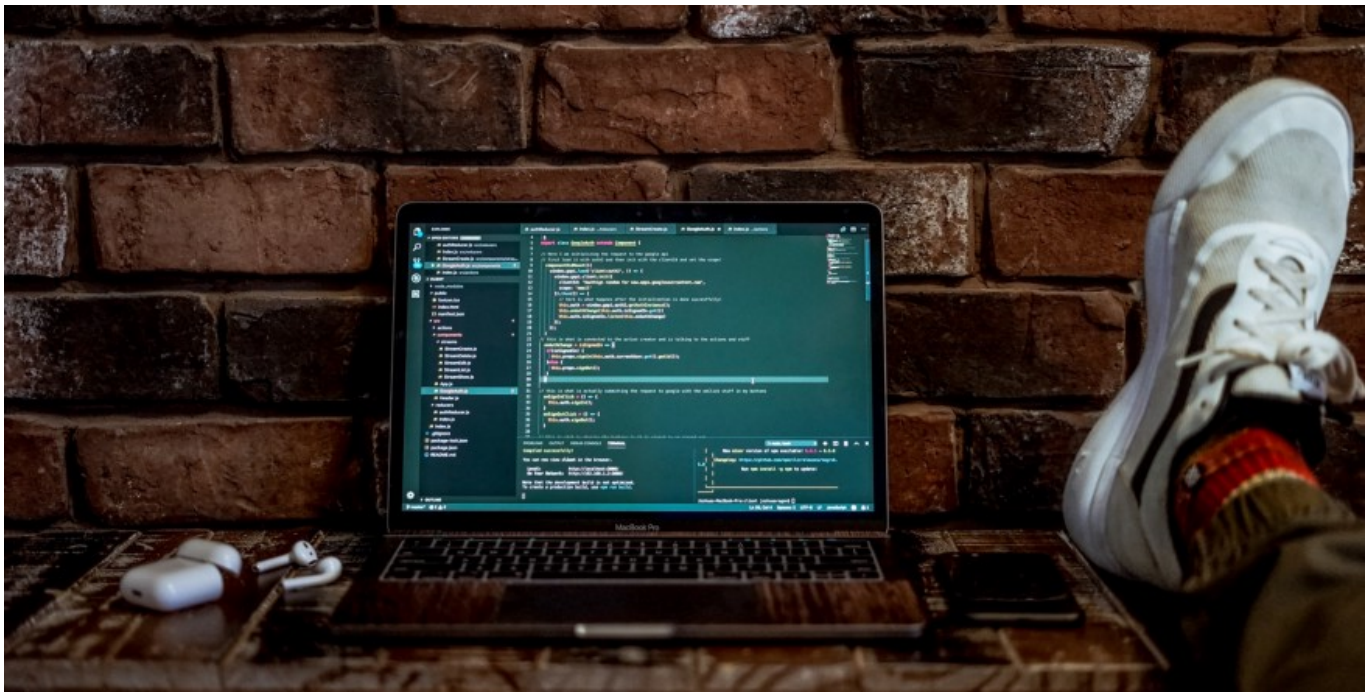

Photo by Joshua Aragon on Unsplash

Lazy Predict: fit and evaluate all the models from scikit-learn with a single line of code

The easiest way to see which models work best for your dataset!

Eryk Lewinson 1 day ago · 5 min read ★

with a model that works fairly well for many datasets (like Random Forest) and then iterate. This way, we can establish a benchmark we aim to improve.

> *"Progress isn't made by early risers. It's made by lazy men trying to find easier ways to do something."*
> *— Robert Heinlein*

Most likely, at some point in time you also considered throwing ALL the models at the dataset and just seeing what will happen. However, this was not that simple to code (or rather, it was quite tedious), so probably you gave up on the idea and just tried a few models that you have worked with in the past. But now, there is a handy library called `lazypredict`, with which you can train all the models available in `scikit-learn` (and more, like XGBoost and LightGBM) with a single line of code. Sounds great, right? Let's see how this works out in practice!

Hands-on Examples

First, you need to install the library by running:

```
pip install lazypredict
```

Most likely, you will encounter some errors about missing libraries, so just install them separately using `pip` or `conda`. I mention this later on as a possible improvement. Then, we load the required libraries:

```
from lazypredict.Supervised import LazyClassifier, LazyRegressor
from sklearn.model_selection import train_test_split
from sklearn import datasets
```

`lazypredict` supports both classification and regression problems, so I will show a brief intro to both.

Classification task

breast cancer dataset. I load the data and split it into training and test sets.

```
# load data
data = datasets.load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=.2, random_state=42)

# fit all models
clf = LazyClassifier(predictions=True)
models, predictions = clf.fit(X_train, X_test, y_train, y_test)
```

Using `lazypredict` is very easy and intuitive for anyone familiar with `scikit-learn`. We first create an instance of the estimator, `LazyClassifier` in this case, and then fit it to the data using the `fit` method. By specifying `predictions=True` while creating the instance of `LazyClassifier`, we will also receive predictions of all the models for each and every observation. Just in case we want to use them for something else later on. Additionally, we can use the `custom_metric` argument to pass a custom metric we would like to use for evaluating the models' performance.

Note: By looking at the code, you can see which estimators are excluded from the list of available models. For classification, we receive the following 4 metrics, as well as an indication of how long it took to fit the model to our dataset.

Another important thing that is rather hidden from the user is that the library automatically applies preprocessing to the dataset. First, it imputes missing values using `SimpleImputer` (using the mean for numeric features and a constant `'missing'` value for categorical ones). Then, it uses `StandardScaler` for numeric features and either `OneHotEncoder` or `OrdinalEncoder` for the categorical features (depending on the cardinality — number of unique values). While this is handy and ensures that the models will actually run, some users might prefer different approaches to preprocessing the dataset. That is why in my opinion this should be a voluntary feature of the library, instead of a mandatory one.

predictions for each model.

Regression task

As mentioned before, the regression task is very similar to the classification problem. I use the Boston Housing dataset and instantiate a different class — `LazyRegressor`. The rest is analogical.

```
# load data
boston = datasets.load_boston()
X, y = boston.data, boston.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=.2, random_state=42)

# fit all models
reg = LazyRegressor(predictions=True)
models, predictions = reg.fit(X_train, X_test, y_train, y_test)
```

Naturally, the table containing the models' performance has different metrics for the regression task, namely the R-Squared and RMSE. We could add more (for example, MAPE) using the `custom_metric` argument. The table below is truncated to keep the article concise, but the list of the available regressors is much longer.

Possible improvements

After briefly playing around with the `lazypredict` library, there are a few things that I believe could be significant improvements:

- this is a simple one, but making sure that the library has a proper list of dependencies so that the users do not have to manually install every library based on the errors they get,
- allow access to the best/all trained models, right now we can only see the table with the results and predictions,
- train the models in parallel — not an issue with small datasets, however, would be nice to speed it up for larger ones,
- create a dedicated `predict` method for getting the predictions,
- make the default preprocessing optional and clearly documented,
- allow for some hyperparameter tuning.

Conclusions

`lazypredict` is a convenient wrapper library, that enables us to quickly fit all the models to our dataset and compare their performance. This way, we can see what works well

tuning can change the performance drastically.

You can find the code used for this article on my GitHub. As always, any constructive feedback is welcome. You can reach out to me on Twitter or in the comments.

References

- https://github.com/shankarpandala/lazypredict