Exploiting Cloze Questions for Few Shot Text Classification and Natural Language Inference

Timo Schick[1,2] **Hinrich Schütze**[1]

[1] Center for Information and Language Processing, LMU Munich, Germany
[2] Sulzer GmbH, Munich, Germany

schickt@cis.lmu.de inquiries@cislmu.org

Abstract

Some NLP tasks can be solved in a fully unsupervised fashion by providing a pretrained language model with "task descriptions" in natural language (e.g., Radford et al., 2019). While this approach underperforms its supervised counterpart, we show in this work that the two ideas can be combined: We introduce Pattern-Exploiting Training (PET), a semi-supervised training procedure that reformulates input examples as cloze-style phrases to help language models understand a given task. These phrases are then used to assign soft labels to a large set of unlabeled examples. Finally, standard supervised training is performed on the resulting training set. For several tasks and languages, PET outperforms supervised training and strong semi-supervised approaches in low-resource settings by a large margin.[1]

1 Introduction

Learning from examples is the predominant approach for many NLP tasks: A model is trained on a set of labeled examples from which it then generalizes to unseen data. Due to the vast number of languages, domains and tasks and the cost of annotating data, it is common in real-world uses of NLP to have only a small number of labeled examples, making *few-shot learning* a highly important research area. Unfortunately, applying standard supervised learning to small training sets often performs poorly; many problems are difficult to grasp from just looking at a few examples. For instance, assume we are given the following pieces of text:

- T_1: This was the best pizza I've ever had.
- T_2: You can get better sushi for half the price.
- T_3: Pizza was average. Not worth the price.

[1]Our implementation is publicly available at https://github.com/timoschick/pet.

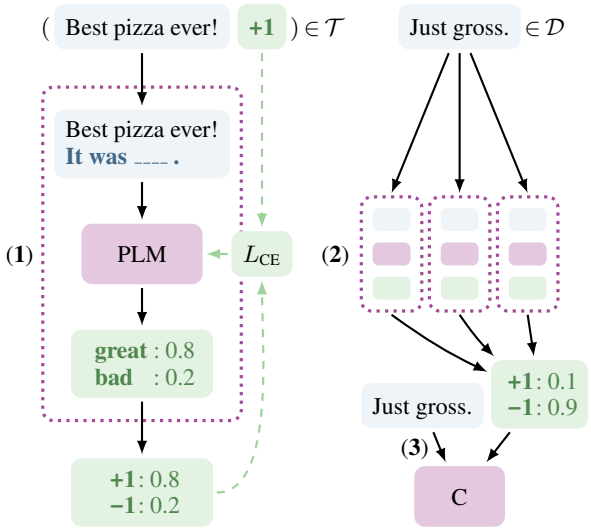

Figure 1: PET for sentiment classification. **(1)** A number of patterns encoding some form of task description are created to convert training examples to cloze questions; for each pattern, a pretrained language model is finetuned. **(2)** The ensemble of trained models annotates unlabeled data. **(3)** A classifier is trained on the resulting soft-labeled dataset.

Furthermore, imagine we are told that the labels of T_1 and T_2 are l and l', respectively, and we are asked to infer the correct label for T_3. Based only on these examples, this is impossible because plausible justifications can be found for both l and l'. However, if we know that the underlying task is to identify whether the text says anything about prices, we can easily assign l' to T_3. This illustrates that solving a task from only a few examples becomes much easier when we also have a *task description*, i.e., a textual explanation that helps us *understand* what the task is about.

With the rise of pretrained language models (PLMs) such as GPT (Radford et al., 2018), BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019), the idea of providing task descriptions has become feasible for neural architectures: We can

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, pages 255–269
April 19 - 23, 2021.

simply append such descriptions in natural language to an input and let the PLM predict continuations that solve the task (Radford et al., 2019; Puri and Catanzaro, 2019). So far, this idea has mostly been considered in zero-shot scenarios where no training data is available at all.

In this work, we show that providing task descriptions can successfully be combined with standard supervised learning in few-shot settings: We introduce **P**attern-**E**xploiting **T**raining (PET), a semi-supervised training procedure that uses natural language patterns to reformulate input examples into cloze-style phrases. As illustrated in Figure 1, PET works in three steps: First, for each pattern a separate PLM is finetuned on a small training set $\mathcal{T}$. The ensemble of all models is then used to annotate a large unlabeled dataset $\mathcal{D}$ with soft labels. Finally, a standard classifier is trained on the soft-labeled dataset. We also devise iPET, an iterative variant of PET in which this process is repeated with increasing training set sizes.

On a diverse set of tasks in multiple languages, we show that given a small to medium number of labeled examples, PET and iPET substantially outperform unsupervised approaches, supervised training and strong semi-supervised baselines.

2 Related Work

Radford et al. (2019) provide hints in the form of natural language patterns for zero-shot learning of challenging tasks such as reading comprehension and question answering (QA). This idea has been applied to unsupervised text classification (Puri and Catanzaro, 2019), commonsense knowledge mining (Davison et al., 2019) and argumentative relation classification (Opitz, 2019). Srivastava et al. (2018) use task descriptions for zero-shot classification but require a semantic parser. For relation extraction, Bouraoui et al. (2020) automatically identify patterns that express given relations. McCann et al. (2018) rephrase several tasks as QA problems. Raffel et al. (2020) frame various problems as language modeling tasks, but their patterns only loosely resemble natural language and are unsuitable for few-shot learning.[2]

Another recent line of work uses cloze-style phrases to probe the knowledge that PLMs acquire during pretraining; this includes probing for factual and commonsense knowledge (Trinh and Le, 2018; Petroni et al., 2019; Wang et al., 2019; Sakaguchi et al., 2020), linguistic capabilities (Ettinger, 2020; Kassner and Schütze, 2020), understanding of rare words (Schick and Schütze, 2020), and ability to perform symbolic reasoning (Talmor et al., 2019). Jiang et al. (2020) consider the problem of finding the best pattern to express a given task.

Other approaches for few-shot learning in NLP include exploiting examples from related tasks (Yu et al., 2018; Gu et al., 2018; Dou et al., 2019; Qian and Yu, 2019; Yin et al., 2019) and using data augmentation (Xie et al., 2020; Chen et al., 2020); the latter commonly relies on back-translation (Sennrich et al., 2016), requiring large amounts of parallel data. Approaches using textual class descriptors typically assume that abundant examples are available for a subset of classes (e.g., Romera-Paredes and Torr, 2015; Veeranna et al., 2016; Ye et al., 2020). In contrast, our approach requires no additional labeled data and provides an intuitive interface to leverage task-specific human knowledge.

The idea behind iPET – training multiple generations of models on data labeled by previous generations – bears resemblance to self-training and bootstrapping approaches for word sense disambiguation (Yarowsky, 1995), relation extraction (Brin, 1999; Agichtein and Gravano, 2000; Batista et al., 2015), parsing (McClosky et al., 2006; Reichart and Rappoport, 2007; Huang and Harper, 2009), machine translation (Hoang et al., 2018), and sequence generation (He et al., 2020).

3 Pattern-Exploiting Training

Let M be a masked language model with vocabulary V and mask token $\_\_\_\_ \in V$, and let $\mathcal{L}$ be a set of labels for our target classification task A. We write an input for task A as a sequence of phrases $\mathbf{x} = (s_1, \ldots, s_k)$ with $s_i \in V^*$; for example, $k = 2$ if A is textual inference (two input sentences). We define a *pattern* to be a function P that takes $\mathbf{x}$ as input and outputs a phrase or sentence $P(\mathbf{x}) \in V^*$ that contains exactly one mask token, i.e., its output can be viewed as a cloze question. Furthermore, we define a *verbalizer* as an injective function $v : \mathcal{L} \rightarrow V$ that maps each label to a word from M's vocabulary. We refer to (P, v) as a *pattern-verbalizer pair* (PVP).

Using a PVP (P, v) enables us to solve task A as follows: Given an input $\mathbf{x}$, we apply P to obtain an input representation $P(\mathbf{x})$, which is then processed

[2] For example, they convert inputs (a, b) for recognizing textual entailment (RTE) to "rte sentence1: a sentence2: b", and the PLM is asked to predict strings like "not_entailment".

by M to determine the label $y \in \mathcal{L}$ for which $v(y)$ is the most likely substitute for the mask. For example, consider the task of identifying whether two sentences a and b contradict each other (label y_0) or agree with each other (y_1). For this task, we may choose the pattern $P(a, b) =$ a? ____, b. combined with a verbalizer v that maps y_0 to "Yes" and y_1 to "No". Given an example input pair

$$\mathbf{x} = (\text{Mia likes pie, Mia hates pie}),$$

the task now changes from having to assign a label without inherent meaning to answering whether the most likely choice for the masked position in

$$P(\mathbf{x}) = \text{Mia likes pie? ____, Mia hates pie.}$$

is "Yes" or "No".

3.1 PVP Training and Inference

Let $\mathbf{p} = (P, v)$ be a PVP. We assume access to a small training set $\mathcal{T}$ and a (typically much larger) set of unlabeled examples $\mathcal{D}$. For each sequence $\mathbf{z} \in V^*$ that contains exactly one mask token and $w \in V$, we denote with $M(w \mid \mathbf{z})$ the unnormalized score that the language model assigns to w at the masked position. Given some input $\mathbf{x}$, we define the score for label $l \in \mathcal{L}$ as

$$s_{\mathbf{p}}(l \mid \mathbf{x}) = M(v(l) \mid P(\mathbf{x}))$$

and obtain a probability distribution over labels using softmax:

$$q_{\mathbf{p}}(l \mid \mathbf{x}) = \frac{e^{s_{\mathbf{p}}(l|\mathbf{x})}}{\sum_{l' \in \mathcal{L}} e^{s_{\mathbf{p}}(l'|\mathbf{x})}}$$

We use the cross-entropy between $q_{\mathbf{p}}(l \mid \mathbf{x})$ and the true (one-hot) distribution of training example $(\mathbf{x}, l)$ – summed over all $(\mathbf{x}, l) \in \mathcal{T}$ – as loss for finetuning M for $\mathbf{p}$.

3.2 Auxiliary Language Modeling

In our application scenario, only a few training examples are available and catastrophic forgetting can occur. As a PLM finetuned for some PVP is still a language model at its core, we address this by using language modeling as auxiliary task. With L_{CE} denoting cross-entropy loss and L_{MLM} language modeling loss, we compute the final loss as

$$L = (1 - \alpha) \cdot L_{\text{CE}} + \alpha \cdot L_{\text{MLM}}$$

This idea was recently applied by Chronopoulou et al. (2019) in a data-rich scenario. As L_{MLM} is typically much larger than L_{CE}, in preliminary experiments, we found a small value of $\alpha = 10^{-4}$ to consistently give good results, so we use it in all our experiments. To obtain sentences for language modeling, we use the unlabeled set $\mathcal{D}$. However, we do not train directly on each $\mathbf{x} \in \mathcal{D}$, but rather on $P(\mathbf{x})$, where we never ask the language model to predict anything for the masked slot.

3.3 Combining PVPs

A key challenge for our approach is that in the absence of a large development set, it is hard to identify which PVPs perform well. To address this, we use a strategy similar to knowledge distillation (Hinton et al., 2015). First, we define a set $\mathcal{P}$ of PVPs that intuitively make sense for a given task A. We then use these PVPs as follows:

(1) We finetune a separate language model $M_{\mathbf{p}}$ for each $\mathbf{p} \in \mathcal{P}$ as described in Section 3.1. As $\mathcal{T}$ is small, this finetuning is cheap even for a large number of PVPs.

(2) We use the ensemble $\mathcal{M} = \{M_{\mathbf{p}} \mid \mathbf{p} \in \mathcal{P}\}$ of finetuned models to annotate examples from $\mathcal{D}$. We first combine the unnormalized class scores for each example $\mathbf{x} \in \mathcal{D}$ as

$$s_{\mathcal{M}}(l \mid \mathbf{x}) = \frac{1}{Z} \sum_{\mathbf{p} \in \mathcal{P}} w(\mathbf{p}) \cdot s_{\mathbf{p}}(l \mid \mathbf{x})$$

where $Z = \sum_{\mathbf{p} \in \mathcal{P}} w(\mathbf{p})$ and the $w(\mathbf{p})$ are weighting terms for the PVPs. We experiment with two different realizations of this weighing term: either we simply set $w(\mathbf{p}) = 1$ for all $\mathbf{p}$ or we set $w(\mathbf{p})$ to be the accuracy obtained using $\mathbf{p}$ on the training set *before* training. We refer to these two variants as *uniform* and *weighted*. Jiang et al. (2020) use a similar idea in a zero-shot setting.

We transform the above scores into a probability distribution q using softmax. Following Hinton et al. (2015), we use a temperature of $T = 2$ to obtain a suitably soft distribution. All pairs $(\mathbf{x}, q)$ are collected in a (soft-labeled) training set $\mathcal{T}_C$.

(3) We finetune a PLM C with a standard sequence classification head on $\mathcal{T}_C$.

The finetuned model C then serves as our classifier for A. All steps described above are depicted in Figure 2; an example is shown in Figure 1.

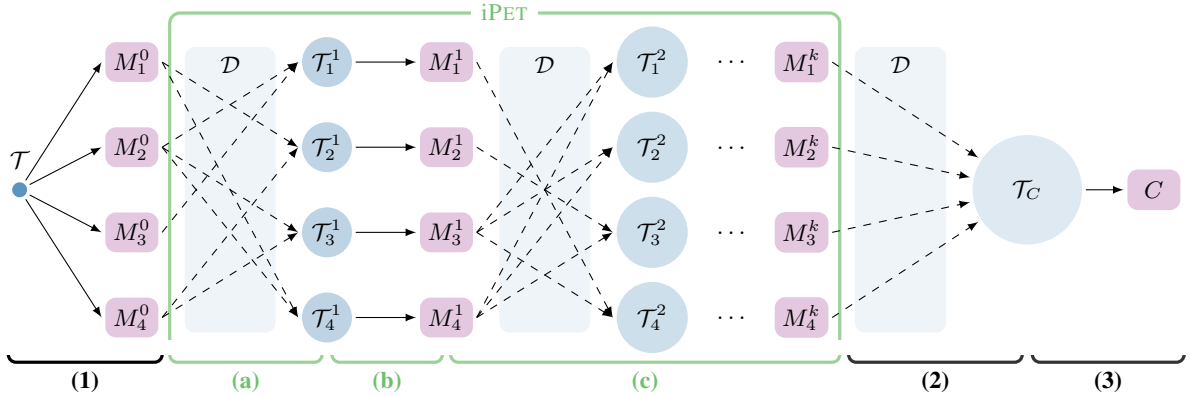

Figure 2: Schematic representation of PET (1-3) and iPET (a-c). **(1)** The initial training set is used to finetune an ensemble of PLMs. **(a)** For each model, a random subset of other models generates a new training set by labeling examples from $\mathcal{D}$. **(b)** A new set of PET models is trained using the larger, model-specific datasets. **(c)** The previous two steps are repeated k times, each time increasing the size of the generated training sets by a factor of d. **(2)** The final set of models is used to create a soft-labeled dataset $\mathcal{T}_C$. **(3)** A classifier C is trained on this dataset.

3.4 Iterative PET (iPET)

Distilling the knowledge of all individual models into a single classifier C means they cannot learn from each other. As some patterns perform (possibly much) worse than others, the training set $\mathcal{T}_C$ for our final model may therefore contain many mislabeled examples.

To compensate for this shortcoming, we devise iPET, an iterative variant of PET. The core idea of iPET is to train several *generations* of models on datasets of increasing size. To this end, we first enlarge the original dataset $\mathcal{T}$ by labeling selected examples from $\mathcal{D}$ using a random subset of trained PET models (Figure 2a). We then train a new generation of PET models on the enlarged dataset (b); this process is repeated several times (c).

More formally, let $\mathcal{M}^0 = \{M_1^0, \ldots, M_n^0\}$ be the initial set of PET models finetuned on $\mathcal{T}$, where each M_i^0 is trained for some PVP $\mathbf{p}_i$. We train k generations of models $\mathcal{M}^1, \ldots, \mathcal{M}^k$ where $\mathcal{M}^j = \{M_1^j, \ldots, M_n^j\}$ and each M_i^j is trained for $\mathbf{p}_i$ on its own training set $\mathcal{T}_i^j$. In each iteration, we multiply the training set size by a fixed constant $d \in \mathbb{N}$ while maintaining the label ratio of the original dataset. That is, with $c_0(l)$ denoting the number of examples with label l in $\mathcal{T}$, each $\mathcal{T}_i^j$ contains $c_j(l) = d \cdot c_{j-1}(l)$ examples with label l. This is achieved by generating each $\mathcal{T}_i^j$ as follows:

1. We obtain $\mathcal{N} \subset \mathcal{M}^{j-1} \setminus \{M_i^{j-1}\}$ by randomly choosing $\lambda \cdot (n-1)$ models from the previous generation with $\lambda \in (0, 1]$ being a hyperparameter.

2. Using this subset, we create a labeled dataset

$$\mathcal{T}_{\mathcal{N}} = \{(\mathbf{x}, \arg\max_{l \in \mathcal{L}} s_{\mathcal{N}}(l \mid \mathbf{x})) \mid \mathbf{x} \in \mathcal{D}\}.$$

   For each $l \in \mathcal{L}$, we obtain $\mathcal{T}_{\mathcal{N}}(l) \subset \mathcal{T}_{\mathcal{N}}$ by randomly choosing $c_j(l) - c_0(l)$ examples with label l from $\mathcal{T}_{\mathcal{N}}$. To avoid training future generations on mislabeled data, we prefer examples for which the ensemble of models is confident in its prediction. The underlying intuition is that even without calibration, examples for which labels are predicted with high confidence are typically more likely to be classified correctly (Guo et al., 2017). Therefore, when drawing from $\mathcal{T}_{\mathcal{N}}$, we set the probability of each $(\mathbf{x}, y)$ proportional to $s_{\mathcal{N}}(l \mid \mathbf{x})$.

3. We define $\mathcal{T}_i^j = \mathcal{T} \cup \bigcup_{l \in \mathcal{L}} \mathcal{T}_{\mathcal{N}}(l)$. As can easily be verified, this dataset contains $c_j(l)$ examples for each $l \in \mathcal{L}$.

After training k generations of PET models, we use $\mathcal{M}^k$ to create $\mathcal{T}_C$ and train C as in basic PET.

With minor adjustments, iPET can even be used in a zero-shot setting. To this end, we define $\mathcal{M}^0$ to be the set of *untrained* models and $c_1(l) = 10/|\mathcal{L}|$ for all $l \in \mathcal{L}$ so that $\mathcal{M}^1$ is trained on 10 examples evenly distributed across all labels. As $\mathcal{T}_{\mathcal{N}}$ may not contain enough examples for some label l, we create all $\mathcal{T}_{\mathcal{N}}(l)$ by sampling from the 100 examples $\mathbf{x} \in \mathcal{D}$ for which $s_{\mathcal{N}}(l \mid x)$ is the highest, even if $l \neq \arg\max_{l \in \mathcal{L}} s_{\mathcal{N}}(l \mid x)$. For each subsequent generation, we proceed exactly as in basic iPET.

4 Experiments

We evaluate PET on four English datasets: Yelp Reviews, AG's News, Yahoo Questions (Zhang et al., 2015) and MNLI (Williams et al., 2018). Additionally, we use x-stance (Vamvas and Sennrich, 2020) to investigate how well PET works for other languages. For all experiments on English, we use RoBERTa large (Liu et al., 2019) as language model; for x-stance, we use XLM-R (Conneau et al., 2020). We investigate the performance of PET and all baselines for different training set sizes; each model is trained three times using different seeds and average results are reported.

As we consider a few-shot setting, we assume no access to a large development set on which hyperparameters could be optimized. Our choice of hyperparameters is thus based on choices made in previous work and practical considerations. We use a learning rate of $1 \cdot 10^{-5}$, a batch size of 16 and a maximum sequence length of 256. Unless otherwise specified, we always use the weighted variant of PET with auxiliary language modeling. For iPET, we set $\lambda = 0.25$ and $d = 5$; that is, we select 25% of all models to label examples for the next generation and quintuple the number of training examples in each iteration. We train new generations until each model was trained on at least 1000 examples, i.e., we set $k = \lceil \log_d(1000/|\mathcal{T}|) \rceil$. As we always repeat training three times, the ensemble $\mathcal{M}$ (or $\mathcal{M}^0$) for n PVPs contains $3n$ models. Further hyperparameters and detailed explanations for all our choices are given in Appendix B.

4.1 Patterns

We now describe the patterns and verbalizers used for all tasks. We use two vertical bars ($\|$) to mark boundaries between text segments.[3]

Yelp For the Yelp Reviews Full Star dataset (Zhang et al., 2015), the task is to estimate the rating that a customer gave to a restaurant on a 1- to 5-star scale based on their review's text. We define the following patterns for an input text a:

$P_1(a) =$ It was ____. a $\quad P_2(a) =$ Just ____! $\| \; a$

$P_3(a) =$ a. All in all, it was ____.

$P_4(a) =$ $a \; \|$ In summary, the restaurant is ____.

[3]The way different segments are handled depends on the model being used; they may e.g. be assigned different embeddings (Devlin et al., 2019) or separated by special tokens (Liu et al., 2019; Yang et al., 2019). For example, "$a \; \| \; b$" is given to BERT as the input "[CLS] a [SEP] b [SEP]".

We define a single verbalizer v for all patterns as

$v(1) =$ terrible $\quad v(2) =$ bad $\quad v(3) =$ okay

$v(4) =$ good $\quad v(5) =$ great

AG's News AG's News is a news classification dataset, where given a headline a and text body b, news have to be classified as belonging to one of the categories *World* (1), *Sports* (2), *Business* (3) or *Science/Tech* (4). For $\mathbf{x} = (a, b)$, we define the following patterns:

$P_1(\mathbf{x}) =$ ____: a b $\quad P_2(\mathbf{x}) =$ a (____) b

$P_3(\mathbf{x}) =$ ____ – a b $\quad P_4(\mathbf{x}) =$ a b (____)

$P_5(\mathbf{x}) =$ ____ News: a b

$P_6(\mathbf{x}) =$ [Category: ____] a b

We use a verbalizer that maps 1–4 to "World", "Sports", "Business" and "Tech", respectively.

Yahoo Yahoo Questions (Zhang et al., 2015) is a text classification dataset. Given a question a and an answer b, one of ten possible categories has to be assigned. We use the same patterns as for AG's News, but we replace the word "News" in P_5 with the word "Question". We define a verbalizer that maps categories 1–10 to "Society", "Science", "Health", "Education", "Computer", "Sports", "Business", "Entertainment", "Relationship" and "Politics".

MNLI The MNLI dataset (Williams et al., 2018) consists of text pairs $\mathbf{x} = (a, b)$. The task is to find out whether a implies b (0), a and b contradict each other (1) or neither (2). We define

$P_1(\mathbf{x}) =$ "a"? $\|$ ____, "b" $\quad P_2(\mathbf{x}) =$ a? $\|$ ____, b

and consider two different verbalizers v_1 and v_2:

$v_1(0) =$ Wrong $\quad v_1(1) =$ Right $\quad v_1(2) =$ Maybe

$v_2(0) =$ No $\quad v_2(1) =$ Yes $\quad v_2(2) =$ Maybe

Combining the two patterns with the two verbalizers results in a total of 4 PVPs.

X-Stance The x-stance dataset (Vamvas and Sennrich, 2020) is a multilingual stance detection dataset with German, French and Italian examples. Each example $\mathbf{x} = (a, b)$ consists of a question a concerning some political issue and a comment b; the task is to identify whether the writer of b

Line	Examples	Method	Yelp	AG's	Yahoo	MNLI (m/mm)
1	$\|\mathcal{T}\| = 0$	unsupervised (avg)	33.8 ±9.6	69.5 ±7.2	44.0 ±9.1	39.1 ±4.3 / 39.8 ±5.1
2		unsupervised (max)	40.8 ±0.0	79.4 ±0.0	56.4 ±0.0	43.8 ±0.0 / 45.0 ±0.0
3		iPET	**56.7** ±0.2	**87.5** ±0.1	**70.7** ±0.1	**53.6** ±0.1 / **54.2** ±0.1
4	$\|\mathcal{T}\| = 10$	supervised	21.1 ±1.6	25.0 ±0.1	10.1 ±0.1	34.2 ±2.1 / 34.1 ±2.0
5		PET	52.9 ±0.1	87.5 ±0.0	63.8 ±0.2	41.8 ±0.1 / 41.5 ±0.2
6		iPET	**57.6** ±0.0	**89.3** ±0.1	**70.7** ±0.1	**43.2** ±0.0 / **45.7** ±0.1
7	$\|\mathcal{T}\| = 50$	supervised	44.8 ±2.7	82.1 ±2.5	52.5 ±3.1	45.6 ±1.8 / 47.6 ±2.4
8		PET	60.0 ±0.1	86.3 ±0.0	66.2 ±0.1	63.9 ±0.0 / 64.2 ±0.0
9		iPET	**60.7** ±0.1	**88.4** ±0.1	**69.7** ±0.0	**67.4** ±0.3 / **68.3** ±0.3
10	$\|\mathcal{T}\| = 100$	supervised	53.0 ±3.1	86.0 ±0.7	62.9 ±0.9	47.9 ±2.8 / 51.2 ±2.6
11		PET	61.9 ±0.0	88.3 ±0.1	69.2 ±0.0	74.7 ±0.3 / 75.9 ±0.4
12		iPET	**62.9** ±0.0	**89.6** ±0.1	**71.2** ±0.1	**78.4** ±0.7 / **78.6** ±0.5
13	$\|\mathcal{T}\| = 1000$	supervised	63.0 ±0.5	**86.9** ±0.4	70.5 ±0.3	73.1 ±0.2 / 74.8 ±0.3
14		PET	**64.8** ±0.1	**86.9** ±0.2	**72.7** ±0.0	**85.3** ±0.2 / **85.5** ±0.4

Table 1: Average accuracy and standard deviation for RoBERTa (large) on Yelp, AG's News, Yahoo and MNLI (m:matched/mm:mismatched) for five training set sizes $|\mathcal{T}|$.

supports the subject of the question (0) or not (1). We use two simple patterns

$$P_1(\mathbf{x}) = \text{“}a\text{”} \parallel \text{\_\_\_\_\_. “}b\text{”} \qquad P_2(\mathbf{x}) = a \parallel \text{\_\_\_\_\_. } b$$

and define an English verbalizer v_{En} mapping 0 to “Yes” and 1 to “No” as well as a French (German) verbalizer v_{Fr} (v_{De}), replacing “Yes” and “No” with “Oui” and “Non” (“Ja” and “Nein”). We do not define an Italian verbalizer because x-stance does not contain any Italian training examples.

4.2 Results

English Datasets Table 1 shows results for English text classification and language understanding tasks; we report mean accuracy and standard deviation for three training runs. Lines 1–2 (L1–L2) show unsupervised performance, i.e., individual PVPs without *any* training (similar to Radford et al., 2018; Puri and Catanzaro, 2019); we give both average results across all PVPs (avg) and results for the PVP that works best on the test set (max). The large difference between both rows highlights the importance of coping with the fact that without looking at the test set, we have no means of evaluating which PVPs perform well. Zero-shot iPET clearly outperforms the unsupervised baselines for all datasets (L3 vs L1); on AG's News, it even performs better than standard supervised training with 1000 examples (L3 vs L13). With just 10 training examples, standard supervised learning does not perform above chance (L4). In contrast, PET (L5) performs much better than the fully unsupervised baselines (L1–L2); training multiple generations using iPET (L6) gives consistent improvements. As we increase the training set size, the performance gains of PET and iPET become smaller, but for both 50 and 100 examples, PET continues to considerably outperform standard supervised training (L8 vs L7, L11 vs L10) with iPET (L9, L12) still giving consistent improvements. For $|\mathcal{T}| = 1000$, PET has no advantage on AG's but still improves accuracy for all other tasks (L14 vs L13).[4]

Ex.	Method	Yelp	AG's	Yahoo	MNLI
$\|\mathcal{T}\| = 10$	UDA	27.3	72.6	36.7	34.7
	MixText	20.4	81.1	20.6	32.9
	PET	48.8	84.1	59.0	39.5
	iPET	**52.9**	**87.5**	**67.0**	**42.1**
$\|\mathcal{T}\| = 50$	UDA	46.6	83.0	60.2	40.8
	MixText	31.3	84.8	61.5	34.8
	PET	55.3	86.4	63.3	55.1
	iPET	**56.7**	**87.3**	**66.4**	**56.3**

Table 2: Comparison of PET with two state-of-the-art semi-supervised methods using RoBERTa (base)

Comparison with SotA We compare PET to UDA (Xie et al., 2020) and MixText (Chen et al., 2020), two state-of-the-art methods for semi-supervised learning in NLP that rely on data augmentation. Whereas PET requires that a task can be expressed using patterns and that such patterns be found, UDA and MixText both use backtranslation (Sennrich et al., 2016) and thus require thousands of labeled examples for training a machine translation model. We use RoBERTa (base) for our comparison as MixText is specifically tailored towards

[4] One of the three supervised MNLI runs for $|\mathcal{T}| = 1000$ underfitted the training data and performed extremely poorly. This run is excluded in the reported score (73.1/74.8).

Examples	Method	De	Fr	It
$\lvert\mathcal{T}\rvert = 1000$	supervised	43.3	49.5	41.0
	PET	**66.4**	**68.7**	**64.7**
$\lvert\mathcal{T}\rvert = 2000$	supervised	57.4	62.1	52.8
	PET	**69.5**	**71.7**	**67.3**
$\lvert\mathcal{T}\rvert = 4000$	supervised	63.2	66.7	58.7
	PET	**71.7**	**74.0**	**69.5**
$\mathcal{T}_{\text{De}}$, $\mathcal{T}_{\text{Fr}}$	supervised	76.6	76.0	71.0
	PET	**77.9**	**79.0**	**73.6**
$\mathcal{T}_{\text{De}} + \mathcal{T}_{\text{Fr}}$	sup. (*)	76.8	76.7	70.2
	supervised	77.6	79.1	75.9
	PET	**78.8**	**80.6**	**77.2**

Table 3: Results on x-stance intra-target for XLM-R (base) trained on subsets of $\mathcal{T}_{\text{De}}$ and $\mathcal{T}_{\text{Fr}}$ and for joint training on all data ($\mathcal{T}_{\text{De}} + \mathcal{T}_{\text{Fr}}$). (*): Best results for mBERT reported in Vamvas and Sennrich (2020).

a 12-layer Transformer (Vaswani et al., 2017). Both Xie et al. (2020) and Chen et al. (2020) use large development sets to optimize the number of training steps. We instead try several values for both approaches directly on the test set and only report the *best* results obtained. Despite this, Table 2 shows that PET and iPET substantially outperform both methods across all tasks, clearly demonstrating the benefit of incorporating human knowledge in the form of PVPs.

X-Stance We evaluate PET on x-stance to investigate (i) whether it works for languages other than English and (ii) whether it also brings improvements when training sets have medium size. In contrast to Vamvas and Sennrich (2020), we do not perform any hyperparameter optimization on dev and use a shorter maximum sequence length (256 vs 512) to speed up training and evaluation.

To investigate whether PET brings benefits even when numerous examples are available, we consider training set sizes of 1000, 2000, and 4000; for each of these configurations, we separately finetune French and German models to allow for a more straightforward downsampling of the training data. Additionally, we train models on the entire French ($\lvert\mathcal{T}_{\text{Fr}}\rvert = 11\,790$) and German ($\lvert\mathcal{T}_{\text{De}}\rvert = 33\,850$) training sets. In this case we do not have any additional unlabeled data, so we simply set $\mathcal{D} = \mathcal{T}$. For the French models, we use v_{En} and v_{Fr} as verbalizers and for German v_{En} and v_{De} (Section 4.1). Finally, we also investigate the performance of a model trained jointly on French and German data ($\lvert\mathcal{T}_{\text{Fr}} + \mathcal{T}_{\text{De}}\rvert = 45\,640$) using v_{En}, v_{Fr} and v_{De}.

Results are shown in Table 3; following Vamvas and Sennrich (2020), we report the macro-average of the F1 scores for labels 0 and 1, averaged over three runs. For Italian (column "It"), we report the average zero-shot cross-lingual performance of German and French models as there are no Italian training examples. Our results show that PET brings huge improvements across all languages even when training on much more than a thousand examples; it also considerably improves zero-shot cross-lingual performance.

Method	Yelp	AG's	Yahoo	MNLI
min	39.6	82.1	50.2	36.4
max	52.4	85.0	63.6	40.2
PET (no distillation)	51.7	87.0	62.8	40.6
PET uniform	52.7	87.3	**63.8**	**42.0**
PET weighted	**52.9**	**87.5**	**63.8**	41.8

Table 4: Minimum (min) and maximum (max) accuracy of models based on individual PVPs as well as PET with and without knowledge distillation ($\lvert\mathcal{T}\rvert = 10$).

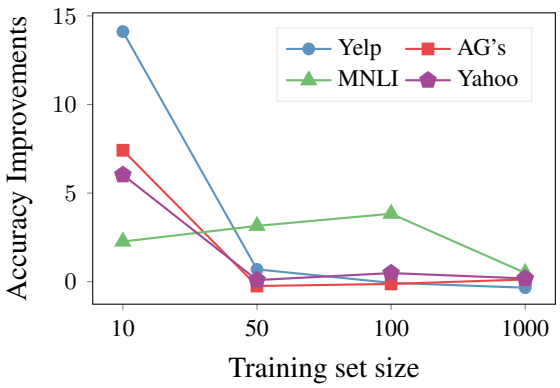

Figure 3: Accuracy improvements for PET due to adding L_{MLM} during training

5 Analysis

Combining PVPs We first investigate whether PET is able to cope with situations were some PVPs perform much worse than others. For $\lvert\mathcal{T}\rvert = 10$, Table 4 compares the performance of PET to that of the best and worst performing patterns after finetuning; we also include results obtained using the ensemble of PET models corresponding to individual PVPs without knowledge distillation. Even after finetuning, the gap between the best and worst pattern is large, especially for Yelp. However, PET is not only able to compensate for this, but even improves accuracies over using only the best-performing pattern across all tasks. Distillation brings consistent improvements over the ensemble; additionally, it significantly reduces the size of the

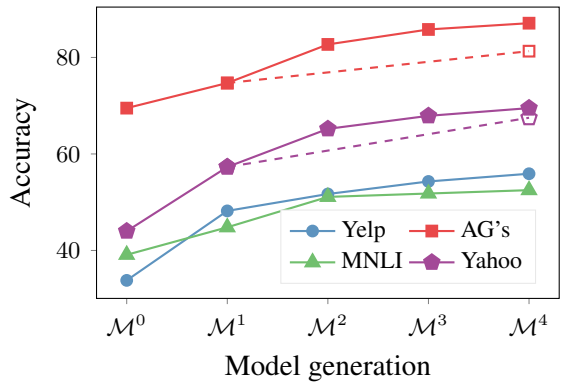

Figure 4: Average accuracy for each generation of models with iPET in a zero-shot setting. Accuracy on AG's News and Yahoo when skipping generation 2 and 3 is indicated through dashed lines.

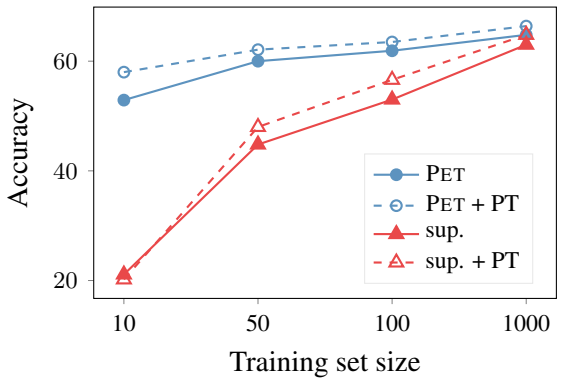

Figure 5: Accuracy of supervised learning (sup.) and PET both with and without pretraining (PT) on Yelp

final classifier. We find no clear difference between the uniform and weighted variants of PET.

Auxiliary Language Modeling We analyze the influence of the auxiliary language modeling task on PET's performance. Figure 3 shows performance improvements from adding the language modeling task for four training set sizes. We see that the auxiliary task is extremely valuable when training on just 10 examples. With more data, it becomes less important, sometimes even leading to worse performance. Only for MNLI, we find language modeling to consistently help.

Iterative PET To check whether iPET is able to improve models over multiple generations, Figure 4 shows the average performance of all generations of models in a zero-shot setting. Each additional iteration does indeed further improve the ensemble's performance. We did not investigate whether continuing this process for even more iterations gives further improvements.

Another natural question is whether similar results can be obtained with fewer iterations by increasing the training set size more aggressively. To answer this question, we skip generations 2 and 3 for AG's News and Yahoo and for both tasks directly let ensemble $\mathcal{M}^1$ annotate $10 \cdot 5^4$ examples for $\mathcal{M}^4$. As indicated in Figure 4 through dashed lines, this clearly leads to worse performance, highlighting the importance of only gradually increasing the training set size. We surmise that this is the case because annotating too many examples too early leads to a large percentage of mislabeled training examples.

In-Domain Pretraining Unlike our supervised baseline, PET makes use of the additional unlabeled dataset $\mathcal{D}$. Thus, at least some of PET's performance gains over the supervised baseline may arise from this additional in-domain data.

To test this hypothesis, we simply further pretrain RoBERTa on in-domain data, a common technique for improving text classification accuracy (e.g., Howard and Ruder, 2018; Sun et al., 2019). As language model pretraining is expensive in terms of GPU usage, we do so only for the Yelp dataset. Figure 5 shows results of supervised learning and PET both with and without this in-domain pretraining. While pretraining does indeed improve accuracy for supervised training, the supervised model still clearly performs worse than PET, showing that the success of our method is not simply due to the usage of additional unlabeled data. Interestingly, in-domain pretraining is also helpful for PET, indicating that PET leverages unlabeled data in a way that is clearly different from standard masked language model pretraining.

6 Conclusion

We have shown that providing task descriptions to pretrained language models can be combined with standard supervised training. Our proposed method, PET, consists of defining pairs of cloze question patterns and verbalizers that help leverage the knowledge contained within pretrained language models for downstream tasks. We finetune models for all pattern-verbalizer pairs and use them to create large annotated datasets on which standard classifiers can be trained. When the initial amount of training data is limited, PET gives large improvements over standard supervised training and strong semi-supervised approaches.

Acknowledgments

This work was funded by the European Research Council (ERC #740516). We would like to thank the anonymous reviewers for their helpful comments.

References

Eugene Agichtein and Luis Gravano. 2000. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, DL '00, page 85–94, New York, NY, USA. Association for Computing Machinery.

David S. Batista, Bruno Martins, and Mário J. Silva. 2015. Semi-supervised bootstrapping of relationship extractors with distributional semantics. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 499–504, Lisbon, Portugal. Association for Computational Linguistics.

Zied Bouraoui, Jose Camacho-Collados, and Steven Schockaert. 2020. Inducing relational knowledge from BERT. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Sergey Brin. 1999. Extracting patterns and relations from the world wide web. In *The World Wide Web and Databases*, pages 172–183, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jiaao Chen, Zichao Yang, and Diyi Yang. 2020. MixText: Linguistically-informed interpolation of hidden space for semi-supervised text classification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2147–2157, Online. Association for Computational Linguistics.

Alexandra Chronopoulou, Christos Baziotis, and Alexandros Potamianos. 2019. An embarrassingly simple approach for transfer learning from pretrained language models. 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 2089–2095, Minneapolis, Minnesota. Association for Computational Linguistics.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Joe Davison, Joshua Feldman, and Alexander Rush. 2019. Commonsense knowledge mining from pretrained models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1173–1178, Hong Kong, China. Association for Computational Linguistics.

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.

Zi-Yi Dou, Keyi Yu, and Antonios Anastasopoulos. 2019. Investigating meta-learning algorithms for low-resource natural language understanding tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1192–1197, Hong Kong, China. Association for Computational Linguistics.

Allyson Ettinger. 2020. What BERT is not: Lessons from a new suite of psycholinguistic diagnostics for language models. *Transactions of the Association for Computational Linguistics*, 8:34–48.

Jiatao Gu, Yong Wang, Yun Chen, Victor O. K. Li, and Kyunghyun Cho. 2018. Meta-learning for low-resource neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3622–3631, Brussels, Belgium. Association for Computational Linguistics.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1321–1330. JMLR.org.

Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. 2020. Revisiting self-training for neural sequence generation. In *International Conference on Learning Representations*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *Computing Research Repository*, arXiv:1503.02531.

Vu Cong Duy Hoang, Philipp Koehn, Gholamreza Haffari, and Trevor Cohn. 2018. Iterative back-translation for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 18–24, Melbourne, Australia. Association for Computational Linguistics.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1:*

Long Papers), pages 328–339, Melbourne, Australia. Association for Computational Linguistics.

Zhongqiang Huang and Mary Harper. 2009. Self-training PCFG grammars with latent annotations across languages. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 832–841, Singapore. Association for Computational Linguistics.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

Nora Kassner and Hinrich Schütze. 2020. Negated and misprimed probes for pretrained language models: Birds can talk, but cannot fly. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7811–7818, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *Computing Research Repository*, arXiv:1907.11692.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *Computing Research Repository*, arXiv:1806.08730.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159, New York City, USA. Association for Computational Linguistics.

Juri Opitz. 2019. Argumentative relation classification as plausibility ranking. In *Preliminary proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019): Long Papers*, pages 193–202, Erlangen, Germany. German Society for Computational Linguistics & Language Technology.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Raul Puri and Bryan Catanzaro. 2019. Zero-shot text classification with generative language models. *Computing Research Repository*, arXiv:1912.10165.

Kun Qian and Zhou Yu. 2019. Domain adaptive dialog generation via meta learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2639–2649, Florence, Italy. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Roi Reichart and Ari Rappoport. 2007. Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 616–623, Prague, Czech Republic. Association for Computational Linguistics.

Bernardino Romera-Paredes and Philip Torr. 2015. An embarrassingly simple approach to zero-shot learning. In *International Conference on Machine Learning*, pages 2152–2161.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Timo Schick and Hinrich Schütze. 2020. Rare words: A major problem for contextualized embeddings and how to fix it by attentive mimicking. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Shashank Srivastava, Igor Labutov, and Tom Mitchell. 2018. Zero-shot learning of classifiers from natural language quantification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 306–316, Melbourne, Australia. Association for Computational Linguistics.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune BERT for text classification? In *Chinese Computational Linguistics*, pages 194–206, Cham. Springer International Publishing.

Alon Talmor, Yanai Elazar, Yoav Goldberg, and Jonathan Berant. 2019. oLMpics – on what language model pre-training captures. *Computing Research Repository*, arXiv:1912.13283.

Trieu H. Trinh and Quoc V. Le. 2018. A simple method for commonsense reasoning. *Computing Research Repository*, arXiv:1806.02847.

Jannis Vamvas and Rico Sennrich. 2020. X-stance: A multilingual multi-target dataset for stance detection. *Computing Research Repository*, arXiv:2003.08385.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Sappadla Prateek Veeranna, Jinseok Nam, Eneldo Loza Mencıa, and Johannes Fürnkranz. 2016. Using semantic similarity for multi-label zero-shot classification of text documents. In *Proceeding of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges, Belgium: Elsevier*, pages 423–428.

Cunxiang Wang, Shuailong Liang, Yue Zhang, Xiaonan Li, and Tian Gao. 2019. Does it make sense? And why? A pilot study for sense making and explanation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4020–4026, Florence, Italy. Association for Computational Linguistics.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. 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.

Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. 2020. Unsupervised data augmentation for consistency training. In *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems*, volume 32, pages 5753–5763. Curran Associates, Inc.

David Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Zhiquan Ye, Yuxia Geng, Jiaoyan Chen, Jingmin Chen, Xiaoxiao Xu, SuHang Zheng, Feng Wang, Jun Zhang, and Huajun Chen. 2020. Zero-shot text classification via reinforced self-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3014–3024, Online. Association for Computational Linguistics.

Wenpeng Yin, Jamaal Hay, and Dan Roth. 2019. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3914–3923, Hong Kong, China. Association for Computational Linguistics.

Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang, and Bowen Zhou. 2018. Diverse few-shot text classification with multiple metrics. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1206–1215, New Orleans, Louisiana. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 649–657. Curran Associates, Inc.

A Implementation

Our implementation of PET and iPET is based on the Transformers library (Wolf et al., 2020) and PyTorch (Paszke et al., 2017).

B Training Details

Except for the in-domain pretraining experiment described in Section 5, all of our experiments were conducted using a single GPU with 11GB RAM (NVIDIA GeForce GTX 1080 Ti).

B.1 Hyperparameter Choices

Relevant training hyperparameters for both individual PET models and the final classifier C as well as our supervised baseline are listed in Table 5. All hyperparameters were selected based on the following considerations and experiments:

Batch size / maximum length Both batch size and maximum sequence length (or block size) are chosen so that one batch fits into 11GB of GPU memory. As Devlin et al. (2019) and Liu et al. (2019) use larger batch sizes of 16–32, we accumulate gradients for 4 steps to obtain an effective batch size of 16.

Learning rate We found a learning rate of $5\text{e}{-5}$ (as used by Devlin et al. (2019)) to often result in unstable training for regular supervised learning with no accuracy improvements on the training set. We therefore use a lower learning rate of $1\text{e}{-5}$, similar to Liu et al. (2019). Experiments with various learning rates can be found in Appendix D.

Training steps As the number of training epochs recommended by Liu et al. (2019) in a data-rich scenario is in the range 2–10, we perform supervised training for 250 training steps, corresponding to 4 epochs when training on 1000 examples. For individual PET models, we subdivide each batch into one labeled example from $\mathcal{T}$ to compute L_{CE} and three unlabeled examples from $\mathcal{D}$ to compute L_{MLM}. Accordingly, we multiply the number of total training steps by 4 (i.e., 1000), so that the number of times each labeled example is seen remains constant ($16 \cdot 250 = 4 \cdot 1000$). For the final PET classifier, we train for 5000 steps due to the increased training set size (depending on the task, the unlabeled set $\mathcal{D}$ contains at least 20 000 examples). Deviating from the above, we always perform training for 3 epochs on x-stance to match the setup of Vamvas and Sennrich (2020) more closely. The effect of varying the number of training steps is further investigated in Appendix D.

Temperature We choose a temperature of 2 when training the final classifier following Hinton et al. (2015).

Auxiliary language modeling To find a suitable value of α for combining language modeling loss and cross-entropy loss, we first observed that in the early stages of training, the former is a few orders of magnitude higher than the latter for all tasks considered. We thus selected a range $\{1\text{e}{-3}, 1\text{e}{-4}, 1\text{e}{-5}\}$ of reasonable choices for α and performed preliminary experiments on Yelp with 100 training examples to find the best value among these candidates. To this end, we split the training examples into a training set and a dev set using both a 90/10 split and a 50/50 split and took the value of α that maximizes average dev set accuracy. We adopt this value for all other tasks and training set sizes without further optimization.

Models per ensemble As we always train three models per pattern, for both iPET and training the final classifier C, the ensemble $\mathcal{M}$ (or $\mathcal{M}^0$) for n PVPs contains $3n$ models. This ensures consistency as randomly choosing any of the three models for each PVP would result in high variance. In preliminary experiments, we found this to have only little impact on the final model's performance.

iPET dataset size For iPET, we quintuple the number of training examples after each iteration ($d = 5$) so that only a small number of generations is required to reach a sufficient amount of labeled data. We did not choose a higher value because we presume that this may cause training sets for early generations to contain a prohibitively large amount of mislabeled data.

iPET dataset creation We create training sets for the next generation in iPET using 25% of the models in the current generation ($\lambda = 0.25$) because we want the training sets for all models to be diverse while at the same time, a single model should not have too much influence.

Others For all other hyperparameters listed in Table 5, we took the default settings of the Transformers library (Wolf et al., 2020).

B.2 Number of parameters

As PET does not require any additional learnable parameters, the number of parameters for both PET

and iPET is identical to the number of parameters in the underlying language model: 355M for RoBERTa (large) and 270M for XLM-R (base).

B.3 Average runtime

Training a single PET classifier for 250 steps on one GPU took approximately 30 minutes; training for 1000 steps with auxiliary language modeling took 60 minutes. Depending on the task, labeling examples from $\mathcal{D}$ took 15–30 minutes per model. Training the final classifier C for 5000 steps on the soft-labeled dataset $\mathcal{T}_C$ took 2 hours on average.

B.4 Comparison with SotA

For comparing PET to UDA (Xie et al., 2020) and MixText (Chen et al., 2020), we reduce the number of unlabeled examples by half to speed up the required backtranslation step. We use the backtranslation script provided by Chen et al. (2020) with their recommended hyperparameter values and use both Russian and German as intermediate languages.

For MixText, we use the original implementation[5] and the default set of hyperparameters. Specifically, each batch consists of 4 labeled and 8 unlabeled examples, we use layers 7, 9 and 12 for mixing, we set $T = 5$, $\alpha = 16$, and use a learning rate of $5 \cdot 10^{-6}$ for RoBERTa and $5 \cdot 10^{-4}$ for the final classification layer. We optimize the number of training steps for each task and dataset size in the range $\{1000, 2000, 3000, 4000, 5000\}$.

For UDA, we use a PyTorch-based reimplementation[6]. We use the same batch size as for MixText and the hyperparameter values recommended by Xie et al. (2020); we use an exponential schedule for training signal annealing and a learning rate of $2 \cdot 10^{-5}$. We optimize the number of training steps for each task and dataset size in the range $\{500, 1000, 1500, \ldots, 10000\}$.

B.5 In-Domain Pretraining

For in-domain pretraining experiments described in Section 5, we use the language model finetuning script of the Transformers library (Wolf et al., 2020); all hyperparameters are listed in the last column of Table 5. Pretraining was performed on a total of 3 NVIDIA GeForce GTX 1080 Ti GPUs.

[5] https://github.com/GT-SALT/MixText

[6] https://github.com/SanghunYun/UDA_pytorch

C Dataset Details

For each task and number of examples t, we create the training set $\mathcal{T}$ by collecting the first $t/|\mathcal{L}|$ examples per label from the original training set, where $|\mathcal{L}|$ is the number of labels for the task. Similarly, we construct the set $\mathcal{D}$ of unlabeled examples by selecting 10 000 examples per label and removing all labels. For evaluation, we use the official test set for all tasks except MNLI, for which we report results on the dev set; this is due to the limit of 2 submissions per 14 hours for the official MNLI test set. An overview of the number of test examples and links to downloadable versions of all used datasets can be found in Table 6.

Preprocessing In some of the datasets used, newlines are indicated through the character sequence "\n". As the vocabularies of RoBERTa and XLM-R do not feature a newline, we replace this sequence with a single space. We do not perform any other preprocessing, except shortening all examples to the maximum sequence length of 256 tokens. This is done using the *longest first* strategy implemented in the Transformers library. For PET, all input sequences are truncated *before* applying patterns.

Evaluation metrics For Yelp, AG's News, Yahoo and MNLI, we use accuracy. For x-stance, we report macro-average of F1 scores using the evaluation script of Vamvas and Sennrich (2020).

D Hyperparameter Importance

To analyze the importance of hyperparameter choices for PET's performance gains over supervised learning, we look at the influence of both the learning rate (LR) and the number of training steps on their test set accuracies.

We try values of $\{1e{-}5, 2e{-}5, 5e{-}5\}$ for the learning rate and $\{50, 100, 250, 500, 1000\}$ for the number of training steps. As this results in 30 different configurations for just one task and training set size, we only perform this analysis on Yelp with 100 examples, for which results can be seen in Figure 6. For supervised learning, the configuration used throughout the paper (LR $= 1e{-}5$, 250 steps) turns out to perform best whereas for PET, training for fewer steps consistently performs even better. Importantly, PET clearly outperforms regular supervised training regardless of the chosen learning rate and number of training steps.

Parameter	PET −LM	PET (En/Xs)	C (En/Xs)	sup. (En/Xs)	In-Dom. PT
adam_epsilon	1e-8	1e-8	1e-8	1e-8	1e-8
* alpha	–	1e-4	–	–	–
block_size	–	–	–	–	256
gradient_accumulation_steps	4	4	4	4	2
learning_rate	1e-5	1e-5	1e-5	1e-5	5e-5
max_grad_norm	1.0	1.0	1.0	1.0	1.0
max_seq_length	256	256	256	256	–
max_steps	250	1000 / –	5000 / –	250 / –	50000
mlm_probability	–	0.15	–	–	0.15
num_train_epochs	–	– / 3	– / 3	– / 3	–
per_gpu_train_batch_size	4	1	4	4	2
* per_gpu_helper_batch_size	–	3	–	–	–
* temperature	–	–	2.0	–	–
weight_decay	0.01	0.01	0.01	0.01	0.0

Table 5: Hyperparameters for training individual PET models without auxiliary language modeling (PET−LM) and with language modeling (PET), the final PET classifier (C), regular supervised training (sup.) and in-domain pretraining (In-Dom. PT). Whenever different values are used for the English datasets (En) and x-stance (Xs), both values are given separated by a slash. (*): PET-specific hyperparameters

Dataset	**Link**	**Test Examples**
AG's News	http://goo.gl/JyCnZq	7600
MNLI (m / mm)	https://cims.nyu.edu/~sbowman/multinli/	10000 / 10000
X-Stance (De / Fr / It)	https://github.com/ZurichNLP/xstance	3479 / 1284 / 1173
Yahoo! Answers	http://goo.gl/JyCnZq	60000
Yelp Review Full	http://goo.gl/JyCnZq	50000

Table 6: Download links and number of test examples for all datasets

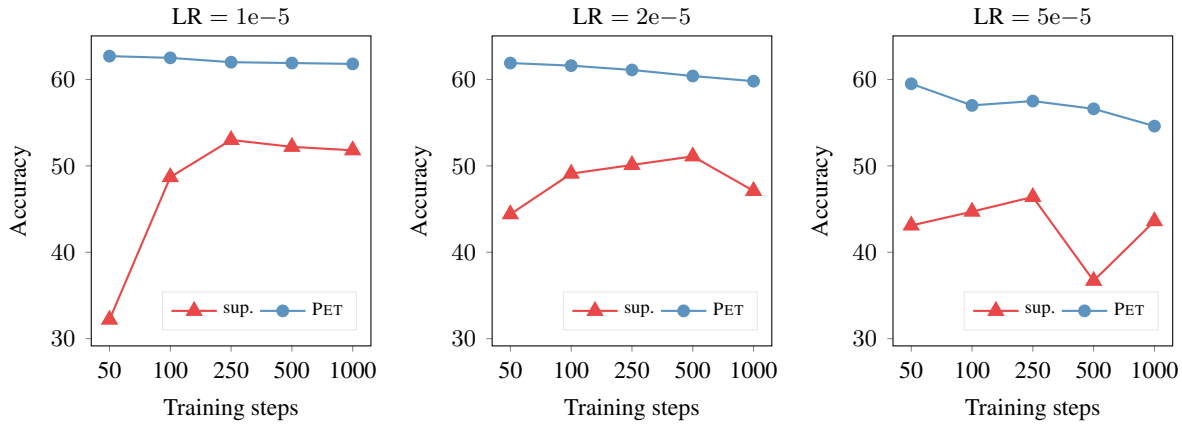

Figure 6: Performance of supervised learning and PET (weighted, without auxiliary language modeling) for various learning rates and training steps on Yelp with 100 training examples

E Automatic Verbalizer Search

Given a set of patterns $P_1, \ldots, P_n$, manually finding a verbalization $v(l)$ for each $l \in \mathcal{L}$ that represents the meaning of l well and corresponds to a single token in V can be difficult. We therefore devise *automatic verbalizer search* (AVS), a procedure that automatically finds suitable verbalizers given a training set $\mathcal{T}$ and a language model M.

Assuming we already have a PVP $\mathbf{p} = (P, v)$, we can easily check whether some token $t \in V$ is a good verbalization of $l \in \mathcal{L}$. To this end, we define $\mathbf{p}[l \leftarrow t] = (P, v')$, where v' is identical to v, except that $v'(l) = t$. Intuitively, if t represents l well, then $q_{\mathbf{p}[l \leftarrow t]}(l \mid \mathbf{x})$ (i.e., the probability M assigns to t given $P(\mathbf{x})$) should be high only for those examples $(\mathbf{x}, y) \in \mathcal{T}$ where $y = l$. We thus define the score of t for l given $\mathbf{p}$ as

$$s_l(t \mid \mathbf{p}) = \frac{1}{|\mathcal{T}_l|} \cdot \sum_{(\mathbf{x},y) \in \mathcal{T}_l} q_{\mathbf{p}[l \leftarrow t]}(l \mid \mathbf{x}) - \frac{1}{|\mathcal{T} \setminus \mathcal{T}_l|} \cdot \sum_{(\mathbf{x},y) \in \mathcal{T} \setminus \mathcal{T}_l} q_{\mathbf{p}[l \leftarrow t]}(l \mid \mathbf{x})$$

where $\mathcal{T}_l = \{(\mathbf{x}, y) \in \mathcal{T} : y = l\}$ is the set of all training examples with label l. While this allows us to easily compute the best verbalization for l as

$$\hat{t} = \arg\max_{t \in V} s_l(t \mid \mathbf{p}),$$

it requires us to already know verbalizations $v(l')$ for all other labels l'.

AVS solves this problem as follows: We first assign random verbalizations to all labels and then repeatedly recompute the best verbalization for each label. As we do not want the resulting verbalizer to depend strongly on the initial random assignment, we simply consider multiple such assignments. Specifically, we define an initial probability distribution ρ_0 where for all $t \in V, l \in \mathcal{L}$, $\rho_0(t \mid l) = 1/|V|$ is the probability of choosing t as verbalization for l. For each $l \in \mathcal{L}$, we then sample k verbalizers $v_1, \ldots, v_k$ using ρ_0 to compute

$$s_l^k(t) = \frac{1}{n \cdot k} \sum_{i=1}^{n} \sum_{j=1}^{k} s_l(t \mid (P_i, v_j))$$

for all $t \in V$.[7] These scores enable us to define a probability distribution ρ_1 that more closely reflects a word's suitability as a verbalizer for a given label:

$$\rho_1(t \mid l) = \frac{1}{Z} \max(s_l^k(t), \epsilon)$$

where $Z = \sum_{t' \in V} \max(s_l^k(t'), \epsilon)$ and $\epsilon \geq 0$ ensures that ρ_1 is a proper probability distribution. We repeat this process to obtain a sequence of probability distributions $\rho_1, \ldots, \rho_{i_{\max}}$. Finally, we choose the $m \in \mathbb{N}$ most likely tokens according to $\rho_{i_{\max}}(t \mid l)$ as verbalizers for each l. During training and inference, we compute the unnormalized score $s_{\mathbf{p}}(y \mid \mathbf{x})$ for each label by averaging over its m verbalizers.

We analyze the performance of AVS for all tasks with $|\mathcal{T}| = 50$ training examples and set $k = 250$, $\epsilon = 10^{-3}$, $i_{\max} = 5$ and $m = 10$.[8] To speed up the search, we additionally restrict our search space to tokens $t \in V$ that contain at least two alphabetic characters. Of these tokens, we only keep the 10 000 most frequent ones in $\mathcal{D}$.

Results are shown in Table 7. As can be seen, carefully handcrafted verbalizers perform much better than AVS; however, PET with AVS still considerably outperforms regular supervised training while eliminating the challenge of manually finding suitable verbalizers. Table 8 shows the most probable verbalizers found using AVS for the Yelp dataset. While most verbalizers for this dataset intuitively make sense, we found AVS to struggle with finding good verbalizers for three out of ten labels in the Yahoo dataset and for all MNLI labels.

	Yelp	AG's	Yahoo	MNLI
supervised	44.8	82.1	52.5	45.6
PET	**60.0**	**86.3**	**66.2**	**63.9**
PET + AVS	55.2	85.0	58.2	52.6

Table 7: Results for supervised learning, PET and PET with AVS (PET + AVS) after training on 50 examples

y	Top Verbalizers
1	worthless, BAD, useless, appalling
2	worse, slow, frustrating, annoying
3	edible, mixed, cute, tasty, Okay
4	marvelous, loved, love, divine, fab
5	golden, magical, marvelous, perfection

Table 8: Most probable verbalizers according to AVS for Yelp with 50 training examples

[7] Note that the score $s_l^k(t)$ jointly considers all patterns; in preliminary experiments, we found this to result in more robust verbalizers.

[8] We tried values of k and $i_{\max}$ in $\{250, 500, 1000\}$ and $\{5, 10, 20\}$, respectively, but found the resulting verbalizers to be almost identical.