Published in Towards Data Science

Joos Korstanje Follow
Aug 31, 2021 · 13 min read · · Listen

Save

The F1 score

All you need to know about the F1 score in machine learning. With an example applying the F1 score in Python.

F1 Score. Photo by Jonathan Chng on Unsplash.

The f1 score is a proposed improvement of two simpler performance metrics. Therefore, before getting into the details of the F1 score, let's step back and do an overview of those metrics underlying the F1 score.

Accuracy

Accuracy is a metric for classification models that measures the number of predictions that are correct as a percentage of the total number of predictions that are made. As an example, if 90% of your predictions are correct, your accuracy is simply 90%.

$$Accuracy = \frac{\#\ of\ correct\ predictions}{\#\ of\ total\ predictions}$$

Accuracy formula. Picture By Author.

Accuracy is a useful metric only when you have an equal distribution of classes on your classification. This means that if you have a use case in which you observe more data points of one class than of another, the accuracy is not a useful metric anymore. Let's see an example to illustrate this:

Imbalanced data example

Imagine you are working on the sales data of a website. You know that 99% of website visitors don't buy and that only 1% of visitors buy something. You are building a classification model to predict which website visitors are buyers and which are just lookers.

Now imagine a model that doesn't work very well. It predicts that 100% of your visitors are just lookers and that 0% of your visitors are buyers. It is clearly a very wrong and useless model.

> Accuracy is not a good metric to use when you have class imbalance.

What would happen if we'd use tl 456 | 3 'a on this model? Your model has predicted only 1% wrongly: all the buyers have been misclassified as lookers. The percentage of correct predictions is therefore 99%. **The problem here is that an accuracy of 99% sounds like a great result, whereas your model performs very poorly.** In conclusion: accuracy is not a good metric to use when you have class imbalance.

Solving imbalanced data through resampling

One way to solve class imbalance problems is to **work on your sample**. With specific sampling methods, you can resample your data set in such a way that the data is not imbalanced anymore. You can then use accuracy as a metric again. In this article, you can find out how to use such methods including undersampling, oversampling, and SMOTE data augmentation.

Solving imbalanced data through metrics

Another way to solve class imbalance problems is to use **better accuracy metrics like the F1 score**, which take into account not only the number of prediction errors that your model makes, but that also look at the type of errors that are made.

Precision and Recall: foundations of the F1 score

Precision and Recall are the two most common metrics that take into account class imbalance. **They are** also **the foundation of the F1 score**! Let's have a better look at Precision and Recall before **combining them into the F1 score** in the next part.

Precision: the first part of the F1 score

Precision is the first part of the F1 Score. It can also be used as an individual machine learning metric. It's formula is shown here:

$$Precision = \frac{\#\ of\ True\ Positives}{\#\ of\ True\ Positives + \#\ of\ False\ Positives}$$

Precision Formula. Picture By Author.

You can interpret this formula as follows. **Within everything that has been predicted as a positive, precision counts the percentage that is correct:**

- A **not precise model** may find a lot of the positives, but its selection method is noisy: it also wrongly detects many positives that aren't actually positives.
- A **precise model** is very "pure": maybe it does not find all the positives, but the ones that the model does class as positive are very likely to be correct.

Recall: the second part of the F1 score

Recall is the second component of the F1 Score, although recall can also be used as an individual machine learning metric. The formula for recall is shown here:

$$Recall = \frac{\#\ of\ True\ Positives}{\#\ of\ True\ Positives + \#\ of\ False\ Negatives}$$

Recall Formula. Picture By Author.

You can interpret this formula as follows. **Within everything that actually is positive, how many did the model succeed to find:**

- A model with **high recall** succeeds well in finding all the **positive cases** in the data, even though they may also wrongly identify some negative cases as positive cases.
- A model with **low recall** is not able to find all (or a large part) of the positive cases in the data.

Precision vs Recall

To clarify, think of the following example of a supermarket that has sold a product with a problem, and they need to recall it: they are only interested in making sure that they find all the problematic products back. It does not really matter to them if clients send back some non-problematic products as well, so the precision is not of interest to this supermarket.

Precision-Recall Trade-Off

Ideally, we would want both: a model that **identifies all of our positive cases** and that is at the same time **identifies only positive cases**.

In real life, we, unfortunately, have to deal with the so-called **Precision-Recall Trade-Off.**

The Precision-Recall Trade-Off represents the fact that in many cases, you can tweak a model to increase precision at a cost of a lower recall, or on the other hand increase recall at the cost of lower precision.

The F1 score: combining Precision and Recall

Precision and Recall are the two building blocks of the F1 score. The goal of the F1 score is to **combine the precision and recall metrics into a single metric.** At the same time, the F1 score has been designed to **work well on imbalanced data.**

F1 score formula

The F1 score is defined as the harmonic **mean of precision and recall**.

As a short reminder, the harmonic mean is an alternative metric for the more common arithmetic mean. It is often useful when computing an average rate.

In the F1 score, we compute the **average of precision and recall.** They are both rates, which makes it a logical choice to use the harmonic mean. The F1 score formula is shown here:

This makes that the formula for the F1 score is the following:

$$F1\ score = 2 * \frac{Precision\ * Recall}{Precision + Recall}$$

F1 Score formula. Picture By Author.

Since the F1 score is an average of Precision and Recall, it means that **the F1 score gives equal weight to Precision and Recall:**

- A model will obtain a **high F1 score** if both Precision and Recall are high
- A model will obtain a **low F1 score** if both Precision and Recall are low
- A model will obtain a **medium F1 score** if one of Precision and Recall is low and the other is high

Should the F1 score replace other metrics?

Throughout the article, you have seen a number of definitions. Before moving starting to implement the F1 Score in Python, let's sum up when to use the F1 Score and how to benchmark it against other metrics.

Accuracy vs Precision and Recall

Accuracy is the simplest classification metric. It simply measures the percentage of correct predictions that a machine learning model has made. You have seen that **accuracy is a bad metric in the case of imbalanced data** because it cannot distinguish between specific types of errors (false positives and false negatives).

Precision and Recall are performance metrics that are more suitable when having imbalanced data because they allow taking into account the type of errors (false positives or false negatives) that your model makes.

F1 Score vs Precision and Recall

The F1 Score combines Precision and Recall into **a single metric**. In many situations, like automated benchmarking, or grid search, it is much more convenient to have only one performance metric rather than multiple.

Should you use the F1 Score?

In conclusion, when you have the possibility to do so, you should definitely **look at multiple metrics** for each of the models that you try out. Each metric has advantages and disadvantages and each of them will give you specific information on the strengths and weaknesses of your model.

The real difficulty of choice occurs when doing automated model training, or when using Grid Search for tuning models. In those cases, you'll **have to specify a single metric** that you want to optimize.

In this case, my advice would be to have a good look at multiple different metrics of one or a few sample models. Then, when you **understand the implications for your specific use case**, you can choose one metric for optimization or tuning.

If you **move your model to production** for long-term use, you should regularly come back to do **model maintenance** and verify if the model is still behaving as it should be.

The F1 score in Python

Let's now get to an example in which we will understand the added value of the F1 Score. We will use an example data set that contains data on a number of website visitors.

The goal of the exercise will be to build a simple classification model that uses four independent variables to predict whether the visitor will buy something. We will see how to use different metrics, and we will see how different metrics will give us different conclusions.

The same data set was used in this article which proposes to use the SMOTE upsampling technique to improve model performance.

We will not use SMOTE here, as the goal is to demonstrate the F1 score. Yet, if you're interested in handling imbalanced data, it could definitely be worth it to combine both methods.

The data

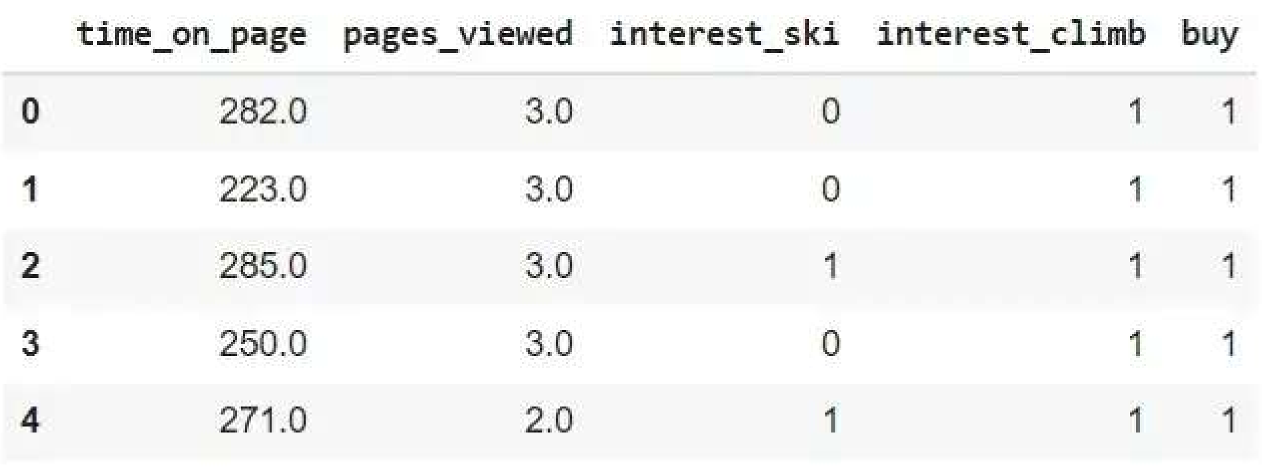
You can import the data into Python directly from GitHub. The following code allows you to read the raw file directly:

```python
import pandas as pd
data = pd.read_csv('https://raw.githubusercontent.com/JoosKorstanje/datasets/main/sales_data.csv
data.head()
```

F1_score_001.py hosted with ❤ by GitHub view raw

F1 Score. Importing the data

You will obtain a data frame that looks as follows:



	time_on_page	pages_viewed	interest_ski	interest_climb	buy
0	282.0	3.0	0	1	1
1	223.0	3.0	0	1	1
2	285.0	3.0	1	1	1
3	250.0	3.0	0	1	1
4	271.0	2.0	1	1	1

F1 Score. The data set. Picture By Author.

In this data set, we have the following five variables:

- `buy` : The variable of interest tells us whether the visitor ended up buying our new mountain sports product.
- `time_on_page` : The amount of time that the visitor spends on the page
- `pages_viewed` : The number of pages of our website that the visitor has viewed
- `interest_ski` : A variable from the customer relations database, that tells us whether the visitor has previously bought any ski-related items.
- `interest_climb` `: A variable from the customer relations database, that tells us whether the visitor has previously bought any mountain climbing-related items.

Verifying class imbalance

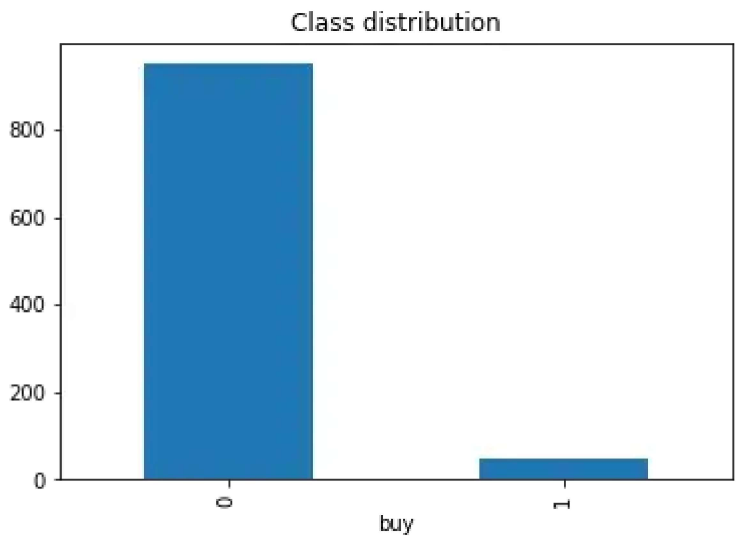
In our data set, we have only a very small percentage of buyers. If you want, you can verify this using the following code:

```
data.pivot_table(index='buy', aggfunc='size').plot(kind='bar', title = 'Class distribution')
```

F1_score_002.py hosted with ❤ by GitHub view raw

Verifying the class distribution with a bar graph

You will obtain the following bar graph:



F1 Score. Verifying that the data is indeed imbalanced. Picture By Author.

You can see here that there are very few buyers compared to the other visitors. This is something that happens very often when building models for e-commerce, as well as for other types of models like fraud detection and more. This confirms that the F1 score will probably come in handy.

Train Test Stratified

Before building any model, we should create a train/test split. If you're unfamiliar with the train/test approach in machine learning, I advise checking out this article first.

However, **it is risky to do a standard random train/test split when having strong class imbalance.** Due to the very small number of positive cases, you might end up with a train and test set that have very different class distributions. You may even end up with close to zero positive cases in your test set.

Stratified sampling is a sampling method that avoids disturbing class balance in your samples. It allows you to generate a train and a test set with the exact same class balance as in the original data. You can use the following code to execute stratified train/test sampling in scikitlearn:

```python
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size = 0.3, stratify=data.buy)
```

F1_score_003.py hosted with ❤ by GitHub view raw

F1 Score. Stratified sampling for the train and test data.

If you want, you can use the same code as before to generate the bar chart showing the class distribution. It will confirm that the class distribution is exactly the same in the total data as in the train set and in the test set.

A baseline model

As a baseline model, we will create a very bad model that predicts that nobody buys anything. This comes down to generating a list of predictions that are all 0. You can do this as follows:

```python
# this very bad model predicts that nobody ever buys anything
preds = [0] * len(test)
```

F1_score_004.py hosted with ❤ by GitHub view raw

F1 score. Creating a very bad baseline model.

Although we already know that this model is very bad, let's still try to find out the accuracy of this model. You can use scikitlearn's accuracy function as follows:

```python
from sklearn.metrics import accuracy_score
accuracy_score(test.buy, preds)
```

F1_score_005.py hosted with ❤ by GitHub view raw

F1 score. Using the accuracy to evaluate our bad baseline model.

The accuracy of our very bad model is surprisingly high: 95%! If you have followed along from the beginning, you probably understand why. In the test data, we know that there are very few buyers. The model predicts that nobody buys anything. Therefore, it is wrong only for the buyers (5% of the data set).

This is the exact reason why we need to worry about Recall and Precision. Let's use the following code to compute the Recall and Precision of this model:

```python
from sklearn.metrics import precision_score, recall_score
print('Precision is: ', precision_score(test.buy, preds))
print('Recall is: ', recall_score(test.buy, preds))
```

F1_score_006.py hosted with ❤ by GitHub view raw

F1 score. Using precision and recall to evaluate our bad baseline model.

Remember, Precision will tell you the percentage of correctly predicted buyers as a percentage of the total number of predicted buyers. In this very bad model, not a single person was identified as a buyer and the **Precision is therefore 0!**

Recall, on the other hand, tells you the percentage of buyers that you have been able to find within all of the actual buyers. Since your model has not found a single buyer, **Recall is also 0!**

Let's use the following code to see what the resulting F1 score is:

```python
from sklearn.metrics import f1_score
print('F1 is: ', f1_score(test.buy, preds))
```

F1_score_007.py hosted with ❤ by GitHub view raw

F1 score. Computing the F1 score of our bad baseline model.

The result is not surprising. As the F1 score is the harmonic mean of precision and recall, the **F1 score is also 0**.

This model in this example was not an intelligent model at all. Yet the example shows that **it can be very dangerous to use accuracy as a metric on imbalanced data sets.** This model is really not performant at all and a performance evaluation of 0 would be the only fair evaluation. Precision, recall, and the F1-score have all proven to be much better cases in this example.

A better model

Since the model in the previous example was very simple, let's redo another example with a real model. We will use a Logistic Regression model for this second example. Let's build the model using the following code and see what happens:

```python
from sklearn.linear_model import LogisticRegression

# Instantiate the Logistic Regression with only default settings
my_log_reg = LogisticRegression()

# Fit the logistic regression on the independent variables of the train data with buy as depend
my_log_reg.fit(train[['time_on_page',   'pages_viewed', 'interest_ski', 'interest_climb']], tra

# Make a prediction using our model on the test set
preds = my_log_reg.predict(test[['time_on_page',        'pages_viewed', 'interest_ski', 'intere
```

Let's do a detailed inspection of the predictions. In binary classification (classification with two outcomes like in our example), you can use the confusion matrix to differentiate between four types of predictions:

- True positives (buyers correctly predicted as buyers)
- False positives (non-buyers incorrectly predicted as buyers)
- True negatives (non-buyers correctly predicted as non-buyers
- False negatives (buyers incorrectly predicted as non-buyers)

You can obtain the confusion matrix in scikitlearn as follows:

```python
from sklearn.metrics import confusion_matrix
```

```
tn, fp, fn, tp = confusion_matrix(test['buy'], preds).ravel()
print('True negatives: ', tn, '\nFalse positives: ', fp, '\nFalse negatives: ', fn, '\nTrue Posi
```

F1 score 009.py hosted with ❤ by GitHub view raw

F1 score. Obtaining the confusion matrix.

The result is the following:

- True negatives: 280
- False positives: 5
- False negatives: 10
- True Positives: 5

This model evaluation data is very detailed. Of course, it would be easier to get this into a single performance metric. The accuracy is the simplest performance metric, so let's see what the accuracy score is on this example:

```
print('Accuracy is: ', accuracy_score(test.buy, preds))
```

F1_score_010.py hosted with ❤ by GitHub view raw

F1 score. Computing the accuracy on the better model.

Interestingly, the accuracy of the logistic regression is 95%: exactly the same as our very bad baseline model!

Let's see what Precision and Recall have to say about this:

```
print('Precision is: ', precision_score(test.buy, preds))
print('Recall is: ', recall_score(test.buy, preds))
```

F1_score_011.py hosted with ❤ by GitHub view raw

F1 score. Computing precision and recall on the better model.

We end up with the following metrics for Precision and Recall:

- Precision is: 0.5
- Recall is: 0.33

Let's also check the F1 score using scikitlearn's f1 score:

```python
print('F1 is: ', f1_score(test.buy, preds))
```

F1_score_012.py hosted with ❤ by GitHub view raw

F1 score. Computing the F1 score on the better model.

The obtained F1 score is 0.4.

Which model and metric is better?

So the accuracy tells us that the logistic regression is just as good as the bad baseline model, but precision and recall tell us that the logistic regression is better. Let's try to understand why:

- The total number of mistakes of the two models is the same. Therefore the **accuracy** is the same.
- The second model is actually capable of finding (at least some) positive cases (buyers), whereas the first model did not find a single buyer in the data. The **recall** of the second model is, therefore, higher (0.33 for the logistic regression instead of 0 for the first model).
- The first model did not find any buyers and the precision is therefore automatically zero. The logistic regression did find some buyers, so we can compute a precision. The **precision** shows how much of the predicted buyers were actually correct. This ends up being 50%.
- The **F1 score** is the metric that we are really interested in. The goal of the example was to show its added value for modeling with imbalanced data. The resulting **F1 score of the first model** was 0: we can be happy with this score, as it was a very bad model.
- The **F1 score of the second model** was 0.4. This shows that the second model, although far from perfect, is at least a serious improvement from the first model. This is valuable information that we could not have obtained using accuracy as a metric because the accuracy of both models is the same.

Conclusion

In this article, the F1 score has been shown as a model performance metric. The F1 score becomes especially valuable when working on classification models in which your data set is imbalanced.

You have seen that the F1 score combines precision and recall into a single metric. This makes it easy to use in grid search or automated optimization.

In the Python example, you have seen a case of imbalanced data set in a classification model. You have seen how accuracy can be very misleading, as it gives a bad model a great score. In the last part, you have seen that the F1 score works much better in estimating the performance of a machine learning model.

I hope this article was useful for you. Thanks for reading!

Artificial Intelligence Data Science Machine Learning Python Deep Dives