(https://www.linkedin.com/shareArticle?trk=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascience.com/imbalanclasses)

(http://www.facebook.com/sharer.php?u=https://elitedatascience.com/imbalanced-classes)

(https://plus.google.com/share?text=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascience.com/imbalaclasses)

(https://twitter.com/intent/tweet?text=How+to+Handle+Imbalanced+Classes+in+Machine+Learning+via+@elitedatascience&url=https://elitedatascience.com/imbalanced-classes)

How to Handle Imbalanced Classes in Machine Learning (https://elitedatascience.com/imbalanced-classes)

(http://service.weibo.com/share/share.php?url=https://elitedatascience.com/imbalanced-classes&title=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning)

(mailto:?subject=Check%20out%20this%20site&body=Check%20out%20this%20site%20https://elitedatascience.com/imbalanced-classes)

(https://getpocket.com/save?title=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascience.com/imbalaclasses)

Share (https://www.facebook.com/sharer.php?u=https%3A%2F%2Felitedatascience.com%2Fimbalanced-classes)

Google (https://plus.google.com/share?text=How+to+Handle+Imbalanced+Classes+in+Machine+Learning&url=https%3A%2F%2Felitedatascience.com%2Fimbalanced-classes)

Linkedin (https://www.linkedin.com/shareArticle?trk=How+to+Handle+Imbalanced+Classes+in+Machine+Learning&url=https%3A%2F%2Felitedatascience.com%2Fimbalanced-classes)

Tweet (https://twitter.com/intent/tweet?text=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascien

<<

Imbalanced classes put “accuracy” out of business. This is a surprisingly common problem in machine learning (specifically in classification), occurring in datasets with a disproportionate ratio of observations in each class.

Standard accuracy no longer reliably measures performance, which makes model training much trickier.

Imbalanced classes appear in many domains, including:

- Fraud detection
- Spam filtering
- Disease screening
- SaaS subscription churn
- Advertising click-throughs

In this guide, we'll explore 5 effective ways to handle imbalanced classes.

Intuition: Disease Screening Example

Let's say your client is a leading research hospital, and they've asked you to train a model for detecting a disease based on biological inputs collected from patients.

But here's the catch… the disease is relatively rare; it occurs in only 8% of patients who are screened.

Now, before you even start, do you see how the problem might break? Imagine if you didn't bother training a model at all. Instead, what if you just wrote a single line of code that always predicts 'No Disease?'


```
# Crappy, but accurate, solution                                    Python
def disease_screen(patient_data):
    # Ignore patient_data
    return 'No Disease.'
```

Well, guess what? Your "solution" would have 92% accuracy!

Unfortunately, that accuracy is misleading.

- For patients who *do not* have the disease, you'd have 100% accuracy.
- For patients who *do* have the disease, you'd have 0% accuracy.
- Your overall accuracy would be high simply because most patients do not have the disease (not because your model is any good).

This is clearly a problem because many machine learning algorithms are designed to maximize overall accuracy. The rest of this guide will illustrate different tactics for handling imbalanced classes.

Important notes before we begin:

First, please note that we're not going to split out a separate test set, tune hyperparameters, or implement cross-validation. In other words, we're not necessarily going to follow best practices.

Instead, this tutorial is focused purely on addressing imbalanced classes.

In addition, not every technique below will work for every problem. However, 9 times out of 10, at least one of these techniques should do the trick.

Balance Scale Dataset

<<

For this guide, we'll use a synthetic dataset called Balance Scale Data, which you can download from the UCI Machine Learning Repository here (http://archive.ics.uci.edu/ml/datasets/balance+scale).

This dataset was originally generated to model psychological experiment results, but it's useful for us because it's a manageable size and has imbalanced classes.

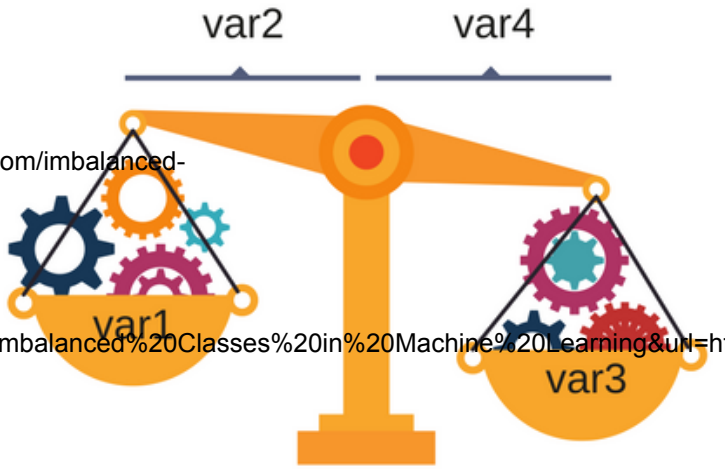
Import libraries and read dataset

```python
import pandas as pd
import numpy as np

# Read dataset
df = pd.read_csv('balance-scale.data',
                 names=['balance', 'var1', 'var2', 'var3', 'var4'])

# Display example observations
df.head()
```

	balance	var1	var2	var3	var4
0	B	1	1	1	1
1	R	1	1	1	2
2	R	1	1	1	3
3	R	1	1	1	4
4	R	1	1	1	5

The dataset contains information about whether a scale is balanced or not, based on weights and distances of the two arms.

- It has 1 target variable, which we've labeled `balance` .
- It has 4 input features, which we've labeled `var1` through `var4` .



The target variable has 3 classes.

- **R** for right-heavy, i.e. when `var3 * var4 > var1 * var2`
- **L** for left-heavy, i.e. when `var3 * var4 < var1 * var2`
- **B** for balanced, i.e. when `var3 * var4 = var1 * var2`

Count of each class

```python
df['balance'].value_counts()
# R    288
# L    288
# B     49
# Name: balance, dtype: int64
```

However, for this tutorial, we're going to turn this into a **binary classification** problem.

We're going to label each observation as **1** (positive class) if the scale is balanced or **0** (negative class) if the scale is not balanced:

Transform into binary classification

```python
# Transform into binary classification
df['balance'] = [1 if b=='B' else 0 for b in df.balance]


df['balance'].value_counts()
# 0    576
# 1     49
# Name: balance, dtype: int64
# About 8% were balanced
```

As you can see, only about 8% of the observations were balanced. Therefore, if we were to always predict **0**, we'd achieve an accuracy of 92%.

The Danger of Imbalanced Classes

Now that we have a dataset, we can really show the dangers of imbalanced classes.

First, let's import the Logistic Regression algorithm and the accuracy metric from Scikit-Learn (http://scikit-learn.org/stable/).

Import algorithm and accuracy metric

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Next, we'll fit a very simple model using default settings for everything.

Train model on imbalanced data

```python

```

```
# Separate input features (X) and target variable (y)
y = df.balance
X = df.drop('balance', axis=1)

# Train model
clf_0 = LogisticRegression().fit(X, y)

# Predict on training set
pred_y_0 = clf_0.predict(X)
```


As mentioned above, many machine learning algorithms are designed to maximize overall accuracy by default.

We can confirm this:

```
# How's the accuracy?
print( accuracy_score(pred_y_0, y) )
# 0.9216
```


So our model has 92% overall accuracy, but is it because it's predicting only 1 class?

```
# Should we be excited?
print( np.unique( pred_y_0 ) )
# [0]
```


As you can see, this model is only predicting **0**, which means it's completely ignoring the minority class in favor of the majority class.

Next, we'll look at the first technique for handling imbalanced classes: up-sampling the minority class.

1. Up-sample Minority Class

Up-sampling is the process of randomly duplicating observations from the minority class in order to reinforce its signal.

There are several heuristics for doing so, but the most common way is to simply resample with replacement.

First, we'll import the resampling module from Scikit-Learn:

```
Module for resampling
from sklearn.utils import resample
```

Next, we'll create a new DataFrame with an up-sampled minority class. Here are the steps:

1. First, we'll separate observations from each class into different DataFrames.
2. Next, we'll resample the minority class **with replacement**, setting the number of samples to match that of the majority class.
3. Finally, we'll combine the up-sampled minority class DataFrame with the original majority class DataFrame.

(https://www.linkedin.com/shareArticle?trk=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascience.com/imbalanced-classes)

Here's the code:

```
# Upsample minority class                                                    Python
# Separate majority and minority classes
df_majority = df[df.balance==0]
df_minority = df[df.balance==1]

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=576,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
df_upsampled.balance.value_counts()
# 1    576
# 0    576
# Name: balance, dtype: int64
```

(http://www.facebook.com/sharer.php?u=https://elitedatascience.com/imbalanced-classes)

(https://plus.google.com/share?text=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascience.com/imbalanced-classes)

(https://twitter.com/intent/tweet?text=How+to+Handle+Imbalanced+Classes+in+Machine+Learning&url=https://elitedatascience.com/imbalanced-classes)

(http://service.weibo.com/share/share.php?url=https://elitedatascience.com/imbalanced-classes&title=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning)

As you can see, the new DataFrame has more observations than the original, and the ratio of the two classes is now 1:1.

(mailto:?subject=Check out this site%20&body=Check out this site%20https://elitedatascience.com/imbalanced-classes)

Let's train another model using Logistic Regression, this time on the balanced dataset:

```
# Train model on upsampled dataset                                           Python
```

(https://getpocket.com/save?title=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascience.com/imbalanced-classes)

```
# Separate input features (X) and target variable (y)
y = df_upsampled.balance
X = df_upsampled.drop('balance', axis=1)

# Train model
clf_1 = LogisticRegression().fit(X, y)

# Predict on training set
pred_y_1 = clf_1.predict(X)

# Is our model still predicting just one class?
print( np.unique( pred_y_1 ) )
# [0 1]

# How's our accuracy?
print( accuracy_score(y, pred_y_1) )
# 0.666666666667
```


Great, now the model is no longer predicting just one class. While the accuracy also took a nosedive, it's now more meaningful as a performance metric.

2. Down-sample Majority Class

Down-sampling involves randomly removing observations from the majority class to prevent its signal from dominating the learning algorithm.

The most common heuristic for doing so is resampling without replacement.

The process is similar to that of up-sampling. Here are the steps:

1. First, we'll separate observations from each class into different DataFrames.
2. Next, we'll resample the majority class **without replacement**, setting the number of samples to match that of the minority class.
3. Finally, we'll combine the down-sampled majority class DataFrame with the original minority class DataFrame.

Here's the code:

Downsample majority class — Python

```
# Separate majority and minority classes
df_majority = df[df.balance==0]
df_minority = df[df.balance==1]

# Downsample majority class
df_majority_downsampled = resample(df_majority,
                                   replace=False,    # sample without replacement
                                   n_samples=49,     # to match minority class
                                   random_state=123) # reproducible results

# Combine minority class with downsampled majority class
df_downsampled = pd.concat([df_majority_downsampled, df_minority])

# Display new class counts
df_downsampled.balance.value_counts()
# 1    49
# 0    49
# Name: balance, dtype: int64
```

This time, the new DataFrame has fewer observations than the original, and the ratio of the two classes is now 1:1.

Again, let's train a model using Logistic Regression:

```
# Train model on downsampled dataset                                    Python
# Separate input features (X) and target variable (y)
y = df_downsampled.balance
X = df_downsampled.drop('balance', axis=1)

# Train model
clf_2 = LogisticRegression().fit(X, y)

# Predict on training set
pred_y_2 = clf_2.predict(X)

# Is our model still predicting just one class?
print( np.unique( pred_y_2 ) )
# [0 1]

# How's our accuracy?
print( metrics.accuracy_score(y, pred_y_2) )
# 0.581632653061
```

The model isn't predicting just one class, and the accuracy seems higher.

We'd still want to validate the model on an unseen test dataset, but the results are more encouraging.

3. Change Your Performance Metric

So far, we've looked at two ways of addressing imbalanced classes by resampling the dataset. Next, we'll look at using other performance metrics for evaluating the models.

Albert Einstein once said, "if you judge a fish on its ability to climb a tree, it will live its whole life believing that it is stupid." This quote really highlights the importance of choosing the right evaluation metric.

For a general-purpose metric for classification, we recommend **Area Under ROC Curve** (AUROC).

- We won't dive into its details in this guide, but you can read more about it here (https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it).
- Intuitively, AUROC represents the likelihood of your model distinguishing observations from two classes.
- In other words, if you randomly select one observation from each class, what's the probability that your model will be able to "rank" them correctly?

We can import this metric from Scikit-Learn:

Area Under ROC Curve — Python

```
from sklearn.metrics import roc_auc_score
```

To calculate AUROC, you'll need predicted class probabilities instead of just the predicted classes. You can get them using the `.predict_proba()` function like so:

Get class probabilities — Python

```
# Predict class probabilities
prob_y_2 = clf_2.predict_proba(X)

# Keep only the positive class
prob_y_2 = [p[1] for p in prob_y_2]

prob_y_2[:5] # Example
# [0.47413197226479618,
#  0.48205962213283882,
#  0.46862327066392456,
#  0.47868378832689096,
#  0.58143856820159667]
```

So how did this model (trained on the down-sampled dataset) do in terms of AUROC?

AUROC of model trained on downsampled dataset — Python

```
print( roc_auc_score(y, prob_y_2) )
# 0.568096626406
```

Ok... and how does this compare to the original model trained on the imbalanced dataset?

AUROC of model trained on imbalanced dataset (Python)

```
prob_y_0 = clf_0.predict_proba(X)
prob_y_0 = [p[1] for p in prob_y_0]

print( roc_auc_score(y, prob_y_0) )
# 0.530718537415
```

Remember, our original model trained on the imbalanced dataset had an accuracy of 92%, which is much higher than the 58% accuracy of the model trained on the down-sampled dataset.

However, the latter model has an AUROC of 57%, which is higher than the 53% of the original model (but not by much).

Note: if you got an AUROC of 0.47, it just means you need to invert the predictions because Scikit-Learn is misinterpreting the positive class. AUROC should be >= 0.5.

4. Penalize Algorithms (Cost-Sensitive Training)

The next tactic is to use penalized learning algorithms that increase the cost of classification mistakes on the minority class.

A popular algorithm for this technique is Penalized-SVM:

Support Vector Machine (Python)

```
from sklearn.svm import SVC
```

During training, we can use the argument `class_weight='balanced'` to penalize mistakes on the minority class by an amount proportional to how under-represented it is.

We also want to include the argument `probability=True` if we want to enable probability estimates for SVM algorithms.

Let's train a model using Penalized-SVM on the original imbalanced dataset:

Train Penalized-SVM on imbalanced dataset (Python)

```

```


<<

```
# Separate input features (X) and target variable (y)
y = df.balance
X = df.drop('balance', axis=1)

# Train model
clf_3 = SVC(kernel='linear',
            class_weight='balanced', # penalize
            probability=True)

clf_3.fit(X, y)

# Predict on training set
pred_y_3 = clf_3.predict(X)

# Is our model still predicting just one class?
print( np.unique( pred_y_3 ) )
# [0 1]

# How's our accuracy?
print( accuracy_score(y, pred_y_3) )
# 0.698

# What about AUROC?
prob_y_3 = clf_3.predict_proba(X)
prob_y_3 = [p[1] for p in prob_y_3]
print( roc_auc_score(y, prob_y_3) )
# 0.5305236678
```

Again, our purpose here is only to illustrate this technique. To really determine which of these tactics works best *for this problem*, you'd want to evaluate the models on a hold-out test set.

5. Use Tree-Based Algorithms

The final tactic we'll consider is using tree-based algorithms. Decision trees often perform well on imbalanced datasets because their hierarchical structure allows them to learn signals from both classes.

In modern applied machine learning, tree ensembles (Random Forests, Gradient Boosted Trees, etc.) almost always outperform singular decision trees, so we'll jump right into those:

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

Now, let's train a model using a Random Forest on the original imbalanced dataset.

Train Random Forest on imbalanced dataset

```
```

```
# Separate input features (X) and target variable (y)
y = df.balance
X = df.drop('balance', axis=1)

# Train model
clf_4 = RandomForestClassifier()
clf_4.fit(X, y)

# Predict on training set
pred_y_4 = clf_4.predict(X)

# How's the accuracy?
print( np.unique( pred_y_4 ) )
# [0 1]

# How's the accuracy?
print( accuracy_score(y, pred_y_4) )
# 0.9744

# What about AUROC?
prob_y_4 = clf_4.predict_proba(X)
prob_y_4 = [p[1] for p in prob_y_4]
print( roc_auc_score(y, prob_y_4) )
# 0.999078798186
```

Wow! 97% accuracy and nearly 100% AUROC? Is this magic? A sleight of hand? Cheating? Too good to be true?

Well, tree ensembles have become very popular because they perform extremely well on many real-world problems. We certainly recommend them wholeheartedly.

However:

While these results are encouraging, the model *could* be overfit, so you should still evaluate your model on an unseen test set before making the final decision.

Note: your numbers may differ slightly due to the randomness in the algorithm. You can set a random seed for reproducible results.

Honorable Mentions

There were a few tactics that didn't make it into this tutorial:

Create Synthetic Samples (Data Augmentation)

<<

Creating synthetic samples is a close cousin of up-sampling, and some people might categorize them together. For example, the SMOTE algorithm (https://www.jair.org/media/953/live-953-2037-jair.pdf) is a method of resampling from the minority class while slightly perturbing feature values, thereby creating "new" samples.

You can find an implementation of SMOTE in the imblearn library (http://contrib.scikit-learn.org/imbalanced-learn/generated/imblearn.over_sampling.SMOTE.html).

Combine Minority Classes

Combining minority classes of your target variable may be appropriate for some multi-class problems.

For example, let's say you wished to predict credit card fraud. In your dataset, each method of fraud may be labeled separately, but you might not care about distinguishing them. You could combine them all into a single 'Fraud' class and treat the problem as binary classification.

Reframe as Anomaly Detection

Anomaly detection, a.k.a. outlier detection, is for detecting outliers and rare events (https://en.wikipedia.org/wiki/Anomaly_detection). Instead of building a classification model, you'd have a "profile" of a normal observation. If a new observation strays too far from that "normal profile," it would be flagged as an anomaly.

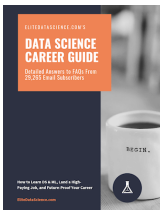
Conclusion & Next Steps

In this guide, we covered 5 tactics for handling imbalanced classes in machine learning:

1. Up-sample the minority class
2. Down-sample the majority class
3. Change your performance metric
4. Penalize algorithms (cost-sensitive training)
5. Use tree-based algorithms

These tactics are subject to the No Free Lunch theorem (http://elitedatascience.com/machine-learning-algorithms), and you should try several of them and use the results from the test set to decide on the best solution for your problem.

<<



Free: Data Science Career Guide

How to Learn Data Science & Machine Learning, Land a High-Paying Job, and Future-Proof Your Career

First Name

Email

Send My Download

« Previous Post

9 Mistakes to Avoid When Starting Your Career in Data Science (https://elitedatascience.com/beginner-mistakes)

Next Post »

The Beginner's Guide to Kaggle (https://elitedatascience.com/beginner-kaggle)

<<

LINKS

Start Here (https://elitedatascience.com/?page_id=1584)

Homepage (/)

Login (https://elitedatascience.com/login)

Articles (https://elitedatascience.com/)

Guides (https://elitedatascience.com/category/guides)

Concept Explainers (https://elitedatascience.com/category/explainers)

Code Tutorials (https://elitedatascience.com/category/tutorials)

Career Help (https://elitedatascience.com/category/career)

Tools & Resources (https://elitedatascience.com/category/resources)

SHARE

Share (https://www.facebook.com/sharer.php?u=https%3A%2F%2Felitedatascience.com)

Google (https://plus.google.com/share?text=How+to+Handle+Imbalanced+Classes+in+Machine+Learning&url=https%3A%2F%2Felitedatascience.com)

Linkedin (https://www.linkedin.com/shareArticle?trk=How+to+Handle+Imbalanced+Classes+in+Machine+Learning&url=https%3A%2F%2Felitedatascience.com)

Tweet (https://twitter.com/intent/tweet?text=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascience.com)

(https://www.linkedin.com/shareArticle?trk=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascience.com/imbalanced-classes)

(http://www.facebook.com/sharer.php?u=https://elitedatascience.com/imbalanced-classes)

(https://plus.google.com/share?text=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascience.com/imbalanced-classes)

(https://twitter.com/intent/tweet?text=How+to+Handle+Imbalanced+Classes+in+Machine+Learning&url=https://elitedatascience.com/imbalanced-classes)

(http://service.weibo.com/share/share.php?url=https://elitedatascience.com/imbalanced-classes&title=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning)

(mailto:?subject=Check%20out%20this%20site%20&body=Check%20out%20this%20site%20https://elitedatascience.com/imbalanced-classes)

(https://getpocket.com/save?title=How%20to%20Handle%20Imbalanced%20Classes%20in%20Machine%20Learning&url=https://elitedatascience.com/imbalanced-classes)

<<