# *Unit 6 Assignment: PyCaret*

Juliana Noronha

University of Maryland, Global Campus

DATA 660: Advanced Topics in Machine Learning

Dr. Ed Herranz

April 22, 2025

**PyCaret Implementation**

Over the course of three units, several machine learning strategies have been applied to the same Portuguese Bank Marketing dataset. To briefly summarize, the original dataset has 3616 observations and an unbalanced target variable, where there are far fewer instances of customers subscribing to deposits (“yes” or 1) than there are instances of the opposite (“no” or 0). Previously, SVMs and ensemble models have been fit to this dataset, with varying degrees of success. For this installment, PyCaret (a library to easily carry out AutoML) is used to quickly fit 5 models.

PyCaret is initialized with the setup function, which has only two required parameters: data and target (Initialize | Docs, 2023). However, given the known issues with target class imbalance, the parameter: `fix_imbalance = True` is specified. When this parameter is set to `True`, the default method to handle class imbalance is SMOTE (Data Preparation | Docs, 2023). The class imbalance handling can be further specified, but SMOTE is a reasonable way to handle target class imbalance in this situation. Additionally, the optional `train_size` parameter is set to 0.8. This is the proportion of the dataset to be used for training and validation (Other setup parameters | Docs, 2023).

Figure 1 shows the output of PyCaret’s setup function. From the summary, the automated data preparation took the dataset from 3616 rows x 17 columns to 5832 rows x 49 columns. Inspecting the logs (Figure 2), it is clear that the column dimension increased from one-hot-encoding of the categorical variables and the row dimension increased from SMOTE.

Five models (Logistic Regression, Random Forest, LightGBM, XGBoost, and Linear SVM) were successfully trained using PyCaret without errors.

**Results Comparison (PyCaret vs. Ensemble Models)**

| | Model | Accuracy | Sensitivity | Specificity | Balanced Accuracy |
|---|---|---|---|---|---|
| Unit 3: SVMs | *Linear SVM (Spark)* | 88% | 0% | 100% | 50% |
| | *Non-linear SVM (SciPy)* | 84% | 33% | 90% | 61% |
| Unit 5: Ensemble Models | *Random Forest (untuned)* | 89% | 17% | 98% | 58% |
| | *Random Forest (tuned)* | 90% | 11% | 99% | 55% |
| | *Gradient Boosting (untuned)* | 89% | 21% | 97% | 59% |
| | *Gradient Boosting (tuned)* | 43% | 72% | 39% | 56% |
| | *Stacked Model (RF + GB + DT)* | 88% | 20% | 97% | 59% |
| Unit 6: PyCaret | *Logistic Regression* | 83% | 74% | 84% | 79% |
| | *Random Forest* | 90% | 21% | 98% | 59% |
| | *LGBM* | 90% | 37% | 96% | 66% |
| | *XGB* | 90% | 44% | 96% | 70% |
| | *Linear SVM* | 44% | 97% | 38% | 67% |

The above table summarizes the results across all model fitting experiments over the last three units. Green cells signify the highest results in each column (i.e. for each metric), while red cells signify the lowest results in each column. Some models stand out as dealing with the class imbalance somewhat poorly -- both SVMs from Unit 3 (although, no techniques were applied to handle the class imbalance for those models), all Random Forest models, the untuned Gradient Boosting model and the stacked model from Unit 5.

Of course, the “best” model depends on which metric one optimizes on. In this case, it is more valuable to accurately estimate “yeses” than “nos”, so high sensitivities are desirable. In that sense, PyCaret’s Linear SVM may have handled the class imbalance best, followed by

PyCaret's Logistic Regression. However, PyCaret's Logistic Regression struck the best balance between specificity and sensitivity by far, so that model may be preferable if overall accuracy is desired.

Although SMOTE was applied to both the manually fitted ensemble models and the PyCaret models, the latter still exhibited superior sensitivity. This could suggest that PyCaret's model architecture and preprocessing pipelines did a better job at handling class imbalance overall, or it could be a symptom of AutoML variance, where the randomly selected search space happened to produce improved results upon first run (AutoML: Benefits and limitations).

**AutoML: Advantages and Disadvantages**

AutoML has proven to produce highly accurate models when compared with the manually trained models from previous units. In fact, both of the "best" models were produced by PyCaret's AutoML and did so in a fraction of the time. The time-savings and ease of use are easily the most alluring advantages of this ML framework. The foundational pillars of `setup()`, `compare_models()` and `predict_model()` also means it is incredibly straightforward for nascent data scientists to use or for teams without engineering expertise. This would normally need many more lines of code and purposeful decision-making. The speed and ease also make AutoML a great choice for initially screening several types of models (AutoML: Benefits and limitations, n.d.).

However, with ease of use comes obscured complexity. An ML engineer who is interested in higher degrees of customization might find AutoML frameworks frustrating due to an inability to see how the model search occurred and an inability to customize models during training. Additionally, using a generalized optimization algorithm means that there are limits to how much the approach can be tuned to the specific dataset, and therefore ML experts are likely

able to make better models using specific expertise and fine-tuning strategies (AutoML: Benefits and limitations, n.d.) From an ethical AI perspective, the abstraction provided by AutoML can also obscure potential sources of model bias or fairness issues, especially if users rely on default settings without scrutinizing how models treat different subgroups in the data.

**References**

*AutoML: Benefits and limitations. (n.d.).* Google for Developers. *https://developers.google.com/machine-learning/crash-course/automl/benefits-limitations*

*Data Preparation | Docs*. (2023). https://pycaret.gitbook.io/docs/get-started/preprocessing/data-preparation

*Initialize | Docs*. (2023). https://pycaret.gitbook.io/docs/get-started/functions/initialize

*Other setup parameters | Docs*. (2023). https://pycaret.gitbook.io/docs/get-started/preprocessing/other-setup-parameters

## Appendix

Figure 1: PyCaret Setup Summary

```
# Initialize PyCaret setup
clf = setup(data=train_data, target='target',
            fix_imbalance = True, train_size=0.8, session_id=42)
```

| | Description | Value |
|---|---|---|
| 0 | Session id | 42 |
| 1 | Target | target |
| 2 | Target type | Binary |
| 3 | Target mapping | no: 0, yes: 1 |
| 4 | Original data shape | (3616, 17) |
| 5 | Transformed data shape | (5832, 49) |
| 6 | Transformed train set shape | (5108, 49) |
| 7 | Transformed test set shape | (724, 49) |
| 8 | Numeric features | 7 |
| 9 | Categorical features | 9 |
| 10 | Preprocess | True |
| 11 | Imputation type | simple |
| 12 | Numeric imputation | mean |
| 13 | Categorical imputation | mode |
| 14 | Maximum one-hot encoding | 25 |
| 15 | Encoding method | None |
| 16 | Fix imbalance | True |
| 17 | Fix imbalance method | SMOTE |
| 18 | Fold Generator | StratifiedKFold |
| 19 | Fold Number | 10 |
| 20 | CPU Jobs | -1 |
| 21 | Use GPU | False |
| 22 | Log Experiment | False |
| 23 | Experiment Name | clf-default-name |
| 24 | USI | 3d14 |

Figure 2: PyCaret Setup Steps -- Log

```
steps=[('label_encoding',
        TransformerWrapperWithInverse(exclude=None, include=None,
                                      transformer=LabelEncoder())),
       ('numerical_imputer',
        TransformerWrapper(exclude=None,
                           include=['Age', 'Avg. credit balance',
                                    'Last contact day',
                                    'Last contact duration (sec)',
                                    'Number of contacts',
                                    'Days passed',
                                    'Previous contacts'],
                           transfo...
                                                 return_df=True,
                                                 use_cat_names=True,
                                                 verbose=0))),
       ('balance',
        TransformerWrapper(exclude=None, include=None,
                           transformer=FixImbalancer(estimator=SMOTE(k_neighbors=5,
                                                                     random_state=42,
                                                                     sampling_strategy='auto')))),
       ('clean_column_names',
        TransformerWrapper(exclude=None, include=None,
                           transformer=CleanColumnNames(match='[\\]\\[\\,\\{\\}\\"\\:]+')))],
 verbose=False)
```