12 July 2021 Report

Brad Burkman

12 July 2021

I didn't have an internet connection for much of the week, so I spent my time digging into scikit-learn. The code is really well annotated, with examples.

Contents

1 New Things I Explored this Week

- Scikit-Learn `class_weight = "balanced"` Parameter
  Made a big difference in linear models, not in tree models.
- Oversampling and Undersampling in Imbalanced-Learn
  I have just started on these. Undersampling is surprisingly effective.
- Balancing class weights in Decision Trees
  Dissertation topic?

2 Class_Weight = "Balanced"

Using the `class_weight = 'balanced'` argument in ML models greatly improved the recall and balanced precision in many models.

There's a file, `test_class_weight.py`, that illustrates what `class_weights` does.

Many models have a `class_weight` parameter, some don't.

In the table below,

- `cw` tells whether the model has a `class_weight` parameter.
- PB tells whether using the `class_weight` parameter gives a significant performance boost.
- bf1 is the balanced f1 score.
- Two bf1 scores indicates without → with `class_weight = "balanced"`
- MLPClassifier gets this good result with these parameters:
  `MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1, solver='lbfgs')`

Type	`cw`	Model	PB	bf1	Comments
Ensemble	No	AdaBoostClassifier		37%	
	No	BaggingClassifier		48%	
	Yes	ExtraTreesClassifier	Yes	5 → 15%	
	No	GradientBoostingClassifier		51%	
	Yes	RandomForestClassifier	Yes	nan → 8%	
		StackingClassifier			Stacks several classifiers together. Not its own classifier.
		VotingClassifier			Same
Linear	Yes	LogisticRegression	Yes	47 → 89%	
	Yes	Perceptron	Yes	80 → 88%	
	Yes	RidgeClassifier	YES	nan → 89%	
	Yes	RidgeClassifierCV	YES	nan → 89%	
	Yes	SGDClassifier	YES	37 → 90%	
Naive Bayes	No	GaussianNB		66%	
Neighbors	No	KNeighborsClassifier		8%	
	No	KNeighborsClassifier(n_neighbors=3)		6%	
	No	RadiusNeighborsClassifier			Error: No neighbors found within radius. Perhaps not applicable for binary?
Neural Network	No	MLPClassifier		63%	
SVM	Yes	LinearSVC	YES	34 → 86%	
	Yes	NuSVC			"Specified nu is infeasible."
	Yes	SVC	Yes	nan → 72%	
Tree	Yes	DecisionTreeClassifier	NO	57 → 48%	
	Yes	ExtraTreeClassifier	NO	31 → 27%	

3 Oversampling and Undersampling

When you have an imbalanced dataset, of, say, 200 nonfatal accidents for each fatal accident, oversampling and undersampling try to balance the training set (NOT the test set).

Oversampling artificially creates, in our case, 200 new fatal accidents for each original fatal, so that the number of fatal and nonfatal is equal. It could do it by just repeating each one 200 times (naïve method), or by creating new fatal records that are like other fatal records and unlike nonfatal ones.

Undersampling cuts out 199 of every 200 nonfatal records so that the sets are balanced. It could choose randomly or with some heuristic.

3.1 Results for Original Dataset

- I found every classifier in scikit-learn. I don't (yet) know what some of them do.
- For each classifier, if it accepted a `class_weight="balanced"` argument, I tried that too.
- For each classifier, there are two rows of the metrics Accuracy, Precision, Recall, and f1. The first row is raw, and the second is balanced. "Balanced Accuracy" is defined in scikit-learn. I used my own definition of "Balanced Precision," and used the balanced precision and accuracy to make "Balanced f1."
- The 2×2 array is the confusion matrix. What we want is for $C[1][0]$, false negatives, to be small and $C[1][1]$, true positives, to be large. We don't care if some of the nonfatal crashes are classified as fatal, but classifying almost everything as fatal (as in GaussianNB) is bad.

```
(128148, 658) (128148,) (127604,) (544,)

AdaBoostClassifier()
Accuracy, Precision, Recall, f1
99.52 32.69 12.41 17.99
56.15 99.12 12.41 22.06
[[31866    35]
 [  120    17]]

BaggingClassifier()
Accuracy, Precision, Recall, f1
99.6 60.53 16.79 26.29
58.37 99.72 16.79 28.74
[[31886    15]
 [  114    23]]

ExtraTreesClassifier()
Accuracy, Precision, Recall, f1
99.58 100.0 0.73 1.45
50.36 100.0 0.73 1.45
[[31901     0]
```

```
 [  136     1]]

ExtraTreesClassifier(class_weight='balanced')
Accuracy, Precision, Recall, f1
99.58 83.33 3.65 6.99
51.82 99.91 3.65 7.04
[[31900     1]
 [  132     5]]

GradientBoostingClassifier()
Accuracy, Precision, Recall, f1
99.54 42.25 21.9 28.85
60.88 99.42 21.9 35.89
[[31860    41]
 [  107    30]]

RandomForestClassifier()
Accuracy, Precision, Recall, f1
99.57 0.0 0.0 0.0
50.0 nan 0.0 nan
[[31901     0]
 [  137     0]]

RandomForestClassifier(class_weight='balanced')
Accuracy, Precision, Recall, f1
99.57 50.0 0.73 1.44
50.36 99.57 0.73 1.45
[[31900     1]
 [  136     1]]

LogisticRegression()
Accuracy, Precision, Recall, f1
99.61 62.5 21.9 32.43
60.92 99.74 21.9 35.91
[[31883    18]
 [  107    30]]

LogisticRegression(class_weight='balanced')
Accuracy, Precision, Recall, f1
95.54 6.82 74.45 12.49
```

```
85.04 94.46 74.45 83.27
[[30507  1394]
 [   35   102]]

Perceptron()
Accuracy, Precision, Recall, f1
99.55 45.21 24.09 31.43
61.98 99.48 24.09 38.79
[[31861    40]
 [  104    33]]

Perceptron(class_weight='balanced')
Accuracy, Precision, Recall, f1
93.28 4.92 80.29 9.27
86.82 92.34 80.29 85.89
[[29776  2125]
 [   27   110]]

RidgeClassifier()
Accuracy, Precision, Recall, f1
99.57 0.0 0.0 0.0
50.0 nan 0.0 nan
[[31901     0]
 [  137     0]]

RidgeClassifier(class_weight='balanced')
Accuracy, Precision, Recall, f1
93.12 5.09 85.4 9.6
89.28 92.58 85.4 88.85
[[29718  2183]
 [   20   117]]

RidgeClassifierCV(alphas=array([ 0.1,  1. , 10. ]))
Accuracy, Precision, Recall, f1
99.57 0.0 0.0 0.0
50.0 nan 0.0 nan
[[31901     0]
 [  137     0]]

RidgeClassifierCV(alphas=array([ 0.1,  1. , 10. ]), class_weight='balanced')
```

```
Accuracy, Precision, Recall, f1
93.11 5.12 86.13 9.66
89.64 92.62 86.13 89.26
[[29713  2188]
 [   19   118]]

SGDClassifier()
Accuracy, Precision, Recall, f1
99.61 67.65 16.79 26.9
58.38 99.8 16.79 28.74
[[31890    11]
 [  114    23]]

SGDClassifier(class_weight='balanced')
Accuracy, Precision, Recall, f1
92.56 4.5 81.02 8.52
86.82 91.64 81.02 86.0
[[29544  2357]
 [   26   111]]

GaussianNB()
Accuracy, Precision, Recall, f1
15.28 0.47 93.43 0.93
54.19 52.35 93.43 67.1
[[ 4769 27132]
 [    9   128]]

MLPClassifier()
Accuracy, Precision, Recall, f1
99.57 49.23 23.36 31.68
61.63 99.56 23.36 37.84
[[31868    33]
 [  105    32]]

MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,
              solver='lbfgs')
Accuracy, Precision, Recall, f1
99.56 47.73 30.66 37.33
65.26 99.53 30.66 46.88
[[31855    46]
```

```
 [   95    42]]

LinearSVC()
Accuracy, Precision, Recall, f1
99.59 62.07 13.14 21.69
56.55 99.74 13.14 23.22
[[31890    11]
 [  119    18]]

LinearSVC(class_weight='balanced')
Accuracy, Precision, Recall, f1
97.88 11.63 59.85 19.48
78.95 96.84 59.85 73.98
[[31278   623]
 [   55    82]]

SVC()
Accuracy, Precision, Recall, f1
99.57 0.0 0.0 0.0
50.0 nan 0.0 nan
[[31901     0]
 [  137     0]]
```

3.2 Naïve Random Oversampling

Instead of 127,604 nonfatal records and 544 fatal, we have 127,604 of each.

I've given the models that have a big improvement in *Balanced f1* from the original dataset.

```
AdaBoostClassifier()
Accuracy, Precision, Recall, f1
92.72 5.0 89.05 9.47
90.89 92.46 89.05 90.72
[[29583  2318]
 [   15   122]]

 GradientBoostingClassifier()
Accuracy, Precision, Recall, f1
93.44 5.52 89.05 10.4
91.25 93.16 89.05 91.06
[[29814  2087]
```

```
 [    15    122]]

 LinearSVC()
Accuracy, Precision, Recall, f1
95.67 6.6 69.34 12.06
82.57 94.27 69.34 79.91
[[30557  1344]
 [    42     95]]
```

3.3 Oversampling with SMOTE

Synthetic Minority Ovesampling TEchnique

Doesn't really work with binary data, so, as expected, no great improvement.

3.4 Undersampling with RandomUndersampler

Undersampling cut the number of nonfatal records in the training set down to the same as the fatal records, 544. The number of records in the test set was unchanged, because we want our model to work in reality, not in our imaginary world.

Big improvements.

Note that for LinearSVC, the class weight being balanced still made a big difference.

```
AdaBoostClassifier()
Accuracy, Precision, Recall, f1
89.82 3.79 93.43 7.28
91.62 90.16 93.43 91.77
[[28648  3253]
 [     9    128]]

BaggingClassifier()
Accuracy, Precision, Recall, f1
87.38 3.1 94.16 6.0
90.76 88.16 94.16 91.06
[[27866  4035]
 [     8    129]]

ExtraTreesClassifier()
Accuracy, Precision, Recall, f1
87.12 3.01 93.43 5.84
90.26 87.86 93.43 90.56
```

```
[[27782  4119]
 [    9   128]]

ExtraTreesClassifier(class_weight='balanced')
Accuracy, Precision, Recall, f1
87.49 3.06 91.97 5.92
89.72 88.01 91.97 89.95
[[27905  3996]
 [   11   126]]

GradientBoostingClassifier()
Accuracy, Precision, Recall, f1
89.75 3.76 93.43 7.23
91.58 90.1 93.43 91.73
[[28627  3274]
 [    9   128]]

RandomForestClassifier()
Accuracy, Precision, Recall, f1
86.17 2.71 89.78 5.26
87.97 86.64 89.78 88.18
[[27485  4416]
 [   14   123]]

RandomForestClassifier(class_weight='balanced')
Accuracy, Precision, Recall, f1
85.72 2.71 92.7 5.26
89.19 86.63 92.7 89.56
[[27335  4566]
 [   10   127]]

GaussianNB()
Accuracy, Precision, Recall, f1
90.02 1.95 45.26 3.73
67.74 82.22 45.26 58.38
[[28780  3121]
 [   75    62]]

MLPClassifier()
Accuracy, Precision, Recall, f1
```

```
88.63 3.3 90.51 6.37
89.56 88.83 90.51 89.66
[[28270  3631]
 [   13   124]]

MLPClassifier(alpha=1e-05, hidden_layer_sizes=(5, 2), random_state=1,
              solver='lbfgs')
Accuracy, Precision, Recall, f1
87.68 3.13 92.7 6.05
90.18 88.25 92.7 90.42
[[27965  3936]
 [   10   127]]

LinearSVC()
Accuracy, Precision, Recall, f1
88.37 3.26 91.24 6.29
89.8 88.68 91.24 89.94
[[28187  3714]
 [   12   125]]

LinearSVC(class_weight='balanced')
Accuracy, Precision, Recall, f1
88.37 3.26 91.24 6.29
89.8 88.68 91.24 89.94
[[28187  3714]
 [   12   125]]

SVC()
Accuracy, Precision, Recall, f1
87.2 3.03 93.43 5.88
90.3 87.93 93.43 90.6
[[27810  4091]
 [    9   128]]

SVC(class_weight='balanced')
Accuracy, Precision, Recall, f1
87.2 3.03 93.43 5.88
90.3 87.93 93.43 90.6
[[27810  4091]
 [    9   128]]
```

```
DecisionTreeClassifier()
Accuracy, Precision, Recall, f1
87.27 2.78 84.67 5.38
85.97 86.94 84.67 85.79
[[27842  4059]
 [   21   116]]

DecisionTreeClassifier(class_weight='balanced')
Accuracy, Precision, Recall, f1
86.88 2.81 88.32 5.45
87.6 87.06 88.32 87.69
[[27715  4186]
 [   16   121]]

ExtraTreeClassifier()
Accuracy, Precision, Recall, f1
76.83 1.58 86.86 3.11
81.82 78.91 86.86 82.69
[[24496  7405]
 [   18   119]]

ExtraTreeClassifier(class_weight='balanced')
Accuracy, Precision, Recall, f1
78.14 1.52 78.83 2.99
78.48 78.28 78.83 78.55
[[24925  6976]
 [   29   108]]
```

4 Balancing Class Weights in Decision Trees

I said in last week's report that, intuitively, trees make more sense for our classification problem. Decision trees accept the `class_weight = 'balanced'` argument, but it doesn't seem to make a difference. In the `class_weight.py` code references, the authors say

> The "balanced" heuristic is inspired by *Logistic Regression in Rare Events Data, King, Zen, 2001.*

So it makes sense that setting `class_weight='balanced'` greatly affects models that rely heavily on logistic regression, but not so much for decision trees.

Question: Is there a way to modify the `gini` and `entropy` penalty functions in decision trees to incorporate class weights?

I think this might be on the scale of a dissertation topic.