Learning Feature Engineering for Classification

Fatemeh Nargesian[1], Horst Samulowitz[2], Udayan Khurana[2]
Elias B. Khalil[3], Deepak Turaga[2]
[1]University of Toronto, [2]IBM Research, [3]Georgia Institute of Technology
fnargesian@cs.toronto.edu, {samulowitz, ukhurana}@us.ibm.com,
lyes@gatech.edu, turaga@us.ibm.com

Abstract

Feature engineering is the task of improving predictive modelling performance on a dataset by transforming its feature space. Existing approaches to automate this process rely on either transformed feature space exploration through evaluation-guided search, or explicit expansion of datasets with all transformed features followed by feature selection. Such approaches incur high computational costs in runtime and/or memory. We present a novel technique, called *Learning Feature Engineering* (LFE), for automating feature engineering in classification tasks. LFE is based on learning the effectiveness of applying a *transformation* (e.g., arithmetic or aggregate operators) on numerical features, from past feature engineering experiences. Given a new dataset, LFE recommends a set of useful transformations to be applied on features without relying on model evaluation or explicit feature expansion and selection. Using a collection of datasets, we train a set of neural networks, which aim at predicting the transformation that impacts classification performance positively. Our empirical results show that LFE outperforms other feature engineering approaches for an overwhelming majority (89%) of the datasets from various sources while incurring a substantially lower computational cost.

1 Introduction

Feature engineering is a central task in data preparation for machine learning. It is the practice of constructing suitable features from given features that lead to improved predictive performance. Feature engineering involves the application of *transformation* functions such as arithmetic and aggregate operators on given features to generate new ones. Transformations help scale a feature or convert a non-linear relation between a feature and a target class into a linear relation, which is easier to learn.

Feature engineering is usually conducted by a data scientist relying on her domain expertise and iterative trial and error and model evaluation. To perform automated feature engineering, some existing approaches adopt guided-search in feature space using heuristic feature quality measures (such as information gain) and other surrogate measures of performance [Markovitch and Rosenstein, 2002; Fan *et al.*, 2010]. Others perform greedy feature construction and selection based on model evaluation [Dor and Reich, 2012; Khurana *et al.*, 2016]. Kanter et al. proposed the Data Science Machine (DSM) which considers feature engineering problem as feature selection on the space of novel features. DSM relies on exhaustively enumerating all possible features that can be constructed from a dataset, given sequences generated from a set of transformations, then performing feature selection on the augmented dataset [Kanter and Veeramachaneni, 2015]. Evaluation-based and exhaustive feature enumeration and selection approaches result in high time and memory cost and may lead to overfitting due to brute-force generation of features. Moreover, although deep neural networks (DNN) allow for useful meta-features to be learned automatically [Bengio *et al.*, 2013], the learned features are not always interpretable and DNNs are not effective learners in various application domains.

In this paper, we propose *LFE (Learning Feature Engineering)*, a novel meta-learning approach to automatically perform interpretable feature engineering for classification, based on learning from past feature engineering experiences. By generalizing the impact of different transformations on the performance of a large number of datasets, LFE learns useful patterns between features, transforms and target that improve learning accuracy. We show that generalizing such patterns across thousands of features from hundreds of datasets can be used to successfully predict suitable transformations for features in new datasets without actually applying the transformations, performing model building and validation tasks, that are time consuming. LFE takes as input a dataset and recommends a set of paradigms for constructing new useful features. Each paradigm consists of a transformation and an ordered list of features on which the transformation is suitable.

At the core of LFE, there is a set of Multi-Layer Perceptron (MLP) classifiers, each corresponding to a transformation. Given a set of features and class labels, the classifier predicts whether the transformation can derive a more useful feature than the input features. LFE considers the notion of feature and class relevance in the context of a transformation as the measure of the usefulness of a pattern of feature value and class label distributions, and transformation.

Different datasets contain different feature sizes and different value ranges. One key challenge in generalizing across different datasets is to convert feature values and their class labels to a fixed size feature vector representation that can be fed into LFE classifiers. To characterize datasets, hand-crafted meta-features, fixed-size stratified sampling, neural networks and hashing methods have been used for different tasks [Michie *et al.*, 1994; Kalousis, 2002; Feurer *et al.*, 2015; Weinberger *et al.*, 2009]. However, these representations do not directly capture the correlation between feature values and class labels. To capture such correlations, LFE constructs a stack of fixed-size representations of feature values per target class. We use Quantile Data Sketch to represent feature values of each class. Quantile has been used as a fixed-size space representation and achieves reasonably accurate approximation to the distribution function induced by the data being sketched [Greenwald and Khanna, 2001].

LFE presents a computationally efficient and effective alternative to other automated feature engineering approaches by recommending suitable transformations for features in a dataset. To showcase the capabilities of LFE, we trained LFE on 85K features, extracted from 900 datasets, for 10 unary transformations and 122K feature pairs for 4 binary transformations, for two models: Random Forest and Logistic Regression. The transformations are listed in Table 1. We empirically compare LFE with a suite of feature engineering approaches proposed in the literature or applied in practice (such as the Data Science Machine, evaluation-based, random selection of transformations and always applying the most popular transformation in the training data) on a subset of 50 datasets from UCI repository [Lichman, 2013], OpenML [Vanschoren *et al.*, 2014] and other sources. Our experiments show that, of the datasets that demonstrated any improvement through feature engineering, LFE was the most effective in 89% of the cases. As shown in Figure 2, similar results were observed for the LFE trained with Logistic Regression. Moreover, LFE runs in significantly lesser time compared to the other approaches. This also enables interactions with a practitioner since it recommends transformations on features in a short amount of time.

2 Related Work

The FICUS algorithm [Markovitch and Rosenstein, 2002] takes as input a dataset and a set of transformations, and performs a beam search over the space of possible features. FICUS's search for better features is guided by heuristic measures based on information gain in a decision tree, and other surrogate measures of performance. The more recent FCTree [Fan *et al.*, 2010] also uses a decision tree to partition the data using original or constructed features as splitting points. The FEADIS algorithm of [Dor and Reich, 2012] relies on a combination of random feature generation and feature selection. FEADIS adds constructed features greedily, and as such requires many expensive performance evaluations. The Deep Feature Synthesis component of Data Science Machine (DSM) [Kanter and Veeramachaneni, 2015] relies on exhaustively enumerating all possible new features, then performing feature selection over the augmented dataset. Cognito [Khurana *et al.*, 2016] recommends a series of transformations based on a greedy, hierarchical heuristic search. Cognito and DSM focus on sequences of feature transformations, which is outside the scope of this paper. In contrast to these approaches, we do not require expensive classifier evaluations to measure the impact of a transformation. ExploreKit [Katz *et al.*, 2016] also generates all possible candidate features, but uses a learned ranking function to sort them. ExploreKit requires generating all possible candidate features when engineering features for a new (test) dataset and as such, results reported for ExploreKit are with a time limit of three days per dataset. In contrast, LFE can generate effective features within seconds, on average.

Several machine learning methods perform feature extraction or learning indirectly. While they do not explicitly work on input features and transformations, they generate new features as means to solving another problem [Storcheus *et al.*, 2015]. Methods of that type include dimensionality reduction, kernel methods and deep learning. Kernel algorithms such as SVM [Shawe-Taylor and Cristianini, 2004] can be seen as embedded methods, where the learning and the (implicit) feature generation are performed jointly. This is in contrast to our setting, where feature engineering is a preprocessing step.

Deep neural networks learn useful features automatically [Bengio *et al.*, 2013] and have shown remarkable successes on video, image and speech data. However, in some domains feature engineering is still required. Moreover, features derived by neural networks are often not interpretable which is an important factor in certain application domains such as healthcare [Che *et al.*, 2015].

3 Automated Feature Engineering Problem

Consider a dataset, $\mathcal{D}$, with features, $\mathcal{F} = \{f_1, \ldots, f_n\}$, and a target class, κ, a set of transformations, $\mathcal{T} = \{T_1, \ldots, T_m\}$, and a classification task, $\mathcal{L}$. The feature engineering problem is to find q best paradigms for constructing new features such that appending the new features to $\mathcal{D}$ maximizes the accuracy of $\mathcal{L}$. Each paradigm consists of a candidate transformation $T_c \in \mathcal{T}$ of arity r, an ordered list of features $[f_i, \ldots, f_{i+r-1}]$ and a usefulness score.

For a dataset with n features and with u unary transformations, $O(u \times n)$ new features can be constructed. With b binary transformations, there are $O(b \times P_2^n)$ new possible features, where P_2^n is the 2-permutation of n features. Given a fixed set of transformations, the number of new features and their combinations to explore, for an exact solution, grows exponentially. Hence, we make the case that a mere enumeration and trial by model training and testing is not a computationally practical option, and a scalable solution to the problem must avoid this computational bottleneck. LFE reduces the complexity of feature space exploration by providing an efficient approach for finding a particularly "good" transformation for a given set of features. Therefore, given n features, for unary and binary transformations, LFE performs, respectively, $O(n)$ and $O(P_2^n)$ transformation predictions.

In order to assess the relative impact of adding new features across different techniques, we add as many new features as those originally in the data. For unary transformations, LFE predicts the most suitable transformation for each feature. For binary and higher arity transformations, LFE considers a random sample of all combinations of features, finds the paradigm for each combination and selects top-k useful ones. In the following section, we describe how LFE learns and predicts useful transformations for features.

4 Transformation Recommendation

LFE models the problem of predicting a useful r-ary transformation $T_c \in \mathcal{T}_r$, ($\mathcal{T}_r$ is the set of r-ary transformations in $\mathcal{T}$), for a given list of features $[f_1, \ldots, f_r]$ as a multi-class classification problem, where the input is a representation of features, $R_{[f_1,\ldots,f_r]}$, and output classes are transformations in $\mathcal{T}_r$. LFE takes a one-vs-rest approach [Rifkin and Klautau, 2004]. Each transformation is modelled as a Multi-Layer Perceptron (MLP) binary classifier with real-valued confidence scores as output. Recommending an r-ary transformation for r features involves applying all $|\mathcal{T}_r|$ MLPs on $R_{[f_1,\ldots,f_r]}$. If the highest confidence score obtained from classifiers is above a given threshold, LFE recommends the corresponding transformation to be applied on feature f. Let $g_k(R_{[f_1,\ldots,f_r]})$ be the confidence score of the MLP corresponding to transformation T_k, and γ is the threshold for confidence scores which we determined empirically. LFE recommends the transformation T_c, for features $[f_1, \ldots, f_r]$, as follows:

$$c = \arg\max_k g_k(R_{[f_1,\ldots,f_r]})$$

$$\text{recommend} : \begin{cases} T_c, & \text{if } g_c(R_{[f_1,\ldots,f_r]}) > \gamma \\ \text{none}, & \text{otherwise} \end{cases} \qquad (1)$$

In the following sections, we describe how numerical features are represented to be the input of transformation MLPs. Next, we explain the process of collecting past feature engineering knowledge as training samples to train MLPs.

4.1 Feature-Class Representation

Transformations are used to reveal and improve significant correlation or discriminative information between features and class labels. The more pronounced this correlation, the higher the chance that a model can achieve a better predictive performance. Each LFE classifier learns the patterns of feature-class distributions for which the corresponding transformation has been effective in improving feature-class correlation. LFE represents feature f in a dataset with k classes as follows:

$$R_f = \left[Q_f^{(1)}; Q_f^{(2)}; \ldots; Q_f^{(k)} \right] \qquad (2)$$

where $Q_f^{(i)}$ is a fixed-sized representation of values in f that are associated with class i. We call this representation *Quantile Sketch Array*. Next, we describe how feature values associated to class i are translated into representation $Q_f^{(i)}$, which is meant to capture the distribution of feature values.

Neural networks have been successful in learning representations for image and speech data [Bengio *et al.*, 2013]. Others have proposed solutions for estimating a Probability Distribution Function (PDF) in an n-dimensional space [Garrido and Juste, 1998]. However, it is not clear how existing representation learning and PDF learning approaches may be applied in the context of raw numerical data. The main challenges is the high variability in the size and the range of feature values (e.g., from 10 to millions). In our setting, features are data points represented with various numbers of dimensions (number of distinct feature values). Hence, Random Projection [Bingham and Mannila, 2001] for dimensionality reduction is not applicable. Although Recurrent Neural Networks can deal with varying input size, we aim at determining a fixed-size representation that captures the correlation between features and target classes.

Previous approaches have used hand-crafted meta-features, including information-theoretic and statistical meta-features, to represent datasets [Michie *et al.*, 1994; Kalousis, 2002; Feurer *et al.*, 2015]. Such meta-features aim at modelling the distribution of values in datasets. Performing fixed-size sampling of feature values is another approach for representing distributions. Samples extracted from features and classes are required to reflect the distribution of values in both feature and target classes. While stratified sampling solves the issue for one feature, it becomes more complex for multiple features with high correlation [Skinner *et al.*, 1994]. Feature hashing has been used to represent features of type “string” as feature vectors [Weinberger *et al.*, 2009]. Although feature hashing can be generalized for numerical values, it is not straightforward to choose *distance-preserving* hash functions that map values within a small range to the same hash value. In Section 5, we empirically show how LFE transformation classifiers perform using each of representations described above.

Quantile Sketch Array (QSA) uses *quantile* data sketch [Wang *et al.*, 2013] to represent feature values associated with a class label. QSA is a non-parametric representation that enables characterizing the approximate Probability Distribution Function of values. QSA is closely related to the familiar concept of histogram, where data is summarized into a small number of buckets. Naumann et al. used quantile representation for numerical features to perform feature classification [Naumann *et al.*, 2002]. We apply the exact brute-force approach to compute quantile sketch for numerical values [Wang *et al.*, 2013], described as follows.

Let $\mathcal{V}_k$ be the bag of values in feature f that are for the training data points with the label c_k and $Q_f^{(i)}$ is the quantile sketch of $\mathcal{V}_k$. First, we scale these values to a predefined range $[lb, ub]$. Generating $Q_f^{(i)}$ involves bucketing all values in $\mathcal{V}_k$ into a set of bins. Given a fixed number of bins, r, the range $[lb, ub]$ is partitioned into r disjoint bins of uniform width $w = \frac{ub-lb}{r}$. Assume, the range $[lb, ub]$ is partitioned into bins $\{b_0, \ldots, b_{r-1}\}$, where the bin b_j is a range $[lb + j * w, lb + (j+1) * w)$. Function $B(v_l) = b_j$ associates the value v_l in $\mathcal{V}_k$ to the bin b_j. Function $P(b_j)$ returns the number of feature values, that are bucketed in b_j. Finally, $I(b_j) = \frac{P(b_j)}{\sum_{0 \le m < r} P(b_m)}$ is the normalized value of $P(b_j)$ across all bins.

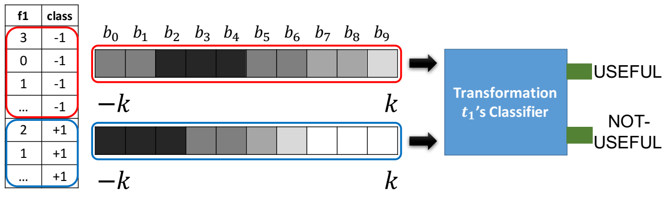
f1
class
3 -1
0 -1
1 -1
... -1
2 +1
1 +1
... +1
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9
$-k$
k
Transformation t_1's Classifier
USEFUL
NOT-USEFUL

Figure 1: An example of feature representation using Quantile Sketch Array. The feature $f1$'s values are binned into 10 equi-width bins, separately for classes -1 and $+1$.

To illustrate, Figure 1 shows an example of the representation of a feature. Note that this representation has certain behaviours. For example, for feature values that are perfectly separated in terms of the class label the representation ends up being very similar regardless of the scale of feature values. In this particular case there is also no need for any transformation since values are already separated. We decide QSA parameters (number of bins and scaling range) based on the performance of classifiers through empirical observations.

4.2 Training

To generate training samples for transformation MLP classifiers, LFE considers numerical features in classification datasets across various repositories. Each classifier is trained with the samples for which the corresponding transformation has been found useful as positive samples and all other samples as negative. In order to decide whether a transformation for a set of features leads to improvement, we evaluate a selected model, $\mathcal{L}$, on the original features and the target class as well as the constructed feature by itself and the target class. If the constructed feature shows performance improvement beyond a threshold, θ, the input features together with their class labels are considered as a *positive* training sample for the transformation classifier.

A sample generated from feature f, in a dataset with k classes, for transformation t is translated into $R_f = [Q_f^{(1)}; \dots ; Q_f^{(k)}]$. Assuming b bins are used for each quantile data sketch, R_f is a vector of size $k \times b$. Then, R_f is fed into the MLP corresponding to t. Assume the corresponding MLP to the unary transformation t has one hidden layer with h units. The probability of t being a useful transformation or not for feature f in a dataset is computed as:

$$[p_{t \text{ is useful}}(f), p_{t \text{ is not useful}}(f)] = \sigma_2(\mathbf{b^{(2)}} + \mathbf{W^{(2)}}(\sigma_1(\mathbf{b^{(1)}} + \mathbf{W^{(1)}}[Q_f^{(1)}; \dots ; Q_f^{(k)}]))) \tag{3}$$

where, $\mathbf{W^{(1)}}$ and $\mathbf{W^{(2)}}$ are weight matrices, $\mathbf{b^{(1)}}$ and $\mathbf{b^{(1)}}$ are bias vectors, and σ_1 and σ_2 are softmaxe and rectified linear unit (ReLU) functions, respectively [Nair and Hinton, 2010]. We use Stochastic Gradient Descent with minibatches to train transformation MLPs. In order to prevent overfitting, we apply regularization and drop-out [Srivastava *et al.*, 2014]. The generated training samples are dependent on model type. In other words, while there may be overlaps in the suggested feature engineering paradigms across models, the optimal use of LFE for a specific model comes from using that same model while training LFE. In Section 5, we show that LFE is robust in terms of the choice of classification model.

5 Experimental Results

We evaluate three aspects of *LFE*: (1) the impact of using Quantile Sketch Array representation on the performance of transformation classifiers compared to other representations, (2) the capability of LFE in recommending useful transformations, (3) the benefit of using LFE to perform feature engineering compared to other alternatives, in prediction accuracy and time taken. We implemented *LFE* transformation classifiers and the meta-feature learner (auto-encoder) in TensorFlow. Transformation computation and model training were implemented using Scikit-learn. To showcase the capabilities of LFE, we considered the following ten unary and four binary transformations, respectively: *log*, *square-root* (both applied on the *absolute* of values), *frequency* (count of how often a value occurs), *square*, *round*, *tanh*, *sigmoid*, *isotonic regression*, *zscore*, *normalization* (mapping to [-1, 1]), *sum*, *subtraction*, *multiplication* and *division*. In order to avoid data leakage, we apply transformations on train and test folds separately.

Without the loss of generality, we focus on binary classification for the purpose of these experiments. We collected 900 classification datasets from the OpenML and UCI repositories to train transformation classifiers. A subset of 50 datasets were reserved for testing and were not used for the purpose of training. In order to better utilize the datasets, we converted the multi-class problems into one-vs-all binary classification problems. Training samples were generated for Random Forest and Logistic Regression using 10-fold cross validation and the performance improvement threshold, θ, of 1%. Different improvement thresholds result in collecting variable number of training samples. Since the distribution of collected samples for transformations is not uniform, we decided on the threshold upon due empirical exploration. We generated approximately 84.5K training samples for unary transformation and 122K for binary transformations. Table 1 reports the statistics of positive training samples generated from all datasets. *Frequency* and *multiplication* are two transformations that incur performance improvement for the majority of features. All transformation classifiers are MLPs with one hidden layer. We tuned the number of hidden units to optimize the F-score for each classifier, and they vary from 400 to 500. The average time to train transformation classifiers offline was 6 hours.

5.1 Feature Representation

We evaluate the efficacy of Quantile Sketch Array (QSA) through the performance of LFE consisting of QSA compared to other representations, as listed in Table 2. These representations are listed in Table 2.

For *hand-crafted meta-feature* representation, we consider the following meta-features used in the literature: first 30 moments of feature values, median, standard deviation, min, max, number of values and its log. For *sampling* representation, we obtain 250 stratified random samples per class. We also perform *meta-feature learning* by training an auto-encoder on all features in our corpus. The input and output of the auto-encoder are the feature hashing representation [Weinberger *et al.*, 2009] of features with 200 hash values per class. The auto-encoder consists of a two-layer en-

Transformation	log	sqrt	square	freq	round	tanh	sigmoid	isotonic-reg	zscore	normalize	sum	subt	mult	div
#Positive Training Samples	6710	6293	5488	73919	15855	70829	70529	624	664	31019	28492	36296	37086	19020
Classifier Performance	0.95	0.96	0.97	0.77	0.92	0.92	0.80	0.98	0.97	0.91	0.98	0.99	0.97	0.98

Table 1: Statistics of Training Samples and F1 Score of LFE Classifiers for 10-fold Cross Validation of Random Forest.

Hand-crafted Meta-features	Stratified Sampling	Meta-feature Learning	Quantile Sketch Array
0.5558	0.5173	0.3256	**0.9129**

Table 2: F1 Score of Transformation Classifiers.

coder and a two-layer decoder, which are symmetric and each has a layer of 100 units connected to another layer of 75 units. For a hashed input feature, we consider the output of the encoder component of a trained auto-encoder as *learned* meta-features of the feature. Finally, for Quantile Sketch Array, we consider scaling range of [-10, 10] and quantile data sketch size of 200 bins.

We use the same set of samples for training classifiers of different representations and tuned the MLPs to have their best possible configuration. In order to overcome the imbalance data classification problem, as shown in Table 1, we oversample the minority class samples to have balanced negative and positive samples during the training phase. The average 10-fold cross validation F1-score of a subset of classifiers is reported in Table 2 as a proxy of the benefit of representations. There is a clear distinction in terms of predictive performance and employed representation. The poorest results are obtained by learned meta-features, possibly since feature values belong to various ranges and hashing representation is not distance preserving, therefore, it is challenging for the auto-encoder to learn useful meta-features. Hand-crafted meta-features perform better than sampling and learned meta-features, however, Quantile Sketch Array outperforms other representations by a significant margin (35.7%).

5.2 Transformation Classifier

To evaluate the predictive power of each classifier, we trained classifiers corresponding to unary and binary transformations, using 10-fold cross validation on all training samples. During the test phase, the corresponding transformation of each classifier is applied on one or a pair of feature(s) and the F1 score of the model based on the transformed feature only is evaluated. The F1-Score of classifiers using Random Forest and improvement threshold of 1%, are shown in Table 1. We considered 0/1 loss evaluation. Table 1 demonstrates the high predictive capability of LFE transformation classifiers. As baseline, a random transformation recommender converged to F1 score of 0.50 for all transformation classifiers.

5.3 Transformation Recommender

To showcase the benefits of LFE in recommending useful features, we compare the predictive performance of test datasets augmented with features engineered by *LFE* and features engineered by the following approaches.

- **Random**, which iterates for a given r runs and in each run recommends a random or no transformation for feature(s). The performance of the dataset augmented with features constructed by randomly selected transformations is evaluated in each run and the transformations in the run with the highest predictive performance are recommended.
- **Majority**, which always recommends the single most effective transformation in the training samples (i.e. frequency for unary and multiplication for binary transformations).
- **Brute-force**, inspired by Feature Synthesis component of Data Science Machine [Kanter and Veeramachaneni, 2015], enumerates the whole feature space by applying all transformations to all features and performs feature selection on the augmented dataset.
- **Model evaluation-based**, which chooses the useful transformation for a feature by model evaluation on each constructed feature by each transformation. Having t unary transformations, this approach performs t model training runs for each feature.

To compare LFE with other feature engineering approaches, we consider 50 binary classification datasets. The detailed statistics of 23 of test datasets are shown in Table 3. Test datasets have a diverse range in number of features (2 to 10,936) and data points (57 to 140,707). All experimental evaluations are based on 10-fold cross validation on a Random Forest model. Since Quantile Sketch Array considers both feature values and class labels, in order to avoid *data leakage* problem, we do not consider the entire dataset when *LFE* computes the recommendations. For experiments with unary transformations for each fold, we ask LFE to recommend at most one transformation for each feature in the train data fold. Next, recommended transformations are applied on the corresponding features of the test data fold and added as new features to the dataset. To only analyze the capability of feature engineering approaches, we do not perform any feature selection, except for brute-force approach to which feature selection is essential.

Table 3 compares the predictive performance and execution time of *LFE* to other feature engineering approaches on test datasets. All columns in this table, except the last, report results when using unary transformations. Since 50% of datasets time out for evaluation-based approach and 62% for brute-force approach, we only report the performance of LFE for binary transformations. All reported times in Table 3 include preparing data, recommending transformations, applying transformations and model training. Since the cost of training classifiers is one-time and offline, we do not consider it while comparing the runtime performance. *LFE* consistently outperforms all approaches on most datasets at a small cost of execution time. We argue that it is effective to spend certain effort offline in order to provide prompt answers at runtime. No feature engineering approach results in improvement of the performance of three datasets *sonar* and *spambase* and *twitter-absolute*. The evaluation-based approach on *AP-omentum-lung*, *convex*, *gisette* and *higgs-boson* timed out due to the excessive number of model training calls. Since the

Dataset	#Numerical Features	#Data Points	Base Dataset	Majority	Brute-Force	Random (10 runs)	Evaluation based	unary LFE	binary LFE
AP-omentum-lung	10936	203	0.883	0.915	0.925	0.908	-	**0.929**	0.904
AP-omentum-ovary	10936	275	0.724	0.775	0.801	0.745	0.788	**0.811**	0.775
autos	48	4562	0.946	0.95	0.944	0.929	0.954	**0.96**	0.949
balance-scale	8	369	0.884	0.916	0.892	0.881	0.882	**0.919**	0.884
convex	784	50000	0.82	0.5	**0.913**	0.5	-	0.819	0.821
credit-a	6	690	0.753	0.647	0.521	0.643	0.748	**0.771**	0.771
dbworld-bodies	2	100	0.93	0.939	0.927	0.909	0.921	**0.961**	0.923
diabetes	8	768	0.745	0.694	0.737	0.719	0.731	**0.762**	0.749
fertility	9	100	0.854	0.872	0.861	0.832	0.833	**0.873**	0.861
gisette	5000	2100	0.941	0.601	0.741	0.855	-	**0.942**	0.933
hepatitis	6	155	0.747	0.736	0.753	0.727	**0.814**	0.807	0.831
higgs-boson-subset	28	50000	0.676	0.584	0.661	0.663	-	**0.68**	0.677
ionosphere	34	351	0.931	0.918	0.912	0.907	0.913	**0.932**	0.925
labor	8	57	0.856	0.827	0.855	0.806	0.862	**0.896**	0.896
lymph	10936	138	0.673	0.664	0.534	0.666	0.727	**0.757**	0.719
madelon	500	780	0.612	0.549	0.585	0.551	0.545	**0.617**	0.615
megawatt1	37	253	0.873	0.874	0.882	0.869	0.877	**0.894**	0.885
pima-indians-subset	8	768	0.74	0.687	**0.751**	0.726	0.735	0.745	0.76
secom	590	470	0.917	0.917	0.913	0.915	0.915	**0.918**	0.915
sonar	60	208	**0.808**	0.763	0.468	0.462	0.806	0.801	0.783
spambase	57	4601	**0.948**	0.737	0.39	0.413	0.948	0.947	0.947
spectf-heart	43	80	0.941	**0.955**	0.881	0.942	0.955	**0.955**	0.956
twitter-absolute	77	140707	**0.964**	0.866	0.946	0.958	0.963	0.964	0.964
Feature Engineering and Model Evaluation Time (seconds)		**geomean**	2.66	11.06	48.54	69.57	403.81	18.28	44.58
		average	19.23	1219.34	13723.51	2041.52	10508.75	97.90	188.17

Table 3: Statistics of Datasets and F1 Score of LFE and Other Feature Engineering Approaches with 10-fold Cross Validation of Random Forest. The best performing approach is shown in bold for each dataset. The improving approaches are underlined.

evaluation-based approach requires model training for each combination of feature and transformation, the *evaluation-based* approach is only applicable to datasets with small number of features and data points.

To broaden the scope of experiments, we performed feature engineering using the same approaches and setting of Table 3 on 50 test datasets. Figure 2 shows the percentage of test datasets whose predictive performance improved by each feature engineering approach. For Random Forest, we observed that for 24% of test datasets, none of the considered approaches, including LFE, can improve classification performance. In all remaining datasets, LFE always improves the predictive performance and for 89% of such datasets LFE generates features that results in the highest classification performance. For the remaining 11% of datasets, the higher performance is achieved by other approaches at higher computational cost than LFE. To investigate the robustness of LFE with respect to the model, we performed the experiments for Logistic Regression and similar results were observed. While LFE is a one shot approach it already achieves an average improvement of more than 2% in F1 score and a maximal improvement of 13% across the 50 test data sets.

6 Conclusion and Future Work

In this paper, we present a novel framework called LFE to perform automated feature engineering by learning patterns between feature characteristics, class distributions, and useful transformations, from historical data. The cornerstone of our framework is a novel feature representation, called Quantile Sketch Array (QSA), that reduces any variable sized features to a fixed size array, preserving its essential characteristics. QSA enables LFE to learn and predict the ability of a transform to improve the accuracy of a given dataset. Our empirical evaluation demonstrates the efficacy and efficiency of LFE in improving the predictive performance at low computational costs for a variety of classification problems. We plan to add more transformations, to design an iterative version of LFE, and combine it with exploration-based methods to achieve even more pronounced improvements. In addition, we aim to use the family of Recurrent Neural Networks to deal with both varying data set sizes and taking into account relationships across multiple features within a dataset to improve transformation recommendation.

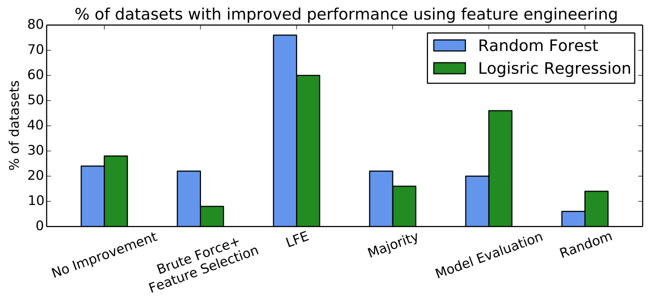

Figure 2: The percentage of datasets, from a sample of 50, for which a feature engineering approach results in performance improvement (measured by F1 score of 10 fold cross validation for Random Forest and Logistic Regression).

References

[Bengio *et al.*, 2013] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE TPAMI*, 35(8), 2013.

[Bingham and Mannila, 2001] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: Applications to image and text data. *KDD*, pages 245–250, 2001.

[Che *et al.*, 2015] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. Distilling knowledge from deep networks with applications to healthcare domain. *arXiv preprint arXiv:1512.03542*, 2015.

[Dor and Reich, 2012] Ofer Dor and Yoram Reich. Strengthening learning algorithms by feature discovery. *Information Sciences*, 2012.

[Fan *et al.*, 2010] Wei Fan, Erheng Zhong, Jing Peng, Olivier Verscheure, Kun Zhang, Jiangtao Ren, Rong Yan, and Qiang Yang. Generalized and heuristic-free feature construction for improved accuracy. *SDM*, 2010.

[Feurer *et al.*, 2015] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. *NIPS*, 2015.

[Garrido and Juste, 1998] Lluis Garrido and Aurelio Juste. On the determination of probability density functions by using neural networks. *Computer Physics Communications*, 115(1):25 – 31, 1998.

[Greenwald and Khanna, 2001] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *SIGMOD*, pages 58–66, 2001.

[Kalousis, 2002] Alexandros Kalousis. *Algorithm selection via meta-learning*. PhD thesis, Universite de Geneve, 2002.

[Kanter and Veeramachaneni, 2015] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. *DSAA*, 2015.

[Katz *et al.*, 2016] Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. *ICDM*, pages 979–984, 2016.

[Khurana *et al.*, 2016] Udayan Khurana, Deepak Turaga, Horst Samulowitz, and Srinivasan Parthasarathy. Cognito: Automated Feature Engineering for Supervised Learning. *ICDM*, 2016.

[Lichman, 2013] M. Lichman. UCI machine learning repository, 2013.

[Markovitch and Rosenstein, 2002] Shaul Markovitch and Dan Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 2002.

[Michie *et al.*, 1994] Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell, editors. *Machine Learning, Neural and Statistical Classification*. 1994.

[Nair and Hinton, 2010] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. *ICML*, pages 807–814, 2010.

[Naumann *et al.*, 2002] F. Naumann, C.-T. Ho, X. Tian, L. Haas, and N Megiddo. Attribute classification using feature analysis. *ICDE*, page 271, 2002.

[Rifkin and Klautau, 2004] Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *JMLR*, 5:101–141, 2004.

[Shawe-Taylor and Cristianini, 2004] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[Skinner *et al.*, 1994] C.J. Skinner, D.J. Holmes, and D. Holt. Multiple frame sampling for multivariate stratification. *International Statistical Review*, 62(3), 1994.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning*, 15(1):1929–1958, 2014.

[Storcheus *et al.*, 2015] Dmitry Storcheus, Afshin Rostamizadeh, and Sanjiv Kumar. A survey of modern questions and challenges in feature extraction. *Proceedings of The 1st International Workshop on Feature Extraction, NIPS*, 2015.

[Vanschoren *et al.*, 2014] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, June 2014.

[Wang *et al.*, 2013] Lu Wang, Ge Luo, Ke Yi, and Graham Cormode. Quantiles over data streams: An experimental study. *SIGMOD*, pages 737–748, 2013.

[Weinberger *et al.*, 2009] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. *ICML*, 2009.