Error Detection for Multimodal Classification

Thomas Bonnier
Centrale Lille Alumni, France
thomas.bonnier@centraliens-lille.org

Abstract

Machine learning models have proven to be useful in various key applications such as autonomous driving or diagnosis prediction. When a model is implemented under real-world conditions, it is thus essential to detect potential errors with a trustworthy approach. This monitoring practice will render decision-making safer by avoiding catastrophic failures. In this paper, the focus is on multimodal classification. We introduce a method that addresses error detection based on unlabeled data. It leverages fused representations and computes the probability that a model will fail based on detected fault patterns in validation data. To improve transparency, we employ a sampling-based approximation of Shapley values in multimodal settings in order to explain why a prediction is assessed as erroneous in terms of feature values. Further, as explanation methods can sometimes disagree, we suggest evaluating the consistency of explanations produced by different value functions and algorithms. To show the relevance of our method, we measure it against a selection of 9 baselines from various domains on tabular-text and text-image datasets, and 2 multimodal fusion strategies for the classification models. Lastly, we show the usefulness of our explanation algorithm on misclassified samples.

1 Introduction

Even though pretrained language models such as BERT can achieve state-of-the-art performance in various NLP tasks such as classification (Devlin et al., 2019), they still have significant limitations (Gawlikowski et al., 2023): they do not always provide reliable uncertainty estimates, they are sensitive to distribution shifts and adversarial attacks, and their decisions are not fully transparent. In that context, it is key to employ monitoring tools that will render decision-making safer in applications where the costs of AI errors can be significant. Error detection attempts to identify mispredicted test inputs (Chen et al., 2021). Therefore, when a model is implemented for critical applications such as diagnosis prediction, autonomous driving or financial investment, it is essential to detect and avoid erroneous predictions that could have severe consequences. When potential failure is identified, the final decision could be, for instance, overridden by subject matter experts. In that case, it can be valuable to explain why a prediction is assessed as erroneous, in terms of feature values.

Here the focus is on multimodal classification tasks. Even though our method could be applied to various types of modalities, we illustrate its relevance in a bimodal context. Firstly, we concentrate on tabular datasets with text fields in English. These datasets consist of categorical and numerical features (i.e. the tabular modality) and fields with free-form text (i.e. the text modality) (Shi et al., 2021). Categorical variables have discrete values (e.g. ordinal, binary or variable with finite number of categories) whereas numerical or quantitative variables have continuous scalar values. Secondly, we consider applications with text and image modalities. Various critical applications rely on such datasets. In the medical field, clinical notes and MRI data could be employed for diagnosis prediction. In financial investment, models could make decisions based on time series (e.g. asset price) and text news for sentiment analysis.

Our objective is to evaluate what method may perform best with regard to error detection for multimodal classifiers. Our approach is illustrated in Figure 1. We highlight the following contributions:

- We introduce *Error Detection with Informative Partition* (EDIP), a method that aims to detect misclassified inputs based on fused representations extracted from a multimodal classifier. It assesses the probability of misprediction for a given test input by computing the error rate of

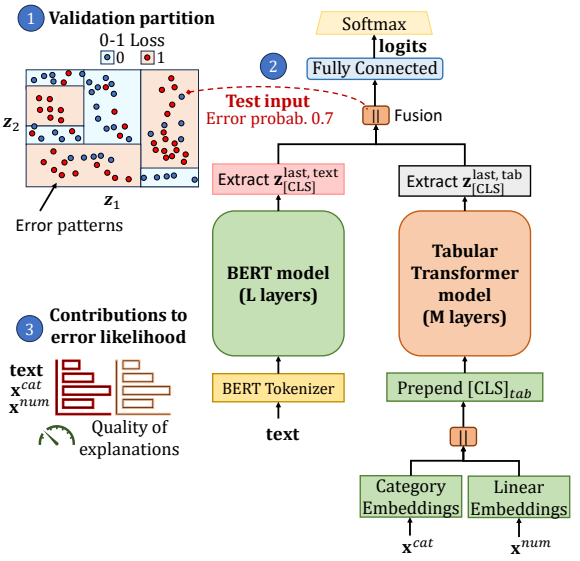

Figure 1: **Illustration of our method**: (1) Error patterns are learned on labeled validation data by leveraging fused representations extracted from a multimodal classifier whose LateFuse architecture is detailed in appendix D; (2) The probability of error for a test input is assessed based on the selected subset of partition; (3) The error explanation module is based on a sampling-based approximation of Shapley values, with evaluation of the consistency of explanations produced by different value functions and algorithms.

(labeled) validation instances involved in similar error patterns. Our method is not specific to any type of modality.

- We show that EDIP can be used as value function in a sampling-based algorithm that approximates Shapley values in multimodal settings. This turns out to be useful in explaining why a prediction is assessed as erroneous in terms of feature values. We evaluate the quality of an explanation by verifying its consistency across various value functions and algorithms.

- We assess our method by comparing it to 9 baselines from different domains on tabular-text and text-image datasets, and 2 multimodal fusion schemes for the classification models. All the methods are external approaches that can be applied to pretrained models without modification.

2 Prior Work

In this section, we summarize the prior work related to error detection. We also describe methods from closely connected domains: out-of-distribution (OOD) detection and uncertainty estimation.

Multimodal fusion. A multimodal model exploits heterogeneous and connected modalities like image and text as inputs. This approach aims to learn representations of cross-modal interactions by fusing information across various modalities (Liang et al., 2024; Xu et al., 2023). With the early fusion strategy, cross-modal interactions occur at an early stage. For a Transformer with early concatenation of two modalities, full pairwise attention will be computed at all layers. In contrast, late fusion of final representations makes cross-modal interactions happen at a later step.

Error detection. To detect model failure during inference, Corbière et al. (2019) propose a method which estimates the true class probability in image classification tasks. Self-training ensembles can be leveraged for error detection and unsupervised accuracy estimation (Chen et al., 2021). Concerning explanation methods, Shapley values (Shapley, 1953), based on cooperative game theory, is a method, when applied to machine learning, which computes the contribution of features to a model's prediction. Parcalabescu and Frank (2023) introduce MM-SHAP, a multimodality score based on Shapley values, which helps detect unimodal collapse. However, Krishna et al. (2024) point out that the outputs of different explanation techniques can disagree with each other, and suggest various metrics to measure disagreement between top-k features: intersection or rank.

Confidence scores and uncertainty. The maximum softmax probability turns out to be a useful baseline to estimate confidence (Hendrycks and Gimpel, 2017). However, as models such as neural networks can be miscalibrated, techniques such as temperature scaling are suggested to better calibrate the class probability estimates (Guo et al., 2017a). Liu et al. (2020) show the relevance of the energy score in OOD detection tasks as it is aligned with the probability density of the input. To quantify predictive uncertainty, methods such as conformal prediction can produce prediction sets based on an expected coverage level (Vovk et al., 2005; Papadopoulos et al., 2002). In particular, Tibshirani et al. (2019) propose a weighted version of conformal prediction under covariate shift. To estimate predictive uncertainty, Lakshminarayanan et al. (2017) employ deep ensembles with random parameter initialization for each neural network, along with random shuffling of the data points. The predictive entropy can be computed after averaging

the predicted probabilities from each network. To avoid the computational cost of Bayesian models, Gal and Ghahramani (2016) introduce a Bayesian approximation for deep neural networks. When evaluating the predictive uncertainty for a test input, the Monte Carlo dropout corresponds to performing various forward passes with dropout. To evaluate the trustworthiness of predictive uncertainty, Ovadia et al. (2019) present a benchmark of different methods under dataset shift (e.g. deep ensembles). To explain uncertainty estimates, Antoran et al. (2021) propose CLUE, a method based on counterfactuals, which identifies which features are responsible for uncertainty in probabilistic models. Lastly, Watson et al. (2023) explain predictive uncertainty by adapting the computation of Shapley values with the conditional entropy as value function.

OOD detection. Certain methods can be used to detect OOD samples. Dataset shifts appear when the respective source (training) and test joint distributions p and q are different: $p(\mathbf{x}, y) \neq q(\mathbf{x}, y)$ for covariates $\mathbf{x}$ and class variable y (Moreno-Torres et al., 2012). In that context, the domain classifier (Rabanser et al., 2019) is trained to discriminate between data from source (class 0) and target (class 1) domains. A dataset shift is detected when this model can easily identify from which domain the samples originate. Distance-based methods, such as non-parametric deep nearest neighbors (Sun et al., 2022), can leverage feature embeddings from a model in order to perform OOD detection.

3 Method

We have a C-class classification problem, where each input $\mathbf{x} \in \mathbb{X}$ contains multimodal features. The true class is $y \in \mathbb{Y} = \{0, 1, \ldots, C-1\}$. We consider a source dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, which includes n points sampled i.i.d. from distribution p. Further, $\mathcal{D}$ is randomly partitioned into a training dataset $\mathcal{D}_{train}$ and a validation dataset $\mathcal{D}_{val}$. We consider a class of hypotheses $\mathcal{H}$ mapping $\mathbb{X}$ to Δ^{C-1}, where Δ^{C-1} is the probability simplex over C classes. Given a classifier $\hat{\pi} \in \mathcal{H}$ fitted on $\mathcal{D}_{train}$, the predicted label is $\hat{y} = \arg\max_{j \in \mathbb{Y}} \hat{\pi}_j(\mathbf{x})$, $\forall \mathbf{x} \in \mathbb{X}$. Further, the 0-1 loss is defined as $\mathcal{L}(\hat{\pi}(\mathbf{x}), y) = \mathbf{1}_{y \neq \hat{y}}$, where $\mathbf{1}_{\text{condition}}$ is 1 if the condition is true, 0 otherwise. We assume we can extract feature embeddings $\mathbf{z}$ from the model $\hat{\pi}$: we have $\mathbf{z} = \phi(\mathbf{x})$, where the multimodal feature encoder $\phi : \mathbb{X} \to \mathbb{R}^d$ includes a fusion scheme (e.g. late fusion), and d is the embedding dimension. For example, $\mathbf{z}$ could be the concatenation of the [CLS] tokens' final hidden states from the two modalities (see Figure 1), where [CLS] corresponds to the classification token defined by Devlin et al. (2019). Given $\hat{\pi}$ and unlabeled test dataset $\mathcal{T}$, our objective is to identify mispredicted inputs.

3.1 EDIP

EDIP (Error Detection with Informative Partition) estimates the likelihood that $\hat{\pi}$ will fail based on detected error patterns. To achieve this, we first construct the label by computing the 0-1 loss for each data point of $\mathcal{D}_{val}$, indexed by $\mathcal{I}_{val}$. EDIP, defined as $\hat{f} : \mathbb{R}^d \to \Delta^1$, learns to detect error patterns: $\hat{f} = \mathcal{C}(\{(\mathbf{z}_i, \mathcal{L}(\hat{\pi}(\mathbf{x}_i), y_i)) : i \in \mathcal{I}_{val}\})$, where $\mathcal{C}$ denotes any classification algorithm leveraging partitions of the feature space. It takes in data indexed by $\mathcal{I}_{val}$ in order to output a classifier fitted on that data, where $\mathbf{z}_i = \phi(\mathbf{x}_i)$. The partition should be informative in terms of error rate, in the sense that each subset conditioned on $\mathbf{z}$ should bring more information than considering the whole validation dataset. In other words, the objective is to construct a partition that provides large information gains $H(\mathcal{L}(\hat{\pi}(\mathbf{X}), Y)) - H(\mathcal{L}(\hat{\pi}(\mathbf{X}), Y)|\mathbf{Z})$, where H denotes the entropy (Shannon, 1948) and $\mathbf{Z} = \phi(\mathbf{X})$. $\mathcal{C}$ could be, for example, a classification tree (Breiman et al., 1984) or a random forest (Breiman, 2001). The choice of this category of algorithm is justified by their flexibility, performance, and ease of interpretation of the following inference formulas.

Thus, for a new test input $\mathbf{x}'$, we address error detection by computing $\hat{f}_1(\mathbf{z}')$, where $\mathbf{z}' = \phi(\mathbf{x}')$. $\hat{f}_1(\mathbf{z}')$ estimates the probability that the 0-1 loss equals 1 given $\mathbf{z}'$. If $\hat{f}$ is a classification tree, we have:

$$\hat{f}_1(\mathbf{z}') = \sum_{i:\mathbf{z}_i \in \lambda(\mathbf{z}')} \frac{\mathcal{L}(\hat{\pi}(\mathbf{x}_i), y_i)}{|\lambda(\mathbf{z}')|}$$

$\lambda(\mathbf{z}')$ is the leaf node where $\mathbf{z}'$ falls into. $|.|$ denotes the cardinality of a set. Therefore, $|\lambda(\mathbf{z}')|$ is the number of validation samples $\mathbf{z}_i = \phi(\mathbf{x}_i)$ that are contained in leaf node $\lambda(\mathbf{z}')$. EDIP thus estimates the probability of misclassification for a given test input by computing the error rate of validation samples involved in similar error patterns. The latter are defined by the decision path that leads to $\lambda(\mathbf{z}')$.

If $\hat{f}$ is a random forest with T trees combined by

bootstrap aggregation, we have:

$$\hat{f}_1(\mathbf{z}') = \frac{1}{T}\sum_{t=1}^{T}\sum_{i:\mathbf{z}_i\in\lambda_t(\mathbf{z}')}\frac{b_t(\mathbf{z}_i)\mathcal{L}(\hat{\pi}(\mathbf{x}_i), y_i)}{|\lambda_t(\mathbf{z}')|}$$

$b_t(\mathbf{z}_i)$ is the number of times that the validation instance $\mathbf{z}_i$ has been chosen by bootstrapping in the construction of tree t. $\lambda_t(\mathbf{z}')$ denotes the leaf node from tree t, which contains $\mathbf{z}'$. $|\lambda_t(\mathbf{z}')|$ is the number of bootstrap (validation) samples that are contained in leaf node $\lambda_t(\mathbf{z}')$ from tree t.

Algorithm 1 Explanation algorithm for one feature

Input: input $\mathbf{x}$ from test dataset $\mathcal{T}$, feature index j, index set of tabular features $\mathcal{I}^{tab}$, index set of text features $\mathcal{I}^{text}$, validation dataset $\mathcal{D}_{val}$, EDIP model $\hat{f}$, feature encoder ϕ component of model $\hat{\pi}$, number of iterations M

Output: Shapley value $\Phi_j(\mathbf{x})$ for given feature (contribution to predicted probability of error)

for $m = 1$ to M **do**
  Sample $\mathbf{x}^* \sim \mathcal{D}_{val}$
  Select random subset of tabular feature indices $\mathcal{R}^{tab} \subset \mathcal{I}^{tab}\backslash\{j\}$
  Select random subset of text feature indices $\mathcal{R}^{text} \subset \mathcal{I}^{text}\backslash\{j\}$
  Initialize $\mathbf{x}_{+j} \leftarrow \mathbf{x}$ ▷ here, the subscript is related to features
  Replace all tabular values in $\mathbf{x}_{+j}$ with index in $\mathcal{R}^{tab}$ by corresponding values from $\mathbf{x}^*$
  Replace all text values in $\mathbf{x}_{+j}$ with index in $\mathcal{R}^{text}$ by [MASK] token when these token values are not in $\mathbf{x}^*$
  Initialize $\mathbf{x}_{-j} \leftarrow \mathbf{x}_{+j}$
  if $j \in \mathcal{I}^{tab}$ **then**
    Replace the tabular value in $\mathbf{x}_{-j}$ with index j by the corresponding value from $\mathbf{x}^*$
  else
    Replace the text value in $\mathbf{x}_{-j}$ with index j by the [MASK] token when this token value is not in $\mathbf{x}^*$
  end if
  $\mathbf{z}_{+j} \leftarrow \phi(\mathbf{x}_{+j})$ and $\mathbf{z}_{-j} \leftarrow \phi(\mathbf{x}_{-j})$
  $\Phi_j^m(\mathbf{x}) \leftarrow \hat{f}_1(\mathbf{z}_{+j}) - \hat{f}_1(\mathbf{z}_{-j})$ ▷ compute marginal contribution
end for
$\Phi_j(\mathbf{x}) \leftarrow \frac{1}{M}\sum_{m=1}^{M}\Phi_j^m(\mathbf{x})$ ▷ approximated Shapley value
return $\Phi_j(\mathbf{x})$

3.2 Explanation method

Explanation algorithm. In this subsection, we focus on tabular-text data. However, the following method could be generalized to additional modalities by adapting the mask to the modality type (e.g. masking image patches with blurring or inpainting). Shapley values (Shapley, 1953), based on cooperative game theory, is a method, when applied to machine learning, which computes the contribution of features to a model's prediction. Here, we present a sampling-based algorithm that aims to explain why a prediction is assessed as erroneous in terms of feature values $\mathbf{x}$. Our method adapts the algorithm from Štrumbelj and Kononenko (2010), which approximates Shapley values by randomly and repeatedly selecting a subset of features instead of all possible coalitions in order to overcome exponential time complexity. We make several adaptations to achieve our objective. Firstly, we do not aim to explain the classifier's predictions; our goal is to justify why a model might fail. Therefore, we leverage a different kind of value function (EDIP) to estimate the feature contributions. Secondly, the context is multimodal; in particular, we focus on tabular-text data and models. It is worth noting that EDIP computes probabilities based on embeddings $\mathbf{z}$ while we want to generate explanations in terms of the input values $\mathbf{x}$. In a nutshell, for a new test input $\mathbf{x}'$ (with $\mathbf{z}' = \phi(\mathbf{x}')$) and EDIP $\hat{f}$, we want to understand what contributes to $\hat{f}_1(\mathbf{z}') - \mathbb{E}_{i\sim\mathcal{I}_{val}}[\hat{f}_1(\mathbf{z}_i)]$, in terms of text and tabular feature values.

The approach is described in Algorithm 1 for a test input $\mathbf{x}$, where we compute the average contribution of a tabular feature with index j or a text feature (i.e. token) with index (i.e. position) j. We perform M Monte Carlo iterations to approximate the Shapley value. In order to assess the marginal contribution of a feature value with feature index j, we construct two new instances $\mathbf{x}_{+j}$ and $\mathbf{x}_{-j}$ from $\mathbf{x}$ by combining the effect of randomness in samples from $\mathcal{D}_{val}$ and in feature indices for tabular and text modalities. To mask tokens, we replace text tokens by the mask token [MASK] (Devlin et al., 2019). As a value function, EDIP model $\hat{f}$ is used to assess the marginal contribution of the feature value to the predicted probability that $\hat{\pi}$ will fail. Lastly, we compute the Shapley value as the average of contributions over M iterations.

Measuring the quality of explanations. To measure the quality of explanations produced by Algo-

rithm 1, we suggest verifying the consistency with outputs generated by other techniques. Firstly, a different value function can be used in Algorithm 1, in order to assess the feature contributions. For instance, deep ensembles (Lakshminarayanan et al., 2017) can be leveraged to compute the contribution to uncertainty: $\Phi_j^m(\mathbf{x}) = u(\mathbf{z}_{+j}) - u(\mathbf{z}_{-j})$, where

$$u(\mathbf{z}) = -\sum_{j \in \mathbb{Y}} \left(\frac{1}{E} \sum_{e=1}^{E} p(j|\mathbf{z}; \theta_e) \right) \log_2 \left(\frac{1}{E} \sum_{e=1}^{E} p(j|\mathbf{z}; \theta_e) \right)$$

In that case, the marginal contribution $\Phi_j^m(\mathbf{x})$ from line 15 in Algorithm 1 equals the difference in predictive entropies computed with E neural networks with respective parameters θ_e

Secondly, in Algorithm 1, each perturbation sample ($\mathbf{x}_{+j}$ and $\mathbf{x}_{-j}$) can be modified into a vector $\mathbf{v} \in \{0,1\}^{(|\mathcal{I}^{tab}|+|\mathcal{I}^{text}|)}$, where each entry from $\mathbf{v}$ equals 1 when the corresponding feature value from $\mathbf{x}$ is present and 0 when it is absent. $|\mathcal{I}^{tab}|$ and $|\mathcal{I}^{text}|$ denote the numbers of tabular features and text tokens, respectively. If we compute Algorithm 1 for the $|\mathcal{I}^{tab}| + |\mathcal{I}^{text}|$ features, we can obtain $2 \times M \times (|\mathcal{I}^{tab}| + |\mathcal{I}^{text}|)$ instances of $\mathbf{v}$ and related $\hat{f}_1(.)$ values (i.e. $\hat{f}_1(\mathbf{z}_{+j})$ and $\hat{f}_1(\mathbf{z}_{-j})$ for M iterations and $|\mathcal{I}^{tab}| + |\mathcal{I}^{text}|$ features). Then, we compute the Kernel SHAP weights for each $\mathbf{v}$ (Lundberg and Lee, 2017) and fit a weighted Lasso regression $\hat{r} : \{0,1\}^{(|\mathcal{I}^{tab}|+|\mathcal{I}^{text}|)} \rightarrow \mathbb{R}$, where $\mathbf{v}$ are the features and $\hat{f}_1(.)$ the response values (or $u(.)$ for deep ensembles). Lastly, the coefficients in this regression function are the Kernel SHAP feature contributions.

The consistency between the outputs obtained with EDIP and those generated by each of these alternative methods can be assessed, by computing the Pearson correlation coefficients.

4 Experiments

We empirically test the relevance of our method on various classification datasets. In the appendix, we provide further details on the experimental settings and results (e.g. datasets, data preprocessing, multimodal architectures, baselines, variability in results).

4.1 Settings

Datasets. We test the relevance of our method on 7 classification datasets, with a number of classes ranging from 2 to 100. For tabular-text applications, we use airbnb, cloth, kick, petfinder, and wine with the 10/100 most frequent classes (referred to as wine10 and wine100, respectively). These datasets have been tested by (Shi et al., 2021) and (Gu and Budhkar, 2021). The text-image use case is based on Food-101 dataset (Bossard et al., 2014) with image and textual information (Gallo et al., 2020). For this dataset, we concentrate on the first five classes arranged in alphabetical order (food5).

Architectures. For the multimodal tabular-text classifier $\hat{\pi}$, we employ four different architectures: (1) AllText-BERT-TaB: The tabular features, converted to strings, and the text fields are concatenated and input into BERT-base-uncased (Devlin et al., 2019) as text; (2) LateFuse-BERT-TaB (Figure 1): A tabular-text dual-stream model with late concatenation of the [CLS] tokens' final hidden states extracted from BERT-base-uncased and a tabular Transformer; (3) AllText-DBERT-TaB: This architecture is similar to AllText-BERT-TaB, except that we employ DistilBERT-base-uncased (Sanh et al., 2019) instead of BERT; (4) LateFuse-DBERT-TaB: Similar to LateFuse-BERT-TaB with DistilBERT-base-uncased for the text stream instead of BERT. Each pretrained model is fully fine-tuned on $\mathcal{D}_{train}$ with a batch size of 32, by minimizing the cross-entropy loss with AdamW algorithm (Loshchilov and Hutter, 2019), with a learning rate of $5e-5$.

For the text-image classifier $\hat{\pi}$, we employ the following architectures: (1) BERT-ViT: A text-image dual-stream model with late concatenation of the [CLS] tokens' final hidden states extracted from BERT-base-uncased and the Vision Transformer ViT-base-patch16-224 (Dosovitskiy et al., 2021); (2) DBERT-ViT: This architecture is similar to BERT-ViT, except that we employ DistilBERT-base-uncased instead of BERT. Each pretrained model is fully fine-tuned on $\mathcal{D}_{train}$ with a batch size of 64, by minimizing the cross-entropy loss with stochastic gradient descent, with a learning rate of $1e-3$.

For all the models, we use early stopping with patience of 1 for the accuracy on $\mathcal{D}_{val}$. An exponential learning rate scheduler with gamma of 0.9 is employed. We keep the best model in terms of epochs, i.e. with the highest accuracy on $\mathcal{D}_{val}$.

Evaluation. For each experiment, all the methods are calibrated on the validation data $\mathcal{D}_{val}$ and evaluated on the same test dataset $\mathcal{T}$ with a size of 1000 rows. Each use case is run over 5 different random dataset partitions. The final hidden state of the classification token [CLS] (referred to as $\mathbf{z}_{[\text{CLS}]}^{\text{last}}$) and the softmax output $\hat{\pi}(\mathbf{x})$ are extracted from $\hat{\pi}$.

Model	Dataset	AC	ACSC	CP	DC	DENS	DNN	EDIP	ENRG	MCD	TCP
AllText-BERT-TaB	airbnb	0.590	0.603	0.612	0.503	0.612	0.527	**0.636**	0.594	0.596	0.515
	cloth	0.746	0.753	0.717	0.508	**0.770**	0.670	0.764	0.726	0.750	0.556
	kick	0.871	0.871	0.598	0.493	**0.884**	0.487	0.874	0.759	0.874	0.599
	petfinder	0.551	0.556	0.547	0.465	0.551	0.496	**0.583**	0.512	0.546	0.506
	wine10	0.859	0.664	0.736	0.511	0.855	0.715	**0.873**	0.807	0.846	0.528
	wine100	**0.864**	0.732	0.844	0.485	0.844	0.644	0.861	0.853	0.852	0.578
LateFuse-BERT-TaB	airbnb	0.629	0.638	0.631	0.511	0.624	0.525	0.636	0.619	**0.644**	0.528
	cloth	0.748	0.712	0.710	0.524	**0.770**	0.666	0.761	0.671	0.753	0.559
	kick	0.838	0.839	0.626	0.484	0.861	0.569	**0.886**	0.651	0.839	0.573
	petfinder	0.591	0.594	0.585	0.496	0.577	0.471	**0.615**	0.509	0.583	0.534
	wine10	0.863	0.864	0.774	0.489	0.864	0.679	**0.865**	0.817	0.853	0.538
	wine100	0.869	0.646	0.852	0.516	0.850	0.715	**0.870**	0.850	0.860	0.579
AllText-DBERT-TaB	airbnb	0.630	0.631	0.620	0.503	0.600	0.547	**0.649**	0.575	0.628	0.527
	cloth	0.763	0.768	0.726	0.514	**0.771**	0.655	0.766	0.694	0.762	0.549
	kick	0.863	0.863	0.572	0.484	**0.875**	0.561	0.865	0.742	0.863	0.608
	petfinder	0.568	0.568	0.561	0.490	0.554	0.527	**0.590**	0.532	0.570	0.502
	wine10	**0.873**	**0.873**	0.758	0.491	0.848	0.755	0.867	0.831	0.858	0.501
	wine100	0.869	**0.871**	0.849	0.486	0.839	0.650	0.865	0.850	0.857	0.555
LateFuse-DBERT-TaB	airbnb	0.618	0.631	0.626	0.502	0.609	0.518	**0.638**	0.606	0.614	0.549
	cloth	0.742	0.748	0.731	0.537	0.769	0.672	**0.774**	0.655	0.747	0.551
	kick	0.842	0.842	0.598	0.501	0.860	0.634	**0.871**	0.628	0.844	0.547
	petfinder	0.574	0.574	0.560	0.493	0.571	0.490	**0.602**	0.524	0.572	0.523
	wine10	0.850	0.745	0.776	0.523	0.855	0.707	**0.869**	0.818	0.835	0.536
	wine100	0.867	**0.871**	0.854	0.494	0.840	0.698	0.868	0.851	0.854	0.596
BERT-ViT	food5	**0.912**	0.754	0.493	0.461	0.897	0.869	**0.912**	0.882	0.905	0.593
DBERT-ViT	food5	**0.894**	0.890	0.555	0.456	0.889	0.852	0.893	0.789	0.865	0.576
Average rank		3.4	3.7	5.9	9.9	3.8	8.1	**1.6**	6.2	3.9	8.5

Table 1: **Evaluation of the methods with AUROC** computed on the test data for 5 random seeds. For a given model and dataset, the best result is in **bold** (higher is better). The last row displays the average rank over models and datasets. The variability in results is displayed in appendix G. The performance (error rate) of classifiers is displayed in appendix E.

For the architectures based on late fusion, $\mathbf{z}_{[\text{CLS}]}^{\text{last}}$ is the concatenation of the Transformer streams' final hidden states of the [CLS] tokens (i.e. states before the classification head). For EDIP, we use a random forest algorithm with the default hyperparameter setting from *Scikit-learn* Python package (Pedregosa et al., 2011). $\mathbf{z}_{[\text{CLS}]}^{\text{last}}$ and $\hat{\pi}(\mathbf{x})$ are concatenated and used as features for EDIP. Our method is compared to the following baselines previously described in section 2.

9 baselines are used for error detection, where the scores are computed for a given test input:

- AC (Average Confidence): The score is one minus the maximum confidence (i.e. one minus the maximum softmax probability).
- ACSC (Average Confidence - SCaled): The score is one minus the maximum confidence after applying temperature scaling to the softmax output. The temperature is set by optimizing the Expected Calibration Error (ECE) (Guo et al., 2017b) with the L-BFGS algorithm (Liu and Nocedal, 1989) on $\mathcal{D}_{val}$.
- CP (Conformal Prediction): The score is the prediction set size computed with the weighted conformal prediction (Tibshirani et al., 2019) based on LAC method (Sadinle et al., 2019).
- DC (Domain Classifier): We use the class 1's predicted probability.
- DENS (Deep Ensembles): The uncertainty is assessed with the predictive entropy, after averaging the probabilities from a deep ensemble of 5 neural networks trained with $\mathbf{z}_{[\text{CLS}]}^{\text{last}}$.
- DNN (deep nearest neighbors): We use the distance to the k-th neighbor ($k = 10$) from the validation data with the deep nearest neighbors fitted with $\mathbf{z}_{[\text{CLS}]}^{\text{last}}$ as features. The feature space is normalized with the L2 norm as a pre-requisite, as advised by Sun et al. (2022).

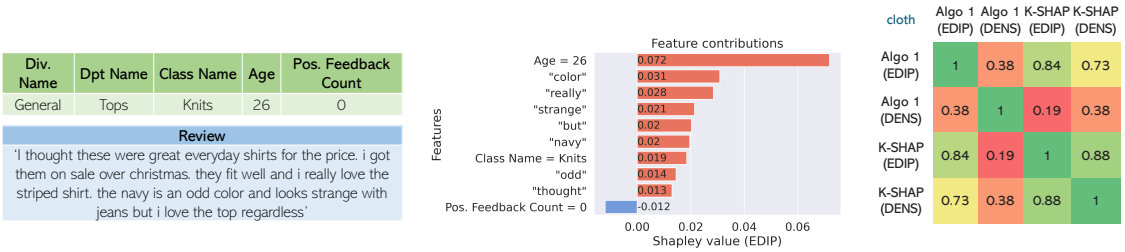

Figure 2: *Left*: Multimodal input from cloth dataset, where the task is to predict a product score granted by the customer from 1 worst, to 5 best. The true rating is $y = 5$ whereas LateFuse-BERT-TaB predicts $\hat{y} = 4$. EDIP outputs a probability of error of 71%. *Middle*: Top 10 feature contributions to the predicted likelihood of error, computed with Algorithm 1 leveraging EDIP as value function. Positive contributions are displayed in red. *Right*: Pearson correlation matrix between the outputs of various explanation methods: algorithms (Algo 1: Algorithm 1, K-SHAP: Kernel SHAP) and value functions (EDIP, DENS).

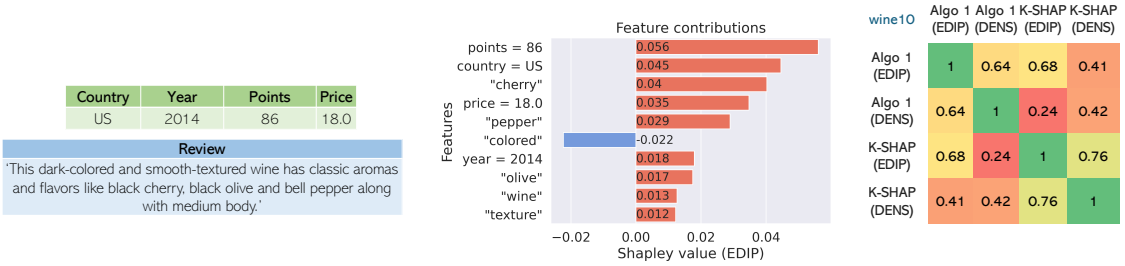

Figure 3: *Left*: Multimodal input from wine10 dataset, where the task is to predict the variety of grapes. The true label is *Cabernet Sauvignon* while LateFuse-BERT-TaB predicts *Red Blend*. EDIP outputs a probability of error of 70%. *Middle*: Top 10 feature contributions to the predicted likelihood of error, computed with Algorithm 1 leveraging EDIP as value function. *Right*: Pearson correlation matrix between the outputs of various explanation methods: algorithms (Algo 1: Algorithm 1, K-SHAP: Kernel SHAP) and value functions (EDIP, DENS).

- ENRG: We employ the energy score.
- MCD (Monte Carlo Dropout): The uncertainty is assessed with the predictive entropy. We enable the dropout layers from $\hat{\pi}$ during test-time. The dropout probability is set to 0.1. For each test example, we perform $P = 5$ forward passes with $\hat{\pi}$ and corresponding parameters θ_p. Then, we calculate the total uncertainty (entropy) after averaging the predicted probabilities:

$$u(\mathbf{x}) = -\sum_{j \in \mathbb{Y}} \left(\frac{1}{P} \sum_{p=1}^{P} \hat{\pi}_j(\mathbf{x}; \theta_p) \right) \log_2 \left(\frac{1}{P} \sum_{p=1}^{P} \hat{\pi}_j(\mathbf{x}; \theta_p) \right)$$

- TCP (True Class Probability): The score is one minus the true class probability estimated with a neural network trained with $\mathbf{z}^{\text{last}}_{[\text{CLS}]}$.

For a given architecture (e.g. LateFuse-BERT-TaB), the performance in error detection is assessed by computing AUROC (Area Under the Receiver Operating Characteristic curve) with all the test data from different seeds: we calculate the scores for accurate (label 0) and incorrect (label 1) predictions, and quantify how well these two labels are separated for a range of thresholds. Lastly, we also perform ablation studies to compare the results of EDIP with (1) Ablation 1: EDIP using only $\mathbf{z}^{\text{last}}_{[\text{CLS}]}$ as features, or (2) Ablation 2: EDIP leveraging only the classifier's output $\hat{\pi}(\mathbf{x})$.

Explanation algorithm. We experiment with two different value functions: EDIP and deep ensembles. In order to accelerate the computation of Shapley values, we stop the iterations when a convergence criteria is reached. To achieve that, we first compute the maximum absolute difference between the previous and updated Shapley values, every 10 iterations and for each value function. We end the process when the maximum of these two values is lower than 0.01.

4.2 Results

Evaluation of the methods. The results in Table 1 show that EDIP outperforms the other methods in error detection over the various model architectures and datasets. EDIP secures first rank in 15 out of 26 use cases, with an average rank of 1.6.

Methods based on the maximum softmax probability (AC, ACSC) and those based on uncertainty quantification (DENS, MCD) also achieve good AUROC performance, with average ranks ranging from 3.4 to 3.9. The domain classifier (DC) and deep nearest neighbors (DNN) may be more appropriate for OOD detection than error detection, which may explain their performance here.

Explanation algorithm. Figure 2 shows an example from cloth dataset where LateFuse-BERT-TaB underestimates the rating for the corresponding input. EDIP estimates a probability of error of 71%. Further, our sampling-based algorithm displays the top 10 feature contributions to the assessed probability of failure. In particular, the bar plot (middle) displays the combination of certain tabular feature values (e.g. Age = 26) and tokens (e.g. "strange", "but") that contribute to EDIP estimation and might explain why LateFuse-BERT-TaB wrongly predicted a lower rating. Lastly, the correlation matrix (right) shows that three explanation methods are in agreement (Algorithm 1 with EDIP, Kernel SHAP with EDIP, Kernel SHAP with DENS) and may be more reliable than the remaining one. This explanation method could be useful in critical applications (e.g. financial or medical field) where subject matter experts need to understand if and why a prediction is likely to be incorrect.

Lastly, another example is displayed in Figure 3, where LateFuse-BERT-TaB predicts an incorrect variety of grapes for a multimodal input from wine10 dataset. EDIP estimates a high likelihood of error (70%). The bar plot (middle) provide clues to explain why the classifier may have mixed up the varieties of grapes, with positive contributions from tabular features values (e.g. country = US, price = 18.0) and tokens (e.g. "cherry", "pepper"). The correlation matrix (right) shows that Algorithm 1 with EDIP is quite reliable in that case: its outputs moderately or strongly correlate with the outputs of the other methods.

Ablation studies. The results for each model architecture and dataset are presented in Table 2. Even though Ablation 1 and EDIP sometimes achieve very close AUROC, EDIP turns out to perform best in detecting errors overall (first rank in 18 out of 26 use cases). Further, EDIP seems to be more stable, while Ablation 2's performance is significantly lower for some of the use cases: e.g. Ablation 2's AUROC on kick dataset with AllText-BERT-TaB.

Model	Dataset	Abl.1	EDIP	Abl.2
AllText-BERT-TaB	airbnb	**0.638**	0.636	0.608
	cloth	**0.764**	**0.764**	0.718
	kick	**0.874**	**0.874**	0.790
	petfinder	0.572	**0.583**	0.536
	wine10	0.872	**0.873**	0.863
	wine100	0.846	**0.861**	0.859
LateFuse-BERT-TaB	airbnb	0.623	0.636	**0.640**
	cloth	**0.762**	0.761	0.725
	kick	0.885	**0.886**	0.764
	petfinder	**0.618**	0.615	0.555
	wine10	0.860	**0.865**	0.864
	wine100	0.855	0.870	**0.871**
AllText-DBERT-TaB	airbnb	0.648	**0.649**	0.614
	cloth	**0.767**	0.766	0.726
	kick	**0.865**	**0.865**	0.773
	petfinder	0.589	**0.590**	0.557
	wine10	0.859	**0.867**	0.866
	wine100	0.849	**0.865**	0.864
LateFuse-DBERT-TaB	airbnb	0.632	**0.638**	0.635
	cloth	**0.776**	0.774	0.732
	kick	**0.874**	0.871	0.747
	petfinder	0.597	**0.602**	0.532
	wine10	0.858	**0.869**	0.856
	wine100	0.856	**0.868**	0.866
BERT-ViT	food5	0.905	**0.912**	0.900
DBERT-ViT	food5	0.890	**0.893**	0.890

Table 2: **Results of the ablation studies with AUROC** computed on the test data for 5 random seeds. For a given model and dataset, the best results are in **bold** (higher is better). Ablation 1 (Abl.1): EDIP using only $\mathbf{z}_{[\text{CLS}]}^{\text{last}}$ as features. Ablation 2 (Abl.2): EDIP leveraging only the classifier's output $\hat{\pi}(\mathbf{x})$.

5 Conclusion

We introduced a method to compute and explain the likelihood of failure in multimodal classification tasks. We compared our method to 9 baselines and evidenced that EDIP can be a useful approach to identify misclassified inputs. Detecting errors and providing explanations to subject matter experts is a first step toward safer machine learning systems. With this type of human-AI interaction, experts can thus make more informed decisions, justify their choice, and override the classifier's output if necessary. Future work could address the case of other modalities and multimodal OOD settings.

6 Limitations

Multimodal datasets for safety-critical tasks. The multimodal datasets employed in this paper are not related to real-world high-stakes applications

such as diagnosis prediction or financial decision-making. We expect more multimodal datasets related to safety-critical tasks to be publicly shared in order to test the relevance of our method in environments where incorrect predictions can lead to serious consequences.

Relevance of perturbation-based explanation method. For text features, the perturbation-based explanation method is based on inserting [MASK]. Such synthetic perturbations do not reflect natural linguistic variation. Consequently, this might affect the accuracy of feature attributions. Other perturbation-based approaches could be considered.

Multimodal OOD settings. We have not evaluated the performance of EDIP when the test data is affected by distribution shifts in a multimodal context; this remains to be seen. In particular, it would be useful to understand how to identify invariant representations so that EDIP could detect mispredicted inputs in shifting environments.

Computational complexity. The computational complexity of EDIP may be a restrictive factor, especially in large-scale multimodal applications (e.g. high embedding dimensions). The method requires training an auxiliary classifier to detect error patterns, and its reliance on Monte Carlo-based Shapley value approximation introduces additional computational overhead. Therefore, applying this method to high-dimensional inputs can significantly step up processing time and memory requirements.

7 Ethical Considerations

Our method is not intended to predict or exploit any sensitive information. On the contrary, it aims to make machine learning systems safer. Therefore, we do not expect any significant risks with respect to social or environmental issues. However, it is important to monitor the performance of our method over time, in order to train it on fresh data when necessary.

References

Javier Antoran, Umang Bhatt, Tameem Adel, Adrian Weller, and José Miguel Hernández-Lobato. 2021. Getting a clue: A method for explaining uncertainty estimates. In *International Conference on Learning Representations*.

Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. 2014. Food-101 – mining discriminative components with random forests. In *Computer Vision – ECCV 2014*, pages 446–461. Springer International Publishing.

L Breiman, JH Friedman, R Olshen, and CJ Stone. 1984. Classification and regression trees.

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

Jiefeng Chen, Frederick Liu, Besim Avci, Xi Wu, Yingyu Liang, and Somesh Jha. 2021. Detecting errors and estimating accuracy on unlabeled data with self-training ensembles. In *Advances in Neural Information Processing Systems*, volume 34, pages 14980–14992.

Charles Corbière, Nicolas Thome, Avner Bar-Hen, Matthieu Cord, and Patrick Pérez. 2019. Addressing failure prediction by learning model confidence. In *Advances in Neural Information Processing Systems*, volume 32.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA. PMLR.

Ignazio Gallo, Gianmarco Ria, Nicola Landro, and Riccardo La Grassa. 2020. Image and text fusion for upmc food-101 using bert and cnns. In *2020 35th International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pages 1–6.

Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. 2023. A survey of uncertainty in deep neural networks. *Artificial Intelligence Review*, 56(Suppl 1):1513–1589.

Ken Gu and Akshay Budhkar. 2021. A package for learning on tabular and text data with transformers. In *Proceedings of the Third Workshop on Multimodal Artificial Intelligence*, pages 69–73, Mexico City, Mexico. Association for Computational Linguistics.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017a. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017b. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*.

Satyapriya Krishna, Tessa Han, Alex Gu, Steven Wu, Shahin Jabbari, and Himabindu Lakkaraju. 2024. The disagreement problem in explainable machine learning: A practitioner's perspective. *Transactions on Machine Learning Research*.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, volume 30.

Paul Pu Liang, Amir Zadeh, and Louis-Philippe Morency. 2024. Foundations & trends in multimodal machine learning: Principles, challenges, and open questions. *ACM Computing Surveys*, 56(10).

Dong C Liu and Jorge Nocedal. 1989. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528.

Weitang Liu, Xiaoyun Wang, John Owens, and Yixuan Li. 2020. Energy-based out-of-distribution detection. In *Advances in Neural Information Processing Systems*, volume 33, pages 21464–21475.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30.

Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern recognition*, 45(1):521–530.

Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. 2019. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in Neural Information Processing Systems*, volume 32.

Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alexander Gammerman. 2002. Inductive confidence machines for regression. In *Machine Learning: ECML 2002, 13th European Conference on Machine Learning, Helsinki, Finland, August 19-23, 2002, Proceedings*, volume 2430 of *Lecture Notes in Computer Science*, pages 345–356. Springer.

Letitia Parcalabescu and Anette Frank. 2023. MM-SHAP: A performance-agnostic metric for measuring multimodal contributions in vision and language models & tasks. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4032–4059, Toronto, Canada. Association for Computational Linguistics.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(85):2825–2830.

Stephan Rabanser, Stephan Günnemann, and Zachary Lipton. 2019. Failing loudly: An empirical study of methods for detecting dataset shift. In *Advances in Neural Information Processing Systems*, volume 32.

Mauricio Sadinle, Jing Lei, and Larry Wasserman. 2019. Least ambiguous set-valued classifiers with bounded error levels. *Journal of the American Statistical Association*, 114(525):223–234.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423.

LS Shapley. 1953. A value for n-person games. In *Contributions to the Theory of Games (AM-28), Volume II*. Princeton University Press.

Xingjian Shi, Jonas Mueller, Nick Erickson, Nick Erickson, Mu Li, Alexander Smola, and Alexander Smola. 2021. Benchmarking multimodal automl for tabular data with text fields. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1.

Erik Štrumbelj and Igor Kononenko. 2010. An efficient explanation of individual classifications using game theory. *Journal of Machine Learning Research*, 11(1):1–18.

Yiyou Sun, Yifei Ming, Xiaojin Zhu, and Yixuan Li. 2022. Out-of-distribution detection with deep nearest neighbors. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 20827–20840. PMLR.

Ryan J Tibshirani, Rina Foygel Barber, Emmanuel Candes, and Aaditya Ramdas. 2019. Conformal prediction under covariate shift. In *Advances in Neural Information Processing Systems*, volume 32.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. 2005. *Algorithmic learning in a random world*. Springer Science & Business Media.

David Watson, Joshua O' Hara, Niek Tax, Richard Mudd, and Ido Guy. 2023. Explaining predictive uncertainty with information theoretic shapley values. In *Advances in Neural Information Processing Systems*, volume 36, pages 7330–7350.

Peng Xu, Xiatian Zhu, and David A. Clifton. 2023. Multimodal learning with transformers: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(10):12113–12132.

A Appendix: Table of Contents

B Datasets and Sampling

All the datasets are publicly available with one of these licenses: "CC0: Public Domain", "Competition Data", or "CC BY-NC-SA 4.0". These datasets can be accessed and used for the purpose of academic research. The text fields are in English.

In Table 3, we give more details on the datasets:

- airbnb[1]: the task is to predict the price range of Airbnb listings. The text fields are listing descriptions.
- cloth[2]: the goal is to classify the sentiment (represented as a class) of user reviews regarding clothing items. The text fields are customer reviews.
- kick[3]: the task is to predict whether a proposed project will achieve its funding goal. The text fields are project descriptions.
- petfinder[4]: the goal is to predict the speed range at which a pet is adopted. The text fields are profile write-ups for the pets.
- wine[5]: the goal is to predict the variety of grapes. The text fields are wine tasting descriptions.
- food5[6]: the goal is to predict the recipe. The cations are textual information about the recipe.

For some of the use cases, we employ the original training dataset as the test dataset does not include the true labels (competition data). In that

[1] https://www.kaggle.com/datasets/tylerx/melbourne-airbnb-open-data

[2] https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews

[3] https://www.kaggle.com/datasets/codename007/funding-successful-projects

[4] https://www.kaggle.com/competitions/petfinder-adoption-prediction/data

[5] https://www.kaggle.com/datasets/zynicide/wine-reviews

[6] https://www.kaggle.com/datasets/gianmarco96/upmcfood101

Dataset	# Train	# Num	# Cat	# Class
airbnb	4,372	27	23	10
cloth	13,955	2	3	5
kick	69,194	3	3	2
petfinder	9,324	5	14	5
wine10	39,320	2	2	10
wine100	65,398	2	2	100
food5	2,700	-	-	5

Table 3: **Information on datasets**: number of samples in training dataset, number of numerical/categorical features, number of classes.

case, we consider the training dataset as the modeling data which is then randomly split into training-validation-test subsets. The datasets are partitioned as follows: (1) The initial dataset is randomly split into two disjoint temporary (80% share) and test $\mathcal{T}$ (20% share) subsets, respectively; (2) The temporary dataset is randomly split into two disjoint training $\mathcal{D}_{train}$ (80% share) and validation $\mathcal{D}_{val}$ (20% share) subsets, respectively. For the evaluation of the methods, 1000 rows are randomly extracted from the original test dataset.

C Data Preprocessing

Feature engineering. When the dataset contains several text fields, these are concatenated in order to obtain a single field. Rows with missing values are dropped and duplicate rows removed. The list of final features for each dataset is described below. We also mention here additional features that were created from the raw dataset.

- airbnb: for this dataset only, we discretize the target variable by employing quantile binning (ten intervals with equal share of data). We also create two new features *host_since_year* and *last_review_year* by extracting the year from *host_since* and *last_review* respectively. Categorical variables: *host_location, host_since_year, host_is_superhost, host_neighborhood, host_has_profile_pic, host_identity_verified, neighborhood, city, smart_location, suburb, state, is_location_exact, property_type, room_type, bed_type, instant_bookable, cancellation_policy, require_guest_profile_picture, require_guest_phone_verification, host_response_time, calendar_updated, host_verifications, last_review_year*; numerical variables: *host_response_rate, latitude, longitude, accommodates, bathrooms, bedrooms, beds, security_deposit, cleaning_fee, guests_included, extra_people, minimum_nights, maximum_nights, availability_30, availability_60, availability_90, availability_365, number_of_reviews, review_scores_rating, review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_scores_location, review_scores_value, calculated_host_listings_count, reviews_per_month*; text fields: *name, summary, description*.

- cloth: categorical variables: *Division Name, Department Name, Class Name*; numerical variables: *Age, Positive Feedback Count*; text fields: *Title, Review Text*.

- kick: we compute the duration to launch (in days) with *deadline* and *launched_at*. We also log-transform *goal*. Categorical variables: *country, currency, disable_communication*; numerical variables: *log_goal, backers_count, duration*; text fields: *name, desc*.

- petfinder: Categorical variables: *Type, Breed1, Breed2, Gender, Color1, Color2, Color3, MaturitySize, FurLength, Vaccinated, Dewormed, Sterilized, Health, State*; numerical variables: *Age, Quantity, Fee, VideoAmt, PhotoAmt*; text field: *Description*.

- wine10 and wine100: we extract the *year* from *title*. Categorical variables: *country, year*; numerical variables: *points, price*; text field: *description*.

Text preprocessing. We perform the following text preprocessing: we keep words, numbers, and whitespaces. We then use the BERT-base-uncased or DistilBERT-base-uncased tokenizer based on WordPiece. For the text sequence length, the value is set to the 0.9 quantile of the text field lengths' distribution in the source dataset. We then take the minimum of this latter value and 512 as this is the maximum sequence length for BERT models. We use truncation and padding to the fixed maximum length.

Image preprocessing. We use the ViT-base-patch16-224 image processor.

Attention mask for text tokens. We use key attention masks in order to specify which text tokens

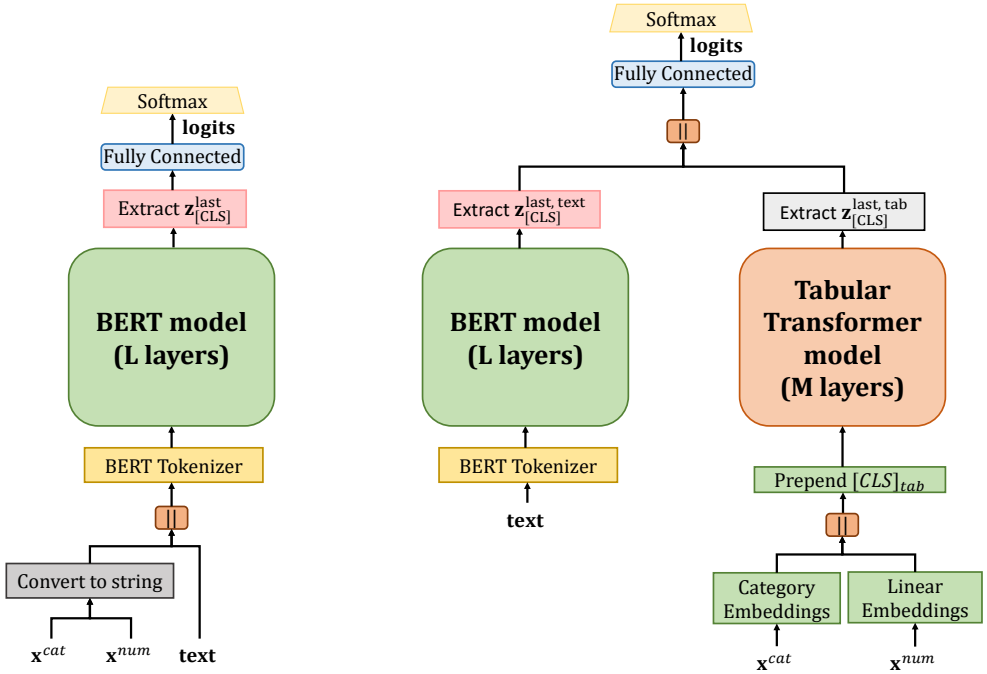

Figure 4: **Classification model architectures**. *Left:* AllText-BERT-TaB architecture. *Right:* LateFuse-BERT-TaB architecture.

should be ignored (i.e. "padding") for the purpose of attention.

D Model Architectures

LateFuse architecture. The architecture is detailed in Figure 4 (right) with BERT-base-uncased for the text stream. For numerical features, we first perform standard scaling. Embeddings of the LateFuse architecture are constructed with linear functions. A linear function applies the following transformation to a scalar feature value $x \in \mathbb{R}$: $x.W_{num} + b$ where $W_{num} \in \mathbb{R}^d$ and the bias $b \in \mathbb{R}^d$. For categorical features, we encode them as category embeddings. In that latter case, the corresponding embedding is computed as $e^T W_{cat}$ where $e \in \mathbb{R}^{n_c \times 1}$ is a one-hot-vector for the associated categorical feature, n_c denote the number of categories for this feature, and $W_{cat} \in \mathbb{R}^{n_c \times d}$. A classification token [CLS] is then added to the beginning of the tabular embedding sequence. The tabular Transformer with self-attention has the following architecture: 3 layers, 8 attention heads, feed-forward dimension of 768, embedding dimension of 768. The dropout (rate 0.1) is applied to the category embeddings, the tabular Transformer (attention, feed-forward networks), and the final fully-connected networks. The text and tabular Transformer's final hidden states of the [CLS] tokens are concatenated before being projected through fully-connected layers to produce the logits. The uniform weight initialization for the category/linear embeddings and the final fully-connected networks is based on Kaiming (He et al., 2015). The final fully-connected layers can be described as follows: $\text{FC}(x) = \text{Linear}(\text{Dropout}(\text{ReLU}(\text{Linear}(x))))$ where the output has a dimension of C (number of classes). Lastly, the architectures have the following number of parameters: LateFuse-BERT-TaB: 124,536,170 and LateFuse-DBERT-TaB: 81,416,810.

AllText architecture. The architecture is detailed in Figure 4 (left) with BERT-base-uncased. The tabular features, converted to strings, and the text fields are concatenated and input into BERT-base-uncased as text. The final hidden state of the [CLS] token (i.e. before the classification head) are projected through fully-connected layers to produce the logits. The uniform weight initialization for the final fully-connected networks is based on Kaiming. The final fully-connected layers can be described as follows: $\text{FC}(x) = \text{Linear}(\text{Dropout}(\text{ReLU}(\text{Linear}(x))))$ where the output has a dimension of C (number of classes). The dropout rate is 0.1 in the final

Model	Dataset	Error rate
AllText-BERT-TaB	airbnb	0.693
	cloth	0.283
	kick	0.140
	petfinder	0.662
	wine10	0.178
	wine100	0.304
LateFuse-BERT-TaB	airbnb	0.684
	cloth	0.297
	kick	0.161
	petfinder	0.657
	wine10	0.185
	wine100	0.334
AllText-DBERT-TaB	airbnb	0.690
	cloth	0.297
	kick	0.131
	petfinder	0.664
	wine10	0.182
	wine100	0.312
LateFuse-DBERT-TaB	airbnb	0.683
	cloth	0.305
	kick	0.150
	petfinder	0.643
	wine10	0.186
	wine100	0.332
BERT-ViT	food5	0.152
DBERT-ViT	food5	0.161

Table 4: **Error rate of classifiers** on the test dataset, by dataset, averaged over 5 random seeds.

fully-connected networks. Lastly, the architectures have the following number of parameters: AllText-BERT-TaB: 109,507,178 and AllText-DBERT-TaB: 66,387,818.

BERT-ViT and DBERT-ViT architectures. The embedding dimension for each stream is 768, therefore the concatenated dimension is 2×768 after fusing the modalities. The dropout rate is 0.1. The architectures have the following number of parameters: BERT-ViT: 197,055,749 and DBERT-ViT: 153,936,389.

E Performance of Classification Models

The performance of classifiers are displayed in Table 4.

F Details on Baselines

[CLS] token's final hidden state as feature. For the methods leveraging the [CLS] tokens' final hidden states, it is worth mentioning that when $\hat{\pi}$ is based on LateFuse architecture, the text and tabular hidden states are concatenated (see Figure 4). In that case, the final vector is of dimension 2×768. The same principle is used for the text-image use case.

Further details on baselines. We provide further details for some of the baselines used to perform error detection:

- DC: For the domain classifier, we employ a Random Forest with 10 estimators. We divide both the validation data and test data into two halves, using the first half to train a domain classifier to classify validation (class 0) and test (class 1) data. We then apply this model to the second half and compute the AUROC. We follow the same process by selecting the second half to fit the domain classifier and computing the AUROC on the first half. Lastly, we average the 2 AUROC values.

- CP: For the weighted conformal prediction, we compute weighted quantiles. Each weight is computed with the domain classifier as $\hat{p}_{dc}(\mathbf{z})/(1-\hat{p}_{dc}(\mathbf{z}))$, where $\hat{p}_{dc}(\mathbf{z})$ is the probability that the input is from the test subset given $\mathbf{z}$. This approach is suggested in (Tibshirani et al., 2019). With the LAC method, the conformity score corresponds to one minus the probability of the true class. For this baseline, we set the quantile to 90%, which is the expected coverage.

- ENRG: In the energy score formula, we set the temperature to 1.

- TCP: The neural network used to estimate the true class probability has the following architecture: $\text{NN}(x) = \text{Linear}(\text{Dropout}(\text{ReLU}(\text{Linear}(x))))$. The dropout probability is set to 0.1. The (input shape, output shape) for the first linear layer is compatible with the dimension of $\mathbf{z}$ (768×768 for the AllText architecture and $(2 \times 768) \times (2 \times 768)$ for architectures based on late fusion). As this is a regression task, the final output has dimension 1. The mean squared error loss is optimized with Adam (learning rate of $1e-3$) for 10 epochs and batch size of 32.

- DENS: An ensemble of 5 neural networks is trained, where each neural network has the following architecture: $\text{NN}(x) = \text{Linear}(\text{Dropout}(\text{ReLU}(\text{Linear}(x))))$. The dropout probability is set to 0.1. The (input shape, output shape) for the first linear layer is compatible with the dimension of $\mathbf{z}$

Model	Dataset	AC	ACSC	CP	DC	DENS	DNN	EDIP	ENRG	MCD	TCP
AllText-BERT-TaB	airbnb	0.010	0.010	0.011	0.013	0.009	0.013	0.010	0.012	0.010	0.010
	cloth	0.009	0.009	0.009	0.009	0.008	0.010	0.008	0.009	0.010	0.012
	kick	0.005	0.005	0.011	0.015	0.005	0.011	0.005	0.007	0.005	0.011
	petfinder	0.012	0.012	0.010	0.012	0.011	0.011	0.010	0.009	0.012	0.013
	wine10	0.007	0.011	0.011	0.012	0.008	0.010	0.007	0.009	0.008	0.013
	wine100	0.006	0.007	0.006	0.009	0.006	0.009	0.005	0.006	0.006	0.012
LateFuse-BERT-TaB	airbnb	0.010	0.008	0.008	0.012	0.011	0.010	0.009	0.010	0.009	0.014
	cloth	0.008	0.006	0.007	0.011	0.007	0.011	0.008	0.010	0.008	0.012
	kick	0.007	0.006	0.009	0.014	0.005	0.012	0.005	0.010	0.006	0.012
	petfinder	0.009	0.009	0.008	0.011	0.010	0.010	0.008	0.012	0.009	0.011
	wine10	0.007	0.007	0.009	0.014	0.006	0.010	0.007	0.009	0.007	0.013
	wine100	0.005	0.009	0.006	0.010	0.006	0.008	0.004	0.006	0.005	0.007
AllText-DBERT-TaB	airbnb	0.010	0.010	0.009	0.012	0.010	0.011	0.009	0.009	0.010	0.011
	cloth	0.008	0.008	0.008	0.012	0.009	0.009	0.007	0.009	0.008	0.011
	kick	0.006	0.006	0.009	0.013	0.005	0.013	0.006	0.009	0.006	0.014
	petfinder	0.010	0.011	0.010	0.010	0.010	0.007	0.013	0.013	0.010	0.009
	wine10	0.007	0.007	0.010	0.011	0.008	0.010	0.008	0.008	0.007	0.015
	wine100	0.006	0.006	0.007	0.013	0.007	0.010	0.007	0.007	0.007	0.008
LateFuse-DBERT-TaB	airbnb	0.011	0.011	0.010	0.011	0.008	0.010	0.010	0.010	0.012	0.010
	cloth	0.008	0.008	0.008	0.008	0.005	0.009	0.007	0.010	0.008	0.011
	kick	0.007	0.007	0.009	0.014	0.006	0.013	0.007	0.013	0.007	0.015
	petfinder	0.009	0.009	0.008	0.008	0.009	0.011	0.008	0.008	0.009	0.010
	wine10	0.008	0.009	0.009	0.013	0.007	0.010	0.007	0.009	0.009	0.013
	wine100	0.006	0.006	0.006	0.010	0.007	0.009	0.007	0.007	0.006	0.010
BERT-ViT	food5	0.005	0.006	0.013	0.012	0.005	0.006	0.004	0.007	0.005	0.015
DBERT-ViT	food5	0.005	0.005	0.012	0.015	0.006	0.006	0.005	0.007	0.006	0.012

Table 5: **Variability in the results** by model and dataset: Standard deviation of AUROC results, computed based on 30 bootstraps with fraction 70% from raw table results (i.e. across seeds).

(768×768) for the AllText architecture and $(2 \times 768) \times (2 \times 768)$ for architectures based on late fusion. As this is a classification task, the final output has dimension C. The cross-entropy loss is optimized with Adam (learning rate of $1e-3$) for 10 epochs and batch size of 32. For each test example, we compute the total uncertainty (predictive entropy), after averaging the predicted probabilities generated by $E = 5$ neural networks with parameters θ_e:

$$u(\mathbf{z}) = -\sum_{j \in \mathbb{Y}} \left(\frac{1}{E} \sum_{e=1}^{E} p(j|\mathbf{z}; \theta_e) \right) \log_2 \left(\frac{1}{E} \sum_{e=1}^{E} p(j|\mathbf{z}; \theta_e) \right)$$

G Variability in Results

The variability in results is presented in Table 5.

H Implementation Information

Hardware and computational cost. We run the experiments with a Tesla T4 GPU. Table 6 summarizes the average computational cost for each method. The methods that require performing several forward passes during inference (e.g. MCD) or training one or several models (e.g. DENS, EDIP) are less efficient than the other baselines.

Python libraries. The implementation is based on Python 3.10 and the following packages: torch 2.4.0+cu121, transformers 4.42.4, scikit-learn 1.3.2, scipy 1.13.1, pandas 2.1.4, numpy 1.26.4, matplotlib 3.7.1, and seaborn 0.13.1. These libraries are publicly available with "BSD", "MIT", or "Apache Software" licenses.

Dataset	AC	ACSC	CP	DC	DENS	DNN	EDIP	ENRG	MCD	TCP
airbnb	0.01	0.01	0.01	2.21	3.66	0.13	4.18	0.01	88.47	0.84
cloth	0.01	0.01	0.01	11.24	11.02	0.33	15.94	0.01	24.85	2.42
kick	0.01	0.01	0.01	4.98	16.03	0.43	17.18	0.01	12.14	3.36
petfinder	0.01	0.01	0.01	5.01	7.41	0.23	9.76	0.01	40.12	1.62
wine10	0.01	0.01	0.01	6.51	15.98	0.46	21.76	0.01	15.84	3.38
wine100	0.01	0.01	0.01	5.92	15.83	0.39	25.72	0.01	15.60	3.26
food5	0.01	0.01	0.01	0.74	2.64	0.14	1.95	0.03	112.36	0.77

Table 6: Average computation time (in seconds) computed for each method, averaged over various model architectures and random seeds.