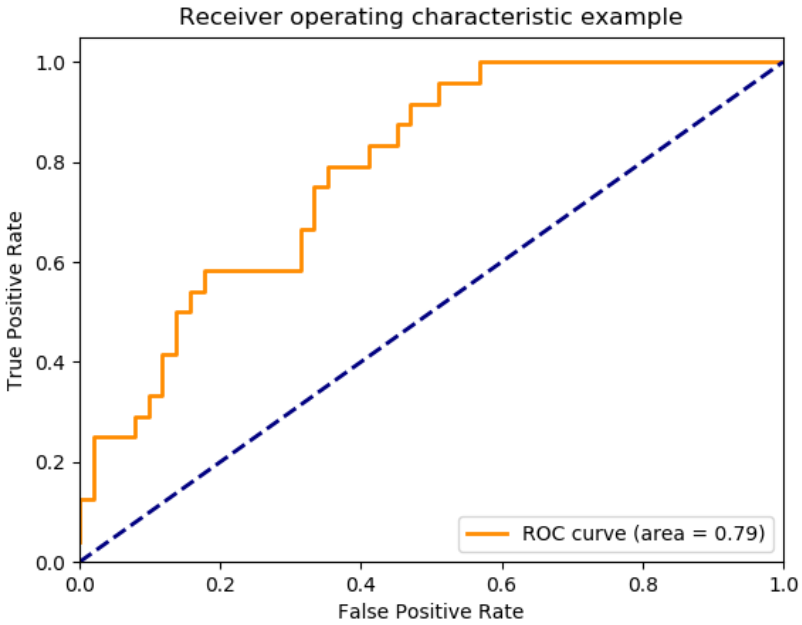
Recall : 1.0

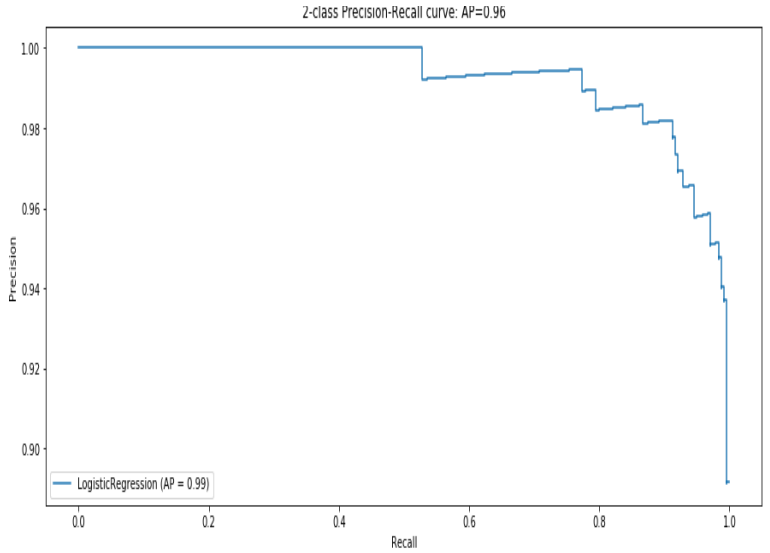
Precision-Recall tradeoff

- To improve your model, you can **either improve precision or recall – but not both!**
  - If you try to reduce cases of non-cancerous patients being labeled as cancerous (FN/type-II),
  - no direct effect will take place on cancerous patients being labeled as non-cancerous.

- Ref. Codes:

```
from sklearn.metrics import plot_precision_recall_curve
disp = plot_precision_recall_curve(clf, X, y)
disp.ax_.set_title('2-class Precision-Recall curve: '
'AP={0:0.2f}'.format(precision))
```





F1-score

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

- The F1-score metric uses a combination of precision and recall. In fact, the **F1 score is the harmonic mean of the two.** The formula of the two essentially is:
- A high F1 score **symbolizes a high precision as well as high recall.** It presents a good balance between precision and recall and gives good results on **imbalanced classification problems**.
- A low F1 score tells you (almost) nothing—it only tells you about performance at a threshold (low F1 doesn't say which cases : With low F1, it's unclear what the problem is (low precision or low recall?), and **whether the model suffers from type-I or type-II error**).
- High F1 means we likely have high precision and recall on a large portion of the decision (which is informative).
- F1 is **a widely used and considered a fine metric** to converge onto a decision

- Ref. codes :

```
f1_score = 2*((precision*recall)/(precision+recall))
print('F1 score: %f' % f1_score)
```

Summary

- One really important thing to note is that you can adjust these metrics to cater to your specific use case.
  - For example, take a **weighted F1-score :** calculates metrics for each label and finds their average weight by support (the **number of true instances for each label**).
  - Another example could be a weighted accuracy, or in technical terms **Balanced Accuracy :** in both binary and multiclass classification problems is used **to deal with imbalanced datasets.** It's defined as the average recall obtained in each class.

- Sensitivity= Recall= TP/(TP+FN)
- Specificity= True Negative Rate= TN/(TN+FP)

Summary

- We discussed the metrics based on the **ML model/application they are mostly used for**
- Other the popular metrics used in the following problems:
    - Statistical Metrics (Correlation)
    - Computer Vision Metrics (PSNR, SSIM, IoU)
    - NLP Metrics (Perplexity, BLEU score)
    - Deep Learning Related Metrics (Inception score, Frechet Inception distance)
- What's next?
  - Linear and logistic regression methods