Chapter 10

Pattern-based Causal Feature Extraction

Diogo Moitinho de Almeida

Abstract T

his cause-effect pairs challenge was motivated by the contrast between the costs of performing controlled experiments in order to determine causality and the abundance of observational data. Our goal was to provide a value representing our confidence of causality determined by the observation data which would help identify the most promising variables for experimental verification of their causal relationship. By identifying patterns in functions that generate relevant features, a feature extraction pipeline was architected to allow for the creation of large amounts of complex features with minimal human intervention. Using this pipeline, we were able to finish second in the public leaderboard and first in the private leaderboard. Furthermore, this process by default generates over $20,000$ features. In this paper, we analyze which aspects are most important, and create a new pipeline that gets comparable performance with only 324 features.
Keywords: Feature Extraction, Machine Learning, Causality.

10.1 Introduction

From the competition homepage [kag]: The problem of attributing causes to effects is pervasive in science, medicine, economy and almost every aspects of our everyday life involving human reasoning and decision making... However, experiments are costly while non-experimental "observational" data collected routinely around the world are readily available. Unraveling potential cause-effect relationships from such observational data could save a lot of time and effort... The objective of the challenge is to rank pairs of variables A, B to prioritize experimental verification of

Diogo Moitinho de Almeida e-mail: diogo149@gmail.com

the conjecture that A causes B. Contestants were given over $20,000$ training pairs of variables deprived of their context of both real variables with known causal relationships from diverse domains and artificially generated variables and their respective causal relationships. Contestants were to use this data in order to calculate a ranking of each pairs of variables were the highest ranked pairs had the first variable, A, cause the second variable, B, and the lowest ranked pairs have B cause A. The rankings that we provided were judged by the average of the AUC's of predicting whether or not A causes B and B causes A. The competition has both a public and private leaderboard, each of which coming with 4050 pairs of variables which contestants were not able to see the labels of. The public leaderboard was available for the duration of the competition, where contestants could submit rankings twice a day in order to see how well their algorithms perform on that dataset. Because competitors could potentially overfit on that dataset, the final winner was determined by only one submission on the private leaderboard, whose data was only made available after competitors submitted their algorithms to be tested.

10.2 Method

Our method was able to attain the second highest score in the public leaderboard and the highest score in the private leaderboard. Our final solution involved not only the feature extraction process outlined in this paper, but also a process for the elimination of features, which turned out to be unnecessary, and scikit-learn's [Pedregosa et al., 2011] gradient boosted decision tree ensemble [Friedman, 2001] classifier with hyper-parameters tuned by Spearmint [Snoek et al., 2012] for the final rankings.

The focus of this paper will solely be on the feature extraction methodology, because the feature extraction process contains all of our novel contributions for and we feel that the rest of our pipeline was fairly standard for a kaggle competition.

10.2.1 Algorithm

Our process revolves around general algorithm templates that one would use to generate causal features, which we call Feature Patterns. These Feature Patterns take in algorithms/functions as parameters and create new algorithms that can then generate features. By simply creating the architecture for a few patterns and changing the parameterization of each pattern, many unique features could be generated with this approach.

Table 10.1: Leaderboard Scores of Top Submissions

Submission	Public	Private
Our Submission (1st place): w/ Feature Selection +	0.81367	0.8196
Our 2nd Best Submission: w/o Feature Selection +	0.81279	0.81743
Our 3rd Best Submission: w/ Feature Selection +	0.81238	0.81681
Team jarfo (2nd place): Top Submissions +	0.81464	0.81052
Team HiDLoN (3rd place): Top Submissions +	0.80191	0.80720

In addition to the architecture sharing that occurs within a Feature Pattern, the commonalities between Feature Patterns allow even greater code reuse and more importantly for the creation of more complex work flows than we could generate from scratch. Take for example our simplest Feature Pattern: a unary function that takes in a numerical variable. This pattern would require the unary function, functions to convert categorical, binary, and numerical variables to numerical, and an aggregation function for the case when the return value of one the transforms is multidimensional (for example, if the transform from categorical to numerical is a one-encoding), in which case we would apply the unary transform to each column of the resulting matrix, and apply the aggregation function to combine the results. Afterwards, we would do the same function with the variables reversed, and then passing the two resulting values to a relative feature function (for example, taking the difference between the two variables, or only returning second of the two). Since all Feature Patterns would have to handle the conversion between numerical, categorical, and binary, as well as handle the relative features and aggregation, if any, afterwards, relatively little effort is needed to create more complex features.

Table 10.2: Feature Patterns. N, B, and C are used for numerical, binary, and categorical

Feature Pattern	Example Parameters
N Unary Function	N Unary Function, N/B/C-to-N Transforms, Aggregator
NN Binary Function	NN Binary Function, N/B/C-to-N Transforms, 2 Aggregators
CN Binary Function	CN Binary Function, N/B/C-to-N/C Transforms, 2 Aggregators
Regression Metric	Regression Predictor, Metric, N/B/C-to-N Transform, Aggregator
Classification Metric	Classifier, Metric, N/B/C-to-N/C Transform, 2 Aggregators

In addition to sharing architecture, the same benefit can also be attained through the unification of the possible values for similar/equivalent parameters. By doing so, the number of features would grow at a rate greater than linearly for each possible value of a shared parameters and thus, it would require less parameters to get an equivalent number of unique features.

Table 10.3: Examples of Pattern Parameters Used

Parameter Type	Examples
Aggregation Functions	Mean, Max, Min, Median, Sum
Regression Preditors	Ridge Regression, Random Forest, k-NN
Classification Predictors	Logistic Regression, Random Forest, Naive Bayes
Classification Metrics	Accuracy, AUC, Hinge Loss
Regression Metrics	Mean Squared Error, Mean Absolute Error
Clustering Metrics	Mutual Information Score, Homogeneity Score
Statistical Tests	Pearson's R, χ^2 Test, ANOVA
Distance Metrics	Euclidean Distance, Cosine Distance
Unary Functions	Normalized Entropy, Skew, Kurtosis

The feature extraction process is then completed by simply iterating over each Feature Pattern and parameter combination to generate valid features for each observation.

10.2.2 Justification

The assumed definition of causality for this section is that which was provided in the competition site [kag]: if $B = f(A, noise)$, then $AisthecauseofB$, and vice-versa. While each feature extraction algorithm created from each Feature Pattern would have its own justification, we believe a specific set of patterns, namely "Regression Metric" and "Classification Metric", deserve special attention. This is because these patterns have the most parameters, and thus contributed the largest amount of features to the final feature sets, and as shown in the experimental results, these features seemed to be the most important for performance. The gist of these patterns is that a model is trained to predict one variable from the other variable, that model is used to generate predictions for that variable, and a function (generally a goodness-of-fit measure) is applied to those predictions.

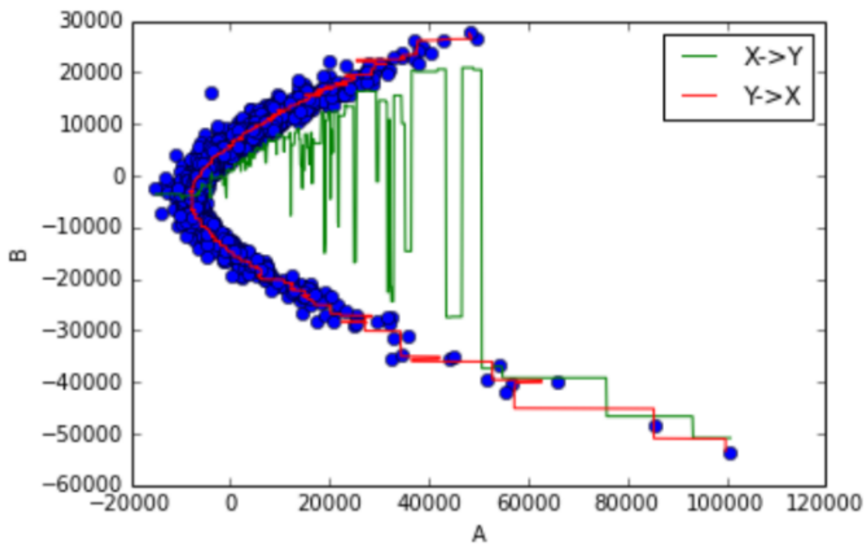

Fig. 10.1: Difference between Goodness of Fit of Invertible and Non-Invertable Functions.

Our motivation behind this class of goodness-of-fit features are the assumptions that the features would be especially good at detecting how likely it is that a function from one variable to the other (namely f in the assumed definition of causality) can exist and that the likelihood that that function exists is a good feature for causal discovery. The first assumption makes sense because the existing machine learning algorithms which are used to provide the fit generally perform quite well at estimating hypothesis functions on real-world data. The second assumption similarly makes sense because B = f(A, noise) or A = f(B, noise) being true relies on the such an f existing.

10.2.3 Considerations

Due to the time constraints of the competition and the limitations of the effort that we could afford to spend, the motivation for creating our process was to easily, quickly, and reproducibly go from insight to features with minimal human intervention. As such, we made the decision to trade off computational time in order minimize human time. The result is our algorithm can be quite inefficient since features generated from similar parameters in general would be quite similar/redundant, though the exact computational cost would depend on the Feature Patterns and parameters used. Another downside of our approach is that bookkeeping on individual features quickly loses human interpretability, since it is so far removed from the original data.

10.2.4 autocause

Because the competition code was mostly hacked together and did not take full advantage of sharing both architecture and parameters, we refactored the codebase of the feature extraction process into its own package: autocause[1]. This allowed us to declaratively iterate different settings and publish the settings in a human-readable, unix-diff-able format as well for further analysis[2].

10.3 Experiments

Because of the aforementioned difficulty of interpreting the individual features, we chose to perform analyses on classes of features that might provide insight into the most effective parts of our feature extraction process and the underlying mechanisms of causal discovery. These analyses were performed by varying the available parameters to Feature Patterns in order to either find which of a set of parameters are most important, or by removing an entire set of parameters, eliminating all Feature Patterns that depend on those parameters. Each set of features was scored by being trained by both a linear model and gradient boosted decision tree ensemble with both models optimized for speed on 80% of the final training data of the competition, and having their predictions score on the final 20% of the data using the same metric as in the competition.

As shown in the experimental results table, after refactoring the code, a lot more features were able to be generated with the same number of parameters. This is good to confirm that the features generated by autocause are comparable to that of the software used during the competition. The results for a set of parameters that were chosen by combining insights from several dozen experiments (see the appendix) for a mix of relatively low dimensionality for the final feature set and good accuracy are also included in the table. This parameter set allows us to get over 98% of the

[1] Available at https://github.com/diogo149/autocause

[2] See the configs subdirectory of https://github.com/diogo149/CauseEffectPairsPaper

Table 10.4: Experimental Results.

Description	#Feat	GBM Score	Linear Score
Competition Features	8686	0.692483139268	0.663782901707
autocause Default	21207	0.696014248897	0.694080271398
Efficient Parameters	324	0.682581676518	0.687203189251
Effective Parameters	14442	0.676321066209	0.681059838814

accuracy of the default model using only 1.5% of the number of features. Using the same methodology as in the last paragraph, we tried to construct a parameter set to only maximize accuracy by only keeping the changes that improved accuracy by a noticable amount. This unfortunately led to features that performed significantly worse than the default settings. This indicates that the methodology of picking and choosing parameters by looking at each individually and combining them after the fact is flawed, and that better means of creating parameter sets should be used.

Table 10.5: Results of Experiment on Fit Features.

Description	#Feat	GBM Score	Linear Score
Both together	21186	0.713640109365	0.700135374168
Only Fit Default	18612	0.693269150278	0.680996251976
No Fit	2574	0.687915176559	0.611305285221

One notable set of results from experiments is that of the results between features that rely on only classification/regression predictors and those that don't at all, because the experiments show the performance of different Feature Patterns. The "No Fit" features contain all the unary/binary function patterns, while the "Only Fit" features contain "Regression Metric" and "Classification Metric" Feature Patterns. The scores of each show the "Only Fit" features to perform significantly better, indicating that its Feature Pattern likely contributed more to the accuracy of our winning submission than the others.

10.4 Conclusion

From the approach of feature extraction for accuracy, there are still more Feature Patterns that could be added to our approach, such as those underlying the previous state of the art features [Cau], and Feature Patterns such as those described in this paper may just be the tip of the iceberg, both in terms of quality and quantity. During the competition, we were certainly biased towards the "Regression Metric" and "Classification Metric" Feature Patterns. Despite them performing the best in our limited experimental results, it may be the case that we haven't appropriately represented other Feature Patterns. There seems to be even more work remaining from the perspective of understanding why our process works as well as it does. We were able to get comparable performance to our 20k-dimensional features with only 324 features, but that still is too many for us to dive deep and find the truly important ones, and even if we did do so, the features generated may be too far removed from the original data to retain any human interpretable meaning.

Acknowledgment

Special thanks to the organizers of the ChaLearn Cause-Effect Pair Challenge hosted by Kaggle.

References

Causality Workbench causality challenge #3: Cause-effect pairs - help. `http://www.causality.inf.ethz.ch/cause-effect.php?page=help`. Accessed: 2013.

Cause-Effect Pairs, howpublished = `http://www.kaggle.com/c/cause-effect-pairs`, note = Accessed: 2013.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

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.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

Appendix

Results

Table 10.6: Results of Experiment on Meta-features.

Description	#Feat	GBM Score	Linear Score
autocause Default	21207	0.696014248897	0.694080271398
No Metafeatures	21186	0.713640109365	0.700135374168
Only Metafeatures	21	0.513249437852	0.514610878117

Table 10.7: Results of Experiment on Relative Features.

Description	#Feat	GBM Score	Linear Score
No Metafeatures	21186	0.713640109365	0.700135374168
Only Difference Features	7062	0.699178700923	0.707910542983
Only A to B	7062	0.646039719086	0.619346087547
Only B to A	7062	0.675171907802	0.633300654009

Table 10.8: Results of Experiment on Aggregation Features.

Description	#Feat	GBM Score	Linear Score
Only Mean	4743	0.703366341099	0.66190548093
Only Mode	4743	0.702727813568	0.664183581517
Only Median	4743	0.691697784105	0.670042628328
Only Min	4743	0.700988250898	0.660565019326
Only Sum	4743	0.701434552265	0.673035609828
Only Max	4743	0.691870225179	0.651876362553
All Of The Above	25773	0.709451429787	0.695304807716
No Aggregation Features	1077	0.676581730474	0.623828720773

Table 10.9: Results of Experiment on Numerical vs Categorical Features.

Description	#Feat	GBM Score	Linear Score
autocause Default	21207	0.696014248897	0.694080271398
Numerical Only	6321	0.656714283634	0.611395370877
Categorical Only	5451	0.579135168216	0.607908620448

Table 10.10: Results of Experiment on Numerical to Categorical Transformation.

Description	#Feat	GBM Score	Linear Score
Discretization Into 10 (default)	5451	0.579135168216	0.607908620448
KMeans Into 10	5451	0.632243266485	0.622365613862
KMeans Into 3	5451	0.614931332437	0.587218538488
KMeans With Gap Statistic	5451	0.583693386727	0.582386249005

Table 10.11: Results of Experiment on Categorical to Numerical Transformation.

Description	#Feat	GBM Score	Linear Score
One-hot Encoding (default)	6321	0.656714283634	0.611395370877
Identity	921	0.671054122146	0.592442871034
PCA to 1 Dimension	921	0.66682459185	0.606938680493
Reshuffling	921	0.651080073736	0.599028952216

Table 10.12: Results of Experiment on Classifiers.

Description	#Feat	GBM Score	Linear Score
Only Naive Bayes	996	0.578082139243	0.59122957333
Only GBM	996	0.633170335784	0.607150621346
Only RandomForest	996	0.646001448959	0.607517231438
Only k-NN	996	0.607753059361	0.572566617101
Only Logistic Regression	996	0.620826820564	0.600463299695
Only Decision Tree	996	0.637150678151	0.61158719868
All Of The Above	5451	0.632243266485	0.622365613862
No Classifier Features	105	0.581141154559	0.572462173005

Table 10.13: Results of Experiment on Regression Predictors.

Description	#Feat	GBM Score	Linear Score
Only RandomForest	261	0.643682857561	0.568762197986
Only GBM	261	0.657848458945	0.596995045661
Only DecisionTree	261	0.618929711403	0.550528710645
Only k-NN	261	0.638331347172	0.577983408646
Only Ridge	261	0.62702149139	0.578773516607
Only Linear Regression	261	0.624841890148	0.577198194394
All Of The Above	921	0.66682459185	0.606938680493
No Regression Predictor Features	129	0.619709098921	0.540686641005