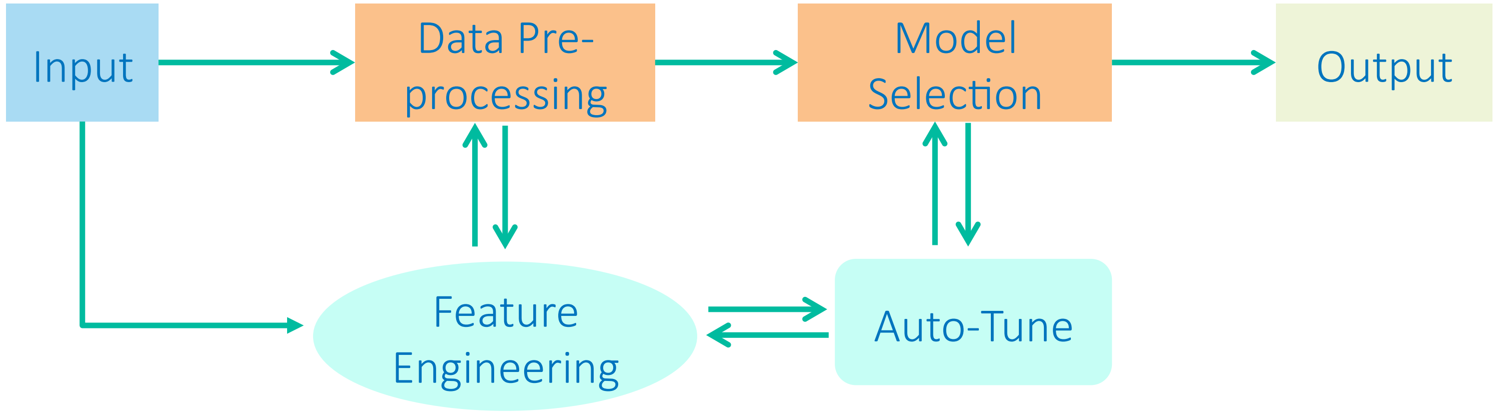
Automated Feature Engineering

Xin Hunt

SAS Institute Inc

Automated Machine Learning Pipeline



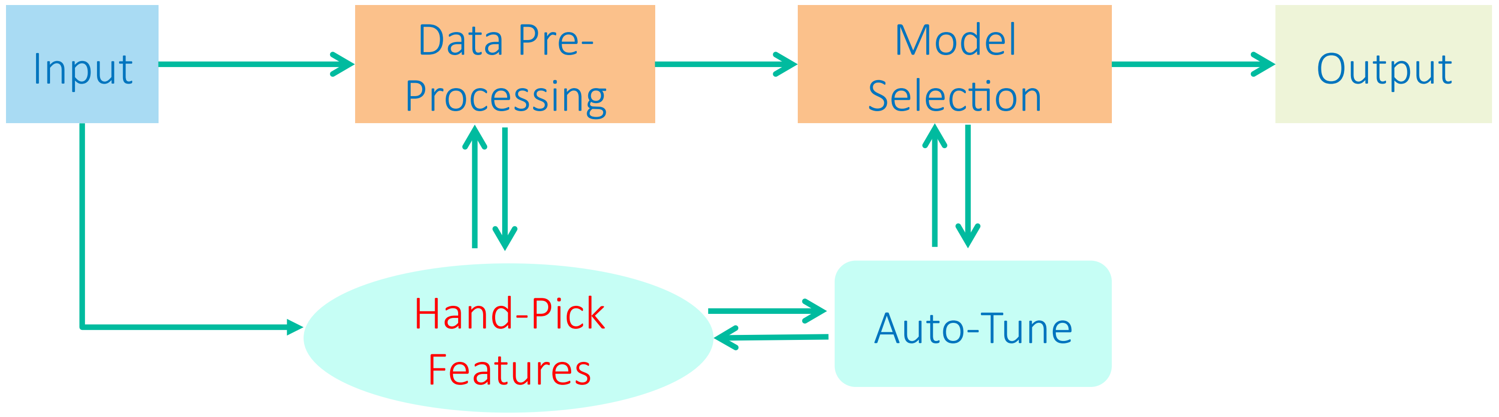
Motivation

"Garbage in, garbage out"

- Data Engineering is the process of cleaning, filtering, and organizing the data for successful mining and modeling, by solving or avoiding problems in the data.
- Could take 60-80% of the whole data mining effort.
- Feature Engineering methods allow us to choose the right representation to train our models.
- Part of the Automation Initiative at SAS®: Automated Feature Engineering
  - Envisioned for SAS® Visual Data Mining and Machine Learning
  - Runs on SAS® Viya®: tested and optimized for Intel® Xeon® for performance and scalability

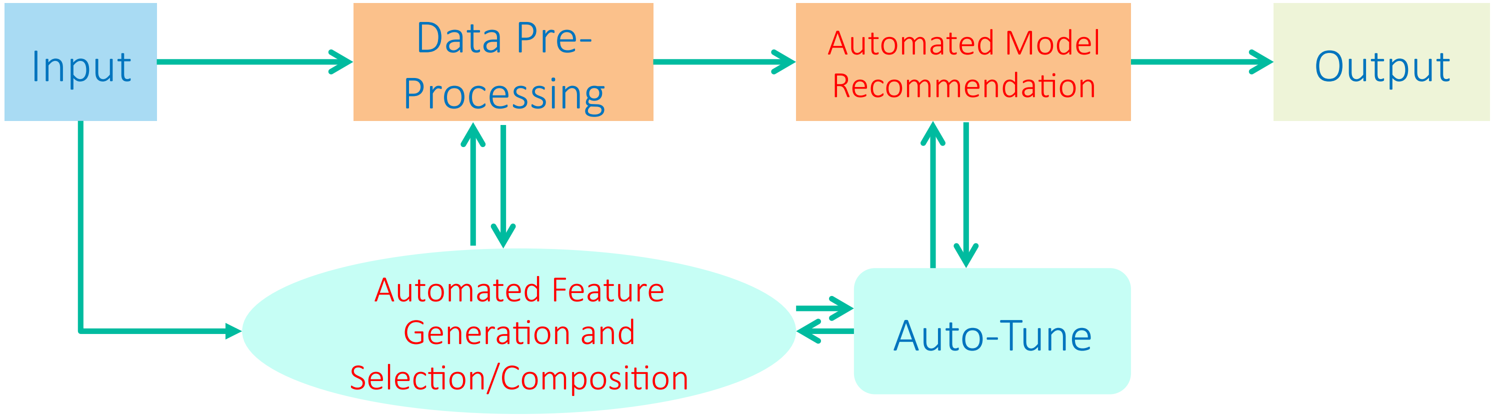
Traditional Feature Engineering

- Performed by data scientists
- Relies heavily on model selected and domain expertise
- Features are designed through trial and error



Automated Feature Engineering

- Performed by data scientists
- Assisted by automated feature generation, selection, and composition methods
- Reduces manual trial and error time
- Expands search width and depth for best features
- Combined with automated model recommendation


Input
Data Pre-Processing
Automated Model Recommendation
Output
Automated Feature Generation and Selection/Composition
Auto-Tune

Problem Formulation

Feature Selection and Composition

- Original dataset X
- Model m
- Set of transformations $T = \{t_i\}, i = 1, 2, \ldots,$ where each $t_i(\cdot)$ outputs a set of features
- Composition and concatenation of transformations

$$C(T, X) = [t_{i_1}(t_{i_2}(\ldots(X))), t_{j_1}(t_{j_2}(\ldots(X))), \ldots]$$

- Objective: find a composition of transformations that maximize model performance

$$[C^*, T^*] = \arg\max_{C,T} R_m(C(T, X))$$

- In reality, C and T are optimized separately

How To Build Good Features?

The two building blocks

- Feature generators
  - Domain specific feature generators
  - General purpose feature generators

- Feature selection and composition algorithm
  - The “best features” are both data and model specific
  - Need to combine with an efficient model selection and recommendation method

How To Find Good Features?

The two building blocks

- Feature generators
  - Domain specific feature generators
  - General purpose feature generators

- Feature selection and composition algorithm
  - The “best features” are both data and model specific
  - Need to combine with an efficient model selection and recommendation method

Feature Extraction and Generation

Domain Specific Features

- Text Data
  - Bag of words, semantic structural representation, latent semantic representations (latent Dirichlet allocation), Word2Vec embeddings
- Image Data
  - Color, texture, shape (edges, corners, blobs), wavelet coefficients, Scale-invariant features (SIFT), bag-of-features + spatial pyramid, deep learning based features
- Time series
  - Spectral features, motifs, shapelets, discords, pattern dictionaries

Feature Extraction and Generation

General Purpose Features

- Single-variable transformations
  - log, exponential, frequency count, one-hot coding, normalization
- Two-variable combinations
  - sum, difference, division, product
- Multivariate and model-based methods
  - Unsupervised feature generation
    - PCA, random projections, meta data learning, distance/cluster based features, relational feature generation, kernel manifold learning
  - Supervised feature generation
    - Linear discriminant analysis (LDA), supervised dictionary learning
  - Deep learning based methods
    - auto-encoders, mid layers of trained deep neural networks

How To Find Good Features?

The two building blocks

- Feature generators
  - Domain specific feature generators
  - General purpose feature generators

- Feature selection and composition algorithm
  - The “best features” are both data and model specific
  - Need to combine with an efficient model selection and recommendation method

Feature Selection and Composition

Pure Selection

- Examples: DSM [Kanter et al. 2015], OneBM [Lam et al. 2017]

- Select using statistics
  - Filter by variance, correlation, mutual information
- Select by model
  - Build models that encourage sparsity (e.g., L1 penalization)
  - Select by filtering out features with low weights
- Grid Search
  - Build model with random subsets of features
  - Compare and choose the subset with best performance

Feature Selection and Composition

Pure Selection

- Examples: DSM [Kanter et al. 2015], OneBM [Lam et al. 2017]

- Limitations:
  - Does not allow feature composition
  - Statistics and sparse model weights do not directly translate to performance when used to train the actual model
  - Grid search is computationally expensive, especially when the number of possible transformations is large

Feature Selection and Composition

Iterative Combination

- Example: ExploreKit [Katz et al., 2016]

- Greedy search for best feature combination
- Initialize with empty feature set $C_0(T, X) = \emptyset$
- At each iteration:
  - Find candidate feature with the highest performance improvement

    $$i_n = \arg\max_i R_m([C_{n-1}(T, X), t_i(X)])$$
  - Add best candidate to the feature set

    $$C_n(T, X) = [C_{n-1}(T, X), t_{i_n}(X)]$$
- Repeat until convergence (low improvement) or time budget is reached

Feature Selection and Composition

Iterative Combination

- Example: ExploreKit [Katz et al., 2016]

- A greedy selection algorithm
- More scalable than grid search

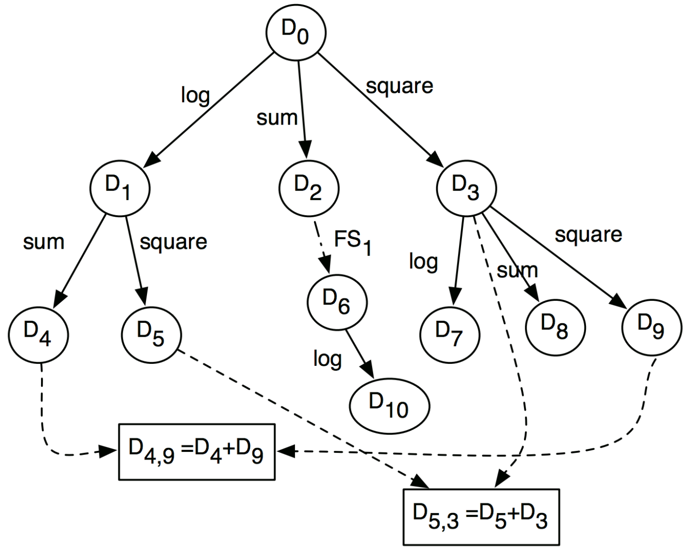
- Limitations:
  - Does not allow feature composition
  - Greedy, which may result in sub-optimal feature selection
  - Time consuming. Iterative algorithm is difficult to parallelize

Feature Selection and Composition

Hierarchical Search

- Example: Cognito [Khurana et al. 2016]

- Use a tree-like structure (transformation graph) to represent possible feature compositions
- Start with one node (original data)
- At each iteration
  - Evaluate possible child nodes based on criteria like node accuracy and depth
  - Add best child node to current structure
- Repeat until time budget is reached



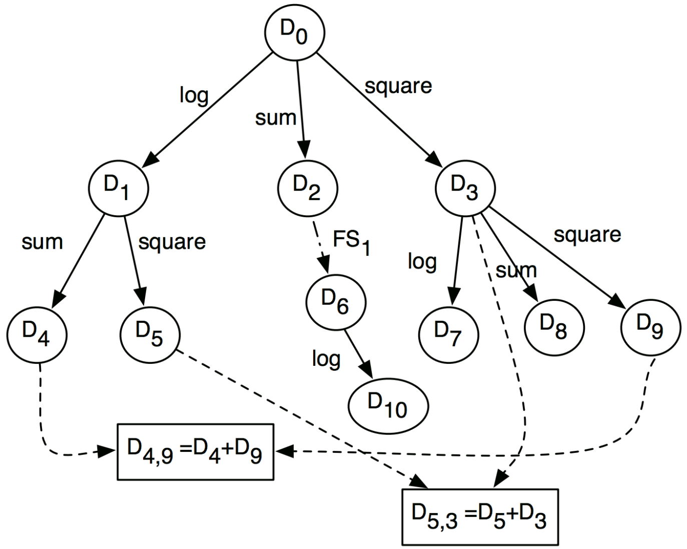
Feature Selection and Composition

Hierarchical Search

- Example: Cognito [Khurana et al. 2016]

- Allow feature composition
- Can generate different feature combinations by changing criteria

- Limitations:
  - Greedy algorithm may lead to sub-optimal solution
  - Time consuming (iterative training and validation)
  - Criteria setup is not intuitive



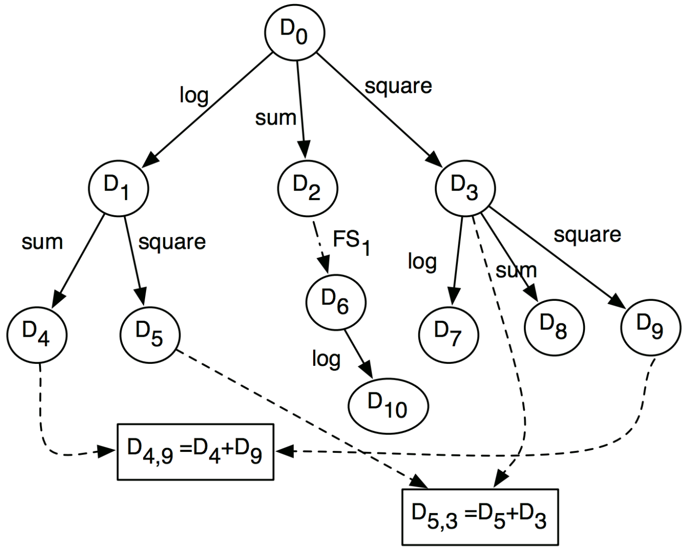
Feature Selection and Composition

Hierarchical Search

- [Khurana et al. 2017]

- Extension: reinforcement learning based search
  - State: a transformation graph and remaining budget value
  - Possible actions: Add any feasible child node to current state
  - Objective: learn optimal action policy given state

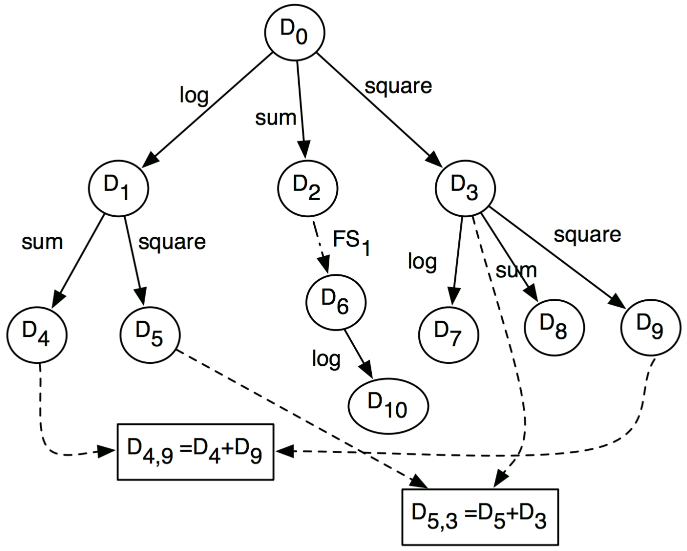
- Policy learned on multiple training datasets



Feature Selection and Composition

Hierarchical Search

- [Khurana et al. 2017]
- Extension: reinforcement learning based search
- Balance exploitation with exploration
- More efficient search with well-trained policy
- Policy training requires extra data, and can take a long time



RULLS

Unsupervised Feature Generation

Namita Lokare Jorge Silva Ilknur Kabul

RULLS

Feature Engineering Method

- Idea: Aggregating features from a random *union of subspace*s by describing points using globally chosen landmarks. Euclidean distances are encoded as features in the final feature matrix.

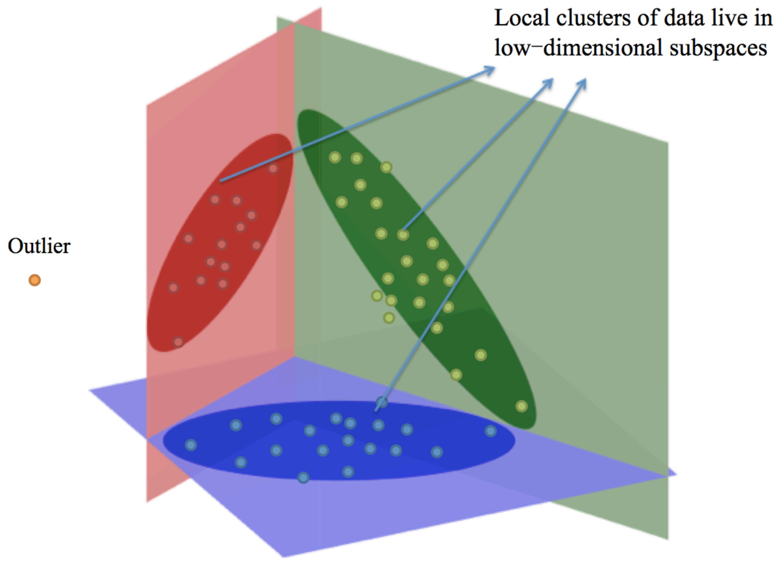
- Features generated are:
  - Sparse
  - Non-negative
  - Rotation invariant
  - Allow fast training when used in conjunction with simple models
  - Can be used for clustering tasks
  - Can be used for classification

RULLS

Union of Subspaces

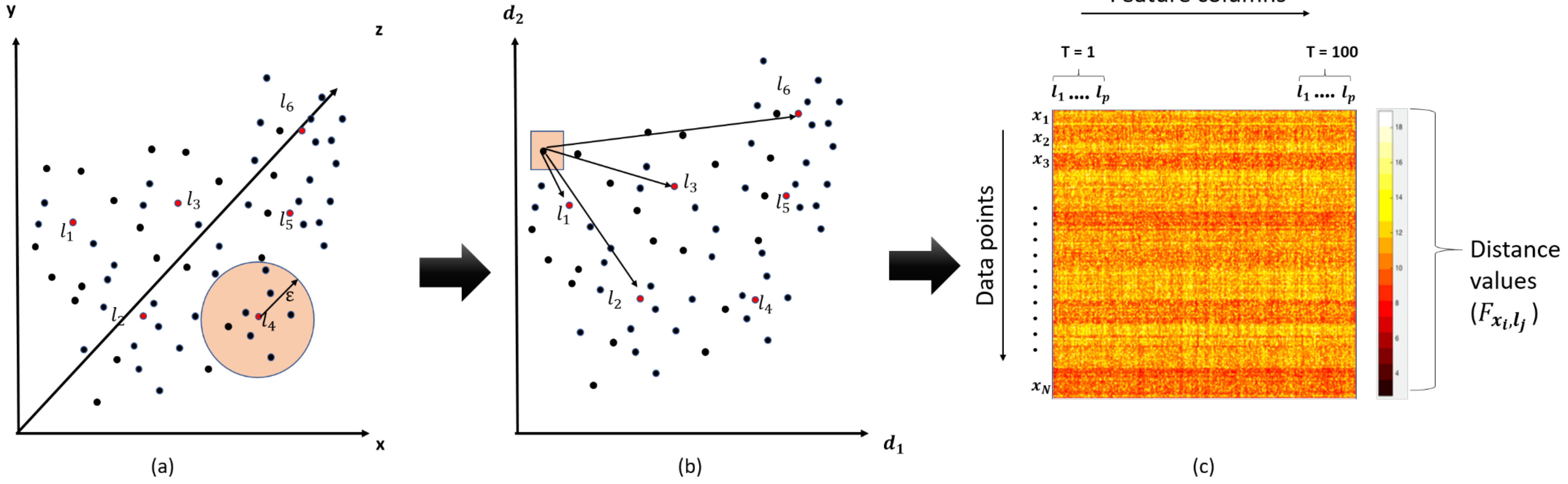
- Assumptions:
  - Globally the data may not be low-dimensional
  - Locally data exhibit low-dimensional structure (subspaces)

- Advantages:
  - Reduces local dimensionality without forcing global dimension reduction
  - Preserves local structure



Workings of RULLS

Pipeline



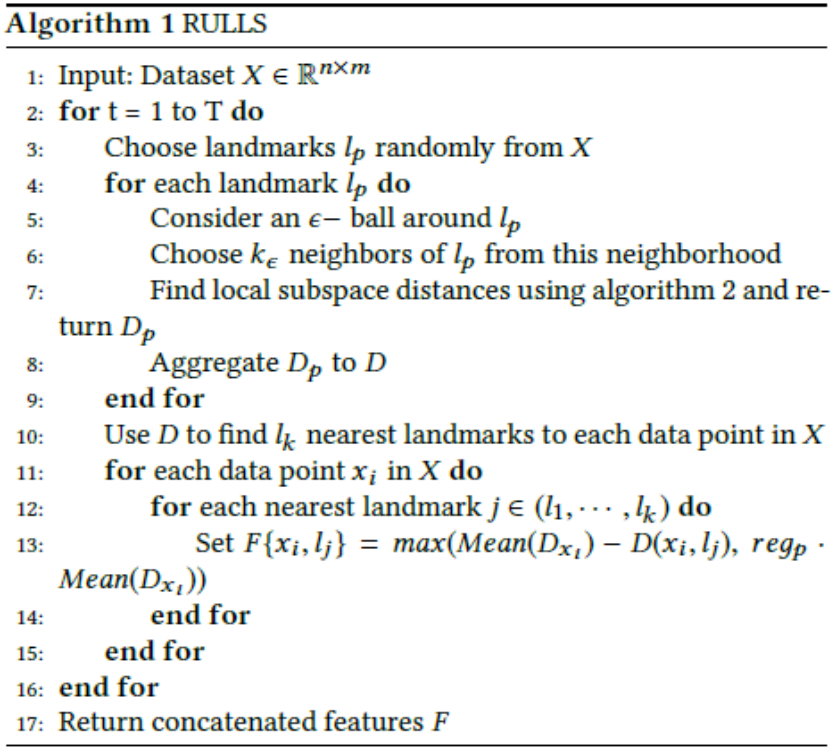
Randomly select landmarks
Construct local subspaces with landmarks' neighbors

Project onto the subspace of each landmark, measure distances to the landmark

Use regularized distances as features

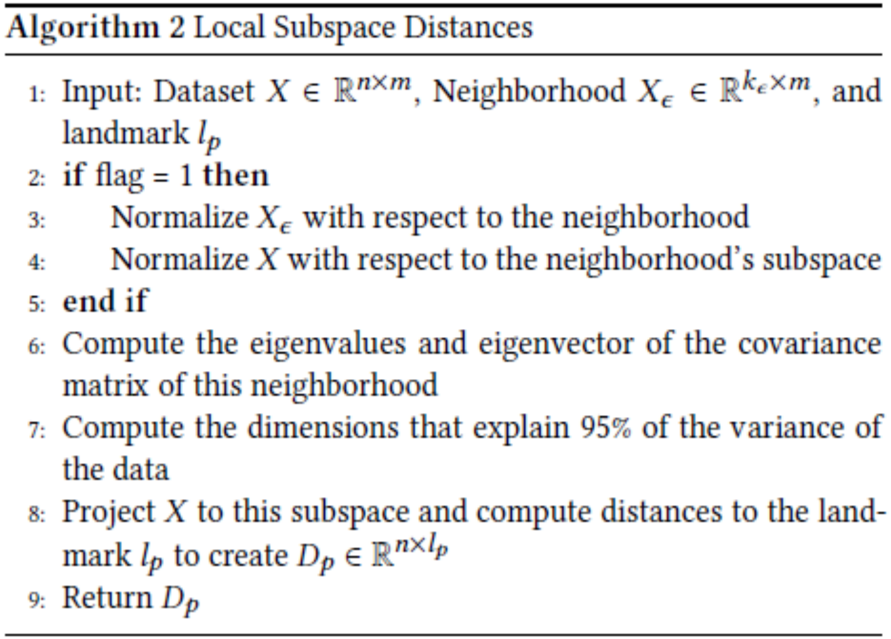
Workings of RULLS

Algorithm



Algorithm 1 RULLS

```
1: Input: Dataset $X \in \mathbb{R}^{n \times m}$
2: for t = 1 to T do
3:     Choose landmarks $l_p$ randomly from $X$
4:     for each landmark $l_p$ do
5:         Consider an $\epsilon-$ ball around $l_p$
6:         Choose $k_\epsilon$ neighbors of $l_p$ from this neighborhood
7:         Find local subspace distances using algorithm 2 and return $D_p$
8:         Aggregate $D_p$ to $D$
9:     end for
10:    Use $D$ to find $l_k$ nearest landmarks to each data point in $X$
11:    for each data point $x_i$ in $X$ do
12:        for each nearest landmark $j \in (l_1, \cdots, l_k)$ do
13:            Set $F\{x_i, l_j\} = max(Mean(D_{x_i}) - D(x_i, l_j), reg_p \cdot Mean(D_{x_i}))$
14:        end for
15:    end for
16: end for
17: Return concatenated features $F$
```



Algorithm 2 Local Subspace Distances

```
1: Input: Dataset $X \in \mathbb{R}^{n \times m}$, Neighborhood $X_\epsilon \in \mathbb{R}^{k_\epsilon \times m}$, and landmark $l_p$
2: if flag = 1 then
3:     Normalize $X_\epsilon$ with respect to the neighborhood
4:     Normalize $X$ with respect to the neighborhood's subspace
5: end if
6: Compute the eigenvalues and eigenvector of the covariance matrix of this neighborhood
7: Compute the dimensions that explain 95% of the variance of the data
8: Project $X$ to this subspace and compute distances to the landmark $l_p$ to create $D_p \in \mathbb{R}^{n \times l_p}$
9: Return $D_p$
```

Variants of RULLS

- Variant I
  - Random projections (no subspace learning)
- Variant II
  - Use Euclidean distance (no projection)
- Use Robust PCA in presence of noise and outliers

Existing Methods

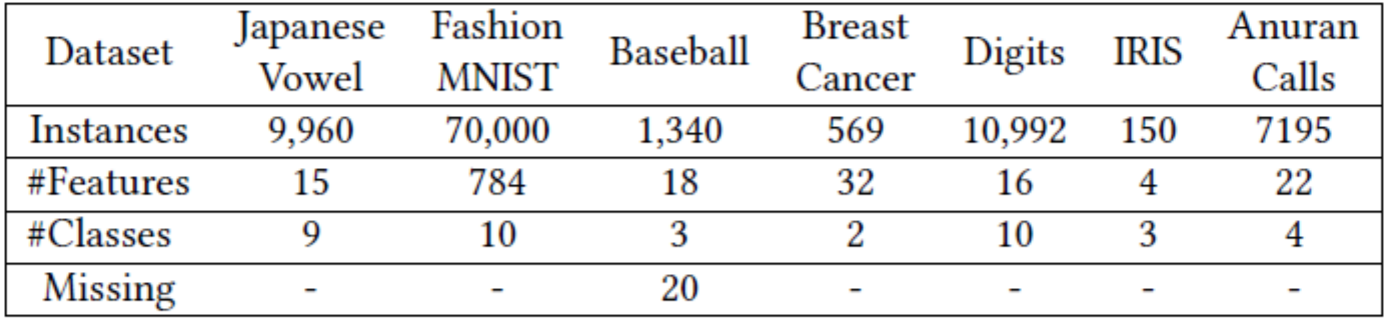
- RandLocal
  - Features are chosen randomly
  - Use only one global neighbor to encode distances
  - Suggested range for T is between 100 and 500

*Suhang Wang, Charu Aggarwal, and Huan Liu. 2017. **Randomized Feature Engineering As a Fast and Accurate Alternative to Kernel Methods**. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 485–494. https://doi.org/10.1145/3097983.3098001

Advantages of RULLS

- RULLS selects features that are locally relevant unlike RandLocal, Variant I, and Variant II
- RULLS can achieve a better performance than all the methods with fewer iterations
- Simple machine learning models when used in conjunction with the features generated by RULLS are fast and efficient to train
- RULLS allows for the use of robust PCA in presence of noise and outliers

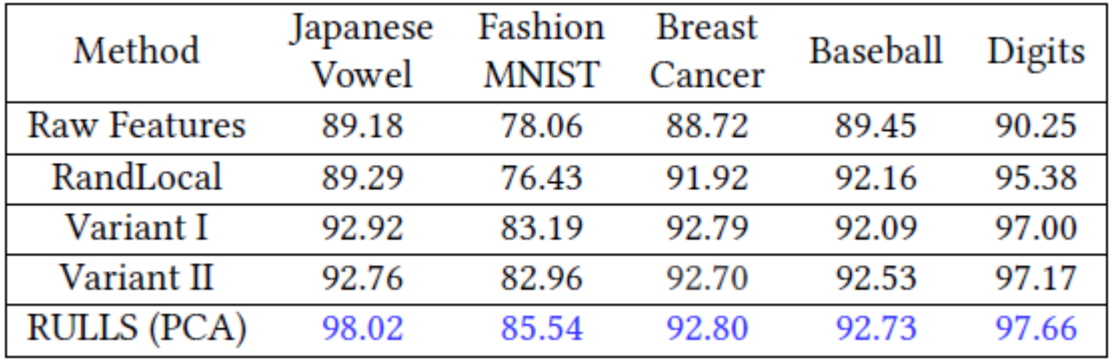
Datasets Tested



Dataset	Japanese Vowel	Fashion MNIST	Baseball	Breast Cancer	Digits	IRIS	Anuran Calls
Instances	9,960	70,000	1,340	569	10,992	150	7195
#Features	15	784	18	32	16	4	22
#Classes	9	10	3	2	10	3	4
Missing	-	-	20	-	-	-	-

Results

Classification Tasks



Method	Japanese Vowel	Fashion MNIST	Breast Cancer	Baseball	Digits
Raw Features	89.18	78.06	88.72	89.45	90.25
RandLocal	89.29	76.43	91.92	92.16	95.38
Variant I	92.92	83.19	92.79	92.09	97.00
Variant II	92.76	82.96	92.70	92.53	97.17
RULLS (PCA)	98.02	85.54	92.80	92.73	97.66

Classification accuracy on datasets. Highlighted text shows the method with the best performance.

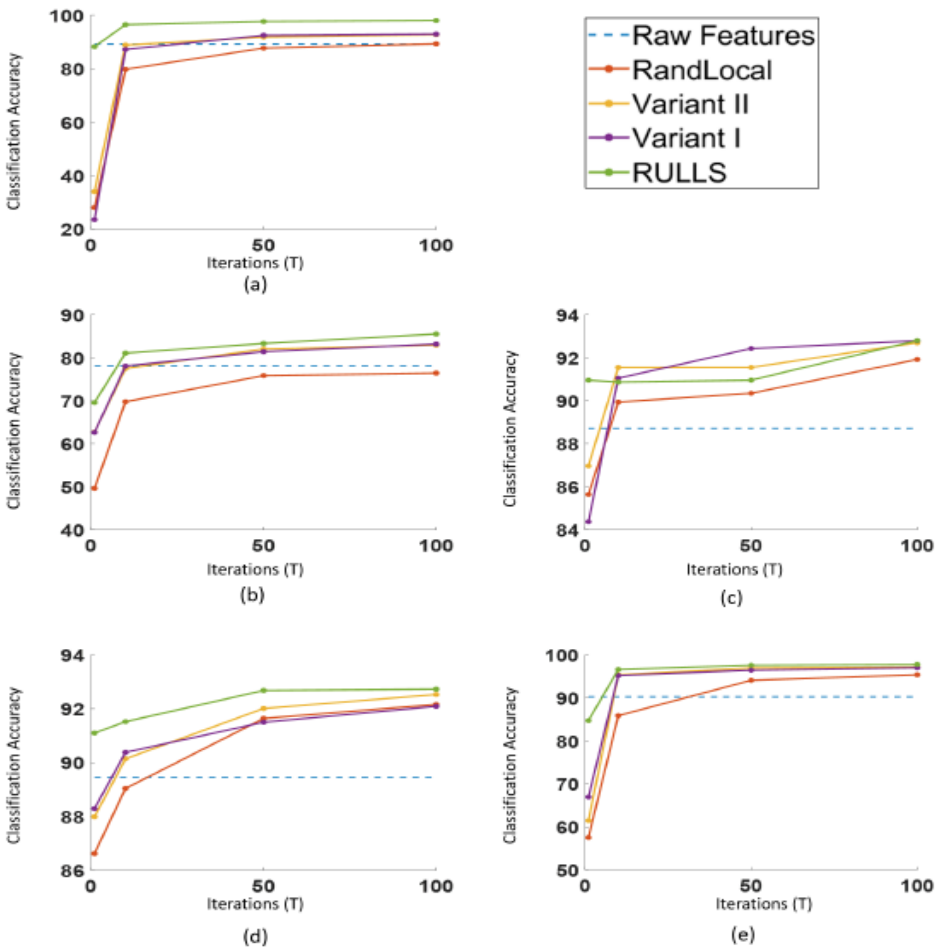
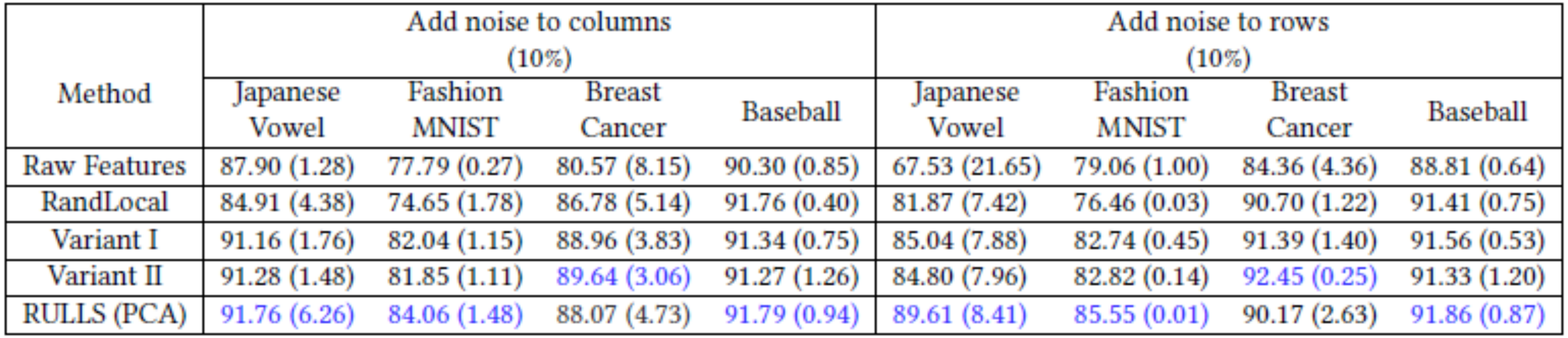


Figure 3: Average classification accuracy (%) with varying iterations (t = 1, 10, 50, and 100) for raw features, RandLocal, Variant I, Variant II and RULLS (PCA). (a) Japanese Vowel, (b) Fashion MNIST, (c) Breast Cancer Wisconsin, (d) Baseball and (e) Digits dataset. Methods compared here beat the raw features score in just a few iterations. RULLS performs better than other methods on all datasets compared.

Results

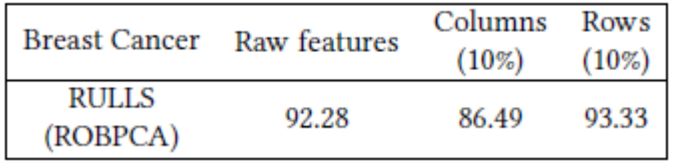
Classification Tasks in presence of noise

Classification performance in presence of 10% noise added to columns and rows in each dataset. Best performance is highlighted in blue. The numbers in the parenthesis indicate the difference between the performance with and without noise.



Method	Add noise to columns (10%)				Add noise to rows (10%)			
	Japanese Vowel	Fashion MNIST	Breast Cancer	Baseball	Japanese Vowel	Fashion MNIST	Breast Cancer	Baseball
Raw Features	87.90 (1.28)	77.79 (0.27)	80.57 (8.15)	90.30 (0.85)	67.53 (21.65)	79.06 (1.00)	84.36 (4.36)	88.81 (0.64)
RandLocal	84.91 (4.38)	74.65 (1.78)	86.78 (5.14)	91.76 (0.40)	81.87 (7.42)	76.46 (0.03)	90.70 (1.22)	91.41 (0.75)
Variant I	91.16 (1.76)	82.04 (1.15)	88.96 (3.83)	91.34 (0.75)	85.04 (7.88)	82.74 (0.45)	91.39 (1.40)	91.56 (0.53)
Variant II	91.28 (1.48)	81.85 (1.11)	89.64 (3.06)	91.27 (1.26)	84.80 (7.96)	82.82 (0.14)	92.45 (0.25)	91.33 (1.20)
RULLS (PCA)	91.76 (6.26)	84.06 (1.48)	88.07 (4.73)	91.79 (0.94)	89.61 (8.41)	85.55 (0.01)	90.17 (2.63)	91.86 (0.87)

RULLS with ROBPCA on the Breast Cancer dataset in the case of raw features and 10% noise added to columns and rows.

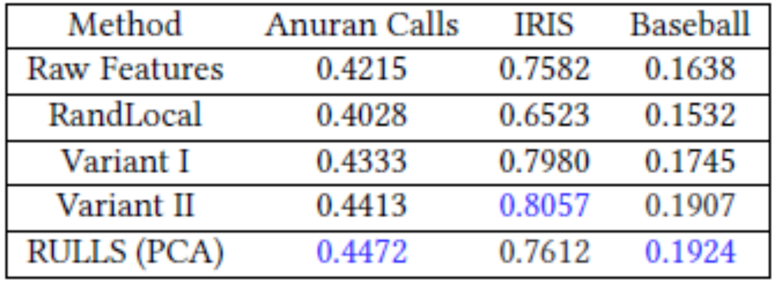


Breast Cancer	Raw features	Columns (10%)	Rows (10%)
RULLS (ROBPCA)	92.28	86.49	93.33

Results

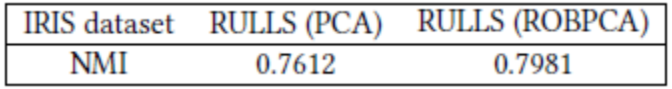
Clustering Tasks

Clustering performance on datasets. We report Normalized Mutual Information (NMI). Highlighted text shows the method with the best performance per dataset.



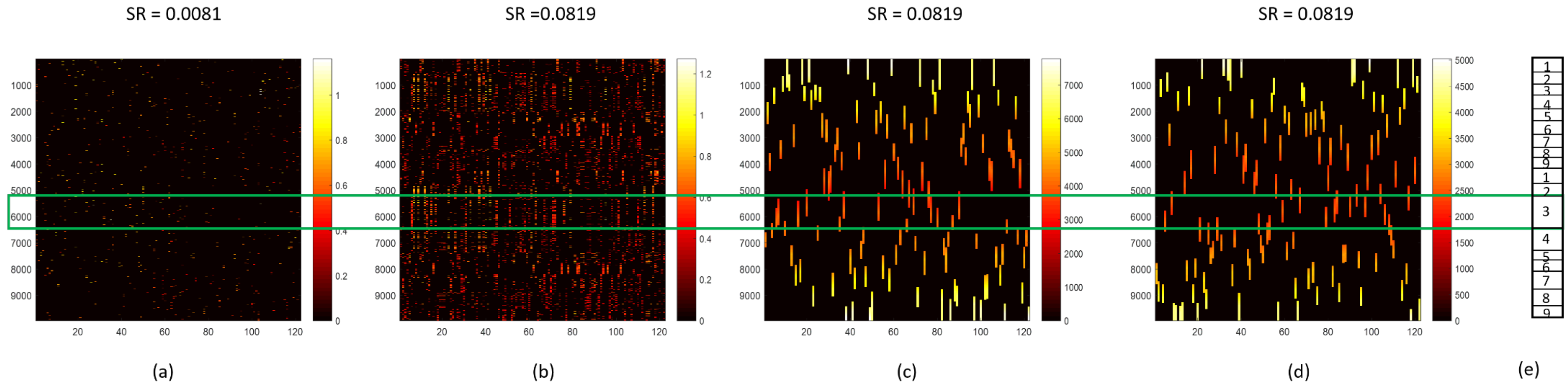
Method	Anuran Calls	IRIS	Baseball
Raw Features	0.4215	0.7582	0.1638
RandLocal	0.4028	0.6523	0.1532
Variant I	0.4333	0.7980	0.1745
Variant II	0.4413	0.8057	0.1907
RULLS (PCA)	0.4472	0.7612	0.1924

Comparison of RULLS with PCA and ROBPCA on IRIS dataset. We see an improvement in performance with ROBPCA



IRIS dataset	RULLS (PCA)	RULLS (ROBPCA)
NMI	0.7612	0.7981

Visual Comparison of features



Visual interpretation of Japanese Vowel dataset features. (a) RandLocal, (b) Variant II, (c) Variant I, (d) RULLS (PCA), (e) ground truth class labels. **The features are generated for $T = 1$, $\ell_p = 122$, $\ell_k = 10$ for RULLS (PCA), Variant I, and Variant II, and $\ell_k = 1$ for RandLocal.**

References

- Dong, Guozhu, and Huan Liu. "Feature Engineering for Machine Learning and Data Analytics." (2018).
- Katz, Gilad, Eui Chul Richard Shin, and Dawn Song. "Explorekit: Automatic feature generation and selection." In Data Mining (ICDM), 2016 IEEE 16th International Conference on, pp. 979-984. IEEE, 2016.
- Khurana, Udayan, Horst Samulowitz, and Deepak Turaga. "Feature Engineering for Predictive Modeling using Reinforcement Learning." arXiv preprint arXiv:1709.07150 (2017).
- Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. (2017). arXiv:cs.LG/cs.LG/1708.07747
- Lu, Yue M., and Minh N. Do. "A theory for sampling signals from a union of subspaces." IEEE transactions on signal processing 56, no. 6 (2008): 2334-2345.
- UCI Machine Learning Repository. 2013. http://archive.ics.uci.edu/ml
- Wang, Suhang, Charu Aggarwal, and Huan Liu. "Randomized Feature Engineering as a Fast and Accurate Alternative to Kernel Methods." In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 485-494. ACM, 2017.
- Namita lokare, Jorge Silva, and Ilknur Kaynar Kabul. "RULLS: Randomized Union of Locally Linear Subspaces for Feature Engineering." arXiv preprint arXiv:1804.09770 (2018).Feature Engineering for Machine Learning and Data Analytics
- Khurana, Udayan, Fatemeh Nargesian, Horst Samulowitz, Elias Khalil, and Deepak Turaga. "Automating Feature Engineering." Transformation 10, no. 10 (2016): 10.
- Hamaad Shah. “Automatic feature engineering using deep learning and Bayesian inference.” https://towardsdatascience.com/automatic-feature-engineering-using-deep-learning-and-bayesian-inference-application-to-computer-7b2bb8dc7351

Union of Subspaces

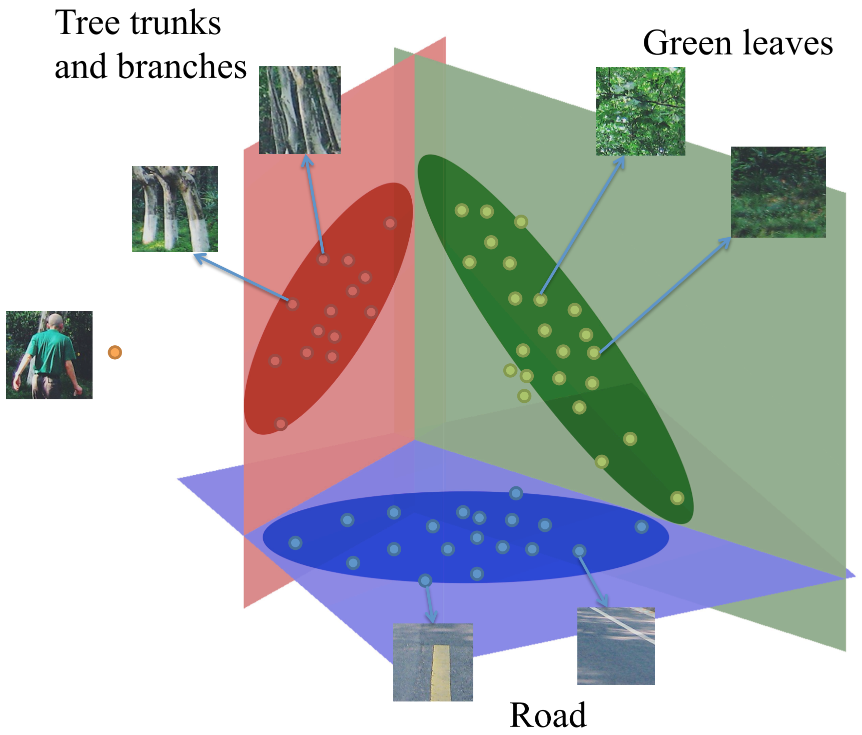
Example: Image Analysis

Union of Subspaces

Example: Image Analysis

Union of Subspaces

Example: Image Analysis


Tree trunks
and branches
Green leaves
Road