Machine Learning — Multiclass Classification with Imbalanced Dataset

Challenges in classification and techniques to improve performance

Javaid Nabi · Follow

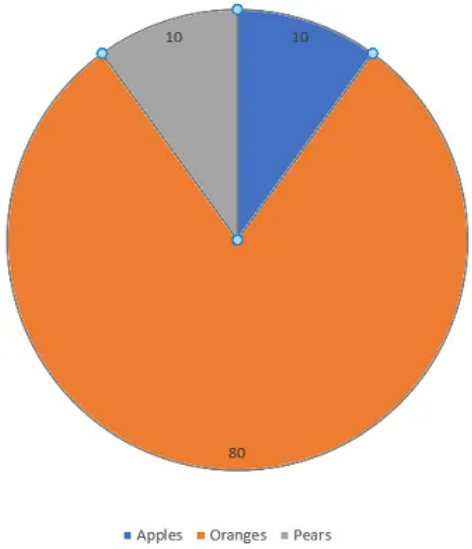
Published in Towards Data Science · 7 min read · Dec 23, 2018

source [Unsplash]

Classification problems having multiple classes with imbalanced dataset present a different challenge than a binary classification problem. The skewed distribution makes many conventional machine learning algorithms less effective, especially in predicting minority class examples. In order to do so, let us first understand the problem at hand and then discuss the ways to overcome those.

1. **Multiclass Classification:** A classification task with more than two classes; e.g., classify a set of images of fruits which may be oranges, apples, or pears. Multi-class classification makes the assumption that each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time.

2. **Imbalanced Dataset:** Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally. For example, you may have a 3-class classification problem of set of fruits to classify as oranges, apples or pears with total 100 instances . A total of 80 instances are labeled with Class-1 (Oranges), 10 instances with Class-2 (Apples) and the remaining 10 instances are labeled with Class-3 (Pears). This is an imbalanced dataset and the ratio of 8:1:1. Most classification data sets do not have exactly equal number of instances in each class, but a small difference often does not matter. There are problems where a class imbalance is not just common, it is expected. For example, in datasets like those that characterize fraudulent transactions are imbalanced. The vast majority of the transactions will be in the "Not-Fraud" class and a very small minority will be in the "Fraud" class.



Dataset

The data set we will be using for this example is the famous "20 News groups" data set. The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering.

scikit-learn provides the tools to pre-process the dataset, refer here for more details. The number of articles for each news group given below is roughly uniform.

```
rec.sport.hockey          600
soc.religion.christian    599
rec.motorcycles           598
rec.sport.baseball        597
sci.crypt                 595
rec.autos                 594
sci.med                   594
sci.space                 593
comp.windows.x            593
comp.os.ms-windows.misc   591
sci.electronics           591
comp.sys.ibm.pc.hardware  590
misc.forsale              585
comp.graphics             584
comp.sys.mac.hardware     578
talk.politics.mideast     564
talk.politics.guns        546
alt.atheism               480
talk.politics.misc        465
talk.religion.misc        377
```

Removing some news articles from some groups to make the overall dataset imbalanced like below.

```
rec.sport.hockey          600
rec.motorcycles           598
rec.sport.baseball        597
rec.autos                 594
talk.politics.guns        546
talk.religion.misc        377
sci.med                   207
sci.electronics           205
sci.space                 197
sci.crypt                 183
misc.forsale              171
comp.os.ms-windows.misc   151
comp.graphics             146
comp.sys.ibm.pc.hardware  137
comp.windows.x            136
comp.sys.mac.hardware     131
soc.religion.christian     86
talk.politics.mideast      67
alt.atheism                63
talk.politics.misc         55
```

Now our imbalanced dataset with 20 classes is ready for further analysis.

```
In [178]: data_imb.head()
```

Out[178]:

	filename	category	news
0	20news-bydate/20news-bydate-train\rec.sport.ba...	rec.sport.baseball	From: cubbie@garnet.berkeley.edu (...
1	20news-bydate/20news-bydate-train\comp.sys.mac...	comp.sys.mac.hardware	From: gnelson@pion.rutgers.edu (Gregory Nelson...
2	20news-bydate/20news-bydate-train\sci.crypt\15246	sci.crypt	From: crypt-comments@math.ncsu.edu\nSubject: C...
3	20news-bydate/20news-bydate-train\comp.sys.mac...	comp.sys.mac.hardware	From: ()\nSubject: Re: Quadra SCSI Problems??...
4	20news-bydate/20news-bydate-train\alt.atheism\...	alt.atheism	From: keith@cco.caltech.edu (Keith Allan Schne...

Build Model

As this is a classification problem, we will use the similar approach as described in my previous article for sentiment analysis. The only difference is here we are dealing with multiclass classification problem.

```
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, GRU
from keras.layers.embeddings import Embedding

EMBEDDING_DIM = 100

print('Build model...')

model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=max_length))
model.add(GRU(units=32,  dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(num_labels, activation='softmax'))

# try using different optimizers and different optimizer configs
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

print('Summary of the built model...')
print(model.summary())
```

The last layer in the model is `Dense(num_labels, activation ='softmax')` *,with* `num_labels=20` *classes, 'softmax' is used instead of 'sigmoid' . The other change in the model is about changing the loss function to* `loss = 'categorical_crossentropy'`, *which is suited for multi-class problems.*

Train Model

```
num_epochs =10
batch_size = 128
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=num_epochs,
                    verbose=2,
                    validation_split=0.2)
```

Training the model with 20% validation set `validation_split=20` and using `verbose=2`, we see validation accuracy after each epoch. Just after 10 epochs we reach validation accuracy of 90%.

```
Train on 3357 samples, validate on 840 samples
Epoch 1/10
 - 6s - loss: 2.1579 - acc: 0.4111 - val_loss: 1.1341 - val_acc: 0.7214
Epoch 2/10
 - 2s - loss: 0.6236 - acc: 0.8591 - val_loss: 0.5506 - val_acc: 0.8488
Epoch 3/10
 - 2s - loss: 0.1342 - acc: 0.9827 - val_loss: 0.4166 - val_acc: 0.8750
Epoch 4/10
 - 2s - loss: 0.0324 - acc: 0.9970 - val_loss: 0.3812 - val_acc: 0.8869
Epoch 5/10
 - 2s - loss: 0.0135 - acc: 0.9994 - val_loss: 0.3602 - val_acc: 0.8964
Epoch 6/10
 - 2s - loss: 0.0057 - acc: 1.0000 - val_loss: 0.3707 - val_acc: 0.8845
Epoch 7/10
 - 2s - loss: 0.0046 - acc: 1.0000 - val_loss: 0.3735 - val_acc: 0.8869
Epoch 8/10
 - 2s - loss: 0.0025 - acc: 1.0000 - val_loss: 0.3613 - val_acc: 0.8929
Epoch 9/10
 - 2s - loss: 0.0019 - acc: 1.0000 - val_loss: 0.3616 - val_acc: 0.8952
Epoch 10/10
 - 2s - loss: 0.0016 - acc: 1.0000 - val_loss: 0.3596 - val_acc: 0.9012
```

Evaluate Model

```
score, acc = model.evaluate(x_test, y_test,
                        batch_size=batch_size, verbose=2)

print('Test accuracy:', acc)
```

```
Test accuracy: 0.8780952383223034
```

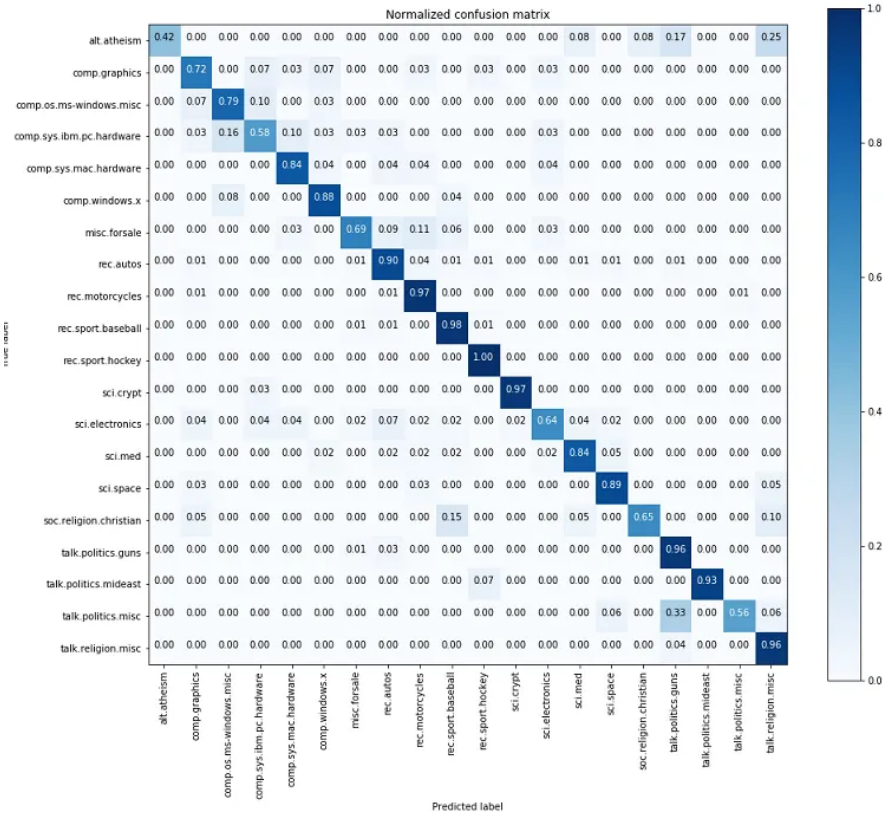
This looks like a very good accuracy but is the model really doing well?

> ***How to measure model performance?*** *Let us consider that we train our model on imbalanced data of earlier example of fruits and since data is heavily biased towards Class-1 (Oranges), the model over-fits on the Class-1 label and predicts it in most of the cases and we achieve an accuracy of 80% which seems very good at first but looking closely, it may never be able to classify apples or pears correctly. Now the question is if the accuracy, in this case, is not the right metric to choose then what metrics to use to measure the performance of the model?*

Confusion-Matrix

With imbalanced classes, it's easy to get a high accuracy without actually making useful predictions. So, **accuracy** as an evaluation metrics makes sense only if the class labels are uniformly distributed. In case of imbalanced classes confusion-matrix is good technique to summarizing the performance of a classification algorithm.

> *Confusion Matrix is a performance measurement for a classification algorithm where output can be two or more classes.*



x-axis=Predicted label, y-axis, True label

When we closely look at the confusion matrix, we see that the classes [*alt.athiesm, talk.politics.misc, soc.religion.christian*] which have very less samples [65,53, 86] respectively are indeed having very less scores [0.42, 0.56, 0.65] as compared to the classes with higher number of samples like [*rec.sport.hockey, rec.motorcycles*]. Thus looking at the confusion matrix one can clearly see how the model is performing on classifying various classes.

How to improve the performance?

There are various techniques involved in improving the performance of imbalanced datasets.

Re-sampling Dataset

To make our dataset balanced there are two ways to do so:

1. **Under-sampling:** Remove samples from over-represented classes ; use this if you have huge dataset
2. **Over-sampling:** Add more samples from under-represented classes; use this if you have small dataset

SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE is an over-sampling method. It creates synthetic samples of the minority class. We use ***imblearn*** python package to over-sample the minority classes .

```
x_train.shape, y_train.shape
```

```
((4197, 15000), (4197, 20))
```

```
from imblearn.over_sampling import SMOTE
#Over-sampling: SMOTE
#SMOTE (Synthetic Minority Oversampling TEchnique) consists of synthesizing elements for the minority class,
#based on those that already exist. It works randomly picking a point from the minority class and computing
#the k-nearest neighbors for this point.The synthetic points are added between the chosen point and its neighbors.
#We'll use ratio='minority' to resample the minority class.
smote = SMOTE('minority')

X_sm, y_sm = smote.fit_sample(x_train, y_train)
print(X_sm.shape, y_sm.shape)
```

```
(4646, 15000) (4646, 20)
```

we have 4197 samples before and 4646 samples after applying SMOTE, looks like SMOTE has increased the samples of minority classes. We will check the performance of the model with the new dataset.

```
history = model.fit(X_sm, y_sm,
                    batch_size=batch_size,
                    epochs=num_epochs,
                    verbose=2,
                    class_weight=class_weight,
                    validation_split=0.2)
```

```
Train on 3716 samples, validate on 930 samples
Epoch 1/10
 - 10s - loss: 0.0593 - acc: 0.9839 - val_loss: 0.2841 - val_acc: 0.9075
Epoch 2/10
 - 2s - loss: 0.0138 - acc: 0.9995 - val_loss: 0.1916 - val_acc: 0.9441
Epoch 3/10
 - 3s - loss: 0.0068 - acc: 0.9997 - val_loss: 0.1903 - val_acc: 0.9387
Epoch 4/10
 - 3s - loss: 0.0057 - acc: 0.9997 - val_loss: 0.1924 - val_acc: 0.9376
Epoch 5/10
 - 2s - loss: 0.0054 - acc: 0.9997 - val_loss: 0.1889 - val_acc: 0.9452
Epoch 6/10
 - 2s - loss: 0.0051 - acc: 0.9997 - val_loss: 0.1899 - val_acc: 0.9430
Epoch 7/10
 - 3s - loss: 0.0050 - acc: 0.9997 - val_loss: 0.1897 - val_acc: 0.9419
Epoch 8/10
 - 2s - loss: 0.0048 - acc: 0.9997 - val_loss: 0.1889 - val_acc: 0.9409
Epoch 9/10
 - 2s - loss: 0.0047 - acc: 0.9997 - val_loss: 0.1900 - val_acc: 0.9398
Epoch 10/10
 - 2s - loss: 0.0047 - acc: 0.9997 - val_loss: 0.1889 - val_acc: 0.9409
```

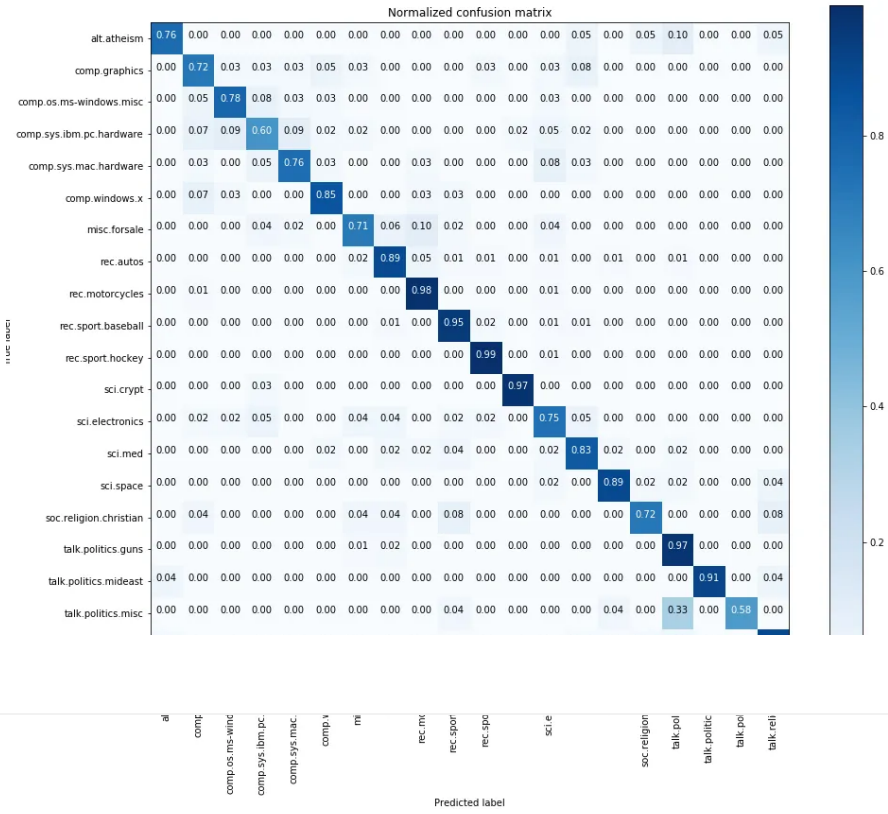
Improved validation accuracy from 90 to 94%. Let us test the model:

```
score, acc = model.evaluate(x_test, y_test,
                            batch_size=batch_size, verbose=2)

print('Test accuracy:', acc)
```

```
Test accuracy: 0.8885714287984939
```

Little improvement in test accuracy than before (from 87 to 88%). Let us have a look at the confusion matrix now.



We see that the classes [*alt.athiesm, talk.politics.misc, sci.electronics, soc.religion.christian*] having improved scores [0.76, 0.58, 0.75, 0.72] than before.Thus the model is performing better than before while classifying the classes even though accuracy is similar.

Another Trick:

Since classes are imbalanced, what about providing some bias to minority classes ? We can estimate class weights in scikit_learn by using `compute_class_weight` and use the parameter `'class_weight'`, while training the model. This can help to provide some bias towards the minority classes while training the model and thus help in improving performance of the model while classifying various classes.

```
from sklearn.utils import class_weight
class_weight = class_weight.compute_class_weight('balanced' ,np.unique(y_train_labels) ,y_train_labels)
num_epochs =10
batch_size = 128
history = model.fit(X_sm, y_sm,
                    batch_size=batch_size,
                    epochs=num_epochs,
                    verbose=2,
                    class_weight=class_weight,
                    validation_split=0.2)
```

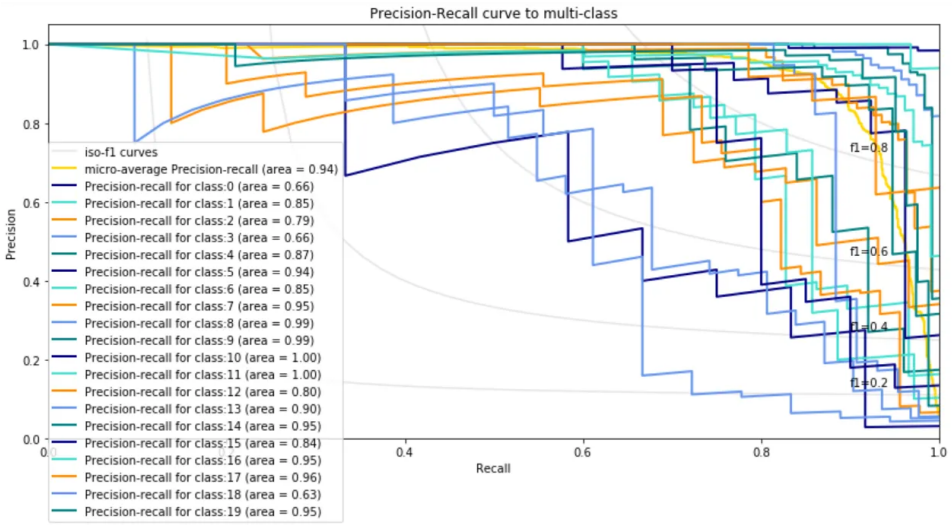
Precision-Recall Curves

Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced. **Precision** is a measure of the *ability of a classification model to identify only the relevant data points,* while ***recall*** is a measure of the *ability of a model to find all the relevant cases within a dataset*.

The precision-recall curve shows the trade-off between precision and recall for different threshold. A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate.

> High scores for both precision and recall show that the classifier is returning accurate results (precision), as well as returning a majority of all positive results (recall). An ideal system with high precision and high recall will return many results, with all results labeled correctly.

Below is a precision-recall plot for 20 News groups dataset using scikit-learn.



Precision-Recall Curve

We would like to have the area of P-R curve for each class to be close to 1. Except classes 0 , 3 & 18 rest of the classes are having area above .75. You can try with different classification models and hyper-parameter tuning techniques to improve the result further.

Conclusion

We discussed the problems associated with classification of multi classes in an imbalanced dataset. We also demonstrated how using the right tools and techniques help us in developing better classification models.

Thanks for reading. The code can be found on Github.

References

8 Tactics to Combat Imbalanced Classes in Your Machine Learning Dataset