Industrial and Applications Paper

Parallel Duplicate Detection in Adverse Drug Reaction Databases with Spark

Chen Wang
CSIRO
Sydney, Australia
chen.wang@csiro.au

Sarvnaz Karimi
CSIRO
Sydney, Australia
sarvnaz.karimi@csiro.au

ABSTRACT

The World Health Organization (WHO) and drug regulators in many countries maintain databases for adverse drug reaction reports. Data duplication is a significant problem in such databases as reports often come from a variety of sources. Most duplicate detection techniques either have limitations on handling large amount of data or lack effective means to deal with data with imbalanced label distribution. In this paper, we propose a scalable duplicate detection method built on top of Spark to address these problems. Our method uses the kNN (k nearest neighbors) classifier to identify labelled report pairs that are most useful for classifying new report pairs. To deal with the high computational cost of kNN, we partition the labelled data into clusters for parallel computing. We give a method to minimize the cross-cluster kNN search. Our experimental results show that the proposed method is able to produce robust duplicate detection results and scalable performance.

1. INTRODUCTION

Adverse drug reactions, or ADRs, impose significant hazards to public health. They are one of the leading causes of hospitalization, disabilities, and death around the world. ADRs incur significant costs to health-care systems [10, 19]. Post-marketing drug safety surveillance plays an increasingly important role in ADR detection in comparison with pre-marketing drug clinical trails as clinical trials have limitations on the number of patients involved and the diversity of patient groups. Post-marketing drug safety surveillance mainly uses *Spontaneous Reporting Systems (SRS)* to detect signals of potential ADRs. These signals are then further assessed by experts to establish a causal relationship between a drug and an ADR. The World Health Organization (WHO) and drug regulators in many countries, such as the FDA in the US and the TGA in Australia maintain databases for adverse drug reaction reports. ADR reports are submitted from a variety of sources including general practitioners, pharmacists, hospitals, and consumers etc. The ADR report database is the major part of the reporting system. Many drug safety assessment methods detect potential ADR signals through the comparison of the reported ADR ratio of a specific drug and that of other drugs in the database. Disproportionality often indicates a potential ADR signal [6, 7]. As these methods are sensitive to the number of ADR reports, data quality in these databases is essential to the performance of ADR detection. One significant problem faced by such a database is *report duplication*. Duplicates often result from two sources. First, reports from different data sources have overlaps as the same ADR may be reported by different organizations through different channels. Second, the follow-up reports of the same ADR are wrongly put as separate records in the databases. Duplicates may distort the report ratio of an ADR and affect the performance of these methods significantly. Nkanza and Walop [17] reported a 5% duplication rate in vaccine adverse event data, providing an indication of the spread of the problem.

Duplicated reports in an ADR database are often not exactly the same in each field. Table 1 shows two examples. In the first example, *report A* and *report B* are duplicate, but they differ in the *reaction outcome description* field and *report description* field. In the second example, *report C* and *report D* are duplicate, but they differ in *patient age*, *ADR name* and *report description* field. The different values in the *patient age* field are likely to be an error introduced when entering a handwritten report.

A database record consists of multiple fields. Duplicate detection techniques therefore have two levels: *field matching* and *record matching*. Field matching mainly concerns comparing numerical, categorical and string values in each field. Record matching concerns whether two or more feature vectors formed by common fields belonging to different records are duplicate.

There are many existing works on duplicate detection in relational databases. Early works are referred as *record linkage* with a focus on linking together two or more separately recorded pieces of information concerning an individual case [16]. Large amount of work deals with the comparison of fields that can identify a particular record, such as name, address, and age. The values of these fields are normally short strings. Many field matching techniques concern approximate comparison of strings based on various similarity metrics. Commonly used string similarity metrics include *edit distance* [13], *Hamming distance* [8], *cosine distance* and *Jaccard coefficient* [3] etc. Field matching alone is not able to detect many duplicates, e.g., two string values in the surname field can be totally different in the case that a woman

(a)

Field Name	Report A	Report B
patient age	46	46
patient sex	M	M
patient state	-	-
onset date	-	-
reaction outcome description	Unknown	Recovered
drug name	Atorvastatin	Atorvastatin
ADR name	Rhabdomyolysis	Rhabdomyolysis
report description	Reference number xxx is a literature report received on 02-Oct-2013 pertaining to a 46 year-old male patient who experienced rhabdomyolysis while on atorvastatin for the treatment of unknown indication.	The 46-year-old male subject started treatment with atorvastatin calcium 80 mg, start date and duration of therapy unknown. In 2009,the subject presented with myalgia shoulder and hips for 2-3 weeks, minimal weakness and was diagnosed with rhabdomyolysi

(b)

Field Name	Report C	Report D
patient age	84	34
patient sex	F	F
patient state	Not Known	Not Known
onset date	30/04/2013 00:00:00	30/04/2013 00:00:00
reaction outcome description	Unknown	Recovered
drug name	Influenza Vaccine,Dtpa Vaccine	Influenza Vaccine,Dtpa Vaccine
ADR name	Vomiting,Pyrexia,Cough,Headache	Cough,Headache,Choking sensation,Chills,Vomiting
report description	On 30 April 2013, in the evening, within hours of vaccination with Boostrix, the subject experienced uncontrollable cough and felt like she was chocking On the same night, the subject experienced headache.On the 01-05-2013 at 3am, the subjet experienced.	In the afternoon of 30-Apr-2013, the patient experienced uncontrollable cough for 2 hours, then started choking and had to call an ambulance. She required oxygen before she felt better and so didn't go to hospital. She then reported a headache, cold shive

Table 1: Sample duplicated reports.

changes her maiden name to her husband's surname, but they belong to the same record.

Record matching considers each field as an element of a feature vector of a record. It combines the differences among common fields of two or more records to determine whether they are duplicate. The ways of combining field similarities of records differentiates record matching techniques. Some techniques calculate the probability of difference of values in each common field, converts these probabilities to log values and add them together [16, 11, 12]. Some techniques use supervised decision tree induction and unsupervised clustering to determine if a record is likely to match a set of other data records [5]. Active learning is also used for record matching to reduce the amount of training data [20].

However, the effectiveness of existing duplicate detection techniques on ADR databases is not well investigated mainly due to the slow progress of the industryâĂŹs adopting information technologies. We collaborate with the Therapeutic Goods Administration (TGA) in Australia and use the ADR reports they collected during 6 month period to carry out this work. In this paper, we study the duplicate detection problem in ADR databases with a focus on detection performance and system scalability that is important for fast growing data in this area. Our contributions are as below:

1. To identify duplicates in ADR reports, pairwise distances between these reports often need to be calculated. The distribution of duplicate pairs and non-duplicate pairs is highly imbalance. This poses a significant challenge when selecting report pairs from a large dataset to train a classifier for exploiting useful information. The classification results are highly influenced by the overwhelming majority of non-duplicate report pairs. There is no generally applicable classification method to address this issue. We develop a kNN based method to address this problem for ADR reports. kNN is helpful for the classifier to learn from individual reports and makes the classification results easy to understand. It also offers flexibility through assigning weights to the information carried in the neighbors in decision making. Our extensive experiments show that our method produces more robust detection results in comparison to SVM based classifiers.

2. A drug regulator database in a country with a relatively small population may already contain a large number of ADR reports that easily overwhelms the computing capacity for effective duplicate detection. In addition, kNN is both data- and compute-intensive when the dataset grows large. The MapReduce model is a convenient way to scale up the duplicate detection. However, its Google and open source implementation are mainly optimized for batch jobs so far [22]. Duplicate detection for ADR databases has a potential use-case for interactive and fast detection of duplicates for a specific report. Apache Spark [24] has the potential to support interactive data analytics. We implement a Spark based parallel system to support fast and scalable kNN classification for duplicates. Taking advantage of the distributed memory management and parallel data processing support offered by Spark, the system is able to handle large amount of ADR reports in a scalable manner.

The rest of this paper is organized as follows: Section 2 introduces the background technologies of our work; Section 3 defines the problem; Section 4 describes the system and our Fast kNN classification method; Section 5 gives the evaluation results of the performance of the system; Section 6 summarizes related works; and Section 7 concludes the paper.

2. BACKGROUND

2.1 kNN Classification

In a D-dimensional space $\mathcal{D}$, s and t are two vectors representing two data objects. We use $d(s,t)$ to denote the distance between s and t. Consider S and T are two sets of such vectors, for a vector $s \in S$, we denote its k nearest neighbors according to a given distance function in T as $knn(s,T,k)$. We assume that each $t \in T$ is associated with a label $L_t, L_t \in \{-1,+1\}$. The labels of vectors in S are unknown. We use $knn^+(s,T,k)$ to denote the vectors in $knn(s,T,k)$ with label "+1" and $knn^-(s,T,k)$ to denote vectors with label "−1". kNN classifier assign a label to s according to the following equation (note that the number of vectors in $knn(s,T,k)$ is an odd number):

$$L_s = \begin{cases} +1, \sum_{t \in knn^+(s,T,k)} L_t + \sum_{t \in knn^-(s,T,k)} L_t > 0 \\ -1, \sum_{t \in knn^+(s,T,k)} L_t + \sum_{t \in knn^-(s,T,k)} L_t < 0 \end{cases} \quad (1)$$

Assigning labels to each $s \in S$ according to their nearest neighbors requires a *kNN join* operation between S and T to identify k nearest neighbors of set S in T, denoted as $S \ltimes_{knn} T$.

$$S \ltimes_{knn} T = \{(s, knn(s,T,k) | \forall s \in S\} \quad (2)$$

2.2 Spark

Spark [24] is a cluster computing framework that supports iterative and interactive data processing. It provides a level of data abstraction called *resilient distributed datasets* (RDDs) [23] to represent a set of immutable data objects. These data objects can be partitioned among a number of cluster nodes. RDDs are fault-tolerant and can be reconstructed when their hosting nodes fail.

There are two types of operations that can be applied to a RDD in the Spark framework: *transformations* and *actions*. A transformation contains operations that produces a new RDD from an existing RDD while an action returns a value after operating on a RDD. Operations are often executed in parallel on a RDD. In addition to support *map*, *reduce* and *aggregate* operations on a RDD, Spark also offers operations such as *join*, *union* and *cartesian* to manipulate multiple RDDs. Different to MapReduce, RDDs where *map* and *reduce* operate on can be persistent in memory in nodes of the cluster, which greatly improves the efficiency of iterative and interactive applications. A duplicate detection system like the one discussed in this paper is an iterative process that contains data processing of multiple stages and it fits the Spark framework well.

3. THE PROBLEM

An adverse drug reaction report database $\mathcal{A}$ stores reports continuously collected by a regulator. We consider that a set of new reports, denoted by R arrive in the database may contain duplicates among themselves as well as with existing reports in the database. The problem is to identify the following set of report pairs:

$$Dupe(R,\mathcal{A}) = \{(r,h) | sim(r,h) < \epsilon, \forall r \in R, \forall h \in \mathcal{A} \cup R - r\} \quad (3)$$

in which, sim is a scoring function that measures the similarity between two reports and ϵ is a threshold that determines whether two reports are duplicate.

Duplicate detection within database $\mathcal{A}$ can been seen as a recursive process in which reports are sorted according to their arrival time to the database and reports with later arrival time are checked for duplication against those with earlier arrival time.

Note that even though an ADR database may contain 5% of reports that have at least one duplicate in the database, when it comes to the number of duplicated report pairs, the rate of duplicates is much lower. This is because the number of report pairs grows quadratically with the number of reports and non-duplicate report pairs grows much faster than duplicate report pairs. This results in highly imbalanced distribution of duplicate and non-duplicate report pairs in the dataset derived from $\mathcal{A}$.

4. THE SYSTEM

4.1 The Workflow

The workflow for duplicate detection in ADR database is shown in Figure 1. It contains the following major components:

- Report database: The report database stores reports collected by a regulator and new reports are continuously added to this database.
- Text processing module: Our system contains a text processing component to clean up text data in a report using common natural language processing techniques. Free text plays an increasingly important role in ADR reports as consumers increasingly participate in reporting drug side effects to regulators in recent years. Free text in a report not only is helpful for understanding drug side effects, but also contains useful information to identify duplicated reports. Compared to string fields such as name and address in existing duplicate detection systems, a free text field such as "report description" is significantly longer with majority of them being 250 and 300 characters long. This requires NLP techniques to extract useful information from the text for duplicate detection. On the other hand, regulators also increasingly monitor various data sources that contain large amount of text for ADR related information.
- Pairwise distance computing module: The module computes the pairwise distance among a set of reports including selected reports from the database and new reports arriving to the system.
- Training datasets (labelled datasets): The system maintains two temporary databases for duplicate detection: one contains report pairs that are known to be duplicate; the other contains samples of non-duplicate report pairs. The initial duplicate/non-duplicate labelling is done manually by domain experts from drug regulators, in our case, the TGA. Afterwords, the collection of report pairs in the two temporary databases are dynamically adjusted when new duplicates and non-duplicates are identified. Note that the duplicate report pair database stores all known duplicates while the non-duplicate report pair database only keeps a subset of known non-duplicates. The difference is due to the highly imbalanced distribution of duplicate and non-duplicate pairs.

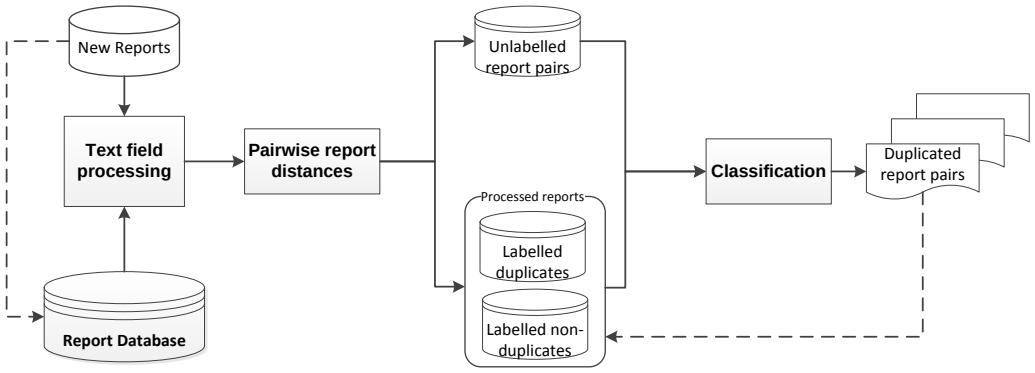


Figure 1: The workflow of the system – the dashed line represents that the source data becomes part of the target data when the processing finishes.

- Classification module: The report pairs are fed into the classification module that computes the scores for each pair and generates a list of duplicate pairs given a score threshold. Many classification algorithms fit into this system framework. We use kNN classifier as the default one for this application area. One advantage of kNN is that the classification results are easy to explain with human intuition and the basis of decision making can be justified clearly. This characteristics is particularly useful when the training dataset is highly imbalanced and global algorithms are difficult to separate them with general models.

4.2 Report Distance Calculation

A typical ADR report contains fields as shown in Table 2. Due to different missing data rates in different fields as well as schema inconsistency in data sources, common practices choose a subset of fields as input for duplicate detection. The fields used in our method are highlighted in bold fonts. The selection of fields is based on the WHO system as described in [18]. In the selected fields, *patient age* ("calculated age") is numerical type. *Patient sex*, *residential state* and *onset date* are treated as categorical data type. *ADR name* ("MedDRA PT code") and *drug name* ("generic name description") have string type. Different methods deal with string type differently, e.g., they are treated as categorical type in probability based methods and programming level string type in SVM based methods. As mentioned, free text becomes increasingly important in ADR reporting systems, we therefore include the *report description* field in our duplicate detection system and treat it as string type.

For a numerical field, if the values of two reports in the field is the same, the distance is 0, otherwise 1. The same calculation applies to categorical field types. For fields of string type, we use *Jaccard similarity coefficient* to measure the distance between two values as below:

$$d(S_1, S_2) = 1 - \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \quad (4)$$

in which, $|S|$ is the size of set S. The free text field is of string type, but as mentioned earlier, majority of values in the free text field are between 250 and 300 characters long. In order to eliminate the impact of typographical errors or different ways of constructing sentences, we apply common techniques to *tokenize* the content in the *report description* field, *remove stop words*, and then *stem tokenized words* to their root forms before computing their distances.

The distances of values in these selected fields of any two reports form a distance vector between the report pair. The comparison between two distance vectors, i.e., measuring how similar a report pair is in comparison to another pair of reports, is based on the Euclidean distance between the two distance vectors of the two pairs.

Information	Fields
Case Details	case number, report date
Patient Details	**calculated age**, **sex**, weight code, ethnicity code, **residential state**
Reaction Information	**onset date**, date of outcome, **reaction outcome code**, reaction outcome description, severity code, severity description, **report description**, treatment text, hospitalisation code, hospitalisation description, MedDRA Low Level Term (LLT) code, LLT name, **MedDRA Preferred Term (PT) code**, PT name
Medicine Information	suspect code, suspect description, trade name code,trade name text, trade name description, generic name code, **generic name description**, dosage amount, unit proportion code, dosage form code, dosage form description, route of administration code, route of administration description, dosage start date, dosage halt date
Reporter Details	reporter type, report type description

Table 2: Data fields of an ADR report in TGA data. The bold font indicates fields used in our duplicate detection method.

4.3 Fast kNN Classification

With the pairwise distances calculated, a kNN join is applied to labelled report pairs, denoted by T, and report pairs containing new reports, denoted by S. The classification of a report pair $s \in S$ is based on the score computed from its nearest neighbors in T. Due to the imbalanced distribution of positive and negative labels, the negative report pairs easily overwhelm the positive ones. We therefore normalize the score using the distance between two pairs as below.

$$score_s = \sum_{t \in knn^+(s,T,k)} \frac{1}{sim(s,t)} - \sum_{t \in knn^-(s,T,k)} \frac{1}{sim(s,t)} \tag{5}$$

The label of s is therefore determined by the following equation, in which θ is a given threshold:

$$L_s = \begin{cases} +1, score_s \geq \theta \\ -1, score_s < \theta \end{cases} \tag{6}$$

4.3.1 Parallelization Strategy

Consider the number of report pairs in T is n and the number of report pairs in S is m, the computing complexity of kNN classification is $O(m \cdot n)$ for join and $O(m \cdot k)$ for score calculation. n exhibits quadratic growth with the number of reports. The amount of data to process easily overwhelms a single server. To make the classification scalable, we partition T and S into a set of clusters, denoted by $\{T_1, T_2, ..., T_b\}$ and $\{S_1, S_2, ..., S_c\}$ respectively. The cluster size in a partition is adjusted to fit into the memory capacity of a computing node. A naive parallelization strategy is to apply kNN join for each partition group $\{(T_i, S_j) | 1 \leq i \leq b, 1 \leq j \leq c\}$ and then merge the nearest neighbors from each partition group. This approach does not reduce the overall computing complexity and incurs high data transfer cost as each partition in S needs to compare with all partitions in T. The merge of intermediate nearest neighbors may potentially become another bottleneck that limits the scalability.

To address this problem, we exploit the locality of report pairs in T in partitioning. We first partition report pairs using $k-means$ clustering to obtain c clusters. The center of each cluster is calculated and stored in memory. Note that clusters produced by $k-means$ form a *Voronoi diagram* where each report pair in a cluster is closer to the center of the cluster it belongs to than to any other cluster centers. We then assign each report pair $s \in S$ to a cluster whose center is the closest to s comparing to other cluster centers. It is likely that most of the k nearest neighbors of s are within the cluster it is assigned, i.e., most of report pairs in $knn(s,T,k)$ can be found in $knn(s,T_i,k)$ where T_i is the cluster to which s is assigned. Certainly, there are chances that some of the k nearest neighbors of s are in clusters sharing borders with the cluster. The two scenarios are illustrated in Fig. 2.

Our method therefore consists of two stages to deal with the two scenarios. In the first stage, we compute the k nearest neighbors within a partition to which each $s \in S$ is assigned. In the second stage, the cross-cluster comparison is performed for those testing report pairs falling into the second scenario in Fig. 2. The key technical challenge is how to determine whether it is necessary to check neighbor partitions when identifying the k nearest neighbors of a testing report pair.

4.3.2 Observations

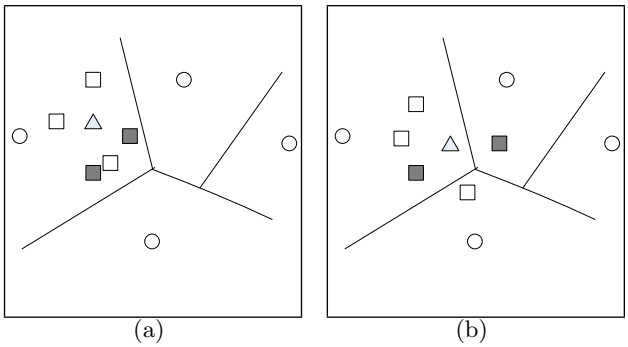


Figure 2: kNN under partitioned dataset: the circle represents the cluster center of a partition; the triangle represents a testing report pair s; the dark square represents a positive report pair in $knn(s,T,k)$ and the light square represents a negative report pair in $knn(s,T,k)$. (a) $knn(s,T,k)$ are all in one partition; (b) $knn(s,T,k)$ are in different partitions.

To address this problem, we develop an optimization algorithm that intends to prune unnecessary cross-cluster comparisons. The algorithm is based on the following observations:

1. The number of positively labelled report pairs is small and it incurs low computational cost to calculate distances between these positive report pairs and report pairs in testing dataset.
2. When the k nearest neighbors of $s \in S$ are all labelled negative, there is no ground to classify s as a duplicate report pair.
3. When the distance between s and its nearest positive neighbor is greater than that between s and the k-th nearest negative neighbor in a subset of T, it is clear that there is no positive report pair in the $knn(s,T,k)$.
4. As mentioned above, $k-means$ produces Voronoi partitions on the training report pairs, hence the hyperplane, denoted by h that separates two partitions is in the middle between the two cluster centers of the two partitions. If the distance between $s \in S$ and its k-th nearest neighbor, denoted by s_k in the partition to which it is assigned is less than its distance to the hyperplane, the distance between s and s_k, denoted by $d(s,s_k)$ is certainly less than distances from s to any report pairs in the other partition.

The scenario of observation 4 is shown in Fig. 3, where the distance between s and the partition hyperplane is $d(s,h)$, the distance between s and the closest report pair s_x in the other partition is $d(s,s_x)$. $d(s,h)$ can be derived according to [9] as below:

$$d(s,h) = \frac{d(s,p_j)^2 - d(s,p_i)^2}{2 \cdot d(p_i,p_j)} \tag{7}$$

in which, p_i and p_j denote the center of partition T_i and T_j respectively. As $d(s,b)$ is the shortest distance between s and the hyperplane b and connecting s and s_x needs to cross the hyperplane, we have $d(s,s_x) \geq d(s,b)$ according to

the triangle inequality. Therefore when $d(s, s_k) \leq d(s, b)$, it is not necessary to include the partition T_j with center C_j in the searching for $knn(s, T, k)$.

Algorithm 1 describes the steps for selecting additional partitions to be included kNN computing for a report pair s. Line 2 – 5 is based on observation 1 – 3 and line 6 – 12 is based on observation 4.

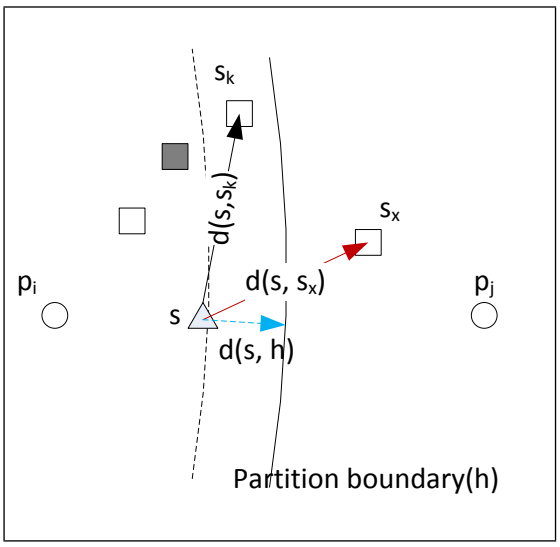


Figure 3: Additional partition selection for kNN under partitioned datasets: p_i and p_j represent two partition centers; the triangle represents a testing report pair s.

Algorithm 1 Additional Partition Selection

Require: $s \in S$
Require: $knn(s, T_i, k)$
Require: $min(s, T^+)$: the mininal distance between s and positive report pairs in T
Require: The centers of partitions : $\{p_j | 1 \leq j \leq b\}$
1: partitions = {}
2: $d(s, s_k) = \max(knn(s, T_i, k))$
3: **if** $d(s, s_k) \leq min(s, T^+)$ **then**
4: **return** partitions
5: **end if**
6: **for** $1 \leq j \leq b, j \neq i$ **do**
7: compute $d(s, h_{ij})$ using Equation 7. h_{ij} denotes the hyperplane separating T_i and T_j.
8: **if** $d(s, s_k) > d(s, h_{ij})$ **then**
9: $partitions = partitions \cup T_j$
10: **end if**
11: **end for**
12: **return** partitions

4.3.3 The Classification Algorithm

Algorithm 2 gives main steps of implementing the duplicate detection method described above with Spark primitives of transformations and actions. The stage 1 (intra-cluster comparison stage) is performed in line 6 – 8. The stage 2 (cross-cluster comparison stage) is performance in line 9 – 16. Stage 2 first computes the distances of the testing report pair to all positive training pairs. It skips cross-cluster comparison if the top k most similar report pairs obtained so far are all negative, indicating there will not be any positive training report pairs closer than the k-th nearest negative report pairs. Otherwise, cross-cluster comparisons are performed in line 12 – 15. The output score of each report pair is used to assign a label to the pair according to Equation 6.

Algorithm 2 Fast kNN Classification.

Require: A training dataset T containing report pairs labelled as duplicate or non-duplicate
Require: A testing dataset S containing unlabelled report pairs
Require: b – the number of clusters for partitioning T; c – the number of partitions for S
1: use $k-means$ to partition T into b clusters: $\{T_1, T_2, ..., T_b\}$ with cluster centers of these partitions denoted by $P = \{p_1, p_2, ..., p_b\}$
2: run *map* operation to compute distances between P and $s \in S$
3: assign s the partition i where $dist(s, p_i), (1 \leq i \leq b)$ is minimal for vectors in P.
4: randomly split S into c partitions : $\{S_1, S_2, ..., S_c\}$
5: **for** $i = 1$ to c **do**
6: run *join* operation on S_i and T^- based on cluster IDs (T^- denotes the negative report pairs in T)
7: run *map* operation to compute the similarity between joined report pairs
8: run *aggregate* operation to obtain the top k most similar report pairs for each $s \in S_i$ based on the output of the previous step
9: run *map* operation to compute distances between $s \in S_i$ and T^+ (T^+ denotes the subset of positive report pairs in T)
10: run *map* operation to combine the results from step 8 and step 9 to update the top k most similar reports pairs to s
11: **if** the output of step 10 contains at least one positive report pair **then**
12: run *map* operation on $\{(s, knn(s, T, k))\}$ for $s \in S_i$ to compute a set of additional partitions to compare using Algorithm 1
13: run *join* operation on S_i and additional partitions for $s \in S_i$ to compare
14: run *map* operation to compute the similarity between joined report pairs
15: run *union* and *reduce* operation to merge top k nearest neighbors in each partition for $s \in S_i$
16: **end if**
17: run *map* to calculate the score for $s \in S_i$ according to Equation 5
18: **end for**
19: **return** all $s \in S$ and corresponding scores

4.3.4 Further Pruning of the Testing Dataset

To further reducing the execution time of kNN classification, we can prune some report pairs from the testing dataset before applying the classifier. As shown in Equation 6, a pre-defined θ determines the label of a report pair. When the distance between a testing report pair and its positively labelled k-nearest neighbor is further than a threshold, the neighbor offers little hint to the label of the testing report

pair because of the low similarity between the two pairs. This observation can be used to prune the testing dataset.

When the testing dataset is large and the positive training pairs accumulate along time, it is necessary to speedup the computation of distances between each report pair in the testing dataset and each labelled positive report pair in the training dataset. We cluster the report pairs labelled as positive and use the cluster centers to determine whether a report pair in the testing dataset should be included in the classification. Assume the distance threshold between two report pairs is a function of θ, denoted as $f(\theta)$, The process is described as below:

Step 1. Cluster the positive report pairs into l clusters using k-means. We denote the cluster centers as $cp_i (0 < i < l)$;

Step 2. Compute the distance of the furthest report pair to its center in each cluster. We denote these distances as $dcp_i (0 < i < l)$;

Step 3. For each report pair t in the testing report pair set, do the following:

(a) For each $0 < i < l$, calculate $dist(t, cp_i)$;

(b) if any $dist(t, cp_i) \leq dcp_i + f(\theta)$, include t into the testing set;

Fig. 4 shows an example of the process in 2-dimensional space. cp_i and cp_j are the centers of two clusters of positive report pairs. For simplicity, we assume the positive report pairs are partitioned into only two clusters. The inner circles are formed using the distance between the furthest report pair and the center in each cluster. The shortest distance between the dashed circle and its corresponding inner circle is $f(\theta)$. In Fig. 4, p and q are testing report pairs. p is outside of both dashed circles and those positive report pairs are considered not helpful to decide the label of p. p is therefore pruned from the testing dataset. On the other hand, q is close enough to cp_i (within its dashed circle) and the positive report pairs in the cp_i cluster may carry useful information for labelling q. q is therefore included in the testing dataset for classification.

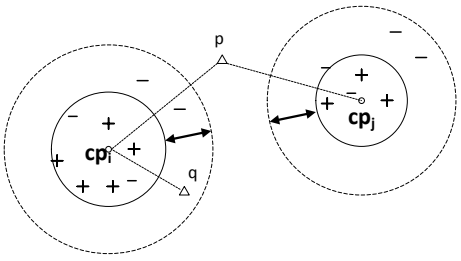


Figure 4: Pruning the testing dataset:"+" represents a positive report pair and "-" represents a negative report pair. The triangle represents a testing report pair.

5. EVALUATIONS

We implement the duplicate detection system in Java with Spark 1.2.1 API. We evaluate the performance of our system in a cluster consisting of 14 physical nodes. Each node has 2 x Intel Xeon E5-2660@2.20GHz CPU (8 cores) and 128GB physical RAM, in which 96GB is allocated to containers. The connection among nodes is via Infiniband networks. The OS in each node is Debian Wheezy. Cloudera CDH5 (5.0.0) with Hadoop 2.3.0 is installed with Yarn mode on. We run multiple *executors* on these nodes.

5.1 Datasets

We obtain ADR report data from TGA Australia. TGA maintains a database to store ADR reports submitted by various parties and collected by themselves. The sources that submit reports to the database include pharmaceutical companies, hospitals, general physicians, patients etc. TGA provides us 10,382 ADR reports they collected for a period of six months from July 2013 to December 2013. These reports consist 286 pairs of reports labelled as known duplicates. These duplicates were annotated by officers of TGA. Table 3 summarizes the dataset.

Report Period	1 Jul. 2013 - 31 Dec. 2013
Number of cases	10,382
Number of fields per report	37
Number of unique drugs	1,366
Number of unique ADRs	2,351
Known duplicate pairs	286

Table 3: Summary of TGA dataset.

The fields in each report in this dataset are listed in Table 2.

5.2 Fast KNN Performance

5.2.1 Baseline

Many classification methods are applicable to duplicate detection problem where distance vectors of report pairs are classified as similar or different. Support vector machine (SVM) is a popularly used one in duplicate detection [2, 20]. In our evaluation, we use a SVM classifier as the baseline for comparison.

SVM based methods take distance vectors between each pair of reports as input and map them into a high-dimensional space. These methods then use a hyperplane to separate distance vectors that represent duplicate report pairs and those representing non-duplicate report pairs. The hyperplane is obtained through learning from a training dataset containing labelled duplicates and non-duplicates. The hyperplane maximizes its margins to points belonging to the two different classes. With the hyperplane, new report pairs are considered duplicates if their distance vectors fall into the match side of the hyperplane with a large margin; otherwise, they are considered non-duplicates.

5.2.2 Precision and Recall

We measure the classification performance using the *area under precision and recall curve (AUPR)*. *Precision* and *recall* are defined as below in our case:

$$precision = \frac{\text{number of correctly identified duplicate pairs}}{\text{number of total identified duplicate pairs}}$$

$$recall = \frac{\text{number of correctly identified duplicate pairs}}{\text{number of total true duplicate pairs}}$$

AUPR shows how the precision values vary with different recall values. *AUPR* is able to visualize the difference of algorithms compared to other metrics and suitable for highly imbalanced datasets [4]. The goal to improve an algorithm with the precision-recall curve metric is to move the curve towards the upper-right corner.

Fig. 5 compare the performance of our Fast kNN algorithm and SVM. Fig. 5(a) and Fig. 5(b) show *AUPR* curves under different training dataset sizes. It is clear that in both cases, our algorithm significantly outperforms SVM based method. The main reason is that with highly imbalanced datasets, it is difficult to build a consistent model using SVM while large number of negative report pairs are surrounding few positive report pairs.

One way to improve the consistency of SVM classifier is to sample representative report pairs into the training dataset in hope that the model is applicable of a wide range of testing dataset. We implement an improved SVM classifier called *SVM clustering* by clustering training set and make sure report pairs in small clusters are included in the training dataset. Fig. 5(c) shows the actual area size varies with training dataset sizes under the three classification methods. It is easy to see that sampling a variety of report pairs into the training dataset does not have significant impact to SVM performance. Our method improves the classification performance by 19.1% in average in comparison to SVM classifier.

5.2.3 Effect of k

We also examine the impact of k on the classification performance and execution time. The results are shown in Fig. 6. We vary k from 5 to 21 and the variation of AUPR values is not significant, as shown in Fig. 6(a). This is due to that the score calculation takes the distance of a neighbor to the report pair being classified into account in Equation 5, which eliminates the impact of neighbors that are far away from the report pair to classify.

On the other hand, increasing k does increase the execution time of the Fast kNN classifier. As shown in Fig. 6(b), the execution time grows by 31% when k is increased from 5 to 21. This is mainly due to that a larger k potentially increases the number of partitions to compare.

5.2.4 Effect of cluster number b

The parallelism is affected by the number of clusters in the $k-means$ step in Algorithm 2, which determines the number of training dataset partitions as well as the number of partitions of joined testing and training datasets. Fig. 7 shows how the system performance is affected by the setting of the cluster number. Fig. 7(a) shows that as the number of clusters increases, the overall number of intra-cluster comparisons decreases in general, while the number of additional clusters to check in the next phase increases proportionally as shown in Fig. 7(b). The increase of cluster number results in smaller number of report reports in each cluster, which leads to the reduction of the total number of intra-cluster comparisons. However, the trend stops when the cluster number increases to 70 and the total number of intra-cluster comparisons slightly increases. This is due to the cluster sizes are uneven and the probability that a large portion of report pairs in the testing dataset is assigned to a large cluster increases. Therefore the overall comparison number increases.

Note, the total number of additional clusters to check increases in the second stage does not mean the total number of cross-cluster comparisons increases. The shrinking cluster size also reduces the number of comparisons in each cluster in stage 2. As shown in Fig. 7(c), the total number of cross-cluster report pair comparisons shows a decreasing trend as the number of cluster increases. Similar to intra-cluster comparison stage, the trend stops when the number of clusters increases to 70. It is also due to the uneven distribution of cluster sizes.

The computational complexity of the cross-cluster comparison stage is low compared to the intra-cluster comparison stage. As shown in Fig. 8(a), the total number of cross-cluster comparisons varies from 1.4% to 1.9% of the total number of intra-cluster comparisons. This also indicates that further reducing the number of cross-cluster comparisons is not able to have big impact on overall execution time.

The execution time change with different cluster numbers reflects the change of overall comparison number. Fig. 8(b) shows that the execution time has a decreasing trend when the number of clusters increases. When the cluster number is set to a number below 25, the memory of each executor can not accommodate joined partitions and frequent swapping triggers a few task failures due to timeout. The automatic retries significantly stretches the execution time. When the cluster number increases from 25 to 55, the execution time is reduced by 31%. When the cluster number becomes 70, the execution time slightly increases by 5.7% comparing to that when the cluster number is 55.

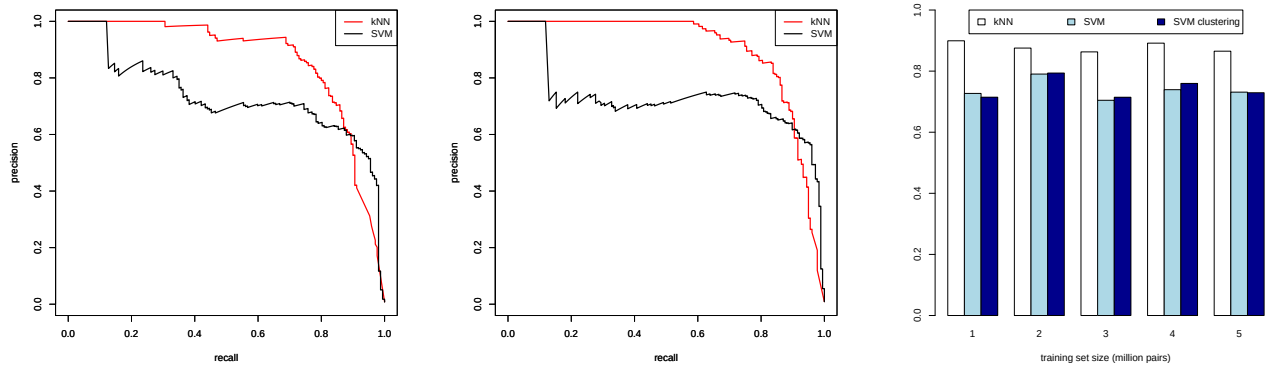
5.2.5 Scalability

We measure the scalability of Fast kNN from two aspects: firstly, we examine how it scales with the size of training dataset; secondly we investigate how it scales with the number of executors. As shown in Fig. 9, with different partition number of the testing dataset, the execution time increases proportionally with the increase of the training dataset size. The execution time increases 1.4 – 2.1 times when the size of training dataset increases 5 times.

Fig. 10(a) shows the execution time change with the number of executors. For different training dataset sizes, the increase of executor number leads to the decrease of execution time. The decrease trends become flatter as the executor number increases. This is due to that the overhead of data shuffle gradually increases while more nodes participate the computation. For comparison purpose, we show the pairwise distance computing time separately in Fig. 10(b). The input data for pairwise distance computing is relatively small and the data distribution cost is low in this step. As a result, its speedup is significant when the number of executors further increases. Fig. 10(a) and Fig. 10(b) also show that the time used in the pairwise distance computing step is only a small portion of the overall execution time.

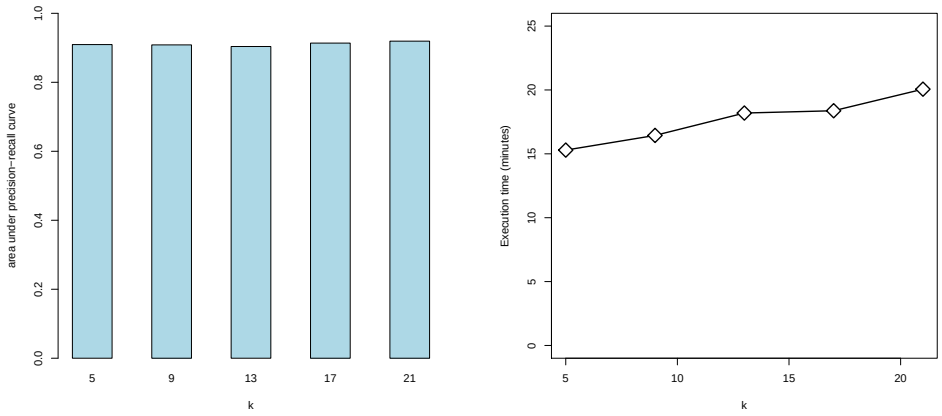
5.2.6 Effect of Testing Set Pruning

In the following, we measure the effectiveness of the testing set pruning method. We use $204,736$ randomly selected report pairs as testing dataset. The training data contains $1,000,000$ report pairs, in which 266 pairs are duplicate. Fig. 11 shows the result. When the distance threshold is set to 0.9, nearly 100% of the testing pairs are included in the classification phase. When the threshold is set to 0.7,



(a) (b) (c)

Figure 5: Comparison of area under precision and recall curve: kNN vs. SVM. Total number of testing pairs – 20,000. (a) Total number of training pairs – 5 millions; (b) Total number of training pairs – 1 millions; (c) Change of area sizes under precision and recall curves: the number of clusters in *SVM clustering* is set to 8.



(a) (b)

Figure 6: Effect of k: Total number of training pairs – 3 millions; Total number of testing pairs – 10,000. (a) Area under precision-recall curve (AUPR) comparison; (b) The execution time comparison.

about 75% of testing pairs are included in the classification phase. Setting the threshold to 0.5 does not produce significant pruning and 73% of testing pairs are included. When the threshold is set to 0.3, 65% of testing pairs remain. The pruning ratio is not exactly proportional to the threshold setting because of the non-uniform distribution of both positive training report pairs and testing report pairs. Note that all these threshold settings enable the duplicate report pairs in the testing dataset being included for classification.

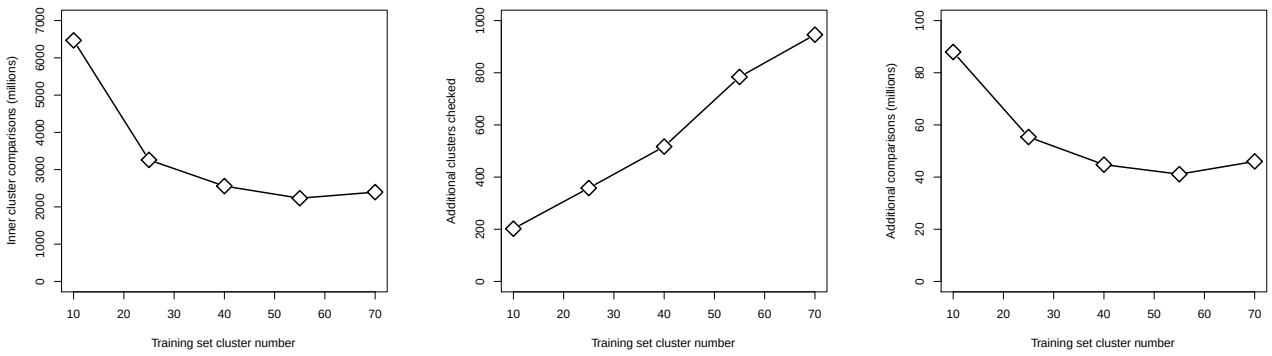
On the execution time of classification aspect, the reduction is significant. The threshold setting of 0.3, 0.5 and 0.7 reduces the execution time to 35%, 65% and 61% of the classification time without pruning. This is mainly due to the reduced data transfer and memory use. Even though setting the threshold to 0.5 slightly prunes more testing pairs than setting it to 0.7, the classification time under threshold 0.5 is slightly longer than that under threshold 0.7. It is related to the report pair distribution and how balance the workload of comparing these report pairs is among Spark data nodes that store them.

The setting of the threshold directly affects the performance improvement of testing set pruning. Potentially, the setting can be learned from the labelled data, which we leave as our future work.

6. RELATED WORK

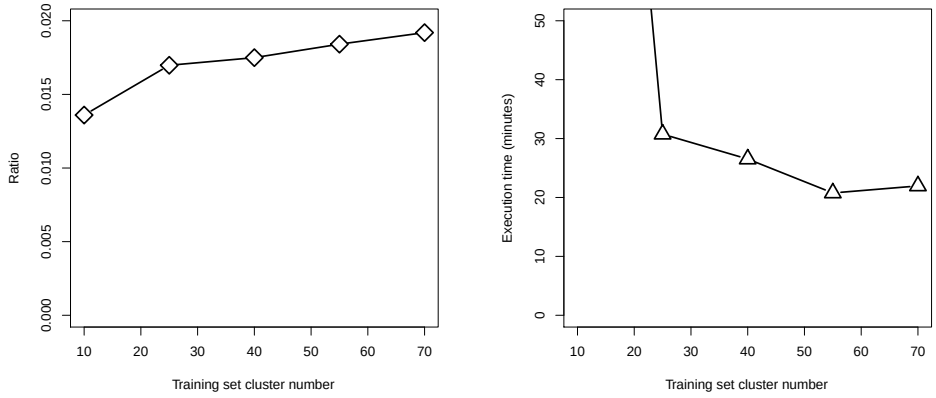
We compare our method with closely related works on parallelizing kNN join as well as approaches for handling datasets with imbalanced label distribution.

C. Zhang et.al [25] describes a basic Block based data partitioning method for kNN join using Hadoop. It then proposes to build an index using R-tree for each local block in a dataset. The built-in kNN functionality of R-tree is able to speedup the kNN search in the local block. The method does not reduce the overall computing and communication complexity.



(a) (b) (c)

Figure 7: Impact of the cluster number: Total number of training pairs – 4 millions; Total number of testing pairs – 10,000. (a) The number of intra-cluster comparisons; (b) The number of additional clusters to check for each element in the testing set; (c) The number of cross-cluster comparisons.



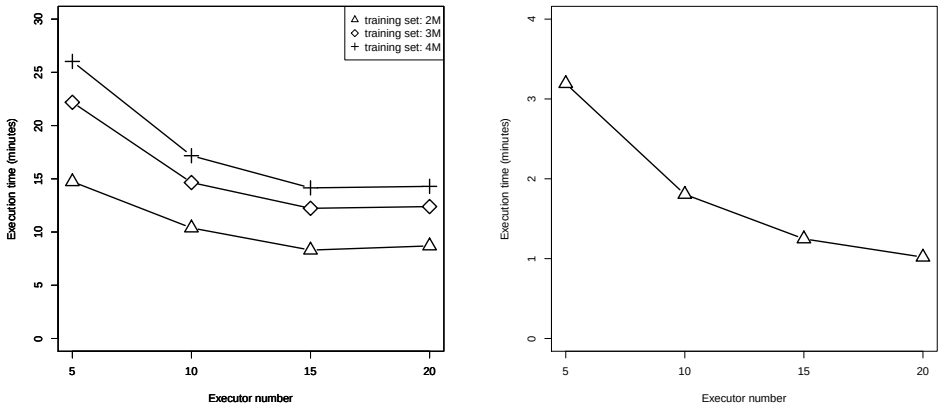
(a) (b)

Figure 8: Impact of the cluster number of training set on cross-cluster comparison: Total number of training pairs – 4 millions; Total number of testing pairs – 10,000. (a) Ratios of cross-cluster/intra-cluster comparison number; (b) The execution time change with the cluster number: memory size of each executor is 32GB.

W. Lu et.al [15] gives an improved algorithm for parallelizing kNN join using MapReduce and aims to reduce the search space. It partitions datasets into a Voronoi diagram. The differences between our approach and [15] lie in the following aspects: firstly, our method uses $k-means$ to partition the training dataset while [15] uses it as an option in data pre-processing to select pivots (partition centers). A *map* operation is then applied to use the selected pivots to partition datasets and collect partition statistics. As $k-means$ produces Voronoi sets already, it is not necessary to run another data partitioning operation. Secondly, our approach uses the characteristics of imbalanced datasets to reduce the cross-cluster comparison rather than introducing another statistics collection step to achieve this goal. Our experimental results show that the cross-cluster comparison cost is low. Thirdly, our approach makes use of distributed memory management of Spark to cache data partitions in comparison to the ad-hoc caching mechanism in [15].

In addition, there are works on set-similarity join using MapReduce [21] and Voronoi partitioning for MapReduce [1]. These works focus on identifying the nearest neighbors. Our work differ from them in using kNN as a means for classifying highly imbalanced datasets, which gives us additional information for reducing the search space, e.g., when the distance between a report pair and its nearest positively labelled neighbor is further than a given distance threshold, the report pair is likely to be non-duplicate and therefore pruned from the search space.

W. Liu and S. Chawla [14] illustrates the problem of imbalanced label distribution in classification and proposes a weighted method for handling imbalanced data in kNN classifier. Our results show that kNN classifier is more robust and less affected by over-represented negative data than SVM on detecting highly imbalanced duplicate/non-duplicate report pairs. Further improving the classification performance of kNN require careful analysis of similarity of report



(a) (b)

Figure 10: Execution time change with the number of executors: testing set size – 10,000; cluster number of the training set – 48; block number – 5; executor memory size – 32GB; executor core – 1. (a) Overall execution time; (b) Pairwise distance computing time (total number of reports – 10,382).

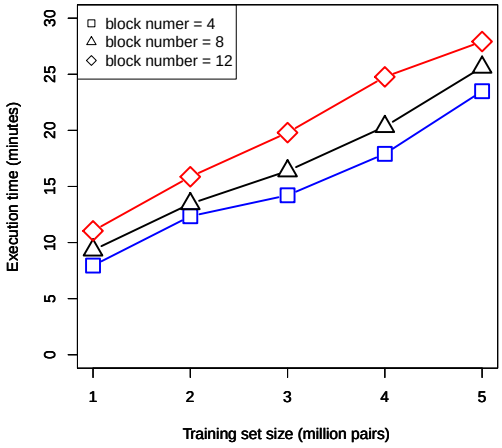


Figure 9: Scalability with the size of training dataset: testing set size – 10,000; cluster number of the training set – 32; total number of executors – 25 (32GB memory and 1 core).

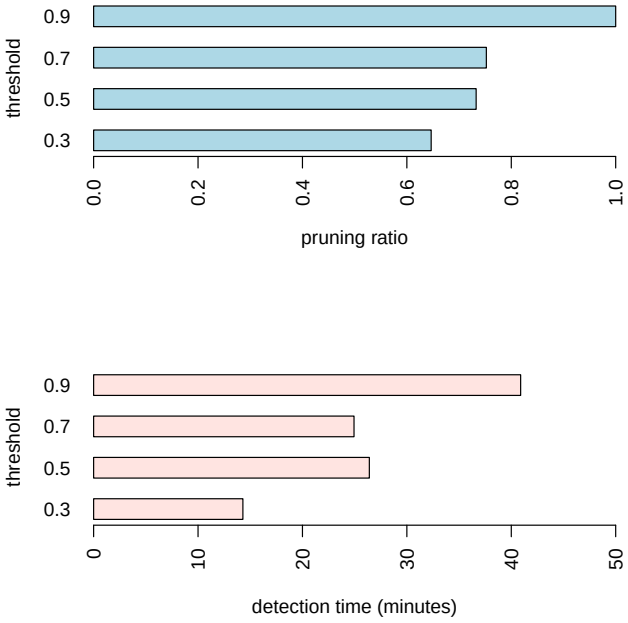


Figure 11: Effectiveness of pruning the testing dataset:training set size – 1,000,000; cluster number of the training set – 200; testing set size – 204,736; cluster number of the testing set – 30; $f(\theta)$ (threshold) is set to 0.3, 0.5, 0.7 and 0.9; executor number – 20; number of cores per execution – 4.

pairs by taking additional domain knowledge into account, which is our future work. Fortunately, the simplicity of kNN provides flexibility to accommodate new models.

7. CONCLUSIONS

In this paper, we studied duplicate detection in adverse drug reaction report databases. We proposed a Fast kNN classification method to deal with highly imbalanced label distribution in the dataset. Comparing to some datasets used for evaluating kNN join methods, there were relatively small number of fields chosen to calculate the pairwise distance vector between reports, however, the ADR report database contained a long text field with non-trivial distance calculation complexity. We showed that our method is effective in detecting duplicates in real ADR data and significantly outperforms SVM based classifier. The system we implemented using this method is capable of handling large amount of adverse drug reaction report data in a scalable way. We built the system using Spark. We effectively reduce the comput-

ing complexity by exploiting label imbalance and Voronoi partitioning of the training dataset. We also gave a method to prune the testing dataset to further improve the performance. We are not aware other parallel duplicate detection system in practical use in this domain with rapidly growing data and increasing importance. Our future work will focus on load balancing among executors for better scalability.

8. REFERENCES

[1] A. Akdogan, U. Demiryurek, F. Banaei-Kashani, and C. Shahabi. Voronoi-based geospatial query processing with mapreduce. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 9–16. IEEE, 2010.

[2] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 39–48. ACM, 2003.

[3] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings Of the 22nd International Conference On Data Engineering*, pages 5–17, Atlanta, GA, 2006.

[4] J. Davis and M. Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[5] M. Elfeky, V. Verykios, and A. Elmagarmid. TAILOR: A record linkage toolbox. In *Proceedings Of the 18th International Conference On Data Engineering*, pages 17–28, San Jose, CA, 2002.

[6] S. Evans, P. Waller, and S. Davis. Use of proportional reporting ratios (PRRs) for signal generation from spontaneous adverse drug reaction reports. *Pharmacoepidemiology and Drug Safety*, 10(6):483–486, 2001.

[7] D. Fram, J. Almenoff, and W. DuMouchel. Empirical Bayesian data mining for discovering patterns in post-marketing drug safety. In *PKDD*, pages 359–368, Washington, DC, 2003.

[8] R. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[9] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces (survey article). *ACM Trans. Database Syst.*, 28(4):517–580, Dec. 2003.

[10] B. Hug, C. Keohane, D. Seger, C. Yoon, and D. Bates. The costs of adverse drug events in community hospitals. *Joint Commission Journal On Quality and Patient Safety*, 38(3):120–126, 2012.

[11] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84(406):pp. 414–420, 1989.

[12] M. A. Jaro. Probabilistic linkage of large public health data files. *Statistics in medicine*, 14(5-7):491–498, 1995.

[13] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

[14] W. Liu and S. Chawla. Class confidence weighted knn algorithms for imbalanced data sets. In *Proceedings of the 15th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, PAKDD'11, pages 345–356, Berlin, Heidelberg, 2011. Springer-Verlag.

[15] W. Lu, Y. Shen, S. Chen, and B. C. Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *Proceedings of the VLDB Endowment*, 5(10):1016–1027, 2012.

[16] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records: Computers can be used to extract "follow-up" statistics of families from files of routine records. *Science*, 130(3381):954–959, 1959.

[17] J. Nkanza and W. Walop. Vaccine associated adverse event surveillance (VAAES) and quality assurance. *Drug Safety*, 27(12):951–952, 2004.

[18] G. N. Norén, R. Orre, and A. Bate. A hit-miss model for duplicate detection in the WHO drug safety database. In *KDD*, pages 459–468, Chicago, Illinois, 2005.

[19] E. Roughead and S. Semple. Medication safety in acute care in Australia: Where are we now? part 1: A review of the extent and causes of medication problems 2002-2008. *Australia and New Zealand Health Policy*, 6(1):18, 2009.

[20] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *The 8th ACM SIGKDD International Conference On Knowledge Discovery and Data Mining*, pages 269–278, Edmonton, Alberta, Canada, 2002.

[21] R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 495–506. ACM, 2010.

[22] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. Mccauley, M. Franklin, S. Shenker, and I. Stoica. Fast and interactive analytics over hadoop data with spark. *USENIX; login*, 37(4):45–51, 2012.

[23] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[24] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[25] C. Zhang, F. Li, and J. Jestes. Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 38–49. ACM, 2012.