Hard and Soft Evaluation of NLP models with BOOtSTrap SAmpling - BooStSa

Tommaso Fornaciari
Università Bocconi
fornaciari@unibocconi.it

Alexandra Uma
Queen Mary University
a.n.uma@qmul.ac.uk

Massimo Poesio
Queen Mary University
m.poesio@qmul.ac.uk

Dirk Hovy
Università Bocconi
dirk.hovy@unibocconi.it

Abstract

Natural Language Processing (NLP)'s applied nature makes it necessary to select the most effective and robust models. However, just producing slightly higher performance is insufficient; we want to know whether this advantage will carry over to other data sets. Bootstrapped significance tests can indicate that ability. Computing the significance of performance differences has many levels of complexity, though. It can be tedious, especially when the experimental design has many conditions to compare and several runs of experiments. We present BooStSa, a tool that makes it easy to compute significance levels with the BOOtSTrap SAmpling procedure. BooStSa can evaluate models that predict not only standard hard labels but soft labels (i.e., probability distributions over different classes) as well.

1 Introduction

Text classification is one of the main applications of NLP, with hundreds of papers published every year at NLP conferences. While these publications cover various domains, they essentially follow the same steps: *Pick a classification problem. Identify baseline models, standard models from previous literature, and State-Of-The-Art (SOTA) models. Propose a novel approach to the problem. Show that it outperforms the previous ones on benchmark data sets.*

Developing better methods for a task is a common feature of the computational linguistics literature, and selecting the best model from a range of options is crucial for the whole experimental procedure. However, showing absolute improvements on several data sets (let alone one) is not sufficient. Variations in model initialization, batch sampling and other factors might result in an improvement that does not generalize. We can use significance tests to assess whether the observed improvements are likely to hold on future data sets.

However, identifying the best method(s) depends on two different but correlated aspects of the evaluation process. The *metrics* adopted for the performance measurement and the *significance test* carried out for the models' comparison.

To compute the significance of the models' improvements over comparison sets, BooStSa relies on bootstrap sampling (Efron and Tibshirani, 1994; Berg-Kirkpatrick et al., 2012). As Søgaard et al. (2014) discussed, the effect size, that is, the performance gap between different methods, can be modelled as a random variable. When this random variable follows a normal distribution, it is possible to use Student's t-test to estimate the significance in the performance difference. However, the assumption of normal distribution does not hold in most NLP applications. Therefore, randomized, sample-based, non-parametric tests such as bootstrap sampling are better suited for NLP.

However, the correct implementation of this method can be non-trivial (especially on top of other experiments). To allow for a safe, flexible application, we release BooStSa to the community. Since the results of bootstrap methods are "very sensitive to sample size, [...] as well as to the existence of multiple metrics" (Søgaard et al., 2014, p. 1), BooStSa incorporates some constraints in the hyper-parameter choice (Section 2.3) to prevent the accidental misuse of the methods.

Concerning the *metrics*, the scenario differs when the prediction uses hard labels or soft labels. Hard labels are standard one-hot encoded labels, where one class is correct and is assigned the value of 1, and the others are wrong and have a value of 0. In the case of soft labels, the actual label value is ultimately uncertain. This uncertainty is expressed as a probability distribution over classes, each receiving a value from 0 to 1, all summing up to 1. In the first case, the use and the interpretation of metrics such as accuracy (*Acc*), precision (*Prec*), recall (*Rec*), and F-measure (*F1*) are well-understood

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
System Demonstrations, pages 127 - 134

in the literature (Forman et al., 2003; Uma et al., 2021). BooStSa computes these metrics for hard labels. In the second case, no metric is generally accepted to evaluate the divergence between probability distributions. BooStSa follows the approach of Uma et al. (2021) and provides cross entropy (*CE*), Jensen-Shannon divergence (*JSD*), entropy similarity (*E-Sim*) and entropy correlation (*E-Corr*) for soft labels.

Contributions. We release BooStSa, an open-source application that computes:

- Standard metrics for hard labels, macro-averaged or over selected target classes;
- Metrics for soft labels, following best practices form previous literature;
- The bootstrap sampling significance test, with safety-constraints for hyper-parameter choices.

The tests can be run efficiently, even for complex experimental designs comparing many different models trained in several runs of experiments. The package can be installed with `pip` and is released on github/fornaciari/boostsa (documentation at boostsa.readthedocs.io).

2 Methods

2.1 Metrics for hard labels

We adopt the standard metrics for classification tasks – F-measure (*F1*), precision (*Prec*), recall (*Rec*), and accuracy (*Acc*). They have been widely studied in literature and their interpretation is generally shared and accepted (Goutte and Gaussier, 2005; Forman et al., 2003).

2.2 Metrics for soft labels

For the soft label evaluation, we consider the four metrics proposed by Uma et al. (2021): 1) cross entropy (*CE*), 2) Jensen-Shannon divergence (*JSD*), 3) entropy similarity (*E-Sim*) and 4) entropy correlation (*E-Corr*). The first two measure the divergence between target and predicted probability distribution (*CE* and *JSD*), the second two evaluate how well the predicted distributions capture the uncertainty embodied in the target distribution, usually generated by humans (*E-Sim* and *E-Corr*).

Cross-entropy. Cross-entropy is a widely used loss-function to measure the divergence between probability distribution. Peterson et al. (2019) also suggest using it to measure the confidence of trained models with respect to target distributions, typically resulting from human predictions.

Jensen-Shannon divergence. Based on the Kullback-Leibler divergence (Kullback and Leibler, 1951) and proposed by Lin (1991), the *JSD* is another common measure of divergence from probability distributions. Compared to the Kullback-Leibler divergence, it is symmetric, making *JSD* more easily interpretable.

Entropy similarity. Proposed by Uma et al. (2021), the *E-Sim* is given by the cosine similarity between two vectors, containing the normalized entropy of the probability distribution of each data point, target or prediction.

Entropy correlation. It relies on Pearson's correlation (Pearson, 1896), applied to the same vectors used for the *E-Sim* (Uma et al., 2021). Based on the probability distributions' entropy, *E-Sim* and *E-Corr* measure how well the predicted distributions detect the uncertainty of the target distributions.

2.3 Bootstrap sampling

We follow the algorithm as described by Berg-Kirkpatrick et al. (2012). The bootstrap sampling procedure relies on the iterative simulation of "new" data sets. Random test sets are repeatedly sub-sampled (with replacement) from the whole test data. At each iteration, we compare the performance difference between baseline and experimental model, δ_{sample}, computed on the original test set, to the difference computed on the sampled subset, $\delta_{sub-sample}$. By counting how many times the sub-sample differences are at least twice as large as the overall difference value, $\delta_{sub-sample} > 2\delta_{sample}$, we can derive the p-value by dividing this count by the number of iterations.

Bootstrap sampling is agnostic to the chosen performance metric, making the tool very versatile. It allows, for example, to use all the metrics described in the previous sections, as long as they follow the Central Limit Theorem (CLT) (Pólya, 1920; Rosenblatt, 1956): i.e., it is required that the distribution of sample means approximates a normal distribution as the sample size becomes wider, regardless of the population's distribution.

However, bootstrap sampling is still sensitive to some experimental parameters, that is, the *overall test set size*, the *sub-sampled test set size* and the *number of iterations*. This point is critical, as the notion of significance is not grey-scaled: a p-value

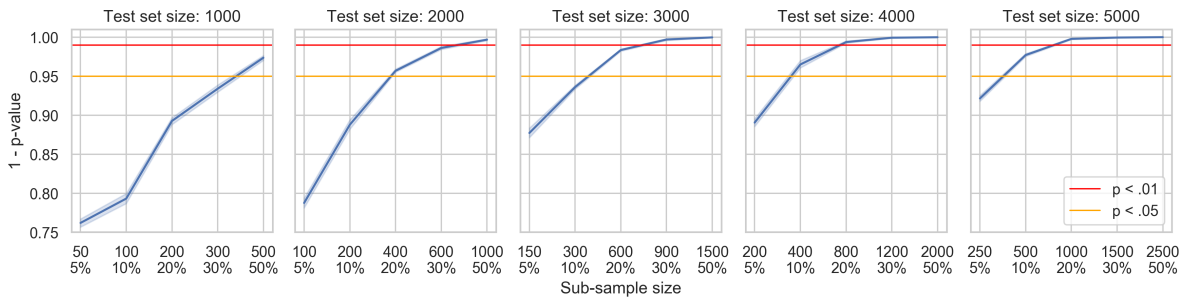

Figure 1: Significance test simulation for classification, showing the effect of interaction between test set size and sub-sample sizes. The test set is perfectly balanced (50% instances for each class). The baseline and model predictions are also balanced ($F1 = Prec = Rec = Acc$), and the difference is one percentage point, $\delta = .01$. Each data point comes from 10 simulations. Every significance test simulation was carried out with 1 000 iterations. The narrow confidence interval (light blue) across the 10 simulations, indicates that 1 000 iterations are sufficient to obtain stable results.

is either significant or not with respect to a chosen threshold, but its value is also affected by the bootstrap hyper-parameters. Their effect can be summarized as follows.

Test set size. The wider the sample, the more robust the results, and the easier to reach significance levels.

If several experiments are run for each experimental condition, BooStSa concatenates the test data, the baseline and experimental predictions as if each of them was a single experiment.

Sub-sample size. The previous hyper-parameter is usually determined by the availability of the data sets, limiting the possibilities of control for the researcher. However, the size of the sub-samples can freely be chosen by the experimenter. Determining its correct value is not trivial, though.

Søgaard et al. (2014, p. 3) observe that "for the bootstrap test to work, the original sample has to capture most of the variation in the population. If the sample is very small, though, this is likely not the case. Consequently, with small sample sizes, there is a risk that the calculated p-value will be artificially low–simply because the bootstrap samples are too similar".

The opposite risk exists as well: if the sub-sample is too similar to the size of the whole test set, the bootstrap samples will be too similar to the test set, producing p-values that are artificially high. In other words, too broad a sample implies that the test set sample actually represents the whole population, which is quite a strong assumption indeed.

Figure 1 shows a simulation of the relationship between test size and sub-sample size. The curves come from an artificially created test set and perfectly balanced predictions, where the experimental model beats the baseline by one point per cent on every standard classification metric (*F1*, *Prec*, *Rec* and *Acc*). The trends are similar to those shown by Berg-Kirkpatrick et al. (2012) on real data.

To the best of our knowledge, there are no clear guidelines in the literature for selecting the "correct" sample size. In BooStSa, we prevent the selection of extreme sub-sample sizes (too small or too big) by only allowing a range between 5% and 50% of the test set size. Beyond this, we suggest choosing smaller sub-sample sizes, if the test set size allows, as this should keep the sub-sample size far from the dangerous extreme values. We also encourage practitioners to use bootstrap sampling responsibly and transparently by always specifying the chosen test parameters.

Number of iterations. On the other hand, tuning the number of iterations is straightforward: the more, the better. The confidence intervals shown in figure 1 suggest that $1,000$ iterations are already sufficient to obtain quite stable results; therefore, the often suggested $10,000$ iterations are a perfectly safe amount.

3 Experiment

Figure 2 shows the BooStSa's output for a real use case. We use the barely significant results of the Part-Of-Speech (POS) tagging classification task carried out by Fornaciari et al. (2021, p.2593, table 1, POS tag, separate test set, STL vs. MTL + Cross-Entropy). In that task, a Single-Task Learning (STL) model (*h0*) is compared with a Multi-Task Learning (MTL) model (*h1*). A hold-out validation procedure is followed, with a test set containing

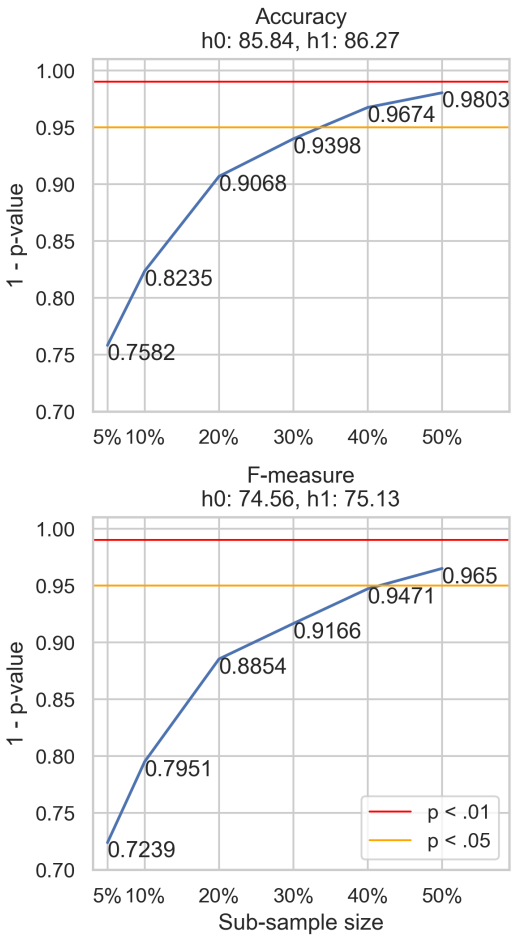

Figure 2: P-levels from significance test simulations with different sub-sample rates and 10 000 iterations. The results concern a POS-tagging classification task from the study of Fornaciari et al. (2021), where a Single-Task Learning (STL) model (*h0*) is compared with a Multi-Task Learning (MTL) model (*h1*).

3064 instances.

We consider two metrics, accuracy and F-measure, and six different sub-sample sizes, from 5% to 50% of test set rate. The *h0*'s accuracy and F-measure were 85.84 and 74.56, the *h1*'s accuracy and F-measure were 86.27 and 75.13, with a delta of 0.43 and 0.57, respectively. Similar to the results shown by Fornaciari et al. (2021), significant p-values appear only for sample sizes greater than 30% of the whole test set.

4 Usage

4.1 Installation

BooStSa can be installed in the shell simply via: `pip install -U boostsa`.

4.2 Getting started

To use BooStSa in a Python script, the first step is to import the library:

```
from boostsa import Bootstrap
```

Second, we need to create a bootstrap instance:

```
boot = Bootstrap()
```

This instance will store the experiments' outcomes and compute performances and the significance test between the experiments that need to be compared.

4.2.1 Inputs

The basic assumption for using BooStSa is that at least two classification models have been trained.

One model is considered the baseline, *control*, or null hypothesis (*h0*). The other is the experimental model, *treatment*, or hypothesis 1 (*h1*). In most settings, we expect this model to beat the baseline.

Their respective performance is tested against the same test set. It does not matter if this is a dedicated test set or resulting from a k-fold cross-validation procedure. In any case, the test set (*targets*) must be the same for both models.

The *h0* predictions, *h1* predictions and *targets* are the inputs for the Bootstrap instance.

4.2.2 Outputs

BooStSa's outputs are directly printed to standard out, and returned as a pandas `DataFrame`, that can be directly used, for example, to export them to a LaTex table.

However, when the `Boostsa()` object is instantiated, it is possible to define which output to save on disk. BooStSa can produce two kinds of outputs:

`results.tsv` It contains the experiments' performance and the (possible) significance levels of the experimental against the *control* conditions;

`outcomes.json` It contains targets and predictions for all the experimental conditions.

Target and predictions are the same that BooStSa takes as inputs. However, as described in Section 4.4, it can be useful to save them in JSON format if the test needs to be rerun, for example, when adding new experimental conditions to those that had already been considered. In these cases, feeding BooStSa with `outcomes.json` allows us to recreate the previous inputs' configuration

without needing to instantiate BooStSa again from scratch.

These outputs can be created using the following parameters:

save_results Type: `bool`; default: `True`. Boolean variable to determine whether to save the performances and the tests' results.

save_outcomes Type: `bool`; default: `True`. Boolean variable to determine whether to save as json the input predictions and targets.

dir_out Type: `str`; default: `''`. String variable that indicates the directory where to save `results.tsv` and `outcomes.json`.

The box below shows an example:

```
boot = Bootstrap(save_outcomes=False,
    dir_out='my/favourite/directory/')
```

4.3 Simple use-case

In the simplest use case, it is necessary to carry out the significance test between the predictions of two experiments. This can be done with the `test` function, which accepts the following parameters:

targs Type: `list`, `numpy.array` or `str`. They are the *targets*, or test set, that is the benchmark to measure the *h0* and *h1* predictions' performance. BooStSa automatically infers from the input shape if hard or soft labels are provided, according to these cases:

- A simple `list` will be assumed to be a list of integers, each corresponding to hard classes' indexes.
- A `list` of `list`s will be assumed to contain in each sub-list, as a row in a 2D matrix, float numbers summing up to one, which will be treated as soft labels.
- A 1D or 1-column `numpy.array` will be considered as containing integers for hard labels.
- A 2D `numpy.array` will be treated as containing float numbers constituting a soft label in each row.
- The `str` input will be processed as a full path to a file, which will have to comply with the following rules:
  - A file with extension '.txt' has to contain an integer in each row, representing hard classes' indexes.
  - A file with extension '.csv' has to contain comma-separated values for soft labels.
  - A file with extension '.tsv' has to contain tab-separated values for soft labels.
  - A file with extension '.npy' has to contain a NumPy binary file.

h0_preds Type: `list`, `numpy.array` or `str`. The *h0* predictions, in the same formats of `targs`.

h1_preds Type: `list`, `numpy.array` or `str`. The *h1* predictions, in the same formats as above.

h0_name Type: `str`, default: `h0`. Expression to describe the *h0* condition.

h1_name Type: `str`, default: `h1`. Expression to describe the *h1* condition.

n_loops Type: `int`, default: `1000`. Number of iterations for computing the bootstrap sampling.

sample_size Type: `float`, default: `.1`. Percentage of data points sampled from their whole set. The admitted values range between 0.05 (5%) and 0.5 (50%).

targetclass Type: `int`, default: `None`. If provided, it is interpreted as a label index, and for hard labels BooStSa will provide performance and significance levels with respect to that class. The parameter has no effect with soft labels.

verbose Type: `bool`, default: `False`. If true, the experiments' performance is printed on the shell.

An example of `test` function use is shown in figure 3. The significance levels are indicated by $** : p \leq .01$ and $* : p \leq .05$. Figure 4 shows an example with soft labels as inputs. Note that, for *CE* and *JSD*, the difference between the baseline and experimental model is negative. In fact, they are distance measures; therefore, lower is better in their case.

```
boot.test(targs='test_boot/hard/h0.0/targs.txt', h0_preds='test_boot/hard/h0.0/preds
    .txt', h1_preds='test_boot/hard/h1.0/preds.txt', h0_name='baseline', h1_name='
    experiment', n_loops=1000, sample_size=.2, verbose=True)
```

```
data shape:  (1000, 1)
sample size: 200
h0: h0 - h1: h1
targs count:    ['class 0 freq 465 perc 46.50%', 'class 1 freq 535 perc 53.50%']
h0 preds count: ['class 0 freq 339 perc 33.90%', 'class 1 freq 661 perc 66.10%']
h1 preds count: ['class 0 freq 500 perc 50.00%', 'class 1 freq 500 perc 50.00%']
F-measure..... - h0: 0.6776  - h1: 0.7407  - diff: 0.0631
accuracy...... - h0: 0.6900  - h1: 0.7410  - diff: 0.0510
precision..... - h0: 0.6994  - h1: 0.7410  - diff: 0.0416
recall........ - h0: 0.6796  - h1: 0.7422  - diff: 0.0626
bootstrap: 100%|==========================| 1000/1000 [00:09<00:00, 100.40it/s]
count sample diff f1   is twice tot diff f1..... 15   / 1000    p < 0.015  *
count sample diff prec is twice tot diff prec..... 65   / 1000    p < 0.065
count sample diff rec  is twice tot diff rec ..... 9    / 1000    p < 0.009  **
count sample diff acc  is twice tot diff acc..... 38   / 1000    p < 0.038  *
```

Figure 3: Input and output of the `test` function.

```
data shape:  (1000, 3)
sample size: 200
h0: h0 - h1: h1
targs distribution:        [0.33241763 0.33790091 0.32968146]
h0_preds distribution:     [0.33571912 0.33230295 0.33391209]
h1_preds distribution:     [0.33337905 0.3336012  0.33301975]
Jensen-Shannon divergence: - h0: 0.2143  - h1: 0.1817  - diff: -0.0326
cross-entropy:             - h0: 1.3515  - h1: 1.0886  - diff: -0.2629
entropy similarity:        - h0: 0.9816  - h1: 0.9857  - diff: 0.0041
entropy correlation:       - h0: 0.0064  - h1: 0.0529  - diff: 0.0465
bootstrap: 100%|==========================| 1000/1000 [00:05<00:00, 181.20it/s]
count sample diff jsd is twice tot diff jsd..... 0    / 1000    p < 0.0    **
count sample diff ce  is twice tot diff ce..... 0    / 1000    p < 0.0    **
count sample diff sim is twice tot diff sim..... 7    / 1000    p < 0.007  **
count sample diff cor is twice tot diff cor..... 315  / 1000    p < 0.315
```

Figure 4: Output of the `test` function with soft labels.

```
boot = Bootstrap(dir_out='my/favourite/dir/')
boot.feed(h0='h0', exp_idx='h0.0', preds=h0_exp1_preds, targs=targs)
boot.feed(h0='h0', h1='h1', exp_idx='h1.0', preds=h1_exp1_preds, targs=targs)
```

```
next_boot = Bootstrap() # if not already instantiated
next_boot.loadjson('my/favourite/dir/outcomes.json')
next_boot.feed(h0='h0', exp_idx='h0.1', preds=h0_exp2_preds, targs=targs)
next_boot.feed(h0='h0', h1='h1', exp_idx='h1.1', preds=h1_exp2_preds, targs=targs)
next_boot.run(n_loops=1000, sample_size=.2)
```

Figure 5: `feed`, `loadjson` and `run` functions.

4.4 BooStSa in a pipeline

In most cases, the experimental conditions are complex and imply the comparison between several baselines and several different experimental models, and for all of them, several runs of experiments can be planned. In those cases, BooStSa can be included in the whole pipeline, collecting the experiments' outcomes while they are produced, storing them and computing performance and significance levels at the end of the whole procedure.

This is done with two functions, `feed` and `run`. The first one collects BooStSa's inputs while they are produced by the experiments; the second one computes performance and bootstrap sampling. The `feed` functions feeds the `outcomes.json` file and takes the following inputs:

`h0` Type: `str`. This is a string that identifies a *control* condition. It must be provided both in the case of *control* experiments (*h0*) and *treatment* experiments (*h1*s) because BooStSa needs to know with which baseline the *treatment* results have to be compared.

`h1` Type: `str`; default: `None`. This is a string

that identifies a *treatment* condition. It must be provided only in case of *treatment* experiments (*h1*s).

exp_idx Type: `str`; default: `None`. This string (ideally, an index) identifies each unique experiment within its experimental condition, defined by `h0` or `h1`. This is useful in case of multiple experiments for the same experimental condition.

targs, **preds** Type: `list`, `numpy.array` or `str`; default: `None`. Equivalent to the `targs`, `h0_preds`, and `h1_preds` parameters in `test`.

idxs Type: `list`, `numpy.array` or `str`; default: `None`. Treated like `targs` and `preds`, `idxs` stores the indexes of the data points, that might have been shuffled during the experiments. The data point order does not affect the bootstrap sampling, but storing the shuffled indexes allows us to link the predictions to the original data points later on.

epochs Type: `int`. Lastly, `epoch` can store the number of training epochs run by the experiment.

The flexibility of `feed` allows storing together several *control* conditions and the relatives *treatment*s for comparison. Also, the process can be stopped and resumed: the function `loadjson` allows to load a previously saved `outcomes.json` file and to keep on feeding it. Once all the inputs are provided, the `run` function computes performance and bootstrap sampling. The `run` parameters are:

n_loops Type: `int`; default: `1000`. The iterations' number of bootstrap sampling.

sample_size Type: `float`; default: `.1`. The sub-sample size, expressed as percent part of the test set. The value is constrained between .05 and .5 (inclusive).

targetclass Type: `int`, default: `None`. Equal to the same parameter in `test`.

verbose Type: `bool`, default: `False`. If true, the experiments' performance is printed on the shell.

Figure 5 shows the whole process.

5 Limitations

Besides the constraints discussed in Section 2.3, aimed to prevent the significance test misuse, BooStSa assumes that *h0* predictions, *h1* predictions, and *targets* are available. When comparing with results from previous literature, this assumption might not hold: in these cases, to apply BooStSa it is necessary to find or reproduce the *h0* outcomes.

6 Conclusion

We present BooStSa, a Python package to allow NLP practitioners to efficiently compute significance values for hard and soft labels using a safe set of hyper-parameters.

We discuss how the test hyper-parameters can affect the outcome, introducing safety constraints for the test use and suggesting, as a good practice, to report the hyper-parameters with the experiments' results.

While the metrics for hard labels consolidated in literature, those for soft labels that measure the divergence between probability distributions are not yet widely used, and the consensus about their interpretation is still on the way to be reached. We follow the extensive survey of Uma et al. (2021), which takes into consideration the most recent trends in NLP.

We also agree with Basile et al. (2021), who point out that incorporating into the models the information about the intrinsic entities' ambiguity, expressed as inter-coders disagreement and represented as a probability distribution over different classes, is a necessary step to create models that carry out NLP task with human-like performance.

Acknowledgments

This research has been partially funded from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program No. 949944, INTEGRATOR and No. 695662, DALI. TF and DH are members of the Data and Marketing Insights Unit at the Bocconi Institute for Data Science and Analysis.

7 Ethical Considerations

Psychology has seen a growing scandal around p-hacking, i.e., the generation of enough experimental variations to produce a significant outcome in one of them. This risk is low in NLP, as significance alone is not sufficient for publication: typically, proof of predictive performance on ideally

several test sets is necessary instead. Reporting significance in NLP is therefore an additional robustness measure, indicating the model's generalizability. While users might abuse the capabilities of BooStSa to make significant results more likely, this would require deliberate tampering (and even then would not guarantee significance). If used as intended, however, BooStSa should reduce unintended variance via researcher degrees of freedom and make results more comparable and reproducible.

References

Valerio Basile, Michael Fell, Tommaso Fornaciari, Dirk Hovy, Silviu Paun, Barbara Plank, Massimo Poesio, and Alexandra Uma. 2021. We need to consider disagreement in evaluation. In *Proceedings of the 1st Workshop on Benchmarking: Past, Present and Future*, pages 15–21, Online. Association for Computational Linguistics.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in NLP. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 995–1005, Jeju Island, Korea. Association for Computational Linguistics.

Bradley Efron and Robert J Tibshirani. 1994. *An introduction to the bootstrap*. CRC press.

George Forman et al. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.*, 3(Mar):1289–1305.

Tommaso Fornaciari, Alexandra Uma, Silviu Paun, Barbara Plank, Dirk Hovy, and Massimo Poesio. 2021. Beyond black & white: Leveraging annotator disagreement via soft-label multi-task learning. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2591–2597, Online. Association for Computational Linguistics.

Cyril Goutte and Eric Gaussier. 2005. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European conference on information retrieval*, pages 345–359. Springer.

Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Jianhua Lin. 1991. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151.

Karl Pearson. 1896. Vii. mathematical contributions to the theory of evolution.—iii. regression, heredity, and panmixia. *Philosophical Transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character*, sblindo(187):253–318.

Joshua C Peterson, Ruairidh M Battleday, Thomas L Griffiths, and Olga Russakovsky. 2019. Human uncertainty makes classification more robust. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9617–9626.

Georg Pólya. 1920. Üon the central limit theorem of probability theory and the problem of moments. *Mathematical Journal*, 8(3):171–181.

Murray Rosenblatt. 1956. A central limit theorem and a strong mixing condition. *Proceedings of the National Academy of Sciences of the United States of America*, 42(1):43.

Anders Søgaard, Anders Johannsen, Barbara Plank, Dirk Hovy, and Hector Martínez Alonso. 2014. What's in a p-value in NLP? In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 1–10, Ann Arbor, Michigan. Association for Computational Linguistics.

Alexandra N Uma, Tommaso Fornaciari, Dirk Hovy, Silviu Paun, Barbara Plank, and Massimo Poesio. 2021. Learning from disagreement: A survey. *Journal of Artificial Intelligence Research*, 72:1385–1470.