Decision-making Bias in Instance Matching Model Selection

Mayank Kejriwal and Daniel P. Miranker

University of Texas at Austin
{kejriwal,miranker}@cs.utexas.edu

Abstract. Instance matching has emerged as an important problem in the Semantic Web, with machine learning methods proving especially effective. To enhance performance, task-specific knowledge is typically used to introduce bias in the model selection problem. Such biases tend to be exploited by practitioners in a piecemeal fashion. This paper introduces a framework where the model selection design process is represented as a factor graph. Nodes in this bipartite graphical model represent opportunities for explicitly introducing bias. The graph is first used to unify and visualize common biases in the design of existing instance matchers. As a direct application, we then use the graph to hypothesize about potential unexploited biases. The hypotheses are evaluated by training 1032 neural networks on three instance matching tasks on Microsoft Azure's cloud-based platform. An analysis over 25 GB of experimental data indicates that the proposed biases can improve efficiency by over 65% over a baseline configuration, with effectiveness improving by a smaller margin. The findings lead to a promising set of four recommendations that can be integrated into existing supervised instance matchers.

Keywords: Instance Matching, Model Selection, Decision-making Bias

1 Introduction

With its growing cross-domain collection of ontologies and instances, the Semantic Web has evolved into a diverse information space [21], [15]. Its growth has motivated researchers to investigate high-quality and automated solutions to the *instance matching* problem, which concerns identifying pairs of instances that refer to the same underlying entity [14].

Given their robust generalization properties, machine learning methods have come to dominate instance matching [17], [2], [19]. Once a machine learning model (e.g. a neural network) is trained on *labeled* data, *unseen* data is classified, and `:sameAs`-like links are forged between equivalent instances [21].

Designing a full instance matching system requires a practitioner to make decisions with respect to the *model selection* problem. For example, a practitioner must decide on a sampling strategy for acquiring labeled data, craft functions for converting raw labeled data into feature vectors, decide on a classifier and hyperparameter optimization strategy (for tuning the classifier), and partition

the labeled data into training and validation sets. In order to keep the model selection process *tractable*, it is necessary to base some of these decisions on *task-specific* knowledge. For example, certain features, such as phonetic and string-similarity features, are known to be especially effective for instance matching tasks that involve names and misspellings [4]. As another example, the real-world observation that class distribution in the instance matching problem often exhibits *data skew* (Section 3) influences *sampling* decisions (Section 4.1).

Given the challenging nature of instance matching in the Semantic Web, systems have become steadily more complex as practitioners have exploited task-specific knowledge as *piecemeal heuristics* to improve overall system performance [12], [17], [19]. These heuristics inevitably *bias* both the design and performance of any system that relies on them. For example, recent studies of *dataset bias* in the computer vision community show that systems that exhibit superior performance on one set of benchmarks may not necessarily be superior on a different dataset [20]. The reason was that, whether consciously or subconsciously, system designers used their knowledge of the task and the dataset to bias model selection decisions. Understanding such *decision-making biases* is a crucial step for subsequent research to reproduce and improve complex models, as well as to adapt them to novel situations without repeating the entire design process.

This paper attempts to achieve this goal by explicitly *modeling* decision-making bias in instance matching model selection as a bipartite undirected graphical model called a *factor graph* [10], with factor nodes representing bias opportunities. Several common decision-making biases in existing instance matching designs are explained and *visualized* by using this model. As a direct *application* of this visualization, we use the model to derive *new* opportunities for decision-making bias that, to the best of our knowledge, are not utilized by the majority of instance matchers. We empirically evaluate the proposed biases by training 1032 neural networks on Microsoft Azure's cloud-based platform using labeled data from three challenging instance matching benchmarks.

Evaluations on the test data lead to a set of four general recommendations that could potentially be used to improve existing supervised machine learning-based instance matchers both in terms of effectiveness and efficiency. Specifically, the analysis shows that (1) proportionate allocation stratified sampling [13] is a better sampling strategy for labeled data than a balanced (and traditionally more favored) approach, that (2) the training and validation sets should be as equal-sized as possible, that, (3) despite much lower efficiency, a hyperparameter optimizer based on grid search is no more effective than a random search conducted around reasonably set default hyperparameter values, and that, (4) under reasonable supervision assumptions, a setting that *favors* validation over training leads to run-time reductions of almost 70%, with a relatively smaller loss in effectiveness. Together, the last two recommendations are shown to lead to *efficiency* savings of over 65% with a small *increase* in *effectiveness* as well.

To enable repeatability, we provide screenshots of the employed experimental template, which may be run in a browser on a free MS Azure subscription. For

further analysis, all 25 GB of structured experimental data are exposed on a high-availability server via a public URL.

2 Related Work

In the general Artificial Intelligence community, *instance matching* is a 50-year old problem that continues to be actively researched, with a good survey of frameworks provided by Köpcke and Rahm [11]. Examples of some Semantic Web instance matching systems are Silk, RDF-AI and Limes [21], [18] [14]. Recent years have seen a proliferation of sophisticated machine learning approaches for improving instance matching performance, with Soru and Ngomo providing a comparative evaluation of various supervised classifiers [19], and Köpcke et al. providing a comparative evaluation of various competing systems that have emerged as popular choices for practitioners [12]. The latter work, in particular, showed that most systems only succeeded in certain settings, with hand-crafted features and with non-trivial amounts of training data [12]. In a similar manner, other systems have made expert-guided decisions on model and feature selection, an example being the random forest-based system of Rong et al. [17]. Instead of developing another instance matcher that competes with existing systems, the goal of this work is to model the myriad model selection decisions made by instance matching practitioners using a unified framework.

There are two important lines of prior research that come closest to this goal. The first line of research concerns knowledge-guided model *construction* in the context of *expert systems* [5]. In contrast, this work considers knowledge-guided model *selection* decisions in a machine learning-based instance matching context. A second, more recent line of research, attempts to unify various applications of *Statistical Relational Learning* (SRL) using Markov Logic [7]. This paper takes a complementary approach by restricting the application (e.g. instance matching) but not restricting the learning technique (e.g. SRL). Instead, we investigate and exploit the decision-making biases that go into the design of a generic machine learning-based instance matcher.

Much of the discourse on machine learning models in this paper is derived from classic material, Bishop's text being the primary reference [3]. Rojas' text is used for a more detailed discourse on neural networks [16]. Factor graphs, a special class of probabilistic graphical models central to the developments herein, are detailed in the text by Koller and Friedman [10].

3 Preliminaries

In the Semantic Web, link discovery is the problem of locating pairs of instances that satisfy a hidden *specification function* [14]. Without loss of generality, the specification function is often assumed to be that of *equivalence*, in which case the problem is referred to as *instance matching* [11]. Forging such *:sameAs*-like links between entities is important for maintaining connectivity in Linked Open Data per the *fourth* Linked Data principle [21].

Before the emergence of the RDF data model, it was often the case in the Relational Database literature that *schema matching* and *record linkage* tasks were considered orthogonal components of the broader data integration application [11]. The dominance of the RDF data model in the Semantic Web enables practitioners in the related sub-areas of ontology matching and instance matching to *cross-fertilize* their research, a manifestation of which is the annual Ontology Alignment Evaluation Initiative[1]. Although this paper primarily covers instance matching, an evaluation task in Section 5 also involves ontology matching.

An important issue that affects real-world instance matching problem instances is *data skew* [15]. Consider two RDF datasets G_A and G_B with respective sets of instances E_A and E_B. A naïve instance matcher attempts to classify the full Cartesian product space $E_A \times E_B$, a process that is time-prohibitive even for moderate datasets. Under reasonable assumptions, the number of true positives is $O(min(|E_A|, |E_B|))$ and is far outnumbered by the number of true negatives (a *quadratic* function) [4], [15]. One common technique that reduces this skew before further processing is *blocking* [15], [14]. A blocking algorithm clusters instances into blocks, based on a heuristic function. Instances sharing a block are paired and become candidates for further evaluation. Although the size of the candidate set is small relative to the Cartesian product, the skew is not completely eliminated and is still quite considerable (Section 5).

This paper assumes that the specification function is unknown but that it can be approximated through training a *machine learning classifier*. A typical machine learning-based instance matcher works as follows. First, a candidate set of instance pairs is generated through blocking, as described above [15]. Next, each pair is converted to a *feature vector* [4]. The choice of features is important, and guided by knowledge of the task [12]. Thus, it is a source of decision-making bias in the model selection design process, as described in the next section. A set of *labeled* feature vectors is split into *training* and *validation* sets and respectively used to train and tune the classifier, which is then used to label *unseen* instance pairs (the *test* data). The choice of classifier, the number of labeled samples on which the classifier is trained, the hyperparameter tuning strategy and the proportion of positively and negatively labeled samples are all issues that are crucial to the design and complexity of the final instance matcher. A framework for this decision-making process is subsequently presented.

4 Decision-making Bias in Model Selection

Before a machine learning classifier can be trained on labeled data, the *model selection* problem must be solved. For instance, a practitioner must craft the features that should be extracted from the data, and decide on hyperparameter optimization and sampling strategies. Usually, there are many choices at each step of the decision process, including default options (e.g. bag-of-words features for text representations [19], and random search for hyperparameter

[1] `http://oaei.ontologymatching.org/`

optimization [1]) derived from a survey of existing instance matchers. Empirical evidence indicates that this passive effort is typically insufficient, with non-trivial tasks demanding considerable model selection effort [12]. We also note that this decision-making process is not restricted to the *machine learning* component of model selection, but is an integral design component of any non-trivial system that *models* some phenomenon and is required to be empirically testable[2].

In this broader model selection process, *decision-making bias* arises because it is infeasible to consider all points in the design space, even when certain aspects of the model are fixed (e.g. restricting the instance matcher to only use machine learning). The decisions are typically justified through a variety of means, most notably early experimental findings or a study of existing systems [12], [4]. As briefly illustrated in Section 2, recent instance matchers have become steadily more complex in an effort to outperform the state-of-the-art. With added complexity, it becomes important to *model* the set of (possibly interlinked) decisions in order to reproduce (and improve upon) the system.

Figure 1 illustrates a possible framework for this process, namely a bipartite undirected graphical model, or a *factor graph* [10]. In the figure, oval nodes represent sets of objects, with shaded nodes representing sets provided to a practitioner *a priori*[3]. As in traditional factor graphs, the square nodes (labeled *Nodes 1-4*) represent points of *interaction* between their neighboring decision nodes [10]. Described below, these nodes can also be used to exercise prior task-specific knowledge to bias certain decisions towards a model selection outcome that is expected to be empirically favorable.

4.1 Node 1: Decision-making Bias in Sampling Strategy

In a supervised setting, data has to be collected and labeled in order to train (and *tune*) the machine learning classifier. Intuitively, the more labeled data is collected, the better the performance of the classifier. Labeling data can be a costly endeavor. The level of supervision, expressed as a percentage of the labeled instances to the total instances, depends directly on the allocated *labeling budget*. Once the level of supervision is determined (e.g. 50%), a practitioner has to pick an appropriate *sampling strategy*. With *simple random sampling*, an instance from the data pool is chosen (for labeling) with uniform probability till the budget is exhausted. Data skew can be problematic for this default strategy, since under moderate budgets, the probability of under-sampling true positives is high.

The risk of under-representing true positives in the labeled set can be mitigated by randomly sampling $q/2$ instances from each of the two classes, with q the total number of labelings allowed by the budget. This technique, which

[2] In the context of instance matching, for example, the choice of machine learning is itself a model selection decision, since it indicates that we *model* the *unknown* link specification function (see Section 3) using a trainable classifier.

[3] Thus, these nodes represent the fixed aspects of the model, or the *design constraints*, described in the previous paragraph.

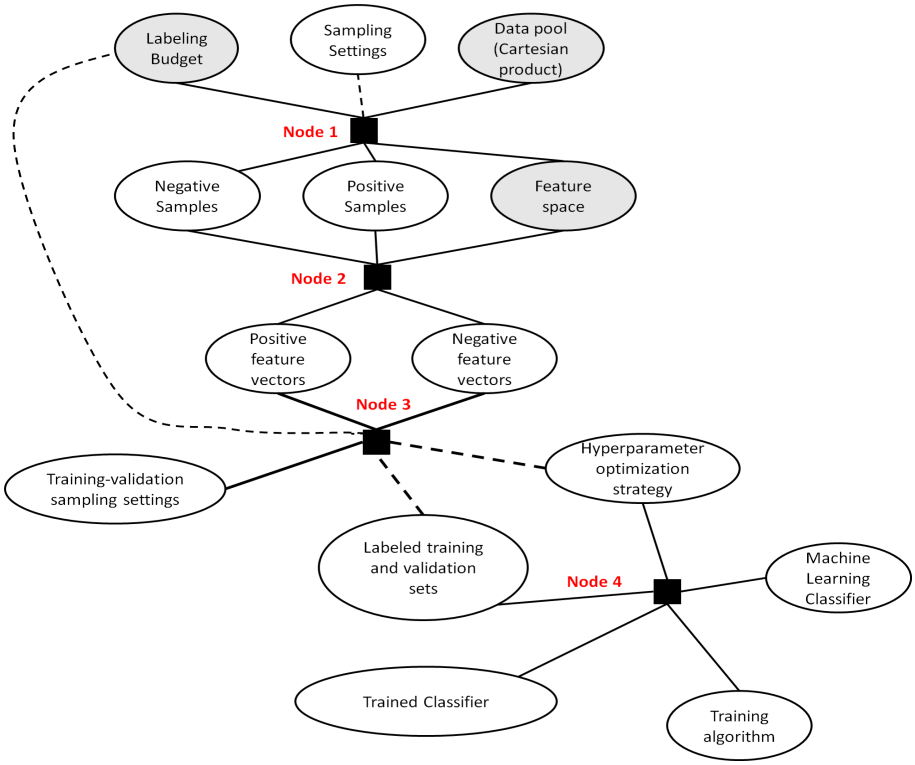

Fig. 1. A factor graph representing the model selection process. The oval nodes represent generic sets of objects (the concrete results of design decisions) while the square nodes represent opportunities for decision-making bias based on exploiting task-specific knowledge. Shaded nodes represent sets provided *a priori* (and are hence, *design constraints*) and the dashed lines indicate specific influences studied in this work.

is a variant of *stratified sampling*, was designed by statisticians to reduce the variance caused by simple random sampling in skewed datasets [13].

Existing stratified sampling strategies used in several instance matchers attempt an approximate *balancing* strategy in order to *eliminate* the skew from the labeled set [12]. This violates a key assumption of predominant machine learning theory, namely that the labeled data has (at least approximately) the same distribution as the unlabeled test data [3]. In keeping with machine learning norm, the authors hypothesize that skew should *not* be eliminated in the labeled set for good empirical performance on test data. In other words, the domain expert should use her knowledge of data skew to perform *proportionate allocation* stratified sampling [13]. For example, if the domain expert estimates, *a priori* or through experience and an educated guess, that 90% of the data pool is negatively labeled, she should use this estimate to sample $0.9q$ and $0.1q$ instances from the negative and positive pools respectively. We empirically com-

ment on this hypothesis in Section 5. In Figure 1, this influence is noted through the dashed line incident on Node 1. We note that decision-making bias arises in the choice of sampling strategy because of knowledge of the task. If, for some reason, the practitioner suspects that her data does not exhibit skew, the bias should be in the opposite direction. A third alternative is that nothing can be said about the data with reasonable probability. In this case, it is dangerous to introduce any bias at all in the choice of sampling strategy; a better approach is to model the data distribution through additional analysis.

4.2 Node 2: Decision-making Bias in Feature Crafting

In many instance matching tasks, special features can be devised (or chosen from a *global* feature space, as illustrated in Figure 1) to maximize performance. Traditionally, string similarity and token similarity features such as *Levenstein* and *tf-idf* have been popular; other practitioners have followed suit with phonetic and numeric functions as well [4], [17]. Crafting features for a machine learning problem using knowledge of the underlying task is by no means unique to instance matching, but also arises in other tasks such as speech recognition and computer vision [3]. Research on this issue in recent years have led to the emergence of deep learning techniques for automatically devising high-level features [8], but to the best of our knowledge, deep learning has not been applied to instance matching. Thus, decision-making bias in crafting features remains important in current instance matchers. For example, Soru and Ngomo favor token-based features in their evaluations [19], while Rong et al. devise special features individually for short text, descriptions and dates [17]. We do not study this bias further in this paper, but leave an extensive treatment for future work.

4.3 Node 3: Decision-making Bias in Training-validation Strategy

The labeled sets of feature vectors need to be split up into *training* and *validation* sets, which are both assumed to have the same proportion of positive and negative samples. A practitioner is typically expected to provide a sampling parameter that determines the ratio[4] of the size of the training set to the labeled set. In the literature, authors have experimented with several ratios, the most common being 50% and 90% (with the other 50% and 10% used for validation) [2], [19]. Once the sampling (and splitting) process is complete, a *hyperparameter optimization strategy* must also be chosen, in order to minimize chances of *overfitting* [3]. In principal, the validation set can be used to provide an unbiased estimate of classifier error [3]. Thus, the chosen optimizer makes hyperparameter assignments so as to maximize *validation* set performance [1].

Two extreme (and common) cases of hyperparameter optimization strategies are *random search* and *grid search* strategies [1]. In a random search, the optimizer randomly tests s different hyperparameter settings before selecting the best one, s being a user-defined parameter. A grid search, which can be

[4] If the training ratio is r, the validation ratio is automatically $1 - r$.

performed at arbitrarily fine levels of granularity, exhaustively searches through combinations of hyperparameter value assignments.

We argue that there is scope for introducing decision-making bias by exploiting information about the *level of supervision* (or indirectly, the labeling budget). This hypothesis is indicated by the dashed lines in Figure 1 incident on Node 3. The rationale behind the hypothesis is as follows. First, the training time for most machine learning models depends directly on the size of the training set [3]. Thus, if the level of supervision is low, training times are also expected to be low. Thus, expensive grid search can potentially compensate for the adverse effects of low supervision. The level of supervision can also be used to inform the training-validation ratio under the assumption that both parameter and hyperparameter optimization (controlled by the training and validation sets respectively) exhibit *diminishing* returns with more labeled data. The choice of ratio is important under efficiency considerations, as making the validation set larger allows the training time to be reduced, along with better hyperparameter optimization. In principle, this form of bias can be used for more efficient training, without significantly sacrificing performance (and possibly improving it), as empirically investigated in Section 5.

4.4 Node 4: Decision-making Bias in Classification

Node 4 offers additional potential for decision-making bias that involves choosing an appropriate machine learning classifier based on task-specific information. As one example, boosted multi-layer perceptrons were recently investigated for *minimally supervised* instance matching tasks (with extremely low labeling budgets) [9]. Similarly, logistic regression models were empirically demonstrated to be suitable for *noisy* instance matching tasks [19]. The takeaway is that, if the characteristics of the dataset are known, prior experience can be used to inform the choice of classifier. Conversely, if nothing is known about the data with reasonable certainty, a caveat similar to the one presented in Section 4.1 applies. A practitioner may be forced to investigate several classifiers and training algorithms (e.g. through pilot experiments) before selecting one.

4.5 Undirected vs. Directed Graphical Representation

There are two reasons why the model selection process in Figure 1 is framed as an *undirected* graphical model rather than a *directed* model (e.g. a *Bayes Network* [10]). First, decision making processes are typically *iterative* in real-world operational settings, and not necessarily *causal* as directed edges in a Bayes Network would seem to indicate. In a 'first pass', for example, a practitioner may choose default settings to get an initial feel for system performance on a specific instance matching task. If the task is inherently less challenging than presupposed[5], the default settings may be adopted with minor changes. For

[5] Two concrete examples are the OAEI benchmarks *Restaurants* and *Persons 1*, which have yielded 90%+ f-scores in several Semantic Web evaluations [19], [9].

more challenging cases, an iterative trial-and-error model selection process may be necessary for good performance.

A second reason for choosing a factor graph is its bipartite nature. Crucially, the graph allows us to distinguish between decision nodes and object nodes and captures the *mutual influence* exerted by various components of the model selection process on each other. The graph can also serve as a visual tool and be crafted in as much detail as warranted by the practitioner and the task. These advantages allow us to detect potential beneficial sources of decision-making bias (or lack thereof). The dashed edges in Figure 1 are examples of bias that the graph allowed us to detect and exploit. To the best of our knowledge, these biases have not been exploited in recent instance matchers. For example, we are unaware of any instance matcher that has explicitly used the labeling budget to inform training-validation ratios.

The factor graph may also be useful where several (possibly conflicting) sources of decision-making bias are involved in the design of the instance matcher. This scenario can occur because of differences in opinion (among collaborators) about dataset characteristics, different practical experiences or simply a lack of evidence. The factor graph is useful in this scenario because, by definition, factors are used to model probability distributions over neighboring object nodes. In principal, this is similar to the framework used by Domingos and Richardson whereby Markov logic was used to unify various applications of Statistical Relational Learning [7]. Although not explored here, the *probabilistic* study of decision-making bias is the natural next step for attempting to *explain* the bias and is left for future work.

4.6 The Cost of Decision-making Bias

Throughout this paper, the assumption is that introducing decision-making bias into the model selection process is *beneficial* to actual system performance, that is, reduces the *variance* of prediction. We note that the concept of both introducing and penalizing *statistical bias* in model selection to reduce variance is well established [3]. The *Occam's razor* principle essentially states that, all else equal, simple hypotheses are *inherently* more superior than complex hypotheses [3]. The *no free lunch* theorem of Wolpert and Macready disproves this superiority in the mathematical sense [22]; in this framework, simplicity itself is a form of decision-making bias that makes model selection tractable.

In the present context, this finding has an intuitive takeaway, namely that each decision-making bias introduced into the system implies a *cost*. The reason is that biases are introduced precisely because a practitioner has task-dependent knowledge; the more the system is tuned for the specific task, the less applicable it will be to other tasks. In the longer run, this may be undesirable if instance matching tasks of many different flavors are involved, and a single system is expected to service them over a long horizon. With respect to the current state of the research, this is also problematic; if researchers exhibited significant decision-making bias in their design, their system may not perform as expected beyond the datasets in their experimental design. The computer vision community has

Table 1. Test suite details. The second column only includes true negatives and positives in the *candidate set* (see Section 5.1). Mean sparsity (with resp. standard deviations in paranthesis) is the average percentage of features set to 0 in an arbitrary feature vector in the true negatives/positives in the candidate set. A *skew* of y% at x% (with x=10/50/90%) indicates that y% of the negative pool equals x% of the positive pool. Exact numbers may vary slightly due to rounding error

Dataset Name	**True Negatives/Positives**	**Mean Negatives/ Positives Sparsity**	**Skew at 10%/50%/90%**
Amazon-Google Products (AGP)	95,889/1300	67.64%/55.68% (7.66%/9.14%)	0.14%/0.68%/1.22%
Abt-Buy (AB)	40,917/1097	63.06%/58.00% (12.27%/12.20%)	0.27%/1.34%/2.41%
Film (F)	53,070/412	88.12%/80.79% (4.85%/6.96%)	0.078%/0.39%/0.70%

already begun addressing this issue as *dataset bias* [20]. Given the complexity of recent instance matchers in the Semantic Web (see Section 2), we believe that costs introduced by decision-making bias are worth investigating.

5 Experiments

5.1 Test Suite: Preparation and Statistics

The proposed biases are tested on three instance matching benchmarks. Two benchmarks, *Amazon-GoogleProducts* and *Abt-Buy* are public e-commerce datasets evaluated by Köpcke et al. on state-of-the-art approaches [12]. As the authors of that evaluation noted, these are difficult test cases in that the best supervised performance (in terms of f-scores; see Section 5.2) achieved on them was well below 70% [12]. The third test case concerns real-world *film* data from IMDB, but with artificial *semantic noise* injected into the data using an established Semantic Web generator [6]. The final test case involves both instance and ontology matching and has been used in OAEI evaluations.

We do not consider other OAEI benchmarks like *Persons* and *Restaurants* (see footnote 5) in these evaluations. The main reason is that even simple systems have performed well on these test cases, leaving little room for more improvement. A second reason is that the datasets used in our evaluations have also been used in at least three recent evaluations [12], [19], [9]. Because performance on them continues to be poor, they present interesting challenges for modern matchers. Finally, the most recent OAEI instance matching benchmarks do not have publicly accessible ground-truths at the time of writing.

As performing classification on all $O(n^2)$ instance pairs (with n being the number of instances) is infeasible, *blocking* techniques are used to heuristically generate a smaller *candidate set* of instance pairs that does *not* exclude the true positives [14], [4]. Instance pairs not in the candidate set are automatically classified as negatives. In this paper, we use a recently proposed unsupervised

method called *Attribute Clustering* to generate a high-performing candidate set [15]. Algorithm parameters are adjusted to ensure that all true positives in the ground-truth are included. Table 1 summarizes test suite details; note that the candidate set is still skewed, but less so (by over three orders of magnitude) than the space of all $O(n^2)$ pairs.

Once generated, all instance pairs in the candidate set are converted into binary feature vectors. Specifically, for every pair of matching attributes in every instance pair (or *properties* in RDF terminology), 28 features are generated, including phonetic, numeric and string comparison features. This feature set is used because of good performance in prior work[6] and shown to exhibit high performance [4]. In pilot experiments, the features were also found to work well in binary rather than real-valued form, possibly because of reduced classifier overfitting due to sparsity. As another advantage, sparsity is also known to lead to faster training convergence [16], [3]. Complete details on feature generation are provided on the project website (footnote 11).

5.2 Methodology

All experiments were run on Microsoft Azure's cloud-based machine learning studio[7] on a free preview subscription. At the time of experimentation, Microsoft was the only cloud vendor that offered a full suite of easy-to-deploy machine learning facilities. Since that time, some other vendors have released similar products, most notably Amazon. We leave repeating the experiments herein on alternate cloud vendors for future work. In all experiments, the classifier is fixed as a fully-connected, two-class neural network with a single hidden layer comprising 100 sigmoid units to guarantee sufficient expressivity [16]. The backpropagation algorithm is used for training the network, with five algorithm parameters[8] constituting the hyperparameters that need optimizing. Since the feature set is also fixed, the experimental goal is to test newly proposed biases.

To this end, four *categorical* variables are introduced to investigate the dashed influences in Figure 1, namely *Skew* ={*True, False*}, *Hyper* ={*Random Search, Grid Search*}, *Level-of-Supervision* ={*10%, 50%, 90%*}, *Training-ratio*= {*90%, 50%, 10%*}. *Skew* relates to Node 1 decision-making bias, with a *True* value indicating that complete data skew is maintained in the labeled set. A *False* value indicates instead that equal *numbers* are maintained (called the *balanced* approach; see Section 4.1). *Hyper* refers to the two hyperparameter optimization strategies employed in this work. Specifically, *Random Search* explores ten randomly chosen hyperparameter vectors in an attempt to improve upon the *default* hyperparameter settings in their *neighborhood*. *Grid Search* performs a full hyperparameter sweep at a pre-defined granularity[9]. *Level-of-Supervision* or

[6] This is an example of Node 2 decision-making bias (Section 4.2).

[7] Accessed at `https://studio.azureml.net/`.

[8] The learning rate, number of learning iterations, initial learning weights diameter, momentum and normalizer type.

[9] Based on the observed data, the MS Azure grid search setting divides the hyperparameter space into roughly 20-30 grid cells.

LoS refers to the quantity $|LabeledDataPool|/|DataPool|$; note that both negatively and positively labeled samples are included in the numerator, with *Skew* determining the relative proportion. *Training-ratio* is the ratio r in footnote 6.

For each test case and *joint* assignment to the four categorical variables, a model selection framework of the form in Figure 1 can be fully *instantiated*, with the end result being an instance matcher. This model is tested on the portion of the data pool that was not sampled and included in the labeled set. We measure instance matching performance using *precision* and *recall* metrics. Let the set of samples labeled as positives by the classifier be denoted as P, with $RP \subseteq P$ being the true positives retrieved by the classifier. Let TP denote the set of all true positives. The ratios $|RP|/|P|$ and $|RP|/|TP|$ respectively quantify precision and recall . Their harmonic mean, or *f-score*, illustrates their tradeoff and is a measure of system *effectiveness* [19].

To measure system *efficiency*, we record and add the run-times of both hyperparameter optimization and classifier training. The Microsoft Azure platform allows the experimenter to record fine-grained run-times of read and write times as well. These are not included herein as they are subject to data-center variance. Note that the actual optimization and classifier training takes place on a *single* node (attested to by the MS Azure documentation), and is not subject to such variance. Thus, the reported run-times are expected to be *low-variance proxies* for true model selection time.

Ten random trials are conducted for each dataset and joint categorical assignment, with the random number generator provided by Microsoft Azure. In total, almost[10] $3 \times 2 \times 2 \times 3 \times 3 \times 10 = 1080$ model selection experiments were conducted to investigate the biases. All 25 GB of experimental data, structured in directories and spreadsheets, have been made available on a high-availability server[11], together with screenshots of the experimental template that was used and all experimental data.

5.3 Results and Analysis

The subsequent analysis focuses is on the data tabulated in Table 2, with *Skew=True*. We comment on the *Skew=False* case, but due to space constraints, reproduce the full table for *Skew=False* only on the project website.

A cursory count of the bold (or *better*) f-score values in the two *Hyper* columns of Table 2 shows that, on 17/27 cases, *Grid Search* outperforms *Random Search* as a hyperparameter optimization strategy. This is expected (since *Grid Search* explores more hyperparameter space) and is closer to the empirical norm in the literature [12]. A more informed analysis indicates that *Random Search* also has its merits. The *mean difference* in f-scores between *Grid Search* and *Random Search* is 0.99%, which is not statistically significant from 0 at the 95%

[10] The actual number was 1032. In the final phase of the experimental runs, 48 trials on the *Film* dataset did not terminate due to subscription exhaustion.

[11] The project website is accessed at `https://sites.google.com/a/utexas.edu/mayank-kejriwal/projects/semantics-and-model-selection`

Table 2. Mean results (over 10 random trials/cell) on described metrics (Section 5.2) for the three test cases (Table 1) and with *Skew=True*. The bold lines separate (from top to bottom) results for *Training-ratio=90%, 50%* and *10%* respectively. Bold values indicate better performance across the two *Hyper* columns, and shaded cells indicate best values across the three *Training-ratio* segments

Test	LoS	Hyper=Random Search				Hyper=Grid Search			
		Precision	Recall	F-score	Run-time	Precision	Recall	F-score	Run-time
AGP	10%	**54.13%**	25.77%	34.89%	14.16s	48.62%	**33.08%**	**39.37%**	26.02s
AGP	50%	**61.51%**	28.77%	**39.20%**	1m 0.47s	54.76%	**29.23%**	38.11%	1m 50.70s
AGP	90%	**73.27%**	27.69%	40.22%	1m 31.93s	65.67%	**33.85%**	**44.67%**	3m 8.97s
AB	10%	70.00%	6.38%	11.70%	7.80s	**72.90%**	**7.90%**	**14.26%**	13.10s
AB	50%	**71.90%**	**20.07%**	**31.38%**	25.01s	63.10%	19.34%	29.61%	44.99s
AB	90%	**91.67%**	20.00%	32.84%	36.81s	85.19%	**20.91%**	**33.58%**	1m 18.35s
F	10%	68.46%	**57.48%**	63.99%	56.72s	**86.20%**	51.35%	**64.36%**	57.85s
F	50%	**83.59%**	**79.13%**	**81.30%**	2m 1.34s	83.42%	78.16%	80.70%	3m 53.09s
F	90%	74.07%	**97.56%**	**84.21%**	3m 29.42s	**76.09%**	85.37%	80.46%	6m 49.84s
AGP	10%	45.47%	**35.64%**	**39.96%**	10.16s	**49.46%**	31.28%	38.33%	17.88s
AGP	50%	55.50%	34.92%	42.87%	32.02s	**57.87%**	**35.08%**	**43.68%**	1m 8.56s
AGP	90%	**66.67%**	36.92%	47.53%	54.73s	62.79%	**41.54%**	**50.00%**	1m 57.03s
AB	10%	**52.10%**	16.31%	24.85%	6.31s	48.23%	**24.82%**	**32.78%**	22.72s
AB	50%	67.77%	14.96%	24.51%	14.93s	67.77%	14.96%	24.51%	31.64s
AB	90%	**76.32%**	26.36%	**39.19%**	23.00s	63.83%	**27.27%**	38.22%	45.24s
F	10%	79.93%	57.95%	67.19%	18.32s	**80.00%**	**62.53%**	**70.20%**	35.97s
F	50%	**85.85%**	85.44%	85.65%	1m 10.78s	85.51%	**85.92%**	**85.71%**	2m 24.32s
F	90%	73.59%	95.12%	82.98%	2m 3.22s	**74.07%**	**97.56%**	**84.21%**	6m 9.18s
AGP	10%	39.88%	**40.94%**	**40.41%**	5.71s	**44.93%**	34.44%	38.99%	8.49s
AGP	50%	**54.02%**	21.69%	30.96%	11.43s	48.06%	**28.62%**	**35.87%**	20.15s
AGP	90%	58.33%	32.31%	41.58%	16.95s	**64.18%**	**33.08%**	**43.66%**	32.02s
AB	10%	0%	0%	0%	4.84s	0%	0%	0%	5.68s
AB	50%	**63.72%**	**13.14%**	**21.79%**	6.92s	62.62%	12.23%	20.46%	10.46s
AB	90%	**68.97%**	18.18%	28.78%	8.85s	60.00%	**19.09%**	**28.97%**	14.86s
F	10%	**75.00%**	32.35%	45.20%	8.38s	74.47%	**32.86%**	**45.60%**	14.07s
F	50%	**84.08%**	64.08%	72.73%	22.57s	82.08%	**68.93%**	**74.93%**	44.91s
F	90%	77.55%	92.68%	84.44%	35.34s	**78.00%**	**95.12%**	**85.71%**	1m 14.33s

confidence level[12] (c.l.). When contrasted with the (statistically significant at the 99% c.l.) average percentage *increase* of 93.44% in run-time of *Grid Search* over *Random Search*, the latter is clearly a better choice.

In terms of run-time, two definitive observations can be derived from Table 2. First, when comparing a row across *LoS* settings, the run-time declines (as *LoS* declines), albeit not proportionally. This is again expected, since the back-

[12] Significance levels were tested using the Student's t-test for sample means.

propagation algorithm does not typically[13] run in linear time in the training set size [16]. Similarly, when comparing across table segments (different values of *Training-ratio*) run-time also declines. The latter observation is interesting because the same amount of labeled data is used for a fixed *LoS*; the change is merely in the relative proportion allocated to training versus validation.

Quantitatively, the average percentage *reduction* in run-time when comparing *Training-ratios* of 90% against 50% (with mean taken at all values of *LoS* and *Hyper* in Table 2) is 30.83%, which is statistically significant at the 99% c.l., while the average f-score *increases* by 4.08% (not statistically significant from 0 at the 95% c.l.). The difference is more dramatic when performing the same comparison but with *Hyper=Grid Search* at *Training-ratio=90%* and *Hyper=Random Search* at *Training-ratio=50%*. The average run-time for this case reduces by 66.70%, while the average f-score *increases* by 11.98%. Although the f-score increase is not significant (even at 95% c.l.), the run-time reduction is significant at the 99% level.To the best of our knowledge, the only (Relational Database) data matcher that has used a *Training-ratio* of 50% is MARLIN, which continues to outperform many systems [2], [12]. Systems can improve efficiency and (potentially) effectiveness by using this *combined* bias (that is, favoring *Hyper=Random Search* and *Training-ratio=50%* over alternate options).

As another example of how Figure 1 can favorably inform the model selection process, consider system performance at *LoS* values of *50%* and *90%*, and with *Training-ratio=10%*. While system performance at *Training-ratio=10%* degrades significantly[14] at *LoS=10%*, with AB achieving 0% f-score, the loss is less drastic at other *LoS* values. At *LoS=50%* and *90%*, the average percentage reductions in f-score (compared to the best f-score achieved by the other two *Training-ratio* settings) are 21.56% and 18.99%, with average run-times decreasing by 70.16% and 67.59% respectively. Although there is a cost[15] (in lost f-score performance) to using *Training-ratio=10%*, a practitioner constrained by *efficiency* (e.g. in *real-time* applications) should consider this bias.

The analysis is concluded with a brief note on the *Skew=False* setting. The broad conclusions noted above were also found to hold for this setting; a comparison *between* the two settings found that *Skew=False* performed much worse overall than *Skew=True*, with the average f-score reducing by 58.27% and 63.71% across the *Hyper=Random Search* and *Grid Search* settings respectively. These values are statistically significant at the 99% level. This confirms machine learning theory [3], in that the training set should be as *representative* of the full skewed distribution as possible. Traditional systems tended to (approximately) favor the balancing (i.e. *Skew=False*) heuristic [12].

[13] Empirically, that is. A closed-form formula is not known [16].

[14] The exception is AGP, which achieves its best f-score performance (40.41%) at *Training-ratio=90%* when *LoS=10%*.

[15] Even with only six sample (mean) data point comparisons for the two *LoS* settings described, the f-score reductions *are* significant at the 95% level but not the 99% level. Both run-time reductions are significant at the 99% level.

Standard deviations on all effectiveness metrics across each set of trial runs were observed to be very low (in many cases 0), an advantage of cloud-based experimentation in obtaining reliable estimates. Run-time standard deviations were also extremely low ($<<$ 5% on average). The last point justifies the chosen proxy for true model selection run-time in Section 5.2.

6 Summary and Future Work

This paper studies the application of decision-making bias to the instance matching model selection problem. First, the model selection design process is presented as a factor graph, with one class of nodes representing opportunities for bias. Existing decision-making biases in the instance matching literature are explicitly cast as special fragments of this model. These biases, and their mutual influences on each other, can then be visualized or further analyzed by a practitioner to understand the full extent of their design decisions.

As one form of analysis, we show that the model can be used to hypothesize about unexploited potential biases. Four specific recommendations were derived from the analysis. First, a practitioner should not artificially balance the labeled data but maintain skew through proportionate allocation stratified sampling (*Skew=True*). Secondly, good results are achieved, on average, when the training and validation ratios are equal (*Training-ratio=50%*). Third, the mean difference in effectiveness (i.e. f-scores) is not significant when a more expensive hyperparameter optimization strategy (*Hyper=Grid Search*) is preferred over simple random search in the neighborhood of default hyperparameter values (*Hyper=Random Search*), despite considerably increased run-times. Together, the last two prescriptions can be used to achieve a run-time reduction of over 65%, along with a slight increase in effectiveness.

Future work will study decision-making bias on a more theoretical foundation, and use the factor graphs for probabilistically reasoning about multiple sources of bias, as explained in Section 4.5. Also interesting is the issue of whether the proposed model can be applied to applications other than instance matching.

Acknowledgements

The authors thank Microsoft Research for providing infrastructure support. The authors were also supported by a US National Science Foundation grant.

References

1. J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.
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. C. M. Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.
4. P. Christen. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer, 2012.
5. W. J. Clancey. Model construction operators. *Artificial Intelligence*, 53(1):1–115, 1992.
6. E. Daskalaki. Instance matching benchmarks for linked data.
7. P. Domingos and M. Richardson. 1 markov logic: A unifying framework for statistical relational learning. *Statistical Relational Learning*, page 339, 2007.
8. G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.
9. M. Kejriwal and D. P. Miranker. Semi-supervised instance matching using boosted classifiers. In *The Semantic Web. Latest Advances and New Domains*, pages 388–402. Springer, 2015.
10. D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.
11. H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.
12. H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment*, 3(1-2):484–493, 2010.
13. J. Neyman. On the two different aspects of the representative method: the method of stratified sampling and the method of purposive selection. *Journal of the Royal Statistical Society*, pages 558–625, 1934.
14. A.-C. N. Ngomo. A time-efficient hybrid approach to link discovery. *Ontology Matching*, page 1, 2011.
15. G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 535–544. ACM, 2011.
16. R. Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 1996.
17. S. Rong, X. Niu, E. W. Xiang, H. Wang, Q. Yang, and Y. Yu. A machine learning approach for instance matching based on similarity metrics. In *The Semantic Web–ISWC 2012*, pages 460–475. Springer, 2012.
18. F. Scharffe, Y. Liu, and C. Zhou. Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. In *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)*, 2009.
19. T. Soru and A.-C. N. Ngomo. A comparison of supervised learning classifiers for link discovery. In *Proceedings of the 10th International Conference on Semantic Systems*, pages 41–44. ACM, 2014.
20. A. Torralba, A. Efros, et al. Unbiased look at dataset bias. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1521–1528. IEEE, 2011.
21. J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Discovering and maintaining links on the web of data. In *The Semantic Web-ISWC 2009*, pages 650–665. Springer, 2009.
22. D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.