Work with Missing value, Outlier, Unbalanced Dataset

Table of Contents

- 1 Imports and Dataset
- 2 Sampler, transformer and estimator
- 3 Lab 1: Missing value
- 4 Outlier removal
- 5 Unbalance dataset

Imports and Dataset

In [ ]:

```
#import warnings
#warnings.filterwarnings('ignore')
```

In [ ]:

```
from tqdm import tqdm
import seaborn as sns                                    # For plotting data
import pandas as pd                                      # For dataframes
import numpy as np
import matplotlib.pyplot as plt                          # For plotting data
%matplotlib inline

# For splitting the dataset
from sklearn.model_selection import train_test_split

# For setting up pipeline
from imblearn.pipeline import Pipeline
from imblearn import FunctionSampler
from sklearn.preprocessing import FunctionTransformer
from sklearn.base import BaseEstimator, TransformerMixin

# For Missing data
from sklearn.impute import SimpleImputer
```

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# For Outlier detection
from sklearn.ensemble import IsolationForest
from sklearn.covariance import EllipticEnvelope
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM

# For Unbalanced dataset
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler

# For classification
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier

# For optimization
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error
```

The **original ForestCover/Covertype dataset** from UCI machine learning repository is a multiclass classification dataset. This dataset contains tree observations from four areas of the Roosevelt National Forest in Colorado. This study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances, so that existing forest cover types are more a result of ecological processes rather than forest management practices.

In this notebook you are asked to predict the forest cover type (the predominant kind of tree cover) from strictly cartographic variables (as opposed to remotely sensed data). The actual forest cover type for a given 30 x 30 meter cell was determined from US Forest Service (USFS) Region 2 Resource Information System data. Independent variables were then derived from data obtained from the US Geological Survey and USFS. The data is in raw form (not scaled) and contains binary columns of data for qualitative independent variables such as wilderness areas and soil type.

This dataset has 54 attributes :

- 10 quantitative variables,
- 4 binary wilderness areas
- and 40 binary soil type variables). Here, outlier detection dataset is created using only 10 quantitative attributes. Instances from class 2 are considered as normal points and instances from class 4 are anomalies. The anomalies ratio is 0.9%. Instances from the other classes are omitted.

Dataset description available on Kaggle.

- Elevation: Elevation in meters.
- Aspect: Aspect in degrees azimuth.
- Slope: Slope in degrees.
- Horizontal_Distance_To_Hydrology: Horizontal distance in meters to nearest surface water features.
- Vertical_Distance_To_Hydrology: Vertical distance in meters to nearest surface water features.
- Horizontal_Distance_To_Roadways: Horizontal distance in meters to the nearest roadway.
- Hillshade_9am: hillshade index at 9am, summer solstice. Value out of 255.
- Hillshade_Noon: hillshade index at noon, summer solstice. Value out of 255.
- Hillshade_3pm: shade index at 3pm, summer solstice. Value out of 255.
- Horizontal_Distance_To_Fire_Point*: horizontal distance in meters to nearest wildfire ignition points.
- Wilderness_Area#: wilderness area designation.
- Soil_Type#: soil type designation.

Wilderness_Area feature is one-hot encoded to 4 binary columns (0 = absence or 1 = presence), each of these corresponds to a wilderness area designation. Areas are mapped to value in the following way:

1. Rawah Wilderness Area
2. Neota Wilderness Area
3. Comanche Peak Wilderness Area
4. Cache la Poudre Wilderness Area

The same goes for Soil_Type feature which is encoded as 40 one-hot encoded binary columns (0 = absence or 1 = presence) and each of these represents soil type designation. All the possible options are:

1. Cathedral family - Rock outcrop complex, extremely stony
2. Vanet - Ratake families complex, very stony
3. Haploborolis - Rock outcrop complex, rubbly
4. Ratake family - Rock outcrop complex, rubbly
5. Vanet family - Rock outcrop complex complex, rubbly
6. Vanet - Wetmore families - Rock outcrop complex, stony
7. Gothic family

8. Supervisor - Limber families complex
9. Troutville family, very stony
10. Bullwark - Catamount families - Rock outcrop complex, rubbly
11. Bullwark - Catamount families - Rock land complex, rubbly.
12. Legault family - Rock land complex, stony
13. Catamount family - Rock land - Bullwark family complex, rubbly
14. Pachic Argiborolis - Aquolis complex
15. ¨unspecified in the USFS Soil and ELU Survey
16. Cryaquolis - Cryoborolis complex
17. Gateview family - Cryaquolis complex
18. Rogert family, very stony
19. Typic Cryaquolis - Borohemists complex
20. Typic Cryaquepts - Typic Cryaquolls complex
21. Typic Cryaquolls - Leighcan family, till substratum complex
22. Leighcan family, till substratum, extremely bouldery
23. Leighcan family, till substratum - Typic Cryaquolls complex
24. Leighcan family, extremely stony
25. Leighcan family, warm, extremely stony
26. Granile - Catamount families complex, very stony
27. Leighcan family, warm - Rock outcrop complex, extremely stony
28. Leighcan family - Rock outcrop complex, extremely stony
29. Como - Legault families complex, extremely stony
30. Como family - Rock land - Legault family complex, extremely stony
31. Leighcan - Catamount families complex, extremely stony
32. Catamount family - Rock outcrop - Leighcan family complex, extremely stony
33. Leighcan - Catamount families - Rock outcrop complex, extremely stony
34. Cryorthents - Rock land complex, extremely stony
35. Cryumbrepts - Rock outcrop - Cryaquepts complex
36. Bross family - Rock land - Cryumbrepts complex, extremely stony
37. Rock outcrop - Cryumbrepts - Cryorthents complex, extremely stony
38. Leighcan - Moran families - Cryaquolls complex, extremely stony

39. Moran family - Cryorthents - Leighcan family complex, extremely stony
40. Moran family - Cryorthents - Rock land complex, extremely stony

Cover_Type: forest cover type designation, its possible values are between 1 and 7, mapped in the following way:

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

We will use a very small part of this dataset with only classes 1 and 7.

In [ ]:
```
import ssl

ssl._create_default_https_context = ssl._create_unverified_context

url = "https://www.i3s.unice.fr/~riveill/dataset/covtype/"
filename = "covtype.csv"
```

In [ ]:
```
# load train and test
train = pd.read_csv(url+"train.csv", delimiter=',')
test = pd.read_csv(url+"test.csv", delimiter=',')
```

In [ ]:
```
columns = list(train.columns)
target = 'Cover_Type'
columns.remove(target)
cat_columns=[c for c in columns if 'Soil_Type' in c or 'Wilderness_Area' in c] # already one hot encode
num_columns=[c for c in columns if c not in cat_columns]
```

In [ ]:
```
y_train = np.array(train[target]).reshape(-1,1)
X_train = train[columns]
```

```
y_test = np.array(test[target]).reshape(-1,1)
X_test = test[columns]

X_train.shape, X_test.shape
```

Out[ ]: ((10000, 54), (10000, 54))

In [ ]:
```
# Class distribution
distribution = pd.Series(y_train.flatten()).value_counts().to_dict()
distribution
```

Out[ ]: {1: 9083, 7: 917}

Sampler, transformer and estimator

There are three types of objects in imblearn/scikit-learn design:

Transformer transform observation (modify only X_train) and implements:

- fit: used for calculating the initial parameters on the training data and later saves them as internal objects state.
- transform: Use the initial above calculated values and return modified training data as output. Do not modify the length of the dataset.

Predictor is a "model" and implements:

- fit: calculates the parameters or weights on the training data and saves them as an internal object state.
- predict: Use the above-calculated weights on the test data to make the predictions.

Sampler is a new element, from imblearn library. A sampler modifies the number of observations in the train set (modify X_train and y_train) and implements:

- fit_resample

The following cells build a pipeline

In [ ]:
```
# A sampler
class mySampler(BaseEstimator):
    def fit_resample(self, X, y):
```

```
        data = np.concatenate((X, y), axis=1)
        # remove rows with NaN
        data = data[~np.isnan(data).any(axis=1), :]
        return data[:,:-1], data[:,-1]
```

It's also possible to build sampler from a function

In [ ]:

```
def mySamplerFunction(X, y, conta=0.1):
    iforest = IsolationForest(n_estimators=300, max_samples='auto', contamination=conta)
    outliers = iforest.fit_predict(X, y)

    X_filtered = X[outliers == 1]
    y_filtered = y[outliers == 1]

    return X_filtered, y_filtered
```

In [ ]:

```
# A transformer
class myTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, strategy="most_frequent"):
        self.strategy = strategy
        self.sample = SimpleImputer(strategy=self.strategy)
    def fit(self, X, y=None):
        return self.sample.fit(X, y)
    def transform(self, X):
        return self.sample.transform(X)
```

Like sampler, it's also possible to build transformer from a function see `sklearn.preprocessing.FunctionTransform`

In [ ]:

```
# A predictor
class myPredictor(BaseEstimator):
    def __init__(self, penalty="l2"):
        self.penalty = penalty
        self.sample = LogisticRegression(solver="lbfgs", penalty=self.penalty, max_iter=10000)
    def fit(self, X, y):
        return self.sample.fit(X, y)
    def predict(self, X):
        return self.sample.predict(X)
```

In [ ]:

```
# Different version of the 2 steps pipeline
```

```
# step 1 : remove or imput missing data
# step 2 : remove outlier
# step 3 : predictor
pipeline = Pipeline([('missing_data', None),
                     ('outlier', FunctionSampler(func=mySamplerFunction)),
                     ('clf', None)])

parameters = [{'missing_data': [mySampler()],
               'clf': [myPredictor()],
               'clf__penalty': ['none'],
              },
              {'missing_data': [myTransformer()],
               'missing_data__strategy': ['most_frequent'],
               'clf': [myPredictor()],
               'clf__penalty': ['none'],
              },
              ]

pipeline
```

Out[ ]:
```
Pipeline(steps=[('missing_data', None),
                ('outlier',
                 FunctionSampler(func=<function mySamplerFunction at 0x7fbb13f97280>)),
                ('clf', None)])
```

In [ ]:
```
# GridSearch with pipeline
grid = GridSearchCV(pipeline, parameters, cv=2,
                    scoring="f1_micro", refit=True,
                    verbose=2)
grid
```

Out[ ]:
```
GridSearchCV(cv=2,
             estimator=Pipeline(steps=[('missing_data', None),
                                       ('outlier',
                                        FunctionSampler(func=<function mySamplerFunction at 0x7fbb13f97280>)),
                                       ('clf', None)]),
             param_grid=[{'clf': [myPredictor()], 'clf__penalty': ['none'],
                          'missing_data': [mySampler()]},
                         {'clf': [myPredictor()], 'clf__penalty': ['none'],
                          'missing_data': [myTransformer()],
                          'missing_data__strategy': ['most_frequent']}],
             scoring='f1_micro', verbose=2)
```

Remember: samplers are only called to perform the "fit" and not to perform the predict. If the data set contains missing values (NaN) in the validation part, a warning may be raised.

In [ ]:
```
# Try to find the best model
grid.fit(X_train[:50], y_train[:50]) # Some data for testing the process... but use all available data
```

```
Fitting 2 folds for each of 2 candidates, totalling 4 fits
```

```
/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:683: UserWarning: Scori
ng failed. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 674, in
_score
    scores = scorer(estimator, X_test, y_test)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_scorer.py", line 199, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_scorer.py", line 236, in _score
    y_pred = method_caller(estimator, "predict", X)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_scorer.py", line 53, in _cached_call
    return getattr(estimator, method)(*args, **kwargs)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/metaestimators.py", line 120, in <lambda
>
    out = lambda *args, **kwargs: self.fn(obj, *args, **kwargs)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/pipeline.py", line 419, in predict
    return self.steps[-1][-1].predict(Xt, **predict_params)
  File "<ipython-input-12-2937ca012408>", line 9, in predict
    return self.sample.predict(X)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_base.py", line 309, in predict
    scores = self.decision_function(X)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_base.py", line 284, in decision_
function
    X = check_array(X, accept_sparse='csr')
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py", line 63, in inner_f
    return f(*args, **kwargs)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py", line 663, in check_array
    _assert_all_finite(array,
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py", line 103, in _assert_all
_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
[CV] END clf=myPredictor(), clf__penalty=none, missing_data=mySampler(); total time=   0.5s
```

```
/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py:683: UserWarning: Scori
ng failed. The score on this train-test partition for these parameters will be set to nan. Details:
Traceback (most recent call last):
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_validation.py", line 674, in
_score
    scores = scorer(estimator, X_test, y_test)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_scorer.py", line 199, in __call__
    return self._score(partial(_cached_call, None), estimator, X, y_true,
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_scorer.py", line 236, in _score
    y_pred = method_caller(estimator, "predict", X)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_scorer.py", line 53, in _cached_call
    return getattr(estimator, method)(*args, **kwargs)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/metaestimators.py", line 120, in <lambda
>
    out = lambda *args, **kwargs: self.fn(obj, *args, **kwargs)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/pipeline.py", line 419, in predict
    return self.steps[-1][-1].predict(Xt, **predict_params)
  File "<ipython-input-12-2937ca012408>", line 9, in predict
    return self.sample.predict(X)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_base.py", line 309, in predict
    scores = self.decision_function(X)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_base.py", line 284, in decision_
function
    X = check_array(X, accept_sparse='csr')
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py", line 63, in inner_f
    return f(*args, **kwargs)
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py", line 663, in check_array
    _assert_all_finite(array,
  File "/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.py", line 103, in _assert_all
_finite
    raise ValueError(
ValueError: Input contains NaN, infinity or a value too large for dtype('float64').

  warnings.warn(
[CV] END clf=myPredictor(), clf__penalty=none, missing_data=mySampler(); total time=   0.5s
[CV] END clf=myPredictor(), clf__penalty=none, missing_data=myTransformer(), missing_data__strategy=most_frequent; tota
l time=   0.5s
[CV] END clf=myPredictor(), clf__penalty=none, missing_data=myTransformer(), missing_data__strategy=most_frequent; tota
l time=   0.5s
/Users/riveill/opt/anaconda3/lib/python3.8/site-packages/sklearn/model_selection/_search.py:918: UserWarning: One or mo
re of the test scores are non-finite: [ nan 0.66]
  warnings.warn(
```

Out[ ]:

```
GridSearchCV(cv=2,
```

```
             estimator=Pipeline(steps=[('missing_data', None),
                                       ('outlier',
                                        FunctionSampler(func=<function mySamplerFunction at 0x7fbb13f97280>)),
                                       ('clf', None)]),
             param_grid=[{'clf': [myPredictor()], 'clf__penalty': ['none'],
                          'missing_data': [mySampler()]},
                         {'clf': [myPredictor(penalty='none')],
                          'clf__penalty': ['none'],
                          'missing_data': [myTransformer()],
                          'missing_data__strategy': ['most_frequent']}],
             scoring='f1_micro', verbose=2)
```

In [ ]:

```python
# Evaluate the model with the whole dataset
y_pred = grid.predict(X_train[:500])
print("Best: {:.2f} using {}".format(
    grid.best_score_,
    grid.best_params_
))
print('Test set score: ' + str(grid.score(X_train[:500], y_train[:500])))
```

```
Best: 0.66 using {'clf': myPredictor(penalty='none'), 'clf__penalty': 'none', 'missing_data': myTransformer(), 'missing
_data__strategy': 'most_frequent'}
Test set score: 0.868
```

Lab 1: Missing value

$$[TODO - Students]$$

Test some algorithms to handle missing data.

- Choose the classifier that you think is preferable for this job.

1. with removal of missing data
2. with of the following imputation methods
   - With SimpleImputer
   - With IterativeImputer
   - With KNNimputer

Build a 2 step pipeline and use a gridsearch to find the right hyperpameters.

Submit your work in the form of an executable and commented notebook at lms.univ-cotedazur.fr

Outlier removal

Removing the outliers modifies the data set, so it is a sampler.

IsolationForest or other sklearn detector are not a sampler. You have to read the imblearn documentation

A small example with parameters:

```
from collections import Counter
from imblearn.under_sampling import RandomUnderSampler

def func(X, y, sampling_strategy, random_state):
  return RandomUnderSampler(
      sampling_strategy=sampling_strategy,
      random_state=random_state).fit_resample(X, y)

sampler = FunctionSampler(func=func,
                          kw_args={'sampling_strategy': 'auto',
                                   'random_state': 0})
X_res, y_res = sampler.fit_resample(X, y)
print(f'Resampled dataset shape {sorted(Counter(y_res).items())}')
```

$$[TODO - Students]$$

Test some algorithms to handle outliers.

- Choose the classifier that you think is preferable for this job.

1. Without taking any precautions
2. By eliminating outliers with one of the following approaches:
   - With Isolation Forest (IF)
   - With Local Outlier Factor (LOF)
   - With Minimum Covariance Determinant (MCD)

Build a 3 step pipeline and use a gridsearch to find the right hyperpameters. The first step, is your best previous "missing data method".

Submit your work in the form of an executable and commented notebook at lms.univ-cotedazur.fr

Unbalance dataset

$$[TODO - Students]$$

Test some algorithms to work with unbalanced dataset. Choose the classifier that you think is preferable for this job.

1. Without taking any precautions
2. With modification of the dataset by Over sampling or Under sampling or SMOTE
3. Without modification of the dataset by weight

Build a 4 step pipeline and use a gridsearch to find the right hyperpameters and use a gridsearch to find the right hyperpameters. The first and second step, is your best previous methods.

Submit your work in the form of an executable and commented notebook at lms.univ-cotedazur.fr

In [ ]:

```

```