Randomized Feature Engineering as a Fast and Accurate Alternative to Kernel Methods

Suhang Wang
Arizona State University
Tempe, AZ 85281
suhang.wang@asu.edu

Charu Aggarwal
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
charu@us.ibm.com

Huan Liu
Arizona State University
Tempe, AZ 85281
huan.liu@asu.edu

ABSTRACT

Feature engineering has found increasing interest in recent years because of its ability to improve the effectiveness of various machine learning models. Although tailored feature engineering methods have been designed for various domains, there are few that simulate the consistent effectiveness of kernel methods. At the core, the success of kernel methods is achieved by using similarity functions that emphasize local variations in similarity. Unfortunately, this ability comes at the price of the high level of computational resources required and the inflexibility of the representation as it only provides the similarity of two data points instead of vector representations of each data point; while the vector representations can be readily used as input to facilitate various models for different tasks. Furthermore, kernel methods are also highly susceptible to overfitting and noise and it cannot capture the variety of data locality. In this paper, we first analyze the inner working and weaknesses of kernel method, which serves as guidance for designing feature engineering. With the guidance, we explore the use of randomized methods for feature engineering by capturing multi-granular locality of data. This approach has the merit of being time and space efficient for feature construction. Furthermore, the approach is resistant to overfitting and noise because the randomized approach naturally enables fast and robust ensemble methods. Extensive experiments on a number of real world datasets are conducted to show the effectiveness of the approach for various tasks such as clustering, classification and outlier detection.

CCS CONCEPTS

•Information systems →Data mining;

KEYWORDS

Randomized Feature Engineering; Unsupervised Feature Learning

ACM Reference format:
Suhang Wang, Charu Aggarwal, and Huan Liu. 2017. Randomized Feature Engineering as a Fast and Accurate Alternative to Kernel Methods. In *Proceedings of KDD'17, August 13–17, 2017, Halifax, NS, Canada.*, , 10 pages.
DOI: http://dx.doi.org/10.1145/3097983.3098001

1 INTRODUCTION

Machine learning algorithms are highly sensitive to the specific feature representation of the data used [3]. Selecting the proper feature representation is important because a good set of features enables the use of fast and simple models, whereas overly compact representations often necessitate the use of complex models that are slow to train and semantically opaque. For example, a well-chosen set of features can often be used in conjunction with a linear support vector machine, which can be trained in time that is linear in the size of the input [14]. The importance of the specific representations has been pointed out quite forcefully [9]: "At the end of the day, some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used." *Therefore, our first goal of feature engineering is to design discriminative engineered features which can enable the use of fast and simple models.*

Kernel methods [1, 2, 11, 22, 30, 32] are among the most popularly used feature transformation methods, which have been successfully incorporated in different algorithms for better performance such as kernel SVMs for classification [20], kernel PCA for dimensionality reduction [17] and kernel K-means for clustering [7]. The essential idea of kernel methods is to transform a non-linearly separable data to a discriminative and linearly separable feature space, in which the discriminative nature of the feature space is primarily ensured by tuning the parameters of the kernel.

Kernel methods use an implicit transformation by working with the similarity of the data points in the input space, but not the explicit representation in the transformed feature space. Explicit features can be readily used as input for off-the-shelf methods, such as linear SVMs for classification and k-means for clustering. On the other hand, customized kernel algorithms need to be developed for different tasks, such as kernel SVMs for classification and kernel K-means for clustering. *Thus, the second goal of feature engineering is to construct/learn explicit features* to enable wider applicability across many problems.

Kernel algorithms could also be interpreted as a two-step approach of first learning explicit features and then applying simple models on the features. However, such an explicit approach is not practical. Consider a data set $\mathbf{X} \in \mathbb{R}^{n \times d}$ containing n data points and d dimensions. Let $\mathrm{S} \in \mathbb{R}^{n \times n}$ be the kernel similarity matrix with the (i, j)-th entry equal to the kernel similarity between the ith and jth data points, i.e., $\mathrm{S}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, where $k(\cdot, \cdot)$ is the kernel function such as RBF. Then, any kernel SVM can be implemented using the following two-step process:

- Diagonalizing the positive semi-definite similarity matrix S as $\mathrm{S} = \mathbf{Q}\Sigma^2\mathbf{Q}^T$, where $\mathbf{Q}$ is an $n \times n$ matrix with orthonormal columns and Σ^2 contains the non-negative eigenvalues.

KDD'17, August 13–17, 2017, Halifax, NS, Canada.
© 2017 ACM. ISBN 978-1-4503-4887-4/17/08…$15.00
DOI: http://dx.doi.org/10.1145/3097983.3098001

Then, an n-dimensional embedding of the data is given by $\mathbf{Q}\Sigma$; and
- Applying a linear SVM on training data with features as $\mathbf{Q}\Sigma$, and class labels in the n-dimensional vector $\mathbf{y}$.

The dual problem of linear SVMs is to maximize $\sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j y_i y_j \mathbf{q}_i \Sigma(\mathbf{q}_j\Sigma)^T$, where a_i are the Lagrange multipliers and $\mathbf{q}_i$ is the i-th row of $\mathbf{Q}$. It is evident that $\mathbf{q}_i\Sigma(\mathbf{q}_j\Sigma)^T = \mathbf{q}_i\Sigma^2\mathbf{q}_j^T = k(\mathbf{x}_i, \mathbf{x}_j)$, and thus the above two-step process is equivalent to kernel SVMs. Furthermore, one can also make a similar argument about the kernel k-means algorithm [7, 8].

The transformed features $\mathbf{Q}\Sigma$ in the two-step approach are explicit features, which is also referred to as the *data-specific Mercer kernel map*. The explicit features $\mathbf{Q}\Sigma$ provide several advantages, such as the ability to manipulate explicit features, drop irrelevant features from $\mathbf{Q}\Sigma$, provide better interpretability, or be used as input to different models for different tasks. These flexibilities are not available when using the kernel trick. However, calculating $\mathbf{Q}\Sigma$ is rarely used in kernel SVMs or kernel PCAs because of (i) the large dimensionality of the matrix $\mathbf{Q}\Sigma$, which is potentially as large as the number of training points n; and (2) the large computational cost, which is of $O(n^3 + n^2 d)$. The fact that the n-dimensional transformation is so cumbersome and space-inefficient limits its practical usage. Furthermore, kernel transformations have several weaknesses from a *modeling* point of view, such as its sensitivity to the multi-locality of the data and noise [23], which will be discussed in detail in Section 2. *Thus, our third goal is to design a feature engineering method that is space and time efficient, and robust to noise.*

Therefore, in this paper, we investigate the problem of designing explicit robust engineered features with lower space and time requirements, which can capture multi-granular locality of the data and thus facilitate classification and clustering with simple models such as linear SVM and K-means. The main contributions of the paper are as follows:

- We investigate the inner working and weaknesses of kernel methods, which inspires a principled way to design engineered features;
- We propose a novel feature engineering method with low space and time requirement called RandLocal, which learns robust features that capture multi-granular locality and is efficient for various tasks with simple models; and
- We conduct extensive experiments to demonstrate the effectiveness of the proposed framework for tasks such as clustering, classification and outlier detection.

The rest of the paper are organized as follows. In Section 2, we investigate inner working and weaknesses of kernel methods. In Section 3, we introduce the proposed framework RandLocal with time/space complexity and connection to nearest neighbor based methods. In Section 5, we conduct experiments to show effectiveness of RandLocal. In Section 6, we conclude with future work.

2 THE INNER WORKING OF A KERNEL TRANSFORMATION AND ITS WEAKNESSES

In this section, we introduce the working and weaknesses of a kernel transformation, which serves as a guidance for designing RandLocal.

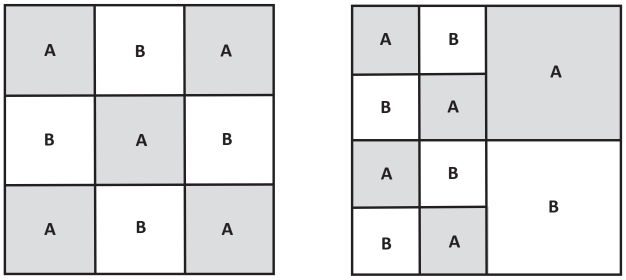
2.1 Inner Working of Kernel Transformations

At its core, a kernel method uses a similarity function between two points that is more sensitive to data locality than the dot product. For example, when one uses a Gaussian kernel with small bandwidth h (relative to average inter-point distance), i.e., $k(\mathbf{x}_i, \mathbf{x}_j = \exp^{-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/h}$, the similarities between many pairs of points are nearly 0 because of the exponential drop-off in similarity with inter-point distance. However, this decay is relatively slow for distances that are much smaller than the bandwidth. It is not the case with the dot-product, which is linearly related to the squared distance between points. Therefore, in high-dimensional embeddings, *pairwise kernel similarity is often extremely sensitive within small data localities of the original representation (depending on the parameters of the kernel such as bandwidth), whereas it is extremely insensitive outside that small locality.*

For example, consider the 9 clusters which belongs two classes illustrated in Figure 1(a). In this case, it is evident that the two classes are perfectly separable using a nonlinear boundary, but they cannot be separated with a linear boundary. This is because the class distributions are localized into small clusters that cannot be linearly separated from one another. However, if we used a Gaussian kernel with bandwidth of about half the radius of each cluster, the resulting embedding, i.e., $\mathbf{Q}\Sigma$, would be such that the dot products across clusters would be nearly zero, except at the boundary regions. This can be approximately achieved in a non-negative embedding in which the top-9 components of this representation will be dominated by points from the nine different clusters. In other words, points from each cluster will have a significantly positive component only along one of these nine features and will have zero values across the other eight features. However, to achieve this goal, one has to increase the dimensionality of the original data from two features to nine features in sparse format. This observation is important and therefore we emphasize it below:

> *Increasing the similarity sensitivity to data locality in a kernel function often creates new features that describe the behavior of small data localities. One can simulate this type of pairwise similarity with feature engineering by defining features specific to small data localities. The feature values are set to 0 (or roughly constant values) if the data point lies outside that locality.*

Parameters such as the kernel bandwidth implicitly control the size of this locality in kernel transformations. The aforementioned feature engineering creates a sparse data representation that is sensitive to data locality. Co-incidentally, this type of *sparse* transformation also makes the data linearly separable by a support-vector machine because the linear SVM is now able to put together the small local pieces represented by each feature on the appropriate side of the linear separator (irrespective of the shape of the original



(a) Classes A and B are not linearly separable

(b) Different localities of data are prone to overfitting/underfitting

Figure 1: Examples of different localities of data points

decision boundary). For example, if we create a 3 × 3 grid of regions in Figure 1(a), such that each region of the grid contains one of the clusters, then we can create nine binary features depending on the presence or absence of a point in each cluster. It can be easily shown that this binary representation is linearly separable. Although this particular example is a little too clean as compared to real settings, it nevertheless sets up the intuition we need for the feature engineering step.

2.2 Weaknesses of Kernel Transformations

With knowing the working of kernel transformations, we will now discuss what they lack. Some key weaknesses are *in addition to* the space- and time-inefficiency of kernel methods.

2.2.1 Inflexibility in Locality of Representation. The most important problem with kernel representations is that one cannot control the granularity of the resulting data locality because it is "baked into the cake" once the parameters of the kernel (e.g., Gaussian bandwidth) have been chosen. In many cases, different bandwidths are more appropriate for different data localities, and kernel functions cannot fully express these factors. An example is illustrated in Figure 1(b), in which a smaller kernel bandwidth is appropriate for the data points on the left, and a larger kernel bandwidth is appropriate for data points towards the right. Choosing a small kernel bandwidth will cause overfitting in some parts of the data, whereas choosing a large kernel bandwidth will cause underfitting in other parts of the data. Therefore, one of our goals will be to design a sparse representation of the data with randomized feature engineering, which is sensitive to data-locality in a multi-granular way. As we will see later, this type of flexibility is one of the major advantages that explicit feature engineering methods have over kernel transformations, because the learning algorithm can now pick and choose the *relevant* granularity of feature representation in each data locality, rather than an opaque use of all transformed features with the kernel method. Although some recent methods partially address this problem [5], they are highly prone to overfitting, which is common with kernel methods.

2.2.2 Irrelevant Input Features and the Curse of Dimensionality. Kernel methods are not immune to the curse of dimensionality. Even though the margin-based approach of an SVM can reduce the impact of irrelevant features in the *transformed space*, the transformation itself includes the impact of irrelevant features in the *input space*. Often small subsets of (input) features may be relevant for various data localities in which the remaining features may not add much information. In fact, it has been shown [4] that the Gaussian kernel, which uses the squared distance in its exponent, is susceptible to poor results with an increasing number of irrelevant (input) features. This is because the Gaussian kernel can be expanded into interactions of various orders between input features. It has been shown that in many data sets, only the lower-order interactions between features are relevant [10], and the use of all features adds noise. For example, in Figure 1(a), if 10 noisy features are added to the data set, a Gaussian kernel would work very poorly because of the impact of noisy features.

2.2.3 Locally Relevant Features. This problem of irrelevant input features is particularly significant when the relevant features vary with data locality. Even though many input features may be relevant on a global basis, only small subsets of input features are often relevant when examining various data localities. For example, rule-based classifiers are often able to provide high-quality classification, even when the antecedents are allowed to contain a maximum of two or three features. This is primarily because rule-based classifiers typically model small localities of the data in their antecedents, in which small subsets of dimensions are effective. A kernel methods such as kernel SVM usually uses all the dimensions and their higher-order interactions while computing similarities and thus cannot capture the multi-level data locality. For sparse representations, even the modeling of second-order interactions is sufficient, as is evident from the recent success of second-order factorization machines [19]. Correspondingly, we will construct a feature representation that is able to model such lower-order interactions among input features, *in addition to* the global patterns.

2.2.4 Opaque Trade-Off between Overfitting and Underfitting. The lack of interpretability of the kernel features makes the trade-off between overfitting and underfitting (with the use of parameters such as the kernel bandwidth). As mentioned earlier, such representations are often inappropriately global in using a single parameter to control the sensitivity of data locality to parameters such as the bandwidth. On the other hand, we will show that randomized feature engineering is able to extract a large number of features of multiple levels of granularity. Because of the multi-granular representation, under-fitting is always avoided in each data locality. Furthermore, it naturally enables ensemble-centric approach to learning, which also avoids overfitting.

2.2.5 Efficiency Issues. Kernel methods are extremely slow because they work with an $n \times n$ matrix of similarities between pairs of points. Therefore, the complexity of a kernel method is somewhere between $O(n^2 d)$ and $O(n^3)$. Therefore, it is useful to examine randomized feature engineering methods that can be extremely efficient in comparison. In fact, we will propose a method whose complexity is linear in to number of data points.

2.2.6 Interpretability. The features produced by a kernel method are often not very interpretable. This precludes their use to make diagnostic inferences about specific predictions.

Given the inner working and weaknesses of kernel transformation, in next section, we will propose RandLocal, a more flexible approach of feature engineering which incorporate these characteristics.

3 RANDOMIZED FEATURE ENGINEERING TO CREATE LOCAL FEATURES

As discussed in section 2, feature engineering methods can simulate greater sensitivity in pairwise similarity in small localities by creating features that correspond to local regions defined by *anchor points* sampled from the data set. The basic idea is to create a nonnegative representation of the data, in which each features are associated with anchor points. A data point takes on a strictly positive value for a particular feature only if it is a nearest neighbor for that particular anchor point. Otherwise it takes on a value of 0 for that feature. Therefore, only one feature is activated by a given data point for a particular group of anchors. Each set of anchor points induces a Voronoi partition of the data space, and the granularity of this Voronoi partition depends on the number of anchor points. When a linear classifier is applied to this representation, it implicitly uses the class distribution of each of the local regions to define the relevance of that (engineered) feature to the corresponding class. A larger number of anchor points helps in defining more detailed decision boundaries but runs a risk of greater overfitting.

Multiple features are made to take on non-zero values by repeatedly sampling different groups of anchor points. The resulting representation is sparse and nonnegative with a high degree of local information captured by the features. The degree of locality (i.e., granularity) is captured by the number of anchor points. Since, the appropriate granularity cannot be known in advance (and may, in fact, vary with data locality), we propose to vary the number of anchor points in the range $[32, 1024]$ using a geometric distribution. The overall process of creating each set of features (from a particular anchor group) is summarized in Algorithm 1.

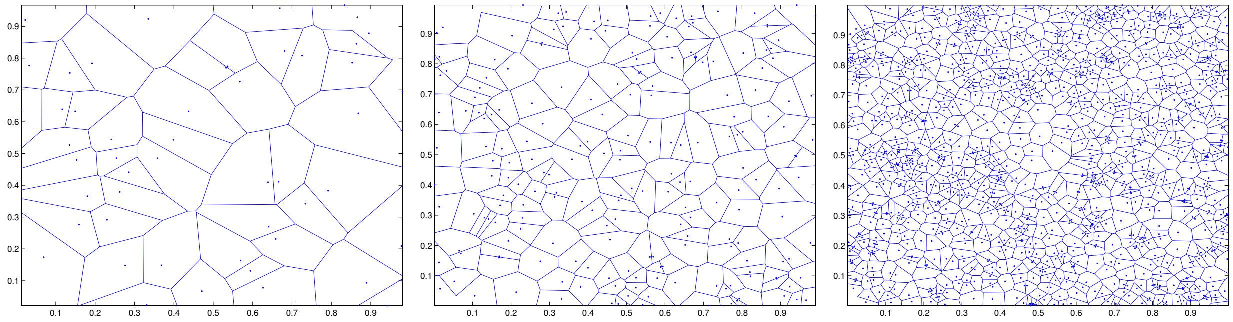
Algorithm 1 RandFea

```
Input: Dataset: $\mathbf{X} \in \mathbb{R}^{N \times d}$, $\epsilon$
Output: Engineered features $\mathbf{F}$
Sample the real number $p$ uniformly at random between $\log_2(32) = 5$, and $\log_2(1024) = 10$.
Set the number of anchors $r$ to $\lfloor 2^p \rfloor$, and sample $r$ anchors
Calculate the distance of $N$ points to $r$ anchors
Let the average of the $r \times N$ anchor-point distances be $D_{avg}$.
Initialize a sparse all zero matrix $\mathbf{F} \in \mathbb{R}^{N \times r}$.
for each data point $\mathbf{x}_i$ do
    Let the minimum distance of $\mathbf{x}_i$ to the $r$ anchors be $D_{min}(\mathbf{x}_i)$, and the nearest anchor be $k$. If the nearest anchor is $\mathbf{x}_i$ itself, use second nearest anchor.
    Set $\mathbf{F}_{ik} = \max\{D_{avg} - D_{min}(\mathbf{x}_i), \epsilon \cdot D_{avg}\}$
end for
return the engineered feature $\mathbf{F}$
```

In Algorithm 1, ϵ is a small number such as 0.0001, and is meant to add regularization and smoothing to the process against the presence of outliers. Euclidean distance is used to calculate the anchor-point distance while other distance metrics can also be adopted. Any data point $\mathbf{x}_i$ that is also an anchor is always assigned to its nearest other anchor (in order to avoid overfitting) and therefore the value of $D_{min}(\mathbf{x}_i)$ is not necessarily 0. Note that this description implicitly assumes that the data set contains significantly more than 1024 points because the number of anchors needs to be smaller than the number of points by a factor of at least 2 or 3. To ensure that the number of anchors is significantly less than the number of points, we can set the upper limit of sampling the real number p to $\min\{\log_2(N/2), \log_2(1024)\}$. The value $\mathbf{F}_{ik} = \max\{D_{avg} - D_{min}(\mathbf{x}_i), \epsilon \cdot D_{avg}\}$ captures the locality information, i.e., its nearest neighbor is k-th anchor and the distance to it is $D_{avg} - D_{min}(\mathbf{x}_i)$. Thus, the resulting feature $\mathbf{F}$ is informative. The reason we use only the nearest anchor is: (i) to make $\mathbf{F}$ sparse which can reduce space requirement and advance classification or clustering models such as SVM and K-means; and (2) the nearest anchor is more informative.

The anchors essentially divide the data space into a set of *Voronoi regions*, which also correspond to the features. Data points lying within these Voronoi regions activate these features. When a linear SVM is applied to this feature representation, it "pieces together" these Voronoi regions in order to create a decision boundary. The Voronoi regions with the use of 50, 200, and 1000 anchor points are illustrated in Figures 2(a), (b), and (c), respectively. In the figure, each dot represents an anchor point. The line surrounding an anchor point forms a Voronoi region for the anchor point. Any data point within the Voronoi region of an anchor point will be closer to it than other anchor points. For example, if we superimposed these Voronoi regions, with the class distributions shown in Figures 1(a) or 1(b), the only errors would be caused by edge effects of the irregular Voronoi boundaries. As a result, the decision boundary would be somewhat jagged, but would still approximate the true decision boundary well. Clearly, a larger number of anchors provides decision boundaries that conform more closely with the training data, but the trade-off is that it could overfit to the specific vagaries of a particular training data set. This is the reason that we vary the number of anchors in order to provide multigranular analysis. Note that using multiple sets of anchors will always result in a smoother decision boundary because of its implicit ensemble-centric approach in hiding the complexity within the feature representation, but using a low-variance linear model at the very end. This type of feature representation can use the features with the appropriate relevance and granularity from each data locality. For example, if a particular combination of low-dimensional features is relevant in a particular data locality, the local features in the anchor points using the closest possible combination of features will be automatically identified by a supervised learning algorithm. Of course, since sampling is used, the precise combination of features may not be available; however, in many cases, a close enough combination of locally engineered features may be available for learning.

The approach can also address the problem in a kernel SVM that it uses a global bandwidth (and corresponding granularity of analysis) in cases different bandwidths are suitable to different data localities. For example, in the case of Figure 1(b), a linear SVM should use coarse features to classify data points in the right-hand side of Figure 1(b), whereas it should use fine-grained features to classify data points on the left-hand side. This is possible in the case of our



(a) 50 anchors

(b) 200 anchors

(c) 1000 anchors

Figure 2: The anchors define the level of granularity covered by the new feature set. The number of anchors defines the trade-off between bias and variance of a learning method. A smaller number of anchors corresponds to larger bandwidths of the Gaussian kernel, whereas a larger number of anchors corresponds to smaller bandwidths of the Gaussian kernel.

approach because the features are associated with anchors with varying levels of granularity. In fact, an L_1-regularization can even be used to select the appropriate features for each data locality, since the coefficients of irrelevant features (in the transformed space) would be set to 0. Note that this type of locally adaptive learning is not possible for a kernel learning method that typically uses a single value of the bandwidth to learn the features.

It is also noteworthy that the local regions created by the features are highly interpretable. For example, if a particular feature has a weight of a particular sign in the coefficient vector of a linear SVM (with labels +1 and −1), the sign of the coefficient defines the class affinity of the data locality of that specific anchor. Similarly, if a k-means algorithm is applied to the engineered feature representation for clustering, it will piece together the dense regions of arbitrary shape in the input space by using the local regions defined by the various features. The dominant features in the centroid of a cluster define the regions relevant to that cluster as well.

3.1 Addressing Locally Irrelevant Features in the Input Space

Most kernel methods perform poorly when many features in the *input space* are globally or locally irrelevant. This is because the kernel method uses all (input) features within the kernel similarity function irrespective to their relevance [4, 10]. The explicit methodology of feature engineering provides a natural way to sample the input space for features, while constructing the overall feature representation by repeated anchor sampling. While sampling each set of anchor points, an additional step is added up front to sample subsets of features. Then, the overall process of feature engineering (which we refer to as *RandLocal*) are summarized in Algorithm 2.

In Algorithm 2, the number of sets of anchors T is an input parameter to the algorithm. **parfor** means *parallel for loop*. The approach starts with a null set of features and progressively samples anchors to add sets of sparse features for T iterations. The value of T defines the dimensionality of the feature representation. Although the precise dimensionality depends on the vagaries of the sampling process, exactly T features will always be non-zero in each data point. Therefore, the representation is very sparse. The value of T is selected to be somewhere between 100 and 500. It is noteworthy that sparse representations like text are also almost always separable with linear SVMs and other low-variance models [14]. Although the subspace sampling process of *RandLocal* is quite safe in the supervised setting (because of the ability to detect the relevance of engineered features a posteriori), one needs to be careful of subspace sampling in the unsupervised setting. We discuss this issue below.

Algorithm 2 RandLocal

```
Input: Dataset: $\mathbf{X} \in \mathbb{R}^{n \times d}$, Iterations: T
Output: engineered feature $\mathbf{F}$
Initialize $\mathbf{F} = \emptyset$
parfor t=1 to T do
    Sample a number $m_t$ between 1 and $d$
    Create $\mathbf{X}_t \in \mathbb{R}^{n \times m_t}$ by sampling $m_t$ features from $\mathbf{X}$
    Create engineered features as $\mathbf{F}_t$ = RandFea($\mathbf{X}_t$)
    Concatenate $\mathbf{F}_t$ to $\mathbf{F}$
end parfor
return engineered feature $\mathbf{F}$
```

3.1.1 *Difference between Supervised and Unsupervised Settings.*

The aforementioned description of *RandLocal* is appropriate for the supervised setting. In supervised settings, low-dimensional combinations of features in the input space often have a high level of predictive power because of the ability to learn them from the *labeled* training data. However, in unsupervised problems, it is difficult to learn the relevance of engineered features, and randomly sampling subspaces often leads to dropping relevant (input) features that cannot be detected a posteriori. Therefore, in order to avoid this problem, one minor change is made to the algorithm *RandLocal* above to make it more appropriate for unsupervised problems. In particular, instead of sampling between 1 and d dimensions in the first step of *RandLocal*, we only sample between $\lceil d/2 \rceil$ and d dimensions. This reduces the likelihood of losing a very large proportion of the relevant input features. This type of subspace sampling is similar to the feature bagging approach [15] that is commonly used in unsupervised outlier detection (albeit in a different context). In the following, it is assumed that supervised applications of *RandLocal* always sample between 1 and d dimensions, whereas unsupervised applications of *RandLocal* sample between $\lceil d/2 \rceil$ and d dimensions.

3.2 Time and Space Complexity

Now we'll demonstrate that RandLocal is time and space efficient. **Time Complexity** Let's consider the t-th time we call RandLocal with $\mathbf{X}_t \in \mathbb{R}^{n \times m_t}$ as input. Assume that the number of anchors we sampled is r_t. Then The complexity of calculating the distance of the n data point to r_t anchor points is $O(nm_tr_t)$. For each data point $\mathbf{x}_i$, getting $D_{min}(\mathbf{x}_i)$ and setting the feature of $\mathbf{x}_i$ cost $O(nr_t)$. Thus, the total time complexity is $O(\sum_{t=1}^{T} nm_tr_t)$ where T is the number of times we perform RandFea. Note that r_t is a small number within 32 and 1024 and performing RandFea T times is embarrassingly parallel. Therefore, RandLocal is efficient for distributed or multi-thread computation.

Space Complexity Similarly, for each call of RandFea with $\mathbf{X}_t \in \mathbb{R}^{n \times m_t}$ as input, the $r_t \times n$ distance matrix takes $O(nr_t)$ space. Since $\mathbf{F}_t \in \mathbb{R}^{n \times r_t}$ is very sparse, i.e., only n nonzero entries, the space complexity of $\mathbf{F}_t$ is $O(n)$. Therefore, the total space complexity of RandLocal in Algorithm 2 is $O(\sum_{t=1}^{T} nr_t)$. The space complexity of the returned high-dimensional sparse engineered feature $\mathbf{F}$ is $O(nT)$. Therefore, RandLocal is also space efficient.

3.3 Connections with Weighted Nearest Neighbor Methods

In addition to the aforementioned discussion, it is possible to show another interesting connection between the feature engineering method discussed earlier and kernel methods. Since the engineered representation uses nearest neighbors in conjunction with anchor points, dot products between pairs of engineered representation can be shown to be heavily localized similarities. It is also well known that kernel SVMs are weighted nearest neighbor classifiers [31]. In other words, one can show that both kernel SVMs and linear SVMs on the engineered representation approximately behave like weighted nearest neighbor classifiers in the original input space. The only difference is that the engineered method has better diversity because of its ability to select different numbers of anchors in various subspaces of the input space, and it can also drop the irrelevant (engineered and input) features in various phases of the learning process. As a result the engineered method is more adaptive and less prone to overfitting.

4 EXPERIMENTS

In this section, we conduct experiments to show the effectiveness of the engineered features by RandLocal for different tasks. Specifically, we aim to answer the following three questions:

- With the sparse representation that captures multi-locality, can RandLocal improve classification performance with linear SVMs and clustering performance with K-means?
- How robust is RandLocal in constructing features from noisy datasets?
- Can RandLocal construct effective features for detecting outliers?

We begin by introducing the datasets and representative state-of-the-art feature engineering methods; we then compare RandLocal with these feature engineering methods on classification and clustering to answer the first question and we investigate the ability of RandLocal in handling noisy datasets to answer the second question. We also study the RandLocal for outlier detection to answer the third question. Finally, further experiments are conducted to investigate the sensitivity of RandLocal to the parameters.

Table 1: Statistics of Datasets for Classification & Clustering

Dataset	IJCNN	COVTYPE	MNIST	GISETTE
# instances	141,691	581,012	70,000	7,000
# features	22	54	780	5,000
# classes	2	7	10	2

Table 2: Statistics of Datasets for Outlier Detection

Dataset	ALOI	KDDCup99	SPAMBASE	PB
# instances	50,000	100,655	4,601	5,473
# features	64	98	57	10
# outliers	1,508	3,377	1,813	560

4.1 Datasets

The classification and clustering experiments are conducted on 4 publicly available benchmark datasets, which includes IJCNN, COVTYPE, MNIST and GISETTE [1]. The statistics of the datasets are summarized in Table 1. From the table we can see that the number of data points varies from 7,000 to 581,012 and the features vary from 22 to 5,000. It also includes binary class and multi-class datasets. In this way, we aim to give a comprehensive understanding of how RandLocal performs with datasets of various conditions.

The outlier detection experiments are conducted on 4 widely used outlier detection benchmark datasets [6], which includes ALOI [2], KDDCup99 [3], SPAMBASE and PageBlock (PB) [4]. The statistics of the datasets are summarized in Table 2. From the table we note that the datasets are extremely imbalanced, and thus can check the ability of RandLocal in dealing with extremely imbalanced datasets.

4.2 Compared Feature Engineering Methods

The representative state-of-the-art feature engineering algorithms RandLocal compare to are listed as follows:

- RAW: Raw features. It is a baseline to understand the performance without feature engineering.
- RBF Kernel: We use RBF kernel transformation to learn feature map $\mathbf{Q}\Sigma$ from the RBF similarity matrix $\mathbf{S}$. This is used as representative kernel transformation method.
- DAE: Denoising autoencoder [24] is a variant of autoencoder which is to learn a feature representation that is able to reconstruct the input data. Specifically, DAE is trained to reconstruct a clean "repaired" input from a corrupted version, which makes it able to extract more robust features. The encoded feature is used as feature representation.
- SDAE: Stacked denoising autoencoder [25] is a deep network based on stacking layers of DAE. Compared with the

[1] The 4 datasets are available from https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/
[2] https://elki-project.github.io/datasets/multi_view
[3] http://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_kddcup99.html
[4] SPAMBASE and PB are avaliable from https://archive.ics.uci.edu/ml/datasets/

Table 3: Classification performance comparison on four datasets in terms of Micro-F1 and Macro-F1.

Dataset	IJCNN		COVTYPE		MNIST		GISETTE	
Name	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
RAW	92.08±0.25	69.20±0.25	63.28±0.08	28.98±0.09	91.61±0.11	91.19±0.11	92.91±0.77	92.91±0.77
RBF Kernel	97.06±0.37	90.71±0.39	75.04±0.10	51.50±0.11	97.08±0.14	97.06±0.13	97.13±0.81	97.19±0.79
DAE	93.19±0.31	71.51±0.38	67.97±0.07	36.81±0.20	97.53±0.07	97.47±0.07	92.99±1.09	92.99±1.09
SDAE	93.88±0.34	72.47±0.40	66.13±0.07	41.02±0.19	97.75±0.09	97.68±0.09	94.24±1.13	94.26±1.13
RandLocal	**98.84**±0.27	**93.80**±0.35	**89.52**±0.06	**84.78**±0.20	**98.02**±0.05	**97.98**±0.05	**97.60**±0.96	**97.60**±0.97

DAE, features learned in a purely unsupervised fashion by SDAE are higher-level and could boost the performance of classification/clustering. We used a three-layer stacked DAE and the third layer feature representation is used for experiments. SDAE is used as a representative deep learning algorithm for unsupervised representation learning.

4.3 Classification Performance

To answer the first question, we first compare RandLocal with representative feature engineering algorithms introduced in Section 4.2 in terms of classification.

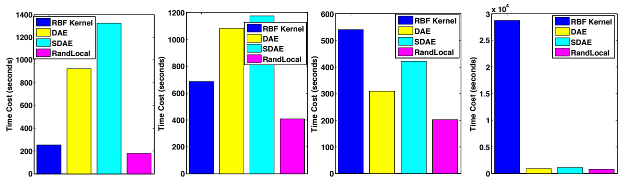
Two widely used classification evaluation metrics, i.e., Micro-F1 and Macro-F1, are adopted to evaluate the classification ability of the proposed framework RandLocal. The larger the Micro-F1 and Macro-F1 scores are, the better the classification results is, which indicates the dicriminativeness of the features.

For each feature learning algorithm, we first learn the representation from $\mathbf{X}$. We then use linear SVM to perform classification with the learned representation. We fix the classifier to be linear SVM as our goal is to test the dicriminativeness of the features learned by each method, not the effectiveness of the classifier. It is noteworthy that learning RBF kernel feature $\mathbf{Q\Sigma}$ (see section 1) is intractable for large datasets due to the huge space and time requirement of creating the RBF kernel similarity matrix and performing eigen decomposition. Thus, instead of explicitly learning $\mathbf{Q\Sigma}$ and applying a linear SVM, we used kernel SVMs to simulate this two step-approach. The equivalence of these two procedures is well known (see section 1).

There are some parameters to be set for the feature engineering methods and linear/kernel SVMs. In the experiment, we use 10-fold cross validation on the training data to set the parameters. Note that no test data are involved in the parameter tuning. Specifically, for RBF kernel, we do a grid search of the bandwidth from $[1e-5, 1e-4, \dots, 10]$. For DAE and SDAE, we search the dimension of latent features from $[\lceil 0.1d \rceil, \lceil 0.2d \rceil, \dots, \lceil 0.6d \rceil]$, where d is number of features. For RandLocal, we empirically set ϵ to be 0.0001 and T to be 400. The sensitivity of parameters ϵ and T on the classification performance of RandLocal will be analyzed in Section 4.7. The experiments are conducted using 5-fold cross validation and the average performance with standard deviation in terms of Macro-F1 and Micro-F1 are reported in Table 3. From the table, we make the following observations

- The transformed features outperforms RAW feature, which implies the importance of feature engineering; and
- Generally, RBF kernel outperforms or has close performance with DAE and SDAE, while RandLocal outperforms RBF kernel and the other compared method.

We also conducted the t-test on all performance comparisons and it is evident from t-test that all improvements are significant, which indicates the effectiveness of RandLocal.



(a) GISETTE (b) MNIST (c) IJCNN (d) COVTYPE

Figure 3: Time cost of feature learning

Table 4: Time cost (Seconds) of training using linear SVM

Dataset	GISETTE	IJCNN	MNIST	COVTYPE
RAW	2.12	2.29	8.55	39.86
RandLocal	1.89	2.49	12.06	98.48

Running Time Comparison: As stated previously, RandLocal is time efficient for both feature construction and model learning.

To show the time efficiency of RandLocal in feature construction, we plot the time cost of learning features for RBF Kernel, DAE, SDAE and RandLocal in Figure 3. From the figure, we can see that: (i) Generally, as the dataset becomes large, the time cost of RBF kernel increases dramatically. For example, on COVTYPE, which has around 580,000 samples, the time cost of RBF is of order 10^4 seconds, which is very time consuming; and (ii) Among the compared methods, RandLocal is time efficient as it's time complexity is linear to number of samples n and feature size d.

To show the efficiency of running linear SVM with the sparse engineered features from RandLocal, we compare the time of training and testing linear SVM with RandLocal features and that of RAW features in Table 4. From the table, we can see that the time cost of training SVM for RandLocal features is similar to that of RAW. Both of them are fast.

4.4 Clustering Performance

To further check the effectiveness of RandLocal, we use datasets in Table 1 for clustering. We first learn features and then perform clustering with K-means on learned features. Following the previous

Table 5: Clustering performance comparison (ACC±std). N/A means the result is not available due to large memory requirement to run RBF Kernel

Dataset	COVTYPE	MNIST	GISETTE
RAW	0.4092±0.0239	0.4642±0.0016	0.6842±0.0014
RBF Kernel	N/A	0.5329±0.0082	0.7059±0.0013
DAE	0.4214±0.0256	0.4843±0.0077	0.7027±0.0047
SDAE	0.4312±0.0267	0.5244±0.0035	0.7157±0.0036
RandLocal	**0.4574**±0.0198	**0.6872**±0.0054	**0.8322**±0.0163

Table 6: Clustering performance comparison (NMI±std)

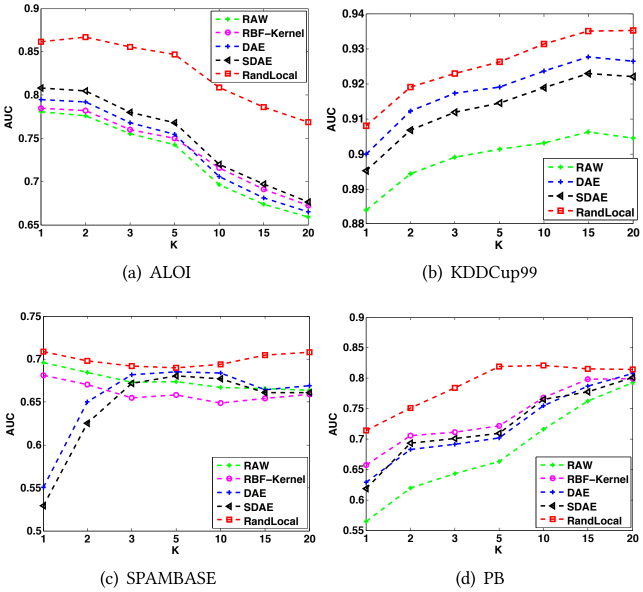
Dataset	COVTYPE	MNIST	GISETTE
RAW	0.1612±0.0136	0.4016±0.0013	0.1219±0.0010
RBF Kernel	N/A	0.5017±0.0046	0.1351±0.0011
DAE	0.1746±0.0149	0.4320±0.0055	0.1311±0.0056
SDAE	0.1788±0.0157	0.4853±0.0024	0.1257±0.0045
RandLocal	0.2011±0.0129	0.6685±0.0042	0.3652±0.0409

work [27], two widely used evaluation metrics, accuracy (ACC) and normalized mutual information (NMI), are employed to evaluate the quality of clusters. Note that labels are only used to evaluate the clustering results and are not used for feature construction or parameter tuning. The larger ACC and NMI are, the better the performance is. We use grid search to select parameters for each method and report the best results. For RandLocal, we empirically set $\epsilon = 0.0001$ and $T = 400$. Since K-means depends on initialization, we repeat the experiments 20 times and the average results are reported in Table 5 and 6. Note that due to the large memory requirement, we cannot learn RBF kernel features for COVTYPE and IJCNN, which we denote as N/A. We only report the results of COVTYPE, MNIST and GISETTE as we have similar observations for IJCNN. From the two tables, we find that RandLocal significantly outperforms RBF Kernel, DAE and SDAE, which indicates the effectiveness of the features constructed by RandLocal.

The experiments on classification and clustering answers our first question, i.e., by creating a sparse representation that captures multi-level granularity , RandLocal can significantly improves classification/clustering performance using a simple model.

4.5 Robustness to Noise

To answer the second question, i.e., the robustness of RandLocal to noise, we create two sets of synthetic datasets by adding and appending noise, respectively, to IJCNN, COVTYPE, MNIST and GISETTE. Specifically, for each dataset, $\mathbf{X} \in \mathbb{R}^{n \times d}$, in Table 1, we first create a noise matrix $\mathbf{G} \in \mathbb{R}^{n \times \bar{d}}$ by randomly sampling from the uniform distribution $[f_{min}, f_{max}]$, where $\bar{d} = \lceil 0.1d \rceil$, f_{min} is the minimal value in $\mathbf{X}$ and f_{max} is the maximal value. To add noise to $\mathbf{X}$, we randomly select $\bar{d}$ features, which we denote as $\mathcal{I}$. Then the synthetic data is created by adding $\mathbf{G}$ to the selected $\bar{d}$ features, i.e., $\tilde{\mathbf{X}} = \mathbf{X}$, $\tilde{\mathbf{X}}(:, I) = \tilde{\mathbf{X}}(:, I) + \mathbf{G}$. We do this for each dataset in Table 1, which generates four synthetic datasets. To append noise to $\mathbf{X}$, we directly concatenate $\mathbf{G}$ at the end of $\mathbf{X}$, i.e., $\hat{\mathbf{X}} = [\mathbf{X}, \mathbf{G}]$, which gives us another four datasets. With the noisy datasets, we then learn features and use linear SVM to perform classification.



(a) ALOI (b) KDDCup99

(c) SPAMBASE (d) PB

Figure 4: Outlier detection performance in terms of AUC

All the experimental settings and compared algorithms are the same as Section 4.3, i.e., we use 10 fold cross validation to tune parameters and perform 5 fold cross validation for classification. For RandLocal, we set $\epsilon = 0.0001$ and $T = 400$. To avoid the randomness of the results, the operations of adding and appending noise are carried out 5 times for each dataset and we report the average results. We only report the classification performance in terms of Micro-F1 as we have similar observations for Macro-F1. The average Micro-F1 with standard deviation on the two sets of synthetic datasets are shown in Table 7. From the table, we make the following observations: (i) In presence of noise, the performance of all the feature learning methods decreases, which satisfies our expectation that noisy features will degrade the classification performance; and (ii) RandLocal outperforms the compared feature engineering methods. In addition, generally, the performance decrease of RandLocal is smaller than that of RBF Kernle, DAE and SDAE, which shows the effectiveness of RandLocal in handling noisy features. This is because RandLocal randomly sample a subset of features to calculate the distance to anchors, which reduces the probability of including lots of noisy features; while the kernel methods uses all the noisy features to calculate the distance and DAE, SDAE tries to learn latent features that are good at reconstructing noise features, which amplifies the affects of noise. Furthermore, RandLocal repeats the feature construction T times, which can be seen as ensembling many feature sets and thus alleviate the problem caused by noise.

4.6 Outlier Detection

To answer the third question, we perform outlier detection using the datasets listed in Table 2. From Table 2 we can see that the datasets are extremely imbalanced, thus, it will be difficult to learn features for classes with few points. Therefore, by performing outlier detection, we aim to see the ability of RandLocal in learning

Table 7: Classification performance in presence of noise. Note that numbers inside parentheses in the table denote the performance reductions compared to the performance without noise in Table 3

Dataset	Add 10% Noise				Append 10% Noise			
Name	IJCNN	COVTYPE	MNIST	GISETTE	IJCNN	COVTYPE	MNIST	GISETTE
SVM-linear	91.39(0.75%)	61.31(3.12%)	89.90(0.49%)	91.89(1.10%)	91.84(0.26%)	61.32(3.10%)	90.35(1.38%)	91.91(1.07%)
SVM-rbf	93.36(3.81%)	49.12(34.54%)	92.15(5.08%)	95.29(1.90%)	93.62(3.54%)	49.12(34.54%)	92.65(4.56%)	95.20(1.99%)
DAE	90.26(3.15%)	63.11(7.15%)	91.64(6.04%)	75.36(18.96%)	90.14(3.27%)	63.54(6.52%)	91.13(6.56%)	67.74(27.15%)
SDAE	90.59(3.50%)	63.08(4.62%)	85.54(12.49%)	77.30(17.98%)	90.27(3.85%)	63.04(4.67%)	83.61(14.47%)	70.31(25.39%)
RandLocal	**96.00**(2.88%)	**80.62**(9.94%)	**93.63**(4.47%)	**96.86**(0.76%)	**95.62**(3.26%)	**78.80**(11.98%)	**93.80**(4.30%)	**96.24**(1.39%)

features from imbalanced datasets. Next, we introduce experimental settings and performances with comparison to other feature learning methods.

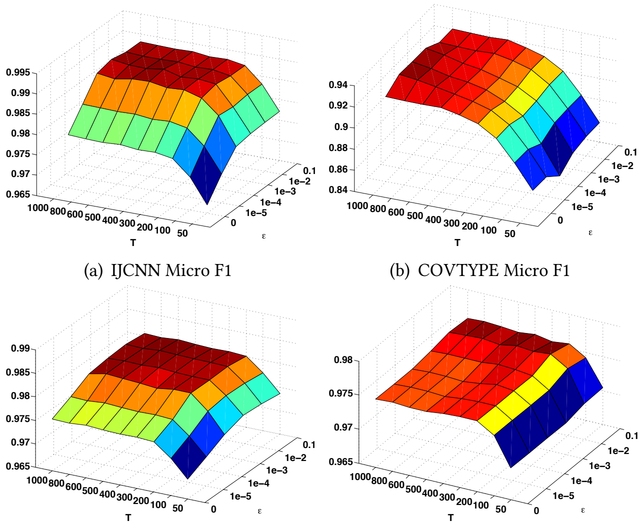
For outlier detection, we use the popular evaluation metric, areas under curve (AUC), which is a standard way to measure the effectiveness of outlier detection [18].

Note that our focus is to measure the effectiveness of the features for outlier detection, not the quality of outlier detection algorithms. Thus, for fair comparison, we first learn features using feature learning algorithms and then apply the same outlier detection algorithm on the learned features. In the experiments, we use the popular k-nearest neighbor distance based outlier detection [12], which uses the distance to the k-nearest neighbor as the outlier score. With the outlier score, we calculate AUC using the ground truth. Due to large memory requirement in calculating RBF kernel transformation on KDDCup99, we don't compare RandLocal with it on KDDCup99. We use grid search to tune the parameters for each feature learning algorithm. Each experiment is conducted 10 times and the average performance in terms of AUC is shown in Figure 4. From the figure, we make the following observations: (i) Generally, SDAE and DAE outperforms RAW, which implies that SDAE and DAE can learn useful features for outlier detection. SDAE is slightly better than DAE as SDAE can learn higher level features; and (ii) RandLocal outperforms the compared methods on the four outlier detection datasets. Especially on ALOI, compared to SDAE, the performance increases by more than 7.5%, which shows the strong ability of RandLocal in handling outliers. In addition, we can see that when K increases, the performance increase/decrease pattern of RandLocal is very similar to K while DAE and SDAE has different pattern to DAE on spambase. That's because RandLocal directly captures the multi-granularity of datasets while DAE and SDAE transforms the raw feature to latent space.

The result shows the strong ability of RandLocal in handling outliers. This is because outliers will typically be far away from their nearest anchors, and the feature values are chosen to be $\max\{D_{avg} - D_{min}(\mathbf{x}_i), \epsilon D_{avg}\}$, Therefore, all feature values are nonnegative, but the feature values of outliers are unusually small.

4.7 Parameter Analysis

The proposed framework has two important parameters, ϵ and T, where ϵ is a small number meant to add regularization and smoothing to the process against the presence of outliers and T is an inter specifies the number of random feature sets we want to get, which controls the level of granularity. In this section, we investigate the impact of the parameters ϵ and T on the performance of RandLocal. We only show the classification results since we have similar observations with clustering and outlier detection. We vary the values of ϵ as $\{0, 1e-5, 1e-4, 1e-3, 1e-2, 0.1\}$ and the values of T as $\{50, 100, 200, 300, 400, 500, 600, 800, 1000\}$. For each datasets, we first learn the features with RandLocal then perform 5-fold cross validation using linear SVM on the features. The average results are shown in Figure 5. It can be observed from the figure that: (i) Generally, with the increment of α, the performance tends to first increase then become stable. When we increase T from 50, each new feature sets are likely to introduce new granularity as there are few feature sets, which provides richer locality information that can improve performance a lot. When T is large enough, increasing T doesn't help as it is more likely to provide redundant features to the rich features we already have with large T. A value of T around 200 to 400 can give relatively good performance and save computational cost; and (ii) When we set $\epsilon = 0$, $\max\{D_{avg} - D_{min}(\mathbf{x}_i), \epsilon \cdot D_{avg}\}$ reduces to $D_{avg} - D_{min}(\mathbf{x}_i)$ and the performance is not as good as the performance when ϵ is not $1e-5$, which shows that adding ϵ can add regularization and smoothing to the process against the presence of outliers. When we increase ϵ from $1e-5$ to 0.1, there are slightly performance increment. In other words, RandLocal is relatively more stable to ϵ, which eases the parameter selection.



(a) IJCNN Micro F1 (b) COVTYPE Micro F1

(c) MNIST Micro F1 (d) GISETTE Micro F1

Figure 5: Parameter Analysis

5 CONCLUSION

The performance of machine learning algorithms i highly sensitive to the specific feature representation of the data used [3]. Good feature representations can significantly improve the machine learning and data mining performance such as classification [21] and recommender systems [13, 16, 29]. In this paper, we propose a novel unsupervised feature engineering method, RandLocal, as an accurate and efficient alternative to kernel methods. RandLocal is space and time efficient and learns features that capture multigranular locality. Extensive experiments on real-world datasets demonstrate the effectiveness of RandLocal for various applications such as classification and clustering. The analysis of RandLocal on noisy datasets and outlier detection shows the robustness of RandLocal in dealing imbalanced datasets and noisy features. Further experiments are conducted to analyze the sensitivity of the hyper-parameters.

There are several interesting directions need further investigation. First, in this paper, we employ Euclidean distance to calculate the distance between data points and anchor points. One direction is to study if other distance metrics can improve the performance of RandLocal. Second, in this paper, we randomly select subset of features to calculate the distance between anchor points. In practice, features may have correlations [28]. For example, for image datasets, nearby pixels are more likely to have similar values. Thus, another direction is to investigate the structures/correlations of features to improve RandLocal performance. Finally, linked data such as linked documents [26] are very pervasive. We also would like to extend RandLocal to include link information.

6 ACKNOWLEDGEMENTS

This material is based upon work supported by, or in part by, the National Science Foundation (NSF) under the grant #1614576.

REFERENCES

[1] Charu C Aggarwal. 2015. *Data mining: the textbook.* Springer.

[2] Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. 2004. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 6.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[4] Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. 2006. The curse of highly variable functions for local kernel machines. *Advances in neural information processing systems* 18 (2006), 107.

[5] Michael Brockmann, Theo Gasser, and Eva Herrmann. 1993. Locally adaptive bandwidth choice for kernel regression estimators. *J. Amer. Statist. Assoc.* 88, 424 (1993), 1302–1309.

[6] Guilherme O Campos, Arthur Zimek, Jörg Sander, Ricardo JGB Campello, Barbora Micenková, Erich Schubert, Ira Assent, and Michael E Houle. 2016. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery* 30, 4 (2016), 891–927.

[7] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. 2004. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 551–556.

[8] Chris Ding and Xiaofeng He. 2004. K-means clustering via principal component analysis. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 29.

[9] Pedro Domingos. 2012. A few useful things to know about machine learning. *Commun. ACM* 55, 10 (2012), 78–87.

[10] David K Duvenaud, Hannes Nickisch, and Carl E Rasmussen. 2011. Additive gaussian processes. In *Advances in neural information processing systems*. 226–234.

[11] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin.

[12] Ville Hautamaki, Ismo Karkkainen, and Pasi Franti. 2004. Outlier detection using k-nearest neighbour graph. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, Vol. 3. IEEE, 430–433.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 173–182.

[14] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 217–226.

[15] Aleksandar Lazarevic and Vipin Kumar. 2005. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 157–166.

[16] Xinyue Liu, Chara Aggarwal, Yu-Feng Li, Xiaugnan Kong, Xinyuan Sun, and Saket Sathe. 2016. Kernelized matrix factorization for collaborative filtering. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 378–386.

[17] Sebastian Mika, Bernhard Schölkopf, Alexander J. Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. 1998. Kernel PCA and De-Noising in Feature Spaces. In *Advances in Neural Information Processing Systems 11, [NIPS Conference, Denver, Colorado, USA, November 30 - December 5, 1998]*. 536–542.

[18] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. 2015. Reverse nearest neighbors in unsupervised distance-based outlier detection. *IEEE transactions on knowledge and data engineering* 27, 5 (2015), 1369–1382.

[19] Steffen Rendle. 2012. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)* 3, 3 (2012), 57.

[20] Bernhard Schölkopf and Alexander J Smola. 2002. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press.

[21] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. 2014. CNN features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 806–813.

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

[23] Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. 2006. Large scale multiple kernel learning. *Journal of Machine Learning Research* 7, Jul (2006), 1531–1565.

[24] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*. ACM, 1096–1103.

[25] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research* 11, Dec (2010), 3371–3408.

[26] Suhang Wang, Jiliang Tang, Charu Aggarwal, and Huan Liu. 2016. Linked Document Embedding for Classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, 115–124.

[27] Suhang Wang, Jiliang Tang, and Huan Liu. 2015. Embedded Unsupervised Feature Selection. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. 470–476.

[28] Suhang Wang, Yilin Wang, Jiliang Tang, Charu Aggarwal, Suhas Ranganath, and Huan Liu. 2017. Exploiting hierarchical structures for unsupervised feature selection. In *Proceedings of the 2017 SIAM International Conference on Data Mining*.

[29] Suhang Wang, Yilin Wang, Jiliang Tang, Kai Shu, Suhas Ranganath, and Huan Liu. 2017. What Your Images Reveal: Exploiting Visual Contents for Point-of-Interest Recommendation. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*. 391–400.

[30] Brian J Worton. 1989. Kernel methods for estimating the utilization distribution in home-range studies. *Ecology* 70, 1 (1989), 164–168.

[31] Hao Zhang, Alexander C Berg, Michael Maire, and Jitendra Malik. 2006. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, Vol. 2. IEEE, 2126–2136.

[32] Ding-Xuan Zhou. 2003. Capacity of reproducing kernel spaces in learning theory. *IEEE Transactions on Information Theory* 49, 7 (2003), 1743–1752.