Beyond Accuracy: Behavioral Testing of NLP Models with C\textsc{heckList} (Extended Abstract*)

Marco Tulio Ribeiro\textsuperscript{1}, Tongshuang Wu\textsuperscript{2}, Carlos Guestrin\textsuperscript{2}, Sameer Singh\textsuperscript{3}

\textsuperscript{1} Microsoft Research
\textsuperscript{2} University of Washington
\textsuperscript{3} University of California, Irvine
marcotcr@gmail.com, {wtshuang,guestrin}@cs.uw.edu, sameer@uci.edu

Abstract

Although measuring held-out accuracy has been the primary approach to evaluate generalization, it often overestimates the performance of NLP models, while alternative approaches for evaluating models either focus on individual tasks or on specific behaviors. Inspired by principles of behavioral testing in software engineering, we introduce CheckList, a task-agnostic methodology for testing NLP models. CheckList includes a matrix of general linguistic capabilities and test types that facilitate comprehensive test ideation, as well as a software tool to generate a large and diverse number of test cases quickly. We illustrate the utility of CheckList with tests for three tasks, identifying critical failures in both commercial and state-of-art models. In a user study, a team responsible for a commercial sentiment analysis model found new and actionable bugs in an extensively tested model. In another user study, NLP practitioners with CheckList created twice as many tests, and found almost three times as many bugs as users without it.

1 Introduction

One of the primary goals of training NLP models is generalization. Since testing “in the wild” is expensive and does not allow for fast iterations, the standard paradigm for evaluation is using train-validation-test splits to estimate the accuracy of the model, including the use of leader boards to track progress on a task [Rajpurkar et al., 2016]. While performance on held-out data is a useful indicator, held-out datasets are often not comprehensive, and contain the same biases as the training data [Rajpurkar et al., 2018], such that real-world performance may be overestimated [Patel et al., 2008]. Further, by summarizing the performance as a single aggregate statistic, it becomes difficult to figure out where the model is failing, and how to fix it [Wu et al., 2019].

A number of additional evaluation approaches have been proposed, such as evaluating robustness to noise [Belinkov and Bisk, 2018] or adversarial changes [Ribeiro et al., 2018; Iyyer et al., 2018], fairness [Prabhakaran et al., 2019], logical consistency [Ribeiro et al., 2019], explanations [Ribeiro et al., 2016], diagnostic datasets [Wang et al., 2019], and interactive error analysis [Wu et al., 2019]. However, these approaches focus either on individual tasks such as Question Answering or Natural Language Inference, or on a few capabilities (e.g. robustness), and thus do not provide comprehensive guidance on how to evaluate models. Software engineering research, on the other hand, has proposed a variety of paradigms and tools for testing complex software systems. In particular, “behavioral testing” (also known as black-box testing) is concerned with testing different capabilities of a system by validating the input-output behavior, without knowledge of the internal structure [Beizer, 1995]. While there are similarities, insights from software engineering are yet to be applied to NLP models.

In this work, we propose CheckList, a new evaluation methodology and accompanying tool for comprehensive behavioral testing of NLP models. CheckList guides users in what to test, by providing a list of linguistic capabilities, which are applicable to most tasks. To break down potential capability failures into specific behaviors, CheckList introduces different test types, such as prediction invariance in the presence of certain perturbations, or performance on a set of “sanity checks.” Finally, our implementation of CheckList includes multiple abstractions that help users generate large numbers of test cases easily, such as templates, lexicons, general-purpose perturbations, visualizations, and context-aware suggestions.

As an example, we CheckList a commercial sentiment analysis model in Figure 1. Potential tests are structured as a conceptual matrix, with capabilities as rows and test types as columns. As a test of the model’s Negation capability, we use a Minimum Functionality test (MFT), i.e. simple test cases designed to target a specific behavior (Figure 1A). We generate a large number of simple examples filling in a template (“I \{NEGATION\} \{POS_VERB\} the \{THING\}.”) with pre-built lexicons, and compute the model’s failure rate on such examples. Named entity recognition (NER) is another capability, tested in Figure 1B with an Invariance test (INV) – perturbations that should not change the output of the model. In this case, changing location names should not change sentiment. In Figure 1C, we test the model’s Vocabulary with a Directional Expectation test (DIR) – perturbations to the input with known expected results – adding negative phrases and checking that sentiment does not become more positive. As

*Extended version appears in the proceedings of Annual Meeting of the Association for Computational Linguistics (ACL) 2020.
### CheckList

Conceptually, users “CheckList” a model by filling out cells in a matrix (Figure 1), each cell potentially containing multiple tests. In this section, we go into more detail on the rows (capabilities), columns (test types), and how to fill the cells (tests). CheckList applies the behavioral testing principle of “decoupling testing from implementation” by treating the model as a black box, which allows for comparison of different models trained on different data, or third-party models where access to training data or model structure is not granted.

#### Capabilities

While testing individual components is a common practice in software engineering, modern NLP models are rarely built one component at a time. Instead, CheckList encourages users to consider how different natural language capabilities are manifested on the task at hand, and to create tests to evaluate the model on each of these capabilities. For example, the Vocabulary+POS capability pertains to whether a model has the necessary vocabulary, and whether it can appropriately handle the impact of words with different parts of speech on the task. For Sentiment, we may want to check if the model is able to identify words that carry positive, negative, or neutral sentiment, by verifying how it behaves on examples like “This was a good flight.” For QQP, we might want the model to understand when modifiers differentiate questions, e.g. accredited in (“Is John a teacher?”,”Is John an accredited teacher?”). For MC, the model should be able to relate comparatives and superlatives, e.g. (Context: “Mary is smarter than John.”, Q: “Who is the smartest kid?”), A: “Mary”.

We suggest that users consider at least the following capabilities: Vocabulary+POS (important words for the task), Taxonomy (synonyms, antonyms, etc), Robustness (to typos, irrelevant changes, etc), NER (appropriately understanding named entities), Fairness, Temporal (understanding order of events), Negation, Coreference, Semantic Role Labeling (understanding roles such as agent, object, etc), and Logic (ability to handle symmetry, consistency, and conjunctions). We will provide examples of how these capabilities can be tested in Section 3. This listing of capabilities is not exhaustive, but a starting point for users, who should also come up with additional capabilities that are specific to their task or domain.

#### Tests Types

We prompt users to evaluate each capability with three different test types (when possible), represented by the columns in the matrix in Figure 1. A Minimum Functionality test (MFT), inspired by unit tests in software engineering, is a collection of simple examples (and labels) to check a behavior within a capability. MFTs are similar to creating small and focused testing datasets, and are particularly useful for detecting when models use shortcuts to handle complex inputs without actually mastering the capability. The Vocabulary+POS examples in the previous section are all MFTs.

We also introduce two additional test types inspired by software metamorphic tests [Segura et al., 2016]. An Invariance test (INV) is when we apply label-preserving perturbations to inputs and expect the model prediction to remain the same. Different perturbation functions are needed for different capabilities, e.g. changing location names for the NER capability for Sentiment (Figure 1B), or introducing typos to test the Robustness capability. A Directional Expectation test (DIR) is similar, except that the label is expected to change in a certain way. For example, we expect that sentiment will not become more positive if we add “You are lame,” to the end of tweets directed at an airline (Figure 1C). The expectation may also be a target label, e.g. replacing locations in only one of the
questions in QQP, such as (“How many people are there in England?”), “What is the population of [England-Turkey]?”), ensures that the questions are not duplicates. INVs and DIRs allow us to test models on unlabeled data – they test behaviors that do not rely on ground truth labels, but rather on relationships between predictions after perturbations are applied (invariance, monotonicity, etc).

Generating Test Cases at Scale Users can create test cases from scratch, or by perturbing an existing dataset. Starting from scratch makes it easier to create a small number of high-quality test cases for specific phenomena that may be underrepresented or confounded in the original dataset. Writing from scratch, however, requires significant creativity and effort, often leading to tests that have low coverage or are expensive and time-consuming to produce. Perturbation functions are harder to craft, but generate many test cases at once. To support both these cases, we provide a variety of abstractions that scale up test creation from scratch and make perturbations easier to craft. These include templates, RoBERTa [Liu et al., 2019] suggestions, lexicons, perturbations, and visualizations. Code is available at https://github.com/marcotcr/checklist.

3 Testing SOTA Models with CheckList

We CheckList the following commercial Sentiment analysis models via their paid APIs: Microsoft’s Text Analytics ( ), Google Cloud’s Natural Language ( ), and Amazon’s Comprehend ( ). We also CheckList BERT-base ( ) and RoBERTa-base (RoB) [Liu et al., 2019] finetuned on SST-2 (acc: 92.7% and 94.8%) and on the QQP dataset (acc: 91.1% and 91.3%). For MC, we use a pretrained BERT-large finetuned on SQuAD [Wolf et al., 2019], achieving 93.2 F1.

Sentiment Analysis Since social media is listed as a use case for these commercial models, we test on that domain and use a dataset of unlabeled airline tweets for INV1 and DIR perturbation tests. We create tests for a broad range of capabilities, and present subset with high failure rates in Table 1. The Vocab+POS MFTs are sanity checks, where we expect models to appropriately handle common neutral or sentiment-laden words. and RoB do poorly on neutral predictions (both are trained on binary labels). Surprisingly, and fail (7.6% and 4.8%) on sentences that are clearly neutral, with also failing (15%) on non-neutral sanity checks (e.g. “I like this seat.”). In the DIR tests, the sentiment scores from and are frequently (12.6% and 12.4%) go down considerably when negative phrases are added, or up (34.6%) for negative ones (e.g. “You are lame.”).

All models are sensitive to addition of random (not adversarial) shortened URLs or Twitter handles (e.g. 24.8% of predictions change), and to name changes, such as locations ( : 20.8%, : 14.8%) or person names ( : 15.1%, : 9.1%). None of the models do well in tests for the Temporal, Negation, and SRL capabilities. Failures on negations as simple as “The food is not poor.” are particularly notable, e.g. (54.2%) and (29.4%). The failure rate is near 100% for all commercial models when the negation comes at the end of the sentence (e.g “I thought the plane would be awful, but it wasn’t”), or with neutral content between the negation and the sentiment-laden word.

Commercial models do not fail simple Fairness sanity checks such as “I am a black woman.” (template: “I am a [PROTECTED] [NOUN]”). which are predicted as neutral. Similar to software testing, absence of test failure does not imply that these models are fair – just that they are not unfair enough to fail these simple tests. On the other hand, always predicts negative when [PROTECTED] is black, atheist, gay, and lesbian, while predicting positive for Asian, straight, etc.

With the exception of tests that depend on predicting “neutral”, and RoB did better than all commercial models on almost every other test. This is a surprising result, since the commercial models list social media as a use case, and are under regular testing and improvement with customer feedback, while and RoB are research models trained on the SST-2 dataset (movie reviews). Finally, and RoB fail simple negation MFTs, even though they are fairly accurate (91.5%, 93.9%, respectively) on the subset of the SST-2 validation set that contains negation in some form (18% of instances). By isolating behaviors like this, our tests are thus able to evaluate capabilities more precisely, whereas performance on the original dataset can be misleading.

Question Paraphrasing and Machine Comprehension While and RoB surpass human accuracy on both QQP and MC, a subset of tests presented in the full paper [Ribeiro et al., 2020] indicate that these models are far from solving these tasks, and are likely relying on shortcuts for their high accuracy. We omit examples here due to space.

Discussion We applied the same process to very different tasks, and found that tests reveal interesting failures on a variety of task-relevant linguistic capabilities. While some tests are task specific (e.g. positive adjectives), the capabilities and test types are general; many can be applied across tasks, as is or with minor variation (changing named entities yields different expectations depending on the task). The selection of tests in [Ribeiro et al., 2020] illustrates the benefits of systematic testing in addition to standard evaluation. These tasks may be considered “solved” based on benchmark accuracy results, but the tests highlight various areas of improvement – in particular, failure to demonstrate basic skills that are de facto needs for the task at hand (e.g. basic negation). Even though some of these failures have been observed by others, we believe the majority are not known to the community, and that comprehensive and structured testing will lead to avenues of improvement in these and other tasks.

4 User Evaluation

The failures discovered in the previous section demonstrate the usefulness and flexibility of CheckList. In this section, we further verify that CheckList leads to insights both for users who already test their models carefully and for users with little or no experience in a task.
### Table 1: A selection of tests for sentiment analysis. All examples (right) are failures of at least one model.

<table>
<thead>
<tr>
<th>Test TYPE and Description</th>
<th>Failure Rate (%)</th>
<th>Example test cases &amp; expected behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MFT:</strong> Short sentences with neutral adjectives and nouns</td>
<td>0.0</td>
<td>The company is Australian. neutral That is a private aircraft. neutral</td>
</tr>
<tr>
<td><strong>MFT:</strong> Short sentences with sentiment-laden adjectives</td>
<td>4.0</td>
<td>That cabin crew is extraordinary. pos I despaired that aircraft. neg</td>
</tr>
<tr>
<td><strong>DIR:</strong> Add positive phrases, fails if sent. goes down by &gt; 0.1</td>
<td>12.6</td>
<td>@SouthwestAir Great trip on 2672 yesterday... You are extraordinary.</td>
</tr>
<tr>
<td><strong>DIR:</strong> Add negative phrases, fails if sent. goes up by &gt; 0.1</td>
<td>0.8</td>
<td>@USAirways your service sucks. You are lame.</td>
</tr>
<tr>
<td><strong>INV:</strong> Add randomly generated URLs and handles to tweets</td>
<td>9.6</td>
<td>@JetBlue that selfie was extreme. @p0QDK INV</td>
</tr>
<tr>
<td><strong>INV:</strong> Switching locations should not change predictions</td>
<td>7.0</td>
<td>@JetBlue I want you guys to be the first to fly to # Cuba Canada...</td>
</tr>
<tr>
<td><strong>INV:</strong> Switching person names should not change predictions</td>
<td>2.4</td>
<td>@VirginAmerica I miss the #n robust in San Jose Denver INV</td>
</tr>
<tr>
<td><strong>MFT:</strong> Sentiment change over time, present should prevail</td>
<td>41.0</td>
<td>I used to hate this airline, although now I like it. pos In the past I thought this airline was perfect, now I think it is creepy. neg</td>
</tr>
<tr>
<td><strong>MFT:</strong> Negated negative should be positive or neutral</td>
<td>18.8</td>
<td>The food is not poor. pos or neutral</td>
</tr>
<tr>
<td><strong>MFT:</strong> Negation of negative at the end, should be pos. or neut.</td>
<td>100.0</td>
<td>I thought the plane would be awful, but it wasn’t. pos or neutral</td>
</tr>
<tr>
<td><strong>MFT:</strong> Negated positive with neutral content in the middle</td>
<td>98.4</td>
<td>I wouldn’t say, given it’s a Tuesday, that this pilot was great. neg</td>
</tr>
<tr>
<td><strong>MFT:</strong> Author sentiment is more important than of others</td>
<td>45.4</td>
<td>Some people think you are excellent, but I think you are nasty. neg Some people hate you, but I think you are exceptional. pos</td>
</tr>
<tr>
<td><strong>MFT:</strong> Parsing sentiment in question, “no” form</td>
<td>96.8</td>
<td>Do I think the pilot was fantastic? No. neg Do I think this company is bad? No. pos or neutral</td>
</tr>
</tbody>
</table>

**CheckListing a Commercial System** We approached the team responsible for the sentiment analysis model sold as a service by Microsoft (on Table 1). Since it is a public-facing system, the model’s evaluation procedure is more comprehensive than research systems, including focused benchmarks built in-house (e.g. negations, emojis) and many cycles of bug discovery (either internally or through customers) and subsequent fixes. Our goal was to verify if CheckListing would add value even in a situation like this, where models are already tested extensively with current practices. After a 5-hour session, the results were very encouraging: the team stated that (1) they tested capabilities they had not considered, (2) they tested capabilities that they had considered but are not in the benchmarks, and (3) even capabilities for which they had benchmarks (e.g. negation) were tested much more thoroughly and systematically with CheckListing. This session, coupled with the variety of bugs we found for three separate commercial models in Table 1, indicates that CheckListing is useful even in pipelines that are stress-tested and used in production.

**User Study: CheckListing MFTs** We conduct a user study to further evaluate different subsets of CheckListing in a more controlled environment, and to verify if even users with no previous experience in a task can gain insights and find bugs in a model. We found that participants from industry and academia testing checked finetuned on QQP created more than 2x more tests with an order of magnitude more test cases, and found almost 3x more bugs with CheckListing than without it (details in [Ribeiro et al., 2020]).

### 5 Conclusion
While useful, accuracy on benchmarks is not sufficient for evaluating NLP models. We propose CheckListing, a model-agnostic and task-agnostic testing methodology that tests individual capabilities of the model using three different test types. To illustrate its utility, we highlight significant problems at multiple levels in the conceptual NLP pipeline for models that have “solved” existing benchmarks on three different tasks. Further, CheckListing reveals critical bugs in commercial systems developed by large software companies, indicating that it complements current practices well. Tests created with CheckListing can be applied to any model, making it easy to incorporate in current benchmarks or evaluation pipelines. Our user studies indicate that CheckListing is easy to learn and use, and helpful both for expert users who have tested their models at length as well as for practitioners with little experience in a task. CheckListing is open source, and available at [https://github.com/marcotcr/checklist](https://github.com/marcotcr/checklist).
References


