Very Large scale data classification based on K-Means clustering & Multi Kernel SVM

Article in Soft Computing,
June 2019

Introduction

Very Large datasets with high dimensions of features:

- Challenge of speed & accuracy

2 Common Problems:

- High dimensional features
    - Feature Reduction/Compression Methods
- Very large number of training data
    - Data Reduction Technology/Method (DRT)

Proposed Method Overview

In this article,
we will go over DRT Method to classify large & very large scale datasets.

K-Means Clustering + Outlier Detection + Multi Kernel SVM

Or

Find Representative Data + Train & Test using SimpleMKL

Steps

1	2	3	4	5
K-Means Clustering	**Duplicate Removal**	**Outlier Detection**	**Human Labeling**	**SimpleMKL**
Select nearest & furthest points of each cluster	**Remove all duplicate data**	**Remove the last ROT-data based on their outlier score**	**Do labeling for the new representative dataset**	**Multi Kernel SVM**

Datasets

- **Large Datasets:**
  - Breast-W (683*10)
  - Messidor (1151*20)
  - Car (1728*6)
    - uacc vs other
  - Spambase (4601*57)

- **Very Large Datasets:**
  - Coil2000 (9’822*85)
  - Bank Marketing (45’211*17)
  - Skin Segmentation (245’057*4)
  - Covertype (581’012*54)
    - Aspen vs other

Proposed Algorithm

- **Clustering Stage**
- **Duplicate Removal Stage**
- **Outlier Detection Stage**
- **Training Stage using SimpleMKL**

Table 1 Proposed algorithm

Input: large-scale unlabeled data
Output: predicted label

Step 1 Clustering stage
1.1 select a small percentage of instances;
1.2 set the parameter of target cluster number of K-means as k=5 to 30;
1.3 set the clustering repeating sessions as RT=5 to 30;
1.4
    for i=1:RT
        use the K-means clustering method to cluster data into k classes;
        select the nearest and farthest instances to each cluster center
            and add them to the initial training set;
    end

Step 2 Outlier detection and reduction stage
2.1 delete repetitive instances in the initial training set;
2.2 compute the outlier scores of all instances;
2.3
    for i=1:ROT
        delete the instances with the highest score;
    end
    get the reduced training set;

2.4 label the reduced set;

Step 3 Training stage
3.1 set multi-kernel SVM parameters;
3.2 train the multi-kernel SVM classifier based on labeled training set;

Step 4 Evaluation
4.1 use the rest of instances to predict and test;
4.2 evaluate the performance.

Functions & Classes

Datasets & Datasets Usage

- **Datasets.ipynb**
- Read each dataset individually
- Change labels as -1 & +1
- Get training set randomly or equal number of data from each class
- Training size ratio must be specifed

```
# No normalization
breast_w_dataset = Breast_W_Dataset('./Datasets/breast-cancer-wisconsin.data', "Breast-W",
                                    train_size=0.1, 'Class', normalization_method='None')

print("data shape: ", breast_w_dataset.dataframe.to_numpy().shape)
print("data-train shape: ", breast_w_dataset.x_train.shape)
print("data-test shape: ", breast_w_dataset.x_test.shape)
print("output classes: ", np.unique(breast_w_dataset.y_test))
breast_w_dataset.dataframe.head()
```

```
Started reading dataset  Breast-W ...
Finished reading dataset  Breast-W ...
data shape:  (683, 11)
data-train shape:  (68, 10)
data-test shape:  (615, 10)
output classes:  [-1  1]
```

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
575	385103	5	1	2	1	2	1	3	1	1	1
608	557583	5	10	10	10	10	10	10	1	1	-1
68	1120559	8	3	8	3	4	9	8	9	8	-1
404	1223543	1	2	1	3	2	1	1	2	1	1
431	1276091	5	1	1	3	4	1	3	2	1	1

K-Means

Do K-Means clustering for RT times.

Each time, select the nearest & furthest data of each cluster.

2 Ways:

- Sklearn K-Means
- Implementing K-Means from scratch

- **Select nearest and furthest data of each cluster**

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

where $\left\| x_i^{(j)} - c_j \right\|^2$ is a chosen distance measure between a data point and its cluster center.

---

1. Define a target cluster number k.
2. Randomly select k data points as the initial cluster centers.
3. Repeat until the mean values of clusters do not change anymore:
   {Each data point is assigned to the most similar cluster according to the distance to the centers;
   Update the mean value of each cluster and calculate new centers;
   }

sklearn K-Means

- **kmeans = KMeans(n_clusters=k,**
  init="kmeans++ | random",
  random_state=0).fit(X)

- **kmeans.labels_**
  - Labels of each data in X

- **kmeans.cluster_centers_**
  - Center of each cluster
  - To find nearest & furthest point of each cluster

K-Means from scratch

- **initialize_centroids()**
  - Initialize centroids based on initialization_method: ‘random’ | ‘kmeans++’
- **main_loop()**
  - Repeat labeling data points and moving centroids until centroids do not change anymore
  - If convergence didn’t happen until iteration 300, stop
- **closest_farthest_of_each_cluster()**
  - Find nearest & furthest data of each cluster

Instance Selection

- **It can be set to use the MyKMeans or kmeans of sklearn**
- **Initialization of cluster centroids can be random or kmeans++**
  - Kmeans++ results in a faster convergence than random
- **Nearest & furthest data of each cluster based on Euclidean distance is chosen as the new x-train**
- **Class Instanceselection select new representative dataset using proposed method**

Instance Selection Class Usage

These two functions call K-Means and finding nearest & furthest data of clusters for KT times.

- **My_Kmeans_main_loop()**
- **Sklearn_Kmeans_main_loop()**

```
# do instance selection using K-Means
print("Instance selection started ...")
instance_selection = InstanceSelection(dataset.x_train, num_of_clusters=k, repeating_time=RT,
                                       KMeans_Model=KMeans_Model,
                                       kmeans_initialization_method=kmeans_initialization_method,
                                       kmeans_max_iter=kmeans_max_iter)
new_x_train = instance_selection.main_loop()
print("Instance selection finished after ", instance_selection.processing_time, "...")
print("x_train shape kmeans after instance selection: ", new_x_train.shape, "\n")
```

Outlier Detection

Used an outlier detection method proposed in 2013 by Michael S Kim, "Robust, Scalable Anomaly Detection for Large Collections of Images"

Method uses the distribution of Euclidean distance to get the average outlier score for each data. Data with highest outlier score is considered to be an outlier.

$$KS(p_j, p_i) = \sup_x |F_{p_j}(x) - F_{p_i}(x)|$$

$$KSE(p_j) = \frac{1}{n-1} \sum_{\substack{i=1 \\ i \neq j}}^{n} KS(p_j, p_i)$$

Outlier Detection

- **KS_euclidean_distance(pj, pi)**
  - Find KS of pj, pi using Euclidean distance of pj with other points in X
- **KS_distribution_of_euclidean_distance(pj, pi)**
  - Find KS of pj, pi using distribution of Euclidean distance of pj with other points in X
  - We use ECDF function
  - from statsmodels.distributions.empirical_distribution import ECDF
- **KSE()**
  - Find average KS statistic
- **outlier_detection_main_loop()**
  - Find outlier score of each data and remove the top KOT-data with highest score

Outlier Detection Usage

```
# remove duplicate instances
print("Started removing duplicate instances ...", dataset_name, "...")
new_x_train = np.unique(new_x_train, axis=0)
print("x_train shape after duplication deleting: ", new_x_train.shape, "\n")

# do outlier detection and deletion using KSE
print("Outlier Detection started ...")
outlier_detection = OutlierDetectionReductionKSE(new_x_train, repeating_time=ROT, KS_type=KS_type, ROT_type=ROT_type)
print("x_train shape after outlier detection: ", outlier_detection.x_train.shape)
print("Outlier Detection finished after ", outlier_detection.processing_time, "...\n")
```

Stage_1_and_2() Function

- **Stage_1_and_2()**
  - Call Instance Selection
  - Remove duplicate instances
  - Call Outlier Detection

Skin Segmentation Instance Selection + Outlier Detection + Human Labeling

```
1 skin_dataset = Datasets.Skin_NonSkin_Dataset('./Datasets/Skin_NonSkin.txt', "Skin Segmentation",
2                                              train_size=0.0005, normalization_method="None")
```

```
Started reading dataset  Skin Segmentation ...
Finished reading dataset  Skin Segmentation ...
```

```
1 new_x_train = Stage_1_and_2(skin_dataset, "Skin Segmentation", k=5, RT=15, ROT=5,
2                             KMeans_Model="My_KMeans", kmeans_initialization_method="kmeans++", kmeans_max_iter=300,
3                             KS_type="distribution_of_euclidean_distance", ROT_type="run_1_time")
4 new_x_train.shape
```

```
x_train shape:  (122, 3)
Started Stage I & II on dataset  Skin Segmentation ...
Instance selection started ...
Instance selection finished after  0.19309711456298828 ...
x_train shape kmeans after instance selection:  (150, 3)

Started removing duplicate instances ... Skin Segmentation ...
x_train shape after duplication deleting:  (24, 3)

Outlier Detection started ...
x_train shape after outlier detection:  (19, 3)
Outlier Detection finished after  0.138962984085083 ...

Finished.
~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.~.
```

```
]: (19, 3)
```

Human Labeling & Usage

- **Find label of each data of new data-train with the original data train**
- **Save new data-train to be used for training in the next stage in NewDatasets folder**
- **Also Save the data test in NewDatasets folder**

Functions:

- **LabelingNewDataset()**
  - Label new datas after instance selection
- **create_new_dataset_csv_file()**
  - Save new labeled dataset-train in csv file fot training stage
- **save_data_test()**
  - Save data-test in csv file for testing stage

```
breast_w_dataset_human_labling = NewDatasetHumanLabeling(breast_w_dataset, new_x_train,
                                                         "./NewDatasets/new_breast_w_train.data")
save_data_test(breast_w_dataset.x_test, breast_w_dataset.y_test, "./NewDatasets/new_breast_w_test.data")
```

LibSVM

- SVC of Sklearn.svm is implmeneted based on LibSVM method

- data-set is splited to almost 60% to 80% for data train & the remaining for data test

```
self.svm_model = SVC(C=self.C, kernel=self.kernel)
self.svm_model.fit(x_train, y_train)
model_score = self.svm_model.score(data_x, data_y)
```

```
libsvm = LibSVM_Method()
```

Testing LibSVM with NewDatasets created by instance selection

Breast-W

```
breast_dataset = ReadDataset('Breast-w', './NewDatasets/new_breast_w_train.data', './NewDatasets/new_breast_w_test.data')

```

```
Started reading data train of dataset  Breast-w ...
Finished reading data train of dataset  Breast-w ...
Started reading data test of dataset  Breast-w ...
Finished reading data test of dataset  Breast-w ...
```

```
libsvm.train(breast_dataset.x_train, breast_dataset.y_train, "Breast-w")
print("\n")
print("Accuracy on data train: ", libsvm.svm_mean_accuracy(breast_dataset.x_train, breast_dataset.y_train, "Breast-w"),
      " | processing time: ", libsvm.prediction_accuracy_processing_time)

print("Accuracy on data test: ", libsvm.svm_mean_accuracy(breast_dataset.x_test, breast_dataset.y_test, "Breast-w"),
      " | processing time: ", libsvm.prediction_accuracy_processing_time)
```

```
Started training LibSVM on dataset Breast-w ...
Training LibSVM on dataset  Breast-w  Finished after  0.4678378105163574


Calculating accuracy of LibSVM on dataset  Breast-w  Finished after  0.001997232437133789
Accuracy on data train:  40.0  | processing time:  0.001997232437133789
Calculating accuracy of LibSVM on dataset  Breast-w  Finished after  0.0020105838775634766
Accuracy on data test:  34.146341463414636  | processing time:  0.0020105838775634766
```

Stage 3

- **After being labeled, the selected data set is used as the training set for building a classifier in the third stage.**
- **It has been proved that the multi-kernel SVM has a better performance than traditional SVMs.**
- **We use SimpleMKL method to train a model because the SimpleMKL implementation converges faster and more efficient when compared with other multi-kernel learning methods.**

Kernel

In kernel methods, the data representation is chosen through kernel K(x, x0).

This kernel plays two roles:

1. It defines the similarity between two examples x and x0,
2. And defining an appropriate regularization term for the learning problem.

Recent applications have shown that using multiple kernels instead of a single one can enhance the interpretability of the decision function and improve performances.

A convenient approach is to consider that the kernel K(x, x0) is actually a convex combination of basis kernels:

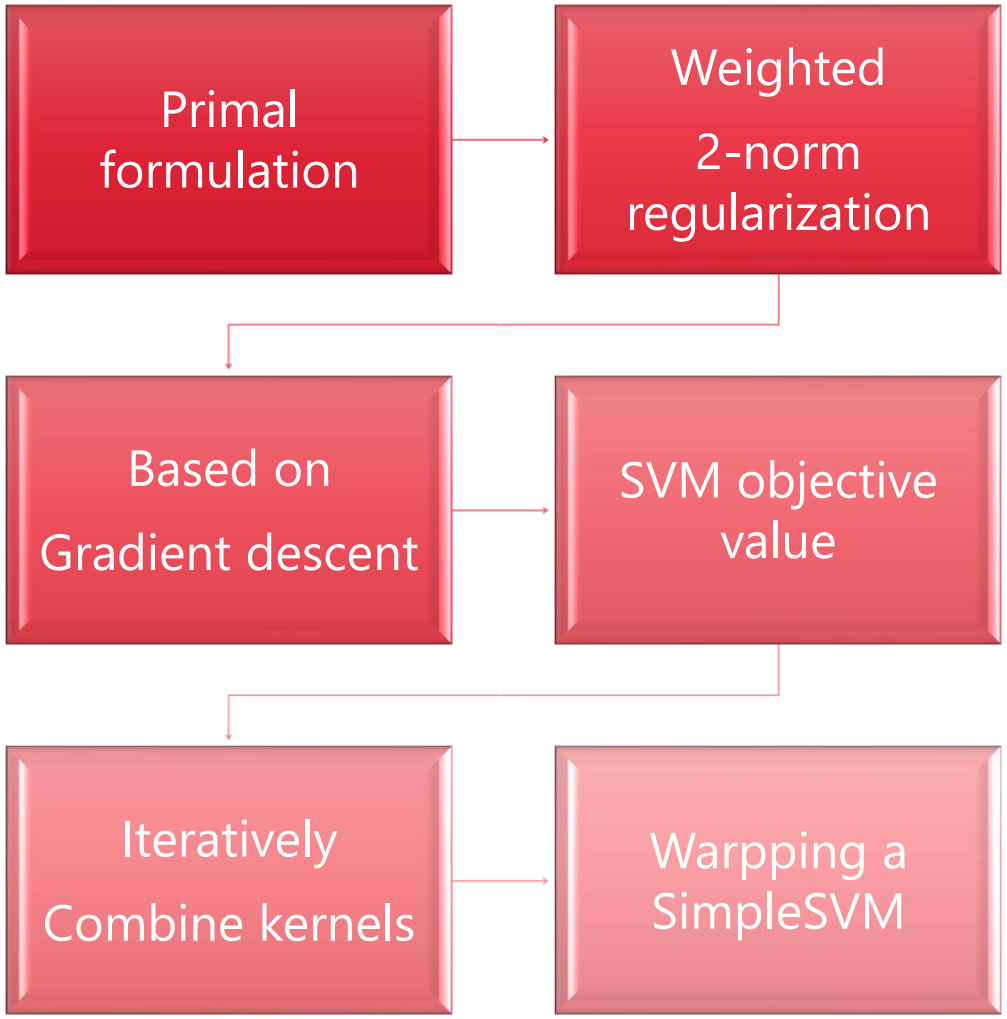
$$K(x,x') = \sum_{m=1}^{M} d_m K_m(x,x') \ , \qquad \text{with } d_m \geq 0 \ , \ \sum_{m=1}^{M} d_m = 1$$

The problem of data representation through the kernel is then transferred to the choice of weights d_m.

SimpleMKL

Learning both the coefficients α_i and the weights d_m in a single optimization problem is known as the multiple kernel learning (MKL) problem.

In SimpleMKL, the mixed-norm regularization was replaced by a weighted 2-norm regularization, where the sparsity of the linear combination of kernels is controlled by a 1-norm constraint on the kernel weights. This leads to a smooth and convex optimization problem.



SimpleMKL

It consist of two main steps:

i. **Solving a canonical SVM optimization problem with given d_m**

ii. **Updating d_m using the gradient calculated with α found in the first step**

Formulas in SimpleMKL

The dual problem is a key point for deriving MKL algorithms. we have the following constrained optimization problem:

$$\min_{d} J(d) \quad \text{such that} \quad \sum_{m=1}^{M} d_m = 1,\ d_m \geq 0 \ ,$$

$$J(d) = \left\{ \begin{array}{ll} \min\limits_{\{f\},b,\xi} & \frac{1}{2}\sum\limits_{m}\frac{1}{d_m}\|f_m\|^2_{\mathcal{H}_m} + C\sum\limits_{i}\xi_i \quad \forall i \\ \text{s.t.} & y_i\sum\limits_{m} f_m(x_i) + y_i b \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad \forall i \ . \end{array} \right.$$

Implementation

- In Kernel class, we implemented various kernel functions and return those methods.
  We compute 8 different kernel to use later.
- In DataSet class, we read the result dataset (representative samples) of previous steps which was saved as a new csv file.
- In Call_on_datasets class, we get the dataset's path of representative samples and fit a SimpleMKL on train and test on test set. Furthermore we report the execution time elapsed.

Formulas in SimpleMKL

$$\begin{aligned} \max_{\alpha} \quad & -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \sum_m d_m K_m(x_i,x_j) + \sum_i \alpha_i \\ \text{with} \quad & \sum_i \alpha_i y_i = 0 \\ & C \geq \alpha_i \geq 0 \quad \forall i \ , \end{aligned}$$

$$J(d) = -\frac{1}{2}\sum_{i,j}\alpha_i^\star\alpha_j^\star y_i y_j \sum_m d_m K_m(x_i,x_j) + \sum_i \alpha_i^\star \ ,$$

The objective value J(d) can be obtained by any SVM algorithm.

The overall complexity of SimpleMKL is tied to the one of the single kernel SVM algorithm.

We implemented this in get_J(.) method in our SimpleMKL class.

Formulas in SimpleMKL

By simple differentiation of the dual function (previous slide) with respect to dm, we have:

$$\frac{\partial J}{\partial d_m} = -\frac{1}{2}\sum_{i,j}\alpha_i^\star \alpha_j^\star y_i y_j K_m(x_i, x_j) \qquad \forall m$$

the complexity of the gradient computation is of the order of m $\cdot n_{sv}^2$, with n_{sv} being the number of support vectors for the current d.

We implemented this in get_derivative_J(.) method in our SimpleMKL class.

Formulas in SimpleMKL

Here J(·) is convex and differentiable with gradient, So use a reduced gradient method, which converges for such functions.

the descent direction for updating d as:

$$D_m = \begin{cases} 0 & \text{if } d_m = 0 \text{ and } \frac{\partial J}{\partial d_m} - \frac{\partial J}{\partial d_\mu} > 0 \\ -\frac{\partial J}{\partial d_m} + \frac{\partial J}{\partial d_\mu} & \text{if } d_m > 0 \text{ and } m \neq \mu \\ \sum_{g \neq \mu, d_v > 0} \left(\frac{\partial J}{\partial d_v} - \frac{\partial J}{\partial d_\mu} \right) & \text{for } m = \mu \quad . \end{cases}$$

The usual updating scheme is d ← d + γD , where γ is the step size.

We implemented this in get_D_direction(. , . , .) method in our SimpleMKL class.

Formulas in SimpleMKL

- **once a descent direction D has been computed, we look for the maximal admissible step size in that direction**
- **check whether the objective value decreases or not. The maximal admissible step size corresponds to a component, say dv, set to zero.**
  We implemented this in get_gamma_max_and_index(. , .) method in our SimpleMKL class.
- **If the objective value decreases, d is updated, we set Dv = 0**
- **This procedure is repeated until the objective value stops decreasing.**
- **Then, we look for the optimal step size γ, which is determined by using a one-dimensional line search, with proper stopping criterion, such as Armijo's rule, to ensure global convergence.**
  We implemented this in Armijo_rule(.) method in our SimpleMKL class.
- **The algorithm is terminated when a stopping criterion is met. Our implementation, based on the duality gap, although we implemented KKT condition as well.**

Formulas in SimpleMKL

the duality gap(the difference between primal and dual objective values), which should be zero at the optimum.

the MKL duality gap is:

$$\text{DualGap} = J(d^\star) - \sum_i \alpha_i^\star + \frac{1}{2} \max_m \sum_{i,j} \alpha_i^\star \alpha_j^\star y_i y_j K_m(x_i, x_j)$$

with KKT condition the necessary optimality conditions are approximated by the following termination conditions:

$$|dJ_{\min} - dJ_{\max}| \leq \varepsilon \quad \text{and} \quad \frac{\partial J}{\partial d_m} \geq dJ_{\max} \quad \text{if } d_m = 0$$

We implemented this in get_duality_gap(. , . , .) and get_KKT_condition(. , . , .) method in our SimpleMKL class.

Algorithm in SimpleMKL

Algorithm 1 SimpleMKL algorithm

set $d_m = \frac{1}{M}$ for $m = 1, \ldots, M$
while stopping criterion not met **do**
  compute $J(d)$ by using an SVM solver with $K = \sum_m d_m K_m$
  compute $\frac{\partial J}{\partial d_m}$ for $m = 1, \ldots, M$ and descent direction D (12).
  set $\mu = \underset{m}{\operatorname{argmax}}\, d_m$, $J^\dagger = 0$, $d^\dagger = d$, $D^\dagger = D$
  while $J^\dagger < J(d)$ **do** {descent direction update}
    $d = d^\dagger$, $D = D^\dagger$
    $\nu = \underset{\{m|D_m<0\}}{\operatorname{argmin}} -d_m/D_m$, $\gamma_{\max} = -d_\nu/D_\nu$
    $d^\dagger = d + \gamma_{\max} D$, $D^\dagger_\mu = D_\mu + D_\nu$, $D^\dagger_\nu = 0$
    compute $J^\dagger$ by using an SVM solver with $K = \sum_m d^\dagger_m K_m$
  end while
  line search along D for $\gamma \in [0, \gamma_{\max}]$ {calls an SVM solver for each γ trial value}
  $d \leftarrow d + \gamma D$
end while

We implemented this in fit(. , .) method in our SimpleMKL class.

Random Selection

- In this class we randomly select representative samples over the dataset.
- Then we run our stage 3 (SimpleMKL) on these random samples to get min, max and average accuracy over 15 rounds evaluation.
- Finally we compare our proposed method results with random.

Article Results

Table 2 Accuracy based on small- to large-sized data sets

Data set	Samples	Training instance size		Accuracy (%)		Running time (s)	
		Proposed	LibSVM	Proposed	LibSVM	Proposed	LibSVM
Breast-w	683*10	**19**	546	92.44	**92.75**	1.61	**0.17**
Messidor	1151*20	**36**	920	60.23	**64.22**	2.22	**0.56**
Car	1728*6	**56**	1282	87.35	**90.20**	3.67	**0.26**
Spambase	4601*57	**26**	3680	**77.75**	77.22	**9.39**	12.30

Bold text highlights the best results of comparisons

Table 3 Performance on very large-scale data sets

Data set	Size	Parameters	Performance						
			Training Size	Accuracy %	Execution time (s)				
					Clustering	Deleting	Training	Testing	Total
Coil2000	9822*85	ratio = 0.02; $k = 26$; RT = 12; ROT = 26	68	92.73	0.39	1.60	2.70	0.32	5.01
Bank marketing	45,211*17	ratio = 0.1; $k = 9$; RT = 15; ROT = 5	13	88.22	2.55	0.02	0.27	0.05	2.89
Skin segmentation	245,057*4	ratio = 0.0005; $k = 5$; RT = 15; ROT = 5	18	93.21	0.07	0.05	0.37	0.61	1.10
Covertype (Aspen vs others)	581,012*54	ratio = 0.02; $k = 10$; RT = 30; ROT = 5	50	95.91	55.87	0.34	1.00	1.42	58.63

Table 5 Accuracy comparison with random-selection method

Data set	Data size	Training number	Random-selection method			Proposed method		
			Min	Max	Average	Min	Max	Average
Coil2000	9822*85	68	88.00	93.90	91.35	89.67	94.01	**92.73**
Bank marketing	45,211*17	13	59.62	88.73	83.13	88.21	88.21	**88.21**
Skin segmentation	245,057*4	18	82.21	98.22	92.47	87.06	97.45	**93.21**
Covertype	581,012*54	50	1.63	98.37	83.05	95.82	96.71	**95.91**

Bold text highlights the best results of comparisons

Our Results

Data Reduction (DRT) Results

part1(large dataset)

Dataset	New Training Size	Clustering time (s) MyKMeans	Outlier Detection time (s)
Breast-W (683*10)	(19)/ 21	0.27	0.18
Messidor (1151*20)	(36)/ 36	0.69	0.45
Car (uacc vs other) (1728*6)	(56)/ 62	0.55	1.34
Spambase (4601*57)	(26)/ 26	7.44	0.42

Data Reduction (DRT) Results

part2(very large dataset)

Dataset	New Training Size	Clustering time (s) MyKMeans	Outlier Detection time (s)
Coil2000 (9822*85)	(68)/ 84	(0.39)/ 6.3	(1.60)/ 3.31
Bank Marketing (45211*17)	(13)/ 13	(2.55)/ 39.0	(0.02)/ 0.07
Skin Segmentation (245057*4)	(18)/ 18	(0.07)/ 0.19	(0.05)/ 0.13
Covertype (Aspen vs other) (581012*54)	(50)/ 15	(55.87)/ 845.9	(0.34)/ 0.09

LibSVM Results

(Train over about 70-80% of dataset – part1 large dataset)

Dataset	Train Accuracy (%)	Test Accuracy (%)	Train processing time (s)	Test processing time (s)
Breast-W	61	(92.75)/ 94.33	(0.17)/ 0.004	0.005
Messidor	76.52	(64.22)/ 75.75	(0.56)/ 2.58	0.006
Car	85.37	(90.20)/ 84.48	(0.26)/ 0.035	0.011
Spambase	92.75	(77.22)/ 93.78	(12.30)/ 222.11	0.05

LibSVM Results

(Train over about 70-80% of dataset – part2 very large dataset)

Dataset	Train Accuracy (%)	Test Accuracy (%)	Train processing time (s)	Test processing time (s)
Coil2000	100	57.64	0.003	0.02
Bank Marketing	100	74.11	1.75	0.07
Skin Segmentation	100	94.38	0.002	0.36
Covertype	-	-	-	-

SimpleMKL Results

part1(large dataset)

Dataset	Train Accuracy (%)	Test Accuracy (%)	Train processing time (s)	Test processing time (s)
Breast-W	44.00	(92.44)/ 64.00	(1.61)/ 43.65	0.04
Messidor	100	(60.23)/ 74.11	(2.22)/ 1.34	0.16
Car	100	(87.35)/ 98.38	(3.67)/ 0.55	0.29
Spambase	96.15	(77.75)/ 88.46	(9.39)/ 13.97	0.04

SimpleMKL Results

part2 (very large dataset)

Dataset	Train Accuracy (%)	Test Accuracy (%)	Train processing time (s)	Test processing time (s)
Coil2000	95.23	(92.73)/ 96.42	(2.70)/ 1.55	(0.32)/ 0.85
Bank Marketing	92.30	(88.22)/ 100	(0.27)/ 0.06	(0.05)/ 0.01
Skin Segmentation	100	(93.21)/ 100	(0.37)/ 0.04	(0.61)/ 0.11
Covertype	100	(95.91)/ 93.33	(1.00)/ 0.05	(1.42)/ 0.09

Random vs. Our Results

part2(very large dataset)

Dataset	Random-selection method			Proposed method		
	Min Accuracy (%)	Max Accuracy (%)	Average Accuracy (%)	Min Accuracy (%)	Max Accuracy (%)	Average Accuracy (%)
Coil2000	(88.00)/ 88.23	(93.90)/ 95.58	(91.35)/ 92.54	(89.67)/ 90.47	(94.01)/ 97.62	(92.73)/ 94.44
Bank Marketing	(59.62)/ 0.00	(88.73)/ 100	(83.13)/ 45.12	(88.21)/ 76.92	(88.21)/ 100	(88.21)/ 90.77
Skin Segmentation	(82.21)/ 66.66	(98.22)/ 94.44	(92.47)/ 81.85	(87.06)/ 72.22	(97.45)/ 94.44	(93.21)/ 82.59
Covertype	(1.63)/ 48.00	(98.37)/ 84.00	(83.05)/ 63.06	(95.82)/ 66.66	(96.71)/ 93.33	(95.91)/ 84.00

Resources

References:

- **Very large-scale data classification based on K-means clustering and multi-kernel**
  SVMhttps://dl.acm.org/doi/abs/10.1007/s00500-018-3041-0

- **Robust, Scalable Anomaly Detection for Large Collections of Images**
  https://ieeexplore.ieee.org/document/6693467

- **Simplemkl**
  https://www.researchgate.net/publication/29623253_Simplemkl

Thank You