Learning Prediction Intervals for Model Performance

Benjamin Elder, Matthew Arnold, Anupama Murthi, Jiri Navratil
IBM T.J. Watson Research Center
benjamin.elder@ibm.com, marnold@us.ibm.com, anupama.murthi@ibm.com, jiri@us.ibm.com

Abstract

Understanding model performance on unlabeled data is a fundamental challenge of developing, deploying, and maintaining AI systems. Model performance is typically evaluated using test sets or periodic manual quality assessments, both of which require laborious manual data labeling. Automated *performance prediction* techniques aim to mitigate this burden, but potential inaccuracy and a lack of trust in their predictions has prevented their widespread adoption. We address this core problem of performance prediction uncertainty with a method to compute *prediction intervals* for model performance. Our methodology uses transfer learning to train an *uncertainty model* to estimate the uncertainty of model performance predictions. We evaluate our approach across a wide range of drift conditions and show substantial improvement over competitive baselines. We believe this result makes prediction intervals, and performance prediction in general, significantly more practical for real-world use.

Introduction

Knowing when a model's predictions can be trusted is one of the key challenges in AI today. From an operational perspective, understanding the quality of model predictions impacts nearly all stages in the model lifecycle, including pre-deploy testing, deployment, and production monitoring. From a social perspective, prediction trust impacts society's willingness to accept AI as it continues replacing human decision making in increasingly important roles.

Techniques such as *performance prediction* (Guerra, Prudêncio, and Ludermir 2008; Schat et al. 2020; Talagala, Li, and Kang 2019) strive to automatically predict the performance of a model with no human intervention. Unfortunately these techniques have not yet gained mainstream adoption, ironically enough, due to their potential unreliability and the resulting lack of trust in their predictions. Performance predictors are often surprisingly accurate when a base model is predicting on data similar to what it has already seen in training and test. However, it is well known that model behavior can be extremely difficult to predict on previously unseen data (Nguyen, Yosinski, and Clune 2015; Su, Vargas, and Sakurai 2019). It is not reasonable to expect a performance prediction algorithm to perfectly predict a base model's behavior in these scenarios. It is, however, reasonable to ask a performance predictor to quantify the uncertainty of its predictions so the application (or end user) can take appropriate precautions.

This paper describes a technique for computing *prediction intervals* on *meta-model* based performance predictions, to convey the degree to which the performance prediction should be trusted. Our technique uses *meta-meta-modeling* in a multi-task setting to train an *uncertainty model* to compute prediction intervals. Our approach makes no assumptions about the base model or performance predictor, and can easily be applied in other settings. The use of a separate *meta-meta-model* to perform the uncertainty quantification allows the simultaneous prediction of both aleatoric (data-driven) and epistemic (model-driven) uncertainty.

A key challenge for an uncertainty model is predicting the unseen - ie, data that is substantially different than anything the model has seen in train or test. A simple cross validation or leave-one-out training is unlikely to produce sufficient feature-space drift to be informative in this regard. Our technique trains for these extreme cases by (1) simulating various levels of drift ranging from mild to extreme, and (2) training on these drift scenarios using *external* data sets that are different from the base model training set but of the same modality. This approach enables the uncertainty model to learn how the performance predictor behaves in a variety of challenging scenarios, and uses this information to help predict risk when challenging scenarios arise in production. Due to its prevalence and commercial importance, we focus on tabular data, and therefore chose logistic regression and random forest base models.

We evaluate our uncertainty model on four different performance predictors and compare the uncertainty model against four different model-free baseline algorithms. In every scenario, our uncertainty model outperforms all baselines, often by a large margin. Even without the use of an uncertainty model, our approach of using drift simulation for calibration yielded significant improvements over traditional baselines.

Related Work

Prior work exists on performance prediction (Guerra, Prudêncio, and Ludermir 2008; Chen et al. 2019; Finn et al. 2019; Redyuk et al. 2019; Schat et al. 2020; Talagala, Li, and

Kang 2019), however, to the best of our knowledge, none that assigns uncertainty bounds to their predictions. The field of domain generalization also includes some related work, for which a good recent review can be found in (Hospedales et al. 2020). For example, (Li et al. 2018) use a cross-domain meta-learning approach to model training similar our procedure, but applied to the base classification problem.

Of relevance are numerous methods for estimating the uncertainty of predictions from machine learning models in general. Many of these, ranging from classical statistical methods to state-of-the-art deep learning (DL) models ((Gal and Ghahramani 2016; Kendall and Gal 2017; Koenker and Bassett 1978; Nix and Weigend 1994), also see (Khosravi et al. 2011) for a review), could be applied to meta-model based performance prediction.

There are well-established parametric methods for prediction intervals, see for example (Geisser 2017). Methods implicitly learning the error distribution are also available, for example by incorporating a feature-dependent variance into the loss function for iterative training procedures (Nix and Weigend 1994). Furthermore, there has been significant progress in constructing neural architectures which simultaneously output a classification and an uncertainty prediction (Brosse et al. 2020; Kabir et al. 2019; Khosravi, Nahavandi, and Creighton 2010; Malinin and Gales 2018).

Non-parametric methods such as the jackknife and bootstrap (Efron 1979; Efron and Gong 1983), and more recent variations (Lei et al. 2018; Vovk et al. 2018; Papadopoulos 2008; Vovk 2012; Vovk, Gammerman, and Shafer 2005) can estimate the uncertainty of a statistical prediction without assuming a particular error model, but rely on the assumption that the distribution of the unlabeled data is the same as the train data. Ensemble methods have been proposed for both traditional ML (Dietterich 2000; Kwok and Carter 1990) and DL models (Hansen and Salamon 1990; Lakshminarayanan, Pritzel, and Blundell 2017; Osband, Aslanides, and Cassirer 2018). The variance of the ensemble predictions can be used as a feature-dependent measure of uncertainty. We implement this strategy as one of our baselines. Bayesian approaches have been extended to non-parametric applications, including neural networks (Bishop 1997; Blundell et al. 2015; Neal et al. 2011) and a popular approximate version of Bayesian neural networks - the Monte-Carlo dropout approximation (Gal and Ghahramani 2015; Gal, Hron, and Kendall 2017) - also serves as a baseline in our work.

Method

Our approach estimates prediction intervals for the performance of a black-box classification model on a pool of unlabeled data. First, we use meta-modeling to predict the accuracy of the base classification model (*performance prediction*). Second, a pre-trained *uncertainty model* is used to estimate a prediction interval, which describes the probable range for the true value of the accuracy.

Performance Predictors

As its name suggests, *performance prediction* is the problem of estimating the value of a performance metric that a machine-learning model will achieve for a given pool of unlabeled data (referred to here as the production set). This work focuses on classification accuracy, but the same methods could be applied to other metrics such as the F1 score or the error of a regression model. We treat the base model as a black box from which only the vector of predicted class probabilities for each sample is available. We developed two types of performance predictors (four variants in all) which we will use as the basis for our prediction intervals.

The performance predictors that we use in this work each output a confidence score for each unlabeled data point in the production set. This score, between zero and one, is an estimate of the likelihood that the base model predicted the correct class label. For the purposes of computing an aggregate accuracy score for the production set, we take the average of these confidences. This confidence averaging produced better estimates of the accuracy than making binary correct/incorrect predictions for each sample.

The `confidence` predictor is a simple, binning-based procedure, which recalibrates the base model confidence score for the most likely class (Zadrozny and Elkan 2001). The values of this confidence on the test set are gathered into a histogram (binned in increments of 0.1), and the base model accuracy is computed for each bin. A performance prediction for a data point is given by the average accuracy of the bin that spans that point's base model confidence.

Our `meta-model` performance predictor uses its own model to predict data points that are likely to be mislabeled by the base model. Training data for the `meta-model` predictor is created by relabeling the test set with binary labels indicating whether the base model correctly classified each sample. The meta-model, which is an ensemble of a Gradient Boosting Machine (GBM) and a logistic regression model, classifies each sample as correct or incorrect, and the probability assigned to the "correct" class is returned as the performance predictor confidence score. Further details of the `meta-model` predictor implementation are provided in the supplementary material.

Uncertainty Model

We propose a new approach for computing prediction intervals by using an *uncertainty model* (a meta-meta-model) that learns to predict the behavior of a performance predictor (a meta-model). This uncertainty model (UM) is a regression model that quantifies the uncertainty of the performance predictor's estimate of the base model accuracy on the production set. We pre-train the UM using a library of training datasets. The UM can then observe the behavior of a performance predictor on a new target dataset and generate a prediction interval.

The UM must be trained using examples of performance prediction errors. Each training sample for the UM consists of a full drift scenario, comprising: (1) labeled train and test datasets, (2) a pool of unlabeled data (the production set), (3) a base classification model trained on the train set, and (4) a performance predictor trained for this dataset and base model. We use two different simulation procedures described below to generate a large number of such training examples. The target values are the (absolute) differences

between the true and predicted accuracy on the entire production set. This method could be extended to predict the signed value of the errors, allowing for asymmetric prediction intervals.

The UM architecture is an ensemble of GBM models. Experimentally, we found that an ensemble of ten models reduced prediction variance and led to improved accuracy. The model was trained using a quantile loss function, which naturally enables the calculation of prediction intervals targeted to capture the true error with a specified probability. Further implementation details are described in the supplementary material.

Features The UM was trained using a set of derived features that are generic enough to be compatible across datasets with varying feature spaces and numbers of classes. The derived features are extracted from a number of models, including (1) the base classification model, (2) the performance predictor, (3) a group of proxy models, and (4) a group of drift models. The proxy models (one logistic regression, one random forest, and one GBM) were trained on the same features and classification task as the base model, and provide a complementary perspective on the classification difficulty of each data point. The drift models are random forest models trained to predict whether a given sample came from the test set or the production set. They provide direct insight into the degree of feature space drift for a given scenario. Further implementation details for the proxy and drift models are provided in the supplementary material.

A full list of the features used for the UM is shown in Table 1. The procedure for constructing the features of type `Distance` starts by choosing a function f that maps any feature vector to a scalar value, for example the highest value from the base model confidence vector. The value of this function is used to construct two histograms, one for the samples in the test set, and one for the production set. Finally, the distance between these two histograms is computed using some distance function D. We used three functions for D: the Kolmogorov–Smirnov metric $D_1 = \max_i |P_i - Q_i|$, the inverse overlap $D_2 = \sum_i \max(p_i - q_i, 0)$, and the squared inverse overlap $D_3 = \sum_i \big(\max(p_i - q_i, 0)\big)^2$. Here p and q are the normalized histograms from the test and production sets, i indexes the bins in the histograms (which must be identically spaced), and P, Q are the CDFs corresponding to p, q.

The remaining features can be grouped into three types: `Prediction`, `Noise`, and `Internal`. The `Prediction` features are directly derived from the predictions of one of the source models (without using the `Distance` procedure). These include the entropy of the base model predictions, the change in accuracy predicted by the performance predictor, and the accuracy of the drift classifiers. The `Noise` features are the size of bootstrap confidence intervals for the average of the top base model confidence and the performance predictor confidence, both of which approach zero as the number of points in the production set approaches infinity. The `Internal` features are white-box quantities extracted from the performance predictor or proxy models, such as the performance predictor *intrinsic* prediction intervals, or the difference between the calibrated and uncalibrated performance predictions. A complete description of the features listed here is given in the supplementary material.

An ablation study is presented in Table 2, showing the performance of the UM, coupled with the `meta-model` performance predictor, obtained using subsets of the full set of features described above. This study shows that the most effective information for understanding the performance prediction uncertainty comes from the drift models. It also shows that including the more numerous and computationally expensive `Distance` and `Internal` features deliver a significant performance boost over the simpler `Prediction` and `Noise` features.

Experimental Methodology

Fig. 1 shows example results for `linear-skew` drift scenarios simulated from the bng-zoo dataset. The middle and right panel show accuracy predictions from the *meta-model* performance predictor, and the uncertainty model prediction intervals calibrated using two different levels of α, (0.5 and 0.9), as described below.

Drift Simulation

Training and evaluating the UM requires examples of data drift. The breadth of the drift examples used for training largely determines the quality and coverage of the resulting model. Tabular labeled datasets containing sufficient naturally occurring drift are difficult to obtain, therefore we chose to generate such examples through resampling-based simulation. We focus on *covariate shift* because it has been shown to encompass a wide range of real-world drift scenarios (Card and Smith 2018).

We use two different algorithms for generating drift: (1) `linear-skew` , is designed for breadth of coverage, for providing training data, and ensuring the generation of extreme drift (2) `nearest-neighbors` is designed to simulate drift more likely to occur in the real world for an additional evaluation scenario. Examples of drift generated by both algorithms is included in the supplementary material.

The `linear-skew` method requires choosing a feature dimension (F) along which the bias will be induced, and a threshold t to split the dataset into two buckets. The sampling parameter R controls the ratio of sampling from the two buckets for the train/test sets and the production set, thus also controlling the amount of drift in the scenario. When $R = 50$, the train, test, and production sets all have the same distribution, and there is no drift. When $R = 0$ or $R = 100$, there is no overlap between the train/test distribution and the production distribution. The `linear-skew` procedure is described in Alg. 1.

The `nearest-neighbors` algorithm strives to simulate a particular demographic either appearing, or disappearing from production traffic. It does so by sampling a data point and then uses nearest neighbors to identify other data points that are similar (nearest neighbors) or dissimilar (furthest neighbors) and remove them from the dataset. We used

Source	Name	Description	Type	Count
Base	top (&1st - 2nd) confidence	highest (& 1st - 2nd) predicted class confidence	D	6
	confidence entropy	entropy of confidence vector	D	3
	class frequency	relative frequency of predicted classes	D	3
	entropy ratio	avg. prod. set entropy/avg. test set entropy	P	1
	bootstrap	size of bootstrap conf. intervals for avg. accuracy	N	1
Perf. Pred.	predicted change	predicted prod. acc. - base test acc.	P	1
	avg. pred. stdev. (& entropy)	stdev. (& entropy) of confidence scores (prod)	P	2
	predicted uncertainty	intrinsic uncertainty interval	I	1
	bootstrap	size of bootstrap conf. intervals for pred. acc.	N	1
	whitebox	internal stats from meta-model ensemble and calibration	I	20
Proxy	top (&1st - 2nd) confidence	highest (& 1st - 2nd) predicted class confidence	D	18
	best feature	projection onto most important feature	D	3
	num import. feat.	num. features to make 90% feat. importance	I	1
Drift	accuracy	test vs. prod classification accuracy	P	3
	(top-2nd) confidence	top - second highest class probability	D	9
Other	PCA projection	projection onto highest PCA component	D	3

Table 1: Source models, names, descriptions, types (D=`Distance`, P=`Prediction`, N=`Noise`, I=`Internal`), and counts for all UM features.

	Average Cost ($\alpha = 0.9$)				
	Base	Perf. Pred.	Proxy	Drift	All
`Distance`	1.46	–	1.21	1.18	1.07
`Internal`	1.61	1.15	–	–	1.05
`Pred.`	–	1.29	1.87	–	1.33
`Noise`	1.46	1.59	–	–	1.32
All	1.30	1.25	1.21	1.16	1.0

Table 2: Average uncertainty model cost, Eq. (1), for the `meta-model` predictor, with $\alpha = 0.9$, normalized by the value when all features are used.

this algorithm to create fairly severe drift, removing 50-70% of the original data points, which we believe covers the range of realistic possible drift. The `nearest-neighbors` algorithm is described in Alg. 2.

Datasets and Settings

For our experiments we use a set of fifteen publicly available tabular datasets, sourced from Kaggle, OpenML, and Lending Club: Artificial Character, Bach Choral, Bank Marketing, BNG Zoo, BNG Ionosphere, Churn Modeling, Creditcard Default, Forest Cover Type, Higgs Boson, Lending Club (2016 Q1, 2017 Q1), Network Attack, Phishing, Pulsar, SDSS, and Waveform. Details of the individual dataset's characteristics and our pre-processing procedures are provided in the supplementary material.

To simulate the effect of training the UM on an offline library of training datasets and then deploying it to make predictions on a new, unseen target dataset, we conducted our experiments in a leave-one-out manner. Each dataset was chosen in turn as the target, and its UM was trained on the remaining training datasets. All results are averaged over these fifteen different UMs.

Since we focused on tabular data, we used random forest and logistic regression base models for all experiments. In the `linear-skew` simulations we chose two features per dataset, and performed Alg. 1 for each feature with fifteen values of the sampling ratio $R = 0, 1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 95, 99, 100$. This was repeated using five random seeds, giving

Algorithm 1 Algorithm to create `linear-skew` drift scenarios

Input: Dataset $X \subset \mathcal{X}$; $p_{tr}, p_{te}, p_{pr} \in [0, 1] : p_{tr} + p_{te} + p_{pr} = 1$; Feature dimension $F : \mathcal{X}^{(F)} \subset \mathcal{X}$; threshold function $t : \mathcal{X}^{(F)} \rightarrow \{X_A, X_B\}$; Sampling ratio $R \in [0, 100]$; minibatch size b
Output: $X_{tr}, X_{te}, X_{pr} \subset X$
$X_A, X_B, X_{tt}, X_{pr} \leftarrow \{\}$
for x **in** X **do**
  Add x to $t(X^{(F)}) \in \{X_A, X_B\}$ ▷ Add data point to bucket, defined by threshold t
end for
while $|X_A| > b$ **and** $|X_B| > b$ **do** ▷ Randomly sample X_{tt}, X_{pr} from buckets until out of data points
  Add $(p_{tr} + p_{te}) \times \frac{R}{100} \times b$ points from X_A and $(p_{tr} + p_{te}) \times (1 - \frac{R}{100}) \times b$ points from X_B into X_{tt}
  Add $p_{pr} \times (1 - \frac{R}{100}) \times b$ points from X_A and $p_{pr} \times \frac{R}{100} \times b$ points from X_B into X_{pr}
end while
Randomly split $X_{tr}, X_{te} \leftarrow X_{tt}$ with proportions $\frac{p_{tr}}{p_{tr}+p_{te}}$ and $\frac{p_{te}}{p_{tr}+p_{te}}$

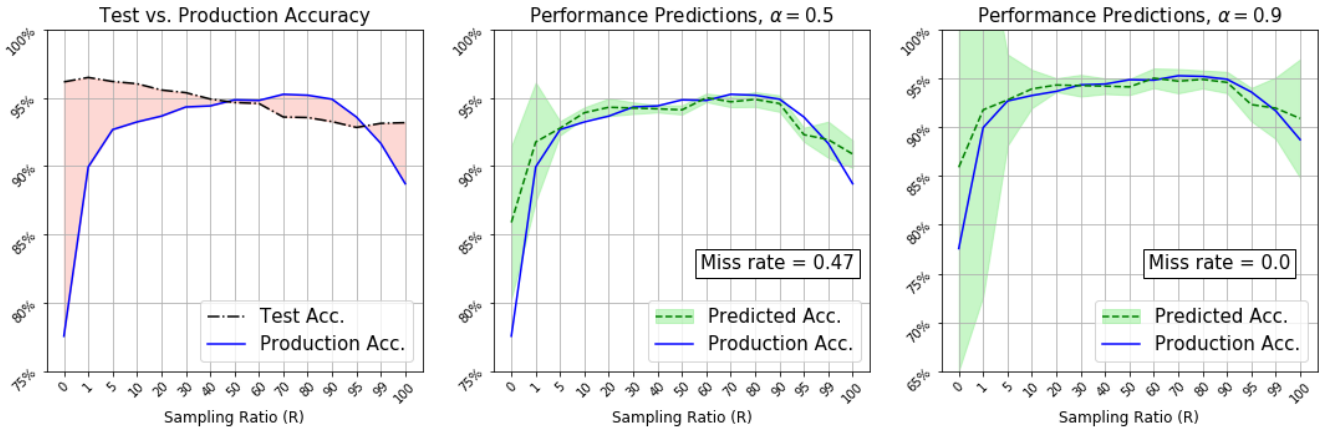

Figure 1: Example of performance prediction and the UM for `linear-skew` drift scenarios. The left plot shows the model accuracy drift (shaded area) induced by Alg. 1. The middle (right) plot shows the accuracy predicted by the `meta-model` predictor, with UM prediction intervals calibrated to $\alpha = 0.5$ ($\alpha = 0.9$).

a total of 300 drift scenarios per dataset.[1] For the `nearest-neighbors` simulations, 300 scenarios were generated for each dataset/base model combination with parameters $P_{set} = 0.5$, $P_{near} = 0.5$, and $P_{down} \in [0.5, 0.7]$, for a total of 9000 scenarios.

Model-Free Baselines

A set of model-free baseline techniques for computing prediction intervals are compared against the UM. The first set of techniques are three *intrinsic* methods which leverage white-box information from the performance predictors. These intrinsic methods produce uncertainty estimates for each point in the production set, and the average of these estimates is used as the (uncalibrated) prediction interval.

For the `confidence` predictor, if the accuracy in bin k is a_k and the number of samples falling into bin k is n_k, we compute an uncertainty score for each point in the $k-$th bin as $u_k = \sqrt{a_k(1-a_k)/n_k}$, which is the standard error of a Bernoulli distribution with parameter a_k.

For the `meta-model` predictor, we created two variants that replace the GBM and logistic regression meta-models with different classifiers. The `crossval` predictor uses an ensemble of ten random forest models, each trained with a different cross-validation fold of the test set, and the standard deviation of their predictions is used as the uncertainty estimate. The `dropout` predictor uses an XGBoost (Chen and Guestrin 2016) model with the DART (Vinayak and Gilad-Bachrach 2015) booster, which applies dropout (Srivastava et al. 2014) regularization to GBM models. In the spirit of the Monte-Carlo dropout approach for Bayesian neural networks (Gal and Ghahramani 2015; Gal, Hron, and Kendall 2017), ten predictions are made for each sample with dropout turned on to introduce randomness in the confidence scores, giving an estimated average and standard deviation for the model accuracy.

Besides the intrinsic baselines, we also compare with three other baseline methods, which produce prediction intervals using: (1) the standard error of the mean of the performance predictor confidences, (2) the size of a bootstrap uncertainty interval for the mean of these confidences, and (3) a constant sized prediction interval.

Evaluation Metric

Evaluating the quality of a set of prediction intervals involves a trade-off between two opposing kinds of errors: prediction intervals that are too small and do not capture the magnitude of the true error (Type I cost), and prediction intervals that are unnecessarily large (Type II cost). In an ad hoc comparison between two methods generating prediction intervals, it is common that one method does not dominate

[1] Except for the Network Attack dataset, which only has one feature amenable to this procedure, see supplementary material.

Algorithm 2 Algorithm to create `nearest-neighbors` drift scenarios

Input: Dataset $X \subset \mathcal{X}$; $p_{tr}, p_{te}, p_{pr} \in [0,1] : p_{tr} + p_{te} + p_{pr} = 1$; $P_{set} \in [0,1]$; $P_{near} \in [0,1]$; $P_{down} \in [0,1]$
Output: $X_{tr}, X_{te}, X_{pr} \subset X$
Randomly split data into X_{pr} and X_{tt} with proportions p_{pr} and $1 - p_{pr}$
With probability P_{set}, set downsample set $X_{down} = X_{tt}$ and $X_{rand} = X_{pr}$, else $X_{down} = X_{pr}$ and $X_{rand} = X_{tt}$
▷ Choose distribution to bias non-randomly
Choose point $p \in X_{down}$ at random
Order points $x \neq p \in X_{down}$ by distance from p
Choose D=nearest (N) with probability P_{near} else D=furthest (F) ▷ Choose nearest or furthest bias
Remove the fraction P_{down} points which are $D \in \{N, F\}$ from p
Remove fraction P_{down} from X_{rand} randomly ▷ Randomly downsample non-biased distribution
Randomly split $X_{tr}, X_{te} \leftarrow X_{tt}$ with proportions $\frac{p_{tr}}{p_{tr}+p_{te}}$ and $\frac{p_{te}}{p_{tr}+p_{te}}$

the other in the sense of having both smaller Type I and Type II error. A comprehensive comparison of such methods requires making a tradeoff between the two.

One common approach to quantifying this trade-off is to scale each set of prediction intervals by a constant factor to achieve a common miss rate (eg 5%), and then compare their average size or average excess. We chose instead to evaluate results based on a cost function which penalizes both types of error:

$$C_\alpha(\vec{\delta},\vec{u}) = \sum_i \Big[\alpha \max\left(\delta_i - u_i, 0\right) + (1-\alpha)\max\left(u_i - \delta_i, 0\right)\Big]. \quad (1)$$

In Eq. (1), $\delta_i = |a_i - p_i|$ is the difference between the true base model accuracy and the performance prediction for production set i, and u_i is the (single-directional) size of the scaled prediction interval. The cost function parameter $\alpha \in [0,1]$ can be adjusted to control the balance between the two types of error.

Calibration

Prediction intervals must be calibrated in order to achieve reliable performance for a scale-sensitive metric such as Eq. (1). We compare two calibration methods, one with external drift scenarios as a holdout set, and one without. Without the holdout set for calibration (Target Calibration), we used the approximation that the performance prediction error is normally distributed, and multiplied the uncalibrated prediction intervals by the Z-score of the desired confidence interval (determined by the cost function parameter α).[2] This calibration method is applied to all baselines.

The second calibration method (TL Calibration) used the simulated drift scenarios from external holdout datasets as a calibration set. A constant scale factor was computed which minimized the cost function Eq. (1) for each set of prediction intervals on this hold-out set. This calibration method was applied to the model-free baselines using 100% of the holdout drift scenarios, and to the UM prediction intervals using an 80%/20% train/test split of the drift scenarios.

Experimental Results

In this section we demonstrate the performance of our UM, trained using the `linear-skew` drift scenarios, and evaluated using both the `linear-skew` and the `nearest-neighbors` style scenarios. We use the four performance predictors and the four baselines described above. The quality of the prediction intervals is measured by the cost function C_α from Eq. (1), with α between 0.5 and 0.95. This range covers most reasonable user preferences for penalizing under- vs. over-shooting of the appropriate prediction interval size.

Fig. 2 compares the cost C_α of the UM and the standard error, bootstrap, and intrinsic baselines, evaluated using

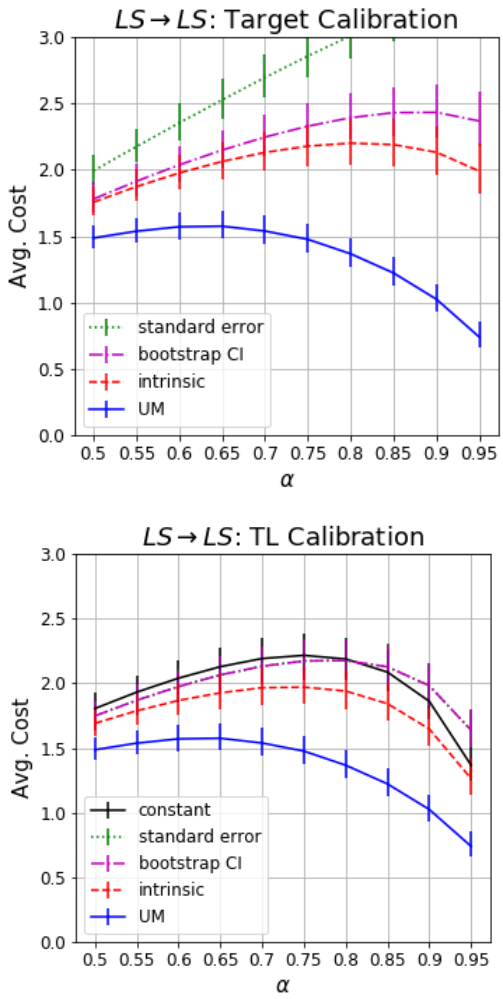

Figure 2: Average cost (Eq. (1)) for prediction intervals. The baselines, calibrated without (top) and with (bottom) external datasets, are compared to the UM. The evaluation uses the `linear-skew` drift scenarios.

the `linear-skew` drift scenarios. The results are averaged across the four performance predictors[3], and calibrated using the target dataset method (top) and external holdout drift scenarios (bottom). The error bars in Fig. 2 indicate the 95% bootstrap confidence interval. The bottom plot also includes the calibrated constant baseline.

It is clear from the upper panel of Fig. 2 that the UM method trained with the `linear-skew` simulated drift scenarios substantially outperforms the baselines. This is especially true for moderate to high values of α, which correspond to penalizing prediction intervals that are too small more than intervals which are too large. Comparing this result to the bottom panel, we see that merely calibrating with the simulated drift scenarios can dramatically improve the performance of the baseline methods. However, the UM still

[2]The cost is minimized by balancing the two terms in Eq. (1), thus the α%-confidence interval chosen for calibration. For example, if $\alpha = 0.95$ is chosen, then the scale factor is $Z = 1.96$.

[3]There are only three sets of results for the "intrinsic" curve, as the `meta-model` predictor has no intrinsic prediction interval.

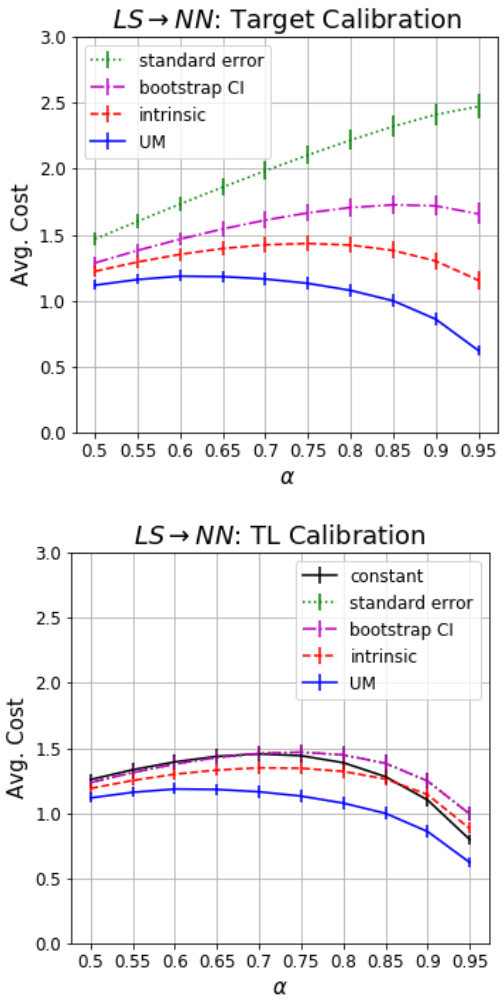

Figure 3: Same experiments as Fig. 2, except using `nearest-neighbors` drift scenarios for evaluation.

provides a major improvement at all values of α.

Fig. 3 shows the same experiments as Fig. 2, but using `nearest-neighbors` scenarios for evaluation (and still using `linear-skew` scenarios for training). The UM still outperforms the baselines for the full range of α. The overall costs tend to be smaller, since the model accuracy drift in the `nearest-neighbors` scenarios was smaller on average than in the `linear-skew` scenarios. This result confirms that the deliberately engineered `linear-skew` scenarios are able to provide effective training for more realistic, organically created `nearest-neighbors` drift scenarios.

Table 3 provides all of the results in numeric form, broken down by performance predictor, and averaged across the same range of α values. This confirms that the individual predictor results align with the previous average results.

Choosing the source datasets for this transfer-learning based approach is an important consideration. For a domain specific application, it is obviously preferable to choose source datasets from the same or a closely related domain.

LS (train) → LS (eval)				
	Predictor			
Method	Conf.	Crossval	Dropout	Meta
SE	2.88	2.57	3.09	2.41
BS	2.38	2.06	2.50	1.89
I	2.23	1.72	2.20	–
SE (TL)	2.13	1.91	2.13	1.77
BS (TL)	1.13	1.91	2.13	1.77
C (TL)	2.11	1.88	2.10	1.83
I (TL)	1.83	1.67	1.86	–
UM	**1.46**	**1.24**	**1.38**	**1.33**
LS (train) → NN (eval)				
	Predictor			
Method	Conf.	Crossval	Dropout	Meta
SE	1.97	1.87	2.48	1.75
BS	1.59	1.41	2.00	1.29
I	1.39	1.14	1.48	–
SE (TL)	1.36	1.25	1.58	1.15
BS (TL)	1.36	1.25	1.58	1.15
C (TL)	1.30	1.20	1.49	1.16
I (TL)	1.26	1.17	1.29	–
UM	**1.08**	**1.00**	**1.14**	**0.97**

Table 3: Costs for both experiments, including the standard error (SE), bootstrap (BS), and intrinsic (I) baselines, the same methods with our transfer-learning (TL) calibration, as well as the calibrated constant (C) and UM, averaged over the values of α shown in the figures (0.5 to 0.95).

In addition to the domain, we expect that it is valuable to approximately match other dataset characteristics such as number of classes, number of features, feature sparsity, etc.

For applications to other data modalities, for example image or text data, many standard benchmark datasets such as ImageNet could be used for pre-training. The models used in the UM and the feature extraction would need to be replaced or supplemented with modality appropriate architectures to extract meaningful features from the data. Finally, the base models used in the drift scenarios for training the UM should include models of the same class as those to which it will be applied at prediction time.

Conclusion

Performance prediction is an invaluable part of the deploying, monitoring, and improving an AI model. This paper addresses this problem by describing a novel technique for quantifying model uncertainty. It leverages multi-task learning and meta-meta-modeling to generate prediction intervals on any model-based performance prediction system. Our method substantially outperforms a group of competitive baselines on dataset shift produced by two different simulation mechanisms. We believe this work helps make performance prediction more practical for real-world use, and may encourage further innovation in this important area.

Ethics Statement

Training AI models to "know what they don't know" is one of the key challenges in AI today (Kindig 2020; Davey 2018; Knight 2018). AI models can be notoriously overconfident in scenarios that their training data did not prepare them for (Nguyen, Yosinski, and Clune 2015), eroding trust in AI and society's willingness to accept AI as it continues to replace human decision making in increasingly important roles. Our work directly addresses this problem, proposing a novel technique for quantifying quantifying model uncertainty that beats all baselines we compare against. In addition, we hope that this work will encourage the community to continue to invest and innovate in this important area. Due to the nature of this problem we do not foresee any negative consequences stemming from our work.

References

Bishop, C. M. 1997. Bayesian Neural Networks. *Journal of the Brazilian Computer Society* 4. ISSN 0104-6500.

Blundell, C.; Cornebise, J.; Kavukcuoglu, K.; and Wierstra, D. 2015. Weight Uncertainty in Neural Networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, volume 37 of *ICML'15*, 1613–1622. JMLR.org.

Brosse, N.; Riquelme, C.; Martin, A.; Gelly, S.; and Moulines, É. 2020. On Last-Layer Algorithms for Classification: Decoupling Representation from Uncertainty Estimation. *arXiv preprint arXiv:2001.08049* .

Card, D.; and Smith, N. A. 2018. The Importance of Calibration for Estimating Proportions from Annotations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, volume 1, 1636–1646. Association for Computational Linguistics.

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, 785–794. Association for Computing Machinery. ISBN 9781450342322.

Chen, T.; Navratil, J.; Iyengar, V.; and Shanmugam, K. 2019. Confidence Scoring Using Whitebox Meta-models with Linear Classifier Probes. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 1467–1475.

Davey, T. 2018. How AI Handles Uncertainty. https://futureoflife.org/2018/03/15/how-ai-handles-uncertainty-brian-ziebart, (last accessed 06/05/2020).

Dietterich, T. G. 2000. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, 1–15. Springer. ISBN 978-3-540-45014-6.

Efron, B. 1979. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics* 7(1): 1–26.

Efron, B.; and Gong, G. 1983. A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation. *The American Statistician* 37(1): 36–48.

Finn, C.; Rajeswaran, A.; Kakade, S.; and Levine, S. 2019. Online Meta-Learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, 1920–1930. PMLR.

Gal, Y.; and Ghahramani, Z. 2015. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *Proceedings of The 33rd International Conference on Machine Learning* .

Gal, Y.; and Ghahramani, Z. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, volume 29, 1019–1027. Curran Associates, Inc.

Gal, Y.; Hron, J.; and Kendall, A. 2017. Concrete Dropout. In *Advances in Neural Information Processing Systems*, volume 30, 3581–3590. Curran Associates, Inc.

Geisser, S. 2017. *Predictive Inference*. CRC Press. ISBN 9781351422291.

Guerra, S. B.; Prudêncio, R. B.; and Ludermir, T. B. 2008. Predicting the performance of learning algorithms using support vector machines as meta-regressors. In *International Conference on Artificial Neural Networks*, volume 5163, 523–532. Springer.

Hansen, L.; and Salamon, P. 1990. Neural Network Ensembles. *I E E E Transactions on Pattern Analysis and Machine Intelligence* 12: 993–1001. ISSN 0162-8828.

Hospedales, T.; Antoniou, A.; Micaelli, P.; and Storkey, A. 2020. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439* .

Kabir, H.; Khosravi, A.; Kavousi-Fard, A.; Nahavandi, S.; and Srinivasan, D. 2019. Optimal Uncertainty-guided Neural Network Training. *arXiv preprint arXiv:1912.12761* .

Kendall, A.; and Gal, Y. 2017. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In *Advances in Neural Information Processing Systems*, volume 30, 5574–5584. Curran Associates, Inc.

Khosravi, A.; Nahavandi, S.; and Creighton, D. 2010. Construction of Optimal Prediction Intervals for Load Forecasting Problems. *IEEE Transactions on Power Systems* 25(3): 1496–1503. ISSN 1558-0679.

Khosravi, A.; Nahavandi, S.; Creighton, D.; and Atiya, A. F. 2011. Comprehensive Review of Neural Network-Based Prediction Intervals and New Advances. *IEEE Transactions on Neural Networks* 22(9): 1341–1356.

Kindig, B. 2020. 5 Soon-to-Be Trends in Artificial Intelligence And Deep Learning. https://www.forbes.com/sites/bethkindig/2020/01/31/5-soon-t0-be-trends-in-artificial-intelligence-and-deep-learning, (last accessed 06/05/2020).

Knight, W. 2018. Google and Others Are Building AI Systems That Doubt Themselves AI will make better decisions by embracing uncertainty. https://www.technologyreview.com/2018/01/09/146337/google-and-others-are-building-ai-systems-that-doubt-themselves, (last accessed 06/05/2020).

Koenker, R. W.; and Bassett, G. 1978. Regression Quantiles. *Econometrica* 46(1): 33–50.

Kwok, S. W.; and Carter, C. 1990. Multiple decision trees. In *Uncertainty in Artificial Intelligence*, volume 9 of *Machine Intelligence and Pattern Recognition*, 327 – 335. North-Holland.

Lakshminarayanan, B.; Pritzel, A.; and Blundell, C. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems*, volume 30, 6402–6413. Curran Associates, Inc.

Lei, J.; G'Sell, M.; Rinaldo, A.; Tibshirani, R. J.; and Wasserman, L. 2018. Distribution-Free Predictive Inference for Regression. *Journal of the American Statistical Association* 113(523): 1094–1111.

Li, D.; Yang, Y.; Song, Y.-Z.; and Hospedales, T. 2018. Learning to Generalize: Meta-Learning for Domain Generalization. *AAAI Conference on Artificial Intelligence* .

Malinin, A.; and Gales, M. 2018. Predictive Uncertainty Estimation via Prior Networks. In *Advances in Neural Information Processing Systems*, volume 31, 7047–7058.

Neal, R. M.; et al. 2011. MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo* 2(11): 2.

Nguyen, A.; Yosinski, J.; and Clune, J. 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 427–436. ISSN 1063-6919.

Nix, D. A.; and Weigend, A. S. 1994. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, 55–60. IEEE.

Osband, I.; Aslanides, J.; and Cassirer, A. 2018. Randomized Prior Functions for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 31.

Papadopoulos, H. 2008. *Inductive Conformal Prediction: Theory and Application to Neural Networks*, 315–330. Citeseer. ISBN 978-953-7619-03-9.

Redyuk, S.; Schelter, S.; Rukat, T.; Markl, V.; and Biessmann, F. 2019. Learning to Validate the Predictions of Black Box Machine Learning Models on Unseen Data. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, 1–4.

Schat, E.; van de Schoot, R.; Kouw, W. M.; Veen, D.; and Mendrik, A. M. 2020. The Data Representativeness Criterion: Predicting the Performance of Supervised Classification Based on Data Set Similarity. *arXiv preprint arXiv:2002.12105* .

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15: 1929–1958.

Su, J.; Vargas, D. V.; and Sakurai, K. 2019. One Pixel Attack for Fooling Deep Neural Networks. *IEEE Transactions on Evolutionary Computation* 23(5): 828–841. ISSN 1941-0026.

Talagala, T. S.; Li, F.; and Kang, Y. 2019. FFORMPP: Feature-based forecast model performance prediction. *arXiv preprint arXiv:1908.11500* .

Vinayak, R. K.; and Gilad-Bachrach, R. 2015. DART: Dropouts meet Multiple Additive Regression Trees. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, 489–497. PMLR.

Vovk, V. 2012. Conditional validity of inductive conformal predictors. In *Asian conference on machine learning*, 475–490.

Vovk, V.; Gammerman, A.; and Shafer, G. 2005. *Algorithmic learning in a random world*. Springer Science & Business Media.

Vovk, V.; Nouretdinov, I.; Manokhin, V.; and Gammerman, A. 2018. Cross-conformal predictive distributions. In *Conformal and Probabilistic Prediction and Applications*, 37–51.

Zadrozny, B.; and Elkan, C. 2001. Obtaining calibrated probability estimates from decision trees and naive Bayesian classifiers. In *Proceedings of the Eighteenth International Conference on Machine Learning*, 609–616. Morgan Kaufmann.