The `autofeat` Python Library for Automated Feature Engineering and Selection

Franziska Horn cod3licious@gmail.com
Technische Universität Berlin
Machine Learning Group
Marchstr. 23, 10587 Berlin, Germany

Robert Pack robert.pack@basf.com
Michael Rieger michael.rieger@basf.com
BASF SE
Carl-Bosch-Str. 38, 67056 Ludwigshafen, Germany

Abstract

This paper describes the `autofeat` Python library, which provides a `scikit-learn` style linear regression model with automated feature engineering and selection capabilities. Complex non-linear machine learning models such as neural networks are in practice often difficult to train and even harder to explain to non-statisticians, who require transparent analysis results as a basis for important business decisions. While linear models are efficient and intuitive, they generally provide lower prediction accuracies. Our library provides a multi-step feature engineering and selection process, where first a large pool of non-linear features is generated, from which then a small and robust set of meaningful features is selected, which improve the prediction accuracy of a linear model while retaining its interpretability.

Keywords: AutoML, Feature Engineering, Feature Selection, Explainable ML

1. Introduction

More and more companies aim to improve production processes with machine learning (ML), for example, by using a ML model to better understand which factors contribute to higher quality products or greater production yield. While advanced ML models such as neural networks (NN) might, theoretically, in many cases provide the most accurate predictions, they have several drawbacks in practice. First of all, with many hyperparameters to set, these model can be difficult and time consuming to fit, which is only aggravated by the current shortage of ML specialists in industry. Second, in many cases there is not enough data available in the first place to train a low bias/high variance model like a NN, for example, because comprehensive data collection pipelines are not yet fully implemented or because obtaining individual data points is expensive, e.g., when it takes several days to produce a single product. Last but not least, the insights generated by a ML analysis need to be communicated to others in the company, who want to use these results as a basis for important business decisions. While great progress has been made to improve the interpretability of NNs, e.g., by using layer-wise relevance propagation (LRP) to reveal which of the input features contributed most to a neural net's prediction [1, 2, 25], this is in practice still not sufficient to convince those with only a limited understanding of statistics.

Especially when dealing with data collected from physical systems, using a plausible model might even be more important than getting small prediction errors [22].

To avoid these shortcomings of NNs and other non-linear ML models, in practice we find it necessary to rely mostly on linear prediction models, which are intuitive to understand and can be trained easily and efficiently even on very small datasets. Of course, employing linear models generally comes at the cost of a lower prediction accuracy. Inspired by the SISSO algorithm [28], we therefore propose a framework to automatically generate several tens of thousands of non-linear features from the original inputs and then carefully select the most informative of them as additional input features for the linear model. We have found that this approach leads to sufficiently accurate predictions on real world data while providing a transparent model that has a high acceptance rate amongst non-statisticians in the company and therefore provides the possibility to positively contribute to important business decisions.

To make this framework more accessible to other data scientists, our implementation is publicly available on GitHub.[1] The rest of the paper is structured as follows: After introducing some related work in the area of automated feature engineering and selection, we describe our approach and the `autofeat` Python library in detail (Section 2). We then report experimental results on several datasets (Section 3) before concluding the paper with a brief discussion (Section 4).

1.1 Related Work

Feature construction frameworks generally include both a feature engineering, as well as a feature selection component [21]. One of the main differences between feature construction approaches is whether they first generate an exhaustive feature pool and then perform feature selection on the whole feature set (which is also the strategy `autofeat` follows), or if the set of features is expanded iteratively, by evaluating at each step whether the inclusion of the new features would improve the prediction accuracy. Both approaches have their drawbacks: The first approach is very memory intensive, especially when starting off with a large initial feature set from which the additional features are constructed via various transformations. With the second approach, important features might be missed if some variables are eliminated too early in the feature engineering process and can therefore not serve to construct more complex, possibly helpful features. Furthermore, depending on the strategy for including additional features, the whole process might either be very time intensive, if at each step a model is trained and evaluated on the feature subset, or can fail to include (only) the relevant features, if a simple heuristic is used for the feature evaluation and selection.

Most existing feature construction frameworks follow the second, iterative feature engineering approach: The FICUS algorithm [21] uses a beam search to expand the feature space based on a simple heuristic, while the FEADIS algorithm [9] and Cognito [17] use more complex selection strategies. A more recent trend is to use meta-learning, i.e., algorithms trained on other datasets, to decide whether to apply specific transformation to the features or not [16, 18, 26]. While theoretically promising, we could not find an easy to use

1. `https://github.com/cod3licious/autofeat`

open source library for any of these approaches, which makes them essentially irrelevant for practical data science use cases.

The well-known `scikit-learn` Python library [29] provides a function to generate polynomial features (e.g. x^2), including feature interactions (e.g. $x_1 \cdot x_2, x_1^2 \cdot x_2^3$). Polynomial features are a subset of the features generated by `autofeat`, yet, while they might be helpful for many datasets, in our experience with `autofeat`, a lot of times the ratios of two features or feature combinations turn out to be informative additional features, which can not be generated with the `scikit-learn` method. The `scikit-learn` library also contains several options for feature selection, such as univariate feature scoring, recursive feature elimination, and other model-based feature selection approaches [13, 19]. Univariate feature selection methods consider each feature individually, which can lead to the inclusion of many correlated features, like those contained in the feature pool generated by `autofeat`. The more sophisticated feature selection techniques rely on the use of an external prediction model that provides coefficients indicating the importance of each feature. However, algorithms such as linear regression get numerically unstable if the number of features is larger than the number of samples, which makes these approaches impractical for feature pools as large as those generated by `autofeat`.

One popular Python library for automated feature engineering is `featuretools`, which generates a large feature set using *deep feature synthesis* [15]. This library is targeted towards relational data, where features can be created through aggregations (e.g. given some customers (data table 1) and their associated loans (in table 2), a new feature could be the sum of each customer's loans), or transformations (e.g. time since the last loan payment). A similar approach is also implemented by the "one button machine" [20]. The strategy followed by `autofeat` is somewhat orthogonal to that of `featuretools`: It is not meant for relational data, found in many business application areas, but was rather built with scientific use cases in mind, where e.g. experimental measurements would instead be stored in a single table. For this reason, `autofeat` also makes it possible to specify the units of the input variables to prevent the creation of physically nonsensical features.

Another Python library worth mentioning is `tsfresh` [6, 7], which provides feature engineering methods for time series, together with a univariate feature selection strategy. However, while `autofeat` can be applied to a variety of datasets, the features generated by `tsfresh` only make sense for time series data, as they are constructed, e.g., using rolling windows.

2. Automated Feature Engineering and Selection with `autofeat`

The `autofeat` library provides the `AutoFeatRegression` model, which automatically generates and selects additional non-linear input features given the original data and then trains a linear regression model with these features. The model provides a familiar `scikit-learn` [29] style interface, as demonstrated by a simple usage example, where `X` corresponds to a $n \times d$ feature matrix and `y` to an n-dimensional target vector (either NumPy arrays [27] or Pandas DataFrames [23]):

```python
# instantiate the model
model = AutoFeatRegression()
# fit the model and get a pandas DataFrame with the original features
# as well as the additional non-linear features
df = model.fit_transform(X, y)
# predict the target for new test data points
y_pred = model.predict(X_test)
# compute the additional features for new test data points
# (e.g. as input for a different model)
df_test = model.transform(X_test)
```

In the following, we describe the feature engineering and selection steps happening during a call to `AutoFeatRegression.fit()` or `AutoFeatRegression.fit_transform()` in more detail. The `autofeat` library requires Python 3 and is pip-installable.

2.1 Construction of Non-Linear Features

Additional non-linear features are generated in an alternating multi-step process by applying user selectable non-linear transformations to the features (e.g. $\log(x)$, $\sqrt{x}$, $1/x$, etc.) and combining pairs of features with different operators $(+, -, \cdot)$. This results in an exponentially growing feature space, e.g., with only three original features, the first feature engineering step (applying non-linear transformation) results in about 20 new features, the second step (combining features), results in about 350 new features, and after a third step (again applying transformations), the feature space has grown to include over 7000 features. As this may require a fair amount of RAM depending on the number of original input features, the data points can be subsampled before computing the new features. In practice, performing only two or three feature engineering steps is usually sufficient.

The new features are computed using the SymPy Python library [24], which automatically simplifies the generated mathematical expressions and thereby makes it possible to exclude redundant features. If the original features are provided with physical units, only 'legal' new features are retained, e.g., a feature representing a temperature would not be subtracted from a feature representing a volume of something. This is implemented using the Pint Python library,[2] which is additionally used to compute several dimensionless quantities from the original features using the Buckingham π-theorem [5]. If categorical features are included in the original features, these are transformed into one-hot encoded vectors using the corresponding `scikit-learn` model and not considered for the main feature engineering procedure.

2.2 Feature Selection

After having generated several thousands of features (often more than data points in the original dataset), it is now indispensable to carefully select only those features that contribute meaningful information when used as input to a linear model. To this end, we employ a multi-step feature selection approach (Fig. 1). In addition to the `AutoFeatRegression` model, the library also provides only this feature selection part alone in the `FeatureSelector` class, which again provides a `scikit-learn` style interface.

Individual features can provide redundant information or they might seem uninformative by themselves yet proof useful in combination with others. Therefore, instead of ranking the features independently by some criterion, it is advantageous to use a wrapper method that

2. `https://pint.readthedocs.io/en/latest/`

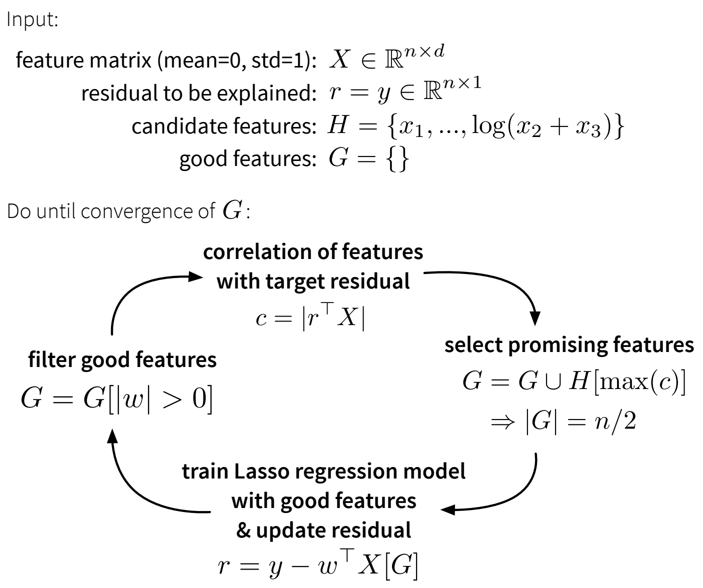
Input:
feature matrix (mean=0, std=1): $X \in \mathbb{R}^{n \times d}$
residual to be explained: $r = y \in \mathbb{R}^{n \times 1}$
candidate features: $H = \{x_1, ..., \log(x_2 + x_3)\}$
good features: $G = \{\}$
Do until convergence of G:
correlation of features with target residual $c = |r^\top X|$
select promising features $G = G \cup H[\max(c)]$ $\Rightarrow |G| = n/2$
train Lasso regression model with good features & update residual $r = y - w^\top X[G]$
filter good features $G = G[|w| > 0]$

Figure 1: The heart of our feature selection algorithm. Given the feature matrix X with all candidate features H, the aim is to select a few informative features (G) that explain the target variable y. The set of good features G is adapted until a stable set of features is reached. First, promising features are identified by computing the correlation between the features and the target residual, and G is extended by those features with the largest absolute correlation until G contains up to $n/2$ features (to guarantee numerical stability in the following regression step). Next, the currently selected good features are used to train a Lasso LARS regression model, based on which the target residual is updated and the good features are filtered by retaining only those with a non-zero regression weight.

considers multiple features at once to select a promising subset [13]. For this we use the Lasso LARS regression model [3, 10, 12] provided in the `scikit-learn` library, which yields sparse weights based on which the features can be filtered. However, with more features than data points, a linear regression model is numerically unstable. Therefore, the features are first ranked based on their absolute correlation with the target residual [11] and the model is only trained on the highest ranked features. Then, to include further features capturing the not yet explained parts of the target variable, these steps are repeated multiple times, where in each iteration the regression model is used to compute a new target residual.

To identify a more robust set of features, this feature selection process can be repeated multiple times using subsamples of the data. The resulting set of features is then filtered by imposing a significance threshold: For this, a Lasso LARS regression model is trained on the selected features, as well as a random permutation of all features. The final set of features is then determined by choosing only those of the real features with a regression coefficient larger than the largest coefficient of the random noise features. After this multi-step selection process, typically only a few dozen of the several thousand engineered features are retained and used to train the final model. For new test data points, the `AutoFeatRegression` model can then either generate predictions directly, or a DataFrame with the new features can be computed for all data points and used to train other models.

By examining the coefficients of the regression model (possibly normalized by the standard deviation of the corresponding features, in case these are not of comparable magnitudes), the most prominent influencing factors related to higher or lower values of the target variable can be identified.

3. Experimental Results

To give an indication of the performance of the `AutoFeatRegression` model in practice, compared to other non-linear ML algorithms, we test our approach on five regression datasets (Table 1), provided in the `scikit-learn` package (*diabetes* and *boston*) or obtainable from the UCI Machine Learning Repository.[3] For further details on the experiments, including the hyperparameter selection of the other models, please refer to the corresponding Jupyter notebook in the GitHub repository.

Table 1: Overview of datasets, including the number of samples n and number of original input features d.

Dataset	n	d	Prediction task
diabetes [10]	442	10	disease progression one year after baseline
boston [14]	506	13	median housing values in suburbs of Boston
concrete [30]	1030	8	compressive strengths of concrete mixtures
airfoil [4]	1503	5	sound pressure levels of airfoils in a wind tunnel
wine quality [8]	6497	12	quality of red & white wines based on physiochemical tests

While on most datasets, the `AutoFeatRegression` model does not quite reach the state-of-the-art performance of a random forest regression model (Table 2), it clearly outperforms standard linear ridge regression, while retaining its interpretability. Across all datasets, with one feature engineering step, `autofeat` generated between 0 and 4 additional features, while with two and three steps, it produced on average 22 additional features (min: 6, max: 39). With only a single feature engineering step, the model often only performs slightly better than ridge regression on the original features. With three feature engineering steps, on the other hand, the `AutoFeatRegression` model can overfit on the training data (as indicated by the discrepancy between the training and test R^2 scores), because the complex features do not only explain the signal, but also the noise contained in the data. However, the only dataset where this is a serious problem here is the *boston* dataset, where over 50k features were generated in the feature engineering process, while less than 500 data points were available for feature selection and model fitting, which means overfitting is somewhat to be expected.

3. `http://archive.ics.uci.edu/ml/index.php`

Table 2: R^2 scores on the training and test folds of different datasets for ridge regression (RR), support vector regression (SVR), random forests (RF), and the `autofeat` regression model with one, two, or three feature engineering steps (AFR1-3). Best results per column are in boldface.

	diabetes		**boston**		**concrete**		**airfoil**		**wine quality**	
	train	test	train	test	train	test	train	test	train	test
RR	0.541	0.383	0.736	0.748	0.625	0.564	0.517	0.508	0.293	0.310
SVR	0.580	0.320	0.959	**0.882**	0.933	0.881	0.884	0.851	0.572	0.411
RF	**0.598**	0.354	**0.983**	0.870	**0.985**	**0.892**	**0.991**	**0.934**	**0.931**	**0.558**
AFR1	0.556	0.396	0.829	0.802	0.800	0.732	0.544	0.532	0.296	0.310
AFR2	0.539	**0.402**	0.886	0.818	0.903	0.859	0.879	0.866	0.348	0.365
AFR3	0.597	0.395	0.929	0.035	0.898	0.858	0.876	0.855	0.346	0.348

4. Conclusion

In this paper, we have introduced the `autofeat` Python library, which includes an automated feature engineering and selection procedure to improve the prediction accuracy of a linear regression model by using additional non-linear features. The regression model itself is based on the Lasso LARS regression from `scikit-learn` and provides a familiar interface. During the model fit, a vast number of non-linear features is generated from the original features and a few of these are selected in an elaborate iterative process to optimally explain the target variable. By combining a linear model with complex non-linear features, a high prediction accuracy can be achieved, while retaining a transparent model that yields traceable results as a basis for business decisions made by non-statisticians.

We have demonstrated on several datasets that the `AutoFeatRegression` model significantly improves upon the performance of a linear regression model and sometimes even outperforms other non-linear ML models. While the model can be used for predictions directly, it might also be beneficial to use the generated features as input to train other ML models. By adapting the kinds of transformations applied in the feature engineering process, as well as the number of feature engineering steps, further insights can be gained with respect to how which of the input features influences the target variable, as well as the complexity of the system as a whole.

The `autofeat` library was developed with scientific use cases in mind and is especially useful for heterogeneous datasets, e.g., containing sensor measurements with different physical units. It should not be seen as a competitor for the existing feature engineering libraries `featuretools` or `tsfresh`, which would be the first choice when dealing with relational business data or time series respectively.

Acknowledgments

FH was a part-time employee at BASF when programming the `autofeat` library.

References

[1] Leila Arras, Franziska Horn, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. “What is relevant in a text document?”: An interpretable machine learning approach. *PLOS ONE*, 12(8):e0181142, 2017.

[2] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):e0130140, 2015.

[3] Richard G Baraniuk. Compressive sensing [lecture notes]. *IEEE Signal Processing Magazine*, 24(4):118–121, 2007.

[4] Thomas F Brooks, D Stuart Pope, and Michael A Marcolini. Airfoil self-noise and prediction. *Technical report, NASA RP-1218*, 1989.

[5] Edgar Buckingham. On physically similar systems; illustrations of the use of dimensional equations. *Physical review*, 4(4):345, 1914.

[6] Maximilian Christ, Andreas W Kempa-Liehr, and Michael Feindt. Distributed and parallel time series feature extraction for industrial big data applications. *arXiv preprint arXiv:1610.07717*, 2016.

[7] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77, 2018.

[8] Paulo Cortez, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis. Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553, 2009.

[9] Ofer Dor and Yoram Reich. Strengthening learning algorithms by feature discovery. *Information Sciences*, 189:176–190, 2012.

[10] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004.

[11] Jianqing Fan and Jinchi Lv. Sure independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(5):849–911, 2008.

[12] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1, 2010.

[13] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[14] David Harrison Jr and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[15] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19-21, 2015*, pages 1–10. IEEE, 2015.

[16] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 979–984. IEEE, 2016.

[17] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasrathy. Cognito: Automated feature engineering for supervised learning. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 1304–1307. IEEE, 2016.

[18] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. Feature engineering for predictive modeling using reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[19] Miron B Kursa, Witold R Rudnicki, et al. Feature selection with the boruta package. *J Stat Softw*, 36(11):1–13, 2010.

[20] Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327*, 2017.

[21] Shaul Markovitch and Dan Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 49(1):59–98, 2002.

[22] Georg Martius and Christoph H Lampert. Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*, 2016.

[23] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[24] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.

[25] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[26] Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B Khalil, and Deepak S Turaga. Learning feature engineering for classification. In *IJCAI*, pages 2529–2535, 2017.

[27] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[28] Runhai Ouyang, Stefano Curtarolo, Emre Ahmetcik, Matthias Scheffler, and Luca M Ghiringhelli. Sisso: A compressed-sensing method for identifying the best low-dimensional descriptor in an immensity of offered candidates. *Physical Review Materials*, 2(8):083802, 2018.

[29] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[30] I-C Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998.