Under Review: *Machine Learning*

Self-Consistent Machine Learning: An Ensemble Smoothing Approach Based on Prediction Confidence

Liben Chen
University of Minnesota, chen7954@umn.edu

Mochen Yang
University of Minnesota, yang3653@umn.edu

Gediminas Adomavicius
University of Minnesota, gedas@umn.edu

Abstract

Suppose a machine learning (ML) model is re-trained after the arrival of some new data; however, all the new data instances are such that the model's prior predictions for them were perfectly accurate. Are the predictions of the updated model consistent with its prior predictions? Such self-consistency of an ML system is an important factor for users' acceptance of and trust in the ML systems. In this study, we design an ensemble approach for improving the self-consistency of the ML system, based on a smoothing technique. Smoothing adds a pseudo-labeled dataset with the model's own predictions into the original training data with an explicit goal of improving self-consistency. Our ensemble approach uses smoothing in a prediction-confidence-aware manner and significantly outperforms generic smoothing methods in terms of both self-consistency and predictive performance. This study contributes to understanding the nuances of self-consistency enhancement in ML systems and provides practical insights for designing more robust and trustworthy decision support systems.

Keywords: machine learning, performance metrics, self-consistency, robustness, smoothing, ensemble learning

1 Introduction

Evaluating the performance of machine learning (ML) systems has always been an important issue. Although vast majority of ML research focuses on predictive-accuracy-based performance metrics, several other perspectives on ML performance measurement have been proposed in prior literature, including ML robustness (Braiek and Khomh 2025, Drenkow et al. 2021, Bansal et al. 2019b, Mukherjee et al. 2006). In this work, we focus on the notion of *self-consistency*, which measures certain aspects of the inherent consistency among the predictions made by an ML system. In particular, *self-consistency* is defined as the consistent agreement of predictions made on the same data instance by the same ML system when *any new incoming training data are in complete agreement with the system's prior predictions* (Adomavicius and Zhang 2012, 2014).[1] For example, consider a trained ML system that predicts that a credit card transaction i is fraudulent but transaction j is not. Suppose transaction i indeed turns out to be a fraud, and the system is retrained with this additional data instance whose label coincides with the system's prior prediction. Now the retrained system makes a prediction on transaction j again. If the system completely changes its previous prediction (i.e., when the *only* new training data is completely aligned with the system's own prior predictions), it might indicate that the ML system is not self-consistent. Such inconsistent predictions may harm users' trust in an ML system and lower the user's acceptance of future suggestions by the system (Komiak and Benbasat 2006, O'Donovan and Smyth 2005).

Prior research has shown that smoothing (or "self-training") can be an effective method for improving the self-consistency of ML systems (Adomavicius and Zhang 2014). Smoothing works by retraining the ML system on additional data instances labeled with the system's own predictions. Previous studies have demonstrated that smoothing is effective in improving self-consistency without negative impact on the prediction performance (Adomavicius and Zhang 2012, 2014).

This study extends prior work by incorporating an important factor, the ML model's *prediction confidence*, to develop a more nuanced smoothing approach, and we test the impact of this approach on both self-consistency and predictive performance of ML systems. We provide computational evidence that smoothing based on the segments of feature space where the model produces high- vs. low-confidence predictions can significantly affect the overall model performance. Specifically, we show that smoothing using high-confidence data instances can bring some predictive performance improvements but minimal self-consistency gains, whereas smoothing with low-confidence instances can significantly improve self-consistency but typically incur predictive performance losses.

Based on these insights, we design an ensemble approach that takes advantage of smoothing with both low- and high-confidence instances. Through comprehensive computational experiments, we show that this ensemble approach is significantly more advantageous than smoothing uniformly or with low-confidence (or high-confidence) instances alone.

[1] "Consistency" can encompass a broad range of notions, including prediction stability with respect to small input perturbations or with respect to retraining on various types of new data, e.g., data that agree or disagree with the model's prior predictions, or data that are noisy, or data with uncertainty in its ground-truth labels (for instance, due to annotator inconsistency), or out-of-distribution data. The specific notion we focus on in this paper represents a basic level of stability: consistency of predictions in cases when all incoming data are in *complete agreement* with the model's prior predictions.

2 Related Work

2.1 Robustness-Oriented ML and Self-Consistency

Although predictive performance has been a primary focus of ML methods development, many studies also focus on robustness-oriented aspects of ML. For example, algorithmic stability defines a fundamental property in ML that quantifies the extent to which a model's output varies in response to minor alterations in its training data. Prior research utilizes algorithmic stability as a theoretical construct to establish analytical bounds on the generalization error of ML models, demonstrating that models exhibiting greater stability tend to generalize more effectively (Bousquet and Elisseeff 2002, Elisseeff et al. 2005, Agarwal and Niyogi 2005, Mukherjee et al. 2006, Agarwal and Niyogi 2009). This improved generalization arises because such models are less sensitive to changes in training data, thus reducing the risk of overfitting. In recent decades, various definitions of algorithmic stability have emerged, including leave-one-out stability, uniform stability, and hypothesis stability (Bousquet and Elisseeff 2002, Elisseeff et al. 2005, Agarwal and Niyogi 2005, Mukherjee et al. 2006, Agarwal and Niyogi 2009). Each of these stability measures provides either tight or loose bounds on the generalization error of ML models.

More recently, backward compatibility of ML models is studied as another important robustness-oriented aspect of ML models in human-computer interaction (HCI) literature. Backward compatibility is defined as the ability to minimize the difference in the model's predictions between two versions of the model as a result of the model update (e.g., retraining with new training instances, upgrades of the model's architecture). Research in HCI has demonstrated, through empirical user studies, that more backward compatible ML models facilitate appropriate levels of user reliance on model predictions (Bansal et al. 2019b). This is because the predictable behavior of backward-compatible ML models enhances users' ability to discern when the model's predictions can be trusted (Bansal et al. 2019b). These studies provide substantial practical implications for fostering user trust and reliance in the human-AI collaboration process (Bansal et al. 2019a,b).

Within the broad landscape of robustness-oriented ML, we examine *self-consistency* as a specific dimension of robustness-oriented ML, because it captures an essential yet often overlooked robustness-oriented characteristic of ML models: their ability to maintain consistency in their predictions when retrained on additional training data that *aligns with their prior predictions*. Unlike the notion of backward compatibility, self-consistency specifically focuses on scenarios where new training data confirms rather than contradicts prior predictions. This distinction is important because, in such cases, changes in predictions after retraining would likely be unexpected and could indicate deeper issues in the model's learning dynamics. Ensuring self-consistency, therefore, helps preserve the logical coherence of a model's behavior and strengthens users' confidence in its predictive reliability.

In this study, we adopt the notion of *self-consistency* that was first introduced in the context of one popular family of ML algorithms, the recommender systems (RS) (Adomavicius and Zhang 2012). The prior work examined the impact of several factors (e.g., choices of RS algorithms, sparsity of the training data, data normalization technique) on the self-consistency of recommendation algorithms. For example, neighborhood-based RS algorithms, such as ItemKNN, demonstrated highest instability among tested algorithms in the study (Adomavicius and Zhang 2012). It was also reported that the algorithms are often less self-consistent

when the training data is sparser or when certain data normalization techniques are applied (Adomavicius and Zhang 2012). Later, several techniques, such as bagging and iterative smoothing, were proposed to improve self-consistency of RS (Adomavicius and Zhang 2014); the computational evidence showed that smoothing can significantly improve self-consistency, typically without a negative impact on predictive performance. In addition, multiple metrics have been proposed for measuring self-consistency for different predictive ML tasks: classification, regression, and ranking (Adomavicius and Zhang 2016). We build upon this prior work to go beyond recommender systems and extend the discussion on *self-consistency* to general ML algorithms. Furthermore, we substantially advance the understanding of smoothing as a mechanism for enhancing self-consistency by examining the differential effects of prediction-confidence-based smoothing strategies.

2.2 Semi-Supervised Learning

Smoothing can be interpreted as a form of *self-training*, a classical approach within the broader field of semi-supervised learning (SSL) (Van Engelen and Hoos 2020, Schmarje et al. 2021). SSL methods leverage both labeled and unlabeled data to improve model performance, and self-training specifically enhances learning by iteratively generating pseudo-labels for unlabeled instances and incorporating them into the training process. This technique has been widely applied in deep learning, yielding substantial predictive performance improvements in computer vision (Xie et al. 2020, Zoph et al. 2020, Liu et al. 2021, Yang et al. 2022) and natural language processing models (Du et al. 2020, Kahn et al. 2020, Xu et al. 2021a). The core idea behind self-training aligns with the concept of smoothing, as both methods involve augmenting the training dataset with pseudo-labeled instances using the model's own predictions.

A key challenge in SSL (including self-training) is the potential introduction of noise through incorrect pseudo-labels, which can negatively impact model performance. To mitigate this issue, many SSL approaches incorporate confidence-based selection mechanisms to prioritize more reliable pseudo-labels (Lee et al. 2013, Xu et al. 2021b, Chen et al. 2023, Sohn et al. 2020). This is often achieved by using the prediction confidence score of pseudo-labeled instances as a hard or soft filtering criterion. Intuitively, more confident pseudo-labels are more likely to be correct. As an example of hard filtering, the Dash approach (Xu et al. 2021b) uses a high prediction confidence threshold when the average training loss is large and decreases the threshold as the training loss reduces. This helps the ML models include fewer pseudo-labeled instances when they are less accurate. As an example of soft filtering, the SoftMatch approach (Chen et al. 2023) adjusts the weight of a pseudo-labeled instance in the loss function by its prediction confidence – highly confident pseudo-labeled instances get higher weight.

Inspired by these insights, in this study we integrate perspectives from both the SSL and self-consistency literature to develop a confidence-aware ensemble smoothing algorithm that aims to enhance both predictive performance and self-consistency of ML models.

3 Problem Background

3.1 Problem Statement

This study focuses on a specific ML task, classification. In this task, we are given a dataset $D : (X, y)$ with features X and a categorical target variable y (which could be binary or multi-class). The ML model, f, models the probability distribution of the target conditional on input features, $P(\hat{y}|X) = f(X)$. This model is trained with labeled data, $D^{train} : (X, y)$, in a supervised learning manner. In many real-world applications, often there are significant unlabeled data available as well.[2] We denote the unlabeled dataset as D^{unlab} and assume it is potentially much larger than the labeled dataset. This assumption reflects realistic situations where obtaining labels can be costly and unlabeled data is abundantly available. Next, we introduce the terminology and measures related to self-consistency and predictive performance.

3.2 Self-Consistency, Accuracy, and Overall Utility

Self-consistency is defined as the extent to which the ML system's predictions for the same data instance stay the same (i.e., consistent) when any new incoming data to the system is in complete agreement with the system's prior predictions (Adomavicius and Zhang 2012). Following the literature, we apply a two-phase approach (Figure 1) to measuring the self-consistency of an ML system. In Phase 1, we train an ML model on $D^{train} : (X, y)$. We denote the trained model in Phase 1 as f_1. It can be used to estimate the probability distribution of target variable for test instances in D^{test}, namely, $f_1(X^{test}) = P(\hat{y}_1^{test}|X^{test})$. The final categorical prediction $\hat{y}_1^{test}$ is typically obtained by thresholding the probability score.

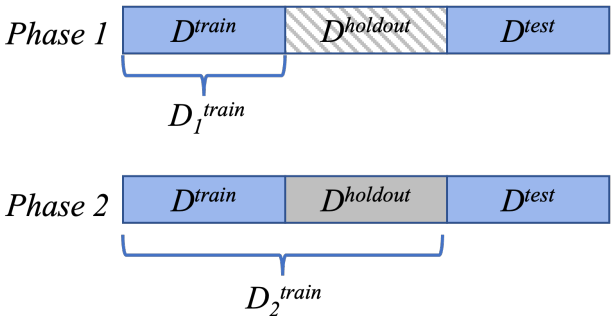


Figure 1: Two-Phase Evaluation of Self-Consistency. (The blue color indicates labeled datasets and the grey color indicates unlabeled datasets. $D^{holdout}$ is not utilized in Phase 1.)

In Phase 2, an ML model with the same hyper-parameter configuration is trained from scratch on D_2^{train}, consisting of both D^{train} and $D^{holdout}$. $D^{holdout}$ is a holdout dataset from the unlabeled dataset D^{unlab}. It is solely used for the specific purpose of evaluating the self-consistency of ML techniques, and we use predictions from the trained model in Phase 1 as pseudo-labels for $D^{holdout}$. I.e., we treat $D^{holdout}$ as a batch of hypothetical new incoming data that completely agrees with the system's prior predictions, which we later also refer to as self-agreeing measuring instances. Finally, the model trained on D_2^{train} in Phase 2 gives the final prediction $\hat{y}_2^{test}$ for D^{test}.

[2] E.g., in an image classification application, these would be images that have not yet been tagged by human experts.

Using this two-phase setup, we measure the consistency between the predictions made in Phase 1 vs. Phase 2 on the same D^{test} with several metrics. Following the previous literature on defining the self-consistency metrics in accordance with the predictive performance metrics of interest (Adomavicius and Zhang 2016), we define self-consistency analogues to standard Accuracy, Precision, Recall, and F1 metrics, denoted SCAcc, SCPr, SCRe, SCF1, respectively. In Tables 1 and 2, we provide definitions of these metrics in terms of binary classification. They can be straightforwardly adapted to multi-class classification.

Table 1: Confusion Matrix (for Measuring Self-Consistency / Accuracy) in Binary Classification Tasks

Measuring Accuracy			Measuring Self-consistency		
	Positive Prediction	Negative Prediction		Positive Pred. at Phase 2	Negative Pred. at Phase 2
Positive Ground Truth	True Positive (TP)	False Negative (FN)	Positive at Phase 1	Positive → Positive (PP)	Positive → Negative (PN)
Negative Ground Truth	False Positive (FP)	True Negative (TN)	Negative at Phase 1	Negative → Positive (NP)	Negative → Negative (NN)

Table 2: Self-Consistency and Accuracy Metrics for Binary Classification

Self-consistency Metric	Formula	Accuracy Metric	Formula
Self-Consistency Precision (SCPr)	$\frac{PP}{PP+NP}$	Precision (Pr)	$\frac{TP}{TP+FP}$
Self-Consistency Recall (SCRe)	$\frac{PP}{PP+PN}$	Recall (Re)	$\frac{TP}{TP+FN}$
Self-Consistency Accuracy (SCAcc)	$\frac{PP+NN}{PP+PN+NP+NN}$	Overall Accuracy	$\frac{TP+TN}{TP+FP+FN+FP}$
Self-Consistency F1 (SCF1)	$\frac{2\times SCPr\times SCRe}{SCPr+SCRe}$	F1	$\frac{2\times Pr\times Re}{Pr+Re}$

Specifically, Self-Consistency Accuracy (SCAcc) reflects the percentage of consistent predictions (across all classes) between Phases 1 and 2. For a particular (e.g., positive) class, Self-Consistency Precision (SCPr) is defined as the number of predictions that were positive in both Phases 1 and 2 (i.e., PP in Table 1), divided by the number of positive predictions in Phase 2 (i.e., $PP + NP$ in Table 1). Similarly, Self-Consistency Recall (SCRe) is defined as the number of predictions that were positive in both Phases 1 and 2 (i.e., PP in Table 1), divided by the number of positive predictions in Phase 1 (i.e., $PP + PN$ in Table 1). Finally, the Self-Consistency F1 (SCF1) is the harmonic mean of SCPr and SCRe.

We also measure the ML system's predictive performance with standard F1 and Accuracy scores. Ideally, we want a classifier to be both self-consistent and accurate at the same time; however, the two may not be simultaneously achievable. Therefore, to take both performance aspects into account, we adopt a multi-objective analysis framework and design an overall utility score that is the weighted sum of percentage improvements in self-consistency and in predictive performance over a certain baseline model (e.g., a model trained on D^{train} without any smoothing). Specifically, the utility score U is calculated as

$$U(S, A) = \alpha \frac{S - S_{base}}{S_{base}} + (1 - \alpha) \frac{A - A_{base}}{A_{base}} \tag{1}$$

where S and A are the self-consistency and predictive performance scores of the model, respectively. Similarly, S_{base} and A_{base} are the self-consistency and predictive performance scores of the baseline model, respectively. Calculating percentage (rather than absolute) improvements ensures that self-consistency and predictive performance changes over the baseline model are normalized to the same scale, and the weight $\alpha \in [0, 1]$ determines the relative importance of self-consistency vs. predictive performance in the multi-objective

assessment.

When applying this multi-objective analysis, we calculate utility score U_{F1} for the related pair of metrics, Self-Consistency F1 and the F1 score. Similarly, we calculate utility score U_{Acc} for the related pair, Self-Consistency Accuracy and the Accuracy score. These pairings ensure that metrics in our multi-objective analysis are of the same nature.

4 Smoothing

Smoothing is a data augmentation technique that utilizes a part of unlabeled data, D^{smooth}, sampled from $D^{unlab} \backslash D^{holdout}$. After the model is trained on the original training dataset D^{train}, the model is applied to make predictions $\hat{y}$ on D^{smooth}. Using these predictions, the model is trained from scratch again on the union of $D^{train} : (X, y)$ and $D^{smooth} : (X, \hat{y})$. We also refer to D^{smooth} as *smoothing data*. In this process, the model's prior knowledge is reinforced by feeding its predictions back to the model. Thus, the model becomes more robust to potential alterations of the decision boundary due to introduction of new data instances that are introduced during future updates. In prior literature (Adomavicius and Zhang 2014, 2012), smoothing is typically carried out uniformly, i.e., utilizing all of $D^{unlab} \backslash D^{holdout}$ or a random sample of it. Following the same two-phase procedure discussed above, we can evaluate the self-consistency of a smoothed model, as shown in Figure 2. The main difference from the evaluation without smoothing is that the models in both Phases 1 and 2 are trained with an additional dataset D^{smooth}.

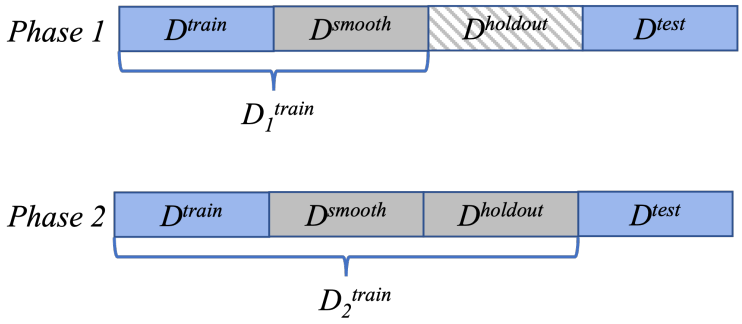


Figure 2: Two-Phase Evaluation of Self-Consistency with Smoothing (The blue color indicates labeled datasets and the grey color indicates unlabeled datasets. $D^{holdout}$ is not utilized in Phase 1.)

4.1 Prediction Confidence and Confidence-Based Smoothing

One of the key contributions of this study is to demonstrate (and to take advantage of the fact) that smoothing can have *differential effects* on self-consistency / predictive performance depending on the prediction confidence of smoothing data instances.

Intuitively, Low-Confidence Smoothing adds low-confidence data instances (i.e., those closer to the decision boundary and, thus, more uncertain) into the training dataset, while High-Confidence Smoothing adds high-confidence instances (i.e., those farther away from the decision boundary and, thus, more certain). To define Low- vs. High-Confidence Smoothing formally, we first define the concept of prediction confidence, which measures the extent to which the classifier is certain about a particular prediction. Given a class

probability distribution of the target variable from a trained model $P(\hat{y}|X_i)$ for the data instance X_i,[3] we calculate the confidence score as

$$C(X_i) = P^{(1)}(\hat{y}|X_i) - P^{(2)}(\hat{y}|X_i) \tag{2}$$

where $P^{(k)}$ is the k^{th} largest probability score in the ranked list of class probability scores from distribution $P(\hat{y}|X_i)$. In other words, for a given instance X_i, it is the difference in the probability score for the highest vs. second-highest predicted class label for that instance.

In binary classification, the prediction confidence score is equivalent to the difference between the maximum and minimum class probability scores. The model is more confident about its prediction if this difference is large. When the maximum class probability score is equal to the minimum (i.e., both are 0.5), the confidence score is appropriately 0, representing lowest prediction certainty. There are other choices for measuring prediction confidence, such as negative entropy of the prediction probability distribution (Tyralis and Papacharalampous 2024, Hüllermeier and Waegeman 2021). We chose this formulation because of its simplicity and interpretability.

Algorithm 1 Prediction-Confidence-Based Smoothing

SMOOTH$(f, SmConf_{lower}, SmConf_{upper}, SmSize)$

1. **Initial Model Training:** Train an ML model f on D^{train}.
2. **Prediction on Unlabeled Data:** Use f to make predictions on $D^{unlab} \setminus D^{holdout}$.
3. **Prediction Confidence Scoring:** Calculate the confidence score of each prediction for $D^{unlab} \setminus D^{holdout}$.
4. **Smoothing Sample Selection:** Select D^{smooth} of size $SmSize$ from $D^{unlab} \setminus D^{holdout}$ whose confidence score percentile lies between $SmConf_{lower}$ and $SmConf_{upper}$.
5. **Model Retraining:** Retrain f on $D = D^{train} \cup D^{smooth}$ to obtain model f'.
6. **Return:** Output f'.

In confidence-based smoothing, we choose smoothing instances from $D^{unlab} \setminus D^{holdout}$ based on the model's prediction confidence score for these instances, instead of choosing the instances uniformly at random. Specifically, with Low-Confidence Smoothing, we randomly sample smoothing instances from a subset of $D^{unlab} \setminus D^{holdout}$ with prediction confidence scores that are below SC_L percentile. Similarly, with High-Confidence Smoothing, we randomly sample smoothing instances from a subset of $D^{unlab} \setminus D^{holdout}$ with prediction confidence scores that are above SC_H percentile. Smoothing confidence thresholds SC_L and SC_H can be fine-tuned as hyperparameters during the learning process.

Algorithm 1 describes the general procedure for prediction-confidence-based smoothing. It can be instantiated as Uniform, Low-Confidence, or High-Confidence Smoothing by setting different values for the lower and upper bounds of the confidence percentiles for the smoothing instances, i.e., by setting $[SmConf_{lower}, SmConf_{upper}]$. Specifically, we set these bounds as $[0.0, 1.0]$ for Uniform Smoothing (i.e., to sample smoothing instances uniformly at random from unlabeled data), as $[0.0, SC_L]$ for Low-Confidence Smoothing (where SC_L can be fine-tuned as a hyperparameter according to the specific modeling objective), and as $[SC_H, 1.0]$ for High-Confidence Smoothing (where SC_H can be fine-tuned accordingly).

[3] We use X_i to represent the feature of a specific data instance indexed by i from the feature matrix X.

While both smoothing schemes serve to reinforce the model's learned knowledge, the reinforcing effect of Low-Confidence Smoothing leads to a more significant self-consistency improvement. Intuitively, data instances with lowest prediction confidence are typically the ones near the model's decision boundary. With only a small alteration in the model's decision boundary, the prediction of these instances can easily flip. Low-Confidence Smoothing introduces additional training data instances near the decision boundary with labels that are in complete agreement with the model's prior predictions, effectively reinforcing the model's prior decision boundary in the low-confidence region of the feature space and, thus, leading to self-consistency improvements. On the other hand, the decision-boundary-reinforcing effect of High-Confidence Smoothing is not likely to be as significant – intuitively, the model is expected to be inherently more self-consistent on data instances for which it has high prediction confidence.

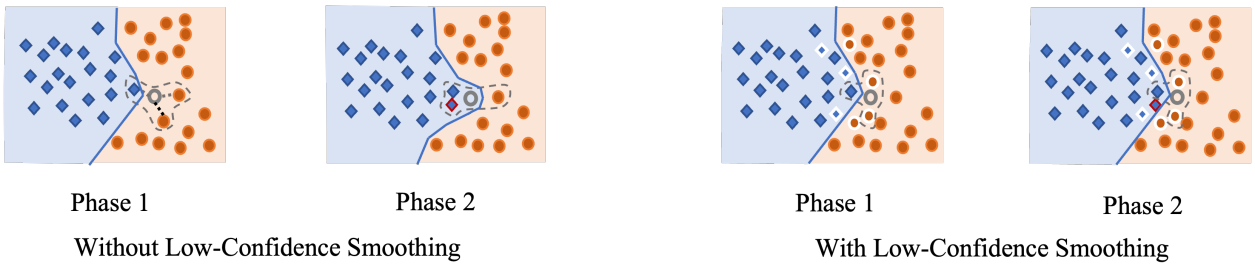
When it comes to the model's predictive performance, Low-Confidence Smoothing is much less likely to improve it (and may even harm it) by adding training instances whose pseudo-labels are very likely to be wrong. This potentially may introduce more noise into the training process. In contrast, High-Confidence Smoothing may even provide some predictive performance improvements by reinforcing the model's correct knowledge learned on the high-confidence feature space, for instance, through "denoising" of the occasional spurious, noisy data points in that space.

In Figure 3, we provide a stylized illustration of how Low-Confidence Smoothing can improve the self-consistency of a three-nearest-neighbor model. Notably, without Low-Confidence Smoothing, the decision boundary can change significantly from Phase 1 to Phase 2 in the two-phase self-consistency evaluation. In contrast, the decision boundary remains largely consistent with application of Low-Confidence Smoothing. This is achieved by adding low-confidence smoothing instances (indicated by the white lining in the third and fourth plots of Figure 3). Notably, when Low-Confidence Smoothing is not applied, adding the self-agreeing measuring instance[4] (indicated by the red lining in Figure 3) changes the composition of the three-nearest-neighbor set of the focal test instance (the circumference-only circle with thick gray lining) and flips its prediction from orange to blue. This is an indication of non-self-consistency. However, adding low-confidence smoothing instances (as shown in the third and fourth plots) maintains the orange-majority of the three-nearest-neighbor set of the focal test instance, and thus the prediction of the focal test instance remains unchanged, reinforcing the original decision boundary. This illustrates the improvement in self-consistency provided by Low-Confidence Smoothing.

Figure 4 provides a stylized illustration of how High-Confidence Smoothing can improve the accuracy of a three-nearest-neighbor model. These plots depict the high-confidence feature region with some minor noise and outliers (indicated by blue diamonds). Notably, without High-Confidence Smoothing, the noise and outliers induce a sizeable region of blue territory in a learned model and, as a result, the model ends up misclassifying several orange test instances (i.e., indicated by the yellow lining in Figure 4). As part of the High-Confidence Smoothing process, smoothing instances reduce the blue territory spanned by minor noise and outliers, correcting the misclassifications of the orange test instances. This illustrates the improvement in predictive accuracy provided by High-Confidence Smoothing.

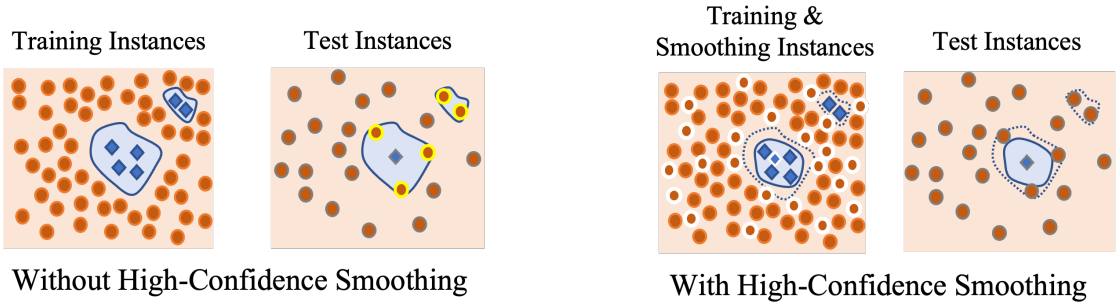
In addition to the above stylized illustrations, in Appendix D we provide a more systematic set of

[4]The self-agreeing measuring instances refer to $D^{holdout}$ added in Phase 2 only for self-consistency evaluation in Figure 1.



The circumference-only circle with thick gray lining is the focal test instance. Instances with red lining are the self-agreeing measuring instances added in Phase 2. Instances with white lining are smoothing instances. The rest of instances are regular training instances. The solid blue line represents the learned decision boundary.

Figure 3: Low-Confidence Smoothing Example with a Three-Nearest-Neighbor Model



The instance with the yellow lining is the misclassified test instance. The instance with the white lining is the smoothing instance. The solid blue line represents the learned decision boundary. The dotted line in the right pair of plots is the learned decision boundary from the model without High-Confidence Smoothing.

Figure 4: High-Confidence Smoothing Example with a Three-Nearest-Neighbor Model

simulation experiments on the predictive performance and self-consistency of k-NN models. The results further corroborate the differential effects of Low- vs. High-Confidence Smoothing.

4.2 Confidence-Based Ensemble Smoothing

As discussed earlier, the value of Low-Confidence Smoothing is most likely in providing significant self-consistency benefits in the low-confidence regions of the feature space (with some possible predictive performance losses). High-Confidence Smoothing is much less likely to provide comparable self-consistency benefits, but it can potentially maintain and even improve predictive performance in the high-confidence regions of the feature space. This leads to a natural ensemble approach to take advantage of both: use Low-Confidence Smoothing approach on test instances with low-confidence predictions, and High-Confidence Smoothing approach on test instances with high-confidence predictions. The confidence threshold $TestConf$ that separates test instances into low- vs. high-confidence can also be fine-tuned as a hyperparameter. In Algorithm 2, we summarize the Ensemble Smoothing approach.

4.3 Smoothing with Oversampling & Radius Sampling

Previously, we assumed the unlabeled dataset D^{unlab} is abundantly available, as unlabeled data collection is often automated and not costly in many application scenarios. Thus, sampling the smoothing dataset D^{smooth}

Algorithm 2 Ensemble Smoothing

EnsembleSmooth$(f, SC_L, SC_H, TestConf, SmSize)$

1 **Low-Confidence Smoothing:** $f^{\text{low}} = \text{Smooth}(f, 0.0, SC_L, SmSize)$.
2 **High-Confidence Smoothing:** $f^{\text{high}} = \text{Smooth}(f, SC_H, 1.0, SmSize)$.
3 **Generate Predictions from Ensemble:**
Set $\hat{y}^{test} = \{\}$.
For each $X_i \in D^{test}$:
if $C(X_i) \leq TestConf$ **then** $\hat{y}_i = f^{\text{low}}(X_i)$;
else $\hat{y}_i = f^{\text{high}}(X_i)$.
Update $\hat{y}^{test} = \hat{y}^{test} \cup \{\hat{y}_i\}$.
4 **Return:** Output $\hat{y}^{test}$.

Algorithm 3 Oversampling

Oversampling(X^{train}, N^{unlab})

1 **Random Sampling:** Randomly sample N^{unlab} instances from X^{train} with replacement to form D^{unlab}.
2 **Return:** Output D^{unlab}.

Algorithm 4 Radius Sampling

RadiusSampling(X^{train}, N^{unlab})

1 **Obtain Training Dataset Size:** Let N^{train} be the number of training instances in X^{train}.
2 **Compute Radius Sampling Size:** Set $n = N^{unlab}/N^{train}$ as the number of unlabeled instances to be sampled from the radius of each training instance.[5]
3 **Compute Empirical Covariance Matrix:** Compute the empirical covariance matrix $\hat{\Sigma}_{X^{train}}$ as an approximation to $\Sigma_{X^{train}}$.
4 **Sample Smoothing Instances within Radius:**
For each $X_i \in X^{train}$:
Randomly sample n instances from the normal distribution $N(X_i^{train} I, \hat{\Sigma}_{X^{train}})$.
5 **Pool:** Combine all sampled instances to form D^{unlab}.
6 **Return:** Output D^{unlab}.

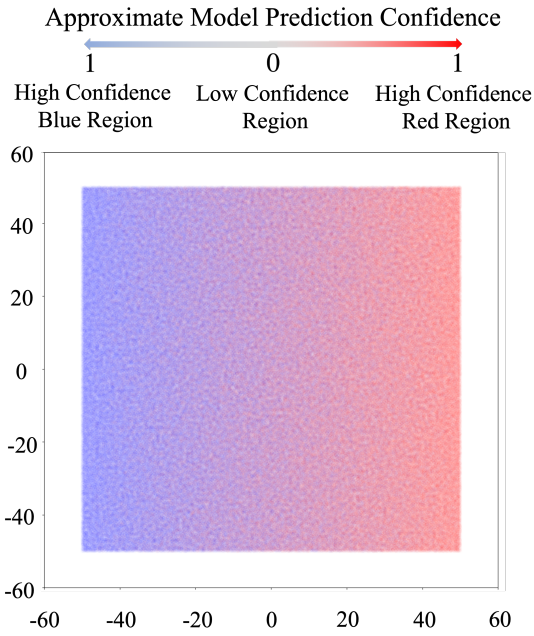
from D^{unlab} is feasible and convenient. However, in scenarios where D^{unlab} is not readily available, two alternative sampling approaches can be applied to *construct* D^{unlab}. The first approach, Oversampling, allows us to obtain actual realizations of data from the same underlying population as D^{unlab}. With Oversampling, we sample instances with replacement from the training dataset X^{train} to form D^{unlab} as if they were unlabeled. This approach maximizes the "realism" of each sample (i.e., every sampled instance is a realized data instance from the underlying population) while potentially sacrificing the coverage of the feature space.

The second approach, Radius Sampling, creates synthetic data to serve as D^{smooth}. With Radius Sampling, we directly sample the unlabeled data instances from the feature space. Specifically, for each training instance X_i^{train}, we sample a number of unlabeled data instances from the normal distribution $N(X_i^{train} I, \Sigma_{X^{train}})$, where I is the identity matrix and $\Sigma_{X^{train}}$ is the covariance matrix of the training feature matrix X^{train}. In practice, we approximate $\Sigma_{X^{train}}$ with the empirical covariance matrix. These sampled unlabeled data instances are pooled together to form D^{smooth} for smoothing. The details of Oversampling and Radius Sampling are summarized in Algorithms 3 and 4 respectively.

5 Experiments: Synthetic Data

5.1 Experiment Setup

To better understand the effects of Low-/High-Confidence Smoothing and the performance of the Ensemble Smoothing approach, we first conduct evaluations on a synthetic dataset, which allows us to have fine-grained control over label uncertainty in the feature space. We generate a two-dimensional synthetic dataset, $Synthetic$, visualized in Figure 5. It has two features, x_1 and x_2, each with values from $(-50, 50)$. The feature space is a 100×100 grid of 10,000 squares, with a side length of 1. We sample uniformly 1000 points in each square and assign labels to these points based on a Bernoulli distribution with a positive class ratio of $(x_2' + 50)\%$, where x_2' is the largest x_2 value within the area of that square. This ensures that different levels of uncertainty in the feature space are well represented in fine granularity in our $Synthetic$ dataset.



Blue instances are positive and red ones are negative.

Figure 5: Visualization of $Synthetic$

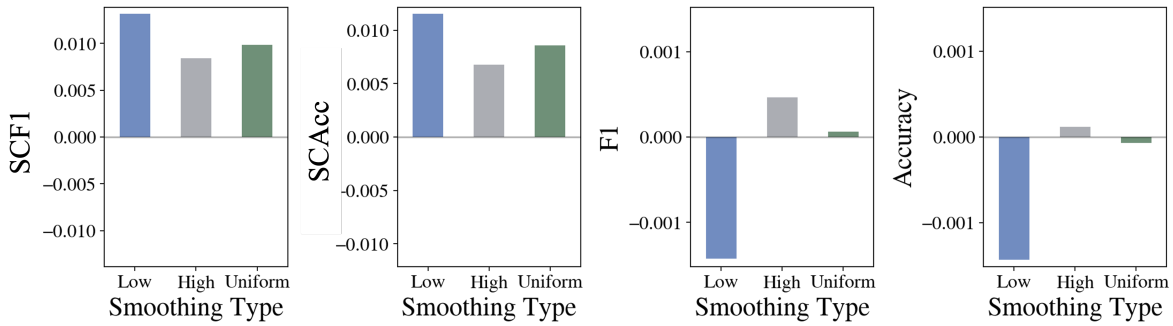
We examine five common ML techniques to generate our baseline models - Neural Net (NN), K-Nearest Neighbors (KNN) (Cover and Hart 1967), Support Vector Machine (SVM) (Cortes and Vapnik 1995), Random Forest (RF) (Breiman 2001), and Xgboost (Chen and Guestrin 2016).

We randomly partition the dataset into two equal sub-datasets, one of which is used without labels (i.e., D^{unlab}). We randomly sample D^{smooth} and $D^{holdout}$ from D^{unlab}. We then randomly sample D^{train}, D^{val}, and D^{test} from $D^{labeled}$. The baseline models without smoothing are also fine-tuned beforehand on a validation dataset, D^{val}. The predefined numbers of examples for these sample datasets can be found in Table B.3 of Appendix B. We use D^{train} for initial training, as detailed in Algorithms 1 and 2. D^{smooth} is used for smoothing. D^{val} is used for fine-tuning the ensemble smoothing model on hyperparameters – $TestConf$,

[5]We assume N^{unlab} is a multiple of N^{train} without loss of generality.

SC_L, SC_H. $D^{holdout}$ is only used in the second phase of self-consistency evaluation, as detailed in Figures 1 and 2. Finally, we evaluate the model's self-consistency and predictive performance performance on D^{test}.

We report the results from this evaluation process by calculating the utility score as the weighted average of the self-consistency and predictive performance improvements over a baseline, as defined in Equation (1). We select four values of weight $alpha$ (i.e., 0.0, 0.3, 0.5, 1.0) to represent the scenarios where self-consistency is not important, moderately important, equally important (to predictive performance), and most important, respectively. The baseline is the ML model trained on D^{train} without smoothing.[6] We evaluate each ML algorithm with four different smoothing schemes: Ensemble Smoothing, Low-Confidence Smoothing, High-Confidence Smoothing, and Uniform Smoothing. For reporting purposes, we average results from 30 runs (with different random seeds) of each algorithm-smoothing variation.[7] We also report statistical test results based on these 30 runs.[8]



"Low": Low-Confidence Smoothing, "High": High-Confidence Smoothing, "Uniform": Uniform Smoothing

Figure 6: Neural Net Performance on Synthetic Dataset under Different Smoothing Schemes

5.2 Results on Synthetic Data

In this section, we first provide computational evidence on the differential effects of Low- and High-Confidence Smoothing. Then, we present an in-depth computational evaluation of the proposed algorithm on the Synthetic dataset.

In Figure 6, as an illustration, we show the percentage change in the neural net[9] performance under three different smoothing schemes[10] with the Synthetic dataset. The change is calculated as the percentage of increase or decrease in the ML model performance after the smoothing, compared to the model performance before the smoothing. We see several common patterns across metrics. In terms of self-consistency performance, Low-Confidence Smoothing always offers the strongest benefit, and High-Confidence Smoothing always

[6] All baseline model performances are shown in Table B.1 and B.2 of Appendix B.

[7] We also fine-tune the key hyper-parameters of smoothing schemes – *TestConf*, SC_L, and SC_H – as described in Appendix B.

[8] When using statistical test to compare performance of smoothed models and the baseline non-smoothed models, we use two-sided pairwise t-test. When using statistical tests to compare performance of Ensemble Smoothing models vs. the other smoothing models, we use one-sided pairwise t-test to assess if Ensemble Smoothing significantly outperforms other smoothing models.

[9] The main performance patterns (i.e., Low-Confidence Smoothing often performs best in terms of self-consistency, while High-Confidence Smoothing performs best in terms of predictive performance) in general are consistent across different ML techniques, including KNN, SVM, RF, and XGBoost. Please see Appendix A.

[10] SC_L and SC_H are set at 50%. Recall that we sample smoothing instances with a prediction confidence score percentile in the interval $[0.0, SC_L]$ and $[SC_H, 1.0]$ for Low- and High-Confidence Smoothing, respectively.

Table 3: Synthetic Data: Utility of Different Smoothing Schemes

		U_{F1}						U_{Acc}			
Models	α	EnsembleSM	LowSM	HighSM	UniformSM	Models	α	EnsembleSM	LowSM	HighSM	UniformSM
NN	0	**0.0053** †††	0.0052 †††	0.0018 ***	0.0020 †***	NN	0	**0.0036** †††	0.0036 †††	0.0014 **	0.0015 †**
	0.3	**0.0058** †††	0.0057 †††	0.0021 †††***	0.0030 †††***		0.3	**0.0052** †††	0.0051 †††	0.0021 †††***	0.0033 †††**
	0.5	**0.0061** †††	0.0060 †††	0.0023 ††***	0.0041 †††**		0.5	**0.0062** †††	0.0061 †††	0.0025 ††***	0.0047 †††*
	1.0	**0.0076** †††	0.0072 ††	0.0028 **	0.0069 ††		1.0	**0.0094** †††	0.0089 †††	0.0035 ***	0.0083 ††
KNN	0	**0.0049** ††	0.0026	0.0049 ††	0.0035 †*	KNN	0	**0.0046** ††	0.0023	0.0046 ††	0.0033 †*
	0.3	**0.0114** †††	0.0111 †††	0.0103 †††	0.0097 †††		0.3	**0.0104** †††	0.0101 †††	0.0094 †††	0.0089 †††
	0.5	**0.0170** †††	0.0167 †††	0.0142 †††**	0.0141 †††**		0.5	**0.0156** †††	0.0154 †††	0.0129 †††***	0.0129 †††**
	1.0	**0.0312** †††	0.0308 †††**	0.0243 †††***	0.0259 †††**		1.0	**0.0288** †††	0.0285 †††**	0.0219 †††***	0.0243 †††**
SVM	0	**0.0062** †††	0.0060 †††	0.0060 †††	0.0053 ††	SVM	0	0.0052 †††	0.0049 ††	**0.0053** †††	0.0046 †
	0.3	**0.0086** †††	0.0086 †††	0.0061 †††**	0.0076 †††		0.3	**0.0071** †††	0.0071 †††	0.0052 †††**	0.0065 ††
	0.5	**0.0105** †††	0.0105 †††	0.0076 †††**	0.0091 †††		0.5	**0.0090** †††	0.0089 †††	0.0064 †††**	0.0077 †††
	1.0	**0.0191** †††	0.0190 †††	0.0112 †††***	0.0129 †††***		1.0	**0.0165** †††	0.0164 †††	0.0094 †††***	0.0110 †††**
RF	0	0.0051 †	-0.0024 **	0.0051 †	**0.0057** †	RF	0	0.0049 †	-0.0028 **	0.0050 ††	**0.0053** †
	0.3	0.0100 †††	0.0058 **	0.0097 †††	**0.0122** †††		0.3	0.0106 †††	0.0064 †**	0.0104 †††	**0.0129** †††
	0.5	0.0151 †††	0.0113 †††**	0.0137 †††	**0.0165** †††		0.5	0.0165 †††	0.0126 †††**	0.0149 †††	**0.0179** †††
	1.0	**0.0285** †††	0.0266 †††	0.0234 †††**	0.0274 †††		1.0	**0.0317** †††	0.0297 †††	0.0263 †††**	0.0305 †††
XGBoost	0	**0.0030** †††	0.0017 *	0.0029 †††**	0.0011 **	XGBoost	0	**0.0025** †††	0.0014 *	0.0024 ††**	0.0009 **
	0.3	**0.0063** †††	0.0056 †††	0.0062 †††**	0.0048 †††**		0.3	**0.0063** †††	0.0058 †††	0.0062 †††**	0.0051 †††**
	0.5	**0.0085** †††	0.0083 †††	0.0084 †††**	0.0078 †††		0.5	**0.0088** †††	0.0087 †††	0.0088 †††**	0.0083 †††
	1.0	**0.0164** †††	0.0161 †††	0.0139 †††***	0.0152 †††		1.0	**0.0177** †††	0.0173 †††	0.0151 †††***	0.0164 †††

"$*$", "$**$", and "$***$" mean p-value < 0.1, < 0.05, and < 0.01 respectively for one-sided pairwise t-test between EnsembleSM and the corresponding smoothed model. "$\dagger$", "$\dagger\dagger$", and "$\dagger\ \dagger\ \dagger$" mean p-value < 0.1, < 0.05, and < 0.01 respectively for two-sided pairwise t-test between the corresponding smoothed model and the model without smoothing. The best-performing smoothing model is highlighted in bold.

offers the least benefit. In contrast, High-Confidence Smoothing always offers the largest benefit on predictive performance. This evidence supports our intuitions explained in Section 4.1.

We now present the in-depth computational evaluation of the proposed approach on the Synthetic dataset. Table 3 shows the performance of different smoothing schemes summarized using the utility scores U_{F1} and U_{Acc}, as defined in Equation (1). We vary the relative importance of self-consistency (as compared to that of predictive performance), α, using four values: 0.0, 0.3, 0.5, 1.0. They represent different application scenarios, where self-consistency is not important, less important, equally important, and most important, respectively.

The key observation from Table 3 is that the proposed Ensemble Smoothing approach increases the model utility significantly as compared to the no-smoothing baseline. It also outperforms other smoothing schemes (i.e., Low-Confidence, High-Confidence, and Uniform Smoothing) in the vast majority of cases across different ML techniques. In cases where Ensemble Smoothing does not exhibit the best overall smoothing performance on this dataset (e.g., in some experiments with the Random Forest technique), the difference in performance between Ensemble Smoothing and the best-performing smoothing alternative is not statistically significant (not even at 0.1 level using one-sided pairwise t-test) in each case.

Table 4: Utility of Different Smoothing Schemes on Real-World Structured Data

		U_{F1}						U_{Acc}			
Models	α	EnsembleSM	LowSM	HighSM	UniformSM	Models	α	EnsembleSM	LowSM	HighSM	UniformSM
			Diabetes						Diabetes		
NN	0	**0.0020**	-0.0008	0.0020	-0.0025 *	NN	0	**0.0019**	-0.0007	0.0018	-0.0028 **
	0.3	**0.0120** †††	0.0083 †††**	0.0084 †††***	0.0071 ††***		0.3	**0.0108** †††	0.0076 †††**	0.0075 †††***	0.0059 ††***
	0.5	**0.0201** †††	0.0143 †††***	0.0129 †††***	0.0135 †††***		0.5	**0.0176** †††	0.0132 †††***	0.0116 †††***	0.0117 †††***
	1.0	**0.0406** †††	0.0295 †††***	0.0271 †††***	0.0295 †††***		1.0	**0.0361** †††	0.0271 †††***	0.0244 †††***	0.0262 †††***
KNN	0	**-0.0054**	-0.0057	-0.0262 ††*	-0.0268	KNN	0	**-0.0013**	-0.0086 †**	-0.0013	-0.0063 ††**
	0.3	**0.0450** ††	0.0424 †	0.0369 †††	0.0187		0.3	**0.0279** †††	0.0217 †††**	0.0279 †††	0.0250 †††**
	0.5	**0.0806** †††	0.0744 ††	0.0790 †††	0.0490		0.5	**0.0505** †††	0.0423 †††**	0.0505 †††	0.0458 †††**
	1.0	**0.1843** †††	0.1544 †††	0.1843 †††	0.1247 ††*		1.0	**0.1094** †††	0.0936 †††**	0.1094 †††	0.0979 †††**
SVM	0	**0.0134** ††	0.0055 *	-0.0009 **	-0.0151 †***	SVM	0	**0.0017**	-0.0052 †††***	0.0007	-0.0082 ††***
	0.3	**0.0412** †††	0.0211 †††***	0.0002 ***	0.0035 ***		0.3	**0.0218** †††	0.0061 †††***	0.0075 †††***	0.0052 ††***
	0.5	**0.0605** †††	0.0315 †††***	0.0078 ***	0.0159 ††***		0.5	**0.0366** †††	0.0137 †††***	0.0162 †††***	0.0141 †††***
	1.0	**0.1098** †††	0.0712 †††***	0.0390 †††***	0.0468 †††***		1.0	**0.0745** †††	0.0326 †††***	0.0397 †††***	0.0365 †††***
RF	0	**0.0045** †	0.0043	-0.0091 †***	-0.0090 †††***	RF	0	**-0.0001**	-0.0007	-0.0043 *	-0.0031
	0.3	**0.0163** †††	0.0159 †††	0.0091 ††	0.0085 ††*		0.3	**0.0093** †††	0.0088 †††	0.0083 †††	0.0093 †††
	0.5	**0.0266** †††	0.0261 †††	0.0213 †††	0.0201 †††		0.5	0.0172 †††	0.0157 †††	0.0172 †††	**0.0176** †††
	1.0	**0.0523** †††	0.0516 †††	0.0521 †††	0.0492 †††		1.0	0.0390 †††	0.0331 †††**	**0.0398** †††	0.0383 †††
XGBoost	0	0.0080 ††	-0.0010 **	**0.0084** ††	-0.0025 **	XGBoost	0	**0.0044** ††	-0.0014 **	0.0044 †	0.0003 *
	0.3	**0.0244** †††	0.0151 †††**	0.0210 †††*	0.0173 †††		0.3	**0.0152** †††	0.0105 †††***	0.0144 †††	0.0135 †††
	0.5	**0.0379** †††	0.0289 †††***	0.0342 †††	0.0305 †††**		0.5	**0.0253** †††	0.0201 †††***	0.0230 †††*	0.0223 †††*
	1.0	**0.0761** †††	0.0636 †††***	0.0670 †††**	0.0634 †††***		1.0	**0.0507** †††	0.0438 †††***	0.0445 †††**	0.0444 †††**
			Adults						Adults		
NN	0	0.0027 †††	0.0003 **	**0.0033** †††	0.0011 **	NN	0	0.0035 †††	0.0006 ***	**0.0044** †††	0.0010 ***
	0.3	**0.0114** †††	0.0110 †††	0.0089 †††***	0.0075 †††***		0.3	**0.0051** †††	0.0047 †††**	0.0048 †††	0.0032 †††***
	0.5	**0.0192** †††	0.0181 †††	0.0129 †††***	0.0118 †††***		0.5	**0.0076** †††	0.0074 †††**	0.0060 †††**	0.0047 †††***
	1.0	**0.0396** †††	0.0376 †††***	0.0229 †††***	0.0224 †††***		1.0	**0.0153** †††	0.0145 †††***	0.0089 †††***	0.0084 †††***
KNN	0	**0.0006**	-0.0004	-0.0002	0.0002	KNN	0	**0.0008**	0.0004	0.0004	0.0006
	0.3	**0.0212** †††	0.0195 †††*	0.0140 †††***	0.0151 †††***		0.3	**0.0070** †††	0.0065 †††**	0.0046 †††***	0.0050 †††***
	0.5	**0.0360** †††	0.0328 †††**	0.0236 †††***	0.0250 †††***		0.5	**0.0113** †††	0.0106 †††**	0.0077 †††***	0.0080 †††***
	1.0	**0.0728** †††	0.0660 †††**	0.0476 †††***	0.0498 †††***		1.0	**0.0218** †††	0.0208 †††	0.0154 †††***	0.0154 †††***
SVM	0	0.0022	-0.0022 ***	**0.0025**	0.0004 ***	SVM	0	0.0006	-0.0022 *	**0.0008**	-0.0009 ***
	0.3	**0.0200** †††	0.0183 †††**	0.0111 †††***	0.0122 †††***		0.3	**0.0046** †††	0.0036 †††**	0.0024 †††**	0.0024 ††***
	0.5	**0.0338** †††	0.0320 †††**	0.0177 †††***	0.0201 †††***		0.5	**0.0082** †††	0.0074 †††**	0.0043 †††***	0.0046 †††***
	1.0	**0.0682** †††	0.0662 †††*	0.0341 †††***	0.0398 †††***		1.0	**0.0176** †††	0.0171 †††*	0.0089 †††***	0.0100 †††***
RF	0	-0.0010	-0.0045 †††***	**-0.0007**	-0.0025 †	RF	0	-0.0008	-0.0037 †††***	**-0.0005**	-0.0021 ††*
	0.3	**0.0137** †††	0.0123 †††***	0.0071 †††***	0.0090 †††***		0.3	**0.0034** †††	0.0025 †††***	0.0009 ***	0.0019 ††**
	0.5	**0.0252** †††	0.0234 †††***	0.0145 †††***	0.0166 †††***		0.5	**0.0072** †††	0.0065 †††***	0.0038 †††***	0.0046 †††***
	1.0	**0.0540** †††	0.0513 †††***	0.0332 †††***	0.0357 †††***		1.0	**0.0175** †††	0.0167 †††***	0.0108 †††***	0.0113 †††***
XGBoost	0	**-0.0001**	-0.0004	-0.0004	-0.0024 ††**	XGBoost	0	-0.0001	-0.0001	**0.0003**	-0.0022 ††**
	0.3	**0.0163** †††	0.0156 †††*	0.0074 †††***	0.0100 †††***		0.3	**0.0046** †††	0.0043 †††*	0.0019 †††***	0.0021 †††***
	0.5	**0.0279** †††	0.0267 †††**	0.0143 †††***	0.0183 †††***		0.5	**0.0083** †††	0.0079 †††*	0.0043 †††***	0.0049 †††***
	1.0	**0.0571** †††	0.0544 †††**	0.0315 †††***	0.0380		1.0	**0.0178** †††	0.0170 †††**	0.0105 †††***	0.0121 †††***
			Survival						Survival		
NN	0	**0.0074** †††	0.0051 ††	0.0059 †	-0.0005 ***	NN	0	0.0051 †††	0.0024 *	**0.0064** ††	-0.0005 **
	0.3	**0.0092** †††	0.0074 †††	0.0074 †††	0.0039 ††***		0.3	0.0069 †††	0.0056 ††	**0.0078** †††	0.0040 †††**
	0.5	**0.0104** †††	0.0089 †††	0.0087 †††	0.0069 †††***		0.5	**0.0096** †††	0.0077 †††	0.0088 †††**	0.0070 †††**
	1.0	**0.0185** †††	0.0136 †††***	0.0151 †††*	0.0144 †††**		1.0	**0.0184** †††	0.0135 †††***	0.0151 †††*	0.0144 †††**
KNN	0	**0.0054** †	-0.0031 **	0.0026 ††	-0.0080 ††***	KNN	0	**0.0056** †	-0.0066 ***	0.0039 †††	-0.0080 ††***
	0.3	**0.0086** ††	0.0004 **	0.0041 †††*	0.0039 *		0.3	**0.0078** ††	0.0016 ***	0.0050 †††	0.0038 †*
	0.5	**0.0168** †††	0.0096 †††*	0.0092 †††*	0.0118 †††		0.5	**0.0158** †††	0.0070 †***	0.0097 †††*	0.0117 †††
	1.0	**0.0408** †††	0.0361 †††	0.0350 †††*	0.0316 †††***		1.0	**0.0406** †††	0.0299 †††**	0.0348 †††*	0.0314 †††***
SVM	0	**0.0005**	-0.0081 †††***	-0.0001 *	-0.0051 ††***	SVM	0	**0.0003**	-0.0088 †††***	-0.0001	-0.0051 ††***
	0.3	**0.0034**	0.0002	0.0012	0.0014		0.3	**0.0035**	-0.0012 *	0.0012	0.0015
	0.5	**0.0084** †††	0.0058 †††*	0.0049 †††**	0.0058 †††*		0.5	**0.0085** †††	0.0046 †††**	0.0049 †††**	0.0058 †††*
	1.0	**0.0218** †††	0.0198 †††	0.0140 †††***	0.0167 †††***		1.0	**0.0218** †††	0.0190 †††***	0.0140 †††***	0.0167 †††***
RF	0	**-0.0039** †††	-0.0042 ††	-0.0050 †††	-0.0048 ††	RF	0	**-0.0040** ††	-0.0041 ††	-0.0045 †††	-0.0044 ††
	0.3	**0.0017**	0.0004	-0.0001 **	0.0002 *		0.3	**0.0014**	0.0005	0.0003 *	0.0004
	0.5	**0.0053** †††	0.0036 †††*	0.0032 †††***	0.0035 ††*		0.5	**0.0053** †††	0.0036 ††*	0.0034 †††**	0.0037 ††*
	1.0	**0.0154** †††	0.0133 †††	0.0113 †††**	0.0117 †††**		1.0	**0.0154** †††	0.0112 †††*	0.0113 †††**	0.0117 †††**
XGBoost	0	**0.0053** ††	0.0021 *	0.0036 **	-0.0022 **	XGBoost	0	**0.0047** †	0.0017 *	0.0038 *	-0.0020 **
	0.3	**0.0072** †††	0.0063 †††	0.0057 ††**	0.0017 **		0.3	**0.0072** †††	0.0064 †††	0.0059 ††**	0.0019 **
	0.5	**0.0097** †††	0.0095 †††*	0.0071 †††***	0.0043 †***		0.5	**0.0097** †††	0.0096 †††	0.0072 †††***	0.0044 †***
	1.0	**0.0181** †††	0.0175 †††**	0.0107 †††***	0.0109 †††***		1.0	**0.0179** †††	0.0175 †††	0.0106 †††***	0.0108 †††***
			Bean						Bean		
NN	0	-0.0001	-0.0005 *	0.0000	**0.0002**	NN	0	**0.0009**	-0.0002	0.0008 †	-0.0001
	0.3	0.0004	0.0003	-0.0001 *	**0.0007** †		0.3	**0.0009**	0.0007	0.0007	0.0005
	0.5	0.0009 ††	0.0008 ††**	-0.0002 ***	**0.0010** ††		0.5	**0.0015** †	0.0013	0.0012	0.0009
	1.0	**0.0023** †††	0.0021 †††**	-0.0005 ***	0.0017 ††		1.0	**0.0030** ††	0.0027 †	0.0027 ††	0.0018
KNN	0	**0.0014** ††	0.0007 ***	0.0005 **	0.0006 ***	KNN	0	**0.0021** †	0.0007 **	0.0008 **	0.0002 **
	0.3	**0.0036** †††	0.0027 †††***	0.0017 †††***	0.0024 †††***		0.3	**0.0037** †††	0.0027 ††	0.0020 ††**	0.0020 *
	0.5	**0.0050** †††	0.0039 †††***	0.0025 †††***	0.0035 †††***		0.5	**0.0057** †††	0.0041 †††*	0.0031 †††**	0.0031 †††**
	1.0	**0.0086** †††	0.0072 †††***	0.0045 †††***	0.0065 †††***		1.0	**0.0110** †††	0.0083 †††***	0.0061 †††***	0.0061 †††***
SVM	0	0.0000	-0.0003	**0.0001**	-0.0003	SVM	0	**0.0008** †	-0.0001	0.0008	-0.0005
	0.3	**0.0015** †††	0.0013 †††**	0.0001 †***	0.0008 †**		0.3	**0.0021** ††	0.0018 †††	0.0013 †††*	0.0009
	0.5	**0.0026** †††	0.0024 †††**	0.0001 †***	0.0015 †††***		0.5	**0.0032** †††	0.0031 †††	0.0016 †††**	0.0018 †††*
	1.0	**0.0054** †††	0.0052 †††***	0.0001 ***	0.0032 †††***		1.0	**0.0074** †††	0.0067 †††**	0.0024 †††***	0.0041 †††***
RF	0	-0.0001	-0.0017 ††	**-0.0008**	-0.0013 †	RF	0	**0.0021** ††	0.0000 *	0.0021 ††	-0.0023 ***
	0.3	**0.0017** †††	0.0016 ††	-0.0004 ***	0.0011 ††		0.3	**0.0042** †††	0.0029 ††*	0.0016 †***	0.0003 ***
	0.5	**0.0039** †††	0.0037 †††	-0.0001 ***	0.0027 †††**		0.5	**0.0063** †††	0.0050 †††***	0.0017 †***	0.0020 †††**
	1.0	**0.0094** †††	0.0092 †††	0.0007 ***	0.0066 †††***		1.0	**0.0127** †††	0.0123 †††*	0.0058 †††***	0.0064 †††***
XGBoost	0	**0.0001**	-0.0011 **	-0.0007	-0.0021 ††***	XGBoost	0	0.0020	0.0009	**0.0025** ††	-0.0013 ***
	0.3	**0.0027** †††	0.0018 †††**	-0.0003 ***	0.0004 ***		0.3	**0.0046** †††	0.0041 †††**	0.0019 †***	0.0010 ***
	0.5	**0.0047** †††	0.0038 †††***	-0.0001 ***	0.0021 †††***		0.5	**0.0070** †††	0.0066 †††*	0.0026 †***	0.0025 †***
	1.0	**0.0095** †††	0.0087 †††***	0.0005 ***	0.0064 †††***		1.0	**0.0131** †††	0.0128 †††	0.0046 †††***	0.0064 †††***

"$*$", "$**$", and "$***$" mean p-value < 0.1, < 0.05, and < 0.01 respectively for one-sided pairwise t-test between EnsembleSM and the corresponding smoothed model. "†", "††", and "† † †" mean p-value < 0.1, < 0.05, and < 0.01 respectively for two-sided pairwise t-test between the corresponding smoothed model and the model without smoothing. The best-performing smoothing model is highlighted in bold.

6 Experiments: Real-World Data

6.1 Results on Real-World Structured Data

In addition to the *Synthetic* dataset, we also experimented with the real-world structured datasets *Diabetes* (Clore 2014), *Adults* (Becker and Kohavi 1996), *Survival* (Harrell 1995), and *Bean* from the UCI dataset repository. *Diabetes* contains data from the electronic health records on diabetes inpatient encounters in 130 U.S. hospitals from 1999-2008 and has a total of 101,766 instances with 47 features. The ML task is to predict whether the patient will be re-admitted to the hospital. *Adults* dataset contains 48,842 records of individuals with 14 demographic features. The task is to predict whether an individual's income exceeds \$50K per year. *Survival* dataset comprises 9,105 individual, critically ill patients across 5 U.S. medical centers from 1989-1991 and 1992-1994. The ML task is to predict whether the patient dies in the hospital, outside, or survives. *Bean* dataset contains images of 13,611 grains of 7 different registered dry beans. The task is to predict the class of beans based on 14 features extracted from the image. We experimented with the same ML techniques and hyperparameter tuning process as previously described; the size of training data (and other data partitions) was chosen so as to obtain adequate baseline model performance.[11]

Table 4 presents the overall utility scores for these two datasets across different smoothing schemes. Importantly, the general performance patterns are consistent with previous findings on the *Synthetic* dataset, demonstrating advantages of the proposed Ensemble Approach not only over the no-smoothing baseline, but also over the other smoothing approaches.

Other interesting findings include the fact that the magnitude of self-consistency and predictive performance improvement with *Diabetes* dataset is significantly larger than those with other datasets. This is likely a result of lower baseline performance on *Diabetes* dataset,[12] which provides much more potential room for both self-consistency and predictive performance improvement. Moreover, we note that the KNN technique is more susceptible to the self-consistency improvement brought by smoothing. When $\alpha = 1.0$ (i.e., the utility score purely reflects the percentage change in the self-consistency metric), the self-consistency improvement with the KNN model can be twice as large as the self-consistency improvement of other ML techniques. This observation holds across all datasets, indicating that KNN inherently (i.e., without any smoothing) might be less self-consistent than other techniques. For instance, for the *Diabetes* dataset, the self-consistency improvement (that is due to smoothing) with the KNN model is 18% in terms of SCF1. Investigating heterogeneity of ML techniques in terms of their inherent self-consistency and the effectiveness of smoothing is an interesting direction for future work.

6.2 Results on Real-World Unstructured Data with Deep Learning Models

In addition to structured data, we also experimented with two unstructured datasets: *IMDB* (Maas et al. 2011) and *CIFAR-10* (Krizhevsky and Hinton 2009). *IMDB* dataset consists of 50,000 textual movie reviews from the IMDB website. The ML task is to classify whether the sentiment of the review is positive or negative.

[11] Readers are referred to Tables B.1-3 in Appendix B for details.
[12] As shown in Table B.1 in Appendix B, the F1 score of baseline models is often 15-20% lower on *Diabetes*.

The *CIFAR-10* dataset contains 60,000 32×32 color images categorized into 10 different classes. The ML task is to classify the objects depicted in a given image according to their category.

With *IMDB* and *CIFAR-10* datasets, we examine two deep learning models: Distilled GPT2 and ResNet18, respectively. The baseline Distilled GPT2 model without smoothing achieves 0.88 weighted F1 and 0.85 Accuracy on IMDB, and the baseline ResNet18 without smoothing achieves 0.66 weighted F1 and 0.67 Accuracy on CIFAR-10. Because a much more extensive computational runtime is needed to train deep learning models, we report average results from 5 runs (with different random seeds) of each algorithm-smoothing variation.

Table 5: Utility of Different Smoothing Schemes Deep Learning Models on Real-World Unstructured Data

U_{F1}					U_{Acc}				
α	EnsembleSM	LowSM	HighSM	UniformSM	α	EnsembleSM	LowSM	HighSM	UniformSM
IMDB, Distilled GPT2					IMDB, Distilled GPT2				
0	**0.0109**	0.0083	0.0100	0.0068	0	**0.0061**	0.0037	0.0061	0.0058
0.3	0.0162	0.0157	**0.0167**	0.0148	0.3	0.0107	0.0103	0.0108	**0.0121**
0.5	**0.0216**	0.0216	0.0214	0.0201	0.5	0.0150	0.0150	0.0140	**0.0163**
1.0	**0.0403**	0.0403	0.0351	0.0334	1.0	**0.0278**	0.0278	0.0232	0.0268
CIFAR-10, ResNet18					CIFAR-10, ResNet18				
0	**0.0591**	0.0585	0.0431	0.0391	0	**0.0570**	0.0564	0.0425	0.0390
0.3	**0.0494**	0.0487	0.0428	0.0421	0.3	**0.0478**	0.0464	0.0423	0.0419
0.5	**0.0482**	0.0448	0.0426	0.0442	0.5	**0.0475**	0.0444	0.0421	0.0439
1.0	0.0472	0.0464	0.0421	**0.0493**	1.0	0.0465	0.0454	0.0417	**0.0488**

The best-performing smoothing model is highlighted in bold.

Table 5 shows that the general patterns observed with the earlier results still hold for these deep learning models on unstructured data. In all cases, Ensemble Smoothing is superior to the no-smoothing baseline; and, in most cases (and for varied α values), Ensemble Smoothing outperforms the other smoothing schemes.

Interestingly, unlike with structured data, where smoothing often improves predictive performance only by a small margin (e.g., as shown in Table 4 when $\alpha = 0$), we observe that smoothing can improve the Accuracy and F1 scores with ResNet18 by a large margin (i.e., by 5.7% and 5.9%, respectively, with Ensemble Smoothing). This is aligned with the computational evidence from prior literature regarding the potential effectiveness of "self-training" for improving predictive performance of the deep learning models for unstructured data (Zou et al. 2019, Xie et al. 2020, Du et al. 2020). This indicates that more data-hungry ML techniques may reap more substantial predictive performance benefits from smoothing techniques.

7 Results with Radius Sampling & Oversampling

Tables 6 and 7 show the overall utility scores achieved by smoothing with different sampling schemes. Notably, Smoothing with Oversampling outperforms that with Radius Sampling in terms of self-consistency by a large margin, while the two sampling schemes seem to offer similar predictive performance improvement in general. Compared with results from the standard sampling (in Table 4), we observe Oversampling often achieves similar predictive performance and self-consistency improvement for most ML models (though it

Table 6: Utility U_{F1} of Smoothing with Radius Sampling vs. Oversampling (Diabetes Dataset)

Radius Sampling

Models	α	EnsembleSM	LowSM	HighSM	UniformSM
NN	0	**0.0029**	0.0017	0.0018	0.0002 **
	0.3	**0.0087** †††	0.0044 ††***	0.0052 †††***	0.0036 ††***
	0.5	**0.0126** †††	0.0067 ††***	0.0079 †††***	0.0059 †††***
	1.0	**0.0228** †††	0.0136 †††***	0.0148 †††***	0.0116 †††***
KNN	0	0.0038 †	**0.0039** †	-0.0009 **	-0.0038 *
	0.3	0.0070 ††	**0.0071** ††	0.0007 ***	-0.0022
	0.5	0.0092 ††	**0.0093** ††	0.0020 **	-0.0012
	1.0	0.0145 ††	**0.0146** ††	0.0051 **	0.0015
SVM	0	**0.0156** ††	0.0138	0.0007 *	0.0037 *
	0.3	**0.0298** †††	0.0203 ††***	0.0106 ††**	0.0123 †††***
	0.5	**0.0421** †††	0.0247 †††***	0.0172 †††***	0.0180 †††***
	1.0	**0.0730** †††	0.0429 †††***	0.0338 †††***	0.0324 †††***
RF	0	**0.0056** †	0.0027 *	0.0056	0.0035 ††
	0.3	**0.0119** ††	0.0083 †††	0.0115 ††	0.0090 †††
	0.5	**0.0160** †††	0.0122 †††	0.0155 ††	0.0126 †††
	1.0	**0.0271** †††	0.0219 †††	0.0255 ††	0.0216 †††
XGBoost	0	**0.0025**	0.0001	0.0022	-0.0050 †**
	0.3	**0.0144** †††	0.0128 †††	0.0083 ††*	0.0080 †††**
	0.5	**0.0250** †††	0.0226 †††*	0.0151 †††**	0.0167 †††**
	1.0	**0.0515** †††	0.0472 †††*	0.0349 †††**	0.0384 †††**

Oversampling

Models	α	EnsembleSM	LowSM	HighSM	UniformSM
NN	0	0.0020 †	0.0011	**0.0023** ††	0.0005
	0.3	**0.0051** ††	0.0027 †**	0.0030 ††*	0.0020 ***
	0.5	**0.0084** †††	0.0037 ††***	0.0035 †***	0.0029 ***
	1.0	**0.0172** †††	0.0063 ††***	0.0071 ††***	0.0054 †***
KNN	0	**0.0175**	0.0155	-0.0194 *	-0.0193 *
	0.3	**0.0736** ††	0.0711 ††**	0.0332 †*	0.0340 †*
	0.5	**0.1159** †††	0.1127 †††**	0.0682 †††*	0.0695 †††*
	1.0	**0.2225** †††	0.2169 †††***	0.1559 †††*	0.1583 †††*
SVM	0	**0.0151** ††	0.0056 *	0.0001 **	-0.0024 **
	0.3	**0.0365** †††	0.0200 †††***	0.0018 ***	0.0107 ††***
	0.5	**0.0520** †††	0.0314 †††***	0.0109 ††***	0.0194 †††***
	1.0	**0.0941** †††	0.0599 †††***	0.0334 †††***	0.0413 †††***
RF	0	**0.0025**	0.0006	-0.0010	-0.0049 †††**
	0.3	**0.0177** †††	0.0149 †††**	0.0151 †††	0.0105 †††***
	0.5	**0.0298** †††	0.0258 †††	0.0265 †††	0.0207 †††***
	1.0	**0.0609** †††	0.0531 †††*	0.0548 †††	0.0462 †††**
XGBoost	0	0.0073	-0.0028 **	**0.0084**	-0.0008 **
	0.3	**0.0229** †††	0.0155 †††**	0.0169 †††***	0.0154 ††*
	0.5	**0.0356** †††	0.0284 †††**	0.0280 †††**	0.0263 †††*
	1.0	**0.0718** †††	0.0622 †††***	0.0557 †††***	0.0533 †††***

"$*$", "$**$", and "$***$" mean p-value < 0.1, < 0.05, and < 0.01 respectively for one-sided pairwise t-test between EnsembleSM and the corresponding smoothed model. "$\dagger$", "$\dagger\dagger$", and "$\dagger\ \dagger\ \dagger$" mean p-value < 0.1, < 0.05, and < 0.01 respectively for two-sided pairwise t-test between the corresponding smoothed model and the model without smoothing. The best-performing smoothing model is highlighted in bold.

significantly underperforms the standard sampling with Neural Networks in terms of self-consistency).

These observations suggest that the "realism" of unlabeled data instances may be an important factor in the effectiveness of smoothing. Using real data instances (even with a smaller coverage of the feature space) for smoothing (as is the case with Oversampling) is often better than using "artificial" data instances with stronger coverage of the feature space (as is the case with Radius Sampling).

8 Results with State-of-the-Art Semi-Supervised Learning Algorithms

As suggested by computational evidence shown in Section 6.2, Ensemble Smoothing can often achieve both significant self-consistency and prediction performance improvement at the same time. Especially, we observe Ensemble Smoothing achieve best predictive performance among all smoothing schemes. However, is this significant improvement in predictive performance comparable to those of state-of-the-art semi-supervised learning (SSL) algorithms? To answer this question, we obtain results with both baseline and state-of-the-art SSL algorithms in our experiment setup for reference.

Table 8 shows the utility score achieved by a standard baseline SSL algorithm, i.e., Pseudo-Labeling, and two popular SSL algorithms, Dash and SoftMatch.[13] These SSL algorithms are originally designed specifically for computer vision tasks; thus, this set of experiments is carried out on a computer vision dataset *CIFAR-10*. Notably, all SSL algorithms underperform Ensemble Smoothing in terms of self-consistency

[13]We adapted these algorithms from a widely-recognized SSL benchmarking library Semilearn (Wang et al. 2022). A brief introduction as well as the specification and fine-tuned parameters of these algorithms can be found in Appendix C.

Table 7: Utility U_{Acc} of Smoothing with Radius Sampling vs. Oversampling (Diabetes Dataset)

		RadiusSM				Oversampling				
Models	α	EnsembleSM	LowSM	HighSM	UniformSM	Models	α	EnsembleSM	LowSM	HighSM	UniformSM
NN	0	**0.0026** **	0.0014	0.0017	-0.0002 **	NN	0	0.0020 †	0.0010	**0.0022** ††	-0.0001 *
	0.3	**0.0077** †††	0.0037 †***	0.0041 ††***	0.0027 †***		0.3	**0.0046**	0.0024 †**	0.0028 ††*	0.0012 ***
	0.5	**0.0112** †††	0.0053 †††***	0.0064 †††***	0.0047 ††***		0.5	**0.0077**	0.0033 ††***	0.0031 ††***	0.0021 ***
	1.0	**0.0201** †††	0.0113 †††***	0.0124 †††***	0.0096 †††***		1.0	**0.0157**	0.0057 ††***	0.0069 †††***	0.0043 ***
KNN	0	**0.0004**	0.0003	0.0000	-0.0002	KNN	0	**-0.0009**	-0.0045 †*	-0.0024	-0.0102 †††***
	0.3	**0.0019** †††	0.0019 †††	0.0015 ††	0.0015 ††		0.3	**0.0269** †††	0.0255 †††	0.0266 †††	0.0208 †††**
	0.5	**0.0032** †††	0.0032 †††	0.0027 †††	0.0027 †††		0.5	**0.0491** †††	0.0456 †††	0.0486 †††	0.0415 †††**
	1.0	**0.0065** †††	0.0064 †††	0.0056 †††	0.0056 †††		1.0	**0.1050** †††	0.0957 †††	0.1041 †††	0.0931 †††*
SVM	0	**0.0019**	0.0002	-0.0003 *	-0.0005	SVM	0	**0.0020**	-0.0031 **	0.0006	-0.0008 *
	0.3	**0.0148** †††	0.0071 †††**	0.0067 †††***	0.0059 ††***		0.3	**0.0194** †††	0.0071 †††***	0.0064 ††***	0.0087 †††***
	0.5	**0.0237** †††	0.0118 †††***	0.0114 †††***	0.0101 †††***		0.5	**0.0317** †††	0.0139 †††***	0.0128 †††***	0.0151 †††***
	1.0	**0.0461** †††	0.0234 †††***	0.0232 †††***	0.0207 †††***		1.0	**0.0625** †††	0.0311 †††***	0.0298 †††***	0.0310 †††***
RF	0	**0.0009**	-0.0006 *	0.0007	0.0003	RF	0	**-0.0004**	-0.0011	-0.0013	-0.0037 †††*
	0.3	**0.0050** †††	0.0029 ***	0.0041 †	0.0036 †††*		0.3	**0.0113** †††	0.0099 †††	0.0102 †††	0.0073 †††***
	0.5	**0.0078** †††	0.0053 ††***	0.0064 ††	0.0057 †††*		0.5	**0.0201** †††	0.0172 †††	0.0180 †††	0.0146 †††***
	1.0	**0.0149** †††	0.0112 †††***	0.0124 †††	0.0111 †††*		1.0	**0.0420** †††	0.0354 †††*	0.0375 †††*	0.0328 †††**
XGBoost	0	**0.0036**	0.0022	0.0030	-0.0027 *	XGBoost	0	0.0030	-0.0031 †**	**0.0033**	-0.0014 **
	0.3	**0.0110** †††	0.0096 †††	0.0069 †††**	0.0062 ††		0.3	**0.0133** †††	0.0096 †††**	0.0097 †††**	0.0096 †††*
	0.5	**0.0179** †††	0.0164 †††**	0.0121 †††**	0.0122 †††*		0.5	**0.0230** †††	0.0184 †††***	0.0173 †††***	0.0169 †††***
	1.0	**0.0357** †††	0.0332 †††*	0.0260 †††**	0.0270 †††**		1.0	**0.0474** †††	0.0418 †††***	0.0361 †††***	0.0352 †††***

"$*$", "$**$", and "$***$" mean p-value < 0.1, < 0.05, and < 0.01 respectively for one-sided pairwise t-test between EnsembleSM and the corresponding smoothed model. "$\dagger$", "$\dagger\dagger$", and "$\dagger\ \dagger\ \dagger$" mean p-value < 0.1, < 0.05, and < 0.01 respectively for two-sided pairwise t-test between the corresponding smoothed model and the model without smoothing. The best-performing smoothing model is highlighted in bold.

(i.e., when $\alpha = 1.0$) by a large margin. This is expected as none of them are designed for self-consistency improvement. In terms of pure predictive performance improvement (i.e., when $\alpha = 0$), Ensemble Smoothing outperforms Pseudo-Labeling, while it underperforms state-of-the-art SSL algorithms, Dash and SoftMatch, by a 1-2% margin. These results suggest that our proposed Ensemble Smoothing algorithm can achieve predictive performance improvement that is close to that of the state-of-the-art SSL algorithms while offering significant self-consistency improvements at the same time.

Table 8: Utility Comparison of Referenced SSL Algorithms (ResNet18 on CIFAR-10)

U_{F1}					U_{Acc}				
α	EnsembleSM	Pseudo-Labeling	Dash	SoftMatch	α	EnsembleSM	Pseudo-Labeling	Dash	SoftMatch
0	0.0591	0.0422	0.0683	**0.0796**	0	0.0570	0.0414	0.0678	**0.0791**
0.3	0.0494	0.0288	0.0472	**0.0517**	0.3	0.0478	0.0284	0.0467	**0.0515**
0.5	**0.0482**	0.0196	0.0323	0.0368	0.5	**0.0475**	0.0190	0.0321	0.0364
1.0	**0.0472**	0.0128	0.0079	0.0065	1.0	**0.0465**	0.0124	0.0068	0.0057

9 Conclusions

In this study, we investigate the role of prediction confidence in shaping the effectiveness of smoothing for improving machine learning systems, especially in terms of their self-consistency. First, we demonstrate the *differential effects* of confidence-based smoothing on predictive performance and self-consistency, providing intuition and empirical evidence based on computational experiments on synthetic and real data. Specifically,

we show that smoothing from high-confidence regions of the feature space can lead to gains in predictive accuracy with limited impact on self-consistency, while smoothing from low-confidence regions substantially improves self-consistency but often at the expense of predictive accuracy. This nuanced understanding contributes to both theory and practice by clarifying the trade-offs inherent in different smoothing strategies.

Based on these insights, we propose an ensemble smoothing approach to improving self-consistency and predictive performance of ML techniques. We demonstrate the advantages of the proposed approach (over the simpler smoothing schemes) using extensive computational experiments. The designed approach is modular and easy to implement, as it is a wrapper-like meta-algorithmic technique that can be easily incorporated as an add-on to existing applications. Although we specifically focus on the classification task in this study, the proposed approach can be extended to regression tasks in a similar manner. We leave this extension as a part of future work. In addition, computational results of this study provide some directions for theorizing the relations between prediction confidence, self-consistency, and predictive performance. For example, we empirically show that smoothing from a high-confidence feature space can potentially improve predictive performance, while smoothing from a low-confidence feature space can provide substantial self-consistency benefits. Understanding and formalizing the theoretical underpinnings of this relationship could lead to further developments of machine learning theory.

References

Gediminas Adomavicius and Jingjing Zhang. Stability of recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 30(4):1–31, 2012.

Gediminas Adomavicius and Jingjing Zhang. Improving stability of recommender systems: a meta-algorithmic approach. *IEEE Transactions on Knowledge and Data Engineering*, 27(6):1573–1587, 2014.

Gediminas Adomavicius and Jingjing Zhang. Classification, ranking, and top-k stability of recommendation algorithms. *INFORMS Journal on Computing*, 28(1):129–147, 2016.

Shivani Agarwal and Partha Niyogi. Stability and generalization of bipartite ranking algorithms. In *Learning Theory: 18th Annual Conference on Learning Theory, COLT 2005, Bertinoro, Italy, June 27-30, 2005. Proceedings 18*, pages 32–47. Springer, 2005.

Shivani Agarwal and Partha Niyogi. Generalization bounds for ranking algorithms via algorithmic stability. *Journal of Machine Learning Research*, 10(2), 2009.

Gagan Bansal, Besmira Nushi, Ece Kamar, Walter S Lasecki, Daniel S Weld, and Eric Horvitz. Beyond accuracy: The role of mental models in human-ai team performance. In *Proceedings of the AAAI conference on human computation and crowdsourcing*, volume 7, pages 2–11, 2019a.

Gagan Bansal, Besmira Nushi, Ece Kamar, Daniel S Weld, Walter S Lasecki, and Eric Horvitz. Updates in human-ai teams: Understanding and addressing the performance/compatibility tradeoff. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 2429–2437, 2019b.

Barry Becker and Ronny Kohavi. Adult. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of machine learning research*, 2(Mar): 499–526, 2002.

Houssem Ben Braiek and Foutse Khomh. Machine learning robustness: A primer. In *Trustworthy AI in Medical Imaging*, pages 37–71. Elsevier, Amsterdam, The Netherlands, 2025.

Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

Hao Chen, Ran Tao, Yue Fan, Yidong Wang, Jindong Wang, Bernt Schiele, Xing Xie, Bhiksha Raj, and Marios Savvides. Softmatch: Addressing the quantity-quality trade-off in semi-supervised learning. *arXiv preprint arXiv:2301.10921*, 2023.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Clore. Diabetes 130-US hospitals for years 1999-2008. UCI Machine Learning Repository, 2014. DOI: https://doi.org/10.24432/C5230J.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1): 21–27, 1967.

Nathan Drenkow, Numair Sani, Ilya Shpitser, and Mathias Unberath. A systematic review of robustness in deep learning for computer vision: Mind the gap? *arXiv preprint arXiv:2112.00639*, 2021.

Jingfei Du, Edouard Grave, Beliz Gunel, Vishrav Chaudhary, Onur Celebi, Michael Auli, Ves Stoyanov, and Alexis Conneau. Self-training improves pre-training for natural language understanding. *arXiv preprint arXiv:2010.02194*, 2020.

Andre Elisseeff, Theodoros Evgeniou, Massimiliano Pontil, and Leslie Pack Kaelbing. Stability of randomized learning algorithms. *Journal of Machine Learning Research*, 6(1), 2005.

Frank Harrell. SUPPORT2 [dataset]. UCI Machine Learning Repository, 1995. `https://doi.org/10.3886/ICPSR02957.v2`.

Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine learning*, 110(3):457–506, 2021.

Jacob Kahn, Ann Lee, and Awni Hannun. Self-training for end-to-end speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7084–7088. IEEE, 2020.

Sherrie YX Komiak and Izak Benbasat. The effects of personalization and familiarity on trust and adoption of recommendation agents. *MIS quarterly*, pages 941–960, 2006.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Dong-Hyun Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 896. Atlanta, 2013.

Hong Liu, Jianmin Wang, and Mingsheng Long. Cycle self-training for domain adaptation. *Advances in Neural Information Processing Systems*, 34:22968–22981, 2021.

Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

Sayan Mukherjee, Partha Niyogi, Tomaso Poggio, and Ryan Rifkin. Learning theory: stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization. *Advances in Computational Mathematics*, 25:161–193, 2006.

John O'Donovan and Barry Smyth. Trust in recommender systems. In *Proceedings of the 10th international conference on Intelligent user interfaces*, pages 167–174, 2005.

Lars Schmarje, Monty Santarossa, Simon-Martin Schröder, and Reinhard Koch. A survey on semi-, self-and unsupervised learning for image classification. *IEEE Access*, 9:82146–82168, 2021.

Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in neural information processing systems*, 33:596–608, 2020.

Hristos Tyralis and Georgia Papacharalampous. A review of predictive uncertainty estimation with machine learning. *Artificial Intelligence Review*, 57(4):94, 2024.

Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine learning*, 109(2):373–440, 2020.

Yidong Wang, Hao Chen, Yue Fan, Wang Sun, Ran Tao, Wenxin Hou, Renjie Wang, Linyi Yang, Zhi Zhou, Lan-Zhe Guo, et al. Usb: A unified semi-supervised learning benchmark for classification. *Advances in Neural Information Processing Systems*, 35:3938–3961, 2022.

Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10687–10698, 2020.

Qiantong Xu, Alexei Baevski, Tatiana Likhomanenko, Paden Tomasello, Alexis Conneau, Ronan Collobert, Gabriel Synnaeve, and Michael Auli. Self-training and pre-training are complementary for speech recognition. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3030–3034. IEEE, 2021a.

Yi Xu, Lei Shang, Jinxing Ye, Qi Qian, Yu-Feng Li, Baigui Sun, Hao Li, and Rong Jin. Dash: Semi-supervised learning with dynamic thresholding. In *International conference on machine learning*, pages 11525–11536. PMLR, 2021b.

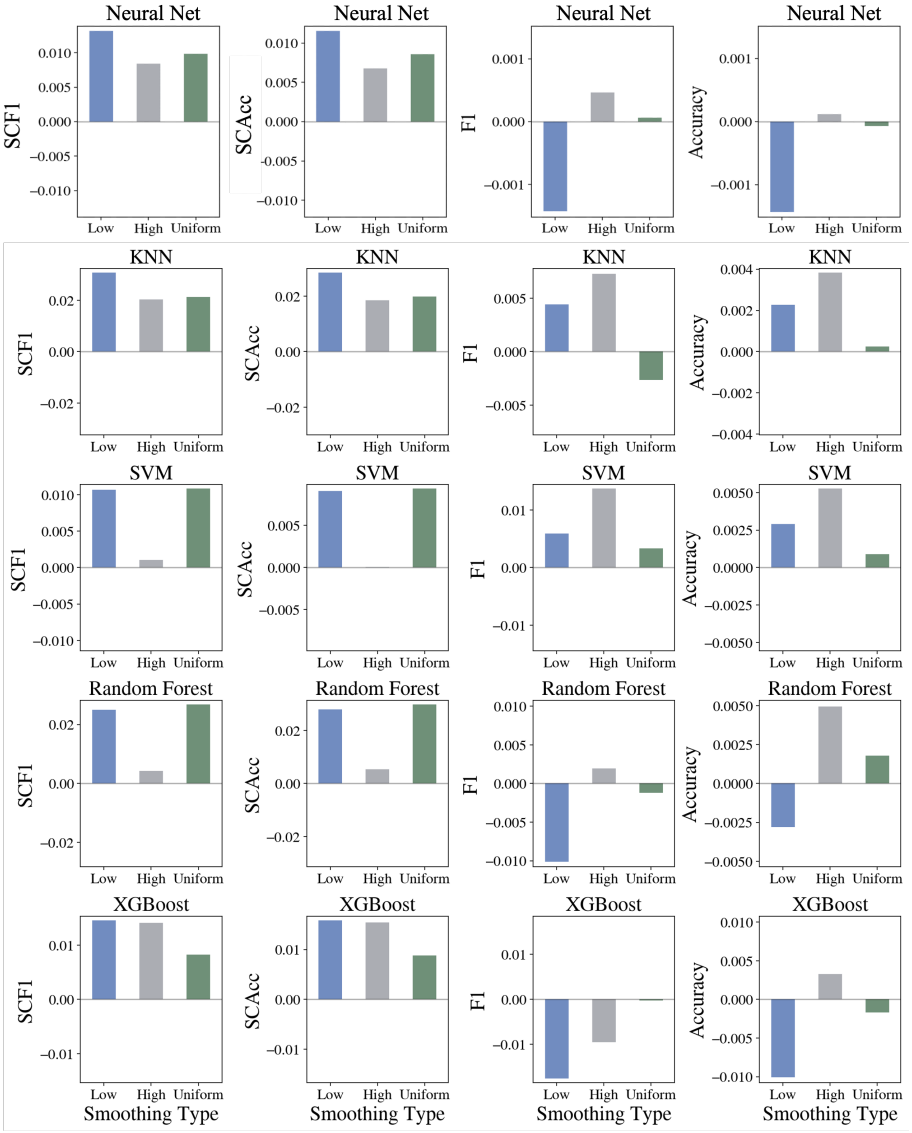
Lihe Yang, Wei Zhuo, Lei Qi, Yinghuan Shi, and Yang Gao. St++: Make self-training work better for semi-supervised semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4268–4277, 2022.

Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin Dogus Cubuk, and Quoc Le. Rethinking pre-training and self-training. *Advances in neural information processing systems*, 33:3833–3845, 2020.

Yang Zou, Zhiding Yu, Xiaofeng Liu, BVK Kumar, and Jinsong Wang. Confidence regularized self-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5982–5991, 2019.

Appendix A Heterogeneous Effect of Low- vs. High-Confidence Smoothing

In Figure B.1, we show the percentage improvement in self-consistency and predictive performance metrics for different smoothing schemes across multiple ML algorithms. This provides further evidence of the heterogeneous effect of Low- vs. High-Confidence Smoothing. Notably, Low-Confidence Smoothing can improve self-consistency (i.e., as measured by SCF1 or SCAcc), but it may harm predictive performance. On the other hand, while High-Cnfidence Smoothing can often improve predictive performance (i.e., as measured by F1 or Accuracy), it often cannot offer significant self-consistency improvements comparable to those of Low-Confidence Smoothing.



"Low": Low-Confidence Smoothing, "High": High-Confidence Smoothing

"Uniform": Uniform Smoothing

Figure B.1 Percentage Performance Improvement on Synthetic Dataset under Different Smoothing Schemes

Appendix B Experiment Setup

We provide further details on the setup of computational experiments. We fine-tune important hyper-parameters that define Low/High-Confidence Smoothing (i.e., SC_L, and SC_H) and how they are applied to the testing data instances (i.e., $TestConf$). For experiments with structured data and traditional ML techniques, we fine-tune a list of 12 values - $[0.0250, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$ for $TestConf$. We fine-tune a list of 5 values, $[0.1, 0.25, 0.5, 0.75, 0.9]$, for SC_L and SC_H. In total, there are 300 combinations of these three parameters. For experiments on unstructured data using deep learning models, we finetune a list of 3 values - $[0.1, 0.5, 0.9]$ for these three hyper-parameters (i.e., *TestConf*, SC_L, and SC_H), resulting in 27 combinations in the experiments. We report the average results from 5 runs for experiments on unstructured data and from 30 runs for experiments on structured data.

In Table B.1 and Table B.2, we present the F1 and Accuracy performance of the baseline machine learning models (i.e., without smoothing) after fine-tuning, respectively. In Table B.3, we show the number of data instances used in each data partition for different datasets.

Table B.1 Baseline Machine Learning Models: Weighted F1 Performance

	NN	**KNN**	**SVM**	**RF**	**XGBoost**
Synthetic	0.8197	0.8201	0.8074	0.8143	0.8044
Diabetes	0.5840	0.5469	0.5835	0.6459	0.6352
Adult	0.8524	0.8533	0.8431	0.8868	0.8876
Survival	0.7598	0.6953	0.7564	0.8338	0.8825
Bean	0.9139	0.8969	0.9162	0.9006	0.9031

Table B.2 Baseline Machine Learning Models: Accuracy Performance

	NN	**KNN**	**SVM**	**RF**	**XGBoost**
Synthetic	0.7102	0.7179	0.7223	0.7150	0.7263
Diabetes	0.5630	0.6092	0.5443	0.5622	0.6040
Adult	0.8155	0.8402	0.8185	0.8202	0.8428
Survival	0.7609	0.8302	0.7001	0.7579	0.8767
Bean	0.9121	0.8997	0.8984	0.9029	0.9049

Table B.3 Number of Data Points in Different Data Partitions by Datasets

Dataset Name	D^{train}	D^{smooth}	$D^{validation}$	$D^{holdout}$	D^{test}
Synthetic	150	100	1000	100	1000
Diabetes, *Adult*, *Survival*, *Bean*	500	200	1000	200	1000
IMDB	2000	1000	1000	1000	3000
CIFAR-10	4000	2000	1000	1000	3000

Appendix C Semi-Supervised Learning Algorithm Specification

In our experiments with semi-supervised learning (SSL) algorithms, we used two state-of-the-art approaches, Dash (Xu et al. 2021b) and SoftMatch (Chen et al. 2023). Both of them are based on the generic Pseudo-Labeling (Lee et al. 2013) approach, which we also used as a baseline SSL algorithm in our experiments. These SSL approaches are characterized by

1. Pseudo-label assignment during model training process (i.e., pseudo-labels are recalculated and re-assigned at every batch gradient descent), as opposed to the pseudo-label assignment after the model training process in self-training/smoothing;
2. Explicit parameterization on the weight of pseudo-labeled data instances in the loss function, as opposed to adjusting the number of pseudo-labeled data instances as implicit weight control in self-training/smoothing.

With this explicit parameterization, methods based on the generic Pseudo-Labeling algorithm (Lee et al. 2013) can control the amount of included pseudo-labeled data and also their approximated prediction quality (e.g., pseudo-labeled data with larger loss is downweighted in the subsequent training process).

We adopted three aforementioned SSL approaches (i.e., Pseudo-Labeling, Dash, and SoftMatch) from the widely-recognized benchmarking library Semilearn (Wang et al. 2022). We make the necessary adaptations for a meaningful comparison with the smoothing techniques. With SSL approaches, we use the same number of pseudo-labeled data instances (as with our smoothing approaches) for proper and consistent comparison.

We fine-tune the key hyper-parameters of the SSL algorithms. For Dash, we extensively fine-tune parameter γ that controls the decay speed of the confidence threshold in batch gradient descents; following the original paper, we fine-tune γ on 15 equally spaced values between 1.0 and 1.3. With SoftMatch, we fine-tune the weight m (as denoted in the original paper) that controls the EMA (Exponential Moving Average) update of the mean and variance of the Gaussian distribution that approximates the prediction confidence distribution. We fine-tune m on 20 equally spaced values between 0.8 and 1.0.

Appendix D Additional Simulation Experiments with k-NN

We carry out a systematic set of simulation experiments to demonstrate the differential effects of Low- vs. High-Confidence Smoothing on the predictive performance and self-consistency of k-NN models.

Simulating Data-Generation Process. Consider a binary classification problem with a single predictive feature $X \in [0, 1]$ and target $Y \in \{0, 1\}$. For any given data instance (x, y), we assume the following data-generation process: $\Pr(y = 1|x) = x$. This represents a general data-generation process, where X is an effective predictor of Y such that greater value of X corresponds to a higher probability of the data instance taking the positive class.

We simulate this data-generation process by uniformly sampling $N = 10,000$ points on interval $[0, 1]$, which serve as training instances. We denote these values as $X^{(t)}$. We realize the ground truth label for each training instance by drawing from a Bernoulli trial with probability $X^{(t)}$ being positive. These $10,000$ data points with realized ground truth labels form the training dataset D^{train}.

Measuring Accuracy. Given a test instance X_0, we find the k nearest neighbors in D^{train} and determine

the predicted label based the majority vote. We measure the prediction accuracy by comparing the predicted label against X_0's realized ground truth label[14] ($Acc = 1$ if the predicted label and the realized ground truth label are same; otherwise, $Acc = 0$).

Measuring Self-Consistency. To measure the classifier's self-consistency on a given data instance X_0, we simulate a holdout instance $X^{(h)}$ as having the same feature value as the k-th neighbor of the given instance (so that it displaces that farthest neighbor). We determine the label of $X^{(h)}$ by finding the actual k neighbors of $X^{(h)}$. We then update the label of X_0 by incorporating $X^{(h)}$ as its k-th neighbor. We measure self-consistency by comparing the original prediction and the updated prediction ($SCAcc = 1$ if the prediction does not change and 0 otherwise).

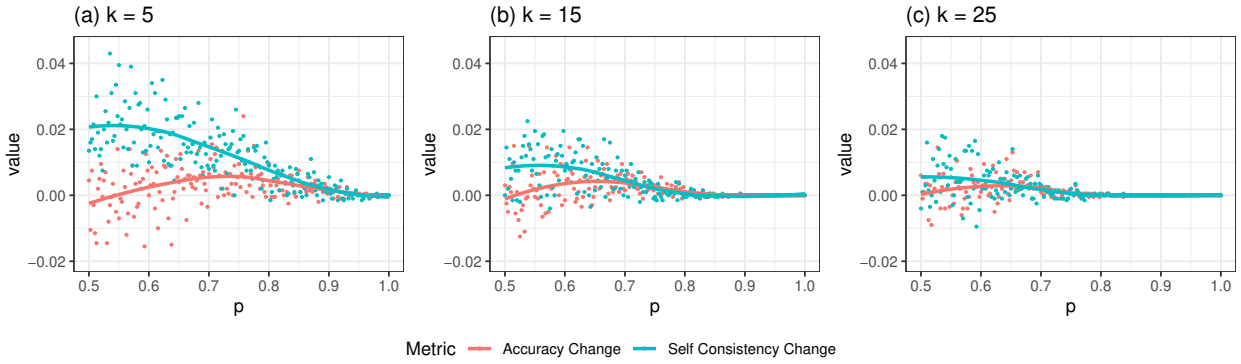


Figure D.1: Simulation: Change in Accuracy and Self-Consistency with Varying Values of k

Smoothing. We simulate the smoothing instance $X^{(s)}$ as having the same feature value as the k-th neighbor of the given test instance X_0 (so that it displaces that farthest neighbor). We determine the label of $X^{(s)}$ by finding the actual k neighbors of $X^{(s)}$. We update the prediction for the test data instance X_0 by incorporating $X^{(s)}$ as its k-th neighbor. We repeat the above process of measuring accuracy and self-consistency after we perform smoothing. Finally, we obtain the change in accuracy and self-consistency before vs. after smoothing for this given instance X_0.

We obtain 200 evenly spaced values of X_0 between 0.5 and 1 to form the test dataset D^{test}. We evaluate the change in accuracy and self-consistency for each test instance[15] in this D^{test}. We show the evaluation results in Figure D.1. In each plot of Figure D.1, every scattered point represents the evaluation result on one test instance out of the total 200. We also show the banded LOESS line fitted on these points to more evidently manifest the pattern.

Notably, Low-Confidence Smoothing offers much stronger self-consistency benefit than High-Confidence Smoothing. Furthermore, Low-Confidence Smoothing provides suboptimal accuracy performance when the prediction confidence level is close to 0.5. In fact, we also observe a similar pattern (i.e., Low-Confidence Smoothing potentially hurts accuracy of the classifier) in our experiments on synthetic and real-world datasets, as shown in Tables 3 and 4 of the main manuscript.

[14] We realize the ground truth for the test instance in the same way to that for training instances. Specifically, we draw the ground truth label from a Bernoulli trial with probability X_0 being positive.

[15] For each test instance, we repeat the aforementioned smoothing process and evaluation procedures for 2000 times with different random seeds and report the average results.