Learning Prediction Intervals for Model Performance

Benjamin Elder, Matthew Arnold, Anupama Murthi, Jiri Navratil

IBM Research

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.

1 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.

2 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...
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 to 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 predictor ([Brosse et al., 2020; Kabir et al., 2019; Khosravi, Navanidi, 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 (see Sec. 4.3).

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.

3 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.

3.1 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.

3.2 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 generate a large number of such examples using the simulation procedures described in Sec. 4.1. 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 \( D_3 = \sum_i (\max(p_i - q_i, 0))^2 \). 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 intrinsic prediction intervals described in Sec.4.3, 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.

### 4 Experimental methodology

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.

**Linear-skew** 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.

**Nearest-neighbors** 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 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.

4.2 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, Pursar, 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. 2 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

| Source | Name | Description | Type | Count |
|--------|------|-------------|------|-------|
| Base   | top confidence | confidence score of predicted class | D | 3     |
|        | (top-2nd) confidence | top - second highest class probability | D | 3     |
|        | 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. | stdev. of confidence scores (prod) | P | 1     |
|        | avg. pred. entropy | entropy of confidence histogram (prod) | P | 1     |
|        | predicted uncertainty | intrinsic uncertainty interval | I | 1     |
|        | bootstrap | size of bootstrap conf. intervals for pred. acc. | N | 1     |
|        | ensemble whitebox | whitebox stats from meta-model ensemble | I | 2     |
|        | calibration whitebox | whitebox stats from calibration | I | 2     |
|        | gbm whitebox | internal stats from gbm meta-model | I | 16    |
| Proxy | top confidence | confidence score of predicted class | D | 9     |
|        | (top-2nd) confidence | top - second highest class probability | D | 9     |
|        | 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 ($\alpha = 0.9$) | Distance | Perf. Pred. | Proxy | Drift | All |
|-------------------------------|----------|-------------|-------|-------|-----|
| Base                          | 1.46     | 1.21        | 1.18  | 1.07  |     |
| Internal                      | 1.61     | 1.15        | 1.29  | 1.87  | 1.33|
| Prediction                    | 1.46     | 1.59        | 1.21  | 1.16  | 1.0 |
| Noise                         | 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.

### Algorithm 1

Algorithm to create linear-skew drift scenarios

**Input:** Dataset $X \subset X'; p_{tr}, p_{te}, p_{pr} \in [0, 1]: p_{tr} + p_{te} + p_{pr} = 1$; Feature dimension $F: X^{(F)} \subset X$; threshold function $t: X^{(F)} \rightarrow \{X_A, X_B\}$; 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** $X$

Add $x$ to $\{(X^{(F)}) \in \{X_A, X_B\} \triangleright$ Add data point to bucket, defined by threshold $t$

**end for**

**while** $|X_A| > b$ **and** $|X_B| > b$ **do**

$\triangleright$ 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 $\frac{p_{te}}{p_{tr} + p_{te}}$
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 model with the DART booster, which applies dropout regularization to GBM models. In the spirit of the Monte-Carlo dropout approach for Bayesian neural networks (Gal and Ghahramani 2015), 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.

### 4.4 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 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 scal each set of prediction intervals by a constant factor to
achieve a common miss rate (e.g., 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:

\[
C_\alpha(\delta, u) = \sum_i \left[ \alpha \max(\delta_i - u_i, 0) + (1 - \alpha) \max(u_i - \delta_i, 0) \right].
\]  

(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.

4.5 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 \(\alpha\)).

This calibration method is applied to all of the model-free baselines described in Sec. 4.3.

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.

5 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 described in Sec. 3.1 and the four baselines described in Sec. 4.3. The quality of the prediction intervals is measured by the cost function \(C_\alpha\) from Eq. (1), with \(\alpha\) 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.

Linear-skew results Fig. 2 compares the cost \(C_\alpha\) of the UM and the standard error, bootstrap, and intrinsic baselines described in Sec. 4.3 evaluated using the linear-skew drift scenarios. The results are averaged across the four performance predictors and calibrated using the target dataset method (top) and external holdout drift scenarios (bottom), as described in Sec. 4.5. 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 \(\alpha\), 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 \(\alpha\%-\)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.
provides a major improvement at all values of $\alpha$.

**Nearest-neighbors results**  Fig. 3 shows the same experiments as Fig. 2 but using nearest-neighbors drift scenarios for evaluation (and still using linear-skew scenarios for training). The UM still outperforms the baselines for the full range of $\alpha$. 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 $\alpha$ values. This confirms that the individual predictor results align with the averaged results previously shown.

### 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 $\alpha$ shown in the figures (0.5 to 0.95).  

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

### 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.
7 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 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.

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. IEEE Transactions on Pattern Analysis and Machine Intelligence 12: 993–1001. ISSN 0162-8828.

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

Kabir, H.; Khosravi, A.; Kavoussi-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.; and Creighton, D. 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-to-be-trends-in-artificial-intelligence-and-deep-learning/

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/
8 Appendix to Submission: Learning Prediction Intervals for Model Performance

8.1 Performance predictors

Here we provide further implementation details of the meta-model performance prediction algorithm introduced in Sec. 3.1 as well as the crossval and dropout variants described in Sec. 4.3.

Meta-model architectures The meta-model predictor uses an ensemble of a gbm and a logistic regression model. The crossval performance predictor uses an ensemble of ten random forest meta-models, each trained on a separate fold of the test set. The dropout predictor uses a single XGBoost (Chen and Guestrin 2016) regression model (the regression target is the confidence that a base model classification is correct), with the DART (Vinayak and Gilad-Bachrach 2015) booster. When training the XGBoost model, early stopping is enabled, with the RMSE computed on a holdout set of 20% of the training data used as the early stopping criteria. The dropout regularization was implemented by the base model is more likely to make mistakes for classes which were very rare in its training data.

Three proxy models were used to generate additional features. These proxy models (a logistic regression model, a random forest model, and a gbm) were trained on the same train set as the base model, and provide a complementary perspective on the relative classification difficulty of a given data point. The random forest and gbm models were trained using fifteen rounds of HPO. Once these three proxy models were trained, the top class confidence and 1st - 2nd class confidences from each model were used as features for the meta-model.

Finally, two additional features were designed to quantify the distance of a given data point from the typical point in the training data. The first feature uses the output of the decision function of a one class svm, which classifies samples as either in or out of the training data set. To construct the second feature, the "umap kde" transformer first uses the UMAP (McInnes, Healy, and Melville 2018) dimensional reduction algorithm to reduce the dimensionality of the (standard scaled) training dataset down to six dimensions (using parameters nearest_neighbors = 10 and min_dist = 0.1). After dimensional reduction, a separate kernel density estimate (kde) is fit to the data points in each class. The kde uses a Gaussian kernel with a bandwidth of 0.2. At inference time, each kde provides a probability score for the likelihood that data point was sampled from the distribution described by that kde. The highest of these probabilities is returned as the value for the "umap kde" transformed feature.

Calibration The meta-models used for performance prediction were all calibrated using a 20% calibration set held out from their training data (which was the same as the "test set" in the drift scenario). Isotonic regression was used to perform the calibration. The targets $T_{ir}$ for the isotonic regression calibration were

$$T_{ir} = \begin{cases} \frac{1}{n_+ + 2} & \text{for } y \text{ incorrect} \\ \frac{n_+ - 1}{n_+ + 2} & \text{for } y \text{ correct} \end{cases},$$

where $n_+$ and $n_-$ are the number of samples in the calibration set which were correctly and incorrectly predicted by the base model respectively. These targets account for finite statistics effects, as they interpolate from equality at

| Parameter                  | Range                                                                 |
|---------------------------|-----------------------------------------------------------------------|
| learning_rate             | gbm-meta-model                                                        |
| min_samples_split         | np.linspace([0.005, 0.01, 5])                                         |
| min_samples_leaf          | np.linspace([0.0005, 0.001, 5])                                       |
| max_leaf_nodes            | [3, 5, 7, 9, 11]                                                      |
| max_features              | [log2, sqrt]                                                          |
| subsample                 | [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]                                   |
| n_estimators              | [100, 150, 200, 250, 300, 350, 400]                                   |
| C                         | [0.001, 0.01, 0.1, 1, 10, 100, 1000]                                   |
| penalty                   | $\ell_2$                                                              |
| n_estimators              | random_integer $\in [5, 100]$                                         |
| max_depth                 | [2, 3, 4, 6, None]                                                    |
| min_samples_split         | [2, 3, 4, 5, 6]                                                       |
| max_features              | [log2, sqrt]                                                          |
| max_depth                 | [2, 4, 6]                                                             |
| learning_rate             | [0.05, 0.1, 0.2, 0.3, 0.4, 0.5]                                       |
| n_estimators              | [50, 100, 150, 200]                                                   |
| reg_lambda                | np.logspace(-1, 1, num = 20)                                          |
| sample_type               | [normal, weighted]                                                   |
| normalize_type            | [tree, forest]                                                       |

Table 4: Hyperparameter ranges included in the random search for HPO performed when fitting the meta-models for performance prediction.
$T_{\text{pr}} = \frac{1}{2}$ with no calibration samples, to the limit of perfect certainty, where the confidence of a correct classification is zero percent for an incorrectly classified sample, and 100% for a correctly classified sample, in the limit of infinite calibration samples (assuming a base model accuracy $0 < \text{acc} < 100$).

Performance prediction results Table 5 presents a comparison of the four performance predictors, in terms of the mean absolute error and the area under the ROC curve for their predictions, for the collection of linear-skew drift scenarios (left) and nearest-neighbors drift scenarios (right). The inferior performance of the crossval and dropout predictors compared to the meta-model predictor illustrates an additional benefit of the UM approach to generating prediction intervals, which is that there is no need to modify the architecture of the meta-model in a potentially detrimental way.

| Predictor    | MAE     | AUC     |
|--------------|---------|---------|
| Confidence   | 4.38    | 0.73    |
| Crossval     | 4.04    | 0.78    |
| Dropout      | 4.70    | 0.77    |
| Meta-model   | 3.84    | 0.79    |

| Predictor    | MAE     | AUC     |
|--------------|---------|---------|
| Confidence   | 3.01    | 0.75    |
| Crossval     | 2.94    | 0.79    |
| Dropout      | 3.73    | 0.77    |
| Meta-model   | 2.78    | 0.80    |

Table 5: Comparison of performance predictor results for linear-skew drift scenarios (left) and nearest-neighbors drift scenarios (right).

8.2 Uncertainty model

The UM is trained using a quantile loss function, with the target quantile equal to the cost function parameter $\alpha$, and the hyperparameters: $\text{max-features} = \log(2)$, $\text{n_estimators} = 200$, and $\text{subsample} = 0.8$. A single hyperparameter configuration was chosen because running HPO to fit the UM led to a large increase in variance between random seeds.

Features A few additional details are required to fully describe the UM features. The proxy models for these features were the same proxy models described in App. 8.1. The drift models were a set of three random forest models which were trained to distinguish data points in the test set from points in the production set. Three of these models were trained which were equivalent except for the set of input features for the model: one drift model was trained on the original feature vectors of the data, one was trained on these features transformed as described in App. 8.1 and the last was trained on the concatenation of the original and transformed feature vectors.

The $\text{Distance}$ features all required a histogram, and therefore depend on the choice of histogram bins. The histograms for the "top confidence" features from the base model and the three proxy models were binned in units of 0.1 from 0 to 0.9, and in units of 0.01 from 0.9 to 1. The top confidences from the drift models tended to exhibit a small set of discrete modes, so a histogram using eleven equally spaced bins, spanning 0 to 1 was used to fully separate these modes. The histograms for the "top-2nd confidence" features from the base model and the three proxy models were binned in units of 0.1 from 0 to 1. For the "confidence entropy" feature from the base model, the histograms were binned by 0.01 from 0 to 0.1, and by 0.1 from 0.1 to 3. The "class frequency" feature from the base model takes categorical values, so it simply has one bin per category. Finally, for the "best feature" and "PCA projection" features, the histograms were constructed by choosing ten equally spaced bins spanning the range of the 10-th through 90-th quantiles of the data. The outlying data points were not included in the histogram.

The "best feature" dimension was chosen to be the feature dimension in the original data which was assigned the highest feature importance from the random forest proxy model. The feature importances from the random forest proxy model were also used to construct the "num important features" feature.

The two "ensemble whitebox" features were designed to capture the level of disagreement between the models in performance predictor’s ensemble. The first feature, $\text{stdev-of-means}$, measures how much disagreement there was in each model’s average performance. It is constructed by computing each meta-model’s mean performance prediction (averaged over all data points in the production set) and then taking the standard deviation of those means (across meta-models). The second feature, $\text{mean-of-stdevs}$, captures how much variation there is between models on a per-data point level. It is constructed by computing the standard deviation (across models) of each individual data point’s performance predictions, then computing the average (across all data points) of those standard deviations.

The two "calibration whitebox" features are signed and absolute difference in base model accuracy predicted before and after applying the isotonic regression calibration. The $\text{gbm whitebox}$ features summarize information observed within the GBM meta-model. The first set of features, $\text{gbm-decision-distance}$, were constructed by observing the individual decision trees at prediction time and measuring the distance between each data point’s feature value and the decision thresholds it is compared against. These distances were averaged over all data points in the production set and over all decision trees in the GBM, but were grouped by level in the tree (level 1, level 2, level 3) as well as an aggregate summed across all levels.

The second set of gbm whitebox features, $\text{gbm-node-frequencies}$, compare how node frequencies of the decision tree traversals change when predicting on test vs production. It is computed by predicting on the test set and counting the number of times each
node in the tree was traversed, then normalizing by the number of data points. This process is then repeated for the production set to produce two vectors of normalized node frequencies (test and prod). These vectors are subtracted and the absolute value taken to produce a node-frequency-delta vector. Features are created by taking the max, average and standard deviation of the node-frequency-delta vector.

9 Experimental methodology

9.1 Drift simulation

As an example, Fig. 4 shows the change in base model accuracy, and performance predictions from the meta-model predictor for linear-skew drift scenarios simulated from the Lending Club dataset. Shown are fifteen drift scenarios, derived using the feature \( F = \) interest rate, using sampling ratios varying from 0 to 100. The solid blue (dot-dashed black) curve is the true accuracy of the random forest base model on the production (test) set. The gap between the two indicates the change in base model accuracy due to the induced drift. The dashed green curve is the meta-model predictor’s estimate of the production set accuracy. It is easy to see from this plot that the performance prediction works very well until the sampling ratio becomes fairly extreme (\( R < 5 \) or \( R > 90 \)).

Fig. 5 shows similar results, for the same dataset and predictor, using drift scenarios generated with the nearest-neighbors method. Shown are the results for 300 scenarios, ordered by amount of change in the base model accuracy (smoothed by a nine-point rolling average).

Figure 4: Example of drift scenarios generated with the linear-skew methodology, and the predicted accuracy from the meta-model predictor.

Figure 5: Example of drift scenarios generated with the nearest-neighbors methodology, and the predicted accuracy from the meta-model predictor.

9.2 Datasets and settings

Table 6 presents more detail about the datasets used in our experimental results. For computational efficiency, several datasets were downsampled (marked with a *). For the Lending Club dataset, only loan records with outcomes of either defaulted or repaid were used, and records which did not yet have a final outcome were excluded. Finally the bach chorale dataset was preprocessed to combined all classes with the same key (eg G1, G2, . . . G7) into a single label. In addition, for several datasets we pruned features as a preprocessing step, in order to improve the data quality (‡).

10 Results

For completeness, the results from Table 3 are replicated here in Table 7 including the standard errors for all values. These standard errors were obtained by bootstrap sampling to construct 95%-confidence intervals. The differences between prediction interval methods within each experiment and performance predictor are all statistically significant, as the confidence intervals do not overlap.
| Dataset                          | Samples | Features | Classes | Source                                                                 |
|---------------------------------|---------|----------|---------|------------------------------------------------------------------------|
| Artificial Character            | 10218   | 7        | 10      | [www.openml.org/d/1459](http://www.openml.org/d/1459)                 |
| Bach Choral                     | 5.665   | 16       | 7†      | archive.ics.uci.edu/ml/datasets/Bach-Choral+Harmony                  |
| Bank Marketing                  | 38,245  | 16       | 2       | [http://archive.ics.uci.edu/ml/datasets/Bank+Marketing](http://archive.ics.uci.edu/ml/datasets/Bank+Marketing) |
| BNG Ionosphere                  | 100,000*| 34       | 2       | [www.openml.org/d/146](http://www.openml.org/d/146)                 |
| BNG Zoo                         | 100,000*| 16       | 7       | [www.openml.org/d/272](http://www.openml.org/d/272)                |
| Churn Modeling                  | 10,000  | 10       | 2       | [www.kaggle.com/barelydedicated/bank-customer-churn-modeling](http://www.kaggle.com/barelydedicated/bank-customer-churn-modeling) |
| Creditcard Default              | 30,000  | 18†      | 2       | [archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients](http://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients) |
| Forest Cover Type               | 100,000*| 54       | 7       | [archive.ics.uci.edu/ml/datasets/covertype](http://archive.ics.uci.edu/ml/datasets/covertype) |
| Higgs Boson                     | 250,000*| 23       | 2       | [archive.ics.uci.edu/ml/datasets/HIGGS](http://archive.ics.uci.edu/ml/datasets/HIGGS) |
| Lending Club Q1                 | 78,629  | 19†      | 2†      | [www.kaggle.com/wendykan/lending-club-loan-data/data](http://www.kaggle.com/wendykan/lending-club-loan-data/data) |
| Lending Club Q2                 | 118,823 | 19†      | 2†      | [www.kaggle.com/wendykan/lending-club-loan-data/data](http://www.kaggle.com/wendykan/lending-club-loan-data/data) |
| Network Attack                  | 25,192  | 41       | 2       | [www.kaggle.com/sampadab17/network-intrusion-detection](http://www.kaggle.com/sampadab17/network-intrusion-detection) |
| Phishing                        | 11055   | 30       | 2       | [www.openml.org/d/4534](http://www.openml.org/d/4534)               |
| Pulsar                          | 17,898  | 8        | 2       | [archive.ics.uci.edu/ml/datasets/HTRU2](http://archive.ics.uci.edu/ml/datasets/HTRU2) |
| SDSS                            | 10,000  | 11†      | 3       | [www.kaggle.com/lucidem/sloan-digital-sky-survey](http://www.kaggle.com/lucidem/sloan-digital-sky-survey) |
| Waveform                        | 5000    | 21       | 3       | [archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator](http://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator) |

Table 6: Dataset size information. Some datasets have been modified from their publicly available versions. A * indicates that this dataset was downsampled from its publicly available version. A † indicates that a modified set of class labels was used (as described in the text).

| Predictor          | SE (TL) | BS (TL) | C (TL) | I (TL) | UM  |
|--------------------|---------|---------|--------|--------|-----|
| Confidence         | 2.88±0.16 | 2.35±0.16 | 2.23±0.17 | 2.13±0.14 | 2.11±0.13 | 1.83±0.12 | 1.46±0.13 | 1.14±0.09 |
| Crossval           | 2.57±0.15 | 2.06±0.14 | 1.72±0.13 | 1.91±0.13 | 1.91±0.13 | 1.88±0.13 | 1.67±0.12 | 1.24±0.09 |
| Dropout            | 3.09±0.16 | 2.50±0.16 | 2.20±0.15 | 2.13±0.14 | 2.13±0.14 | 2.10±0.13 | 1.86±0.13 | 1.38±0.09 |
| Meta-model         | 2.41±0.17 | 1.89±0.16 | -       | 1.77±0.13 | 1.77±0.15 | 1.83±0.13 | -       | 1.33±0.12 |

Table 7: Results with standard errors for all 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 results. These costs are averaged over the ten values of $\alpha$ shown in the Fig. 2 (0.5 to 0.95).