Supervised Learning

Data Science & Business Analytics

Lesson 3

Tiago Cabo

Lesson Plan

- Improve feature engineering on the titanic dataset
- Improve python code in order to become more production ready
- More supervised classification algorithms:
  - Random Forest
  - XGBoost
  - K-NN
- Techniques to handle unbalanced datasets
- Techniques to scale values
- Regression supervised algorithms
- Example

Lesson goals

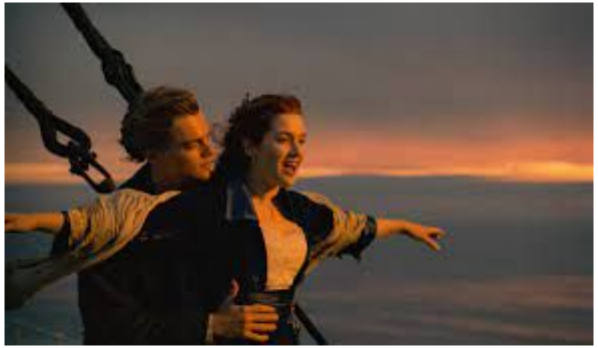
- Have a clear idea on how to build and iterate a production model
- Understand imbalance dataset and how to deal with that
- Understand main regression algorithms

Let’s Improve titanic model

Current kaggle performance is 0.77.

Any ideas on how to improve:

1. Test other models i.e. Random Forest, K-NN
2. Transform Fare feature into categorical
3. Same for Age
4. Create features based on Sibsp and Partch

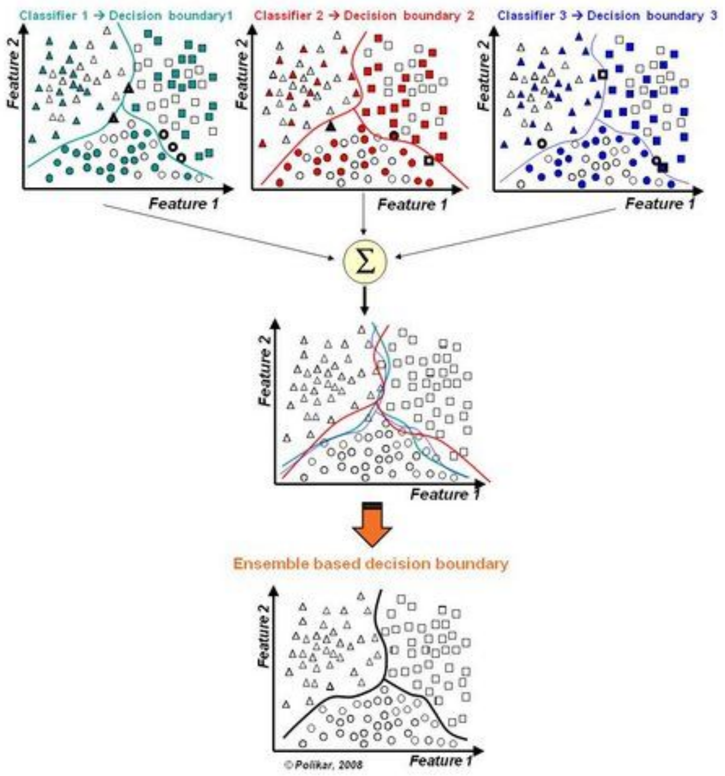


More algorithms

- Ensemble Algorithms
- Bagging Algorithms
- Stacking Algorithms
- Boosting Algorithms
- Random Forest
- KNN

Ensemble algorithms

“ **Ensemble learning** is a general meta approach to machine learning that seeks better predictive performance by combining the predictions from multiple models.” *machine learning mastery*


Classifier 1 → Decision boundary1
Classifier 2 → Decision boundary 2
Classifier 3 → Decision boundary 3
Feature 1
Feature 2
Σ
Ensemble based decision boundary
© Polikar, 2008

Bagging Algorithms

In this kind of strategy, usually an unpruned algorithms (for example a decision tree) is applied to different parts of the dataset. Then the result is combined. Usually using a mean.

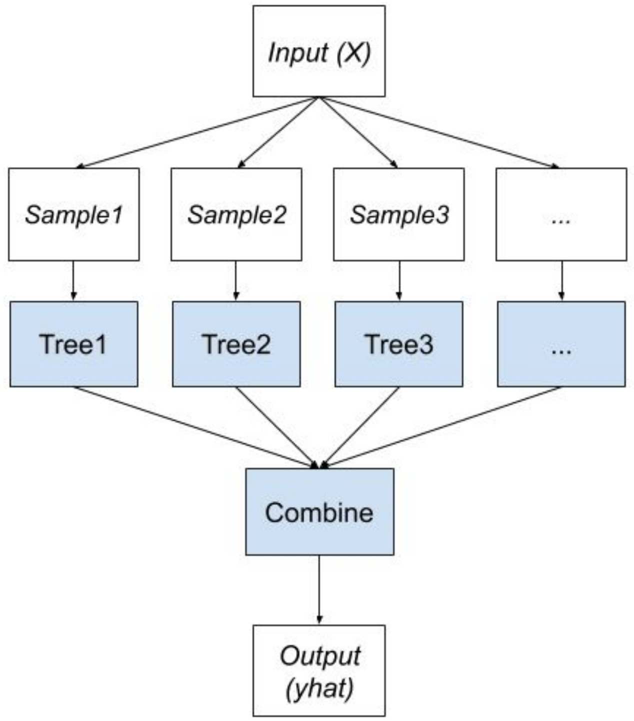
This kind of method works because by each weak learner is trained with datasets with bigger variance. One important part to note is that the sampling part is done with replacement.

Also, the combine algorithms can change.

Algo:

- Random Forest
- Bagged Decision Trees
- Extra Trees

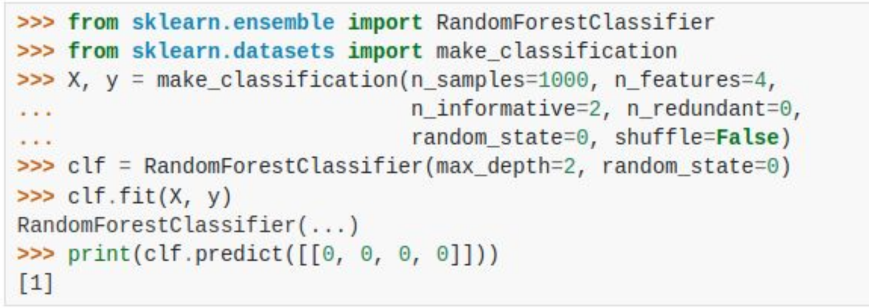
Bagging Ensemble



Random Forests

Implementation using Decision Trees. So most parameters are the same.

Random Forest Explanation



```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                            n_informative=2, n_redundant=0,
...                            random_state=0, shuffle=False)
>>> clf = RandomForestClassifier(max_depth=2, random_state=0)
>>> clf.fit(X, y)
RandomForestClassifier(...)
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

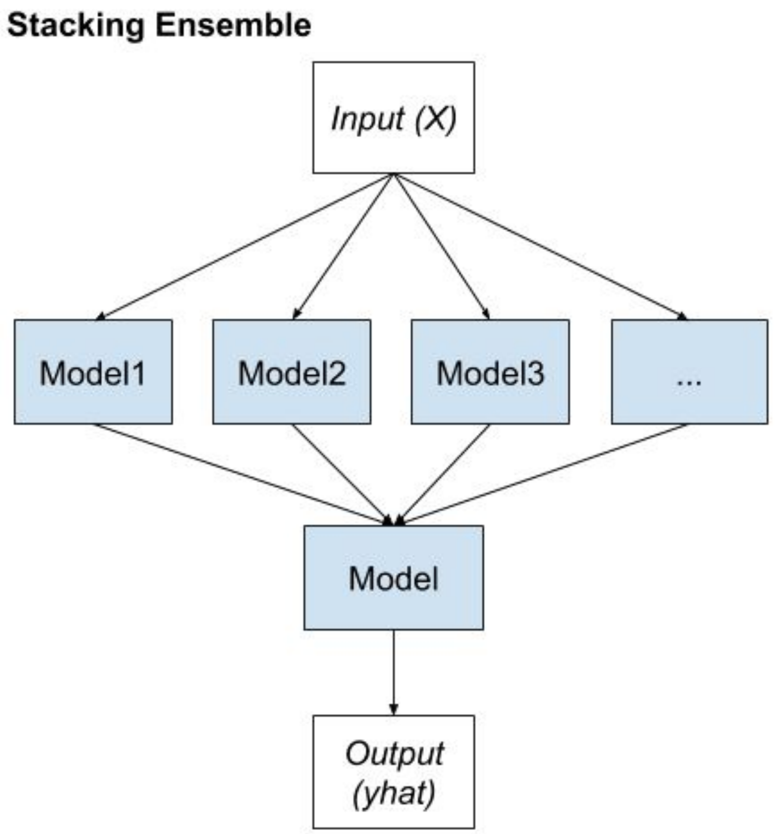
https://www.youtube.com/watch?v=J4Wdy0Wc_xQ&ab_channel=StatQuestwithJoshStarmer

Stacking Algorithms

In this strategy, we seek to train different models with the same data. The results are then combined using a new model. This kinda like a more complex voting system. Normally the decision layer is a linear model, but other such as logistic regression can be applied.

This can be extended with more layers.

The all point of this is increase the diversity of learning. So for that is desirable to use as much different as possible algorithms.



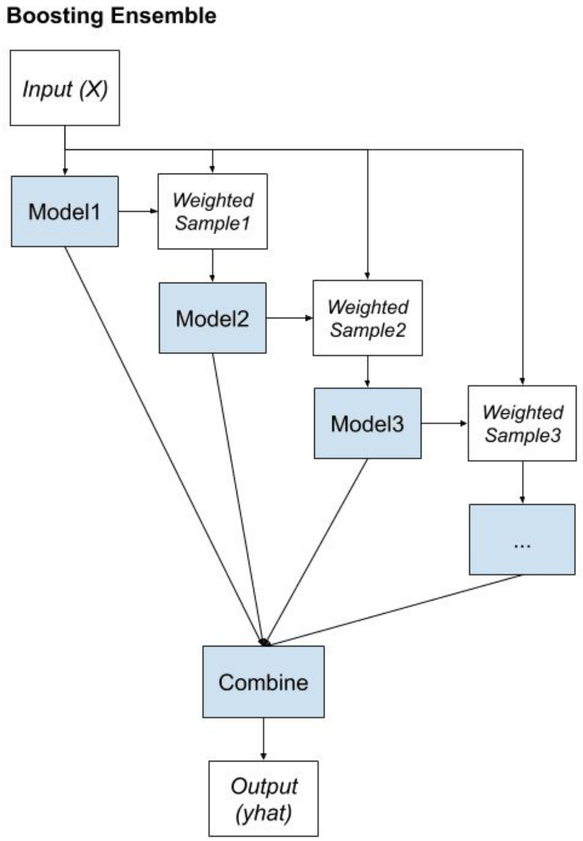
Boosting Algorithms

In this strategy, the trained dataset is changed depending on the results of each iteration. The idea, is to train a new model with the labels that our model has less certainty about. So, the point is to fix the prediction errors. A very common model used in this approach are decision three, that are applied in cascade.

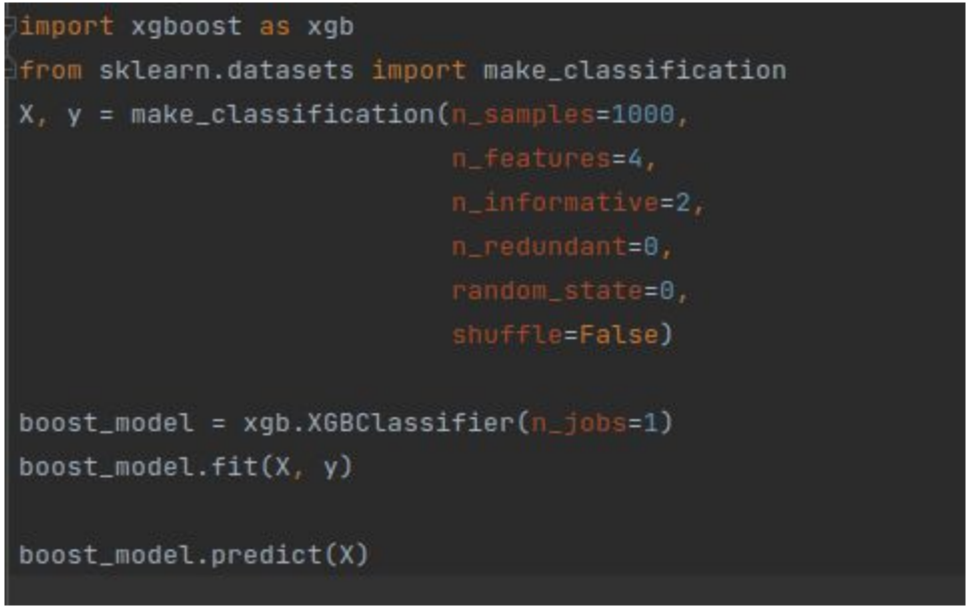
The name of the strategy comes from boosting weak learners and build a strong learner.

Some algorithms are:

- AdaBoost
- Gradient Boosting Machines
- Stochastic Gradient Boost (XGboost)



XGBoost

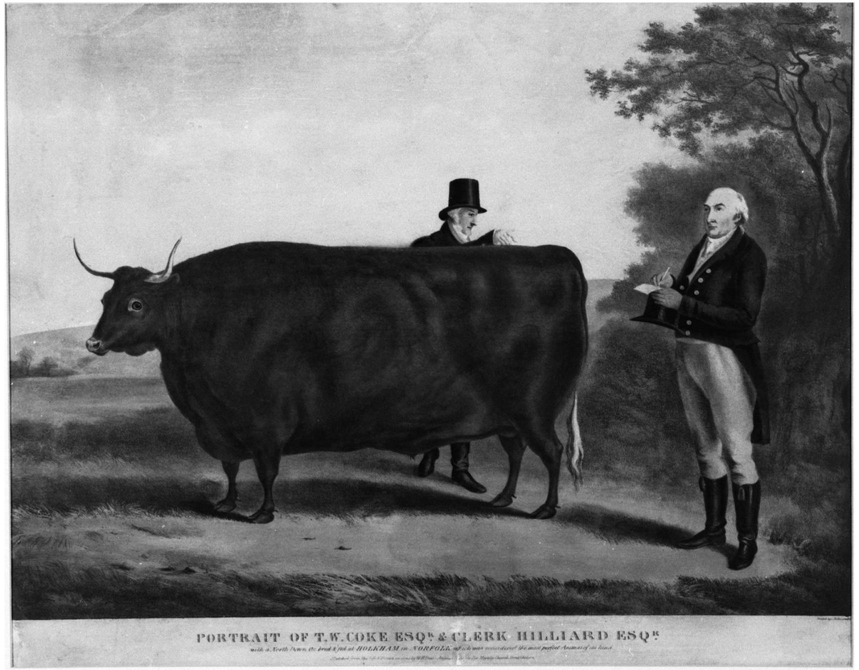


```
import xgboost as xgb
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000,
                           n_features=4,
                           n_informative=2,
                           n_redundant=0,
                           random_state=0,
                           shuffle=False)

boost_model = xgb.XGBClassifier(n_jobs=1)
boost_model.fit(X, y)

boost_model.predict(X)
```

https://www.youtube.com/watch?v=8b1JEDvenQU&ab_channel=StatQuestwithJoshStarmer
25 min video

Wisdom of the crowds

In 1907, during a bovine weight guessing competition in an England county fair, the famous statistician and scientist Sir Francis Galton realized the aggregated predictions of many non-expert individuals could estimate the actual weight of the bovine with exceptional precision, much better than any single expert could do.



KNN

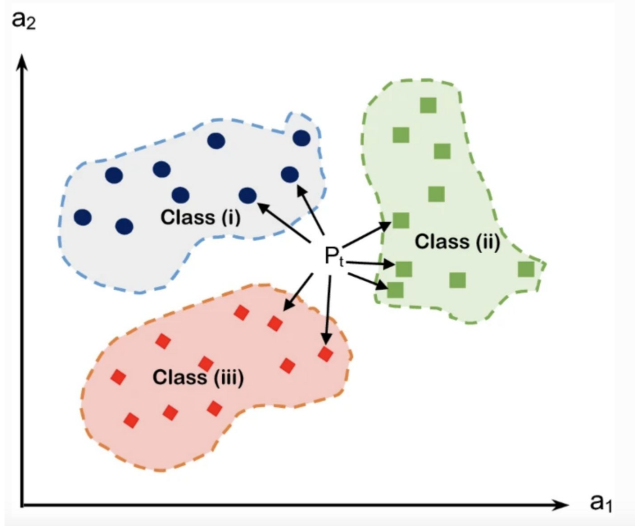
K Nearest Neighbours where k is the number of groups.

KNN assumes that related data is close to each other. This can be computed using distances between the data points. An example of this distances is the Euclidean distance.

$$\sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)}$$

The most common parameters that are required to finetune is the k and the distance metric.

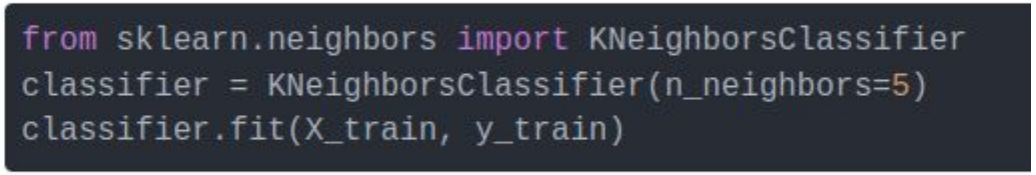
As disadvantage this algorithm does not work with highly dimensional data because the distance computations becomes harder.



https://www.youtube.com/watch?v=HVXime0nQeI&ab_channel=StatQuestwithJoshStarmer

Let’s apply this models in our titanic dataset

Example implementation



```python
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

Let’s talk about more data preprocessing techniques

So far in our example, we haven't talked about the effect of label balance in the outcome of the results. Neither, the data absolute value impact in the model. In order to understand that, we are going to talk about

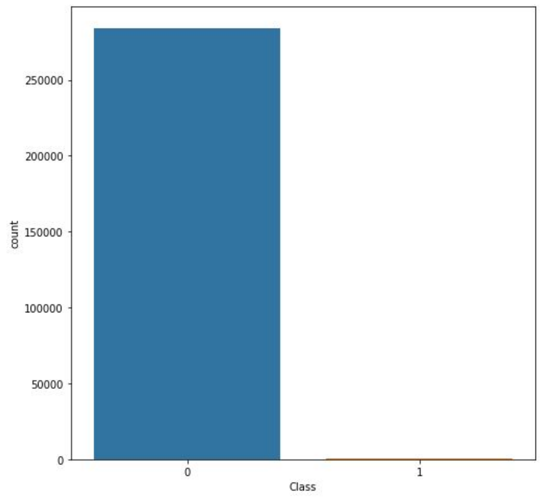
- Dataset imbalance
- Data scaling
  - Data normalization
  - Data Standardization

Data imbalance

Data imbalance happens when one of the predict labels dominates the other ones. This is also referred as having a results distribution skewed towards a particular value.

Typical use cases of this:

- Predictive maintenance
- Disease detection algorithms



There are several techniques to solve depending on what is causing the imbalance:

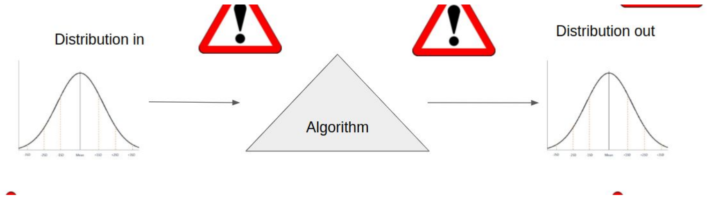
- Collect more data
- Change performance metric
- Resample dataset
- Generate Synthetic Samples
- Different algorithms
- Penalized Models
- Different Approach

Collect more data

If we recall the slide of the previous lesson, the all idea of machine learning is to learn the distribution of the input data.

If the imputed data is very skews, this might mean that our sample is not representative of the population we want to study.

So, it leads to issues of label balance. Collecting more data can fix this. This for example can happen, if the data acquisition phases involves physical processes. So issues on the equipment may justify the imbalance.

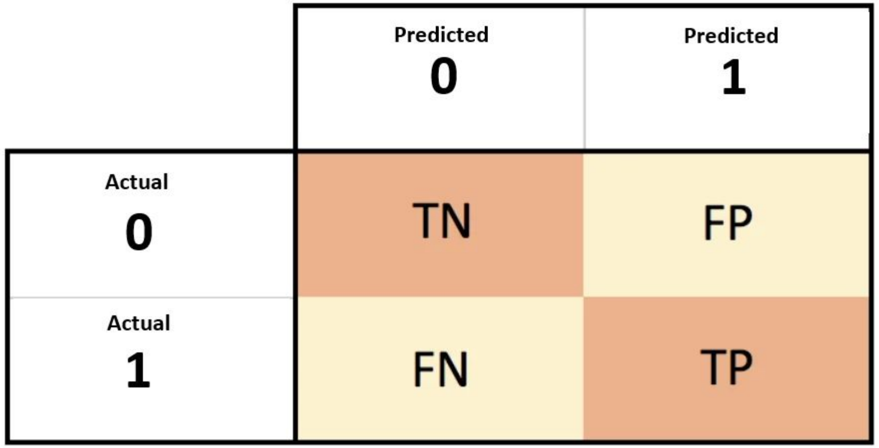


Change performance metric

As we have seen, accuracy is not the best metric to evaluate an imbalance dataset. This is because, for 90% positive class distribution, if we use a random model, we will have an accuracy around 90% without any effort.

I would recommend the following metrics:

- Precision
- Recall
- F1 score
- ROC curve (AUC)



	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

Resample dataset

This technique while very simple, can be very powerful. There are two approaches:

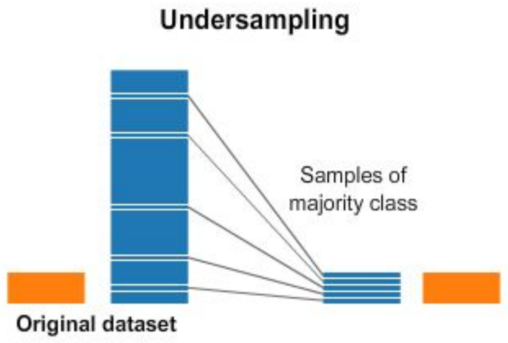
- Undersampling

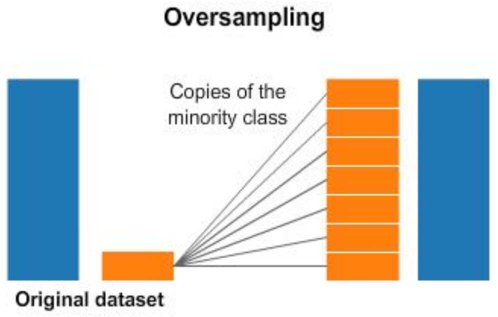
In this technique we select only a few labels of the most abundant label in order to ensure a good balance. 50/50 is not mandatory

- Oversampling

This technique replicates the minority class n amount of times until we achieve nice balance.

```
%sh pip install imbalanced-learn
from imblearn.under_sampling import RandomUnderSampler
```

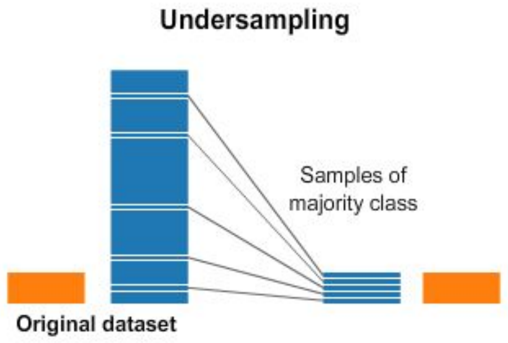


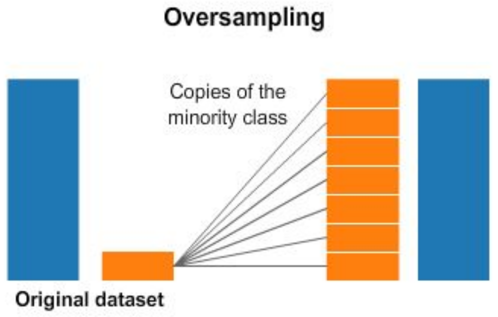


Resample dataset

Tips:

- Consider testing under-sampling when you have an a lot data
- Consider testing over-sampling when you don't have a lot of data
- Consider testing random and non-random (e.g. stratified) sampling schemes.
- Consider testing different resampled ratios





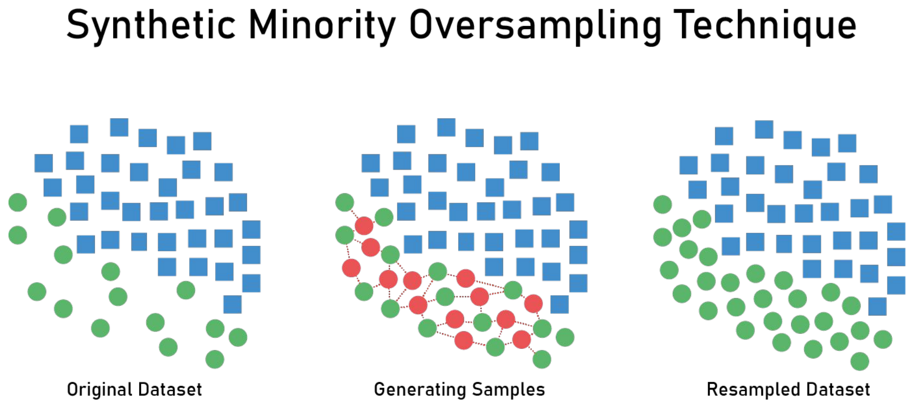
Generate Synthetic Samples

In this approach the idea is to create synthetic data to balance the minority class. The biggest difference compared with oversampling is the fact of having new but different data.

One of the most popular techniques is called SMOTE. This algorithm uses clustering (using a distance method) to separate the data into clusters, and generate data for the same cluster.

Implementation:

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(ratio='minority')

X_sm, y_sm = smote.fit_sample(X, y)
```



Can someone tell what are the dangerous of generating synthetic data?
Tip: is related with distributions

Pt startup https://www.ydata.ai/

Different algorithms

Here, the rationale is based on the way the models are built that make some to handle better the imbalance labels when compared to other. This is also dependent on the data, so one model amy work in one dataset, but do not work in another.

Typically, models created based on Decision Tree tend to work well.

An alternative is to use models that penalize more the mistakes made on the minority class. This can be done either by designing custom loss functions or by choosing model that have this implemented by design.

Examples:

- Penalized-SVM
- Penalized-LDA

Normally this is implemented by tuning the keyword `class-weight`

Let’s practice

1. Create a new conda environment
2. Load credit card csv
3. EDA (Exploratory data analysis) of the predict label
4. If we created a random model what would be our accuracy?
5. Apply undersampling and plot results
6. Apply oversampling using SMOTE
7. Compare Random Forest Results with vanilla data, with undersampling and with SMOTE.
8. What is the best metric to evaluate our dataset

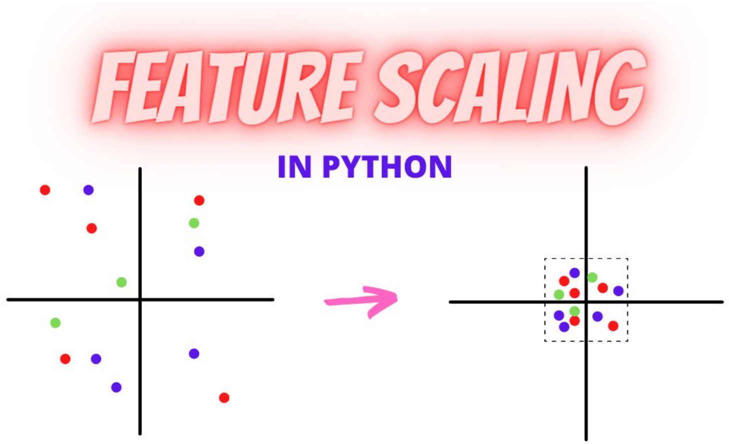
Data Scaling

In machine learning the size matters! Why?

The all point of data normalization is to eliminate the effect of data size in the prediction. Imagine for example that you want to predict the age of someone given the height and the weight. Do you think it makes difference if we use for example meters or centimeters?

There are two main methods:

- Data normalization
- Data scaling



Data normalization

This consists on transforming the data in a normally distributed distribution. This is extremely important because most models require data to be normally distributed.

A second reason is the fact of some model use linear space i. most distance base methods like KNN, linear regression, K-means. So under this circumstances we need to scale the data into the same space.

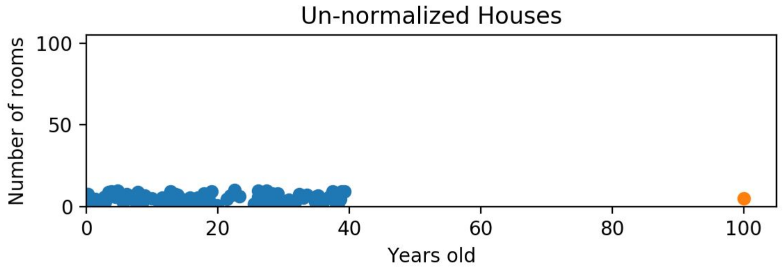
This is done by applying the formula.

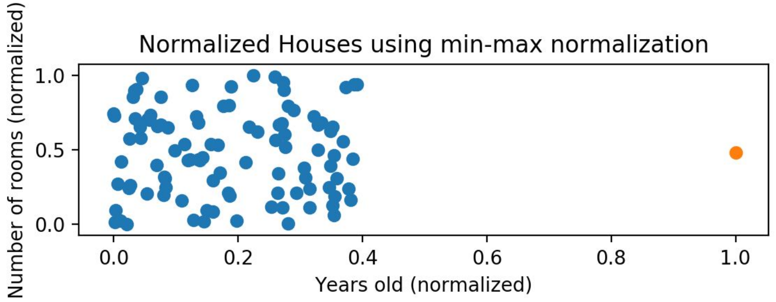
$$z = \frac{x - \mu}{\sigma}$$

In python this is done like:

```
from sklearn.preprocessing import
StandardScaler
```

```
ss = StandardScaler()
```





Effect of normalization on data distribution

Min-Max Scale

This method is perhaps the most simple, but most of the times ML practitioners tends to forget.

In python that can be done by:

```
from sklearn.preprocessing import MinMaxScaler

data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]

scaler = MinMaxScaler()

scaler.fit_transform(data)
```

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Data preprocessing final considerations

Q. What are the impacts that you see in the production pipelines about the application of scaling methods?

A. The parameters of the scaling in the training data must be passed to the test dataset. Why?

A. Because the assumption is that both train and test dataset belong to the same distribution, so the parameters are the same.

More preprocessing methods can be found here:

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing

Let’s practice

1. Use the scikit-learn package in order to download the wine dataset
2. Plot the distribution of the magnesium column with and without min-max scaling
3. Plot the distribution of the magnesium column with and without normalization
4. Let’s apply a vanilla knn and compare the results