An End-To-End Machine Learning Pipeline That Ensures Fairness Policies

Samiulla Shaikh
IBM Research
samiullas@in.ibm.com

Harit Vishwakarma
IBM Research
harivish@in.ibm.com

Sameep Mehta
IBM Research
sameepmehta@in.ibm.com

Kush R. Varshney
IBM Research
krvarshn@us.ibm.com

Karthikeyan Natesan Ramamurthy
IBM Research
knatesa@us.ibm.com

Dennis Wei
IBM Research
dwei@us.ibm.com

ABSTRACT

In consequential real-world applications, machine learning (ML) based systems are expected to provide fair and non-discriminatory decisions on candidates from groups defined by protected attributes such as gender and race. These expectations are set via policies or regulations governing data usage and decision criteria (sometimes explicitly calling out decisions by automated systems). Often, the data creator, the feature engineer, the author of the algorithm and the user of the results are different entities, making the task of ensuring fairness in an end-to-end ML pipeline challenging. Manually understanding the policies and ensuring fairness in opaque ML systems is time-consuming and error-prone, thus necessitating an end-to-end system that can: 1) understand policies written in natural language, 2) alert users to policy violations during data usage, and 3) log each activity performed using the data in an immutable storage so that policy compliance or violation can be proven later. We propose such a system to ensure that data owners and users are always in compliance with fairness policies.

Keywords

fairness, compliance, transparency, blockchain

1. INTRODUCTION

Today we see machine learning (ML) being applied in various domains and affecting people across all walks of life [18]. Application of ML algorithms in domains such as criminal justice, credit scoring, and hiring is promising, but at the same time, concerns of algorithmic fairness to individuals with certain protected traits are being raised [13]. Due to such considerations, there is a growing demand for fairness, accountability and transparency from ML systems [7]. ML systems rely heavily on training data and hence are prone to learn various biases present in the data. For example, multiple issues could arise if one builds a system to predict hiring decisions using attributes such as academic qualification, work experience, location, and gender. The system designer/developer may use some sensitive attributes to train the model, which is against policy (gender and location are sensitive attributes in this example) and the algorithm may learn various biases present in the data e.g. if the training data has a gender bias, the model may also learn this pattern. If such a model is deployed in the hiring process, then it may be unfair and the associated parties could be liable for prosecution.

Let us consider the well-known example of the COMPAS recidivism dataset, which contains the criminal history and personal information of offenders in the criminal justice system [13]. This type of data is used by the COMPAS risk assessment tool[1] for estimating the criminal recidivism (re-offending) risk of individuals. Recently, an offender named Loomis challenged the use of COMPAS risk assessment system for sentencing, claiming that it took away the defendant's right to due process and suspecting it of using gender to predict the risk.

The final decision in the Loomis lawsuit[2] can act as a policy document for appropriate use of algorithmic decision making in criminal sentencing. One important line in the ruling is as follows:

> "6 The court of appeals certified the specific question of whether the use of a COMPAS risk assessment at sentencing "violates a defendant's right to due process, either because the proprietary nature of COMPAS prevents defendants from challenging the COMPAS assessment's scientific validity, or because COMPAS assessments take gender into account."

The system we envision will automatically perform knowledge extraction and reasoning on such a document to identify the sensitive fields (gender in this case), and support testing for and prevention of biased algorithmic decision making against groups defined by those fields.

Checking for policy violations is a non-trivial task since it requires a machine to first understand policies written in natural language and then assess if they are being violated

[1] http://doc.wi.gov/about/doc-overview/office-of-the-secretary/office-of-reentry/compas-assessment-tool

[2] https://www.leagle.com/decision/inwico20160713i48

Bloomberg Data for Good Exchange Conference.
24-Sep-2017, New York City, NY, USA.

at any point in the processing pipeline, whether in data collection, feature transformation, algorithm development, or results interpretation. Although it is expected that the ML system will adhere to the policy guidelines, we also need it to be auditable in case of any violations that occur. Hence we need a system which will do the following:

- interpret policies written in natural language from policy documents,
- monitor data access and generate alerts if any access related violations occur either in the development or production setup of the ML system,
- log each activity performed by the ML system in an immutable storage so that it can be audited, and
- test the ML system for fairness policies.

Recent work has focused on developing learning algorithms which are fair [19, 11] and developing methods to assess whether an ML system is biased [1, 20, 3]. However, an overall system architecture that ensures adherence to the policies associated with the data is missing. In this paper we propose a system (framework) to realize the above objectives.

2. RELATED WORK

There is increasing interest and a growing body of literature on the topics of *fairness*, *accountability* and *transparency* in machine learning systems. Researchers are interested in developing methods to ensure fairness assuming the dataset is biased, and also in designing methods to audit any black-box predictive model for its adherence to fairness policies.

A new notion of unfairness, *disparate mistreatment* defined in terms of misclassification rates was introduced in [19]. A fair classifier is formulated by encoding this measure as additional constraints on the learning problem. In [11], the authors propose a method to certify disparate impact based on the predictability of the protected attributes from non-protected attributes. They also describe methods by which this bias can be removed from the data.

Adebayo and Kagal propose a method based on orthogonal projection of features to diagnose bias in a black box predictive model [1]. A measure that indicates whether a black-box classifier is biased against a group of samples is presented in [5]. In [3], the authors study the indirect influence that some features have on outcomes, through other related features in black box models. When the model is directly examined, the features that are indirectly influencing outcomes may not be used by the model at all. The degree to which the input features influence the outputs of the system are quantified in [10], to to explain the decisions made by the system. The predictive bias of a classifier is identified using a subset scan method in [20].

The techniques proposed in above works are independent approaches that either ensure fairness or audit the model for fairness. In contrast, our goal is to have an end-to-end system which can flag any fairness policy violations and provide support for auditing. Such a system is useful to large organizations in ensuring that their developers follow the policies associated with the data. In such a system, the methods we reviewed in the previous paragraphs, especially the ones built for auditing purposes, could be used as modules to check for fairness. We note that, to the best of our knowledge, a system we desire is not available. In the next sections we describe the proposed system architecture in detail.

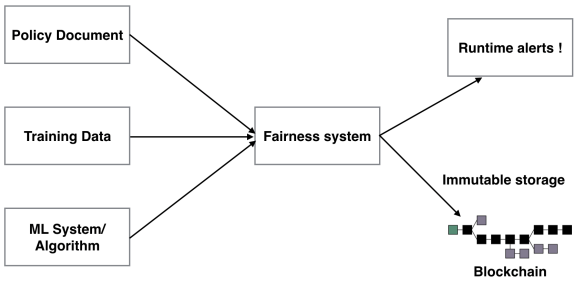

Figure 1: High-level system architecture.

3. SYSTEM ARCHITECTURE

The high level architecture of the proposed system is shown in Figure 1. The inputs to the system are:

1. ML algorithm or ML based system,
2. natural language policy document(s) associated with the domain or dataset used by the ML based system, and
3. ⟨optional⟩ ontology or schema associated with the dataset.

Given these inputs, the goals of the proposed system are:

- understand the natural language policies and store them in machine readable format,
- identify the set of policies that lie in the scope of our system (fairness and data usage policies will mainly be chosen),
- identify policy violations in run time by both passively and actively probing the running system, and
- log all activities in an immutable storage like Blockchain so that policy violations or compliance can be proved by any party (data owners, policymakers, data users).

3.1 Interpreting Natural Language Policies

Policy documents can be large domain-specific lists of sentences. The output of this step is a structured, machine-readable set of policies in XML or JSON format. If the schema associated with the data is given, the policies can

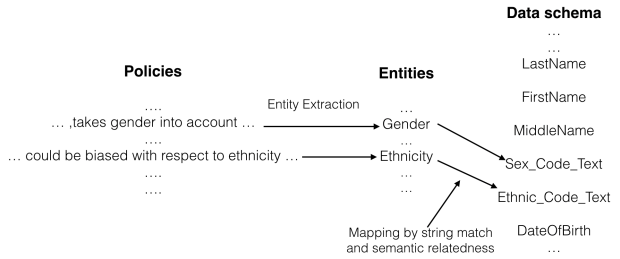

Figure 2: Entity resolution and mapping with the fields in a dataset.

be easily associated with the entities in the schema. The entities identified from the policy documents can be mapped to the entities in the schema. Figure 2 shows how this can be achieved when the schema of the data is known. In the example of the figure, the policy concerning *gender* is from Loomis and the policy concerning *ethnicity* is just a motivating example.

While structured data sources often have well-defined schemata, ML systems relying on plain text corpora tend to extract entities from the corpus as per their own semantics. These entities can be as simple as words, word counts, punctuation mark, etc., or as complex as person or organization having many simple or complex attributes. In ML systems using extractors like SystemT [9] where extractors explicitly define the ontology for the extracted and consumed entities, mapping the policies to the entities is still a feasible task.

The remaining case where the dataset is unstructured plain text and the ML algorithm is a black box with no machine readable specifications defining input entities or features is the toughest to handle. In such cases, there is still some hope if we have access to the documentation of the system. Here, we must map policies to upper ontologies like SUMO [16] or Wordnet [14] as shown in Figure 3. Afterwards, the extracted concepts (rather than the full policy documents) are manually inspected.

3.2 Representing Policies

The policies interpreted in the preceding step must be represented in a structured machine-readable format such as XML or JSON with defined schema. We find XACML (eXtensible Access Control Markup Language) [6] to be a strong and flexible representation that is the most suitable standard; we can extend the specifications to accommodate fairness-related directives in addition to access control. For our running example, Figure 4 shows how these policies can be stored in a machine readable format. We only display simple XMLs in the figure for convenience, but this is extensible to more complex schemata involving time-frames and locations associated with the policies.

3.3 Run Time Policy Check

Policy checking during run time is the primary task of our system. The work flow of this module varies greatly based on the type of the input and the transparency of the system. Simple policies like access-control are easier to check. For

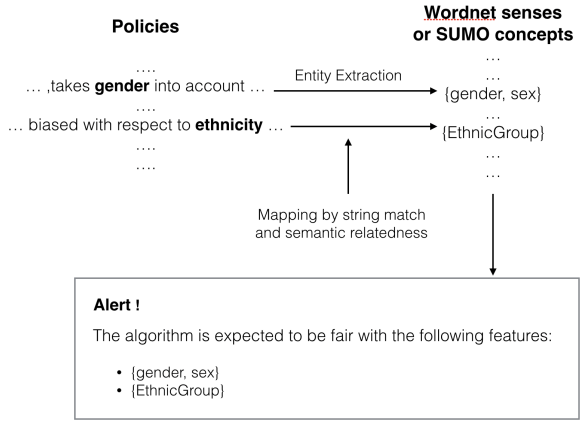

Figure 3: Alerting the user with the list of entities associated with the policies when no information about the system is available.

example, a policy stating *'ML systems should not use gender information from the dataset for decision making'* is much easier to detect than a policy stating *'Systems using this dataset should not be biased in decision making with respect to gender and race'*. In the later case, depending on the structure of the data source and the transparency of the system, multiple cases need to be considered.

In general, the term *transparent ML algorithms* is used for the ML algorithms whose behavior can be explained/interpreted by looking at the learned model [17]. Our expectation of transparency are only limited to input specification, which is sufficient for the context of policy check. Hence, we will use the term transparent-input algorithms. The internals of the algorithm may be opaque (how the algorithm uses the input data, how much importance is given to a particular field, etc.).

Case 1: Structured data and transparent-input algorithm

Here, the data is structured, i.e., organized with a well-defined schema. Additionally, the ML system defines the list of inputs that are consumed. The mapping between the fields of the data and the inputs of the system may or may not be available, but because of their structured nature, identifying the mapping automatically is very easy either by direct string matching or by identifying synonymous and semantically-related field names [15]. Once we have this mapping, we exactly know which input fields are being used by the system. With this information, we can quickly identify the policies associated with these fields.

Case 2: Structured data and opaque algorithm

Here, we do not have information about the subset of the fields that are being used by the ML algorithm. In such cases, the user of our system may specify the fields that are used by the algorithm, and the flow becomes similar to case 1. If not, we can safely assume that the system is using all the fields from the data and proceed accordingly.

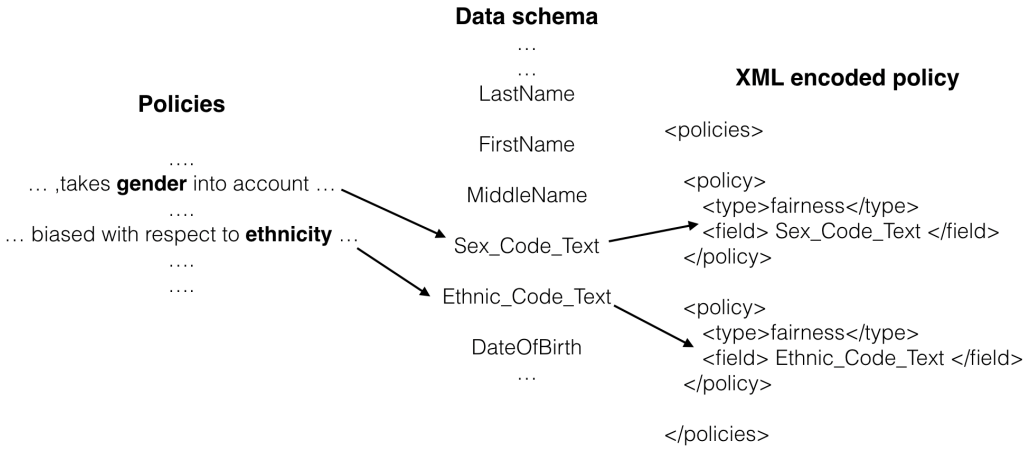

Figure 4: XML representation of the extracted policies

Case 3: Unstructured data and transparent-input algorithm

This is the typical case mostly represented by the ML systems associated with their own feature extraction logic. The unstructured data can be in any format like plain text, image, audio stream, etc. Since we have the input specification of the ML system, we will have to map the policies directly to the input specifications of the ML systems. Since the policies are written by the data owner, and the ML system is designed by a different person, the fields referred in the policies and those in the input specifications may not have too many exact string matches. Hence, we need to map the fields from policies and the input specifications to a common ontology, and create a mapping. In some cases, the names could be from different languages. Here, we will have to use cross-lingual linked concept hierarchies such as linked wordnets of multiple languages [8]. One good example for this case is an ML system consuming features extracted from plain text using extractors like SystemT [9]. Here, the extractor explicitly defines the ontology of the entities and relationships that are extracted from the input text.

Case 4: Unstructured data and opaque algorithm

This is the toughest case to handle automatically. Here, our system will expect some manual work from the user. The user of our system could be either the developer of the ML algorithm or the end-user of the ML algorithm. In both cases, the person who handles the data is responsible for ensuring that the policies are not violated. If the developer is creating the trained model, he or she is aware of the information being extracting from the unstructured data, and specify it accordingly in our system. If the developer is just shipping the algorithm to the end user without any trained model, and the end-user wants to train the algorithm on data with associated policies, the end-user must read the documentation or release notes written by the developers to understand the usage of the data by the algorithm. Based on this information, the end-user can specify the list of fields against which the policies can be checked.

In all the above cases, we ultimately identify the list of fields from the data that are consumed or extracted by the ML system and map them to the policies. Once this is done, the main task is to check the underlying ML system for fairness. There are existing frameworks like FairML [2], which can detect bias in the model for the known sensitive features. FairML does this by analyzing the deviation in the output when the sensitive features are perturbed. It also takes the correlation among the features into account while tweaking the input data. Here, FairML directly works with the actual underlying trained model, and curated input feature vectors. We are targeting a more general case, where we could have a black box system where inputs may not be encoded into vector format externally, and the internal vector representation of the features used by the algorithm may not be known. The tweakable features could be in any format like string, Boolean, number or complex type as per the system specification and the available dataset. Our system will work on one higher level of abstraction compared to FairML. Though, the principle will be similar to that of FairML in a way that we will change the sensitive feature to see the deviation in the output.

3.4 Immutable Auditing

In addition to policy interpretations and run time checks, the owners of the dataset may want to ensure that the dataset is not used for biased decision making. The users of such data sources can choose to train their algorithm using the dataset via our pipeline. Our system will log the information about the fields of the dataset that were accessed at the given time along with the report about the fairness with respect to the sensitive features extracted from the policies. Figure 5 shows how the immutable logging helps in auditing the policy compliance by the users. The immutable logs can be used to show that a particular instance of the trained model was fair with respect to the sensitive features mentioned in the policies. The other way in which these immutable logs can be used is the data owner tracking the usage and the compliance of the policies by different users.

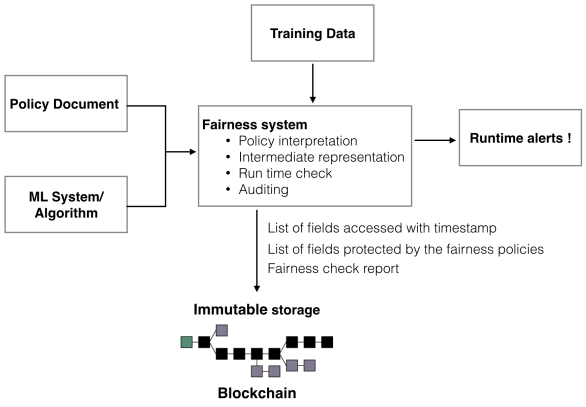

Figure 5: Immutable logging

4. CONCLUSION AND FUTURE WORK

As a first milestone, we have established a generic architecture for an end-to-end fairness pipeline for ML. The system will ensure quick fairness policy interpretations, reliable compliance checks and notifications of violations. This lays a foundation for an industry-wide standardization where data owners can use our system to specify policies directly into machine-readable format. This will make it easier for any party to be as clear as possible with respect to compliance.

Our fairness pipeline will act as a trusted authority between policymakers and users. The key goals that can be achieved by this architecture are twofold. Firstly, it will save lot of time on the user's side in terms of manual policy interpretation and assessment of their system. Secondly, the immutable publicly accessible logs can be used to prove policy compliance by the user.

5. REFERENCES

[1] J. Adebayo and L. Kagal. Iterative orthogonal feature projection for diagnosing bias in black-box models. volume abs/1611.04967, 2016.

[2] J. A. Adebayo et al. *FairML: ToolBox for diagnosing bias in predictive modeling*. PhD thesis, Massachusetts Institute of Technology, 2016.

[3] P. Adler, C. Falk, S. A. Friedler, G. Rybeck, C. Scheidegger, B. Smith, and S. Venkatasubramanian. Auditing black-box models for indirect influence. In *ICDM 2016*.

[4] P. Adler, C. Falk, S. A. Friedler, G. Rybeck, C. Scheidegger, B. Smith, and S. Venkatasubramanian. Auditing black-box models by obscuring features. *CoRR*, abs/1602.07043, 2016.

[5] Y. Alufaisan, M. Kantarcioglu, and Y. Zhou. Detecting discrimination in a black-box classifier. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 329–338, 2016.

[6] A. Anderson, A. Nadalin, B. Parducci, D. Engovatov, H. Lockhart, M. Kudo, P. Humenn, S. Godik, S. Anderson, S. Crocker, et al. extensible access control markup language (xacml) version 1.0. *OASIS*, 2003.

[7] S. Barocas, S. Friedler, M. Hardt, J. Kroll, S. Venkatasubramanian, and H. Wallach. Fairness, accountability, and transparency in machine learning, 2014.

[8] F. Bond and R. Foster. Linking and extending an open multilingual wordnet. In *ACL (1)*, pages 1352–1362, 2013.

[9] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. R. Reiss, and S. Vaithyanathan. Systemt: an algebraic approach to declarative information extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 128–137. Association for Computational Linguistics, 2010.

[10] A. Datta, S. Sen, and Y. Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 598–617, 2016.

[11] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian. Certifying and removing disparate impact. In *KDD 2015*.

[12] G. D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.

[13] L. K. J. Larson, S. Mattu and J. Angwin. Compas dataset, 2016.

[14] G. Miller and C. Fellbaum. Wordnet: An electronic lexical database, 1998.

[15] S. Patwardhan and T. Pedersen. Using wordnet-based context vectors to estimate the semantic relatedness of concepts. In *Proceedings of the eacl 2006 workshop making sense of sense-bringing computational linguistics and psycholinguistics together*, volume 1501, pages 1–8. Trento, 2006.

[16] A. Pease, I. Niles, and J. Li. The suggested upper merged ontology: A large ontology for the semantic web and its applications. In *Working notes of the AAAI-2002 workshop on ontologies and the semantic web*, volume 28, pages 7–10, 2002.

[17] G. Su, D. Wei, K. R. Varshney, and D. M. Malioutov. Learning sparse two-level boolean rules. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, Sept 2016.

[18] K. R. Varshney. Data science of the people, for the people, by the people: A viewpoint on an emerging dichotomy. In *Proc. Data for Good Exchange Conf.*, New York, NY, Sept. 2015.

[19] M. Zafar, Bilal, I. Valera, M. Gomez-Rodriguez, and K. P. Gummadi. Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment. In *WWW 2017*.

[20] Z. Zhang and D. B. Neill. Identifying significant predictive bias in classifiers. In *NIPS 2016*.