Accuracy and Metrics

AcadView

June 5, 2018

---

1. Overview

After doing the usual Feature Engineering, Selection, and of course, implementing a model and getting some output in forms of a probability or a class, the next step is to find out how effective is the model based on some metric using test datasets. Different performance metrics are used to evaluate different Machine Learning Algorithms. For now, we will be focusing on the ones used for Classification problems. We can use classification performance metrics such as Log-Loss, Accuracy, AUC(Area under Curve) etc. Another example of metric for evaluation of machine learning algorithms is precision, recall, which can be used for sorting algorithms primarily used by search engines.

The metrics that you choose to evaluate your machine learning algorithms are very important.

Choice of metrics influences how the performance of machine learning algorithms is measured and compared. They influence how you weight the importance of different characteristics in the results and your ultimate choice of which algorithm to choose.

2. How we demonstrate the accuracy and metrics in this article

Metrics are demonstrated for both classification and regression type machine learning problems.

• For classification metrics, the Pima Indians onset of diabetes dataset is used as demonstration. This is a binary classification problem where all of the input variables are

numeric (download from the link below).
https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv

• For regression metrics, the Boston House Price dataset is used as demonstration. this is a regression problem where all of the input variables are also numeric (update: download data from the link below).
https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.data In each recipe, the dataset is downloaded directly from the UCI Machine Learning repository.

A caveat in these recipes is the cross val score function used to report the performance in each recipe.It does allow the use of different scoring metrics that will be discussed, but all scores are reported so that they can be sorted in ascending order (largest score is best).

Some evaluation metrics (like mean squared error) are naturally descending scores (the smallest score is best) and as such are reported as negative by the cross val score() function. This is important to note, because some scores will be reported as negative that by definition can never be negative.

3. 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.

In this section we will review how to use the following metrics:

• Classification Accuracy.

• Logarithmic Loss.

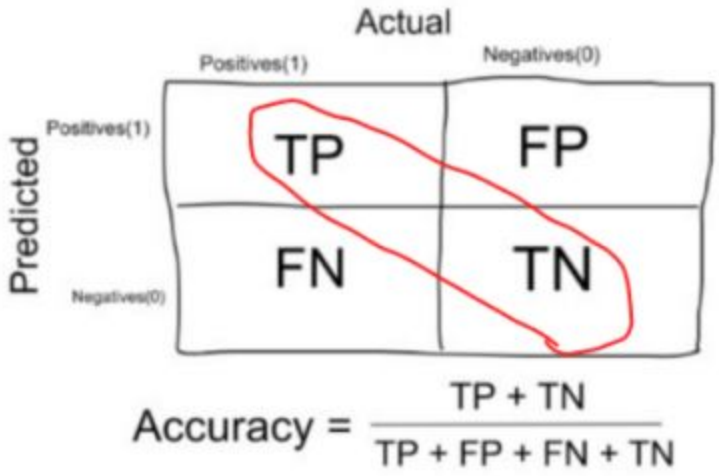
• Area Under ROC Curve.

• Confusion Matrix.

• Classification Report.

3.1. 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.

```
# Cross Validation Classification Accuracy
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-d
iabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LogisticRegression()
scoring = 'accuracy'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Accuracy: %.3f (%.3f)") % (results.mean(), results.std())
```

You can see that the ratio is reported. This can be converted into a percentage by multiplying the value by 100, giving an accuracy score of approximately 77% accurate.

When to use Accuracy: Accuracy is a good measure when the target variable classes in the data are nearly balanced. Ex:60% classes in our fruits images data are apple and 40% are

oranges. A model which predicts whether a new image is Apple or an Orange, 97% of times correctly is a very good measure in this example.

When NOT to use Accuracy: Accuracy should NEVER be used as a measure when the target variable classes in the data are a majority of one class.

Important Example: In our cancer detection example with 100 people, only 5 people has cancer. Let's say our model is very bad and predicts every case as No Cancer. In do- ing so, it has classified those 95 non-cancer patients correctly and 5 cancerous patients as Non-cancerous. Now even though the model is terrible at predicting cancer, The accuracy of such a bad model is also 95%.

3.2 Logarithmic Loss

Logarithmic loss (or logloss) 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 logloss for Logistic regression predictions on the Pima Indians onset of diabetes dataset.

```
# Cross Validation Classification LogLoss
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-d
iabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LogisticRegression()
scoring = 'neg_log_loss'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Logloss: %.3f (%.3f)") % (results.mean(), results.std())
```

Smaller logloss is better with 0 representing a perfect logloss. As mentioned above, the measure is inverted to be ascending when using the cross val score() function.

Logloss: -0.493

3.3. Area Under ROC Curve

Area under ROC Curve (or AUC for short) is a performance metric for binary classification problems.

The AUC represents a models 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.

ROC can be broken down into **sensitivity and specificity.** A binary classification problem is really a trade-off between sensitivity and specificity.

Sensitivity is the true positive rate also called the recall. It is the number instances from the positive (first) class that actually predicted correctly. Specificity is also called the true negative rate. Is the number of instances from the negative class (second) class that were actually predicted correctly.

You can learn more about ROC on the Wikipedia page.(link given below)
https://en.wikipedia.org/wiki/Receiver_operating_characteristic

The example below provides a demonstration of calculating AUC.

```
# Cross Validation Classification ROC AUC
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabet
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LogisticRegression()
scoring = 'roc_auc'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("AUC: %.3f (%.3f)") % (results.mean(), results.std())
```

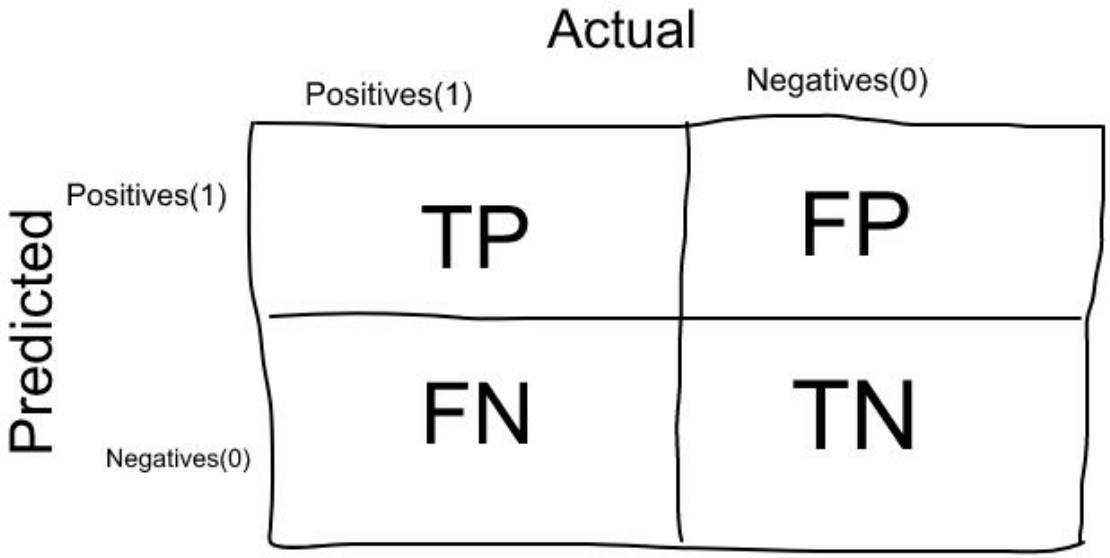
You can see the the AUC is relatively close to 1 and greater than 0.5, suggesting some skill in the predictions.

```
AUC: 0.824
```

3.4. 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.



Confusion Matrix

3.4.1 Terms associated with Confusion matrix:

• **True Positives (TP):** True positives are the cases when the actual class of the data point was 1(True) and the predicted is also 1(True) Ex: The case where a person is actually having cancer(1) and the model classifying his case as cancer(1) comes under True positive.

• **True Negatives (TN):** True negatives are the cases when the actual class of the data point was 0(False) and the predicted is also 0(False).

Ex: The case where a person NOT having cancer and the model classifying his case as Not cancer comes under True Negatives.

• **False Positives (FP):** False positives are the cases when the actual class of the data point was 0(False) and the predicted is 1(True). False is because the model has predicted incorrectly and positive because the class predicted was a positive one. (1)

Ex: A person NOT having cancer and the model classifying his case as cancer comes under False Positives.

• **False Negatives (FN):** False negatives are the cases when the actual class of the data point was 1(True) and the predicted is 0(False). False is because the model has predicted incorrectly and negative because the class predicted was a negative one. (0)

Ex: A person having cancer and the model classifying his case as No-cancer comes under False Negatives.

The ideal scenario that we all want is that the model should give 0 False Positives and 0 False Negatives. But that's not the case in real life as any model will NOT be 100% accurate most of the times. The Confusion matrix in itself is not a performance measure as such, but almost all of the performance metrics are based on Confusion Matrix and the numbers inside it.

Below is an example of calculating a confusion matrix for a set of prediction by a model on a test set.

```
# Cross Validation Classification Confusion Matrix
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diab
etes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_size=
test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

Although the array is printed without headings, you can see that the majority of the predictions fall on the diagonal line of the matrix (which are correct predictions).

```
[[141  21]
 [ 41  51]]
```

3.5. Classification Report

Scikit-learn does provide a convenience report when working on classification problems to give you a quick idea of the accuracy of a model using a number of measures.

The classification report() function displays the precision, recall, f1-score and support

for each class.

The example below demonstrates the report on the binary classification problem.

```
# Cross Validation Classification Report
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-d
iabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, test_si
ze=test_size, random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print(report)
```

You can see good prediction and recall for the algorithm.

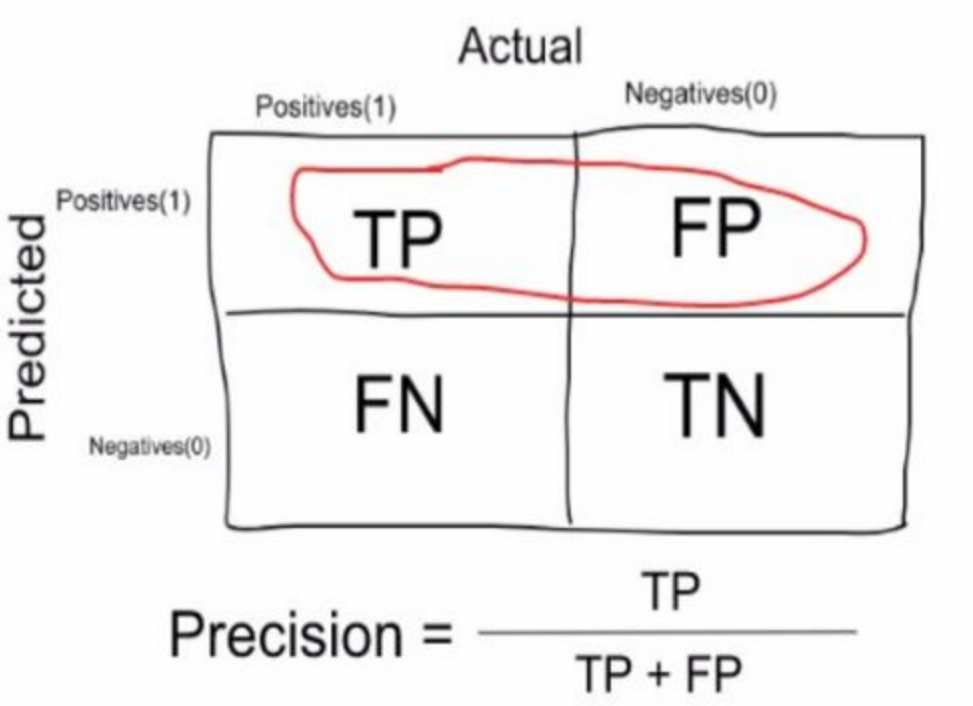
```
             precision    recall  f1-score   support

        0.0       0.77      0.87      0.82       162
        1.0       0.71      0.55      0.62        92

avg / total       0.75      0.76      0.75       254
```

3.5.1 Precision:

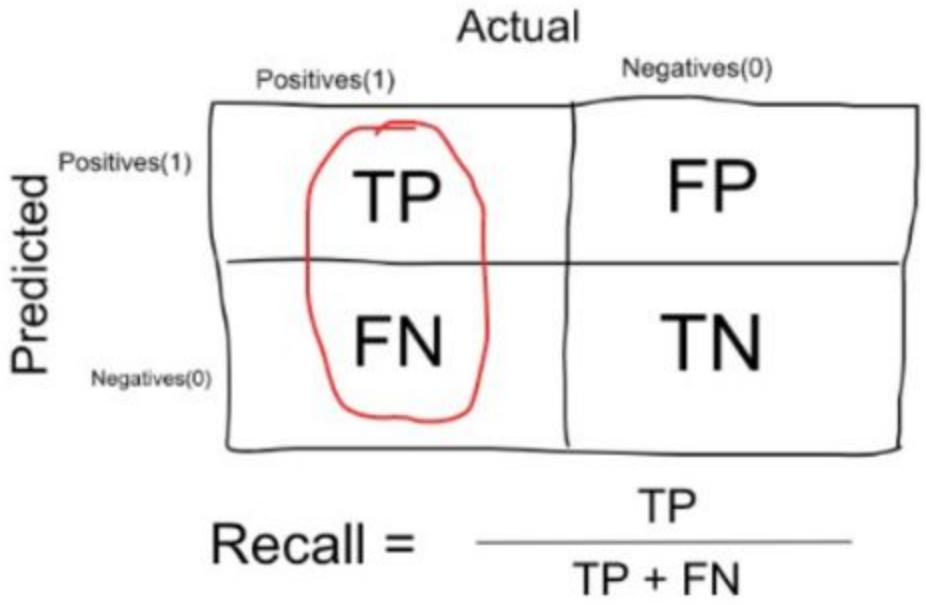
Precision is a measure that tells us what proportion of patients that we diagnosed as having cancer, actually had cancer. The predicted positives (People predicted as cancerous are TP and FP) and the people actually having a cancer are TP.



Ex: In our cancer example with 100 people, only 5 people have cancer. Let's say our model is very bad and predicts every case as Cancer. Since we are predicting everyone as having cancer, our denominator(True positives and False Positives) is 100 and the numerator, person having cancer and the model predicting his case as cancer is 5. So in this example, we can say that Precision of such model is 5%.

3.6. Recall or Sensitivity:

Recall is a measure that tells us what proportion of patients that actually had cancer was diagnosed by the algorithm as having cancer. The actual positives (People having cancer are TP and FN) and the people diagnosed by the model having a cancer are TP. (Note: FN is included because the Person actually had a cancer even though the model predicted otherwise).



Recall or Sensitivity

Ex: In our cancer example with 100 people, 5 people actually have cancer. Let's say that the model predicts every case as cancer.

So our denominator(True positives and False Negatives) is 5 and the numerator, person having cancer and the model predicting his case as cancer is also 5(Since we predicted 5 cancer cases correctly). So in this example, we can say that the Recall of such model is 100%. And Precision of such a model(As we saw above) is 5%.

3.6.1 When to use Precision and When to use Recall?

It is clear that recall gives us information about a classifiers performance with respect to false negatives (how many did we miss), while precision gives us information about its performance with respect to false positives(how many did we caught).

Precision is about being precise. So even if we managed to capture only one cancer case, and we captured it correctly, then we are 100% precise.

Recall is not so much about capturing cases correctly but more about capturing all cases that have cancer with the answer as cancer. So if we simply always say every case as cancer, we have 100% recall.

So basically if we want to focus more on minimising False Negatives, we would want our Recall to be as close to 100% as possible without precision being too bad and if we want to

focus on minimizing False positives, then our focus should be to make Precision as close to 100% as possible.

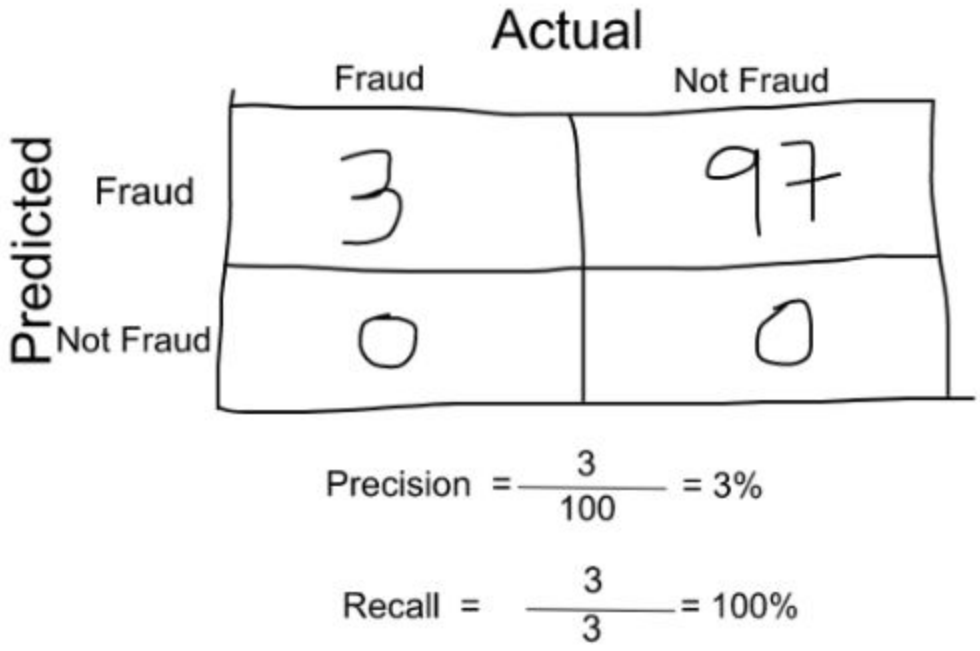
3.6.2. F1 Score

We don't really want to carry both Precision and Recall in our pockets every time we make a model for solving a classification problem. So it's best if we can get a single score that kind of represents both Precision(P) and Recall(R).

One way to do that is simply taking their arithmetic mean. i.e (P + R) / 2 where P is Precision and R is Recall. But thats pretty bad in some situations.

Suppose we have 100 credit card transactions, of which 97 are legit and 3 are fraud and let's say we came up a model that predicts everything as fraud.

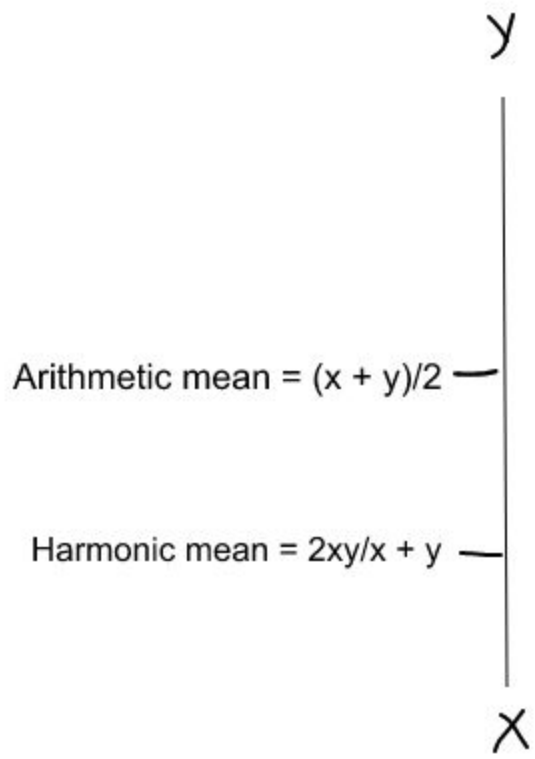
Precision and Recall for the example is shown in the fig below.



Precision and Recall for Credit Card example

Now, if we simply take arithmetic mean of both, then it comes out to be nearly 51%. We shouldn't be giving such a moderate score to a terrible model since its just predicting every transaction as fraud.

So, we need something more balanced than the arithmetic mean and that is harmonic mean.



The Harmonic mean is given by the formula shown in the figure on the left.

Harmonic mean is kind of an average when x and y are equal. But when x and y are different, then it's closer to the smaller number as compared to the larger number.

For our previous example, F1 Score = Harmonic Mean(Precision, Recall)

F1 Score = 2 ∗ Precision ∗ Recall/(Precision + Recall)=2 ∗ 3 ∗ 100/103 = 5%

So if one number is really small between precision and recall, the F1 Score kind of raises a flag and is more closer to the smaller number than the bigger one, giving the model an appropriate score rather than just an arithmetic mean.

4. Regression Metrics

In this section will review 2 of the most common metrics for evaluating predictions on regression machine learning problems:

• Mean Absolute Error.

• Mean Squared Error.

• R2 Metric

4.1. Mean Absolute Error

The Mean Absolute Error (or MAE) is the sum 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). The example below demonstrates calculating mean absolute error on the Boston house price dataset.

```
# Cross Validation Regression MAE
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.data"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = pandas.read_csv(url, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LinearRegression()
scoring = 'neg_mean_absolute_error'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MAE: %.3f (%.3f)") % (results.mean(), results.std())
```

A value of 0 indicates no error or perfect predictions. Like logloss, this metric is inverted by the cross_val_score() function.

```
MAE: -4.005
```

4.2. Mean Squared Error

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). The example below provides a demonstration of calculating mean squared error.

```
# Cross Validation Regression MSE
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.data"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
'PTRATIO', 'B', 'LSTAT', 'MEDV']
dataframe = pandas.read_csv(url, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LinearRegression()
scoring = 'neg_mean_squared_error'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MSE: %.3f (%.3f)") % (results.mean(), results.std())
```

This metric too is inverted so that the results are increasing. Remember to take the absolute value before taking the square root if you are interested in calculating the RMSE.

4.3. R2 Metric

The R2 (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. The example below provides a demonstration of calculating the mean R2 for a set of predictions.

```
# Cross Validation Regression R^2
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.data"
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', '
dataframe = pandas.read_csv(url, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LinearRegression()
scoring = 'r2'
results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("R^2: %.3f (%.3f)") % (results.mean(), results.std())
```

You can see that the predictions have a poor fit to the actual values with a value close to zero and less than 0.5.