Ensembles with Automated Feature Engineering

Udayan Khurana UKHURANA@US.IBM.COM
Horst Samulowitz SAMULOWITZ@US.IBM.COM
Deepak Turaga TURAGA@US.IBM.COM
IBM Research AI

Abstract

Feature engineering for supervised learning problems results in the creation of several data versions through feature transformations. While feature engineering focuses on creating the version of data that results in the single best performing model, a trail of several other models/data are usually discarded. We observe that upon carefully selecting a subset of these subpar models, simple but effective ensembles can be created that outperform the impact of feature engineering alone. We present a novel automated ensemble method that explores feature transformations through reinforcement learning; it is trained with the objective of optimizing ensemble generalization error through models of high quality as well mutual diversity. A subset of the explored models are then chosen as ensemble candidates by minimizing ensemble generalization error explicitly. While there exist automated ways of constructing ensembles through data subsets, such as Bootstrapped Aggregation, we are not aware of a technique that systematically uses transformation functions to create additional features that are effectively consumed in an ensemble. We provide results and a preview of our system demonstrating the effectiveness of the described technique.

Keywords: Ensembles, Feature Engineering, AutoML, Reinforcement Learning

1. Introduction

Training supervised learning models of high quality goes beyond finding a suitable learning algorithm. Two widely practiced activities to improve model performance are model ensembles and feature engineering. Ensembles use predictions from multiple models to provide a more accurate prediction through a variety of aggregation techniques such as averaging or majority voting. Feature engineering involves transforming feature the given feature space to better represent the given learning problem. It is typically performed through applying mathematical functions or aggregates on existing features. Recently, few algorithms have been proposed to automatically perform feature engineering. We observe that automatic feature engineering processes produce several transformed versions of the data. Models trained on several of those data versions may or may not offer significant improvement over the base dataset or over each other. However, the diversity in different datasets owing to different constructor functions can be a good basis to construct ensembles that are better than any single version created from feature engineering exploration itself. In this paper, we present a technique to create effective ensembles on models obtained through a modified feature engineering exploration. It uses Q-learning to explore transformations, that optimizes ensemble accuracy through a reward signaling individual model performance as well as mutual diversity. Upon exploration, a subset of all data versions are explicitly selected based on an effective greedy search algorithm that maximizes averaging ensemble performance. Our empirical evaluation on a range of openly available datasets demonstrates that our proposed technique produces feature engineered ensembles of high quality.

In recent years, different approaches have been proposed for performing automated feature engineering. One particular approach by Khurana et al. (2016) is based on trial or exploration of different transform functions and finding sequences of transformations with high returns based on initial feedback. The feedback and optimization goal is the reduction in model error. Khurana et al. (2018) refine it by providing an exploration strategy in limited number of trials, through reinforcement learning on historical datasets. In this paper, we extend this framework by principally modifying the reward mechanism to optimize for ensemble goals of both, high accuracy and diversity instead of model accuracy alone.

There are other relevant approaches to feature engineering which we only briefly mention here. Nargesian et al. (2017) directly predict the most likely useful transformation per feature based on learning effectiveness of transforms on sketched representations of historical data through a perceptron. FICUS by Markovitch and Rosenstein (2002) performs a beam search over the space of possible features, constructing new features by applying constructor functions. Its is guided by heuristic measures based on information gain in a decision tree. Data Science Machine by Kanter and Veeramachaneni (2015) relies on applying all transformations on all features at once (but no combinations of transformations), then performing feature selection and model hyper-parameter optimization over the augmented dataset. FEADIS by Dor and Reich (2012) works through a combination of random feature generation and feature selection. ExploreKit by Katz et al. (2016) expands the feature space explicitly. It employs learning to rank the newly constructed features and evaluating the most promising ones. A detailed explanation including relationships between these approaches can be found in Khurana (2018). Hyper-parameter optimization has also been employed to some limited settings of feature engineering, such as Feurer et al. (2015).

Model ensembles are used extensively in machine learning to aggregate the output of several weak predictors into a single strong predictor (see e.g., Dietterich (2000)). Ensembles can be constructed in many different ways. The basic ask, however, is that the base predictors perform above a threshold (above random chance, such as $p > 0.5$ for binary classification), and be diverse enough. Different methods to ensemble ensure diversity through different means. For example, Random Forests by Breiman (1999) use systematic *randomization* to create data subsets and perturbation in branch splitting to reduce variance in a large number of decision trees. The aggregation is a simple unweighted average.

In the given scenario with respect to feature engineering, we are dealing with a different landscape. Due to the associated cost of exploration involving model building and validation, there is a moderate number of base models (typically 10-100), all of which are fairly correlated to each other, with small differences. The mutual correlation is because most of them contain an overlapping set of (original) features, apart from different transformed features which often cause mild to moderate deviation in the behavior of the model. Hence, a simple averaging of all base models seems ineffective, and a rather careful selection of base models is required. We also observed that feature engineering can often overfit to the evaluation set, which presents a blessing in disguise for ensembles. Benefits of overfitted base models are summarized by Sollich and Krogh (1996). We acknowledge that other known mechanisms to improve model diversity can be used along with our proposed methods. These include the use of multiple learning algorithms, with different hyper-parameter configurations, random subset selection, and others. In fact, these mechanisms are complimentary to the core idea suggested in this paper. However, in order to evaluate the merit of the core technique proposed here, we stick to the choice of a single learning algorithm with a fixed set of hyper-parameters and do not include data sub-selection explicitly (it may happen within a learning algorithm). To further emphasize the benefits of the core approach, we present promising results with simple averaging instead of employing more sophisticated approaches such as stacking (Gunes et al. (2017)) or boosting (Freund and Schapire (1997)), that are obvious additions to our ongoing work.

There exists a large corpus of work on generating diversity for ensembles. Melville and Mooney (2003), for instance, generate artificial examples to construct diverse predictors whose ensemble provides good gains. Cunningham and Carney (2000) and Zenobi and Cunningham (2001) introduce diversity based on feature selection through hill climbing algorithms. A survey of diversity promoting methods is presented by Brown et al. (2005).

We first describe the formula we use to calculate the ensemble performance of a group of models through averaging. We adopt this because of its ease of computation, especially in an incremental manner with respect to adding newer models. That enables us to outline an effective algorithm to select a subset of given models (their predictions and truth values for a set of examples), which minimize the generalization error in an ensemble. It is through this algorithm, that we assess the ensemble potential of a given set of models, by finding its best subset's ensemble. That estimate is used to generate the signal for the reinforcement learner's environment, that helps the agent learn the appropriate exploration strategy.

2. Searching for Ensemble Constituents

Krogh and Vedelsby (1995) provide a useful expression for computing the ensemble generalization error. Suppose there are N base models and the output of model α on input x be $V^\alpha(x)$. Let a weighted average ensemble output on x be, $\overline{V}(x) = \sum_\alpha w_\alpha V^\alpha(x)$. The ambiguity on input x of a single member of the ensemble is defined as $a^\alpha = (V^\alpha(x) - \overline{V}(x))^2$. The overall ambiguity of the ensemble on input x is: $\bar{a}(x) = \sum_\alpha w_\alpha a^\alpha(x) = \sum_\alpha w_\alpha (V^\alpha(x) - \overline{V}(x))^2$. This is the variance of the weighted ensemble around the weighted mean. Let $y(x)$ be the true outcome value for input x. The squared errors for the model α and the ensemble respectively are: $\epsilon^\alpha(x) = (f(x) - V^\alpha(x))^2$, and $e(x) = (f(x) - \overline{V}(x))^2$. Now, let the weighted average error of the models be $\bar{\epsilon}(x) = \sum_\alpha w_\alpha \epsilon^\alpha(x)$. By rearrangement, we obtain, $e(x) = \bar{\epsilon}(x) - \bar{a}(x)$. Averaging the above over several inputs:

$$E = \overline{E} - \overline{A} \tag{1}$$

which states that the generalization error of the ensemble equals the weighted average of the generalization errors of the individual models ($\overline{E} = \sum_\alpha w_\alpha E^\alpha$) minus the weighted average of the ambiguities of the individual models ($\overline{A} = \sum_\alpha w_\alpha A^\alpha$). E^α and A^α are model α's average error and ambiguity.

The significance of the result by Krogh and Vedelsby (1995) in Equation 1, is that it can help evaluate the relative performance of different model sets through individual model performance and a measure of average ambiguity. Using this, we define the following greedy algorithm to find the suitable subset of ensemble constituents in an efficient manner. The

Algorithm 1 iterates over all the predictions of all currently available models, and continues adding the predictions to the set of selected ones M as long as adding them improves over $E(M)$. Computation of $E(M+n)$ can be performed efficiently using $E(M)$, n and some bookkeeping by utilizing the definition of generalization error from Equation 1. We omit the proof here. The algorithm subroutine is referred to as $E_{min}(M)$.

```
Data: N (Set of prediction vectors from available models)
$M \leftarrow \phi$
while $|N| > 0$ do
    $n^* \leftarrow \arg\min_{n \in N} E(M+n)$
    if $E(M+n^*) \leq E(M)$ then
        $M \leftarrow M + n^*$
        $N \leftarrow N - n^*$
    else
        break
    end
end
Result: M
```

Algorithm 1: Greedy model subset selection: given a set of predictions from models, the algorithms selects a subset based on $E(M)$. We call this algorithm subroutine as $E_{min}(M)$.

3. Feature Engineering Exploration for Ensembles

We adapt the basic hierarchical structure of feature engineering exploration introduced in Khurana et al. (2018) with a bias towards discovering models useful for effective ensembles. For a given feature set, X_0, a target, y, and using a finite set of transformations, T: A *Transformation Tree*, G, is a defined as a tree in which each node corresponds to a either X_0 or a feature set derived from it through a transformation sequence based on elements in T. Every node's dataset contains the same number of rows. For nodes X_i, where $i > 0$, and its parent node X_j, $i > j \geq 0$, and the connecting edge from X_j to X_i corresponds to a transform $t \in \mathcal{T}$ (including feature selection), i.e., $X_j = t(X_i)$. X_j consists of features from X_i and additional features generated upon applying the transformation function on all subsets of X_i which satisfy the definition of t's input (e.g., for $log(f)$, $f > 0$). At each node, a model construction-evaluation happens through cross-validation, resulting in a vector of predictions, V^i, estimating y, through k-fold cross-validation.

The problem of constrained exploration in B trials (a hyperparameter) constructs an B-node tree that is the subset of the unbounded complete tree. The rationale for the budget is that due to the associated cost of model building-validation, we attempt to constraint the total cost. Consider the tree exploration process as a *Markov Decision Process (MDP)*, where the *state* at step i is a combination of two components: (a) transformation tree snapshot with i nodes; (b) the remaining budget at step i, i.e., $b_{ratio} = \frac{i}{B_{max}}$. Note that a state of G implies the knowledge of impact of various transformations, which play a role in the definition of the state. Let the entire set of states be S. On the other hand, an *action* at step i is a pair of existing tree node and transformation, i.e., $< n, t >$ where n is a $node(G_t)$, $t \in T$ and $\nexists n' \in G_i$ such that $edge(n, n') = t$; it signifies the application of the one transform (which hasn't already been applied) to one of the exiting nodes in the graph. Let the entire set of actions be C. A policy, $\Pi : S \to C$, determines which action is taken given a state. Note that the objective of RL here is to learn the optimal policy (exploration strategy) by learning the action-value function, which we elaborate later in the section.

Such formulation uniquely identifies each state of the MDP, including the context of "remaining budget", which helps the policy implicitly play an adaptive *explore-exploit* tradeoff. It decides whether to focus on exploiting gains (depth) or exploring (breadth) or a compromise, in different regions of the graph, at different steps. Overall, the policy selects the action with the highest expected long-term reward contribution; however, upon evaluating a new node's actual immediate contribution, the expectations are often revised and explore/exploit gears are (implicitly) calibrated through the policy. For example, upon finding an exceptionally high improvement at a node during early stages, the (breadth) exploration can be temporarily localized under that node instead of the same level as it. Overall, value estimation a complex function (which is to be learned through RL) of multiple attributes of the MDP state such as current remaining budget, graph structure and relative performance at various nodes, etc. Note that the runtime explore/exploit trade-off mentioned above is different from the explore/exploit tradeoff seen in RL training in context of selecting actions to balance reward and not getting stuck in a local optimum. For the latter, we employ an $\epsilon - Greedy$ methodology, where an action is chosen at *random* with probability ϵ (random exploration), and from the *current policy* with probability $1 - \epsilon$ (policy exploitation). The trade-off in this case is exercised randomly and is independent of the state of MDP. The value of ϵ is constant and is chosen based on experimentation.

At step i, the occurrence of an action results in a new node, n_i, and hence a new dataset on which a model is trained and tested, and its best ensemble error $E(n_i)$ is obtained with respect to the set of nodes $\{0, 1 \ldots n_i\}$, using the algorithm 1. To each step, we attribute an immediate scalar reward (with a slight abuse of notation):

$$r_i = \frac{E_{min}(nodes(G_i)) - E_{min}(nodes(G_i))}{E(X_0)}$$

with $r_0 = 0$, by definition. The cumulative reward over time from state s_i onwards is defined as $R(s_i) = \sum_{j=0}^{B_{max}} \gamma^i . r_{i+j}$, where $\gamma \in [0, 1)$ is a discount factor, which prioritizes earlier rewards over the later ones. The objective is to find the optimal policy Π^* that maximizes the cumulative reward.

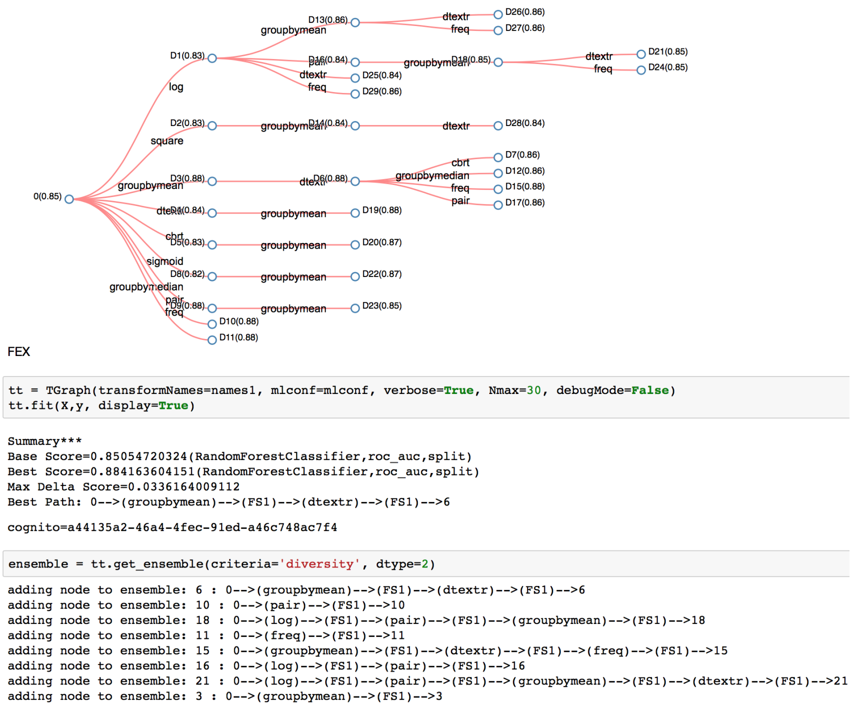
We utilize Q-learning Watkins and Dayan (1992) with function approximation to learn the action-value Q-function. For each state, $s \in S$ and action, $c \in C$, Q-function with respect to policy Π is defined as: $Q(s, c) = r(s, c) + \gamma R^{\Pi}(\delta(s, c))$, where $\delta : S \times C \to S$ is a hypothetical transition function, and $R^{\Pi}(s)$ is the cumulative reward following state s. The optimal policy is: $\Pi^*(s) = \arg\max_c [Q(s, c)]$. Due to the very large number of states, S, it is infeasible to learn Q-function directly. Instead, a linear approximation the Q-function is used as follows: $Q(s, c) = w.f(s)$, where w is a weight vector and $f(s) = f(g, n, t, b)$ is a vector of the state characteristics described in the previous section and the remaining budget ratio. Therefore, we approximate the Q-functions with linear combinations of characteristics of a state of the MDP. The update rule for w is as follows, where g' is the state of the graph at step $j+1$, and α is the learning rate parameter:

$$w^j \leftarrow w^j + \alpha.(r_j + \gamma.\max_{n',t'} Q(g',c') - Q(g,c)).f^j(g,b) \tag{2}$$

4. Experiments

We now summarize the results of proposed ensembles over a wide variety of OpenML datasets[1]. The detailed results can be found in the Appendix (Section 5). We randomly split each dataset into 70% fraction for training, feature engineering and building ensembles; the remaining 30% are used for evaluation on which all the results are reported. We used the following set of transforms: *cube root, sin, cos, tan, log, square, sigmoid, frequency, groupby+mean, groupby+median*; exploration was run for $B_{max} = 50$ iterations. For classification problems, we used the Random Forest classifier from Scikit-learn (Pedregosa et al. (2011)) as the learning algorithm across the board and measured performance through Area Under ROC curve (AUROC); the error measured hence was 1 - AUROC. We conducted the evaluation on 40 classification datasets and on average obtained a 13% reduction in error using feature engineering alone, and a 47% reduction in error through proposed ensembles over base dataset, respectively. We obtained similar improvements with regression datasets.

Figure 1: Jupyter Notebook based tool for Feature Engineering and Ensembles that extends Nargesian et al. (2018).



```
[63]: tt = TGraph(transformNames=names1, mlconf=mlconf, verbose=True, Nmax=30, debugMode=False)
      tt.fit(X,y, display=True)

      Summary***
      Base Score=0.85054720324(RandomForestClassifier,roc_auc,split)
      Best Score=0.884163604151(RandomForestClassifier,roc_auc,split)
      Max Delta Score=0.0336164009112
      Best Path: 0-->(groupbymean)-->(FS1)-->(dtextr)-->(FS1)-->6

[63]: cognito=a44135a2-46a4-4fec-91ed-a46c748ac7f4

[64]: ensemble = tt.get_ensemble(criteria='diversity', dtype=2)

      adding node to ensemble: 6 : 0-->(groupbymean)-->(FS1)-->(dtextr)-->(FS1)-->6
      adding node to ensemble: 10 : 0-->(pair)-->(FS1)-->10
      adding node to ensemble: 18 : 0-->(log)-->(FS1)-->(pair)-->(FS1)-->(groupbymean)-->(FS1)-->18
      adding node to ensemble: 11 : 0-->(freq)-->(FS1)-->11
      adding node to ensemble: 15 : 0-->(groupbymean)-->(FS1)-->(dtextr)-->(FS1)-->(freq)-->(FS1)-->15
      adding node to ensemble: 16 : 0-->(log)-->(FS1)-->(pair)-->(FS1)-->16
      adding node to ensemble: 21 : 0-->(log)-->(FS1)-->(pair)-->(FS1)-->(groupbymean)-->(FS1)-->(dtextr)-->(FS1)-->21
      adding node to ensemble: 3 : 0-->(groupbymean)-->(FS1)-->3
```

1. Open ML data repository: https://www.openml.org/search?type=data

References

Leo Breiman. Random forests. *UC Berkeley TR567*, 1999.

Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. Diversity creation methods: A survey and categorisation. *Information Fusion*, 6(1):5–20, 2005. ISSN 15662535. doi: 10.1016/j.inffus.2004.04.004.

P Cunningham and J Carney. Diversity versus quality in classification ensembles based on feature selection. *Machine Learning: ECML 2000*, pages 109–116, 2000.

Thomas G. Dietterich. Ensemble Methods in Machine Learning. *Multiple Classifier Systems*, 1857:1–15, 2000. ISSN 0010-4485. doi: 10.1007/3-540-45014-9.

Ofer Dor and Yoram Reich. Strengthening Learning Algorithms by Feature Discovery. *Information Sciences*, 189:176–190, 2012.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997. ISSN 0022-0000. doi: 10.1006/jcss.1997.1504. URL http://dx.doi.org/10.1006/jcss.1997.1504.

Funda Gunes, Russ Wolfinger, and Pei-Yi Tan. Stacked Ensemble Models for Improved Prediction Accuracy. pages 1–19, 2017.

James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *IEEE International Conference on Data Science and Advanced Analytics*, pages 1–10, 2015.

Gilad Katz, Eui Chul, Richard Shin, and Dawn Song. ExploreKit: Automatic Feature Generation and Selection. In *Proceedings of the IEEE 16th International Conference on Data Mining*, pages 979–984, 2016.

Udayan Khurana. Transformation-based feature engineering in supervised learning: Strategies toward automation. In Guozhu Dong and Huan Liu, editors, *Feature Engineering for Machine Learning and Data Analytics*, chapter 9, pages 221–243. Chapman & Hall/CRC, 2018.

Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasarathy. Cognito: Automated feature engineering for supervised learning. In *Proceedings of the IEEE 16th International Conference on Data Mining Workshops*, pages 1304–1307, 2016.

Udayan Khurana, Horst Samulowitz, and Deepak Turaga. Feature engineering for predictive modeling using reinforcement learning. *Proceedings of AAAI Conference on Artificial Intelligence*, 2018.

A Krogh and J Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in neural network processing systems*, 7:8–231, 1995. ISSN 10495258. doi: 10.1.1.37.8876.

Shaul Markovitch and Dan Rosenstein. Feature Generation using General Constructor Functions. *Machine Learning*, 2002.

Prem Melville and Raymond J. Mooney. Constructing diverse classifier ensembles using artificial training examples. *IJCAI International Joint Conference on Artificial Intelligence*, (August):505–510, 2003. ISSN 10450823.

Fatemeh Nargesian, Horst Samulowitz, Udayan Khurana, Elias B. Khalil, and Deepak Turaga. Learning feature engineering for classification. In *Proceedings of the Twenty-sixth International Joint Conference on Artificial Intelligence*, pages 2529–2535, 2017.

Fatemeh Nargesian, Udayan Khurana, Horst Samulowitz, and Deepak Turaga. Dataset evolver: An interactive feature engineering notebook. *Proceedings of AAAI Conference on Artificial Intelligence*, 2018.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Peter Sollich and Anders Krogh. Learning with ensembles: How over-fitting can be useful. *Proceedings of the 1995 Conference*, pages 4–10, 1996. ISSN 0262201070.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Gabriele Zenobi and Pádraig Cunningham. Using Diversity in Preparing Ensembles of Classifiers Based on Different Feature Subsets to Minimize Generalization Error. *Machine Learning: ECML 2001*, 2167(1995):576–587, 2001. ISSN 16113349. doi: 10.1007/3-540-44795-4_49.

5. Appendix

Table 1: Random Forest Classifier on OpenML datasets. Average error reduction by Ensemble+FE=47%, FE alone=13%. Results on 30% randomly selected test set.

Dataset	Base	FE	FE+Ensemble
letter	0.9966108870656221	0.9953327461796382	0.9998315795979542
bank-marketing	0.8855916878258837	0.8923103907613512	0.9262943079440853
puma8NH	0.8776937028617386	0.8777043718044464	0.9033604433903163
puma32H	0.9110996461053803	0.9494450724348115	0.9587070940407212
MagicTelescope	0.9095447243878099	0.9157578213367308	0.9337497185608221
musk	0.9999639953914101	1.0	1.0
delta_elevators	0.934670943320951	0.9295367687663142	0.9458541834420338
numerai28.6	0.49993885644178376	0.510337456766305	0.5164687487318682
Satellite	0.9204799606369655	0.920569422079084	0.9375670960815887
mc1	0.8738547948989457	0.874115679143072	0.9198087871951874
elevators	0.8777608136089918	0.9202388126200366	0.9418008614752892
letter-challenge	0.9864923670352769	0.9908006404977245	0.9949924431539927
sylva_agnostic	0.9963341571182506	0.9963903242716992	0.9983078708904366
nomao	0.9897707204793937	0.9886614893133139	0.993876695957177
sylva_prior	0.9981572342320459	0.9986167111974739	0.9992696947615578
house_16H	0.9337797976616882	0.9332133349147937	0.951447760512366
Run_or_walk	0.9980922332602676	0.997858869874573	0.9991247656139529
wind	0.9196093277960062	0.9201527778840034	0.9360560225232228
fried	0.9661912095401244	0.974430182903653	0.9814278254512926
cpu_act	0.9693858741119918	0.9698463790603944	0.9786827397748658
bank8FM	0.9811563424847973	0.9789775037171602	0.9868396887535644
electricity	0.9539766327196657	0.9599501186476314	0.9704407156950852
houses	0.9959766484510233	0.9971818541136945	0.9987190526689121
BNG(breast-w)	0.9968744019330839	0.9968052992907832	0.998535537317138
cpu_small	0.9643294352467247	0.9577432691664927	0.9719213522369938
kin8nm	0.8748987025161199	0.8935702927880572	0.918131373528175
pendigits	0.9994060862103784	0.9994695676699578	0.999758429613547
house_8L	0.9298770158933359	0.9298770158933359	0.9470661098153816
ringnorm	0.9845483144439143	0.990248014615063	0.9965075876356996
ailerons	0.9320882531371242	0.9376464052627003	0.9527538664519372
mv	0.9999280803800221	0.9999717052620748	0.9999802373198177
2dplanes	0.9677205544248797	0.9714907934558795	0.9781168110502179
CreditCardSubset	0.9994216310005783	0.9995129524215397	0.9996955952634623
bank32nh	0.8177800257483061	0.8496702007605634	0.8795565813536013
page-blocks	0.9714977752951827	0.9768540811937002	0.9870113193494744
eeg-eye-state	0.9544718380028294	0.9699032405090264	0.9850866113197818
JapaneseVowels	0.9950423249657006	0.9950500289401475	0.9988839285714286
optdigits	0.9807129200886704	0.9906692835864503	0.996768548689317
twonorm	0.9895636011045016	0.9931679936118742	0.9959346156556048
pol	0.9978300919523928	0.9978300919523928	0.999059737291541