A Re-examination of Neural Selective Prediction for Natural Language Processing

Anonymous ACL submission

Abstract

We provide a survey and careful empirical comparison of the state-of-the-art in neural selective classification for NLP tasks. Across multiple trials on multiple datasets, only one of the surveyed techniques – Monte Carlo Dropout – significantly outperforms the simple baseline of using the maximum softmax probability as an indicator of prediction confidence. Our results provide a counterpoint to recent claims made on the basis of single-trial experiments on a small number of datasets. We also provide a blueprint and open-source code to support the future evaluation of selective prediction techniques.

1 Introduction

Despite the massive improvements that deep learning has brought to natural language processing over the past decade, neural networks still do make mistakes. There has thus been a growing interest in confidence estimation techniques that perform well on deep neural networks.

A prominent subarea of confidence estimation is *selective prediction* (El-Yaniv et al., 2010; Geifman and El-Yaniv, 2017). Selective prediction focuses on developing classifiers that choose to abstain when sufficiently uncertain. There is less focus on absolute measures of confidence, and more on a classifier's ability to successful rank its predictions, enabling techniques that maximize prediction quality given a desired yield (Geifman and El-Yaniv, 2019) or that maximize yield given a desired quality (Geifman and El-Yaniv, 2017).

This paper provides a survey and rigorous empirical comparison of the state-of-the-art in *neural selective classification* (i.e. selective prediction where the underlying classifier is a neural network) specifically as it pertains to natural language processing. Our main contributions are the following:

- We survey a variety of recent techniques proposed in the ML and NLP literature.
- We compare them across **six** classification tasks from the GLUE benchmark (Wang et al., 2018), we do careful hyperparameter tuning for all surveyed techniques, and we perform multiple trials of each technique to get an adequate sense of median and variance.
- We discover and remedy a flaw in an evaluation metric proposed by (Xin et al., 2021), resulting in a simple metric called *worst-case normalized Kendall-Tau distance* that provides a calibrated measure of the performance of selective classification techniques.
- We determine that, despite various recent claims to have identified techniques that outperform the simple baseline (Hendrycks and Gimpel, 2017) of using maximum softmax probability as a confidence indicator, the only surveyed technique that demonstrates significant improvement across multiple tasks and trials is Monte Carlo Dropout (Gal and Ghahramani, 2016).
- We release a documented and unit-tested Python package called spred (**s**elective **pred**iction) to make our experiments transparent and reproducible. To facilitate evaluation of future techniques, the package provides tutorials about how to add and evaluate novel selective prediction methods.

2 Selective Prediction

2.1 Preliminaries

A *prediction function* is a function $f : \mathcal{X} \to \mathcal{Y}$ that maps an instance space $\mathcal{X}$ to a label space $\mathcal{Y}$. We refer to the output $f(x)$ of the prediction function as its *prediction* for instance $x \in \mathcal{X}$. We use the notation $\overline{f}(x)$ to refer to the gold prediction for a particular instance. The following denotes the correctly and incorrectly predicted instances of

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
$f^{(A)}(x) = \bar{f}(x)?$	✓	✗	✓	✗	✗	✗	✓	✓	✓	✓
$\tilde{g}_1^{(A)}(x)$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\tilde{g}_2^{(A)}(x)$	0.5	0.1	0.6	0.2	0.3	0.4	0.7	0.8	0.9	1.0
$\tilde{g}_3^{(A)}(x)$	0.1	0.7	0.2	0.8	0.9	1.0	0.3	0.4	0.5	0.6

Figure 1: Three confidence functions for an example prediction function that has an overall accuracy of 6/10 on the evaluation set.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
$f^{(B)}(x) = \bar{f}(x)?$	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
$\tilde{g}_1^{(B)}(x)$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\tilde{g}_2^{(B)}(x)$	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.1	0.9	1.0
$\tilde{g}_3^{(B)}(x)$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	1.0	0.8	0.9

Figure 2: Three confidence functions for a stronger prediction function that has an overall accuracy of 9/10 on the evaluation set.

prediction function f on set $\mathbf{x} \subseteq \mathcal{X}$:

$$\begin{aligned} \mathcal{C}(f, \mathbf{x}) &= \{x_i \in \mathbf{x} \mid f(x_i) = \overline{f}(x_i)\} \\ \overline{\mathcal{C}}(f, \mathbf{x}) &= \{x_i \in \mathbf{x} \mid f(x_i) \neq \overline{f}(x_i)\} \end{aligned}$$

If we pair a prediction function with a *selection function* $g : \mathcal{X} \to \{0, 1\}$, we obtain a *selective model* (f, g). For instance $x \in \mathcal{X}$, a selective model $h = (f, g)$ *publishes* its prediction $f(x)$ if $g(x) = 1$, and *abstains* if $g(x) = 0$. In short:

$$h(x) = \begin{cases} f(x) & \text{if } g(x) = 1 \\ \bot & \text{if } g(x) = 0 \end{cases}$$

where $\bot$ is a symbol representing abstention.

A convenient way to implement a selection function is to use a *confidence function* $\tilde{g} : \mathcal{X} \to \mathbb{R}$ that assigns a real-valued confidence to any input $x \in \mathcal{X}$. We can derive a selection function g_θ from confidence function $\tilde{g}$ by specifying a minimum confidence threshold θ for publishing predictions:

$$g_\theta(x) = \mathbb{1}[\tilde{g}(x) > \theta]$$

2.2 Examples

In Figure 1, we show three confidence functions $\tilde{g}_1^{(A)}$, $\tilde{g}_2^{(A)}$, $\tilde{g}_3^{(A)}$ for an example prediction function $f^{(A)}$. The first confidence function $\tilde{g}_1^{(A)}$ is pretty good; it assigns its highest confidences to four out of the six correct predictions, though unfortunately it also gives its lowest confidence to the correct prediction $f^{(A)}(x_1)$. By contrast, $\tilde{g}_2^{(A)}$ is a best-case confidence function (assigning its highest confidences to the six correct predictions) and $\tilde{g}_3^{(A)}$ is a worst-case confidence function (assigning its lowest confidences to the six correct predictions).

Figure 2 shows three more confidence functions $\tilde{g}_1^{(B)}$, $\tilde{g}_2^{(B)}$, $\tilde{g}_3^{(B)}$ for a stronger prediction function $f^{(B)}$. This time, the first confidence function $\tilde{g}_1^{(B)}$ is not particularly good; it assigns its third-highest confidence to the only incorrect prediction. Again, $\tilde{g}_2^{(B)}$ is a best-case confidence function (assigning its highest confidences to the nine correct predictions) and $\tilde{g}_3^{(B)}$ is a worst-case confidence function (assigning its lowest confidences to the nine correct predictions).

2.3 Evaluation with AUC Metrics

Typically, one evaluates the goodness of a confidence function by quantifying the trade-off between the quality and quantity of its published predictions. The prominent approaches – risk/coverage curves (El-Yaniv et al., 2010), receiver-operator (ROC) curves (Davis and Goadrich, 2006), and precision-recall curves (Hendrycks and Gimpel, 2017) – share many of the same benefits and drawbacks. In this paper, we will use precision-recall curves, mainly due to the NLP community's increased familiarity with them.

In Figure 3, we show the precision-recall curves for the six confidence functions from the previous subsection. The aspiration of any confidence function is to achieve an **A**rea **U**nder the **P**recision-**R**ecall curve (AUPR) of 1, which means that it has perfectly separated the correct and incorrect predictions of the prediction function. Among the examples, this has been achieved by confidence functions $\tilde{g}_2^{(A)}$ and $\tilde{g}_2^{(B)}$.

A drawback with AUPR (and its analogs) is that its value is not interpretable without knowledge of the goodness of the prediction function. Consider the two confidence functions $\tilde{g}_1^{(A)}$ and $\tilde{g}_1^{(B)}$. Whereas $\tilde{g}_1^{(B)}$ is worse than choosing a random confidence function, $\tilde{g}_1^{(A)}$ is considerably better. However, the AUPR of the former exceeds that of the latter. This is because AUPR conflates the goodness of the confidence function and the goodness of the prediction function.

One could imagine calibrating AUPR by taking into account the worst-case AUPR (i.e. the AUPRs for worst-case confidence functions $\tilde{g}_3^{(A)}$ and $\tilde{g}_3^{(B)}$)

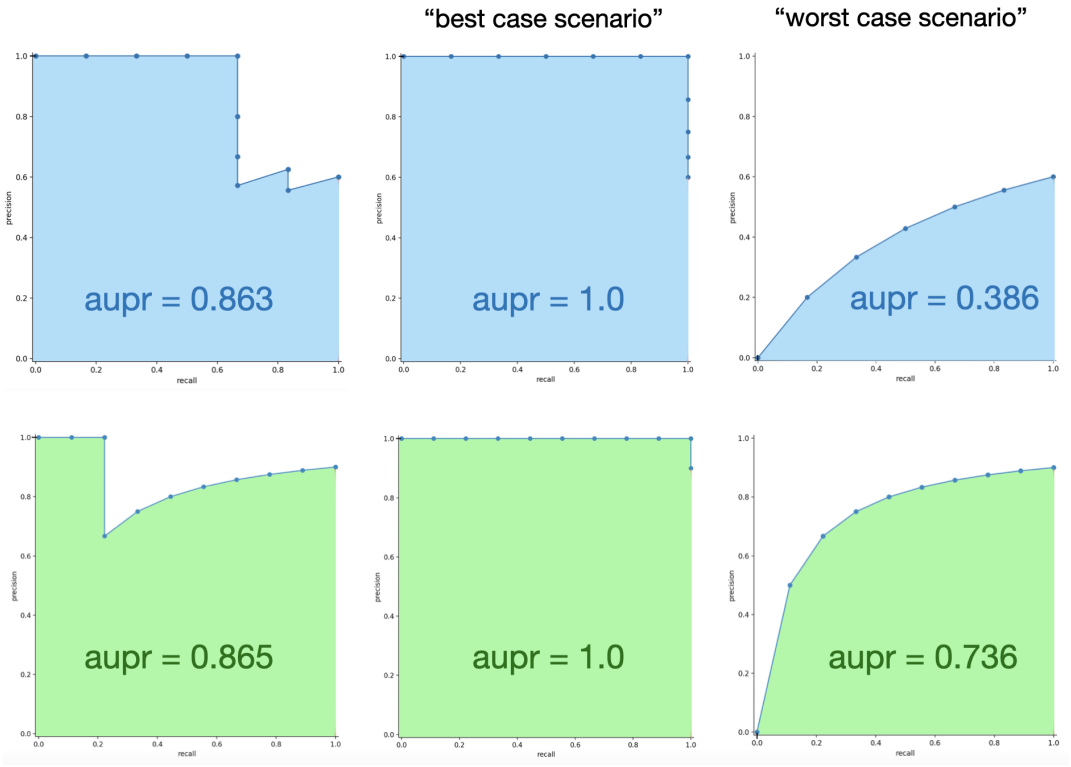

Figure 3: Precision-recall curves for confidence functions $\tilde{g}_1^{(A)}$, $\tilde{g}_2^{(A)}$, $\tilde{g}_3^{(A)}$ (top, blue) and confidence functions $\tilde{g}_1^{(B)}$, $\tilde{g}_2^{(B)}$, $\tilde{g}_3^{(B)}$ (bottom, green).

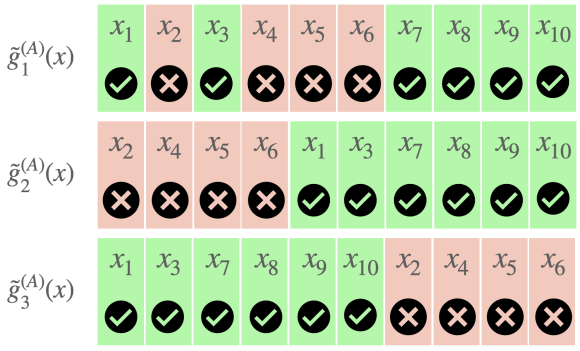

Figure 4: The predictions of Figure 1, sorted by increasing confidence.

but we will adopt an even simpler approach by amending a recent proposal by (Xin et al., 2021).

2.4 Evaluation with Kendall-Tau Distance

If we sort predictions by increasing confidence (as in Figure 4), a best-case confidence function (e.g. $\tilde{g}_2^{(A)}$) ranks all incorrect predictions below all correct predictions, while a worst-case confidence function (e.g. $\tilde{g}_3^{(A)}$) ranks all *correct* predictions below all *incorrect* predictions. Observing this, (Xin et al., 2021) proposed a rank-based evaluation metric for selective prediction called **R**eversed **P**air **P**roportion (RPP), which is a normalized count of pairwise ranking errors. Although they do not make this connection in their paper, RPP is a normalized version of Kendall-Tau distance (Kendall, 1948):

$$\tau_{dist}(\tilde{g}; f, \mathbf{x}) = \sum_{\substack{x_i \in \mathcal{C}(f,\mathbf{x}) \\ x_j \in \overline{\mathcal{C}}(f,\mathbf{x})}} \mathbb{1}[\tilde{g}(x_i) < \tilde{g}(x_j)]$$

$$\text{RPP} = \frac{\tau_{dist}(\tilde{g}; f, \mathbf{x})}{|\mathbf{x}|^2}$$

For instance, confidence function $\tilde{g}_1^{(A)}$ has a τ_{dist} of 7 (it ranks correct instance x_1 below 4 incorrect predictions, and instance x_3 below 3 incorrect predictions) and an RPP of $\frac{7}{100}$. For the best-case confidence function $\tilde{g}_2^{(A)}$, $\tau_{dist} = \text{RPP} = 0$.

Unfortunately, choosing $|\mathbf{x}|^2$ as their denominator means that RPP suffers the same problem as AUPR: its value cannot be interpreted[1] independently of the goodness of the prediction function. Consider the RPP for our "worse-than-random" confidence function $\tilde{g}_1^{(B)}$. Like $\tilde{g}_1^{(A)}$, it has a τ_{dist} of 7 (it ranks incorrect instance x_8 above 7 correct

[1](Xin et al., 2021) seem unaware of this issue, directly comparing the RPP of confidence functions for different prediction functions, leading to some unjustified conclusions.

Figure 5: The predictions of $\tilde{g}_1^{(B)}$, sorted by increasing confidence. Both τ_{dist} and RPP for $\tilde{g}_1^{(A)}$ and $\tilde{g}_1^{(B)}$ are equivalent.

predictions) and thus an RPP of $\frac{7}{100}$. Even though $\tilde{g}_1^{(A)}$ is better than random and $\tilde{g}_1^{(B)}$ is worse than random, they end up with the same RPP.

Fortunately, there is a simple remedy. One possibility is to use alternative ranking statistics that account for ties in the two lists[2] we are comparing. Examples of these alternative statistics include Kendall-Tau-b and Kendall-Tau-c. However, these are a bit heavyweight for our purposes here. All we really need to do is normalize by the worst-case Kendall-Tau distance, which is not $|\mathbf{x}|^2$, but rather $c(|\mathbf{x}| - c)$, where c is the number of correct predictions made by the prediction function. This gives us a measurement we will refer to as *worst-case normalized Kendall-Tau distance*:

$$\tau_{wcn}(\tilde{g}; f, \mathbf{x}) = \frac{\tau_{dist}(\tilde{g}; f, \mathbf{x})}{c(|\mathbf{x}| - c)} \quad (1)$$

where $c = |\mathcal{C}(f, \mathbf{x})|$. Worst-case normalized Kendall-Tau distance has the following attractive properties:

- For a perfect confidence function $\tilde{g}$, $\tau_{wcn}(\tilde{g}; f, \mathbf{x}) = 0$.
- For a worst-case confidence function $\tilde{g}$, $\tau_{wcn}(\tilde{g}; f, \mathbf{x}) = 1$.
- For a random confidence function $\tilde{g}$, the expected value of $\tau_{wcn}(\tilde{g}; f, \mathbf{x})$ is 0.5.

Unlike RPP and the various area under the curve metrics, τ_{wcn} directly assesses the quality of the confidence function, and its value is interpretable without knowing the quality of the associated prediction function.

3 Surveyed Techniques

Our main goal in this paper is a reproducible and rigorous comparison of a broad range of selective prediction techniques on NLP tasks. In this section, we describe the techniques we compare.

[2]In our case, the two lists are the list of confidences and the 0-1 list of prediction correctness. The second of these, having only zeroes and ones, has lots of ties.

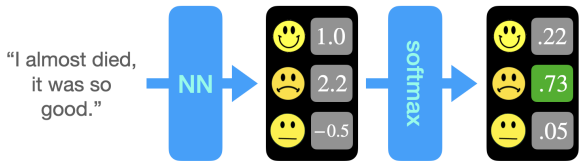

Figure 6: MAXPROB is the simplest confidence function. After applying softmax to the neural network output, it uses the maximum probability of the resulting distribution as its measure of confidence.

3.1 Confidence Functions

The following are ways to create a confidence function for an already trained neural prediction function.

MaxProb

For neural prediction functions, the simplest-to-implement confidence function is likely MAXPROB, pictured in Figure 6 for a three-way sentiment analysis task. After applying softmax to the neural network output, MAXPROB (sometimes known as SOFTMAXRESPONSE) uses the maximum probability of the resulting distribution as its measure of confidence. The surprising effectiveness of such a simple approach was observed by (Hendrycks and Gimpel, 2017), among others, although more recent papers have claimed to have made significant improvements over MAXPROB with more involved techniques.

Monte Carlo Dropout

(Gal and Ghahramani, 2016) proposed leveraging dropout (Srivastava et al., 2014) to assess the uncertainty of a neural network on a particular instance. As usual, dropout is disabled at test time to make the prediction. But then the input instance is redecoded k times with dropout enabled. This yields k samples for the softmax probability of the prediction. There are two common methods (Kamath et al., 2020) for synthesizing these k samples into a confidence measure: either we take the mean (Lakshminarayanan et al., 2017) of the samples (a strategy we refer to as MCDM), or the negative[3] variance (Feinman et al., 2017; Smith and Gal, 2018) of the samples (a strategy we refer to as MCDV).

[3]We use the *negative* variance so that a greater value indicates a greater confidence.

Trustscore

(Jiang et al., 2018) advocated a nearest-neighbor-based confidence function. First, the training instances are converted[4] into vector encodings, and grouped according to their gold labels. Outliers are then filtered from each labeled group. Specifically, they sort the vectors (i.e. points in $\mathbb{R}^d$ space) by the radius of the minimal ball centered at that vector that contains k points from their labeled group. The percentage $\alpha \in [0, 1]$ of points with the largest such radii (i.e. the outliers) are removed. This filtered set[5] is called an *α-high density set*.

The confidence assigned to an instance prediction, called TRUSTSCORE, is the ratio of (a) the distance between the instance's vector encoding and the closest α-high density set of a *non-predicted* label, (b) the distance between the instance's vector encoding and the α-high density set of the *predicted* label.

3.2 Specialized Loss Functions

We also survey techniques that simultaneously train a prediction function and an associated confidence function.

Error Regularization

(Xin et al., 2021) suggests adding an "error regularization" term to the task's loss function $\mathcal{L}$ that directly penalizes ranking errors made by the confidence function $\tilde{g}$:

$$\epsilon(f, \mathbf{x}) = \sum_{\substack{x_i \in \mathcal{C}(f,\mathbf{x}) \\ x_j \in \overline{\mathcal{C}}(f,\mathbf{x})}} \text{RELU}(\tilde{g}(x_i) - \tilde{g}(x_j))^2$$

$$\mathcal{L}_{ereg}(f, \mathbf{x}) = \mathcal{L}(f, \mathbf{x}) + \lambda \cdot \sum_{\mathbf{b} \in \text{batches}(\mathbf{x})} \epsilon(f, \mathbf{b})$$

where $\lambda \in \mathbb{R}^+$ is a tunable hyperparameter and $\text{batches}(\mathbf{x})$ is the set of minibatches of training set $\mathbf{x}$.

At training time, (Xin et al., 2021) uses MAXPROB for the confidence function $\tilde{g}$, though at test time, they additionally experiment with MCDM and MCDV.

Deep Abstaining Classifiers

A Deep Abstaining Classifier (Thulasidasan et al., 2019), abbreviated DAC, explicitly introduces an extra abstention output $\perp$ to the neural network, and trains with a loss function that allows the prediction function to gain benefit from abstaining on difficult instances:

$$(1 - p_\perp)\mathcal{L}(f, \mathbf{x}) + \alpha \log \frac{1}{1 - p_\perp} \qquad (2)$$

where $p_\perp$ is the probability according to abstention output $\perp$ after applying softmax, $\mathcal{L}(f, \mathbf{x})$ is standard cross-entropy loss over the non-abstention outputs, and α is a real-valued weight that is zero for the first k (warmup) epochs of training, and is linearly scaled from α_{min} to α_{max} during the remaining epochs. The initial value α_{min} is set to be a fixed fraction $\frac{1}{\rho}$ of a moving average of the loss during the warmup epochs. The authors provide code that we use in our experiments.

At test time, MAXPROB is used[6] as the confidence function, though with a slight modification – if the probability associated with the abstention label is the maximum softmax probability, then the next highest probability is used as the confidence.

4 Experiment Design

To draw reliable conclusions on a sufficiently varied set of NLP tasks, we evaluated the techniques on six classification[7] tasks of the GLUE benchmark (Wang et al., 2018): COLA, MNLI, MRPC, QNLI, RTE, and SST-2.

Bearing in mind that our goal is to compare selective classification techniques, not to produce state-of-the-art prediction functions, we randomly partitioned each training set into two halves, using GLUETRAIN-A for training and GLUETRAIN-B for early stopping and hyperparameter tuning. Since the gold labels for GLUE test sets are not all publicly available, we reserved the development set (GLUEDEV) of each task for final evaluation.

We trained the prediction function by fine-tuning BERT-BASE-CASED using the `transformers` package (Wolf et al., 2020), mostly using the training parameters recommended by its `run_glue.py` script (the sole deviation is that we run each training for 6 epochs, rather than 3).

[4] They are agnostic about how best to do so. We will return to this issue.

[5] They fix $k = 10$, but treat α as a tunable hyperparameter.

[6] We also experimented with using $1 - p_\perp$ (i.e. the total probability mass accorded to non-abstention outputs) as the confidence, but this yielded poor results.

[7] We did not include WNLI because the training set was too small to train a prediction function that does better than random guessing. We did not include QQP because we had training difficulties that we could not resolve before the submission deadline. STS-B is a regression task, not a classification task. For evaluating MNLI, we used matched accuracy, since the focus of this paper is not on domain shift.

For the techniques that required specialized loss functions, we substituted the default BERT loss function with the alternative specified by the selective prediction technique.

4.1 Hyperparameter Tuning

In an effort to fairly evaluate each technique, we began with smaller-scale experiments to determine an appropriate setting of a technique's hyperparameters for the GLUE tasks. For these experiments, we used GLUETRAIN-A for training and GLUETRAIN-B for validation. We selected three GLUE tasks of various sizes and genres (one single-sentence task, one similarity-and-paraphrase task, and one inference task) as proxies: SST-2, MRPC, and RTE. We ran 5 trials[8] for each hyperparameter setting.

Monte Carlo Dropout

Monte Carlo Dropout has a single hyperparameter k: the number of decodings of the training instance with dropout enabled. We experimented with $k \in \{10, 30, 50\}$. We found little discernible difference (see Figure 10) between $k = 30$ and $k = 50$. Slightly better results with $k = 30$ versus $k = 10$ convinced us to use $k = 30$ for further experiments.

TrustScore

To use TRUSTSCORE, we need to encode each instance as a vector. Following common practice, we used BERT's final layer encoding (after finetuning) of the [CLS] token.

To select the hyperparameter settings for TRUSTSCORE, we followed (Jiang et al., 2018) and experimented with several powers of two for hyperparameter α, specifically $\alpha \in \{0.5, 0.25, 0.125\}$. Also, since TRUSTSCORE is too slow in practice to run on large training sets, we sample N training instances (without replacement) prior to running the TrustScore algorithm. In our tuning experiments, we tried the values $N \in \{800, 1600\}$.

We found little difference between the six hyperparameter settings (see Figure 11) and set $N = 800$ and $\alpha = 0.25$ for further experiments.

Error Regularization

Error Regularization has hyperparameter λ (the multiplier for the regularization term). Following the appendix of (Xin et al., 2021), we experimented with $\lambda \in \{0.01, 0.05, 0.1, 0.5\}$. Because Error Regularization uses an alternative loss function that can potentially affect the overall quality of the prediction function, we used AUPR (which blends the quality of the prediction function with the quality of the selection function) as our main evaluation metric. We found high variance between trials, and selected $\lambda = 0.05$ (with the most consistent performance) for further experiments.

Deep Abstaining Classifier

One of the virtues of the Deep Abstaining Classifier is that it automatically adjusts its weights according to the cross-entropy loss observed during the warmup epochs, but it still has hyperparameters ρ and α_{max} to determine precisely how this is done. In the code accompanying (Thulasidasan et al., 2019), the default settings are $\rho = 64$ and $\alpha_{max} = 1.0$. Given these defaults, we experimented with $\rho \in \{32, 64, 128\}$ and $\alpha_{max} \in \{0.5, 1.0, 2.0\}$. The technique did not appear to be particularly sensitive to the choice of hyperparameters (see Figure 13) and so we kept the default settings for further experiments. We used two warmup epochs (sufficient to reach decent baseline accuracy for all GLUE tasks), and accordingly increased the total number of training epochs from 6 to 8.

5 Results

For final evaluation, we ran ten experiment trials on the six GLUE tasks. Specifically, we trained ten prediction functions with different random seeds for each loss function: the basic BERT loss ("basic"), BERT loss with error regularization ("ereg"), and the Deep Abstaining Classifier loss ("dac"). For each resulting prediction function, we evaluated the various confidence functions. In all experiment trials, we used the hyperparameter settings established in Section 4.

Figure 7 uses a violin plot[9] to visualize the results[10] for two GLUE datasets (MRPC and SST-2). Each "string" of the violin corresponds to the τ_{wcn} of a single trial on GLUEDEV, while the "body" of the violin is a kernel density estimation of the result distribution. We include a random baseline,

[8] More detailed results from these experiments are provided in the appendix.

[9] We used the `seaborn` package to create the plots: https://seaborn.pydata.org/generated/seaborn.violinplot.html.

[10] For brevity, we omit certain loss/confidence pairs, for instance `ereg(mcdm)` and `ereg(mcdv)`, from the reported results. In our experiments, the improvement provided by the MC Dropout techniques provided similar improvement for all loss functions.

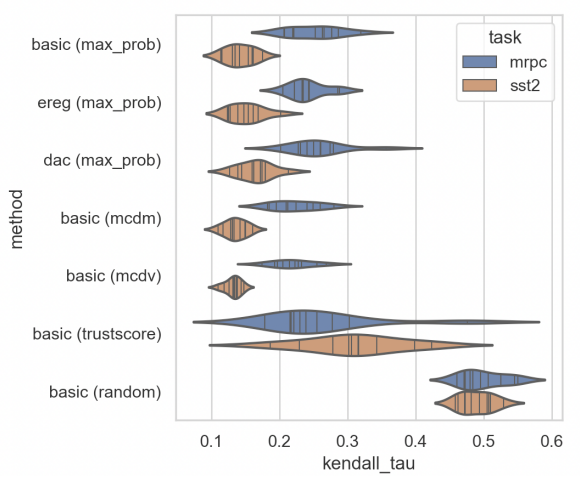

Figure 7: Results of several selective prediction techniques on two GLUE datasets. Each "string" of the violin corresponds to the τ_{wcn} of a single trial on GLUEDEV, while the "body" of the violin is a kernel density estimation of the result distribution.

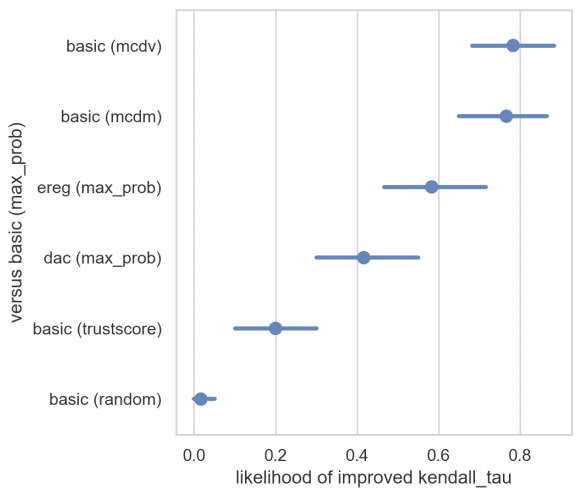

Figure 8: Estimate of the likelihood $E(X_{t,\tau_{wcn}})$ that technique t outperforms the basic MAXPROB baseline according to the τ_{wcn} metric. The bars show a 95% confidence interval for this estimate.

which assigns a random confidence to each prediction. This provides empirical validation that the expected value of τ_{wcn} for a random confidence function is 0.5, and also gives a sense of the experimental variance for a particular dataset.

Figure 7 indicates that all techniques have considerable variation from trial to trial, and suggests that it would be easy to draw incorrect conclusions from a single-trial study. Beyond this, it is difficult to eyeball the results and make an informed decision about which technique to use. One can possibly dismiss TRUSTSCORE (or our particular implementation of it) based on Figure 8, but what should we make of the advantages that MCDM and MCDV seem to offer over the basic MAXPROB approach? The MC Dropout techniques are considerably more expensive to run (since they require multiple independent decodings). Are they meaningfully better than MAXPROB?

Let's quantify the phrase "meaningfully better" by estimating the likelihood that a candidate technique outperforms the basic MAXPROB baseline. For a candidate technique t and performance metric m, define random variable $X_{t,m}$ as the result of the following trial:

- Choose a random task from a probability distribution P_{task} over tasks.
- Execute the candidate selective prediction technique t and the baseline technique (i.e. MAXPROB) and evaluate each using metric m (e.g. τ_{wcn} or AUPR).
- If the candidate technique t outperforms the baseline according to metric m, return 1. Otherwise, return 0.

The expected value $E(X_{t,m})$ tells us the likelihood that technique t will outperform the baseline according to metric m . Since we performed 10 trials for each of 6 GLUE tasks, we therefore have 60 samples[11] for estimating $E(X_{t,m})$. Figure 8 shows the $E(X_{t,\tau_{wcn}})$ estimate for the techniques from Figure 7, along with a 95% confidence interval. Somewhere between 62% to 85% of the time (with 95% confidence), both MCDM and MCDV improve upon the MAXPROB baseline according to the τ_{wcn} metric.

None of the other techniques provide significant advantage over the basic MAXPROB baseline. Moreover, there is a further complication. Worst-case normalized Kendall-tau distance specifically focuses on the quality of the confidence function. Modifications to the basic loss function (e.g. error-regularization or DAC loss) might improve the efficiency of the confidence function while simultaneously sacrificing the quality of the prediction function. To check the extent to which this occurs, we should also evaluate the performance of our techniques using AUPR.

Figure 9 shows the $E(X_{t,\text{AUPR}})$ estimate, along

[11] In this case, the task distribution P_{task} is a uniform distribution over 6 GLUE tasks. Whether this is an effective proxy for NLP tasks in general is a legitimate question, but the NLP community does seem to have adopted GLUE as an important benchmark.

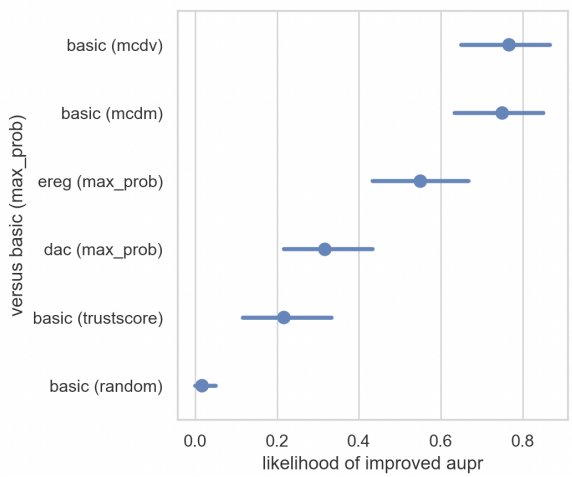

Figure 9: Estimate of the likelihood $E(X_{t,\text{AUPR}})$ that technique t outperforms the basic MAXPROB baseline according to the AUPR metric. The bars show a 95% confidence interval for this estimate.

with a 95% confidence interval. In particular, the results for DAC loss are noticeably worse from an AUPR perspective.

6 Related Work

Selective prediction has a long tradition in machine learning, dating back to the 1950s (Chow, 1957). There is an extensive literature (Hellman, 1970; Fumera and Roli, 2002; Cortes et al., 2016) on training classifiers with the ability to abstain (also known as the "reject option"), usually specific to alternative classifiers like support vector machines.

There is also a significant literature (Platt et al., 1999; Guo et al., 2017; Kumar et al., 2018; Wang et al., 2020; Desai and Durrett, 2020) on the topic of *calibration*, i.e. the development of probabilistically interpretable confidence measures. In this paper, we restrict our focus to the relative rankings of selective predictors, and not the confidence values themselves.

While our survey focuses on techniques designed to identify ambiguous instances in the evaluation set (and, for certain techniques, to also ignore label noise in the training set), there is also interest in selective prediction techniques that operate successfully under domain shift (Kamath et al., 2020; Liu et al., 2020), i.e. when the distribution of evaluation instances differs from the training instances. Evaluation of such techniques is beyond the scope of the work described here, but we have plans to expand the spred package to evaluate selective prediction under domain shift.

7 Conclusion

With this effort, we have tried to write a paper that we would like to see more of in the NLP literature: a careful survey and empirical comparison of a diverse selection of recent techniques on a broad set of tasks. We have purposefully avoided introducing new techniques to avoid "having a horse in the race," instead focusing on doing our best to optimize each evaluated technique and provide a fair comparison. As a companion to the paper, the documented and unit-tested Python package spred affords the following benefits:

1. **reproducibility:** JSON configurations for each experiment performed[12] are provided with the library, as well as instructions for replicating them.

2. **transparency:** The documented code and unit tests can be easily inspected to independently confirm the accuracy of our implementations.

3. **extensibility:** We designed the code to make it simple to add new techniques and tasks. We provide tutorials[13] demonstrating how to do so. These are intended both for selective prediction techniques that we have invariably overlooked in our survey, as well as novel contributions to the literature.

Ever since (Hendrycks and Gimpel, 2017) identified MAXPROB as a strong baseline for selective prediction, many papers have proposed techniques that reportedly improve upon it. By and large, these papers reported improvements based on single-trial experiments on a small selection of datasets. In our more comprehensive study, the only technique that demonstrated significant improvement over MAXPROB was Monte Carlo Dropout (Gal and Ghahramani, 2016). Our results should make it clear that, at least in the realm of selective prediction, there is significant variance between tasks, and between trials of the same experiment. Thus, this paper suggests caution in drawing conclusions from single-trial experiments, especially when one may be subconsciously invested in a particular technique.

[12] With the submission, we also provide a `CSV` file containing the final experiment results.

[13] Reviewers are invited to try the tutorials to learn more about the spred package. See the `README.md` file in the `ZIP` file provided with the submission for more details.

References

Chi-Keung Chow. 1957. An optimum character recognition system using decision functions. *IRE Transactions on Electronic Computers*, (4):247–254.

Corinna Cortes, Giulia DeSalvo, and Mehryar Mohri. 2016. Learning with rejection. In *International Conference on Algorithmic Learning Theory*, pages 67–82. Springer.

Jesse Davis and Mark Goadrich. 2006. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 233–240, New York, NY, USA. Association for Computing Machinery.

Shrey Desai and Greg Durrett. 2020. Calibration of pre-trained transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 295–302, Online. Association for Computational Linguistics.

Ran El-Yaniv et al. 2010. On the foundations of noise-free selective classification. *Journal of Machine Learning Research*, 11(5).

Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. 2017. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*.

Giorgio Fumera and Fabio Roli. 2002. Support vector machines with embedded reject option. In *International Workshop on Support Vector Machines*, pages 68–82. Springer.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR.

Yonatan Geifman and Ran El-Yaniv. 2017. Selective classification for deep neural networks. *Advances in Neural Information Processing Systems*, 30:4878–4887.

Yonatan Geifman and Ran El-Yaniv. 2019. Selectivenet: A deep neural network with an integrated reject option. In *International Conference on Machine Learning*, pages 2151–2159. PMLR.

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

Martin E Hellman. 1970. The nearest neighbor classification rule with a reject option. *IEEE Transactions on Systems Science and Cybernetics*, 6(3):179–185.

Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Heinrich Jiang, Been Kim, Melody Y Guan, and Maya R Gupta. 2018. To trust or not to trust a classifier. In *NeurIPS*.

Amita Kamath, Robin Jia, and Percy Liang. 2020. Selective question answering under domain shift. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5684–5696, Online. Association for Computational Linguistics.

Maurice George Kendall. 1948. Rank correlation methods.

Aviral Kumar, Sunita Sarawagi, and Ujjwal Jain. 2018. Trainable calibration measures for neural networks from kernel mean embeddings. In *International Conference on Machine Learning*, pages 2805–2814. PMLR.

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

Weitang Liu, Xiaoyun Wang, John D Owens, and Yixuan Li. 2020. Energy-based out-of-distribution detection. *arXiv preprint arXiv:2010.03759*.

John Platt et al. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74.

Lewis Smith and Yarin Gal. 2018. Understanding measures of uncertainty for adversarial example detection. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 560–569. AUAI Press.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Sunil Thulasidasan, Tanmoy Bhattacharya, Jeff Bilmes, Gopinath Chennupati, and Jamal Mohd-Yusof. 2019. Combating label noise in deep learning using abstention. In *International Conference on Machine Learning*, pages 6234–6243. PMLR.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Shuo Wang, Zhaopeng Tu, Shuming Shi, and Yang Liu. 2020. On the inference calibration of neural machine

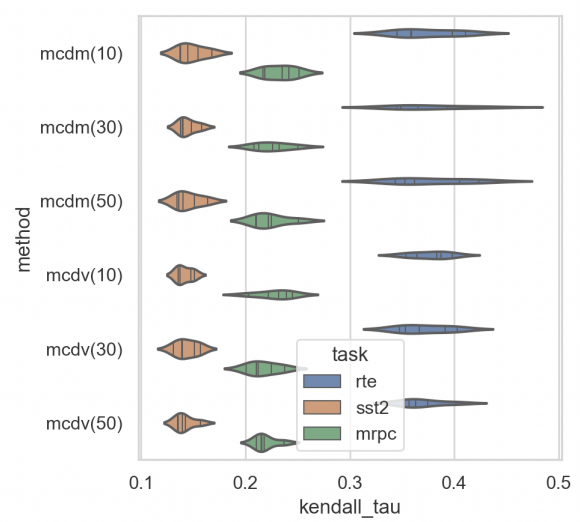

Figure 10: Results of the hyperparameter tuning experiments for Monte Carlo Dropout.

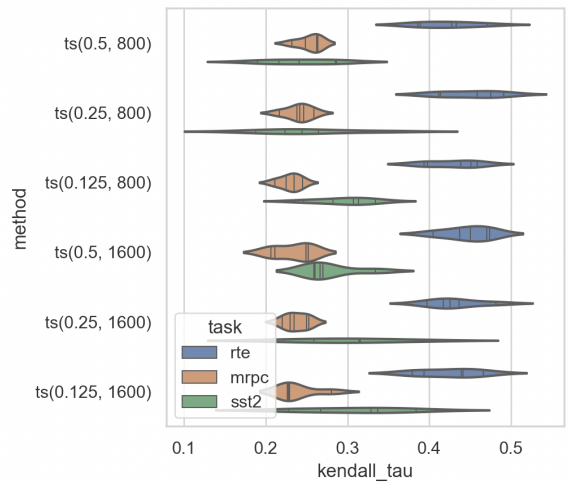

Figure 11: Results of the hyperparameter tuning experiments for TrustScore.

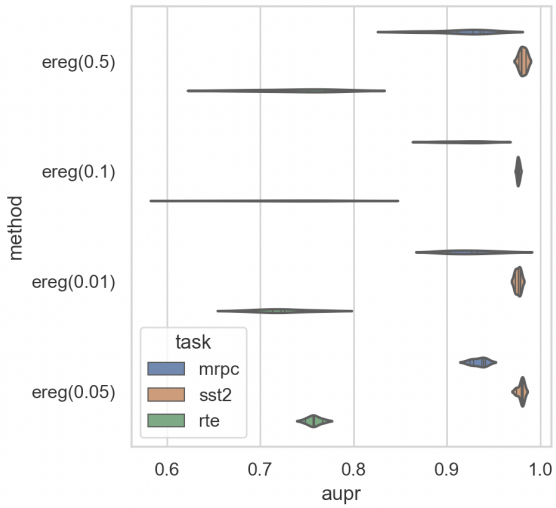

Figure 12: Results of the hyperparameter tuning experiments for Error Regularization.

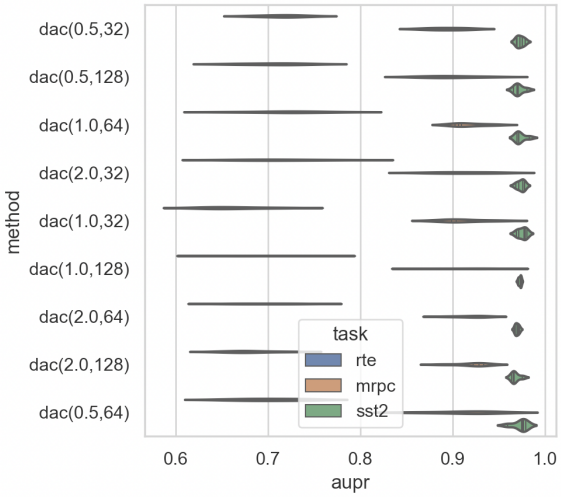

Figure 13: Results of the hyperparameter tuning experiments for the Deep Abstaining Classifier.

translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3070–3079, Online. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Ji Xin, Raphael Tang, Yaoliang Yu, and Jimmy Lin. 2021. The art of abstention: Selective prediction and error regularization for natural language processing. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1040–1051, Online. Association for Computational Linguistics.

A Hyperparameter Tuning Results

Figure 10, Figure 11, Figure 12, and Figure 13 show the experimental results for our hyperparameter tuning experiments. As with the final results, we visualize these using violin plots – each "string" of the violin corresponds to the result of a single trial.

B Machine Architecture and Running Time

The experiments were run on a single workstation with the following specifications:

- **Operating System:** Ubuntu 20.04
- **Processor:** AMD Threadripper 3990X: 64 cores, 2.90 GHz, 256 MB cache
- **GPUs:** 2x RTX 3090
- **Memory:** 256 GB
- **Operating System Drive:** 2 TB SSD (NVMe)
- **Data Drive:** 2 TB SSD (SATA)

To give the reader a sense of the relative cost of running each technique, we provide a representative result of a single trial on the above machine for the RTE task:

- **Training time for basic BERT loss:** 201s
- **Training time for BERT loss + error regularization:** 200s
- **Training time for DAC loss:** 266s
- **Evaluation time for MAXPROB:** 1.72s
- **Evaluation time for MCDM/MCDV:** 56s
- **Evaluation time for TRUSTSCORE:** 40s