Grammar Based Directed Testing of Machine Learning Systems

Sakshi Udeshi, *Member, IEEE,* Sudipta Chattopadhyay, *Member, IEEE,*

Abstract—The massive progress of machine learning has seen its application over a variety of domains in the past decade. But how do we develop a systematic, scalable and modular strategy to validate machine-learning systems? We present, to the best of our knowledge, the first approach, which provides a systematic test framework for machine-learning systems that accepts grammar-based inputs. Our OGMA approach automatically discovers erroneous behaviours in classifiers and leverages these erroneous behaviours to improve the respective models. OGMA leverages inherent robustness properties present in any well trained machine-learning model to direct test generation and thus, implementing a scalable test generation methodology. To evaluate our OGMA approach, we have tested it on three real world natural language processing (NLP) classifiers. We have found thousands of erroneous behaviours in these systems. We also compare OGMA with a random test generation approach and observe that OGMA is more effective than such random test generation by up to 489%.

1 Introduction

In recent years, the application of machine-learning models has escalated to several application domains, including sensitive and safety-critical application domains such as the automotive industry [9], human resources [7] and education [8]. One of the key insight behind the usage of such models is to automate mundane and typically error-prone tasks of decision making. On the flip side, these machine-learning models are susceptible to erroneous behaviour, which may induce unpredictable scenarios, even costing human lives and causing financial damage. As an example, consider the following sentence that might be processed by an automated emergency response service:

> *"My house is on fire. Please send help in Sebastopol, CA. There is a huge forest fire approaching the town."*

While processing this text using a well trained text classifier model [3], it provides the following classification classes for the text:

```
'Hobbies and Interests', 'Science',
'Arts and Entertainment', 'Home and Garden',
'Religion and Spirituality'
```

It is needless to mention that the respective text classifier is unsuitable for categorising the emergency aspect underneath the text and therefore, is broken for the usage in emergency text classification. In short, systematic validation of machine-learning models is of critical importance before deploying them in any sensitive application domain.

In this paper, we broadly consider the problem of systematically testing the erroneous behaviours of arbitrary machine-learning models. Moreover, we consider these models are amenable only to text inputs conforming to certain grammars – a common feature across a variety of systems including models used in text classification. While the nature of erroneous behaviours in a machine-learning model depends on its input features, it is often challenging to formally characterise such behaviours. This is due to the inherent complexity of real world machine-learning models. To deal with such complexity, we leverage differential testing. Thus, instead of checking whether the output of a classifier is correct for a given input, we compare the output with the respective output of a different classifier realising the same problem. If the outputs from two classifiers are vastly dissimilar, then we call the respective input to be *erroneous*. The primary objective of this paper is to facilitate rapid discovery of erroneous inputs. Specifically, *given a pair of machine-learning models and a grammar encoding their inputs, our* OGMA *approach systematically searches the input space of the models and discovers inputs that highlight the erroneous behaviours.*

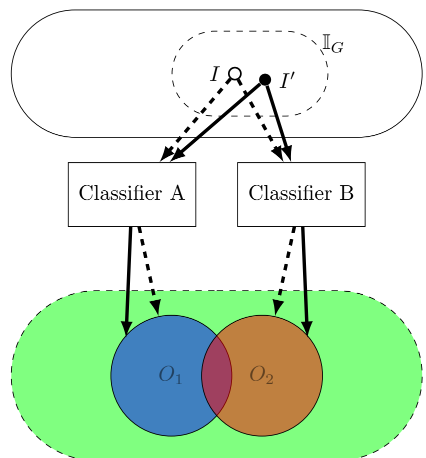


Fig. 1: Erroneous behaviour of Classifier A and/or Classifier B

As an example, consider the behaviours of *Classifier* A and *Classifier* B, which are targeted for the same classification job over an input domain conforming to grammar G (i.e. $\mathbb{I}_G$), in Figure 1. Despite being targeted for the same classification task, *Classifier* A and *Classifier* B generate largely dissimilar classification classes O_1 and O_2 for the same input I. The dissimilarity in outputs is indicative of one or both of the outputs being incorrect. Such erroneous behaviours in the classifiers might

• *S. Udeshi and S. Chattopadhyay are with Singapore University of Technology and Design.*

appear due to the outdated or inappropriate training data. We use our OGMA approach to automatically discover erroneous inputs such as I. Moreover, we can use these inputs to retrain and reduce the erroneous behaviours of the classifiers.

The directed strategy embodied within OGMA forms the crux of its scalability and effectiveness. Concretely, OGMA leverages the robustness property of common machine-learning models. According to the robustness property [13], the classification classes of two similar inputs do not vary substantially for well trained machine-learning models. As an example, consider the input I' in Figure 1, to be similar to input I. The classification classes for input I' will be similar to O_1 and O_2 for $Classifier\ A$ and $Classifier\ B$, respectively. In other words, if input I is an erroneous input, then input I' is likely to be erroneous too. To realise this *robustness property* for test generation, OGMA designs a perturbation function to continuously derive similar inputs to I and I' and thus, exploring the neighbourhood of erroneous inputs for a given classifier. Such a perturbation cannot simply be obtained by mutating a raw input, as the mutated input may not conform to the grammar. To this end, OGMA perturbs the derivation tree to explore the erroneous input subspace.

The grammar-based test input generation and the directed strategy make our OGMA approach generic in terms of testing arbitrary erroneous behaviours of machine-learning classifiers. In contrast to existing works that use concrete inputs from the training dataset to test machine-learning models [21], [28], our OGMA approach does not require training data for testing the models. Instead, we abstract the input space of the model via a grammar, which is a common strategy to encode an arbitrarily-large space of structured inputs. Thus, the tests generated by OGMA can explore these large input space, potentially discovering more errors when compared to limiting the test generation via the training data. In contrast to previous works, OGMA is not limited to test specific applications [17] or properties [14], [25]. OGMA works completely blackbox and can be easily adapted to test real-world classification systems for a variety of different applications. Finally, we show that the erroneous inputs generated by OGMA are useful and can be used for retraining the model under test and reducing erroneous behaviours.

The remainder of the paper is organised as follows. After providing the relevant background and an overview of OGMA approach in Section 3, we make the following contributions:

1) We present OGMA, a novel approach for systematically testing erroneous behaviours of arbitrary machine-learning models. The OGMA approach is based on a directed strategy to discover and explore the erroneous input subspace. Since the directed strategy embodied in OGMA is based on the fundamental robustness property of well-trained machine-learning models, we believe that OGMA can be adopted for testing arbitrary machine-learning models exhibiting robustness (Section 4).
2) We provide an implementation of OGMA in python. Our implementation and all experimental data are publicly available (Section 5).
3) We evaluate OGMA on three real-world text classifier service providers, namely Rosette [3], uClassify [4] and Aylien [1]. We show that our OGMA approach discovers up to 90% error inducing inputs (with respect to the total number of inputs generated) across a variety of grammars. We also show that the directed strategy in OGMA substantially outperforms (up to 489%) a strategy that randomly generates inputs conforming to the given grammar (Section 5).
4) We design and evaluate an experiment to show how the error inducing inputs generated by OGMA can be utilised to repair the test classifiers. We show that by retraining the test classifiers with the generated error inducing inputs, the erroneous behaviour can be reduced as much as 24%.

After discussing the related work (Section 6) and threats to validity (Section 7), we conclude and reflect in Section 8.

2 BACKGROUND

In this section, we introduce the relevant background and the key concepts based on which we design our OGMA approach.

Systems based on machine learning: In this paper, we are concerned about a machine learning model that accepts an input I and classifies it into one of the n classes from the set $\{C_1, C_2, C_3, \cdots, C_n\}$. Moreover, such an input I conforms to a grammar G, which encodes the set of all valid inputs for the model. Some classic examples of such models include deep-learning-based systems to categorise news items and systems that analyse the sentiments from Twitter feeds, among others. As of today, most machine-learning models are tested on their accuracy for well-defined sets of data. Such a strategy only validates a model on the available datasets. However, it lacks capability to systematically and automatically explore the input space accepted by the model and not captured by the available datasets. This is crucial, as inputs not captured by the available datasets may be presented to the model in a production setting and lead to catastrophic error, potentially costing human lives [5] [6]. In summary, a systematic validation of machine-learning model demands the machinery of automated software testing, a field that is largely unexplored in the light of testing machine-learning models.

Challenges in validating machine-learning-based systems: There exist multitudes of challenges in systematically validating machine-learning models. Consider an arbitrary machine-learning model M that accepts input I conforming to grammar G and classifies I in one of the category $\{C_1, C_2, C_3, \cdots, C_n\}$. Firstly, without precisely knowing the *correct* categorisation of input I, it is not possible to validate the model M. In other words, validation of machine-learning models faces the *oracle problem* [10] in software testing. Secondly, there has been significant effort in the software engineering research community to design directed test input generation strategies. The insight behind such directed strategies is to uncover bugs faster. For instance, to check the presence of crashes in C programs, a directed strategy may steer the test execution towards statements accessing pointers. Such directed strategies are well studied for deterministic software and their correctness properties. However, systematically steering the execution of a machine-learning model, in order to make its prediction dramatically wrong, is still immature. Finally, the error inducing inputs for a machine-learning model may not necessarily highlight a bug in the respective code (unlike classic software debugging process). Instead error inputs may highlight flaws in the data on which the respective machine-learning algorithm was trained to obtain the model under test. Therefore, the systematic usage of the error inducing inputs, to debug the machine-learning model, is also of critical importance.

Differential testing: To solve the oracle problem in testing machine-learning models, we leverage differential testing. Specifically, consider two models M_1 and M_2 that expect valid inputs conforming to the same grammar G and classifies each input from the same set of categories $\{C_1, C_2, C_3, \cdots, C_n\}$. For an input I conforming to G, if the prediction of M_1 and M_2 are drastically different, then we conclude that I is an error inducing input for at least one of M_1 and M_2. In Section 4, we formally define the criteria for identifying such an error inducing input. Although our testing strategy requires two models from the same problem domain, we believe this is practical, given the presence of a large class of machine-learning models targeting real-world problems. Moreover, our proposed strategy can also be useful to discover regression bugs via comparing the outputs from two different versions (e.g. a stable version and a developing version) of the same machine-learning model.

Robustness in machine learning: The insight behind the directed testing in OGMA is based on the *robustness* of common machine-learning models. Conceptually, robustness in machine-learning captures a phenomenon stating that a slight change in the input does not change the output dramatically in well-trained machine-learning models [13]. This means that error inducing inputs are likely to be clustered together in the input space of well-trained models. Technically, assume a model f, and let I be an input to f and δ be a small value. If f is robust, then $f(I) \approx f(I \boxtimes \delta)$, where $I \boxtimes \delta$ captures an input obtained via small δ perturbation of input I. In such case, we say that input $I \boxtimes \delta$ is in the *neighbourhood* of input I. Since $f(I) \approx f(I \boxtimes \delta)$, we hypothesise that if an input I causes an error, then it is likely that input $I \boxtimes \delta$ will cause an error too. This hypothesis forms the crux of our directed testing methodology.

State-of-the-art in testing machine-learning-based systems: Adversarial testing [11] techniques have the objective to fool a machine-learning model with minute perturbation on inputs and guiding the model towards a dramatically wrong prediction. However, such testing strategies are only limited to minimal and unobservable input perturbations and require a set of seed inputs. Therefore, adversarial techniques are neither sufficient nor general enough to check the erroneous behaviour of machine-learning models. Besides, adversarial testing does not solve the test design problem in the broadest sense due to their dependency on a set of seed inputs and due to their incapability to discover faults that may only appear with observable differences across inputs. Finally, if the processed data by the machine-learning model requires security clearance (e.g. healthcare data, finance data), then we need a systematic process to generate these inputs during the automated validation stage of the model.

In recent years, the software engineering research community have stepped up to develop testing methodologies for deep-learning systems [14], [17], [21], [24]. These works, however are, limited either to specific applications [17] or rely on the presence of sample inputs [21], [28]. Moreover, none of the prior works are applicable to generate grammar-based inputs in a fashion that such inputs steer the execution of machine-learning models to erroneous behaviour. In the subsequent sections, we will discuss the key ingredients of our OGMA approach that accomplishes this objective.

3 OVERVIEW OF OGMA

In this section, we will outline the working principle of OGMA via simple examples.

Consider a context free grammar as shown in Figure 3. We assume that the sentences generated from this grammar are used as inputs to machine-learning-based systems, such as classifiers. These classifiers may be used to identify a sentence into a specific category, such as *hobby*, *sports* and so on.

Differential Testing: Our OGMA approach starts with an initial input I. Such an initial input is randomly generated from the grammar. Let us assume that the initial input is "*Mary saw my dog*". To check whether this input leads to any classification error, we feed it into two text classifiers. Usually the real-world text classifiers, as used in our evaluation, return a set of classification classes. Let us assume $\mathbb{C}_1$ and $\mathbb{C}_2$ are the set of classification classes returned by the two classifiers M_1 and M_2, respectively. To check whether the initial input lead to a classification error, we evaluate the Jaccard Index $\frac{|\mathbb{C}_1 \cap \mathbb{C}_2|}{|\mathbb{C}_1 \cup \mathbb{C}_2|}$. If the Jaccard Index between $\mathbb{C}_1$ and $\mathbb{C}_2$ is below a user-defined threshold J, then we conclude that either M_1 or M_2 exhibits a classification error. We note that the threshold J controls the error condition. For instance, a very low value for J will enforce a strong condition on identifying a classification error.

Directed Testing: One of the key challenge for testing machine-learning models is to systematically direct the test generation process. This is to discover erroneous behaviours as fast as possible. While directed test generation is well studied for deterministic software systems, a similar development is limited in the case of machine-learning systems. In this paper, we leverage the robustness of well trained machine learning models to design a directed test generation method. According to robustness, similar inputs and similar outputs are clustered together for such machine-learning models [13]. Thus, we hypothesise that error inducing inputs are also clustered together. However, as our OGMA approach targets inputs conforming to certain grammars, it is not straightforward to define the neighbourhood of an input that are likely to be classified in a similar fashion. Specifically, we need to explore the following for a directed test generation:

1) *The grammar under test:* We should be able to generate a substantial number of inputs that are derived similarly (e.g. by applying similar sequence of production rules) from the grammar. This is to facilitate exploring the neighbourhood of an input conforming to the grammar and thus, exploiting the robustness property of the machine-learning model under test. As observed from the grammar introduced earlier in this section, it does encode several inputs that are derived via similar sequence of production rules.
2) *The distance between inputs conforming to a given grammar:* We need an artifact to formally define and explore the neighbourhood of an arbitrary input conforming to a grammar. To this end, we chose the *derivation tree* of an input generated from the grammar. We consider two different inputs I and I' (both conforming to the grammar) in the same neighbourhood if I and I' have the same derivation tree, but the exception of a terminal symbol. For instance, the input sentences *Mary saw my dog* and *Bob saw my dog* are in the same neighbourhood, as they differ only in the production rules $NP \rightarrow Mary$

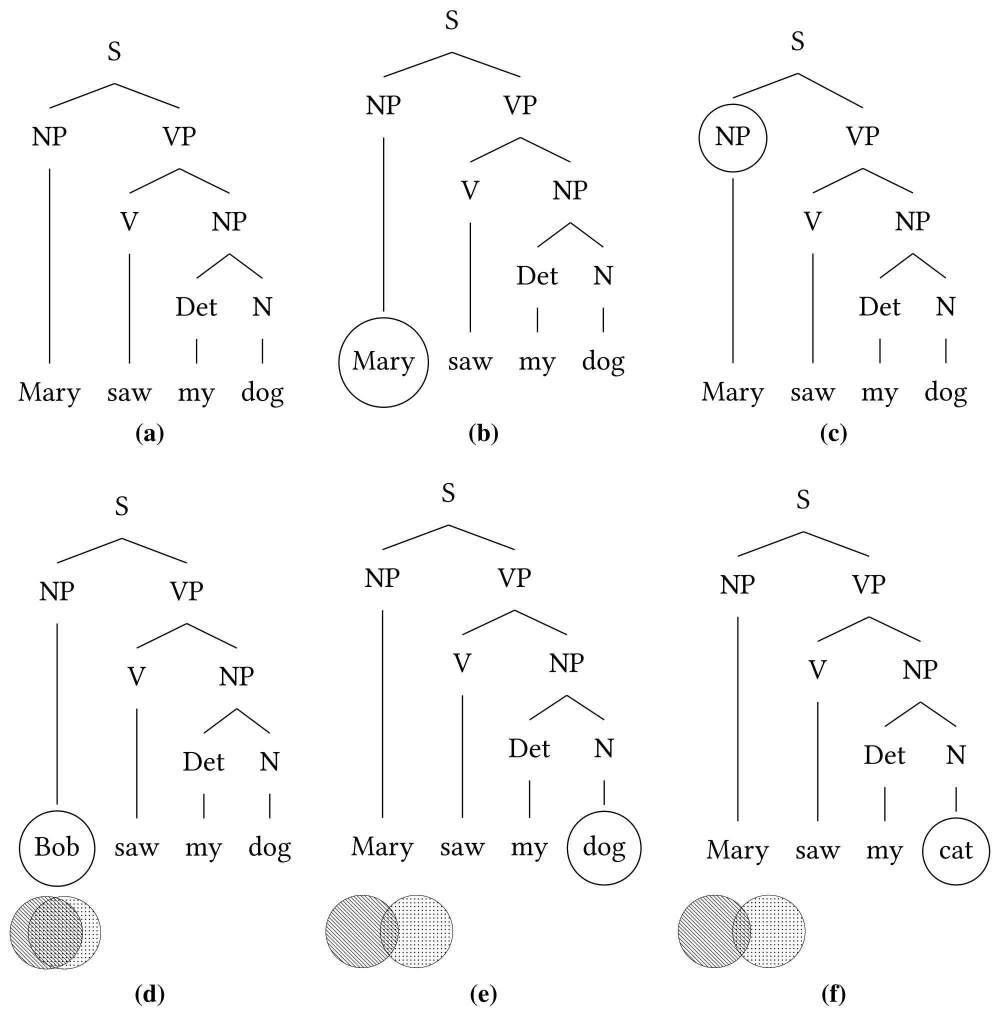


Fig. 2: Different stages of OGMA: (a) Derivation tree of the sentence "*Mary saw my dog*", (b) to perturb the sentence slightly, OGMA first chooses a terminal symbol at random, in this case, the word "*Mary*", (c) OGMA discovers the production rule generating the word "*Mary*", in this case, the rule: *NP* $\rightarrow$ *"John"* | *"Mary"* | *"Bob"* | *Det N* | *Det N PP*, (d) OGMA perturbs the initial sentence with a different terminal symbol as per the production rules of the non-terminal NP, in this case, OGMA chooses to replace "*Mary*" with "*Bob*" and gets a new sentence "*Bob saw my dog*". This new sentence, however, does not lead to any error as the set of classification classes from two models (as shown via the two circles) are largely similar. (e) OGMA backtracks and randomly chooses another terminal symbol from "*Mary saw my dog*" to perturb, in this case, the terminal "*dog*", (f) OGMA generates a perturbed sentence "*Mary saw my cat*" that also leads to an error (i.e. little overlap between the set of classification classes from two models).

```
S -> NP VP
VP -> V NP | V NP PP
V -> "saw" | "ate"
NP -> "John" | "Mary" |
       "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "dog" | "cat" | "cookie" | "park"
PP -> P NP
P -> "in" | "on" | by" | "with" | "the"
```

Fig. 3: Example Grammar

and $NP \rightarrow Bob$ (see Figure 2(a) and Figure 2(d)), respectively.

OGMA in Action: Figure 2 captures an excerpt of OGMA actions when initiated with an input sentence $I \equiv$ *Mary saw my dog*. For the sake of illustration, let us assume that the initial sentence led to a classification error. Thus, OGMA aims to explore the neighbourhood of input I and targets to discover more error inducing inputs. Figure 2(a) captures the derivation tree of the input I. We wish to find an input I' that has the same derivation tree as I except for one lead node. To this end, we randomly chose a terminal symbol appearing in I. As shown in Figure 2(b), OGMA randomly chooses the terminal symbol *Mary*. Subsequently, we discover the production rule generating the randomly chosen terminal symbol. In Figure 2, OGMA identifies this production rule to be $NP \rightarrow Mary$. Finally, OGMA generates I' by randomly choosing a production rule other than $NP \rightarrow Mary$. As observed in Figure 2(d), OGMA identifies the production rule $NP \rightarrow Bob$, leading to the new test input $I' \equiv$ *Bob saw my dog*.

The test input $I' \equiv$ *Bob saw my dog* might not lead to a classification error, as reflected in Figure 2(d). Intuitively, this can be viewed as OGMA moving outside the neighbourhood of error

TABLE 1: Notations used in OGMA approach

G	A grammar used to generate test inputs
$\mathbb{I}_G$	All inputs described by a grammar G
$\mathbb{T}_G$	The derivation trees of any input $I \in \mathbb{I}_G$
f_1, f_2	Classifiers under test.
J	A pre-determined Jaccard Threshold
τ_G	A function $\mathbb{I}_G \rightarrow \mathbb{T}_G$ which outputs the derivation tree of an input $I \in \mathbb{I}_G$
S	The initial input to the directed search. S conforms to grammar G

inducing inputs with $I' \equiv$ *Bob saw my dog*. Thus, OGMA stops performing any more modifications to input I' and backtracks. To realise this backtracking, OGMA sets the input I' to the original input *Mary saw my dog*. OGMA, then chooses another terminal symbol randomly, as observed in Figure 2(e). A terminal symbol *"dog"* (cf. Figure 2(e)) is chosen. Subsequently, OGMA finds the production rule resulting the terminal *"dog"* in a similar manner. Once OGMA finds this rule, a random terminal other than *"dog"* is chosen from this rule. This new terminal symbol will replace *"dog"* in I'. As seen in Figure 2(f), $I' \equiv$ *Mary saw my cat*. Now, input I' leads to a classification error, as indicated by ✓. Thus, OGMA follows the same steps, as explained in the preceding paragraphs, to perturb I' and generate more error inducing inputs.

In the case where the initial input $I \equiv$ *Mary saw my dog* was *not* error inducing, we continue to perturb the input until an error inducing input is discovered. In our experiments, we observed that such a strategy discovers an error inducing input quickly even though the initial input is not error inducing. Once an error inducing input is discovered, the test effectiveness of OGMA accelerates due to the presence of more error inducing inputs in the neighbourhood. Thus, the initial input could be randomly generated and it has negligible impact on the effectiveness of OGMA.

4 DETAILED METHODOLOGY

In this section we discuss our OGMA approach in detail. Our approach revolves around discovering erroneous behaviours by systematically *perturbing the* derivation tree of an input that conforms to a grammar G. First, we introduce the notion of Jaccard index, erroneous inputs, tree similarities and input perturbation before delving into the algorithmic details of our approach. We capture the notations used henceforth in Table 1.

Definition 1. ***(Jaccard Index)*** *For any two sets A and B the Jaccard Index JI is defined as follows [23]*

$$JI(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$$0 \leq JI(A, B) \leq 1$$

If A and B are both empty, we define $JI(A, B) = 1$.

Within our OGMA approach, JI is used to compare the output classification classes of two test classifiers. It is worthwhile to mention that we choose the Jaccard Index due to the choice of subject classifiers in our empirical evaluation. The choice of such a metric is modular and can be fine tuned. This means that OGMA is extensible for not only other set similarity metrics, but also for regressors.

Definition 2. ***(Erroneous input)*** *We say that input $I \in \mathbb{I}_G$ is an erroneous input if the output sets of the classifiers f_1, f_2 satisfy the following condition*

$$JI(f_1(I), f_2(I)) < J$$

The threshold J is a user-defined threshold. A lower value of J indicates a stricter condition for finding erroneous inputs.

Definition 3. ***(Tree Similarity)*** *We say two trees T_1 and T_2 are similar if we can construct a tree T by replacing exactly one leaf node in T_1 (respectively, T_2) such that T is identical to T_2 (respectively, T_1).*

Definition 4. ***(Input perturbation)*** *Let $\tau_G : \mathbb{I}_G \rightarrow \mathbb{T}_G$ be a function such that for an arbitrary input $I \in \mathbb{I}_G$, $\tau_G(I)$ is the derivation tree for I. We define $Perturb$ as a function $Perturb : \mathbb{I}_G \rightarrow \mathbb{I}_G$ such that for an input $I \in \mathbb{I}_G$, if $I' = Perturb(I)$, then $\tau_G(I)$ and $\tau_G(I')$ are similar trees.*

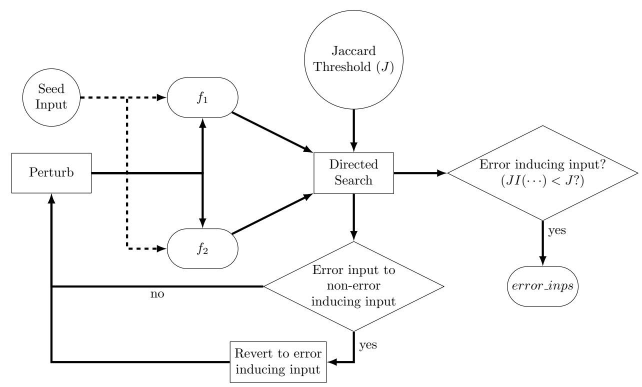


Fig. 4: Workflow of OGMA, it requires only one seed input to commence test generation.

An overview of our overall approach can be found in Figure 4. The main contribution of this paper is an automated directed test generator for grammar-based inputs. Our AUTs are machine-learning models which accept inputs conforming to certain grammars. OGMA starts with a seed input (Figure 4) that can be randomly generated from the grammar. Subsequently, OGMA involves two major steps: 1) Directed Search (DIRECTED_SEARCH) in the input domain $\mathbb{I}_G$ and 2) Input perturbation (PERTURB). In the following, we describe these two procedures in detail.

4.1 Directed Search in OGMA

The motivation behind our directed search (cf. procedure DIRECTED_SEARCH) is to contain the search in the subset of the input space $\mathbb{I}_G$ where the errors are localised. Conceptually, robustness in machine-learning captures a phenomenon stating that a slight change in the input does not change the output dramatically in well-trained machine-learning models [13]. This means that error inducing inputs are likely to cluster together in certain input subspace of well-trained models. The goal of OGMA is to discover these subspace(s) and the instances of erroneous behaviours that are present in these subspace(s).

The directed search requires the two classifiers under test (f_1, f_2), a grammar (G), a starting seed input (S) which can be randomly generated conforming to the grammar G and a Jaccard Threshold (J). The search algorithm evaluates the input S initially.

Algorithm 1 Directed Search

1: **procedure** DIRECTED_SEARCH(f_1, f_2, S, J, G)
2: $error\_inps \leftarrow \phi$
3: ▷ N is the number of iterations in the search
4: $I_{cur} \leftarrow S$
5: $Eval_S \leftarrow$ Evaluate($f_1(S), f_2(S), J$)
6: **if** $Eval_S$ is True **then**
7: $error\_inps \leftarrow error\_inps \cup \{S\}$
8: **end if**
9: **for** i in (0, N) **do**
10: ▷ See Algorithm 2
11: $I_{cand} \leftarrow$ Perturb(I_{cur}, G)
12: ▷ Evaluate if $Eval_{cand}$ and $Eval_{cur}$ are error inducing
13: $Eval_{cand} \leftarrow$ Evaluate($f_1(I_{cand}), f_2(I_{cand}), J$)
14: $Eval_{cur} \leftarrow$ Evaluate($f_1(I_{cur}), f_2(I_{cur}), J$)
15: **if** $Eval_{cand}$ is True **then**
16: $error\_inps \leftarrow error\_inps \cup \{I_{cand}\}$
17: **end if**
18: $I' \leftarrow I_{cand}$
19: ▷ This condition prevents the process from going to
20: ▷ a non-error inducing input from an error inducing
21: **if** $Eval_{cand}$ is False and $Eval_{cur}$ is True **then**
22: $I' \leftarrow I_{cur}$
23: **end if**
24: $I_{cur} \leftarrow I'$
25: **end for**
26: **return** $error\_inps$
27: **end procedure**

Algorithm 2 Perturbation

procedure PERTURB(I, G)
Let i be a random terminal symbol in I
$T \leftarrow \tau_G(I)$
Let n be the leaf node in T that contains i
▷ Parent node of n, i.e., production rule which creates i
Let $P \leftarrow$ Parent(n)
Let σ be a set of all terminal symbols in production rule P
Let $K \leftarrow \{k \mid k \in \sigma \setminus \{i\}\}$
if $K = \emptyset$ **then**
return print ("Cannot Perturb Terminal")
end if
Let i' be a randomly chosen terminal symbol in K
▷ Replace i with i' in I
$I' \leftarrow I[\![i \rightarrow i']\!]$
return I'
end procedure

Algorithm 3 Evaluate

procedure EVALUATE(A, B, J)
if $JI(A, B) < J$ **then**
return True
else
return False
end if
end procedure

It finds the Jaccard Index (cf. Definition 1) of the output sets $f_1(S)$ and $f_2(S)$. If $JI\,(f_1(S), f_2(S))$ is lower than the threshold J, then the input S is added to the set $error\_inps$ and S is assigned to I_{cur} for the first iteration. Intuitively, this means S falls in the region of error inducing input subspace and thus, it is likely to lead to more error inducing inputs via perturbation.

At any point, the directed search process keeps track of two crucial inputs, namely I_{cur} (Current input) and I_{cand} (Candidate input), respectively. I_{cur} is the input that was discovered in the latest iteration of the directed search. I_{cur} can be an error or non error input (cf. Definition 2). I_{cand} is the perturbed input resulting from I_{cur} (cf. procedure PERTURB), i.e., $I_{cand} = Perturb(I_{cur})$ according to Definition 4. The goal of OGMA with the perturbation is to either discover more error inputs (if I_{cur} is already an error input) or to discover a subspace of $\mathbb{I}_G$ which contains error inputs (if I_{cur} is a non-error input).

It is crucial for OGMA to keep track of the transition sequence between error and non-error inducing inputs during the test generation process. Specifically, OGMA prevents the test generation process from entering an input subspace containing non-error inducing inputs from the subspace containing an error inducing inputs. The rationale behind such a strategy is backed by the robustness property of machine-learning models, as perturbing error inducing inputs is certainly more effective than perturbing non-error inducing inputs. As an example, let I_{cur} be "*Mary saw my dog*", which is an input that causes erroneous behaviour (cf. Definition 2) in the classifiers f_1 and/or f_2. It is part of a subset of $\mathbb{I}_G$ in these classifiers which causes these classifiers to exhibit erroneous behaviours. Let the perturbation of I_{cur} result in "*Bob saw my dog*", which is assigned to I_{cand}. Let us assume I_{cand} does not show erroneous behaviours and thus, is located in an input subspace that is unlikely to exhibit erroneous behaviours. In this case, therefore, we discard I_{cand} (line 21 in Algorithm 1) and backtracks the test generation process to induce a different perturbation to I_{cur}.

In the case where I_{cand} does induce an erroneous behaviour, the test generation process is focused to search in the vicinity of I_{cand} to find more such inputs. In this case, we update I_{cur} to the value in I_{cand} and proceed to the subsequent iterations to repeat the perturbation steps.

It is worthwhile to note that there are four possible transitions between inputs in each iteration. These are namely, *Non-error inducing input → Error inducing input*, *Error inducing input → Error inducing input*, *Non-error inducing input → Non-error inducing input* and *Error inducing input → Non-error inducing input*. We only move out of a subspace of interest of $\mathbb{I}_G$ in the last case. This is to avoid getting stuck in a region that does not contain error inducing inputs.

4.2 Perturbation in OGMA

The perturbation function has two responsibilities. The first responsibility of this function is to discover the input subspace $\mathbb{I}_G$ that contains erroneous inputs. The second responsibility is to explore this subspace to find instances of error inducing inputs in the same. The first responsibility is captured when the initial seed input S is non-error inducing. As we can see in Figure 8, in some cases OGMA produces several non-error inputs in the initial stages of the test generation process. This is because the initial input to OGMA is in some part of the input subspace of $\mathbb{I}_G$ where the inputs show non erroneous behaviour. Thus, we need to perturb the input to get the process out of this subspace and find a subspace which shows erroneous behaviour. OGMA continuously perturbs the input to find such a subspace. As we can see in Figure 8(a), eventually after ≈ 200 iterations, OGMA finds the subspace of $\mathbb{I}_G$ where the inputs do indeed show erroneous behaviours.

The function PERTURB chooses a random terminal i from an input $I \in \mathbb{I}_G$. Then, we obtain the derivation tree $T = \tau_G(I)$. In this derivation tree, we find the leaf node that contains i and discover the parent of this node. This, in turn, gives the production rule P that produced the terminal symbol i. In the next step we construct a set K, which includes all the terminal symbols we have found in the production rule P, except for the terminal symbol i. If we reconsider the example in Figure 2(b), the set of such terminal symbols would be $K = \{$“*John*”, “*Bob*”$\}$. We choose a random terminal symbol $i' \in K$. We will use this terminal symbol to replace $i \in I$ to create a new input I'. In the example, if we choose “*John*”, and replace “*Mary*”, the new sentence $I' \in \mathbb{I}_G$ will be “*John saw my dog*”.

As a result of the design of the perturbation function, the choice of grammar plays an important role in the success of OGMA. The idea of the perturbation is to generate a substantial number of inputs with similar derivation trees.

4.3 Similar Sentences and Perturbation

It is important to note that sentences that might appear similar, may not be considered similar (cf. Definition 3) in OGMA. This difference is best brought out with the help of an example. Concretely, consider the grammar seen in Figure 5 and the derivation tree of the sentence “*Frank saw my dog*” generated from this grammar.

```
S -> NP VP
VP -> V NP | V NP PP
V -> "saw" | "ate"
NP -> "John" | "Mary" |
        "Bob" | Det N | Det N PP | X
Det -> "a" | "an" | "the" | "my"
N -> "dog" | "cat" | "cookie" | "park"
PP -> P NP
P -> "in" | "on" | by" | "with" | "the"
X -> "Thomas" | "Frank" | "Alex"
```

Fig. 5: Modified Example Grammar

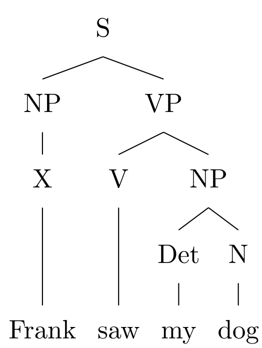


Fig. 6: Derivation tree for “*Frank saw my dog*”

Two sentences that conform to this grammar are $I_1 =$ “*Mary saw my dog*” and $I_2 =$ “*Frank saw my dog*”. The derivation tree for I_1 can be seen in Figure 2(a). As we can clearly see the structures of the derivation trees for I_1 and I_2 are different and as a result, these similar sentences are not considered similar inputs in OGMA (cf. Definition 3). In other words, OGMA considers inputs to be *similar* only if they are derived *similarly* from the candidate grammar. This is a stricter condition over the similarity of the actual sentences. The similar sentences (cf. Definition 3) for I_1 would be “*Bob saw my dog*” and “*John saw my dog*” according to OGMA. Likewise, the similar sentences for I_2 would be “*Thomas saw my dog*” and “*Alex saw my dog*”.

TABLE 2: Notations used in Evaluation

#inputs	Total number of generated test inputs
#err	Total number of erroneous inputs
err_r	$\frac{\#err}{\#inputs}$
Imp%	Improvement of err_r of OGMA with respect to the err_r of random test

5 Results

Experimental Setup

We evaluate OGMA across three industrial text classification models provided by uClassify [4], Aylien [1] and Rosette [3] text analytics. We have chosen these classifiers for two reasons. Firstly, these service providers are used in industry scale, such as in Amazon, Airbnb, Microsoft and Oracle among others. Secondly, our chosen service providers use text classifiers that categorise input text into a standard text classification taxonomy called the IAB content taxonomy [2]. At a broader perspective, such a classification taxonomy acts as a guideline on the types of classes that a text can be categorised to. In other words, this ensures a standardisation of classes across a variety of text classifiers. A sample classification via the IAB (Interactive Advertising Bureau) Content Taxonomy can be found in Figure 7. “Automotive” is the broadest level of classification (Tier 1). Underneath this classification, there exists increasingly specific categories. For instance, Tier 2 under the category ”Automotive” includes “Auto Body Styles”, “Auto Type” and “Auto Technology”. Tier 3 is the most specific classification. Examples under “Auto Type” include “Budget Cars”, “Classic Cars”, “Concept Cars” etc. For our evaluation, we have only considered the top tier classification. This is because we expect the classifier to at least have similar classification at the broadest category (i.e. Tier 1).

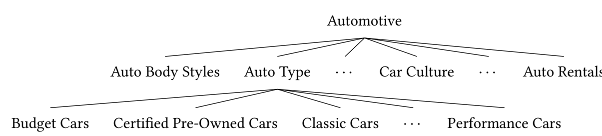


Fig. 7: IAB Content Taxonomy hierarchy

As we leverage differential testing, we aim to validate whether the output classes from two different text classifiers are similar. Since all our classifiers implement the IAB content taxonomy, we can compare their outputs coherently. Subsequently, we guide our test generation methodology to discover inputs that lead to vastly dissimilar classifier outputs (according to the IAB content taxonomy). We access the services of uClassify, Aylien and Rosette via client-side APIs. We engineer each API call to classify a sentence (as automatically generated via OGMA) and to return a set of at most the five most likely results. For each test environment, we consider a pair of classifiers from different service providers to facilitate differential testing. To check the similarity between classifier outputs, we compute the Jaccard Index of outputs from two classifiers. If the computed Jaccard Index is below a certain threshold J (c.f. Definition 2), then we consider the input, leading to the respective Jaccard Index, as erroneous for at least one of the text classifier. The threshold J is user defined and we evaluate OGMA to check its sensitivity with respect to the threshold J.

For the sake of brevity, we refer to Aylien as $\mathbb{A}$, Rosette as $\mathbb{R}$ and uClassify as $\mathbb{U}$ for the rest of this section. We also use the notations in Table 2 to describe the evaluation results.

Choice of Input Grammars

We validate OGMA using six different grammars (see Appendix for all the grammars used). As explained in Section 4, OGMA essentially perturbs the derivation trees for an input generated from a grammar. Such a perturbation forms the crux of our systematic test generation while searching the neighbourhood of an erroneous input. We consider two inputs to be in the same neighbourhood if their derivation tree have the same structure (cf. Definition 2). Thus, to continue test generation via OGMA, the chosen grammar must encode a substantial number of inputs with the same derivation tree structure (see Figure 2). To this end, we chose grammars that support production rules with multiple possible terminal symbols. For example, consider the grammar used in Section 3. In this grammar, production rules from each non-terminal V, NP, Det, N and P lead to multiple possible terminal symbols.

Key Results

We construct three possible pairs of classifiers from the three text classifiers under test. Each pair of classifiers were validated with the six subject grammars chosen for evaluation. Table 3 outlines our key findings averaging over all such evaluation scenarios. The average is calculated over a varying threshold $[0.1, 0.3]$ (cf. Definition 2) to check the dis(similarity) of classifier outputs.

In our evaluation, we intend to check whether our directed strategy indeed improves the state-of-the-art test generation methodologies for arbitrary machine-learning models. However, to the best of our knowledge, there does not exist any directed strategy for grammar-based test input generation with the objective to uncover errors in such models. Thus, to evaluate the effectiveness of OGMA, we compare it with a strategy that randomly generates sentences (Random) conforming to the input grammar and employs differential testing as embodied within OGMA. We aim to show that if OGMA generates more error inducing inputs than Random, then it is a step forward in designing directed, yet scalable methods for grammar-based test input generation targeting arbitrary machine-learning models.

As observed in Table 3, OGMA outperforms Random by a significant margin (up to 54%). We attribute this improvement to the directed test strategy integrated within OGMA. Specifically, OGMA discovers more erroneous inputs than Random by exploiting the robustness property of common machine-learning models and realising this via a focused search in the neighbourhood of already discovered erroneous inputs. To evaluate the effectiveness of OGMA in detail, we have answered the following research questions (RQs).

TABLE 3: Key Results (the initial input is not erroneous)

	OGMA			Random			
	#inputs	#err	err_r	#inputs	#err	err_r	Imp%
$\mathbb{R}$-$\mathbb{A}$	1949	1757	0.9	1798	1276	0.7	27.06
$\mathbb{U}$-$\mathbb{A}$	1920	1286	0.87	1778	1305	0.73	19.69
$\mathbb{R}$-$\mathbb{U}$	1917	1312	0.68	1798	797	0.44	54.3

RQ1: Can the robustness property be leveraged for systematically testing real-world text classifiers?

Intuitively, robustness is a concept in machine learning which states that changing the input to any well trained machine learning model by some small value δ should not change the respective output dramatically. This means that similar inputs with similar outputs are likely to be clustered together. Thus, inputs similar to an error inducing input are also likely to lead to classification errors.

We present two cases (cf. Figure 8) where we have discovered clear indications of robustness playing a role in error discovery. In both Figure 8(a) and Figure 8(b), the graphs demonstrate whether a given test iteration leads to an error (output *True*) or not (output *False*). We have the following crucial observations from Figure 8. Firstly, we observe that once OGMA finds an error inducing input (e.g. around iteration 200 in Figure 8(a)), it continues to discover more error inducing inputs (e.g. approximately until iteration 300). This is because OGMA by design ensures to explore the neighbourhood of an input, whereas the robustness property of machine-learning models ensure that the neighbourhood of an error inducing inputs are likely to be error inducing too. Thus, OGMA discovers a stretch of error inducing inputs, as observed between iterations 200 to 300 in Figure 8(a). Secondly, we observe from Figure 8 that the directed search embodied in OGMA is useful in terms of steering the execution to errors. For example, even though OGMA discovers non-error inducing inputs, it can quickly revert to find error inducing inputs (e.g. approximately between iterations 300 and 350 in Figure 8(a)). Similar characteristics can also be observed in Figure 8(b). Due to the aforementioned characteristics, OGMA significantly outperforms the random test generation. This validates our hypothesis that robustness of well trained machine learning model can indeed be leveraged to design systematic and scalable test generation methodologies.

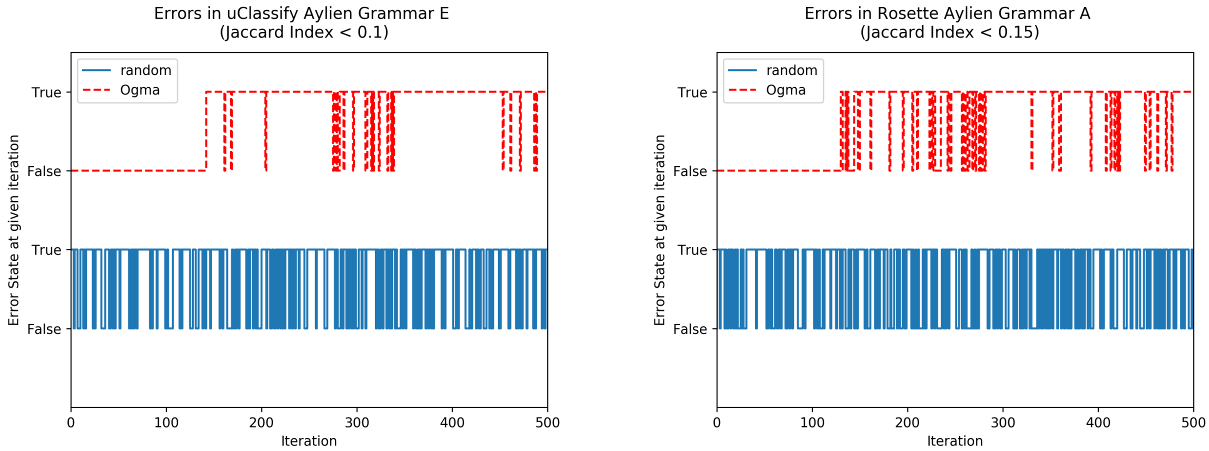
RQ2: How effective is OGMA in terms of generating error inducing inputs for a variety of real-world text classifiers?

We measure the effectiveness of OGMA as the ratio of the number of errors found with respect to the number of unique inputs generated (cf. Table 3). On average the random approach discovers errors with a ratio of 0.7, 0.73 and 0.44 for the pairs of classifiers $\mathbb{R}$-$\mathbb{A}$, $\mathbb{U}$-$\mathbb{A}$, $\mathbb{R}$-$\mathbb{U}$, respectively. In comparison, the ratio of errors discovered via OGMA are 0.9, 0.87 and 0.68 respectively. Thus, on average, we observe that the directed search strategy embodied in OGMA improves the effectiveness of test generation by 33.68%.

RQ3: Does the effectiveness of OGMA depend on initial test input?

We validated whether the initial input plays a major role in the effectiveness of OGMA. To this end, we conducted two sets of experiments – one where the initial input induced an error (i.e. two classifiers under test had dissimilar outputs) and another where the initial input was not an error inducing input (i.e. two classifiers under the test had similar outputs). We discovered that the initial input does not play a major role in the effectiveness of OGMA. Figure 9 outlines our finding. Specifically, Figure 9 captures the average ratio of error inducing inputs discovered over all grammars and text classifiers.

In Figure 9, the effectiveness of random test generation (in terms of discovering error inducing inputs) improves marginally



(a) **(b)**

Fig. 8: The rationale behind using robustness for error discovery

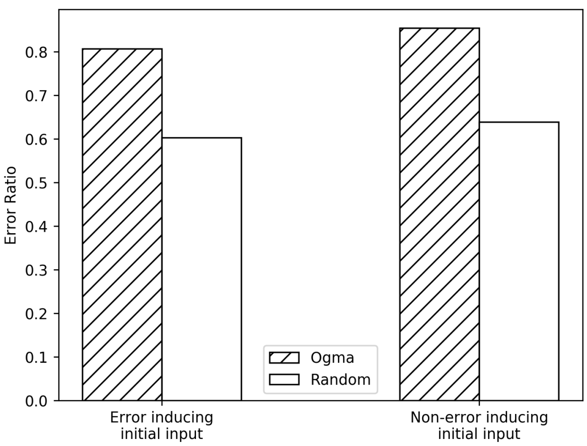


Fig. 9: Sensitivity of OGMA w.r.t. the choice of initial input

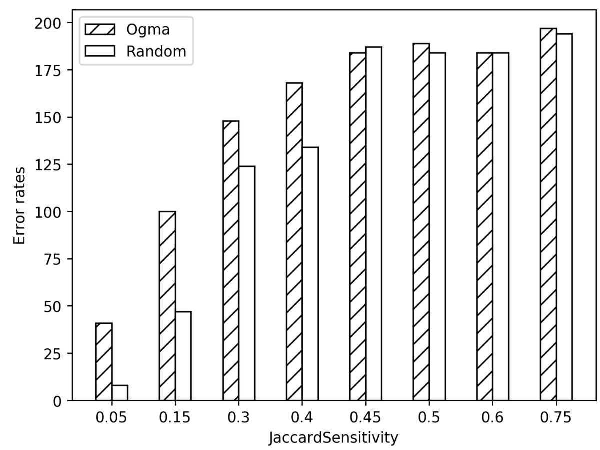


Fig. 10: Sensitivity of OGMA w.r.t. the threshold J for checking Jaccard Index

by 1.57% when initiated with an error inducing input. In general, the effectiveness of random test generation should be unaffected by the initial input, as each test input is generated independently. Concurrently, the effectiveness of OGMA also improves by a negligible 4.94% when the initial input is error inducing. Thus, we conclude that the initial input does not influence the effectiveness of our test generation methodologies significantly. However, as also seen in Figure 9, the relative improvement due to the directed strategy in OGMA, over the random test generation strategy, remains over 33% regardless of the category of initial input.

TABLE 4: Sensitivity of OGMA w.r.t. the threshold J for checking Jaccard Index

	OGMA			Random			
	#inputs	#err	err_r	#inputs	#err	err_r	Imp%
0.05	172	41	0.23	198	8	0.04	489.97
0.15	189	100	0.53	196	47	0.24	120.65
0.3	196	148	0.76	200	124	0.62	21.79
0.4	195	168	0.86	195	134	0.69	25.37
0.45	196	184	0.93	193	187	0.96	-3.11
0.5	193	189	0.98	197	184	0.93	4.85
0.6	195	184	0.94	197	184	0.93	1.0
0.75	197	197	1	195	194	0.99	0.51

RQ4: How sensitive is OGMA w.r.t. the threshold J (cf. Definition 2) to check the similarity of classifier outputs?

To answer this research question, we varied the threshold J (cf. Definition 2) for grammar A (see Appendix). The initial input for the test generation led to a Jaccard Index > 0.15, but < 0.3. Thus, for threshold values $[0.05, 0.15]$, the initial input was not error inducing, whereas for threshold ≥ 0.3, the initial input was error inducing. Finally, the reported values in this experiments were averaged over all possible pairs of classifiers (i.e. $\mathbb{R}$-$\mathbb{A}$, $\mathbb{U}$-$\mathbb{A}$ and $\mathbb{R}$-$\mathbb{U}$).

We observe a direct correlation between the chosen threshold J and the effectiveness of OGMA (cf. Table 4). In particular, a low threshold value for J (cf. Definition 2) indicates that the tested classifier outputs have vastly dissimilar content. Thus, the lower the threshold J, the lower is also the probability to discover erroneous inputs. In other words, if we keep the threshold J low, it is difficult for a random test generation strategy to discover error inducing inputs. As a result, for such scenarios, the directed

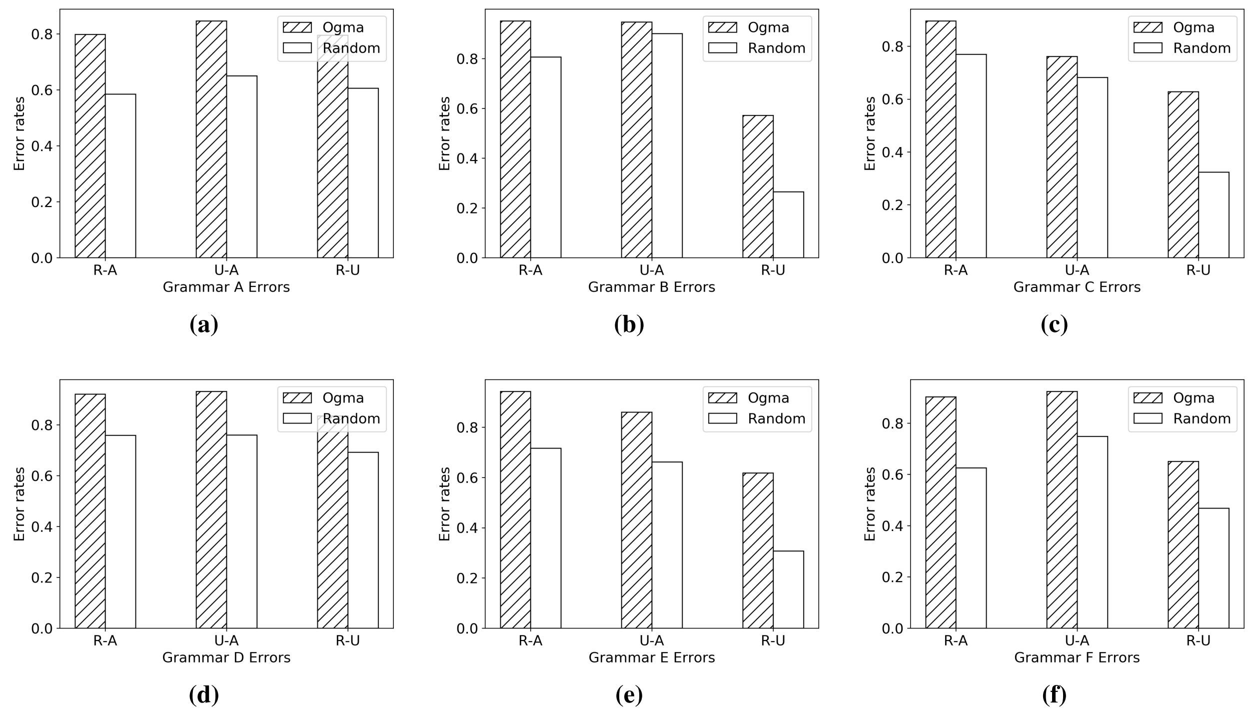


Fig. 11: Sensitivity of Ogma with respect to different grammars (see Appendix for the grammars)

test strategy in Ogma outperforms random test generation by a significant margin (up to 489%). In contrast, for a higher threshold J, even a slightly dissimilar classifier outputs might be categorized as errors. As such, for higher threshold (e.g. between 0.45 and 0.75), the effectiveness of Ogma and the random test generation strategy is similar.

Figure 10 provides the trend of discovered error inputs with respect to the threshold J. The number of errors found by Ogma is consistently higher than the random approach except for threshold value 0.45. For threshold value 0.45, random strategy is marginally better due to the ease of finding error inputs with high probability. The observations in Table 4 and in Figure 10 reveal that Ogma should be used for finding error inducing inputs where the error condition is strict (i.e. low threshold J for the computed Jaccard Index). This is because such error inducing inputs are unlikely to be discovered via a random search, while Ogma can discover these inputs effectively by leveraging the robustness property of machine-learning models.

RQ5: How sensitive is Ogma w.r.t. the chosen grammar?

As discussed earlier in this section, we employ a directed search in the neighbourhood of an error inducing input. This is accomplished by only perturbing a leaf node of the derivation tree, yet keeping the structure of the derivation tree similar. As such, we have chosen grammars (see Appendix) to generate a substantial number of test inputs by perturbing only leaf nodes of the derivation tree for a given input.

We evaluate the effectiveness of Ogma for six different grammars chosen for our evaluation and our findings are demonstrated via Figure 11. For each set of experiments, we measure the ratio of error inducing inputs (with respect to the total number of generated inputs) discovered for both random testing and Ogma. As observed from Figure 11, our Ogma approach is consistently more effective than random test generation and its effectiveness is not compromised across a variety of grammars. Specifically, we obtain a maximum improvement of up to 94% (for classifiers $\mathbb{R}$-$\mathbb{U}$ and with Grammar C) and an average improvement of up to 33% across all grammars and classifiers.

μRQ: Can we use the error inducing inputs generated by Ogma to improve the accuracy of classifiers?

As part of this research question, we intend to check the usage of error inducing inputs generated by Ogma. A natural usage of these inputs is to retrain the classifier under test. Such a retraining can be accomplished by augmenting the training sets with the generated error inducing inputs. However, as we only had usage-level access to the text classifiers from Rosette, Aylien and uClassify, we were unable to retrain these classifiers. Thus, for this research question, we evaluated two classifiers from scikit-learn implementations of a regularized linear model with stochastic gradient descent and the multinomial Naive Bayes classifier. The objective of these classifiers is to classify a given sentence based on which grammar they were generated from. We used the following grammars seen in Figure 12 and Figure 13 in the evaluation:

Initially, the classification accuracy was 99.75% for the classifiers. Subsequently, we used Ogma to generate inputs such that the outputs of the chosen two classifiers are different. We add a sample of the generated erroneous test inputs into the training set and retrain the classifier. To generate the correct labels for the erroneous test inputs, we considered one classifier to be the oracle and assign the output generated by the oracle as the label.

```
S -> NP VP
VP -> V NP | V NP PP
V -> "saw" | "ate"
NP -> "John" | "Mary" |
        "Bob" | Det N | Det N PP
Det -> "a" | "an" | "the" | "my"
N -> "dog" | "cat" | "cookie" | "park"
PP -> P NP
P -> "in" | "on" | by" | "with" | "the"
```

Fig. 12: Toy Grammar 1

```
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP | "I"
VP -> V NP | VP PP
V -> "shot" | "killed" | "wounded"
Det -> "an" | "my"
N -> "elephant" | "pajamas" | "cat" |
     "dog"
P -> "in" | "outside"
```

Fig. 13: Toy Grammar 2

Subsequently, we train the other classifier with the augmented training set.

We generated 1000 test inputs via OGMA to discover the number of error inducing inputs before and after retraining. Since OGMA has randomness involved in its core, we repeated the test generation 50 times and take the average over all 50 iterations. Our findings are summarized in Table 5. On average, OGMA generated 553 error inducing inputs (out of 1000) before retraining, whereas the number of error inducing inputs reduced to 416 (i.e. 24.77% decrease) after the retraining. This experiment clearly shows that the test inputs generated by OGMA can be utilized to reduce the erroneous behaviours in classifiers.

TABLE 5: Subject classifiers used to evaluate μRQ

Classifier		Original	Retrained
SGDClassifier	Accurary	99.75	99.76
Multinomial NB		99.75	99.76
	Average Errors	553	416

Examples of Error Inducing Inputs

In this section, we introduce some of the interesting error inducing inputs automatically discovered by OGMA. For instance, consider the following sentence generated from one of our subject grammars:

$$the\ monkey\ shot\ Bob$$

The text classifier $\mathbb{U}$ returns the following result (top three categories) where the first element in the pair captures the classification class (according IAB content Taxonomy Tier 1) and the second element captures the weight (i.e. a score reflecting how likely is the respective category):

1) 'HOBBIES AND INTERESTS', 0.371043
2) 'SOCIETY', 0.167253
3) 'SPORTS', 0.118665

Another classification for the example sentence

$$I\ shot\ John\ with\ Mary$$

leads to the following classification classes:

1) 'SOCIETY', 0.840454
2) 'ARTS AND ENTERTAINMENT', 0.159546
3) 'SPORTS', 6.65587e^{-11}

As observed from the preceding examples, the computed categories were clearly erroneous.

We contacted the developers of the service providers of text classifiers and pinpointed them to the erroneous inputs. Developers confirm that these are indeed erroneous behaviours of the classifiers. They also confirmed that the primary reason for such erroneous behaviours is that the respective classifiers were inadequately trained for the type of text inputs generated by OGMA. Thus, for these texts, the classifiers failed to provide a reasonable classification class. This experience clearly indicates the utility of OGMA, as the directed test strategy embodied within OGMA can rapidly discover such erroneous behaviours due to inappropriate training. Moreover, as observed in our μRQ, OGMA can also augment the training set with the generated erroneous inputs. This, in turn, helps to improve the accuracy of classifiers, as observed in our experiments.

6 RELATED WORK

In this section, we review the related literature and position our work on testing machine-learning systems.

Testing of machine-learning models: DeepXplore [21] is a whitebox differential testing algorithm for systematically finding inputs that can trigger inconsistencies between multiple deep neural networks (DNNs). The neuron coverage was used as a systematic metric for measuring how much of the internal logic of a DNNs had been tested. More recently, DeepTest [24] leverages metamorphic relations to identify erroneous behaviors in a DNN. The usage of metamorphic relations somewhat solves the limitation of differential testing, especially to lift the requirement of having multiple DNNs implementing the same functionality. A feature-guided black-box approach is proposed recently to validate the safety of deep neural networks [28]. This work uses their proposed method to evaluate the robustness of neural networks in safety-critical applications such as traffic sign recognition. DeepGauge [18] formalizes a set of testing criteria based on multi level and -granularity coverage for testing DNNs and measures the testing quality. AEQUITAS [25] aims to uncover fairness violations in machine learning models. DeepConcolic [22] designs a coherent framework to perform concolic testing for discovering violations of robustness. More recently, Wang et al. [26] leverages mutation testing at the level of DNN model parameters to find adversarial samples.

The aforementioned works are either not applicable for structured inputs [25] or they require a set of concrete seed inputs to initiate the test generation process [21], [24], [28]. On the contrary, OGMA encodes input domain via grammars and systematically generates inputs conforming to the grammar by exploiting the robustness property. Due to the grammar-based input generation, OGMA can explore an input subspace that could be beyond the capability of techniques relying on concrete seed inputs. Presence of an input grammar is also common for several machine learning models, especially for models in the domain of text classification. Moreover, the objective of the works, as explained in the preceding paragraph, is largely to evaluate salient properties, e.g., fairness and robustness, of a given machine-learning model. In contrast, our OGMA approach is targeted to discover classification errors in machine-learning models in a generic fashion, while leveraging the robustness property of these well trained models.

Verification of Machine Learning models: AI^2 [15] uses abstract interpretation to verify the robustness of a given input against adversarial perturbations. AI^2 leverages zonotopes to approximate ReLU outputs. The authors guarantee soundness, but not precision. ReluVal [27] uses interval arithmetic [20] to estimate a neural network's decision boundary by computing tight bounds on the output of a network for a given input range. The authors leverage this to verify security properties of a Deep Neural Network. Similarly, Reluplex [16] uses SMT solvers to verify these security properties. They present an SMT solver and encode properties of interest into this SMT solver. Dvijotham et al. [12] transform the verification problem into an unconstrained dual formulation using Lagrange relaxation and use gradient-descent to solve the respective optimization problem.

In contrast to these works, our OGMA approach has the flavor of testing. Specifically, our OGMA approach does not generate false positives, i.e., all witnesses generated by OGMA indeed capture erroneous behaviours in test classifier(s). Moreover, these witnesses generated by OGMA can be used to retrain the test classifiers and thus reducing the number of erroneous classifications.

Search based testing: Search-based testing has a long standing history in the domain of software engineering. Common techniques for search-based software testing are hill climbing, simulated annealing and genetic algorithms [19]. These have been applied extensively to test applications that largely fall in the class of deterministic software systems. With this work we aim to uncover ways to adapt these techniques to statistical software in general.

7 Threats to Validity

Choice of Grammar: OGMA implements a perturbation algorithm which perturbs the structure of the derivation tree of an input. The key requirement of OGMA is that there should be many inputs which have the same structure for their derivation trees. This is not possible with grammars that have only one terminal symbols in their production rules. A derivation tree constructed from such a grammar will not be perturbed by OGMA, and would lead to very restricted testing. However, the rationale behind perturbation in OGMA is to exploit the robustness property in machine-learning models for scalable testing. Specifically, we postulated that inputs having similar derivation tree structure are likely to be classified similarly and our empirical results validated this.

Robustness: OGMA is based on the hypothesis that the machine-learning models under test exhibit robustness. This is a reasonable assumption, as we expect the models under test to be deployed in real-world settings. As evidenced by our evaluation, OGMA approach, which is based on the aforementioned hypothesis, was effective to localize the search in the neighbourhood of regions exhibiting erroneous behaviours.

Complex Inputs: Currently, OGMA only works on input domain encoded by context free grammars. OGMA is not evaluated on more complex input structures such as images and videos. To adapt our OGMA approach for such complex inputs, a model that encodes these inputs is needed. This can be accomplished in a future extension of OGMA.

Size of Input Text: We have tested classifiers that are claimed to not perform well for short text. It was brought to our notice that the classifier models need more context for the task of classification. We cannot conclude the effectiveness of OGMA for longer texts. However, the open architecture of OGMA allows for extensive evaluation of grammars generating longer texts.

8 Conclusion

In this paper, we present OGMA, a fully automated technique to generate grammar-based inputs which exhibit erroneous behaviours in machine-learning models. At the core of OGMA lies a novel directed search technique. The key insight behind OGMA is to exploit the robustness property inherent in any well trained machine-learning model. OGMA provides comprehensive empirical proof for errors in text classifiers. To the best of our knowledge, OGMA is the only grammar-based machine learning testing solution to date. We provide a generic and modular framework to any user of our tool to extend the application of OGMA beyond classifiers. We also try and retrain a toy classifier models to show the potential use cases of these discovered erroneous behaviours. In future, we plan to extend the capability of OGMA to automatically localize the cause of errors discovered in the machine-learning models.

OGMA lifts the state of the art by introducing a novel approach to testing for machine-learning models. We envision to extend OGMA beyond just text classifier testing and we hope it can be used to test any machine-learning model whose input domain can be formalized not only via grammars, but also via other techniques such as via leveraging logic based on satisfiability modulo theory (SMT). We would also like to extend OGMA to video and image inputs. We hope that the central idea behind our OGMA approach would influence the rigorous software engineering principles and help validate machine-learning applications. For reproducibility and advancing the state of research, we have made our tool and all experimental data publicly available:

https://github.com/sakshiudeshi/Ogma-Data
https://github.com/sakshiudeshi/Ogma

References

[1] Aylien, 2019. URL: https://aylien.com.

[2] IAB Content Taxonomy, 2019. URL: https://www.iab.com/guidelines/taxonomy/.

[3] Rosette Text Analytics, 2019. URL: https://www.rosette.com.

[4] uClassify, 2019. URL: https://www.uclassify.com.

[5] Tesla : A Tragic Loss, 2106. URL: https://www.tesla.com/blog/tragic-loss.

[6] Tesla death smash probe: Neither driver nor autopilot saw the truck, 2107. URL: https://www.theregister.co.uk/2017/06/20/tesla_death_crash_accident_report_ntsb/.

[7] Amazon scraps secret AI recruiting tool that showed bias against women, 2108. URL: https://uk.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUKKCN1MK08G.

[8] How Is AI Used In Education – Real World Examples Of Today And A Peek Into The Future, 2108. URL: https://www.forbes.com/sites/bernardmarr/2018/07/25/how-is-ai-used-in-education-real-world-examples-of-today-and-a-peek-into-the-future/#29b9f6a4586e.

[9] Nissan-Renault alliance to join Google on self-driving cars, 2109. URL: https://asia.nikkei.com/Business/Companies/Nissan-Renault-alliance-to-join-Google-on-self-driving-cars.

[10] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE Trans. Software Eng.*, 41(5):507–525, 2015. URL: https://doi.org/10.1109/TSE.2014.2372785, doi:10.1109/TSE.2014.2372785.

[11] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57, 2017. URL: https://doi.org/10.1109/SP.2017.49, doi:10.1109/SP.2017.49.

[12] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A. Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 550–559, 2018. URL: http://auai.org/uai2018/proceedings/papers/204.pdf.

[13] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers' robustness to adversarial perturbations. *Machine Learning*, 107(3):481–508, Mar 2018.

[14] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 498–510, 2017. URL: http://doi.acm.org/10.1145/3106237.3106277, doi:10.1145/3106237.3106277.

[15] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin T. Vechev. AI2: safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 3–18, 2018. URL: https://doi.org/10.1109/SP.2018.00058, doi:10.1109/SP.2018.00058.

[16] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 97–117, 2017. URL: https://doi.org/10.1007/978-3-319-63387-9_5, doi:10.1007/978-3-319-63387-9_5.

[17] Lindsey Kuper, Guy Katz, Justin Gottschlich, Kyle Julian, Clark Barrett, and Mykel J. Kochenderfer. Toward scalable verification for safety-critical deep networks. *CoRR*, abs/1801.05950, 2018. URL: http://arxiv.org/abs/1801.05950, arXiv:1801.05950.

[18] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. Deepgauge: multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 120–131, 2018.

[19] Phil McMinn. Search-based software test data generation: a survey. *Softw. Test., Verif. Reliab.*, 14(2):105–156, 2004.

[20] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009. URL: https://doi.org/10.1137/1.9780898717716, doi:10.1137/1.9780898717716.

[21] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 1–18, 2017. URL: http://doi.acm.org/10.1145/3132747.3132785, doi:10.1145/3132747.3132785.

[22] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 109–119, 2018. URL: https://doi.org/10.1145/3238147.3238172, doi:10.1145/3238147.3238172.

[23] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[24] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 303–314, 2018. URL: https://doi.org/10.1145/3180155.3180220, doi:10.1145/3180155.3180220.

[25] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. Automated directed fairness testing. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 98–108, 2018. URL: https://doi.org/10.1145/3238147.3238165, doi:10.1145/3238147.3238165.

[26] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. Adversarial sample detection for deep neural network through model mutation testing. *CoRR*, abs/1812.05793, 2018. URL: http://arxiv.org/abs/1812.05793, arXiv:1812.05793.

[27] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 1599–1614, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi.

[28] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I*, pages 408–426, 2018. URL: https://doi.org/10.1007/978-3-319-89960-2_22, doi:10.1007/978-3-319-89960-2_22.

Appendix A
Grammars and Additional Graphs

In the appendix below we provide all the grammars that we used for testing and the additional experimental results that we obtained.

```
S -> NP VP

VP -> V NP | V NP PP | VP PP

PP -> P NP

V -> "saw" | "ate" | "walked"
     | "shot" | "killed" | "wounded"

NP -> "John" | "Mary" | "Bob"
     | Det N | Det N PP | "I"

Det -> "a" | "an" | "the"
     | "my" | "an" | "my"

N -> "man" | "dog" | "cat"
     | "telescope" | "park"
     | "elephant" | "pajamas"
     | "monkey" | "fish"

P -> "in" | "on" | "by" | "with"
     | "outside"
```

Fig. 14: Grammar A

```
S -> NP VP

VP -> V NP | V NP PP | VP PP

PP -> P NP

V -> "went" | "caught" | "ran"
     | "injured" | "captured"
     | "wounded" | "viewed"

NP -> "Mark" | "Elise" | "Steve"
     | Det N | Det N PP | "I"

Det -> "a" | "an" | "the" | "my"

N -> "man" | "monkey" | "squirrel"
     | "binoculars" | "lawn"
     | "giraffe" | "hedgehog"
     | "fish"

P -> "in" | "on" | "by" | "with"
     | "outside" | "near"
```

Fig. 15: Grammar B

```
S -> NP VP

VP -> V NP | V NP PP | VP PP

PP -> P NP

V -> "spent" | "grabbed" | "chased"
     | "damaged" | "apprehended"
     | "disabled" | "saw"

NP -> "Gary" | "Gemma" | "Nick"
     | Det N | Det N PP | "I"

Det -> "a" | "an" | "the" | "my"

N -> "woman" | "lemur"
     | "baboon" | "park"
     | "elephant" | "gibbon"
     | "salmon" | "river" | "owl"

P -> "in" | "on" | "by" | "with"
     | "outside" | "near" | "inside"
```

Fig. 16: Grammar C

```
S -> NP VP

VP -> V NP | V NP PP | VP PP

PP -> P NP

V -> "began" | "built" | "caught"
     | "fought" | "heard"
     | "meant" | NP

NP -> "Stephen" | "Irene"
     | "James" | Det N
     | Det N PP | "I"

Det -> "a" | "an" | "the"
     | "my" | "an" | "my"
     | N | N PP
N -> "man" | "tree" | "cat"
     | "telescope" | "ship"
     | "monkey" | "pajamas"
     | "mountain" | "country" | PP

P -> "in" | "on" | "by" | "with"
     | "outside" | NP
```

Fig. 17: Grammar D

```
S -> NP VP

VP -> V NP | V NP PP | VP PP

PP -> P NP

V -> "started" | "made" | "captured"
        | "conflicted" | "embarked"
        | "studied" | NP

NP -> "Marcus" | "Holly" | "Dylan"
        | Det N | Det N PP | "I"

Det -> "a" | "an" | "the"
        | "my" | N | N PP

N -> "man" | "forest" | "cat" |
        "camera" | "bus" | "snake"
        | "pajamas" | "hill"
        | "province" | PP

P -> "in" | "on" | "by"
        | "with" | "outside" | NP
```

Fig. 18: Grammar E

```
S -> NP VP

VP -> V NP | V NP PP | VP PP

PP -> P NP

V -> "knew" | "thought"
        | "looked" | "tried"
                | "needed" | "stood" | NP

NP -> "Alexander" | "Olivia"
        | "Thomas" | Det N
        | Det N PP | "I"

Det -> "a" | "an" | "the"
        | "my" | N | N PP

N -> "company" | "school" | "room"
        | "school" | "woman" | "week"
        | "home" | "business"
        | "country" | PP

P -> "in" | "on" | "by"
        | "with" | "outside" | NP
```

Fig. 19: Grammar F

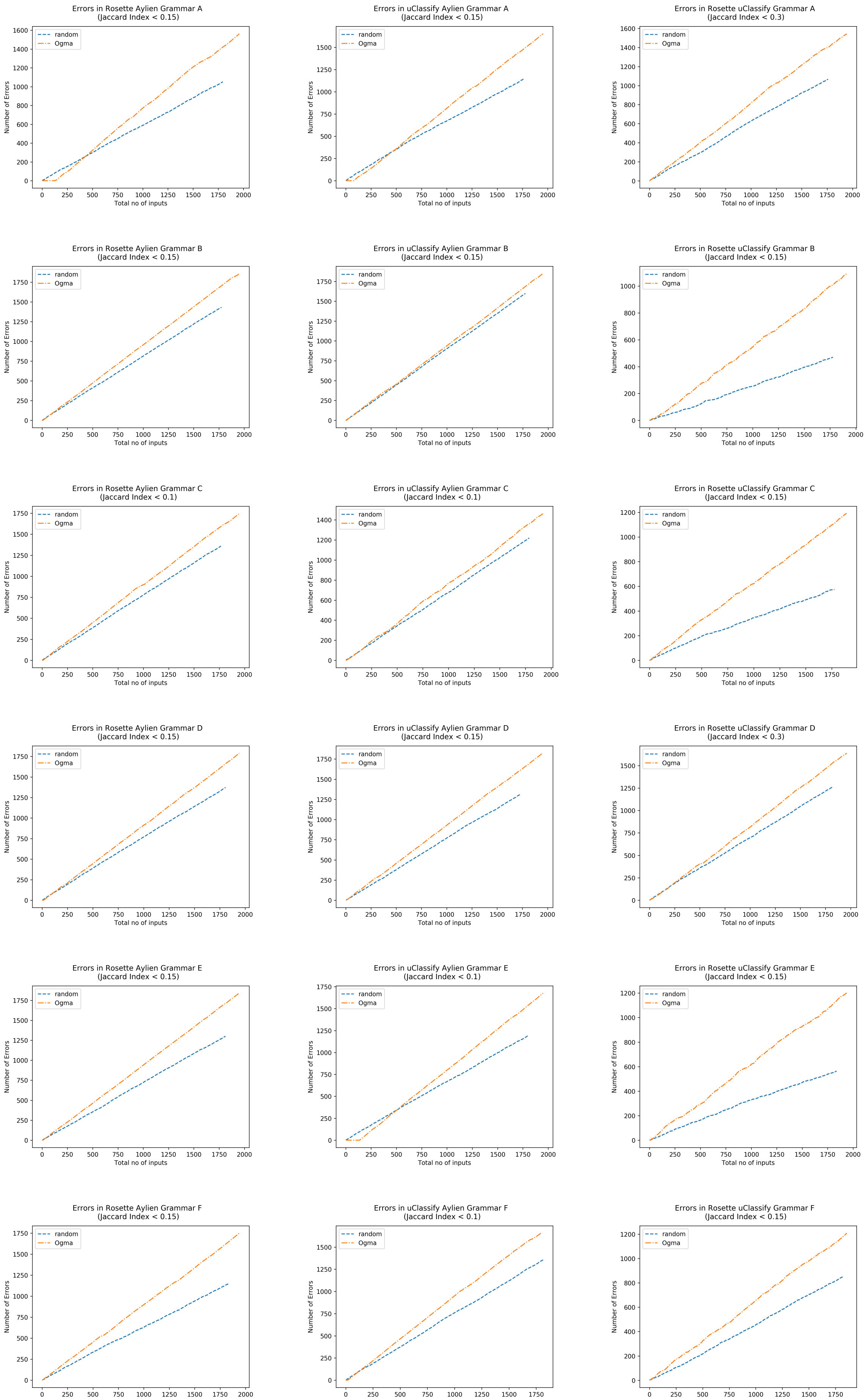


Fig. 20: Results with initial input being non-error inducing

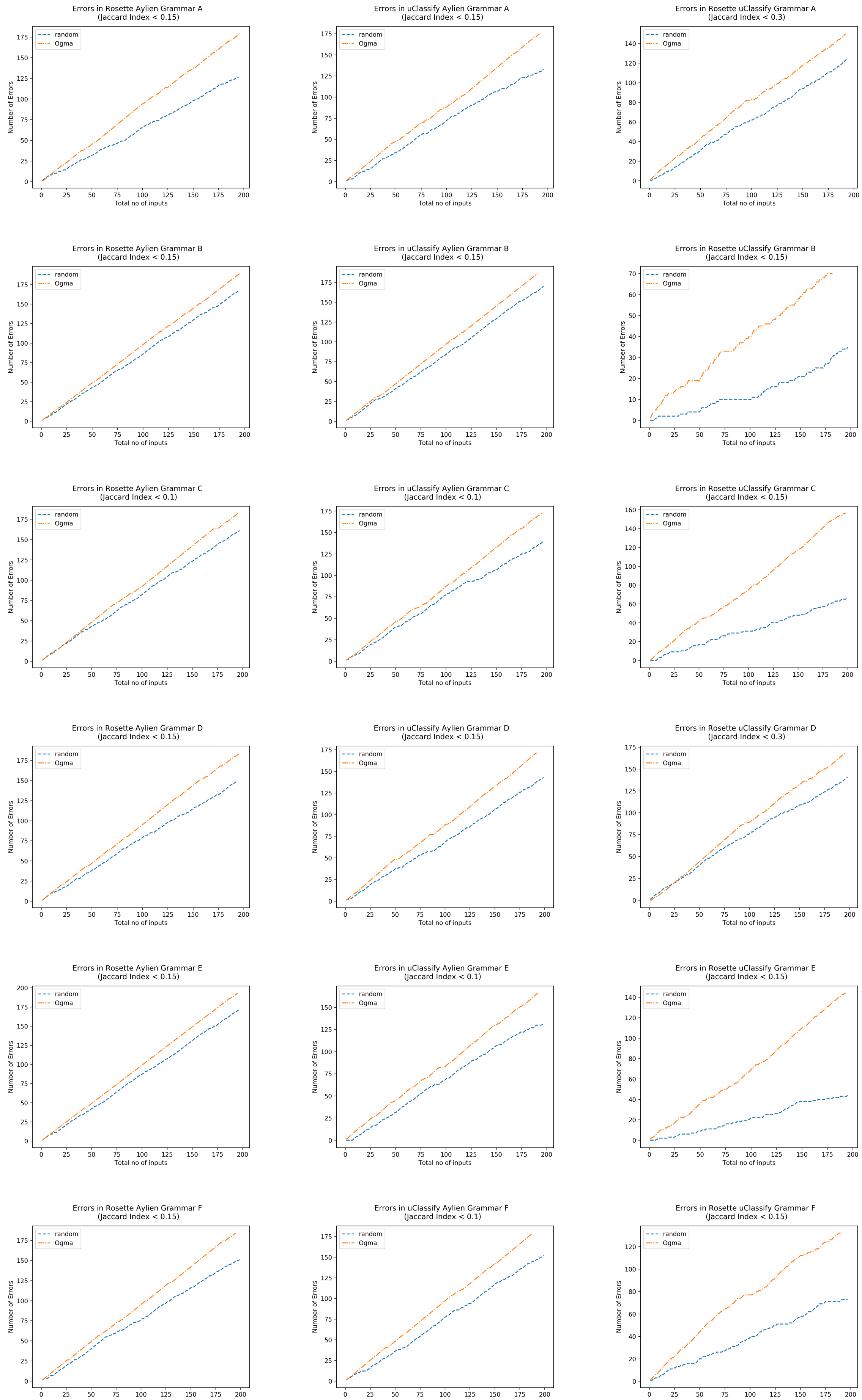


Fig. 21: Results with initial input being error inducing