Predicting temporal performance drop of deployed production spoken language understanding models

Quynh Do, Judith Gaspers, Daniil Sorokin, Patrick Lehnen

Amazon Alexa AI, Germany

{doquynh,gaspers,dsorokin,plehnen}@amazon.de

Abstract

In deployed real-world spoken language understanding (SLU) applications, data continuously flows into the system. This leads to distributional differences between training and application data that can deteriorate model performance. While regularly retraining the deployed model with new data helps mitigating this problem, it implies significant computational and human costs. In this paper, we develop a method, which can help guiding decisions on whether a model is safe to keep in production without notable performance loss or needs to be retrained. Towards this goal, we build a performance drop regression model for an SLU model that was trained offline to detect a potential model drift in the production phase. We present a wide range of experiments on multiple real-world datasets, indicating that our method is useful for guiding decisions in the SLU model development cycle and to reduce costs for model retraining.

1. Introduction

In real-world machine learning applications the incoming data in the production phase may distributionally differ from the training and test data used offline at the development phase. This violates a fundamental assumption in machine learning that the training and test sets are drawn from the same underlying distribution, leading to a potential performance drop at production time compared to the offline evaluation. To tackle this issue, one can train a model on new data and deploy it to customers regularly. However, re-training and deploying new models in a large-scale setting implies significant costs for human and computational resources. In addition, it is unclear how to pick the cycle time for releasing new models, as longer cycles decrease computational costs but increase the risk of performance drops due to distributional differences in the data, which, in turn, may result in a drop in user satisfaction with the application.

In this paper, we develop a method to determine if the deployed model faces a potential performance drop and needs to be updated with new data or if it is safe to keep. Our proposed solution is computationally cheap and can be applied to check the production model at regular intervals.

Since data annotations are costly and take time to collect, the need of making model update decisions without knowing the test set's actual labels has spurred research on unsupervised drift detection. Previous work aimed at identifying significant differences between unlabeled data from reference and detection time periods [1, 2]. In our method, we predict the performance drop of an SLU model on an unlabeled test set at production time using the historical performance of the model. We argue that using a performance drop as the signal for model check is more direct and is easier to interpret than the difference between unlabeled datasets.

Spoken Language Understanding (SLU), which plays a crucial role in the development of voice-controlled devices like Alexa or Siri, is often composed of two sub-tasks: intent classification (IC), which identifies the user intent, and slot filling (SF), which extracts semantic constituents. For instance, if a user requests "play Hello by Adele", IC should identify *PlayMusic* as the intent, while SF should classify "play", "Hello", "by", "Adele" as *Other*, *SongName*, *Other* and *ArtistName*, respectively. In real-world SLU applications, data continuously flows into a system. The corresponding data distribution is affected as i) customer usage patterns change over time, when, for example, different songs become popular or new movies are released, and ii) new functionalities are added. Over time, if the offline training data is not regularly updated, its data distribution may drift more and more apart from the data flowing into the deployed model, which potentially results in a performance drop.

In this paper, we build a linear regression model to predict performance drop when an SLU model trained offline is deployed in production. To the best of our knowledge, this has not yet been studied in the literature. This is a challenging problem due to multiple difficulties: the requirement of a large dataset over a long time period, the complexity of SLU models, and the absence of features that can directly reflect the possible performance drop to be used in a machine learning model. Our main contribution is an approach to detect the SLU models that do not face a potential performance drop in production and are safe to keep. We use a performance drop regression trained on historical data to achieve this and show a positive impact of our approach on an SLU model development cycle through an extensive evaluation on multiple real-world datasets.

2. Related Work

Supervised methods for drift detection record a set of metrics on labelled data and correlate them with changes in model performance. They need labelled data for the reference and detection datasets to compute the metrics and the performance. This data is used to learn a mapping from the metrics to the model performance drop [3].

Unsupervised drift detection operates in the absence of labelled data by comparing data distributions in the reference and detection data windows [1]. These methods can only indirectly detect a model performance drop. In [4], the Kolmogorov-Smirnov test is used to compare historic and new data to detect data drift. $\mathscr{A}$-distance shows how difficult it is to differentiate between two domains and is used in [5] for detecting domain changes for named entity recognition and document classification. In contrast to unsupervised drift detection, our method is able to estimate performance drop directly and, after being trained, it does not require the existence of the reference and target datasets' actual labels.

Predicting performance drop under domain shift with a regression model from an ensemble of metrics is the subject of [3]. They propose a comprehensive set of metrics: (1) $\mathscr{H}$-divergence based metrics [6], (2) confidence-based metrics, which measure a drop in confidence for model predictions on the target data and (3) reverse classification accuracy. The metrics are an input to a regression model, which is fit on a dataset of pairs of training and target data, where the model performance is known. The experiments on POS tagging and sentiment data in [3] show that confidence-based metrics are good predictors of the performance drop and are robust to adversarial shifts. Although our goal is similar to [3], we focus on the drop occurring in the same domain but over different time periods. In particular, instead of working with two data distributions representing the source and the target domains, we deal with three data distributions representing the training set, the reference and the detection test sets. Moreover, we go beyond the performance drop prediction by studying its application for improving model development cycle.

3. SLU Model

Following a common trend in current SLU approaches [7, 8, 9, 10], we model IC and SF jointly using a pre-trained contextual language model i.e. BERT. In particular, our model consists of a BERT encoder, an intent classifier and a slot classifier. The BERT model produces two outputs at sentence level and token level which are used as inputs for the intent and slot classifiers, respectively. In our model, the intent classifier is a standard feed-forward network including two dense layers. Meanwhile, the slot classifier uses a CRF layer on top of two dense layers. During the training a cross-entropy loss for IC and a CRF loss for SF are optimized jointly with balanced weights.

Following [9], to evaluate SLU performance, we use a semantic error rate, which measures IC and SF jointly and is defined as follows:

$$SemER = \frac{\#\text{(slot+intent errors)}}{\#\text{slots in reference + 1}} \tag{1}$$

4. Predicting Performance Drop for SLU

In this section we define the performance drop prediction task for SLU, and present our proposed model to tackle the task.

4.1. Formal Task Definition

Given a dataset $\mathbf{D}$, in which each instance x contains the following information: (i) a reference test set T_{ref}, (ii) a detection test set T_{detect}, (iii) a training set L, and (iv) a model M trained on L, our goal is to *learn a function g mapping each instance $x \in \mathbf{D}$ to a real value representing the model performance drop* of M on T_{detect} compared to T_{ref}:

$$\begin{aligned} g &: \mathbf{D} \to \mathbb{R} \\ x &\mapsto \triangle E_M(T_{detect}, T_{ref}) \approx E_M(T_{detect}) - E_M(T_{ref}) \end{aligned} \tag{2}$$

where $E_M(D)$ is an error rate metric evaluating the performance of M on D. Note that, the two test sets are labeled during the training phase and unlabeled during the prediction phase.

4.2. Regression Model

We use Linear Regression to fit g:

$$g(x) \approx F(x)^T w + \varepsilon \tag{3}$$

where F is a feature extraction function, T denotes the transpose, w is the weight vector and ε is the disturbance term. g is optimized by minimizing the mean squared error between the actual and the predicted drops.

4.3. Features

Two important concepts used in our proposed features are the differences between two datasets, and the representations of utterances and tokens.

Differences between two sets of vector representations Given an embedding function ϑ which maps each data item e e.g. an utterance or token, to a real vector in $\mathbb{R}^n$, a set D of data items can be represented as: (i) the mean representation of its items: $\text{Rep}^{\vartheta}(D) = \frac{1}{|D|}\sum_{e\in D}\vartheta(e)$, or (ii) a matrix of size $|D| \times n$: $[\vartheta(e)|e \in D]$.

To compute the differences between two sets D_1 and D_2, we use two scores as follows:

i) Euclidean distance between two mean representations:

$$\text{Euc}^{\vartheta}(D_1, D_2) = \text{d}(\text{Rep}^{\vartheta}(D_1), \text{Rep}^{\vartheta}(D_2)) \tag{4}$$

ii) Average p_{score} of student t-test across n dimension of the matrix representations:

$$\Delta_p^{\vartheta}(D_1, D_2) = \frac{1}{n}\sum_{i=\overline{1,n}} p_{\text{score}}(dim_i([\vartheta(e)|e \in D_1]), \quad dim_i([\vartheta(e)|e \in D_2])) \tag{5}$$

where $dim_i(A)$ is the dimension i of matrix A.

Utterance and token representations To map each utterance and token to a vector representation, we use two types of utterance and token embeddings: (i) *BERT*: Sentence and token embeddings produced by a general pre-trained BERT model, and (ii) *SLU*: Sentence and token embeddings, which are the outputs produced by the encoder of our trained BERT-based SLU model.

Feature list We assume that the model performance drop is caused by the differences between the three datasets (T_{ref}, T_{detect}, L), and the differences between model M's understanding of T_{ref} and T_{detect}. Therefore, we extract the following features to learn performance drops:

i) Difference between the mean representations:

$$Euc^{\vartheta}(L, T_{detect}) - Euc^{\vartheta}(L, T_{ref}) \text{ for } \vartheta \in \{BERT, SLU\}, \tag{6}$$

where ϑ is the sentence embedding.

ii) Statistical difference between the representations:

$$\Delta_p^{\vartheta}(L, D_{detect}) - \Delta_p^{\vartheta}(L, D_{ref}) \text{ for } \vartheta \in \{BERT, SLU\}, \tag{7}$$

where ϑ is the sentence embedding.

iii) Difference between the predicted IC and SF tag groups:

$$Euc^{\vartheta}\Big(\frac{1}{|G^{tsk}|}\sum_{t\in G^{tsk}} M^{tsk}(t, T_{detect}), \frac{1}{|G^{tsk}|}\sum_{t\in G^{tsk}} M^{tsk}(t, T_{ref})\Big), \tag{8}$$

where $\vartheta \in \{BERT, SLU\}$ is the sentence embedding, $tsk \in \{IC, SF\}$, G^{IC} is the IC tag set, $M^{IC}(t, D)$ is the set of utterances in D which are assigned tag t by model M, G^{SF} is the SF tag set, and $M^{SF}(t, D)$ is the set of tokens in D which are assigned tag t by model M.

iv) Reference vs. detection binary classification: For each sentence representation in $\{BERT, SLU\}$ we build a Logistic Regression model to classify each utterance in T_{ref} and T_{detect}

into reference and detection windows. The cross-validation 5-fold accuracy scores of the binary classification models are used as our feature for performance drop prediction.

In addition, the following features are each computed for $window \in \{detect, ref\}$.

v) Intent confidence scores:

$$\frac{1}{|G^{IC}|} \sum_{t \in G^{IC}} \left(\frac{1}{|M^{IC}(t, T_{window})|} \sum_{u \in M^{IC}(t, T_{window})} conf^{IC}(M, u) \right) \quad (9)$$

where $conf^{ic}(M,u)$ is the confidence score of M on u's IC prediction.

vi) SF confidence scores:

$$\frac{1}{|G^{SF}|} \sum_{t \in G^{SF}} \left(\frac{1}{|M^{SF}(t, T_{window})|} \sum_{u \in M^{SF}(t, T_{window})} conf^{SF}(M, u) \right) \quad (10)$$

where $conf^{sf}(M,u)$ is the confidence score of M on u's SF prediction. Here, the SF prediction is computed using the Viterbi algorithm.

5. Detecting Low-Risk SLU Models

Aiming to provide help in making decisions regarding model retraining in the SLU development cycle, we apply our performance drop regression model to detect low-risk SLU models.

5.1. Problem Definition

Given a dataset **D** as described in Sec. 4.1, and an error rate metric $0 \leq E \leq 1$, we define *a significant drop threshold* $-1 \leq \alpha \leq 1$, so that an SLU model M *is low-risk* if its actual drop is not larger than the threshold: $E_M(T_{detect}) - E_M(T_{ref}) \leq \alpha$.

The task of detecting low-risk SLU models aims at judging whether a model can be kept with low risk of a performance drop or not by automatically predicting its performance drop without knowing the test sets' ground truth labels.

Since predicting SLU model performance drop is a very challenging task as outlined in the introduction, when using the learned regression function g to predict low-risk points, we add an error tolerance hyper-parameter β to the significant drop threshold. In particular, a model *is predicted as low-risk* if: $g(T_{ref}, T_{detect}, L, M) \leq \alpha - \beta$. In our experiments, we set $\beta = 0.01$ (1 error rate point), but we truncate the value of $\alpha - \beta$ so that it will never be larger than 0.

5.2. Evaluation

Given N SLU models, N_s is the number of actual low-risk models which have $E_M(T_{detect}) - E_M(T_{ref}) \leq \alpha$. By using the learned regression function g, Q models are predicted as low-risk. However, only Q_s of the predicted ones are actually low-risk. To evaluate g, we compute the two metrics: i) precision: $P = \frac{Q_s}{Q}$ and ii) recall: $R = \frac{Q_s}{N_s}$.

Let us consider the use case in which we only update the models which are *not* predicted as low-risk. Here, we argue that precision is more important than recall. With lower precision, more risky models are missed leading to a negative impact on production model performance, and thus potentially yielding a drop in customer satisfaction with the device. In contrast, a low recall has an unexpected influence on model updating cost. We consider customer satisfaction as the more important target. Thus, in this work, a regression model g is considered as good if it has a high Precision and an acceptable Recall.

We use the precision obtained by picking low-risk models randomly as our baseline: $B_p = \frac{N_s}{N}$.

6. Experiments

In this section, we first describe the SLU model settings used for all experiments. Subsequently, we first present an evaluation of our performance drop regression model with application to detecting low-risk models. Afterwards, we present an evaluation w.r.t. the impact of our method on the model development cycle.

SLU models We use a pre-trained TinyBERT [11] model for German, which is of size 312, and first-pooling for sentence representation. Each of our classifiers has 2 dense layers of size 256 with gelu activation. The dropout values used in IC and SF classifiers are 0.5 and 0.2, respectively. For optimization, we use the Adam optimizer with learning rate 0.5 and a Noam learning rate scheduler. We trained an SLU model on the supervised training data for each data period, which is available for the corresponding experiments.

6.1. Predicting Performance Drop and Low-Risk Models

We first evaluate our regression model directly on a drop prediction task and then apply it to detect low-risk models.

Datasets We extracted data samples from a German commercial SLU system. All data was de-identified and are annotated with intents and slots. We include data from four domains, namely Music, Shopping, Video and SmartHome. The data capture a long time range, i.e. for each domain we include data from 52 subsequent model cycles spanning three weeks each. A model cycle at a time step t is composed of: (a reference test set T^t_{ref}, a detection test set T^t_{detect}, a training dataset L^t, a model M^t trained on L^t). L^t and T^t_{ref} are the training and test data available at t, while T^t_{detect} is the test data available at $t+1$ which is the production time period of M^t.

Results Tab. 1 shows the 5-fold cross-validation results for predicting performance drop using our proposed regression model. *First*, it can be seen that the mean absolute error $\mathbf{MAE_{all}}$ depends on the fluctuation range of the actual performance drops. When the actual drops fluctuate widely e.g. in Video and Shopping, we observe the highest mean absolute errors. In contrast, on Music which has the smallest range of actual performance drops, the regression model performed best with $\mathbf{MAE_{all}}$ of only 0.88. *Second*, the proposed regression model performs better on the data instances having actual drops ≤ 0 than for the ones with actual drops > 0 on 3 over 4 datasets ($\mathbf{MAE_{pos.}} > \mathbf{MAE_{neg.}}$). This is most likely due to the majority of the negative drops on our datasets.

To gain a better interpretation of the regression model's effectiveness, we evaluate it on the application of detecting low-risk SLU models, i.e. models which can likely be kept in production without performance loss. In Tab. 2, we present the 5-fold cross-validation results. Overall, the regression model outperforms the baseline in 10 out of 12 test cases. It proves that the regression can learn useful information about performance drop. For some use cases, especially on Music, we can reach very good precision scores while maintaining acceptable recall. We notice that the precision is lower when we reduce the significant drop threshold α. The precision decreases by more then 20 p.p. on all datasets except Music. Noticeably, both of the two failed test cases in

Dataset	max	min	$\mathrm{MAE}_{\mathrm{all}}$	$\mathrm{MAE}_{\mathrm{pos.}}$	$\mathrm{MAE}_{\mathrm{neg.}}$
Music	0.0280	-0.0285	0.0088	0.01018	0.0079
Shopping	0.0409	-0.0612	0.0188	0.0210	0.0171
Video	0.1001	-0.0748	0.0299	0.0291	0.0310
SmartHome.	0.0330	-0.0399	0.0122	0.0181	0.0090

Table 1: *5-fold cross-validation results on predicting performance drop:* ***max***, ***min***, $\mathbf{MAE_{all}}$, $\mathbf{MAE_{pos.}}$, $\mathbf{MAE_{neg.}}$ *are the maximum and minimum of the actual performance drops, the Mean Absolute Error over all data instances, the instances with actual drop* > 0 *and the ones with actual drop* ≤ 0*, respectively.*

Dataset	α	B_p	Recall	Precision
Music	0.02	96.15	66.67	**100.00**
Music	0.01	88.46	69.56	**94.12**
Music	0.005	76.92	45.00	**94.74**
Shopping	0.02	80.64	69.05	**90.63**
Shopping	0.01	71.15	70.27	**81.25**
Shopping	0.005	63.46	45.45	**71.42**
Video	0.02	73.01	48.71	**82.60**
Video	0.01	61.54	46.88	**65.23**
Video	0.005	53.84	35.71	**55.55**
SmartHome	0.02	96.15	72.00	**97.30**
SmartHome	0.01	**84.62**	68.18	81.08
SmartHome	0.005	**78.85**	29.27	63.15

Table 2: *Cross-validation 5 folds results on detecting low-risk models:* α *and* B_p *are the significant drop threshold, precision baseline, respectively.*

Tab. 2 belong to HomeAutomation with low α. This suggests that it is still very challenging to predict small drops.

To determine the utility of our combination of features, we conducted ablation studies with subgroups of features, i.e. we removed certain (groups of) features and re-ran the experiments. The results confirmed that the combination of all proposed features gives the best results, and that among the tested feature groups in particular SLU representations play a crucial role. That is, when removing corresponding features the average precision is largely reduced.

6.2. Impact on Model Development Cycle

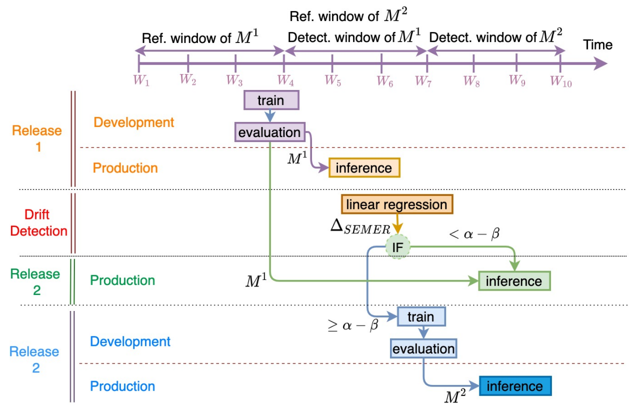

Figure 1: *SLU model development cycle.*

As our ultimate goal is to determine whether predicting performance drop can help improving the model development cycle, we conduct an experiment on making model update decisions based on the proposed performance drop regression model. In particular, given a consecutive series of SLU model cycles, for each detected low-risk model, we make a decision to reuse it for the next development cycle without any model updating. This decision is considered as correct if the model has comparable performance to an updated model on the test set of the next cycle (see Fig. 1 for illustration).

Datasets Based on the same data source as described in Sec. 6.1, we create a larger dataset for the Music domain named Music$_{large}$. It contains 61 consecutive SLU model cycles spanning three weeks each. At a time step t, a model M^t trained on L^t, which contains the training data available at t and $t-1$ instead of only t as in Sec. 6.1. Meanwhile, T^t_{ref} and T^t_{detect} are the test data available at t and $t+1$, respectively.

Evaluation Similarly to Sec. 5, we use the significant drop threshold α and the error tolerance hyper-parameter β to evaluate the decision making mechanism.

For each time step t, we make the decision to reuse the model for the next time step $t+1$ without any model updating if: $g(T^t_{ref}, T^t_{detect}, L^t, M^t) \leq \alpha - \beta$. Note that β is also fixed to 0.01 as in Sec. 5.

The decision is considered as correct if on the test set of the next cycle $(t+1)$, the actual performance that we will loose is less than the significant drop threshold: $E_{M^t}(T^{t+1}_{detect}) - E_{M^{t+1}}(T^{t+1}_{detect}) \leq \alpha$. To evaluate the decision making mechanism, we also compute the precision and recall scores, and compare with a precision baseline when the decisions are made randomly.

Dataset	α	B_p	Recall	Precision
Music$_{large}$	0.02	93.33	76.79	**95.55**
Music$_{large}$	0.01	76.67	80.43	**82.22**
Music$_{large}$	0.005	60.00	47.22	**77.27**

Table 3: *Cross-validation 5-fold results of making model update decision on Music*$_{large}$. α *and* B_p *are the significant drop threshold, precision baseline by making decision randomly, respectively.*

Results As can be seen in Tab. 3, the decision making mechanism aided by the regression model outperforms the random decision making baseline. This proves the positive impact of our performance drop regression model on the tested model development cycles. For example, our model can predict with a precision of 82.22% that 45 of 60 model updates can be avoided because of the estimated performance drop being smaller than 1% absolute with respect to the SEMER metric.

7. Conclusion

We have presented a method which can help making re-training decisions for deployed SLU models based on predicting performance drops when an SLU model trained offline is deployed in production. Experiments on real-world data indicate that without human annotation of the production traffic, our method can detect performance drops in production with high precision. Thus, it can be used to reduce model development cost, as, for instance, our model can predict with a precision of 82.22% that 45 of 60 model updates can be avoided because of the estimated performance drop being smaller than 1% absolute.

8. Acknowledgement

We thank Fabian Triefenbach and Tobias Falke for their comments and the three anonymous reviewers for their insightful suggestions and careful reading of the manuscript.

9. References

[1] R. N. Gemaque, A. França, J. Costa, R. Giusti, Eulanda, and M. dos Santos, "An overview of unsupervised drift detection methods," *WIREs Data Mining and Knowledge Discovery published by Wiley Periodicals LLC.*, 2020.

[2] C. H. Tan, V. C. S. Lee, and M. Salehi, "Online semi-supervised concept drift detection with density estimation," 2019. [Online]. Available: http://arxiv.org/abs/1909.11251

[3] H. Elsahar and M. Gallé, "To annotate or not? predicting performance drop under domain shift," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP).* Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 2163–2173.

[4] D. M. dos Reis, P. Flach, S. Matwin, and G. Batista, "Fast unsupervised online drift detection using incremental kolmogorov-smirnov test," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1545–1554.

[5] M. Dredze, T. Oates, and C. Piatko, "We're not in Kansas anymore: Detecting domain changes in streams," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Cambridge, MA: Association for Computational Linguistics, Oct. 2010, pp. 585–595.

[6] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, "Analysis of representations for domain adaptation," in *Advances in Neural Information Processing Systems*, B. Schölkopf, J. Platt, and T. Hoffman, Eds., vol. 19. MIT Press, 2007.

[7] Q. Chen, Z. Zhuo, and W. Wang, "BERT for Joint Intent Classification and Slot Filling," *arXiv preprint arXiv:1902.10909*, 2019.

[8] Q. N. T. Do and J. Gaspers, "Cross-lingual transfer learning for spoken language understanding," *Proceedings of the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2019. [Online]. Available: http://arxiv.org/abs/1904.01825

[9] J. Gaspers, Q. Do, and F. Triefenbach, "Data balancing for boosting performance of low-frequency classes in spoken language understanding," in *INTERSPEECH*, 2020.

[10] W. Xu, B. Haider, and S. Mansour, "End-to-end slot alignment and recognition for cross-lingual nlu," *arXiv:2004.14353*, 2020. [Online]. Available: https://arxiv.org/abs/2004.14353

[11] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling BERT for natural language understanding," *CoRR*, vol. abs/1909.10351, 2019. [Online]. Available: http://arxiv.org/abs/1909.10351