Model Evaluation and Metrics in Machine Learning

Prof.Dr. Bahadır AKTUĞ
Machine Learning with Python

**Compiled from sources given in the references.*

Evaluation and Metrics

- Sklearn provides 3 different APIs for evaluating the quality of a model's predictions.
- Estimator score method:
  - Estimators have a score method providing a default evaluation criterion for the problem they are designed to solve.
- Scoring parameter:
  - Model-evaluation tools using cross-validation (such as model_selection.cross_val_score and model_selection.GridSearchCV) rely on an internal scoring strategy.
- Metric functions:
  - The sklearn.metrics module implements functions assessing prediction error for specific purposes. These metrics can be grouped as Classification metrics, Multilabel ranking metrics, Regression metrics and Clustering metrics.

Accuracy

- The accuracy_score function computes the accuracy, either the fraction (default) or the count (normalize=False) of correct predictions.
- In multilabel classification, the function returns the subset accuracy. If the entire set of predicted labels for a sample strictly match with the true set of labels, then the subset accuracy is 1.0, otherwise it is 0.0.

If $\hat{y}_i$ is the predicted value of the i-th sample and y_i is the corresponding true value, then the fraction of correct predictions over n_{samples} is defined as

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

where $1(x)$ is the indicator function.

Accuracy

- Sklearn.metrics module has accuracy_score function to compute accuracy from true and predicted values.

```
>>>
import numpy as np
from sklearn.metrics import accuracy_score
y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2, 3]
accuracy_score(y_true, y_pred)

accuracy_score(y_true, y_pred, normalize=False)
```

Precision

- The precision is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative. The best value is 1 and the worst value is 0.
- Sklearn.metrics module has precision_score function to compute accuracy from true and predicted values.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

```
from sklearn.metrics import precision_score
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
precision_score(y_true, y_pred, average='macro')
```

		Predicted: Negative	Predicted: Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Recall

- The recall is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples. The best value is 1 and the worst value is 0.
- Sklearn.metrics module has recall_score function to compute accuracy from true and predicted values.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

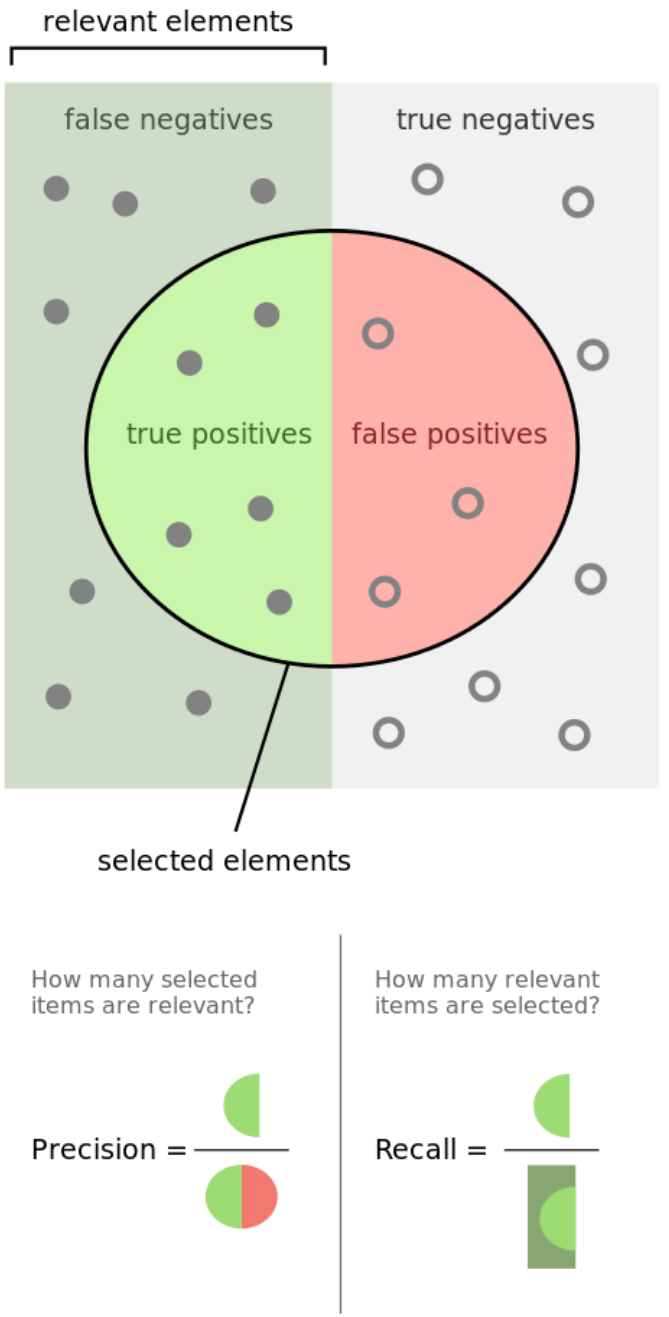
```
from sklearn.metrics import recall_score
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
recall_score(y_true, y_pred, average='macro')
```

		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

F1 Score

- The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.
- The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

F1 = 2 * (precision * recall) / (precision + recall)

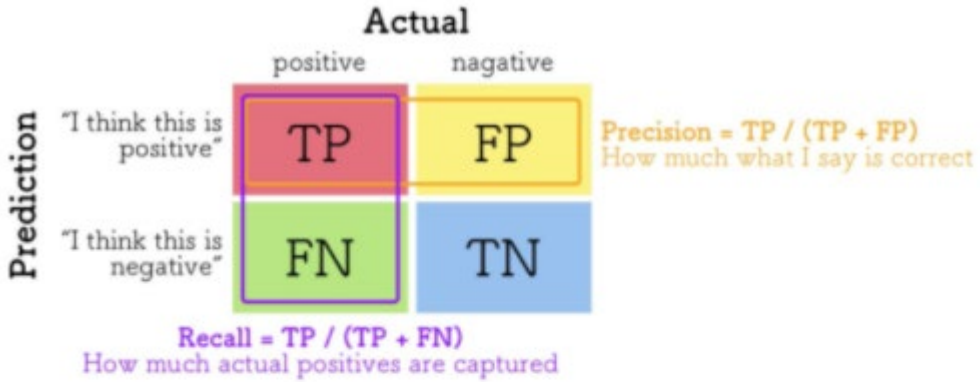


F1 Score

- There are several variants of F1-score.
- Sklearn.metrics module has f1_score function to compute accuracy from true and predicted values.

```
from sklearn.metrics import f1_score
y_true = [0, 1, 2, 0, 1, 2]
y_pred = [0, 2, 1, 0, 0, 1]
f1_score(y_true, y_pred, average='macro')

f1_score(y_true, y_pred, average='micro')

f1_score(y_true, y_pred, average='weighted')

f1_score(y_true, y_pred, average=None)

y_true = [0, 0, 0, 0, 0, 0]
y_pred = [0, 0, 0, 0, 0, 0]
f1_score(y_true, y_pred, zero_division=1)
```

Summary of Fundamental Metrics



$$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

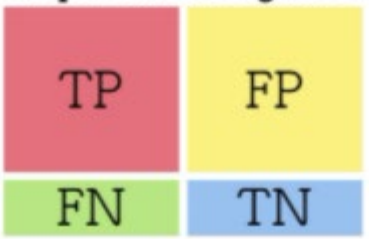
$$\text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

$$\text{accuracy} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{tn} + \text{fp} + \text{fn}}$$

$$\text{F}_1 \text{ score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

High precision, low recall

Low precision, high recall



Classification Metrics

- Classification problems are perhaps the most common type of machine learning problem and as such there are a myriad of metrics that can be used to evaluate predictions for these problems.
- Some metrics for classification problems:
  - Classification Accuracy
  - Log Loss
  - Area Under ROC Curve
  - Confusion Matrix
  - Classification Report

Classification Accuracy

- Classification accuracy is the number of correct predictions made as a ratio of all predictions made.
- This is the most common evaluation metric for classification problems, it is also the most misused. It is really only suitable when there are an equal number of observations in each class (which is rarely the case) and that all predictions and prediction errors are equally important, which is often not the case.
- Below is an example of calculating classification accuracy.
  - from sklearn import model_selection
  - kfold=model_selection.KFold(n_splits=10,random_state=7, shuffle=True)
  - model = LogisticRegression(solver='liblinear')
  - results=model_selection.cross_val_score(model,X,Y,cv=kfold,scoring='accuracy')
  - print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))

Log Loss

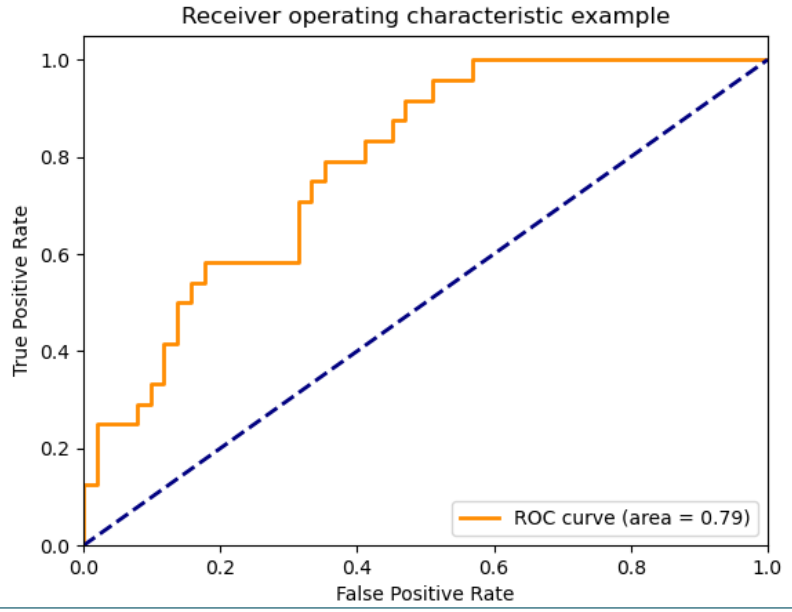
- Logistic loss (or log loss) is a performance metric for evaluating the predictions of probabilities of membership to a given class.
- The scalar probability between 0 and 1 can be seen as a measure of confidence for a prediction by an algorithm.
- Predictions that are correct or incorrect are rewarded or punished proportionally to the confidence of the prediction.
- Below is an example of calculating classification accuracy.
  - from sklearn import model_selection
  - kfold=model_selection.KFold(n_splits=10,random_state=7, shuffle=True)
  - model = LogisticRegression(solver='liblinear')
  - results=model_selection.cross_val_score(model,X,Y,cv=kfold,scoring='neg_log_loss')
  - print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))

Area Under ROC Curve

- Area Under ROC Curve (or ROC AUC for short) is a performance metric for binary classification problems.
- The AUC represents a model's ability to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predictions perfectly. An area of 0.5 represents a model as good as random.
- A ROC Curve is a plot of the true positive rate and the false positive rate for a given set of probability predictions at different thresholds used to map the probabilities to class labels.
- The area under the curve is then the approximate integral under the ROC Curve. Below is an example of calculating classification accuracy.
- Below is an example of calculating classification accuracy.
  - from sklearn import model_selection
  - kfold=model_selection.KFold(n_splits=10,random_state=7, shuffle=True)
  - model = LogisticRegression(solver='liblinear')
  - results=model_selection.cross_val_score(model,X,Y,cv=kfold,scoring='roc_auc')
  - print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))

Receiver Operating Curve (ROC)

- A ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied.
- It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings.
- TPR is also known as sensitivity, and FPR is one minus the specificity or true negative rate."




```
import numpy as np
from sklearn.metrics import roc_curve
y = np.array([1, 1, 2, 2])
scores = np.array([0.1, 0.4, 0.35, 0.8])
fpr, tpr, thresholds = roc_curve(y, scores,
pos_label=2)
```

Receiver Operating Curve (ROC)

- A receiver operating characteristic (ROC), or simply ROC curve, is a graphical plot which illustrates the performance of a binary classifier system as its discrimination threshold is varied.
- It is created by plotting the fraction of true positives out of the positives (TPR = true positive rate) vs. the fraction of false positives out of the negatives (FPR = false positive rate), at various threshold settings.
- PR is also known as sensitivity, and FPR is one minus the specificity or true negative rate."
- This function requires the true binary value and the target scores, which can either be probability estimates of the positive class, confidence values, or binary decisions.
- Here is a small example of how to use the roc_curve function:
- Classification Accuracy
  - Log Loss
  - Area Under ROC Curve
  - Confusion Matrix
  - Classification Report

Confusion Matrix

- The confusion matrix is a handy presentation of the accuracy of a model with two or more classes.
- The table presents predictions on the x-axis and accuracy outcomes on the y-axis. The cells of the table are the number of predictions made by a machine learning algorithm.
- For example, a machine learning algorithm can predict 0 or 1 and each prediction may actually have been a 0 or 1. Predictions for 0 that were actually 0 appear in the cell for prediction=0 and actual=0, whereas predictions for 0 that were actually 1 appear in the cell for prediction = 0 and actual=1. And so on.

		Predicted value	
		Positive	Negative
Actual value	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

Confusion Matrix

- Below is an example of calculating classification accuracy.
  - from sklearn import model_selection
  - from sklearn.metrics import confusion_matrix
  - kfold=model_selection.KFold(n_splits=10,random_state=7, shuffle=True)
  - model = LogisticRegression(solver='liblinear')
  - matrix = confusion_matrix(Y_test, predicted)
  - print(matrix)

- Sklearn.metrics module has confusion_matrix and plot_confusion_matrix functions to compute accuracy from true and predicted values.

```
>>> from sklearn.metrics import confusion_matrix
>>> y_true = [2, 0, 2, 2, 0, 1]
>>> y_pred = [0, 0, 2, 2, 0, 2]
>>> confusion_matrix(y_true, y_pred)
```

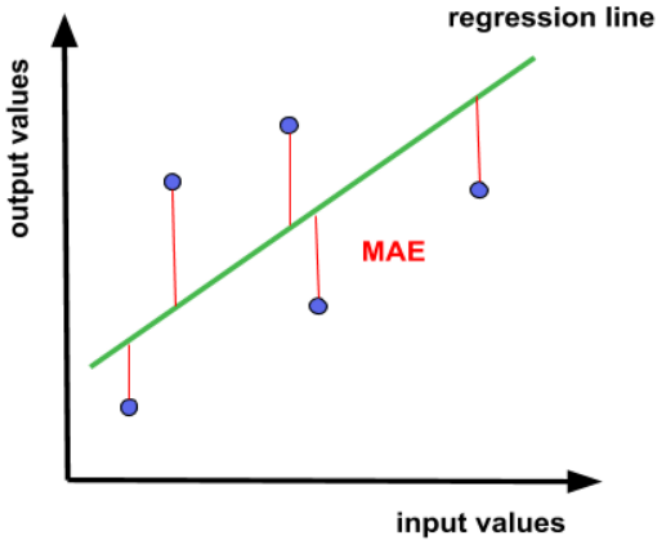
Regression Metrics

- The metrics used in regression problems is are different than those of classification.
- We always need to make sure that the evaluation metric we choose for a regression problem does penalize errors in a way that reflects the consequences of those errors for the business, organizational, or user needs of our application.
- Some metrics for classification problems:
  - Mean Absolute Error.
  - Mean Squared Error.
  - R^2
- If there are outliers in the data, they can have an unwanted influence on the overall R^2 or MSE scores.
- MAE is robust to the presence of outliers because it uses the absolute value. Hence, we can use the MAE score if ignoring outliers is important to us.
- MAE is the best metrics when we want to make a distinction between different models because it doesn't reflect large residuals.
- If we want to ensure that our model takes the outliers into account more, we should use the MSE metrics.

Mean Absolute Error (MAE)

- The Mean Absolute Error (or MAE) is the average of the absolute differences between predictions and actual values. It gives an idea of how wrong the predictions were.
- The measure gives an idea of the magnitude of the error, but no idea of the direction (e.g. over or under predicting).

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



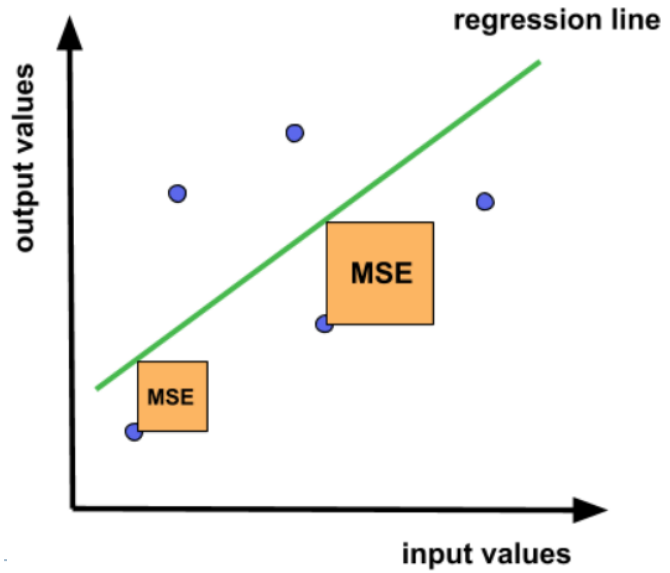
Mean Absolute Error (MAE)

- Below is an example of calculating Mean Absolute Error:
  - from sklearn import model_selection
  - kfold=model_selection.KFold(n_splits=10,random_state=7, shuffle=True)
  - model = LinearRegression()
  - results=model_selection.cross_val_score(model,X,Y,cv=kfold,scoring='neg_mean_absolute_error')
  - print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))

Mean Squared Error (MSE)

- The Mean Squared Error (or MSE) is much like the mean absolute error in that it provides a gross idea of the magnitude of error.
- Taking the square root of the mean squared error converts the units back to the original units of the output variable and can be meaningful for description and presentation. This is called the Root Mean Squared Error (or RMSE).

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

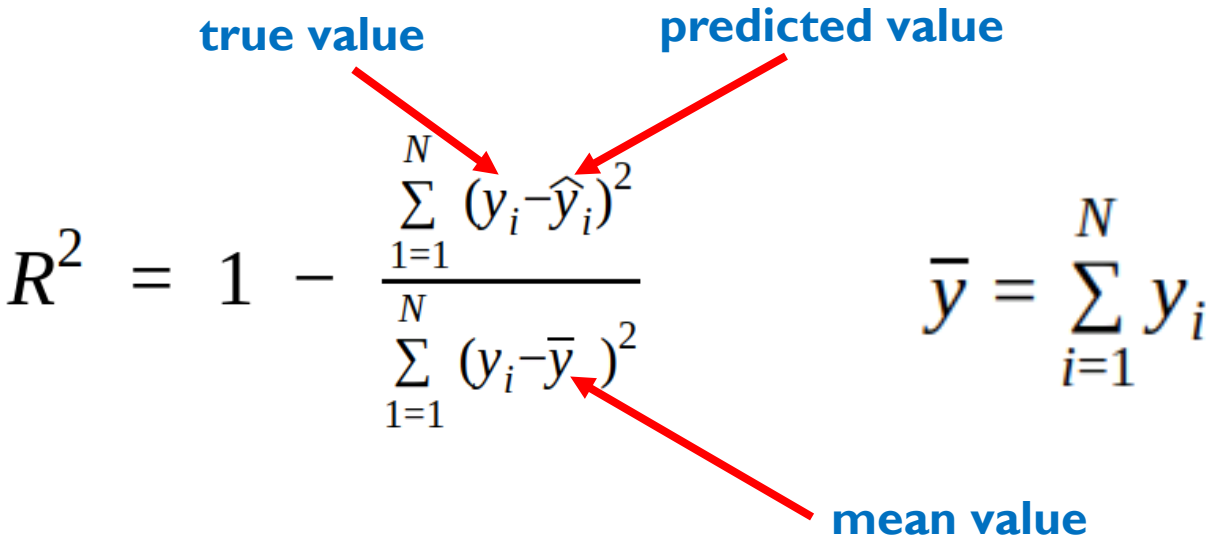


Mean Squared Error (MSE)

- Below is an example of calculating Mean Squared Error:
  - from sklearn import model_selection
  - kfold=model_selection.KFold(n_splits=10,random_state=7, shuffle=True)
  - model = LinearRegression()
  - results=model_selection.cross_val_score(model,X,Y,cv=kfold,scoring='neg_mean_squared_error'')
  - print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))

R^2 score (the coefficient of determination)

- The R^2 (or R Squared) metric provides an indication of the goodness of fit of a set of predictions to the actual values. In statistical literature, this measure is called the coefficient of determination.
- This is a value between 0 and 1 for no-fit and perfect fit respectively.



R^2 score (the coefficient of determination)

Below is an example of calculating R^2 (or R Squared) metric:

- from sklearn import model_selection
- kfold=model_selection.KFold(n_splits=10,random_state=7, shuffle=True)
- model = LinearRegression()
- results=model_selection.cross_val_score(model,X,Y,cv=kfold,scoring='r2')
- print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))

References

1 https://scikit-learn.org/
2 https://towardsdatascience.com/
3 McKinney, W. (2017). Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython 2nd Edition.
4 Albon, C. (2018). Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning
5 Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 1st Edition
6 Müller, A. C., Guido, S. (2016). Introduction to Machine Learning with Python: A Guide for Data Scientists
7 Burkov, A. (2019). The Hundred-Page Machine Learning Book.
8 Burkov, A. (2020). Machine Learning Engineering.
9 https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/
10 https://towardsdatascience.com
11 https://docs.python.org/3/tutorial/
12 https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d
13 https://developers.google.com/edu/python/
14 http://learnpythonthehardway.org/book/