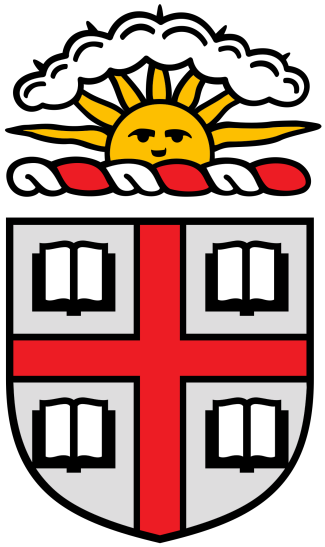
Brown University

RIPPED: Recursive Intent Propagation using Pretrained Embedding Distances

A framework for bootstrapping natural language understanding in data-poor domains



by
Michael Ball

A thesis submitted in partial fulfillment for the degree of
Bachelor of Science in Computer Science with Honors.

in the
Department of Computer Science

Advisor: Michael L. Littman; Reader: Stephen H. Bach

April, 2019

Contents

Abstract

In this paper, we introduce a novel framework for improving intent classification performance in domains with limited labeled data. We call our framework RIPPED: Recursive Intent Propagation using Pretrained Embedding Distances. Unlike most graph-based semi-supervised approaches, RIPPED uses dense pretrained embeddings to construct the representation graph. In combining this representation scheme with a recursive variant of the label propagation algorithm, RIPPED is able to accurately propagate labels throughout the unlabeled dataset in domains with a large number of unbalanced classes and complex, noisy decision boundaries. In a given data-poor domain, RIPPED acts as an augmentation system, adding to the labeled dataset by classifying unlabeled examples, thus allowing a more effective inductive classifier to be trained. As a result, RIPPED can be easily incorporated into any classification pipeline.

RIPPED is simple to apply to new domains, and our results indicate its empirical effectiveness. On four intent classification datasets, given access to only a few labeled examples per class, RIPPED achieved performance comparable to state-of-the-art classifiers given access to the entire training dataset. In some cases (including the one-shot setting), RIPPED outperformed the next-best semi-supervised methods by more than 70%. We propose that RIPPED can be used as an out-of-the-box tool for bootstrapping natural language understanding systems in data-poor domains.

Acknowledgements

First and foremost, I'd like to thank my advisor, Michael Littman for his guidance and sensible advice. Michael, your ability to clarify the mess in my head allowed the ideas to flow, and I thank you for reminding me in the moments of chaos that it's all really just fun and games.

To Stephen Bach, thank you for your pedagogy and masterful management of discussion in introducing me to the world of limited labeled data. Many aspects of this work were inspired by tangential discussions only made possible by your support of intellectual curiosity.

I want to thank my partner Lauren, my constant source of inspiration. Her limitless patience and loving encouragement are what made this all possible.

I am also grateful for the support of my family and friends here and in New Zealand. Special thanks to my mum Kate and dad Craig for their relentless encouragement, and to Nick, Phil, and Jae for telling me to go outside.

I am very fortunate to have met a mentor along the way in Nicolas Santini who took a chance and helped me grow. And to all my colleagues at Ambit, I am grateful for your constant positivity and determination to make things happen.

Finally, I want to thank Brown. The last four years have been a whirlwind, all thanks to the people whose paths I've been lucky enough to cross. I am forever grateful for the friends I've made and the experiences I've shared in this incredible melting-pot.

List of Figures

List of Tables

1 Introduction

The desire to build a computer capable of holding human-level conversation has been ingrained in artificial intelligence research since the field's inception; in 1950, Alan Turing proposed this challenge as the key criterion for assessing a machine's intelligence (Turing, 1950). In recent years, the combination of online chat service proliferation and relentless advances in natural language processing have stoked a resurgence of interest in conversational agents, or chatbots. Evidence of the recent focus can be seen both in academic literature (Ferrara et al., 2016; Vinyals and Le, 2015; Mazaré et al., 2018) and in industry[1,2] (Zhou et al., 2018).

At the core of conversational intelligence is natural language understanding (NLU), the ability to extract important semantic information from raw sequences of text (or audio). The first step in this process is intent classification (IC), a subset of text classification in which short sequences of text (utterances) are classified according to the action or dialogue the utterance is intended to prompt. Assuming the purpose of language in a conversation is to communicate information, we can view the semantic meaning of an utterance as the result it is intended to have on the recipient. In this intuitive framing, intents are simply collections of utterances with similar meanings.

The IC task has some key challenges that we believe to be under-explored:

1. In building NLU systems for new domains, labeled data is often very limited. By contrast, unlabeled data is generally accessible, for example, by parsing human-to-human chat logs or scraping unstructured text online
2. Many domains in which NLU systems are desired contain complex, domain-specific language (for example, in building a medical chatbot or a virtual legal assistant).
3. IC is generally a *many*-class problem, as a large number of intents are needed to define the scope of conversation in even a heavily-restricted domain
4. Intents are defined purely by semantics, independent of syntactic or lexical features. This contrasts with many general text classification problems, where classes are often determined by high-level themes or stylistic features.

In light of the first three challenges and motivated by the need to develop more effective techniques for bootstrapping NLU in new domains, we are especially interested in the few-shot IC setting, where we only have access to a few labeled examples per class. The combination of limited labeled data and relatively prevalent unlabeled data lends this setting to semi-supervised learning (SSL), the broad collection of learning strategies that make use of the structure of unlabeled data in addition to the few given labels. Many SSL approaches to text classification have been proposed (Wang et al., 2013, 2016), demonstrating varying levels of success in a variety of classification domains. But there are two unifying problems with these methods. Firstly, they often still require a substantial amount of labeled data to be successful, and are therefore unsuitable for the few-shot setting. Secondly, they are typically unable to handle many-class problems, especially those with complex, noisy decision boundaries.

One possible solution to the latter problem could be the use of neural transfer learning, the transferring of linguistic knowledge from models trained on large corpora to the specific task at hand, perhaps improving the ability of SSL approaches to handle semantically complex problem spaces. A variety of such models and strategies have been suggested in recent years that both improve performance on text classification tasks and reduce the number of labeled examples required to train

[1] `https://www.drift.com/chatbots/`

[2] Google trends reports a 10-fold increase in searches for 'chatbot' since the end of 2015

classifiers in downstream domains (Ruder, 2019). However, neither the incorporation of features extracted from pretrained representations nor the fine-tuning of entire pretrained models is feasible when faced with only a few labeled examples per class, especially in the many-class setting.

In this paper, we outline a framework for substantially improving few-shot intent classification by proposing RIPPED: **R**ecursive **I**ntent **P**ropagation using **P**retrained **E**mbedding **D**istances. Our method is designed specifically to address the aforementioned challenges of the intent classification task, and as such it outlines a framework for bootstrapping NLU in data-poor domains. RIPPED combines a variant of pretrained feature extraction specific to limited labeled data domains with a recursive extension of the semi-supervised label propagation algorithm. The method is an inductive approach that expands the labeled dataset by classifying a subset of the unlabeled examples, thus allowing a more effective classifier to be trained than would have been possible with access to only the original labeled data.

The principal contributions of this paper are as follows:

- We emphasize that pairwise distance functions between pretrained sentence representations are able to reflect sophisticated semantic relations, and use this to demonstrate that the knowledge contained in these distance values, computed over embeddings untuned to the specific downstream task at hand, is able to accurately represent the semantic structure of the intent classes in a number of IC domains.
- We show that our recursive extension of the label propagation algorithm significantly outperforms other semi-supervised algorithms in many-class, few-shot classification domains, especially in tasks with noisy decision boundaries.
- On four benchmark IC datasets, using only a few labeled examples per class, we achieve comparable performance to state-of-the-art supervised methods given access to the entire training dataset. We conduct experiments along a number of axes to investigate the relative importance of the key components of RIPPED, demonstrating the robustness of our method.
- We present RIPPED as a general framework for boostrapping NLU in novel domains. The method acts as an augmentation tool, applied to the task at hand prior to classifier training, and as such it can be incorporated into any IC pipeline. Code for our model can be found on the author's Github.[3].

The remainder of the document proceeds as follows. In §2 we give an overview of work related to our work, thereby framing our motivation for this research. We outline some necessary theoretical background in §3, before giving a comprehensive account of our proposed method in §4. In §5 and §6 we describe and present the results of our experiments, before discussing some additional takeaways in §7. We add some concluding remarks in §8, and include a collection of supplementary information in §10.

[3] `https://github.com/michaelhball/Recursive-Intent-Propagation-using-Pretrained-Embedding-Distances`. Full documentation and a packaged library are forthcoming.

2 Related Work

The principal intention of our research was to develop a system for improving IC performance in domains with limited labeled data. There are a variety of related approaches that share that goal, though most target text classification rather than the more specific IC. The two most relevant categories of techniques are semi-supervised learning (SSL) and transfer learning, outlined in §2.1 and §2.2 respectively. We also describe several related approaches that specifically target few-shot text classification in §2.3. Throughout this section, we reference the drawbacks of the approaches discussed relative to their application in our specific setting. These are not intended as criticisms of the work – the mentioned approaches were generally not designed for this domain in the first place – but rather as motivating examples of the problems we sought to address in designing RIPPED.

2.1 Semi-Supervised Learning

Semi-supervised learning can be broadly divided into three main approaches: generative models, low-density separation models, and graph-based models. The seminal generative approach is outlined by (Nigam et al., 2006), whereby a bag-of-words text representation is combined with various expectation-maximisation algorithms[4] to propagate labels. A major drawback of generative approaches is that they struggle with local maxima when dealing with many-class problems and domains with complex decision boundaries (Liu et al., 2002).

Low-density separation models typically construct a structured representation of the input domain, before using some variant of the Transductive Support Vector Machine algorithm (TSVM; Gammerman et al., 1998) to label the unlabeled data such that the decision boundary margin is maximised over the entire dataset. Three previously state-of-the-art examples are (Chapelle and Zien, 2005; Sindhwani and Keerthi, 2006; Keerthi et al., 2012), where the latter is the most relevant to our work as it extends the standard TSVM algorithm to the multi-class setting. These approaches typically operate over sparse representations of text, as SVMs in general are especially well-suited to this setting (Joachims, 1998). Although some work has been done towards improving these methods' robustness (Li and Zhou, 2011), TSVM-based approaches suffer from inconsistent performance: the use of unlabeled examples sometimes degrades performance when compared with a supervised SVM, often in ways that are unpredictable. These issues are exacerbated in the many-class setting, we believe due to the inability of sparse representations to capture the complex semantics of the representation space.

Our proposed method is a graph-based approach that extends the original label propagation (LP) algorithm outlined by (Zhu, 2002). An approach similarly motivated to ours is the dynamic LP variant of (Wang et al., 2013), also designed to extend LP to multi-class problems (though DLP operates in the transductive setting). Other related approaches in this area are (Yang et al., 2016), which learns graph embeddings over the representation graph to understand neighbourhood context, and (Pawar et al., 2016) which combines LDA (Pritchard et al., 2000) with LP to perform weakly-supervised text classification. The key difference between RIPPED and these related approaches is the choice of text representation (pretrained dense representations vs. sparse, bag-of-words-based representations). The representation method used in graph-based approaches is especially important, as (Jebara et al., 2009) indicates that strong classification performance is more dependent on robust graph construction than a sophisticated propagation algorithm.

A number of recent proposals incorporate the representational power of neural networks (NNs) into the semi-supervised learning process. For example, (Wang et al., 2016) suggest an iterative algorithm

[4]We refer the reader to Andrew Ng's excellent lecture on the fundamentals of EM: `http://cs229.stanford.edu/notes/cs229-notes8.pdf`

to optimise an objective function combining k-means and the learning of sequence embeddings, (Johnson and Zhang, 2015) train a CNN using both supervised learning over the labeled examples and unsupervised learning over the unlabeled examples, and (Johnson and Zhang, 2016) use a similar unsupervised CNN in conjunction with an LSTM trained to represent 'region embeddings.' While all three methods surpass the performance of a number of semi-supervised baselines on several classification tasks, they require a relatively large number of labeled examples to be successful. For example, the embedding method of (Wang et al., 2016) achieves state-of-the-art performance on the TREC dataset (Li and Roth, 2002) when given 10% of the labels (550 examples over 6 classes), but classification performance plummets (from an adjusted mutual index of 43 to 12) when this is reduced to 1% of the labels (55 examples).

2.2 Transfer Learning

We take transfer learning here to refer to the definition of sequential transfer learning given in (Ruder, 2019): "the setting where source and target tasks are different and training is performed in sequence." In general, this encompasses the entire collection of recent advances in using large corpora to learn universal representations that can be incorporated downstream both to improve performance and to expedite training. Therefore, these approaches are especially useful in limited labeled data domains. One particularly successful collection of transfer-learning approaches are pretrained language models (Radford, 2018; Ruder and Howard, 2018; Chronopoulou et al., 2019), whereby general linguistic representations are trained on millions of tokens of text. Almost every benchmark in NLP has been advanced in recent years through the incorporation of pretrained language models in downstream architectures. Universal embedding methods such as BERT (Devlin et al., 2018) and ELMo (Peters et al., 2018) also fall under the umbrella of transfer learning, as these can be incorporated into downstream architectures either out-of-the-box or with fine-tuning.

The main drawback of these transfer learning approaches in the context of our intended domain is that in very data-limited settings there often isn't enough data to either train a classifier using features extracted from pretrained embeddings or to fine-tune an entire pretrained model. For example, ULMFiT was unable to achieve a validation error below 35% when fine-tuned on the six-class TREC dataset with 16 examples per class (Ruder and Howard, 2018). Even training a binary, linear classifier using the smallest pretrained embeddings (generally 50 dimensions) requires training 50 parameters, a challenging task if we only have a few labeled examples in each class.

2.3 Few-Shot Learning

A variety of approaches have been proposed for few-shot learning in text classification. (Bailey and Chopra, 2018) incorporate topic modelling and a human-in-the-loop to select the 'most representative' labeled examples in each class, before the remaining examples are classified using simple distance calculations. As a transfer learning-related approach to the few-shot setting, (Pushp and Srivastava, 2017) propose learning a collection of binary tag classifiers on a large corpora. To perform text classification in a new domain, these tags are mapped to classes, and then a thresholded vote is used to determine the classification of each unlabeled example. (Yu et al., 2018) also suggest training models on general tasks before adapating these to the target domain, in their case by considering 'meta-training tasks.' These works all report strong results on a variety of classification tasks, though they don't experiment in the many-class setting.

3 Background

This section provides a theoretical overview of key concepts, algorithms, and models that our research uses and extends.

3.1 Intent Classification

RIPPED is designed for use in intent classification (IC), a subset of text classification where the class of a given short sequence of text (utterance) is defined according to the action or dialogue that that utterance is intended to produce. For example, in a banking customer support system, a user's message "I want to open a new savings account" might have the intent 'NewAccount', while another message "What are your standard weekend hours?" could have the intent 'GeneralInquiry.'

We reproduce here (from the introduction) the four key challenges in intent classification that motivated our work:

1. In building NLU systems for new domains, labeled data is often very limited. By contrast, unlabeled data is generally accessible, for example by parsing human-to-human chat logs or scraping unstructured text online
2. Many domains in which NLU systems are desired contain complex, domain-specific language. For example, consider building a medical chatbot, or designing a virtual legal assistant
3. IC is generally a *many*-class problem, as a large number of intents are needed to define the scope of conversation in even a heavily-restricted domain
4. Intents are defined purely by semantics, independent of syntactic or lexical features. This contrasts with many general text classification problems, where classes are often determined by high-level themes or categorical features.

3.2 Semi-Supervised Learning

Semi-supervised learning (SSL) is a class of learning that makes use of both labeled and unlabeled data. Although this spans all problem spaces between fully-supervised and fully-unsupervised learning, SSL is usually taken to refer to the more specific case where the number of unlabeled examples substantially outweighs the number of labeled examples.

Any semi-supervised approach works on the assumption that some property of the inherent structure of the domain, X, is useful in learning the function mapping points to their labels, $f : X \to Y$. Most commonly, these methods utilize the continuity assumption: that points closer together are more likely to have the same label. In addition, these methods can be used for either of the two fundamental types of learning: inductive or transductive. In inductive learning, labeled training data (and unlabeled data) are used to learn a general classifier that can be applied to unseen examples, while in transductive learning the goal is to label only a defined set of test examples.

As noted in §2, a wide range of semi-supervised methods have been proposed in the text classification domain. We outline the theory behind several here that are explicitly used in our work. The following sub-sections only describe the core theory underlying each algorithm. For our specific implementation of label propagation and extensions of the algorithm, refer to §4.3. For implementation-specific details on all other semi-supervised algorithms, refer to §5.3.2.

3.2.1 Label Propagation

Label Propagation (LP; Zhu, 2002) is an algorithm that classifies unlabeled data using the underlying structure of all labeled and unlabeled data points, typically applied in the transductive setting. LP operates over a fully-connected graph where nodes are data points and edges are weighted such that pairs of points, i, j, that are close in space according to some distance function, d_{ij}, are given a large edge weight, w_{ij}.

More specifically, let the number of classes C be known. Let $X_L = \{x_1, \ldots, x_l\}$ and $Y_L = \{y_1, \ldots, y_l\}$ be the labeled data, and $X_U = \{x_1, \ldots, x_u\}$, $Y_U = \{y_1, \ldots, y_u\}$ be the unlabeled data, where Y_U are unknown. Let each $x_i \in \mathbb{R}^m$. Then our edge weights between each pair of points are given by

$$w_{ij} = \exp(-\frac{d_{ij}^2}{\sigma^2}) = \exp(-\frac{\sum_{d=1}^{m} \left\|x_i^d - x_j^d\right\|_2}{\sigma^2}) \tag{1}$$

where d_{ij} here is the pairwise euclidean distance, and the weights are controlled by the normalising parameter σ.

We then define two matrices: a label matrix, $Y \in \mathbb{R}^{(l+u)\times C}$, whose i_{th} row represents the soft label distribution of node x_i (probabilities of being in classes $1, \ldots, C$), and a probabilistic transition matrix, $T \in \mathbb{R}^{(l+u)\times(l+u)}$, that specifies how easily a node's label distribution will propagate to its neighbours. T is given by

$$T_{ij} = Pr(j \to i) = \frac{w_{ij}}{\sum_{k=1}^{l+u} w_{kj}} \tag{2}$$

. Given these two matrices, one iteration of the LP algorithm consists of the following three steps:

1. Propagate $Y \leftarrow TY$
2. Row normalise Y
3. Clamp the labeled data (concentrate soft label distribution around ground-truth label)

where step 2 is required to maintain the interpretation of Y as label distributions and step 3 enforces a constant signal from correct labels. Intuitively, the labeled nodes 'push' their labels through high-density regions of the graph, allowing class boundaries to naturally settle in low-density gaps. The algorithm proceeds until some task-specific convergence criterion is met.

3.2.2 Self-Training

Self-training is one of the oldest semi-supervised algorithms. In this approach, a supervised classifier is trained on the labeled data alone, before the trained classifier is used to label the unlabeled data. Any examples that are classified with a probability higher than some threshold t are added to the labeled dataset. This process is repeated iteratively until either all the unlabeled data has labeled, or no new unlabeled examples are labeled in a given iteration.

3.2.3 k-Nearest Neighbour

The k-nearest neighbours (KNN) classification algorithm is one of the simplest classification techniques, whereby a point is classified according to a weighted 'vote' over the classes of the k nearest labeled points in X_L (where 'nearest' is determined by some distance function). Let x be the point to classify, $x_i \in X_L,\ i = 1, \ldots, k$ be the k neighbours in consideration, d_i be the distance between x and x_i, and w_i be the weight given to x_i's vote for the class of x. The most common weighting

scheme is simply to weight each of the k neighbours' votes equally, namely by setting $w_i = \frac{1}{k}$. Another common scheme is to set $w_i = \frac{1}{d_i}$, thereby weighting the votes of closer neighbours more highly.

3.2.4 k-Means

The k-means algorithm is generally used for unsupervised clustering, though it can be adapted to the SSL setting. Given a set of data points, $X = \{x_1, \ldots, x_n\}$, where each $x_i \in \mathbb{R}^d$, k-means attempts to partition the data into k clusters. More concretely, the objective is to partition X into the k sets $S_1, \ldots, S_k$ that minimize the within-cluster sum of squares (a.k.a variance or inertia) according to

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \mu_i\|^2 \tag{3}$$

where μ_i is the centroid of cluster S_i and can be used as a prototype for the cluster as a whole. We refer the reader to the scikit-learn documentation for a thorough explanation of the k-means algorithm[5].

More recently, (Gowda et al., 2016) proposed the use of a recursive extension of the k-means algorithm for semi-supervised text classification. In their approach, the k-means algorithm is applied recursively on each cluster until the clusters contain only labeled examples from a single class. The initial clustering sets $k = C$, where C is the number of classes, while each recursive k-means application on cluster S_i sets $k = C_i$, where C_i is the number of different classes in S_i.

3.3 Text Embedding

Any semi-supervised method that relies on the continuity assumption needs a representation for each data point that defines the structure of X. As described previously, we use pretrained embeddings for our representation scheme. In this section, we outline the general process for learning dense embeddings to represent text, as well as the theory specific to the four universal embedding models we use in our intent classification framework.

Text embedding is a collective term referring to any system for mapping sequences of text to real-numbered vectors, though we use the term to refer to the specific case where embeddings 1) are learned, either in an unsupervised or a supervised setting, and 2) are dense, fixed-length, high-dimensional vectors. A recent trend has been the development of 'universal embeddings,' representations that are pretrained on large corpora and are useful in a variety of downstream applications. We make use of four such universal embeddings in this work, two word embedding models and two sentence embedding models.

3.3.1 Universal Embedding Models

In this section we outline the four universal embedding models we use in this paper. For details on our implementation of these models within RIPPED, refer to §4.2.1.

GloVe (Pennington et al., 2014) is a word embedding model trained on data from the Common Crawl corpus[6]. GloVe is a log-bilinear regression model that incorporates both local context windows and global matrix factorisation into its objective for training embeddings. The model itself begins

[5] `https://scikit-learn.org/stable/modules/clustering.html#k-means`
[6] `http://commoncrawl.org/the-data/`

by constructing a matrix of ratios of co-occurrence probabilities between words, and can be most generally expressed as

$$w_i^T \tilde{w_k} + b_i + \tilde{b_k} = \log(X_{ik}) \tag{4}$$

where $w \in \mathbb{R}^d$ are the word vectors to learn, $\tilde{w} \in \mathbb{R}^d$ are 'context' word vectors, b are bias terms, and X_{ik} contains the number of times word i occurs in the context of word k in the entire training corpus. This objective is solved via a weighted least-squares regression algorithm, yielding a word vector w_i for every word in the corpus that can be extracted and used as a universal representation downstream.

ELMo (Peters et al., 2018) is a word embedding model that learns vector representations for words by training a bi-directional LSTM[7] as a language model[8] on the 1B Word Benchmark (Chelba et al., 2013). The output word embeddings are learned functions of the internal states of the LSTM, and are therefore specific to the context in which a word appears (as opposed to GloVe which learns single embeddings for each word to be used in all cases). For each token t_k, the 2-layer LSTM used in ELMo computes five representations given by

$$R_k = \{x_k^{LM}, \overrightarrow{h_{k,j}^{LM}}, \overleftarrow{h_{k,j}^{L}M} | j = 1, 2\} \tag{5}$$

where j is the layer index, x_k^{LM} is the input layer, and the $h_{k,j}^{LM}$ terms are the hidden states in layer j for the forward and reverse LSTM directions. The function combining these representations into a single embedding is then learned for a specific downstream task.

BERT (Devlin et al., 2018) is a sentence embedding model that learns vector representations by training a deep, bi-directional Transformer[9] network on a collection of data from Wikipedia and BookCorpus (Zhu et al., 2015). The network is simultaneously trained on both next-sentence prediction and the prediction of 'masked' words in a given sentence, and output sentence embeddings are simply the final hidden state of the first token in the input sequence.

InferSent (Conneau et al., 2017) is a sentence embedding model that learns vector representations by training a wide, bi-directional LSTM in a supervised setting on the SNLI dataset, described in the next section. Output sentence embeddings are learned combinations of the output layer hidden states.

3.3.2 Evaluation of Representational Ability

To understand the representational power and relative strengths of these four embedding models, we compiled results evaluating them on a range of tasks, presented in Table 3.1. As far as possible, we report results that use these four models in an 'untuned' manner, meaning the pre-trained representations are applied without modification to the downstream task. The sources of all listed results are given in §10. Note that these results do not represent the findings of any experiments we conducted; we are merely compiling results from various sources to provide insight into the relative ability of the four embedding models.

The tasks we use for this evaluation can be categorised into three main groups: semantic-textual similarity (STS), inference (Inf), and paraphrase detection (Para). We also include one probing tasks, semantic odd-man-out (SOMO), a binary classification task where the goal is to predict whether a single noun or verb in a sentence has been replaced with another, semantically out-of-place word. SOMO comes from the SentEval suite of datasets designed to evaluate embeddings (Conneau et al.,

[7] We refer the reader to Colah's blog for a detailed explanation of LSTM networks: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

[8] A language model is trained to predict the next word in a sequence given a context sequence of previous words.

[9] `http://jalammar.github.io/illustrated-transformer/`

Emb	Avg.	STS			Inf		Para		Probe
		SICK-R	STS-B	SICK-E	QNLI	MNLI	MRPC	QQP	SOMO
GloVe	72.8	80.0	65.0	79.0	75.1	58.7	72.6	79.4	54.2
ELMo	81.2	86.1	75.9	86.3	79.9	79.6	76.0	84.8	**58.2**
BERT	**85.2**	86.4	**82.9**	84.8	**90.1**	**84.6**	**78.1**	**89.4**	–
InfSnt	79.9	**89.0**	77.0	**86.4**	79.8	65.1	75.7	86.1	55.8

Table 3.1: Universal sentence embedding performance on a range of NLP tasks. For STS tasks, we report Pearson correlation, scaled to be in $[-100, 100]$; for MNLI we report accuracy (%) on the mismatched test split; for all other tasks we report accuracy (%) on the standard test set. Avg. is the unweighted average of all *two-sentence* tasks. Bold denotes best result per task; underline denotes best result per task among methods where the given result was achieved with *untuned* embeddings.

2018).

In STS, the goal is to predict the semantic similarity between two given sentences. We report results on the Sentences Involving Compositional Knowledge relatedness subtask (SICK-R; Marelli et al., 2014) and the STS-Benchmark (Cer et al., 2017). In paraphrase detection, the goal is to determine whether two sentences are semantically equivalent. We present results on the Microsoft Research Paraphrase Corpus (MRPC) and the Quora Question Pairs dataset (QQP), both from the GLUE suite of language tasks (Wang et al., 2018). In Inf tasks, the goal is to make some inference as to the relation of two input sentences. We report results on two natural language inference (NLI) tasks, the Multi-Genre Natural Language Inference Corpus (MNLI; Williams et al., 2018) and the SICK entailment sub-task (SICK-E). The goal in NLI is to predict whether the given premise sentence contradicts the given hypothesis sentence, entails it, or neither. We also present results on the Question NLI (QNLI) dataset from GLUE, where the goal is to determine whether the given context sentence contains the answer to the given question sentence.

These tasks were deliberately selected to test the ability of each embedding model to represent semantic properties that are relevant to their use in our semi-supervised setting. As was outlined in §3.1, the classes in intent classification are defined by semantics. In order to use these embeddings to compute distances between sentences that correspond to class boundaries, the embedding model will have to capture complex semantic features in its representation. We predict that the STS and Para tasks will be the most reliable indicators of the performance of a given embedding model in our framework, as strong performance on these tasks requires the ability to predict pairwise semantic similarity, independent of syntactic or lexical differences. Success on the Inf and SOMO tasks also requires the representation of subtle semantic features, hence their inclusion.

As can be seen, all four models demonstrate the ability to represent relevant semantic features of sentences. BERT appears to be the most effective across a range of tasks, though many of the given results using BERT were obtained by tuning embeddings on the specific downstream task. By contrast, all results reported for InferSent were achieved using the untuned embeddings, indicating its effectiveness as a universal, out-of-the-box representation solution.

4 Proposed Method

In this section we describe our proposed method for improving intent classification performance in limited labeled data domains. §4.1 contains a high-level overview of our approach, before §4.2 and §4.3 outline the two key components of our approach in turn.

4.1 Overview

Our method, RIPPED, is designed for use in the inductive semi-supervised setting, where we have some labeled data X_L and some unlabeled data X_U ($X_L << X_U$) and the goal is to train a general classifier for use on unseen examples. In this setting, RIPPED acts as a tool to add unlabeled examples to the labeled dataset for use prior to classifier-training, and as such can be incorporated into any existing classification pipeline. In addition, the method can be applied to a new domain with no 'training', the only required domain-specific decision being the choice of sentence representation.

RIPPED consists of a given sentence representation scheme used in combination with our recursive implementation of label propagation. In lieu of the recent success obtained by transferring the knowledge encoded in pretrained embeddings to a wide range of natural language processing tasks, we use pretrained embeddings as our sentence representation scheme in all cases. In §4.2, we outline the seven different sentence embedding models we experiment with. Although RIPPED uses a recursive implementation of label propagation, we also define and test two other label propagation variants in our work, outlined in §4.3.2.

We implement RIPPED as an augmentation system, improving classification performance by classifying sentences in X_U in order to augment X_L, thus allowing a stronger classifier to be trained than would have been possible using only the original labeled data. The process can be summarised as follows

1. Transform all sentences in X_L and X_U into embedding space using one of the pretrained embedding models outlined in 4.2.
2. Compute the distance between each pair of embeddings and initialise the two matrices needed for label propagation.
3. Set the key parameter, σ, according to the entropy heuristic detailed in §4.3.3.
4. Recursively propagate intent labels using the lp-R algorithm outlined in §4.3.2, transferring newly labeled sentences from X_U to X_L.

4.2 Sentence Embedding

We use two categories of sentence embedding models in our approach: the four state-of-the-art universal embeddings described in §3.3.1, and our own embeddings trained from scratch on the STS task, outlined in §3.3.2. In both cases, the embeddings are pretrained prior to their use in a specific classification task and receive no fine-tuning. The motivation for using the embeddings in an untuned fashion is twofold: 1) RIPPED is intended for use on datasets with very few labeled examples, generally too few for effective fine-tuning, and 2) we designed RIPPED such that it can be incorporated into new classification pipelines as easily as possible.

To extend the continuity assumption that is central to the label propagation algorithm, our approach assumes that pretrained embedding models can convert the task domain into an embedding space in

which pairwise distance calculations between data points reflect the intent class boundaries. Given that we use untuned embeddings, this is a hefty assumption, and one that we evaluate in depth in our experiments.

4.2.1 Universal Sentence Embeddings

Background on the four universal embedding models we use, as well as a comprehensive evaluation of the representational ability of each model, is provided in §3.3. This section describes our implementation of the four models.

For GloVe, we use the 300d-840B model, available for download[10]. To extract sentence representations from the given word embeddings, we max-pool the word embeddings for each token in a sentence, resulting in 300-dimensional embeddings. For ELMo, we use AllenNLP's default implementation[11]. The original paper recommends learning the optimal function over the five different LSTM representations for each downstream task, but in order to use these embeddings in an untuned fashion, we use only the hidden states of the highest layer in all cases. We max-pool over the states for each word to get 3072-dimensional embeddings.

For BERT, we use the HuggingFace implementation of the base-uncased variant[12]. To extract sentence representations, we use the default final hidden state of the first token in the input sequence, resulting in 3072-dimensional embeddings. For InferSent, we use the InferSent1 model whose embedding layer was initialised with GloVe 300d-840B vectors[13]. To extract sentence representations, we max-pool the output layer hidden states, resulting in 4096-dimensional embeddings.

As was outlined in §3.3.2, this selection of embedding models was motivated by a slate of recent work asserting their ability to represent important semantic features. In addition to the results we compiled in Table 3.1, we reference here some indications of the strength of these embeddings when applied untuned to downstream tasks. Firstly, our pooling implementation of GloVe and ELMo has been shown to be a strong baseline on a wide range of benchmark NLP tasks, despite sacrificing all knowledge of word order (Wieting et al., 2016; Arora et al., 2017). Secondly, although BERT generally performs better with fine-tuning as opposed to untuned feature-extraction, (Peters et al., 2019) demonstrated that the untuned embeddings still yield close to state-of-the-art results on a variety of tasks. Finally, (Zhu et al., 2018) found that out-of-the-box InferSent embeddings excel at distinguishing subtle semantic relations between sentences such as negation, synonymy, and semantic equivalence.

4.2.2 STS-Trained Embeddings

In addition to the universal sentence embeddings, we also train several embedding models from scratch on the STS task (the STS task is described in §3.3.2 as one of the tasks used to evaluate the universal embedding models). We use three datasets – SICK, the STS-Benchmark, and a random, equal combination of the two – to train three models, referred to as STS-sick, STS-bench, and STS-both. Our motivation behind this approach was simple: explicit training on semantic similarity prediction should result in embeddings that are better able to express pairwise semantic similarity. Since the label propagation algorithm only uses these embeddings to calculate distance between points, we hope that these STS-trained embeddings will express the desired correspondence between

[10] `https://nlp.stanford.edu/projects/glove/`
[11] `https://allennlp.org/elmo`
[12] `https://github.com/huggingface/pytorch-pretrained-BERT`
[13] `https://github.com/facebookresearch/InferSent`

semantic distance and intent class boundaries.

Model Architecture

We use the same general training architecture as (Conneau et al., 2017), illustrated in Figure 4.1. For each training example, a sentence encoder is used to embed both sentences. The absolute difference and element-wise product of these two embeddings are fed into a small feed-forward multilayer perceptron (MLP) that we call the 'prediction network,' which in turn outputs a real-valued semantic similarity prediction.

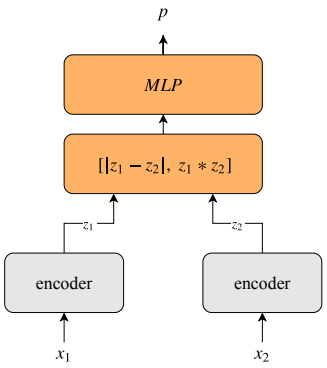


Figure 4.1: STS embedding training architecture.

For the encoder, we use a one-layer, bi-directional LSTM with adaptive max-pooling over hidden states. We use a hidden size of 300 and initialise the input layer with GloVe 300d-840B embeddings (which are kept frozen throughout training, as we saw no performance gain through fine-tuning). The prediction network contains a single hidden layer of size 300, with dropout and ReLu non-linearity (Glorot et al., 2011) following the input and hidden layers. At training completion, we extract the encoder to use as an untuned embedding model in our semi-supervised intent classification framework.

Training

As we trained these models purely for downstream use (not to evaluate on the STS task itself), we use the test dataset for hyper-parameter tuning and early stopping. The architecture was trained end-to-end using the Adam optimiser (Kingma and Ba, 2015) with a learning rate of $6e^{-4}$ and batch-size of 64. Weights were regularised using per-batch L2-regularisation of $5e^{-3}$, and the prediction network dropout was set to 0.2. After pooling the hidden states for both the forward and reverse directions, each of these three encoding models outputs 600-dimensional embeddings. The three trained models achieved Pearson correlations of 0.872, 0.820, and 0.843 on their respective test sets, and while these results are not comparable to the results given in Table 3.1 evaluating the universal embeddings (due to our tuning on the test set), they do give an indication that our encoders learn representations that capture a strong intuition of semantic similarity.

4.3 Label Propagation

In this section, we discuss theory and implementation details related to our use of the label propagation (LP) algorithm. We begin in §4.3.1 by outlining our choice of the distance function that is used to convert pretrained embedding representations into the graph over which LP operates. In §4.3.2, we outline the three label propagation variants we experiment with in our framework: basic (lp-B), threshold (lp-T), and recursive (lp-R). Our proposed method, RIPPED, uses the lp-R variant, while the other two are included both in order to outline the process by which we arrived at lp-R and because we compare lp-R to these variants in our experiments. Finally, §4.3.3 describes the heuristic we use to set the key parameter used in LP.

4.3.1 Distance Function

In order to initialise the transition matrix T that is needed in LP, we need a pairwise distance function with which to compute distances between every pair of sentence embeddings. We considered Euclidean and Cosine distance, the two distance functions most widely used with embedding representations. We note that, if the embeddings are normalised, cosine distance can be defined as euclidean distance with the application of an additional monotonic transformation, meaning that comparing distances between pairs of points will result in the same relative 'ordering' when using either function.

More concretely, let u, v be two sentence embeddings, then the euclidean distance between them is defined as

$$\|u - v\|_2 = \sqrt{\sum_{i=0}^{m} (u_i - v_i)^2} \tag{6}$$

where m is the dimensionality of the embeddings. Using the LHS as shorthand for the euclidean distance, we get that

$$\|u - v\|_2^2 = (u - v)^T (u - v) = \|u\|_2^2 + \|v\|_2^2 - 2u^T v \tag{7}$$

And when u, v are normalised, $\|u\|_2^2 = \|v\|_2^2 = 1$, so $\|u - v\|_2^2 = 2u^T v = 2(1 - \cos(u, v))$, where $1 - \cos(u, v)$ is the cosine distance. This means squared euclidean distance is proportional to cosine distance, and since squaring is a monotonic function, the relative ordering of values obtained using either distance function will be the same.

We found in our implementation that euclidean distance was orders of magnitude faster when computing $n \times n$ distance values, and since the two measures produce the same relative results, we use euclidean distance in all our algorithms. We also experimented with using the STS similarity output by our pretrained STS prediction networks, p, to produce a measure of distance, $1 - p$, but we found this to yield poor results. We suspect this is because the learned network was too closely aligned to the sentence structure and language of its training domain. This is an interesting area for future research.

4.3.2 LP Variants

In this section we describe the recursive algorithm used by RIPPED – lp-R – by first outlining two simpler variants, where each successive algorithm builds off the previous.

The first variant, lp-B, follows the standard algorithm outlined in §3.2.1. Given labeled and unlabeled data, it constructs a fully-connected graph, weighting edges using the pairwise euclidean distance between embeddings, before propagating labels through the graph. We define convergence of the propagation to be $\|Y_i - Y_{i-1}\|_2 < \text{tol}$, where Y_i is the label matrix after iteration i. This tolerance value is set experimentally depending on dataset size, primarily in order to manage run-time; we found performance to be robust to a range of tolerance values. After convergence, we classify the unlabeled data according to

$$y_{l+i} = \arg\max_j Y_{l+i} \tag{8}$$

where l is the number of labeled examples, i indexes over unlabeled examples, and $j = 1, \ldots, C$, where C is the number of classes. This variant labels *all* unlabeled examples with their most probable class, adding these to the labeled set for classifier training.

Extending the basic variant slightly, lp-T makes use of the interpretation of Y as a distribution over classes to enforce a stricter constraint on the labeling process. The same core LP algorithm is used

Algorithm 1: Recursive label propagation algorithm (lp-R)

while *not complete* **do**
  initialise Y using Y_L, T using w_{ij};
  set $Y_L = Y_{:l}$;
  while $\left\|Y^i - Y^{i-1}\right\|_2 > tol$ **do**
    $Y \leftarrow TY$ (propagate);
    $Y \leftarrow \|Y\|_2^{-1} Y$ (row-normalise);
    $Y_{:l} \leftarrow Y_L$ (clamp ground-truth labels);
  end
  $y_{l+i} = \arg\max_j Y_{l+i} \iff y_{l+i} > t$;
  transfer all new y's from X_U to X_L;
end

until convergence, though now the following classification criteria is used

$$y_{l+i} = \arg\max_j Y_{l+i} \iff y_{i+1} > t \tag{9}$$

where t is a threshold parameter. Put simply, we no longer force the algorithm to use all unlabeled examples, instead selecting only those classifications that the algorithm is confident in. The motivation here is to pull the algorithm back from attempting to label all examples at the complex decision boundaries, allowing it to achieve higher accuracy.

The final variant, lp-R, is a recursive extension of lp-T. For each lp-R iteration, we apply the entire lp-T propagation process, classifying some unlabeled examples using (9) and augmenting the labeled dataset. We then take this new labeled/unlabeled split and use it as input to the next lp-R iteration. More concretely: lp-B and lp-T both apply the core label propagation algorithm a single time, and only differ in terms of the criteria they use after convergence to classify unlabeled examples into the labeled dataset. By contrast, lp-R applies the core LP algorithm recursively, using the newly augmented labeled dataset after one iteration as the input to the next. The recursion continues until either there are no unlabeled examples left, or no unlabeled examples are classified in a given iteration. A complete outline of the lp-R algorithm is shown in Algorithm 1.

The progression of these three variants was designed to address the challenges specific to intent classification, principally that the problem space is many-class and unbalanced, often with complex class boundaries. Since our method is used as a data augmentation tool as part of the pipeline for training an inductive classifier, there is no requirement to use all unlabeled data. Therefore, the introduction of a threshold allows the algorithm to achieve far greater accuracy on the examples it does choose to classify. The motivation for lp-R (and therefore RIPPED) is that by performing only a single propagation run, lp-B and lp-T will likely miss some unlabeled examples they could have classified if they were *certain* of a few more labels to begin with.

The single propagation run of lp-T slowly pushes the initial labels through dense regions of the graph, before the threshold pulls back the labels to only those the algorithm is most confident in, thus avoiding the complex decision boundaries. Instead of stopping here, lp-R takes this new labeled dataset (where the class probabilities of the newly labeled examples have been fixed to their predicted label, i.e. they are no longer soft distributions) and repeats the entire lp-T process. The hope is that lp-R can push labels further through the graph while maintaining high accuracy. To clarify these intuitions, we include a visualisation of the propagation process in Figure 4.2, outlining the classifications that would be returned by lp-B, lp-T, and lp-R.

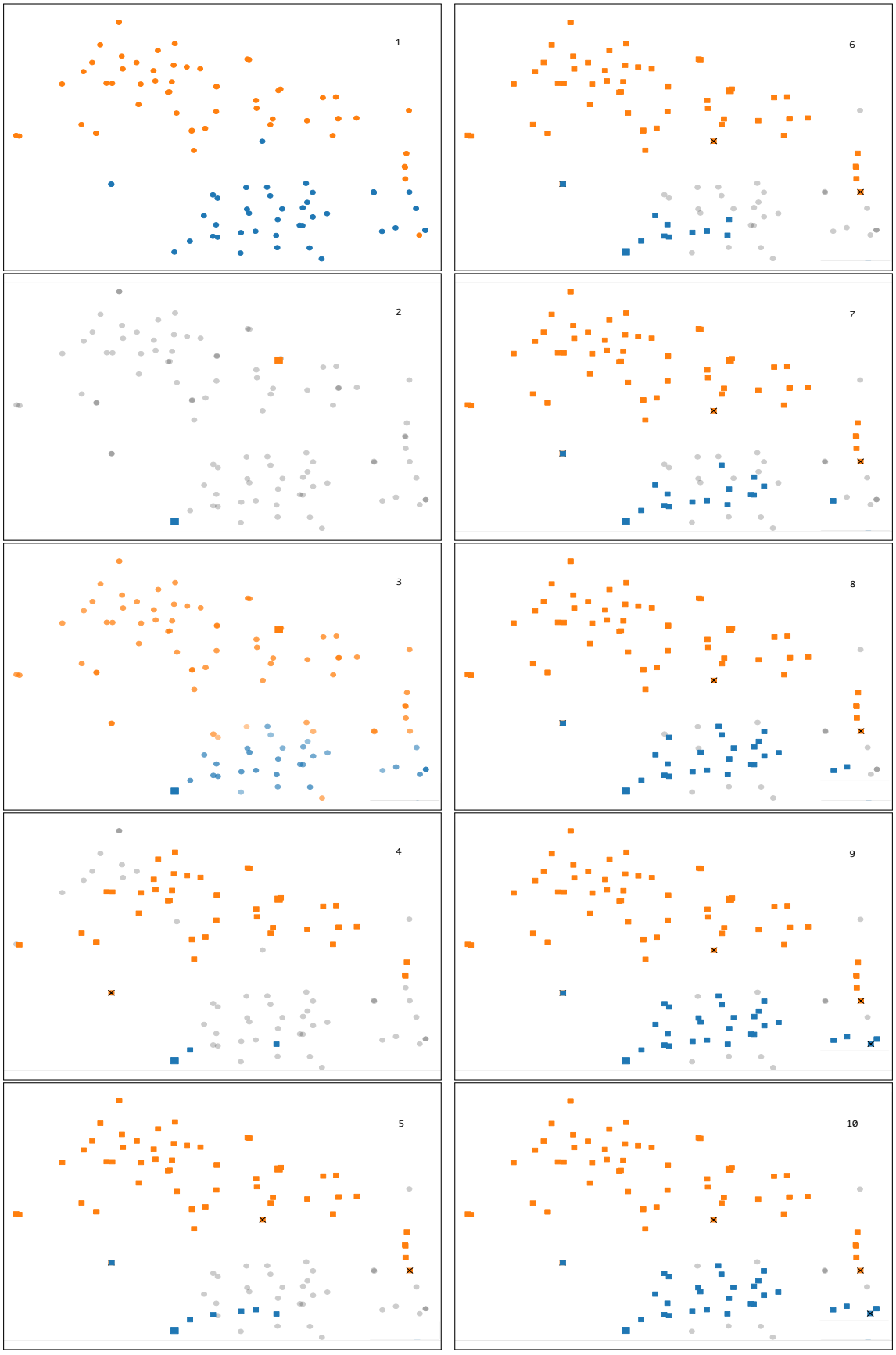


Figure 4.2: Intent label propagation process visualisation. *1* shows the true labels and *2* shows a randomly sampled starting point with one labeled example per class. *3* gives the output of lp-B, where labels are propagated throughout the graph with different confidences (and many classifications are incorrect). *4* gives the output of lp-T, showing how the threshold pulls back the labels to only those with high confidence. *5-10* show the recursions of lp-R, with *10* giving the final output. As can be seen, lp-R involves a gradual high-confidence of push of labels, resulting in high accuracy and high labelling coverage.

4.3.3 Parameter Setting

The critical parameter in the label propagation algorithm is σ, the normalising value used to initialise the transition matrix before each propagation run. We also introduce a threshold parameter t in two of our LP variants. The challenge in setting these parameters experimentally is that we have very little information about the distribution of classes over the representation graph. This section outlines the heuristic we use to do so.

Firstly, we reproduce the equation for the edge weights between two points in the embedding graph from §3.2.1. Recall that we want w_{ij} to be inversely proportional to the distance between points i and j

$$w_{ij} = \exp(-\frac{d_{ij}^2}{\sigma^2}) = \exp(-\frac{\sum_{d=1}^{D} \left\|x_i^d - x_j^d\right\|_2}{\sigma^2}) \tag{10}$$

We can see that as $\sigma \to 0$ the influence of the nearest labeled point dominates, and therefore the algorithm propagates labels to unlabeled points based only on their single nearest neighbour. And as $\sigma \to \infty$, the edge weights tend to zero, concentrating the entire graph around a single point, and therefore all unlabeled points will have soft label probabilities reflecting the labeled dataset class frequency. The optimal value of σ clearly lies in between these values, and it seems that we intuitively want it to correspond in some way to the distances between intent classes.

Before discussing our heuristic for setting σ, we note that our threshold parameter t simply defines the cutoff at which our method decides it is confident enough in the classification of a given unlabeled example; it does not affect the propagation process itself. Furthermore, t is explicitly related to σ. σ is used to define the transition matrix, T, which is multiplied by Y at each iteration until convergence. The lower the value of σ, the more confident the algorithm will be in making classifications, and therefore the higher t will have to be in order to ensure high augmentation accuracy. Because of the interdependence of these two parameters, we set $t = 0.99$ in all cases, and focus on designing a sophisticated heuristic for σ to ensure that easy classification decisions result in very high confidence values.

We build off several methods proposed in (Zhu, 2002) to develop a heuristic that reliably sets σ given very little knowledge of how the true class labels relate to the overall embedding graph structure. As a starting point, we use the minimum-spanning tree (MST) heuristic, whereby we first construct the MST using Kruskal's Algorithm (Kruskal, 1956) and the pairwise euclidean distances d_{ij} between every pair of points, both labeled and unlabeled. We then define the distance of the shortest edge connecting two labeled points in different classes as d_0. We define the output of the MST heuristic, σ_0, as

$$\sigma_0 = \frac{d_0}{3} \tag{11}$$

where the division by three comes from the 3σ rule of Normal Distributions[14]. The motivation here is that, by setting $\sigma = \sigma_0$, the edge weight w_{ij} between the two points defining the minimum arc will be close to zero, thereby hopefully confining propagation to occur within intent classes.

However, σ_0 is completely dependent on the location of the initial labeled examples within the underlying graph class structure, and therefore setting σ using the MST heuristic is highly variable. Assuming that a good σ value will make classification decisions with high confidence (and therefore low entropy), we refine σ_0 by minimising the entropy of Y_U, the output probability distribution predicting labels over the the unlabeled data. Entropy in this case is defined as

$$H(Y_U) = -\sum_{ij} Y_U^{ij} \log(Y_U^{ij}) \tag{12}$$

[14] `https://www.encyclopediaofmath.org/index.php/Three-sigma_rule`

. However, there is a complication in that entropy is minimised by setting $\sigma = 0$ (because this assigns hard labels using the single nearest neighbour, as discussed above). To avoid this, we smooth the transition matrix by interpolating it with the uniform transition matrix. More concretely, let the transition matrix be T, the uniform matrix be U (where $U_{ij} = \frac{1}{l+u}$), and interpolation parameter be ϵ. We then replace the transition matrix T with the smoothed transition matrix $\tilde{T}$, defined as

$$\tilde{T} = \epsilon U + (1 - \epsilon)T \tag{13}$$

When $\sigma \to 0$, the uniform matrix dominates $\tilde{T}$, thus generating almost uniform class probabilities. This means Y_U will have high entropy, consequently avoiding the minima at $\sigma = 0$. By introducing this interpolation, we've added another parameter ϵ. However, we discovered that by arbitrarily setting ϵ to be extremely small, we avoid the minima at $\sigma = 0$ and yield extremely consistent σ values for a given dataset. The complete sigma-setting process is summarised in the following paragraph.

Given a specific labeled/unlabeled dataset split, X_L/X_U (e.g. a real-life classification problem, or a single trial in our experiments), we use the MST heuristic to get σ_0. We then set $\epsilon = e^{-50}$ and perform a single label propagation run for each value of $\hat{\sigma} \in [0.01, \sigma_0]$ in intervals of 0.001. This range of values was chosen because 1) we don't want $\sigma \to 0$, and 2) the minimum-distance edge found using the MST-heuristic can't be higher than the actual minimim distance between classes in the complete graph. The propagation run using each value of $\hat{\sigma}$ produces a Y_U matrix, and we then set σ as follows

$$\sigma = \underset{\hat{\sigma} \in [0.01,\ \sigma_0]}{\operatorname{argmin}} H(Y_U) \tag{14}$$

where we refer to the entire heuristic process yielding this final value of σ as the entropy heuristic. Finally, to perform the actual augmentation process, we set σ as given in (14), set $\epsilon = 0$ (no interpolation), and run our recursive label propagation algorithm.

In §5.4 we outline several experiments designed to test both the consistency of our entropy heuristic (the variability in the σ values returned) and the impact of different σ settings on the performance of our augmentation method.

5 Experiments

Our principal hypothesis in performing this research was that RIPPED, the integration of pretrained embeddings into a recursive label propagation algorithm, could be used to improve few-shot intent classification by augmenting the small labeled dataset with unlabeled examples, thus allowing a stronger classifier to be trained. In §5.1 we outline the intent classification datasets used, before assessing the validity of our continuity assumption in §5.2. We do so by investigating the ability of the untuned embeddings to construct an embedding graph that corresponds to the intent class semantics.

We then turn to our core experimentation in §5.3, comparing the performance of a classifier augmented with RIPPED to that of the fully-supervised baseline (i.e. only using the initial labeled data), as well as to the performance of a number of other semi-supervised algorithms and data augmentation techniques. We also perform several ablation studies investigating the impact of the critical parameter used in our label propagation variants.

5.1 Datasets

We use four intent classification datasets to evaluate our method, deliberately chosen to span a variety of semantic domains, difficulties, and dataset sizes. Dataset statistics are summarised in Table 5.1, while a few example sentences from each dataset are provided in Table 5.2

Stat	**Dataset**			
	Chatbot	**AskUbuntu**	**Webapps**	**Sied**
# train	100	53	30	992
# test	106	109	54	121
# classes	2	5	8	21
# eg/class	50	10	4	42
vocab	132	191	122	461
# oov	52	245	165	29
sent len	9.4/19/4	8.8/21/3	8.3/21/3	8.5/28/2

Table 5.1: Dataset statistics. '# eg/class' is the average number of training examples per intent class; 'vocab' is the number of unique words; '# oov' is the number of words in the test set that don't appear in the training set; the values for 'sent len' denote average/maximum/minimum sentence length.

The first three datasets – Chatbot, AskUbuntu, and Webapps – come from the NLU Evaluation Corpora (Braun et al., 2017). This corpus was designed for evaluating the natural language understanding capability of conversational agents, and as such the datasets are constructed to contain primarily 'chatty' language. The sentences in Chatbot comes from a Telegram chatbot for public transport queries in Munich, those in AskUbuntu come from a selection of software-related posts on AskUbuntu, and those in Webapps come from the Web Applications topic forum on StackExchange. The fourth dataset, Sied, contains customer-support focused queries extracted from a variety of banking and business chatbot interactions[15]. In all cases, the only preprocessing we applied was Spacy[16] tokenisation to obtain a list of tokens for each sentence.

[15] `https://github.com/sebischair/NLU-Evaluation-Corpora`
[16] `https://spacy.io/`

Dataset	Sentences	Intent-Class
Chatbot	"how i can get from marienplatz to garching"	'FindConnection'
	"when is the next bus from ostbahnhof?"	'DepartureTime'
AskUbuntu	"How do I update Xubuntu 11.10 to Xubuntu 12.04 LTS?"	'MakeUpdate'
	"Is there a lightweight tool to crop images quickly?"	'SoftwareRecom'
Webapps	"Change subject line in new Gmail compose window"	'None'
	"Get rid of Russian junk from my Gmail"	'FilterSpam'
Sied	"What can I use the loan money for?"	'BorrowUse'
	"Where can I register for a company direct?"	'AppProcess'

Table 5.2: Example sentences and their intent classes from each of the four datasets.

The small size of the datasets is deliberate, as our motivation for RIPPED was to design a framework for bootstrapping natural language understanding systems, starting from domains with only very few labeled examples. The four datasets have differing numbers of classes, allowing us to assess the robustness of our method to classification problems of varying complexity. In addition, the classes in each dataset are unbalanced, in some cases exceedingly so. This presents our model with a substantial challenge, as semi-supervised learning approaches in general are known to struggle on many-class problems with complex decision boundaries, especially given that we assume no knowledge of class distribution. We outline the class breakdowns of each dataset in §10.

5.2 Evaluating our Continuity Assumption

One of the core assumptions we made in designing our method was our specific extension of the continuity assumption: that the structure of the graph created by using untuned embedding models in a given domain would correspond to that domain's intent class semantics. In order to test this assumption, we use the complete train and test datasets to compute a measure of statistical separation between the embedded sentences in each intent class. These labels are not available in the practical application for which our model is intended. However, we believe this exploratory experiment to be useful in assessing the validity of one of our core assumptions. Furthermore, it allows for a useful comparison of the ability of the different embedding models when used in our framework.

For a given embedding method on a given dataset, we embed all sentences and compute the mean (centroid) and variance of each intent class (cluster). More concretely, let E_k be the set of indices of the examples belonging to intent class k, and z_i be the sentence embedding of sentence i. We calculate the centroid and variance of each intent class as follows

$$\mu_k = \frac{1}{|E_k|} \sum_{i \in E_k} z_i \tag{15}$$

$$\sigma_k^2 = \frac{1}{|E_k|} \sum_{i \in E_k} \|z_i - \mu_k\|_2 \tag{16}$$

We then define a measure, I, to capture an embedding model's 'intent-semantic representational ability', a measure of how well the semantics encoded by the model align with the intent class boundaries. For embedding type e on dataset d, we define $I_e(d)$ as

$$I_e(d) = \frac{1}{C} \sum_{i}^{C} \frac{\sigma_i^2}{\min\{\|\mu_i - \mu_j\| \mid j \neq i\}} \tag{17}$$

where C is the number of intent classes, and μ_i and σ_i^2 are the centroid and variance of cluster i. Intuitively, I measures the average 'overlap' between intent clusters in embedded space, a low value thereby indicating that a given method naturally separates the sentences into their respective intent classes.

To clarify the intuition behind these findings, we also analyse these results visually. We apply PCA[17] to reduce the embeddings to their 50 principal components before using t-SNE[18] on the reduced embeddings to get two-dimensional points. We found this approach to be more consistent than using either reduction technique in isolation, though to counteract the remaining variability we repeat the process 10 times for each embedding method. Note that the numerical I calculations are performed on the original embeddings (with all dimensions) and therefore show no such variability.

5.3 Intent Classification

This section outlines the primary experiments conducted in our research.

5.3.1 Experimental Setup

Our experimental setup consists of four independent variable axes: embedding type, semi-supervised algorithm, dataset, and the fraction of training examples used as labeled data. The dependent variables are the test statistics representing intent classification performance. For each dataset, we use the standard train/test split before splitting the training data into labeled and unlabeled datasets (X_L and X_U) to give us a semi-supervised setting. We experiment with a large range of labeled fractions, from using 90% of the training data labels down to the one-shot setting, with a single labeled example per class. In all cases, we enforce the constraint that X_L contains at least one example from each class.

As the core baseline against which we evaluate our semi-supervised approach, we use the best fully-supervised classifier on each dataset, determined via a grid search over classifier types in {Naive Bayes, SGD, MLP, RNN}. Full details of this process are outlined in §10. We use the optimal fully-supervised classifier to ensure that our baseline, when given access to the complete training dataset, equals the state-of-the-art results outlined by (Braun et al., 2017) and discussed in a more recent blog post[19].

In our experimental framework, a 'method' consists of an embedding type in {GloVe, ELMo, BERT, InferSent, STS-sick, STS-bench, STS-both} and a semi-supervised algorithm in {KNN, kmeans-B, kmeans-R, lp-B, lp-T, lp-R}, where the first three algorithms are semi-supervised baselines with which to compare our LP variants, described in §5.3.2. As explained in §4.3, RIPPED is defined as the combination of any embedding type with lp-R. Each method acts as an augmentation tool, expanding the labeled dataset using the unlabeled examples, before the same classifier as used

[17] A lovely visual explanation of PCA: `http://setosa.io/ev/principal-component-analysis/`
[18] `https://lvdmaaten.github.io/tsne/`
[19] `https://medium.com/botfuel/benchmarking-intent-classification-services-june-2018-eb8684a1e55f`

in the supervised baseline is trained on the expanded labeled set and evaluated on the test data. The seven embedding types used are outlined in §4.2, and our three label propagation variants are outlined in §4.3. The addition of the three semi-supervised baselines gives us a total of 42 methods, each of which we evaluate on the four datasets using a range of fractions of labeled data. While our experiments constitute a holistic comparative evaluation of these 42 methods, we are primarily seeking to compare the performance of RIPPED to the other approaches. In order to best assess our hypotheses, we perform separate analyses isolating the impact of the two constituents of each method (semi-supervised algorithm and embedding model) in turn.

As our framework constitutes an inductive application of the semi-supervised algorithms, the result of each method is a general classifier intended for use on unseen examples. We perform an experiment comparing this approach to the standard transductive setting by using the entire training dataset as labeled data and the test dataset as unlabeled data that needs to be classified. Note that in this setting, each method is required to label *all* unlabeled data, meaning that of our LP variants we are restricted to using only lp-B.

In all experiments, we measure both the accuracy of the augmentation process and the fraction of unlabeled data used in the augmentation, as well as the intent classification accuracy on the test data. Since the datasets are unbalanced multi-class problems, we also measure precision, recall, and f1 score, the latter being our primary target statistic. For these three statistics, we use a weighted averaging scheme across classes, taking all classes to have equal importance. For each 'method' on each dataset, using each fraction of labeled data, we perform 200 trials (50 for Sied) to take into account the variability due to the initial set of labeled examples (where this is randomly sampled from the training set in each trial).

5.3.2 Baseline Semi-Supervised Algorithms

To implement the k-nearest neighbors algorithm in our framework, we use a knn classifier to classify X_U, adding all newly labeled points to X_L[20]. For each dataset, we find the best knn classifier using a grid search over all parameters: number of neighbours, neighbour weighting scheme, and knn algorithm. Full details of this process are provided in §10.

We also experiment with two variants of the k-means semi-supervised algorithm, in both cases using the 'k-means++' algorithm for initialisation (Arthur and Vassilvitskii, 2007). In the base variant (kmeans-B), we apply k-means clustering over all labeled and unlabeled data, labeling all unlabeled points in a given cluster with the majority label of the labeled points. For the recursive variant (kmeans-R), we use the implementation proposed in (Gowda et al., 2016), whereby k-means clustering is applied recursively on each partition until each cluster contains labeled points of a single class. Background on both variants is given in 3.2.4.

We also implemented the standard self-training algorithm, a nice overview of which is given in (Triguero et al., 2013). This algorithm saw improvements over the fully-supervised baseline on the Chatbot dataset (2 classes), but significantly degraded performance on all others (with all settings of the threshold), hence we don't include it as one of the core semi-supervised algorithms in the set above. Self-training is conceptually similar to our lp-R variant, recursively using the new labeled/unlabeled dataset splits it generates after one iteration to initialise the next, but we believe the poor performance in this case is due to the classification decisions being simply too hard (with

[20]We also tested a thresholded variant (with threshold t), whereby points are only added to X_L if more than $t * n$ of their n neighbors are in the same class. However, this was outperformed by the basic variant in all cases, so results are not reported.

so little data) for a supervised classifier to feed itself new data with high accuracy.

Finally, we tested a range of data augmentation techniques as approaches to improve classification performance without the use of unlabeled data. We experimented with synonym replacement using WordNet (Miller, 1992; Zhang et al., 2015), Easy Data Augmentation (Wei and Zou, 2019), and back-translation (Sennrich et al., 2016). However, despite testing a range of parameter settings in each case, none of these approaches came close to achieving the performance of the supervised baseline. We believe that these techniques struggle to preserve class labels when performing the augmentation, likely due to the specificity of the intent classes and the complexity of their decision boundaries. As such, we do not report results using these techniques.

5.4 Impact of σ

We also conduct two experiments to investigate the impact of the σ parameter on the performance of RIPPED. Firstly, we test the robustness of the entropy heuristic we use to set σ (described in §4.3.3). For a given fraction of labeled data, we run 200 σ-setting trials, where each trial consists of a random sampling of points in the training data as the labeled examples. We perform these experiments on four different fractions of labeled data for each dataset. The aim here is that, for a given labeled fraction, the heuristic returns consistent σ values across the 200 trials, not showing substantial variation relative to the initial selection of labeled examples.

Secondly, we perform an ablation study to investigate the impact of σ on the intent classification process itself. We use RIPPED with a range of different σ values, recording for each the accuracy of the augmentation, the fraction of unlabeled examples used in the augmentation, and the final classification f1. We compare the results using the σ values selected by our entropy heuristic to the best-performing σ values from this study to assess whether our approach could be improved through developing a stronger σ-setting heuristic.

6 Results

This section presents the results of all experiments outlined in §5.

6.1 Evaluating our Continuity Assumption

In this section we present the numerical and visual results of our exploratory experiments. The intent-semantic representational ability (I) of each embedding method is shown in Table 6.1, while in Figure 6.1 we present an example visualisation comparing the best and worst of each embedding category on the Chatbot dataset (visualisations with dim-reduced embeddings are a mess on all other datasets as they have too many classes). Note that while ordinal comparison of I values between datasets is possible, these values are not scaled to allow any form of ratio or multiplicative comparison. Furthermore, the I values of the two categories of embedding don't appear to be directly comparable: the best STS-models have lower I but show substantially worse performance. This discrepancy is due to the clusters using STS embeddings having far greater distances between centroids, where this in turn is caused by the STS embeddings expressing far more variability in general than those in the universal category. This indicates a flaw with our I measurement, though comparisons within categories seem generally reliable.

Emb	Avg.	Chatbot	AskUbuntu	Webapps	Sied
GloVe	2.51	1.70	4.52	1.89	1.91
ELMo	**1.04**	1.23	**1.22**	0.76	**0.94**
BERT	1.28	**1.21**	1.33	1.63	0.95
InferSent	1.10	1.45	1.24	**0.74**	0.98
STS-sick	1.85	<u>1.24</u>	1.77	2.38	2.02
STS-bench	<u>0.99</u>	1.37	<u>1.14</u>	<u>0.56</u>	<u>0.90</u>
STS-both	1.09	1.70	1.16	0.60	0.91

Table 6.1: The 'intent-semantic' representational ability (I) of each embedding method on the given datasets. Lower is better. Note that the two categories of embeddings are not comparable, as outlined above (the STS models are worse at clustering despite lower absolute scores). Avg. is the unweighted mean over the four datasets; bold denotes best universal embedding value per dataset; underline denotes best STS-trained embedding value per dataset.

As can be seen, ELMo is the strongest universal embedding type across the four datasets (with InferSent in a close second), while STS-bench is the best of the STS-trained models. This does correspond in general to the performance using these embeddings with RIPPED. While the I values don't correspond perfectly to intent classification performance, there is a clear relationship between these measures. It seems that I as a proxy for 'intent-semantic representational ability' is a decent indicator as to an embedding model's usefulness on a given dataset, although small ΔI's can't accurately predict relative performance.

It's important to note that both the numerical I calculations and the visualisations are done by using *untuned* embeddings for the sentences in the dataset, and that the embedding model has no knowledge of the intent classes when creating these embedding spaces. Yet the ability of the embeddings to represent the intent-class boundaries out-of-the-box is clear. This validation of our continuity assumption is fundamental to the success of RIPPED.

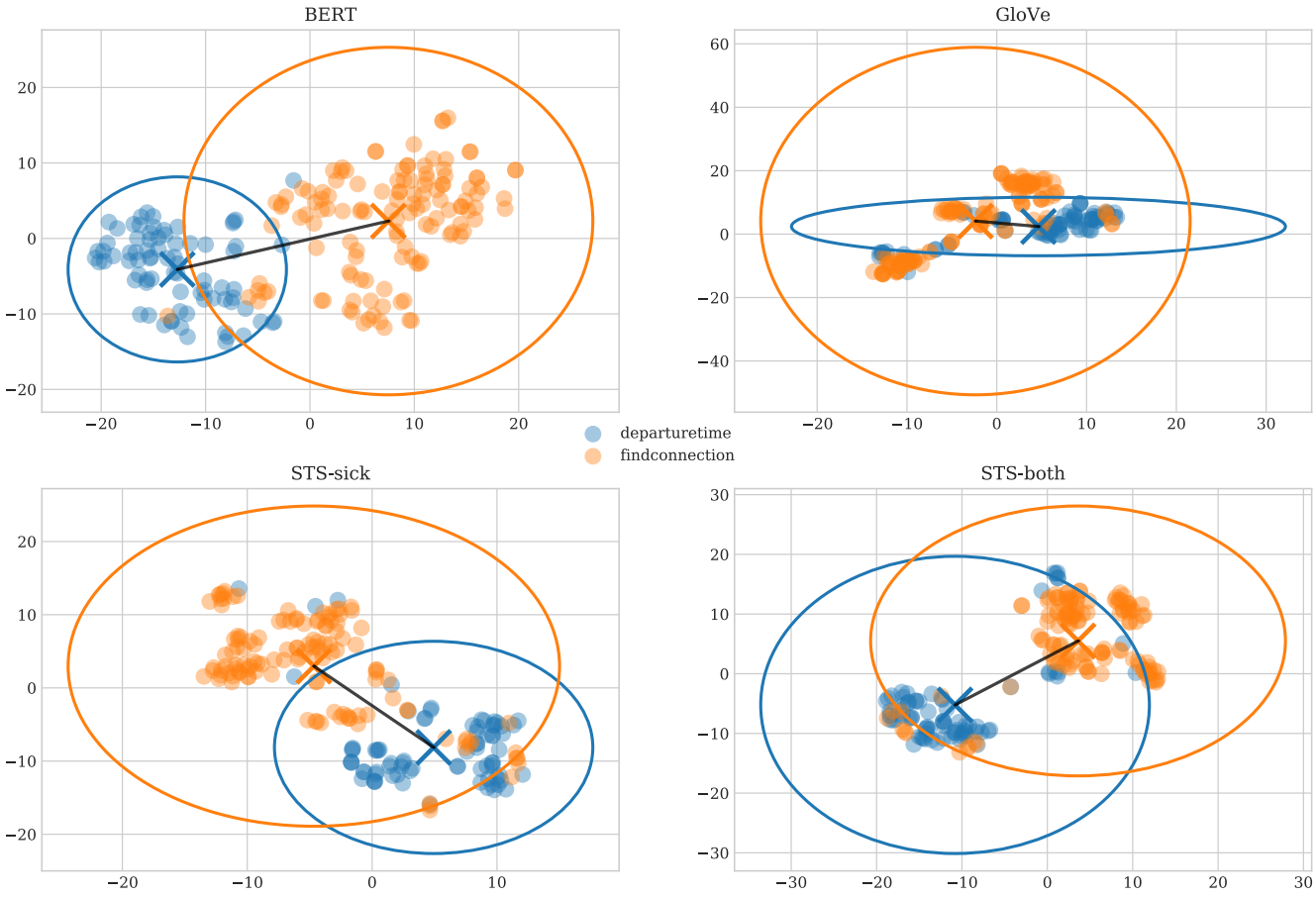
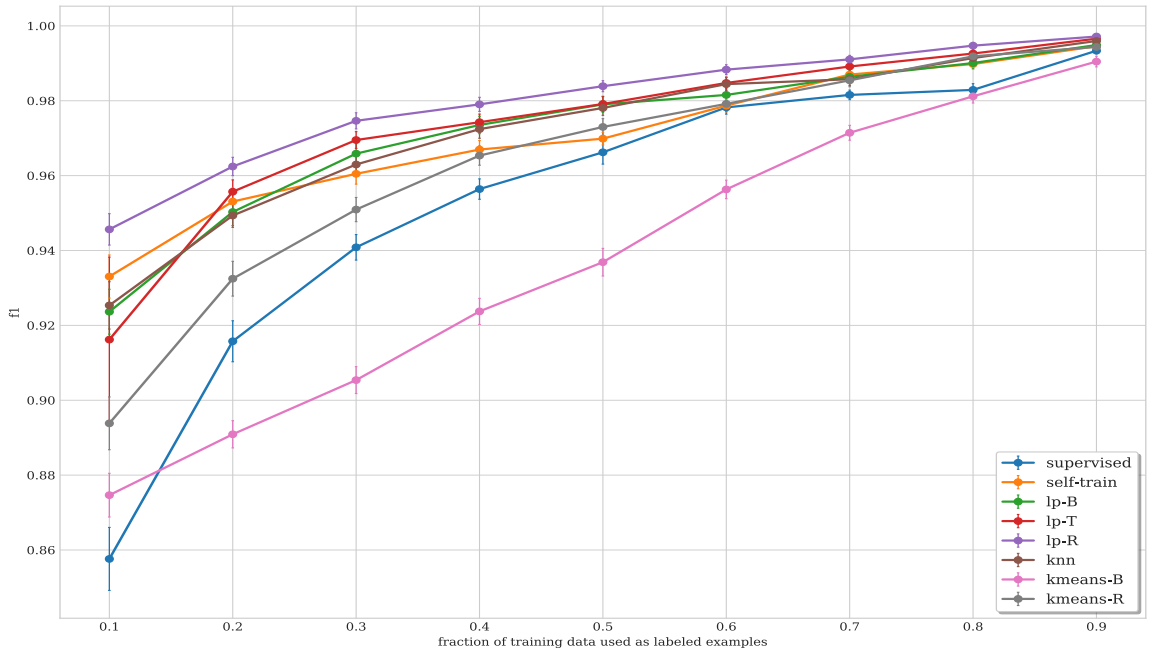


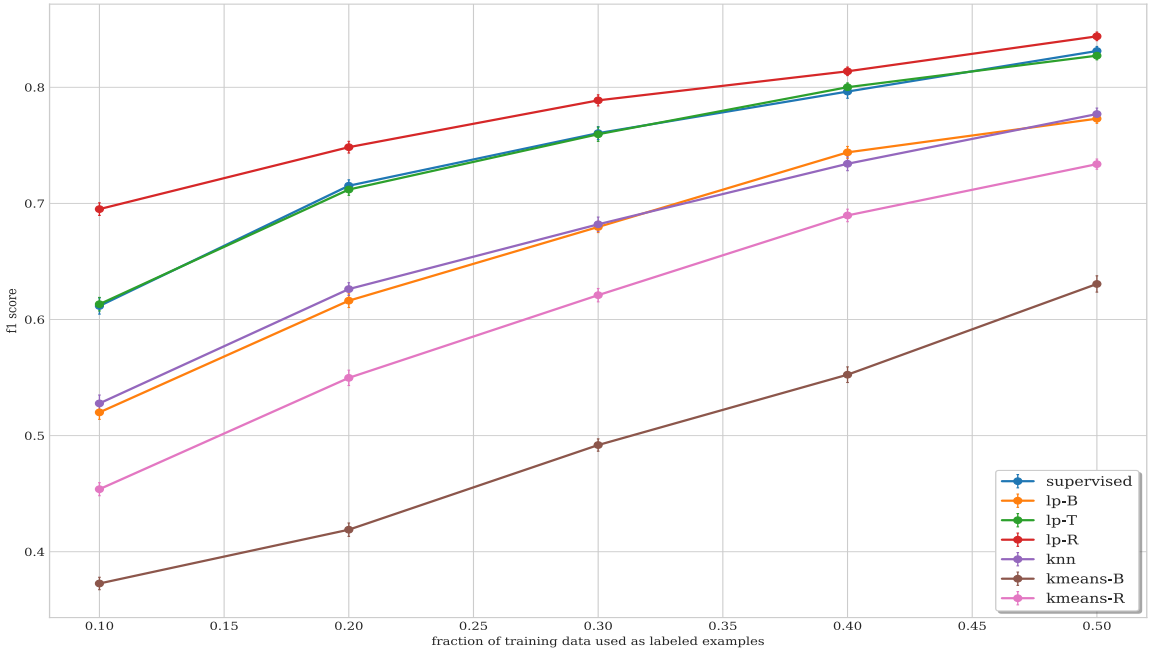
Figure 6.1: The intent-semantic representational ability of different embedding models on the Chatbot dataset. The x and y axes represent the first and second components of PCA+tSNE dimensionality-reduced embeddings; points are colored by intent; crosses represent the centroid of each intent cluster (with straight lines indicating the distance between them); ellipses represent the variance around each centroid. The I values of Table 6.1 correspond to the ratio of the size of the ellipses to the distance of the line between the centroids; the lower I values of BERT and STS-sick in Table 6.1 are reflected by more effective intent clustering.

6.2 Intent Classification

In this section we present the results of our intent classification experiments, a comparative evaluation of classification performance across four independent variable axes. We structure our results to demonstrate the impact of the two key variables – embedding type §6.2.2 and semi-supervised algorithm §6.2.1 – across different fractions of labeled data on each of the four datasets. Results in the transductive setting are presented in §6.2.3. In all experiments conducted, precision, recall, f1, and classification accuracy were closely related: all measures reflect the same relative results between methods. As such, we report f1 in all cases.



(a) Chatbot



(b) Sied

Figure 6.2: f1 scores for each semi-supervised algorithm on the Chatbot and Sied datasets, averaged across embedding types. All results obtained are the mean of 200 trials; 95% confidence intervals are shown.

6.2.1 Semi-Supervised Algorithm Comparison

The core hypothesis of our work was that by exploiting the knowledge contained in pretrained sentence embeddings to construct a representation graph, recursive label propagation could be used to substantially improve few-shot intent classification. In this section, we average over the seven embedding types to compare the performance of the six semi-supervised algorithms.

In Figure 6.2 we present the f1 scores achieved by the supervised baseline and the different methods on the Chatbot and Sied datasets (corresponding plots for the other two datasets are provided in §10). As can be seen, lp-R (RIPPED) is the most effective method. It improves classification over the supervised baseline across all fractions of labeled data, with the biggest improvements coming when only given access to a few labeled examples per class. For reference, the lowest fractions tested for the two given datasets correspond to roughly five and four labeled examples per class respectively. Furthermore, notice that on the challenging 21-class Sied dataset, RIPPED was the only method able to improve over the supervised baseline at all. Many-class classification problems with challenging decision boundaries are known to be challenging for semi-supervised algorithms, so this is an impressive result indicating the power of our proposed method.

To summarise the holistic performance gains achieved by each method, we compute a measure representing this as a single value for each dataset. To this end, we use the area between the f1 curve of each augmentation method and the supervised baseline, calculated using the trapezoidal rule[21]. Values for each method on each dataset are presented in Table 6.2. As can be seen, lp-R gives the best performance across the board, especially on the two most challenging datasets where other methods can't outperform the baseline. lp-R achieves a gain in classification f1 of at least 9.7 percentage points over the best semi-supervised baseline across the board!

Method	Avg.	Chatbot	AskUbuntu	Webapps	Sied
self-train	–	12.2	–	–	–
knn	−7.39	14.0	8.55	−35.3	−16.8
kmeans-B	−36.2	−4.14	8.43	−56.6	−92.3
kmeans-R	−17.0	8.34	13.1	−46.8	−42.8
lp-B	−7.33	13.9	12.3	−37.5	−18.0
lp-T	11.6	15.1	16.5	7.50	7.21
lp-R	**17.5**	**19.9**	**22.8**	**9.02**	**18.3**

Table 6.2: The area between the f1 curves of each augmentation method and the supervised baseline, averaged across embedding models. This gives a holistic measure of how much a given method can improve performance over a basic fully-supervised classifier; -ve values indicate worse performance. The areas are scaled to be in $[0, 100]$, so they represent the total gain in f1 score across all fractions of labeled data. Results for each augmentation method are the mean of 200 trials. Avg. is the unweighted mean across the four datasets; bold denotes the best result per dataset; underline denotes best in category (our lp methods vs others).

To further understand the success of RIPPED, we present in Figure 6.3 the trade-off between augmentation accuracy and the fraction of unlabeled data used in the augmentation. To average across datasets, we use the five lowest fractions of labeled data tested for each (e.g. the lowest labeled fraction is 0.1 for Chatbot and 0.4 for Webapps; these are averaged with the corresponding

[21] `https://en.wikipedia.org/wiki/Trapezoidal_rule`

fractions in the other two datasets and reported as index 1 on the x axis).

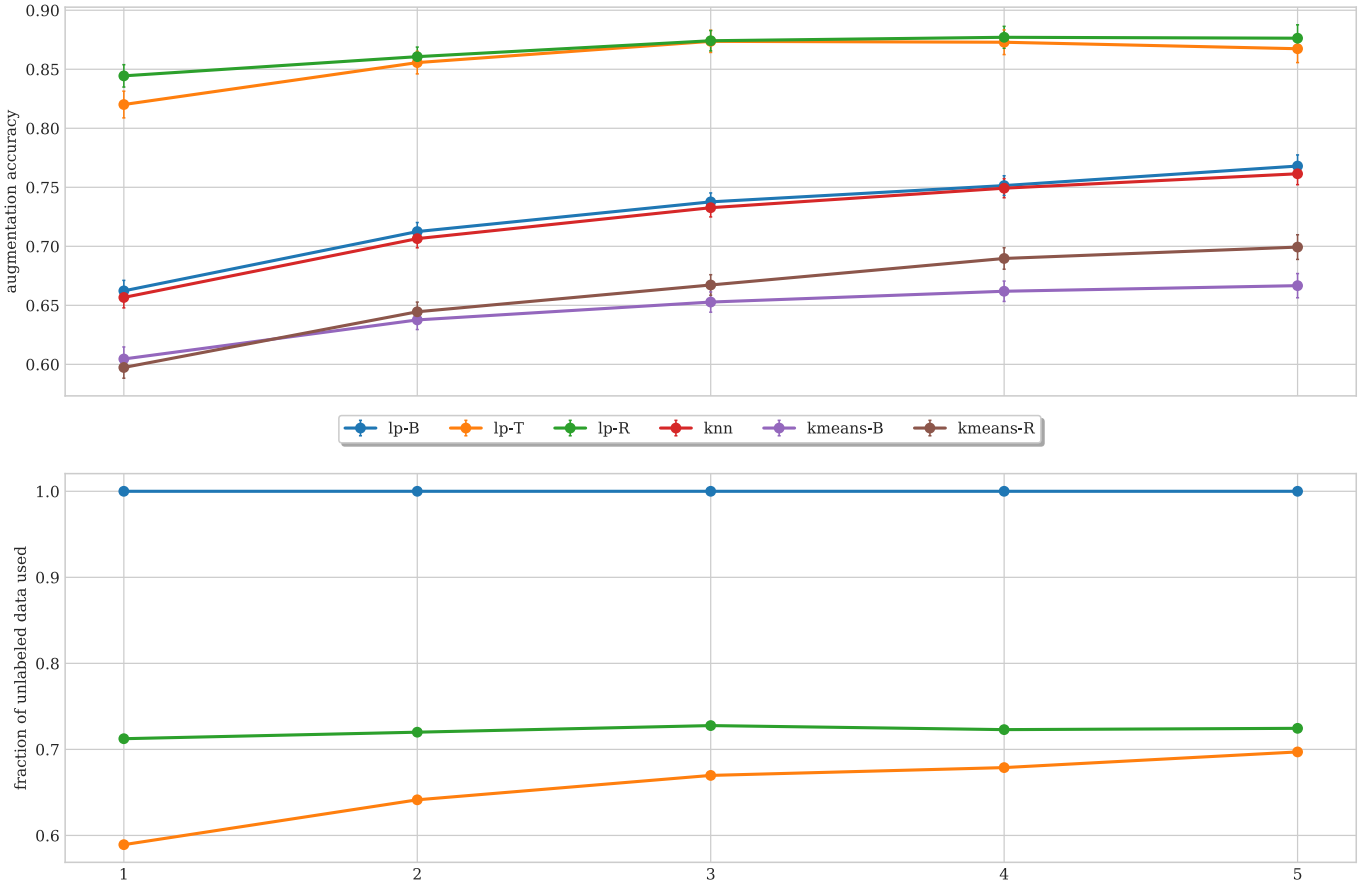


Figure 6.3: Augmentation accuracy and fraction of unlabeled data used for all augmentation methods, averaged across embedding types and datasets. All results obtained are the mean of 200 trials; 95% confidence intervals are shown. The x axis contains indices corresponding to the lowest five labeled fractions used for each dataset.

As can be seen, the augmentation accuracies of the lp-R and lp-T variants are both substantially higher than those of the other semi-supervised approaches, the reason for which becomes apparent when looking at the fraction-used sub-figure: these other approaches all use 100% of the unlabeled data. Due to the large number of classes and complexity of the problem space in these datasets, methods attempting to label all unlabeled data are unable to do so to a high accuracy. This was our motivation for the two lp variants in the first place. Furthermore, this diagram demonstrates that the superior f1 performance of lp-R over lp-T is due to the former attaining the same levels of augmentation accuracy while utilising a larger fraction of the unlabeled data. In some sense, lp-T is 'missing out' on labeling some data it could make confident predictions about because it only makes a single label propagation pass. This is a clear demonstration of why we use lp-R in RIPPED, and corresponds to the propagation visualisation we presented in §4.3.2.

In Table 6.3, we present results in the one-shot setting, where we compare the classification f1

Method	Avg.	Chatbot	AskUbuntu	Webapps	Sied
supervised	61.7	78.3	57.9	72.4	38.0
knn	58.1	83.0	61.7	51.0	36.7
kmeans-B	54.5	81.7	67.6	41.0	27.4
kmeans-R	55.1	83.5	61.5	45.1	30.4
lp-B	58.1	82.1	62.1	51.7	36.4
lp-T	64.8	83.0	62.5	73.8	39.9
lp-R	**75.4**	**91.3**	**68.9**	**75.0**	**66.4**

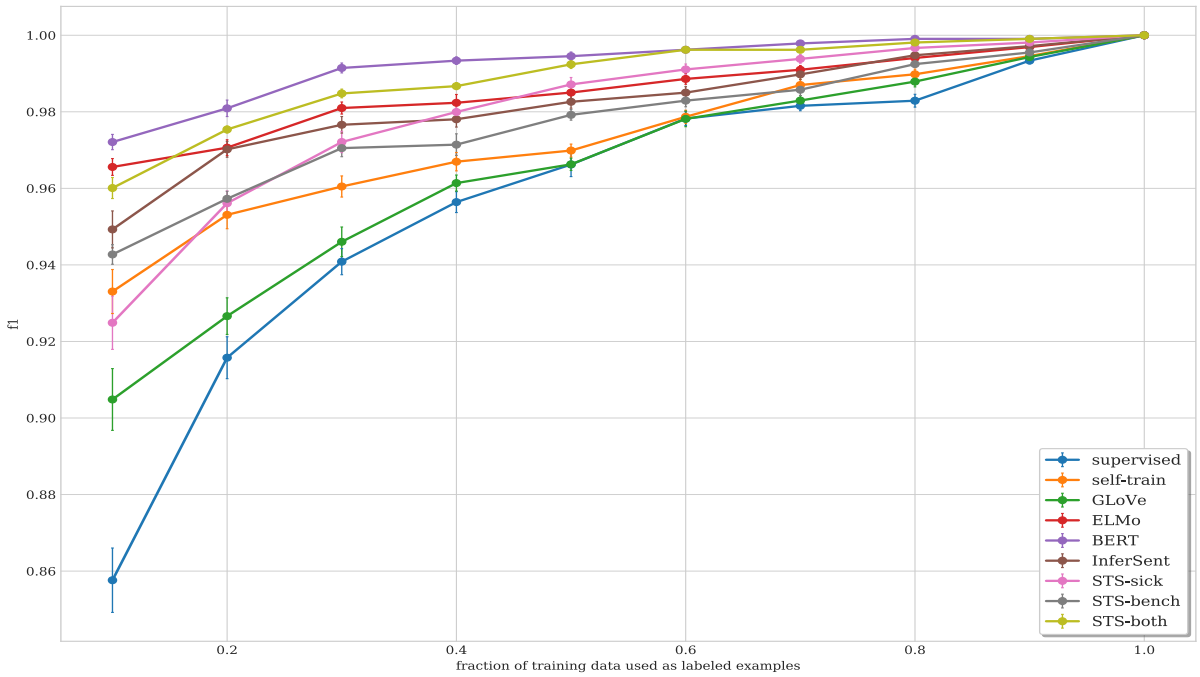
Table 6.3: One-Shot Performance: F1 scores for each augmentation method using a single labeled example per class. Reported results are averaged over embedding types, each embedding type + augmentation algorithm combination being tested for 200 trials. All embedding types were used excluding STS-sick, which showed v. poor performance with all methods. Avg. is the unweighted mean across the four datasets; bold denotes the best result per dataset; underline denotes best in category (our lp variants vs others).

obtained by the different semi-supervised algorithms on each dataset when given access to only one labeled example per class (and the rest of the complete training set as unlabeled data). Once again, these results are averaged across embedding types. The results reinforce our findings that lp-R is the most effective algorithm, especially in the most data-poor scenarios. Using lp-R, RIPPED improves classification f1 over the supervised baseline by 13.7 percentage points on average, an especially promising result given that most other methods are substantially worse than the fully-supervised classifier on the two most challenging datasets, Webapps and Sied.

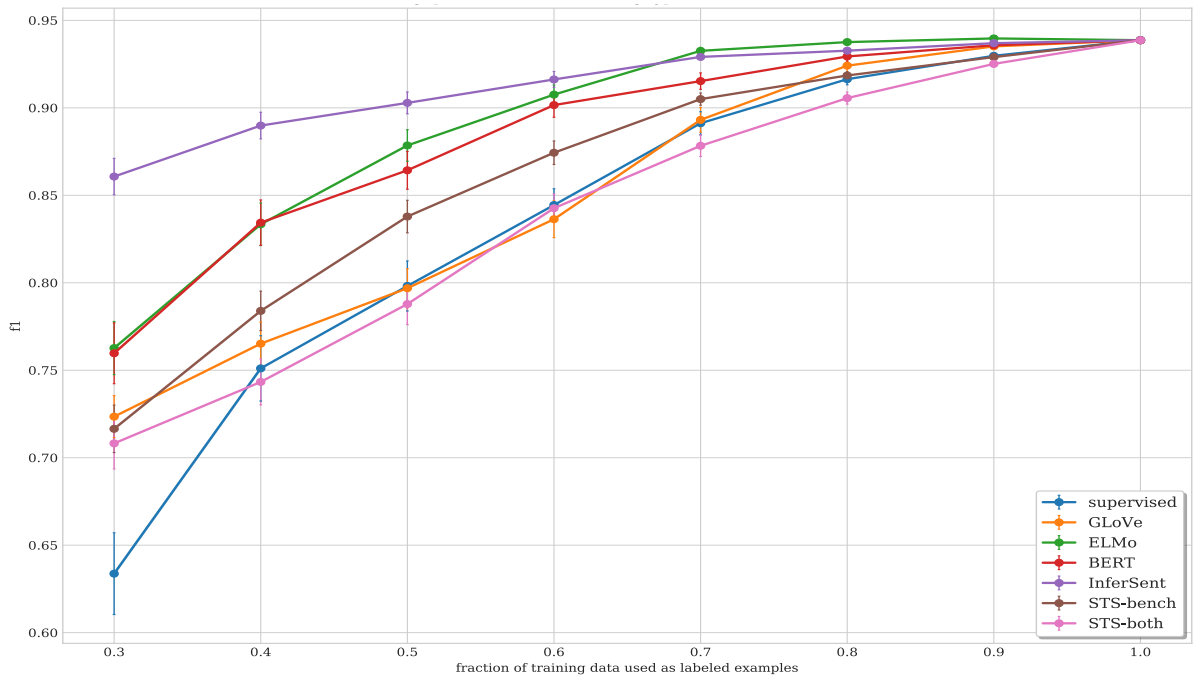
6.2.2 Embedding Type Comparison

Having compared the relative performance of the different semi-supervised algorithms (one half of each 'method'), we now present results outlining the impact of the embedding type on performance. In all cases in this section, we compare the performance of different embedding models using RIPPED, as the recursive LP variant it uses was by far the best-performing semi-supervised algorithm. Given that each pretrained embedding is used by RIPPED purely in order to compute pairwise distances, and that success depends on the ability of these distance values to correspond to the intent class semantics of a given dataset, classification performance is likely to show large variation depending on the embedding model used. As a result, the results in this section are important for two reasons: 1) we want to work towards a theory for selecting the best embedding model on a given dataset in advance, and 2) we want to evaluate the performance of our STS-trained embeddings in comparison with state-of-the-art universal models.

We present in Figure 6.4 example results from two datasets indicating classification f1 performance using RIPPED with each embedding type. We show the supervised baseline in both cases, as well as the self-training baseline for Chatbot. Similar diagrams for the other two datasets are presented in §10.



(a) Chatbot



(b) AskUbuntu

Figure 6.4: F1 scores using lp-R with each embedding type on the Chatbot and AskUbuntu datasets. STS-sick is excluded from (b) because of v. poor performance. All results obtained are the mean of 200 trials; 95% confidence intervals are shown.

As expected, the plots show substantial variation in performance depending on the embedding model used. In addition, it's clear that the performance of the best embedding type on each dataset is *significantly* higher than the averages using RIPPED reported in §6.2.1. For example, BERT on the Chatbot dataset achieves an f1 score of 99.5 using 50% of the training data labels, and 97.2 using only four labeled examples per class (a decrease of only 2.8% compared with state-of-the-art classification using the entire training dataset). Similarly, in the lowest fraction of labeled data tested on the AskUbuntu dataset (3 labeled examples per class), InferSent achieves an f1 score of 86.1, a decrease of only 7.7% compared with fully-supervised classification using the entire training dataset. Furthermore, it is clear that no embedding model is superior across the board: BERT is best on Chatbot, InferSent is best on AskUbuntu (and Sied, shown in §10), and ELMo is best on Webapps (also shown in §10). GloVe was by far the worst-performing universal embedding.

One of our hypotheses in this research was that simple sentence embedding models trained on the STS task would work well in our label propagation framework. While STS-sick and STS-both perform well on the Chatbot dataset (the latter being the second best method), our STS embeddings were significantly outperformed by the universal embeddings on the other three datasets. That being said, at least one of STS-bench or STS-both was able to outperform the supervised baseline on every dataset. As the success of RIPPED is heavily dependent on the strength of the representation graph constructed by a given embedding model, this suggests that training better embeddings specifically for our framework may be an avenue for further improvement.

To solidify these results, we report in Table 6.4 the average and maximum areas between the supervised baseline f1 curve and those using RIPPED with different embedding types (the averages are reproduced from Table 6.2). This demonstrates the scale of the classification performance improvements that are possible when using the best embedding model for a given domain.

Stat	**Avg.**	**Chatbot**	**AskUbuntu**	**Webapps**	**Sied**
lp-R avg	17.5	19.9	22.8	9.02	18.3
lp-R max	28.5	$22.4^{\triangleright}$	$44.8^{\star}$	$14.9^{\dagger}$	$32.0^{\star}$

Table 6.4: The average and maximum areas between the f1 curves of the supervised baseline and lp-R when using different embedding types. Avg. is the unweighted mean of the four datasets; $\star$ denotes that the result was obtained using InferSent embeddings, $\triangleright$ denotes BERT, † denotes ELMo.

6.2.3 Transductive Setting

To illustrate the advantage of the RIPPED framework, where semi-supervised learning is used as a data-augmentation process prior to the inductive training of a classifier, we provide transductive results for comparison. In Table 6.5 we present the percentage difference between the supervised baseline given access to all training data labels and the performance of each semi-supervised algorithm in the transductive setting. Reported results are averaged over the results of the four universal embedding types (the STS embeddings performed poorly in this setting). In this setting, no method was able to outperform a fully-supervised classifier trained on the complete training dataset. This corresponds to what we discovered in §6.2.1, whereby methods that label all unlabeled examples perform substantially worse than those that use thresholds to more confidently label a subset, and validates our implementation of RIPPED in the inductive setting.

Method	Chatbot	AskUbuntu	Webapps	Sied
supervised	99.5	93.9	83.6	90.0
lp-B	**−2.32**	−15.6	**−10.6**	**−7.17**
knn-B	−3.02	−26.5	−15.0	−8.03
kmeans-B	−16.4	**−15.2**	−32.3	−51.0
kmeans-R	−8.70	−31.0	−40.2	−19.5

Table 6.5: Transductive Setting: %Δf1 between the supervised baseline and the four valid augmentation methods when labelling all test examples using the entire training dataset as labeled data. Reported results are the average across the four universal embedding types, each of which is the mean of 50 trials. Bold denotes the best result per dataset, although all results are negative.

6.3 Impact of σ

In this section we present the results of our two experiments investigating the impact of the σ parameter on the performance of RIPPED.

6.3.1 Robustness

In Table 6.6, we present example results demonstrating the robustness of our σ-setting process. We report results using InferSent on the AskUbuntu dataset and ELMo on the Webapps dataset in this table, though other embedding model + dataset combinations were similarly robust. For each example, we ran 200 trials of the entropy heuristic σ-setting process for each of four different fractions of labeled data. We recorded the σ_0 value output by the MST heuristic and the σ value output by the entropy heuristic for every trial.

Heuristic	Stat	0.9	0.5	0.3	OS
MST	μ	0.174	0.195	0.212	0.226
	std%	5.63	7.00	7.94	9.61
Entropy	μ	0.066	0.069	0.070	0.071
	std%	3.02	1.27	1.24	1.20

(a) Infersent on AskUbuntu

Heuristic	Stat	0.9	0.7	0.5	OS
MST	μ	0.131	0.132	0.132	0.135
	std%	3.35	4.78	5.56	7.06
Entropy	μ	0.089	0.094	0.096	0.097
	std%	3.55	2.26	1.79	1.12

(b) ELMo on Webapps

Table 6.6: σ values generated by the MST and Entropy heuristics. The columns correspond to the fraction of the training dataset that was labeled (OS = one-shot); μ denotes the mean value of 200 trials; std% denotes the standard deviation as % of μ.

As expected, the MST heuristic becomes increasingly variable as the number of labeled examples decreases, as the σ_0 value this heuristic returns is wholly determined by the randomly-sampled labeled examples in a given trial. By contrast, the σ values generated by the entropy heuristic are extremely consistent, demonstrating a robustness to the different labeled example selection. The fact that σ values increase as the fraction of labeled examples decreases is also an intuitive

result for both heuristics. The value returned by the MST heuristic increases simply because the labeled dataset is less likely to contain any of the pairs of points that define the true shortest edges when given only a few labeled examples per class. For the entropy heuristic, the value returned increases because data points in different classes are likely to appear 'further away' than they really are if we only have a few labeled points, and therefore the assumed intent class structure will overestimate the distance between classes. Furthermore, note that the entropy heuristic σ values are substantially lower than those returned by the MST heuristic, reflecting the entropy minimisation process pushing σ towards a value that better corresponds to the true minimum distance between classes.

6.3.2 Performance

While illuminating, the robustness results reported above say nothing about the optimality of the entropy heuristic in terms of augmentation or downstream classification performance.

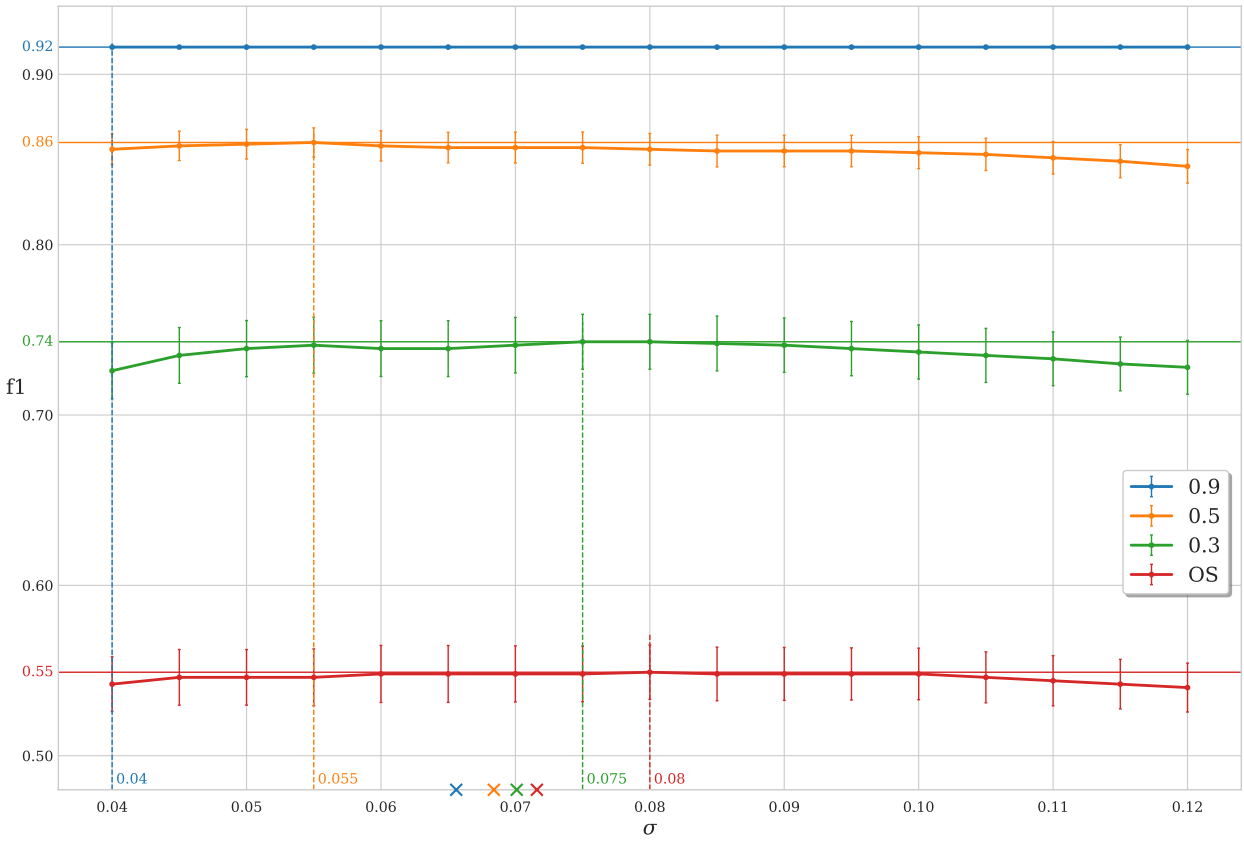


Figure 6.5: Impact of σ on classification f1 using RIPPED with InferSent on the AskUbuntu dataset. The different plots denote the fractions of training data that were labeled (OS=one-shot). For each fraction, the maximum f1 attained and corresponding σ value are shown with colored, axis-intercepting lines. The σ values set using our entropy heuristic are denoted by colored $\times$'s on the x axis. Error bars indicate 95% confidence intervals over the 200 trials.

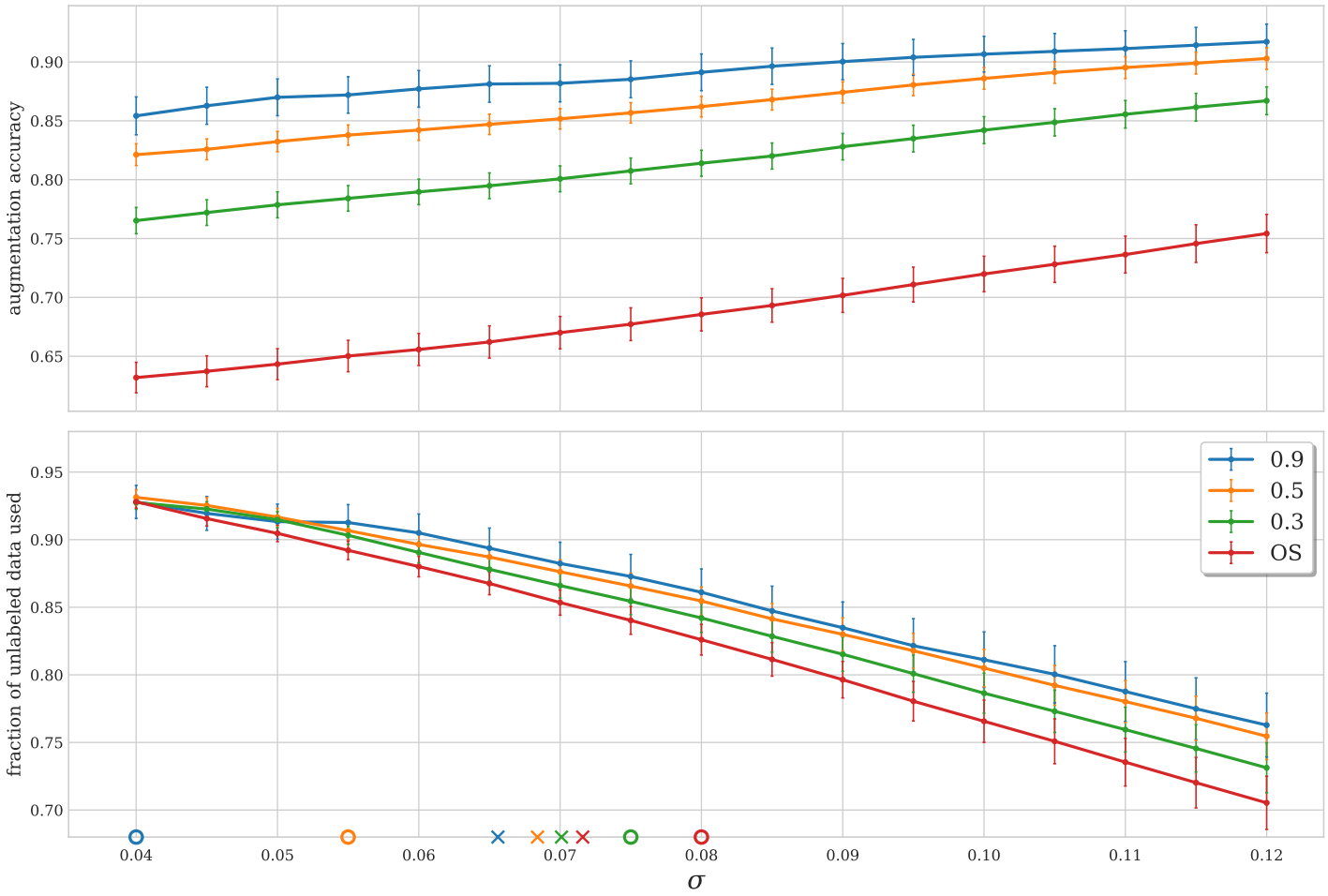


Figure 6.6: Impact of σ on the trade-off between augmentation accuracy & the fraction of unlabeled data used in the augmentation. The given results are obtained using RIPPED w. InferSent on the AskUbuntu dataset. The different plots denote the different fractions of training data that were labeled (OS=one-shot). The optimal σ values (in terms of f1 performance), $\bigcirc$, and those set by the entropy heuristic, $\times$, are shown for reference on the x axis. Error bars indicate 95% confidence intervals over the 200 trials.

To this end, we performed a simple ablation study evaluating the impact of σ in terms of 1) the classification f1 performance resulting from the augmentation (Figure 6.5), and 2) the trade -off between augmentation accuracy and the fraction of unlabeled data used in the augmentation (Figure 6.6). Both experiments are performed using RIPPED with a range of σ values on four different fractions of labeled data. Our reported results here correspond to using InferSent embeddings on the AskUbuntu dataset, though other embedding type + dataset combinations yielded comparable results.

As can be seen, the f1 performance of the downstream intent classifier is fairly robust to changes in σ, especially for the higher fractions of labeled data. However, there is a sharp drop in performance that continues beyond the right-hand edge of the plot. This demonstrates that the optimal σ values are far lower than those returned by the MST heuristic (given in Table 6.6). In addition, we can see that the optimal σ values are higher for smaller fractions of labeled data, as expected. In terms of the σ values output using our entropy heuristic, these have the same relative ordering as the optimal

values, but are too tightly clustered to approach those values. However, despite the large differences between our σ values and the optimal ones, the percentage improvement in classification f1 by using the optimal σ over our entropy-selected value is less than 0.4% on all four fractions of labeled data.

In terms of the augmentation trade-off, there is a clear inverse relationship between these two measures: if the algorithm uses more unlabeled examples (all else being equal), the accuracy of the augmentation will be worse. In addition, it's clear from looking at the these two measures for the optimal σ values that the ideal balance between augmentation accuracy and augmentation fraction used changes depending on the fraction of labeled examples given. When given fewer labeled examples, it's preferable for RIPPED to focus on accuracy while labeling a smaller chunk of the unlabeled dataset. This is an intuitive result. The label propagation algorithm works by slowly pushing ground-truth labels through dense regions of the embedding space, and the further that labels are propagated through the graph, the less confident the algorithm becomes. If the σ parameter didn't assume larger distances between classes when the algorithm knows very little about the overall class structure (in the few labeled example settings), RIPPED would confidently label many more examples incorrectly. The trade-off between these two measures is explicitly controlled by σ, and is ultimately what determines the performance of the intent classification.

However, even through the augmentation trade-off shows significant variability, it's clear that RIPPED is generally robust to different ratios of augmentation accuracy to fraction used. Through comparing the two experiments, we can see that f1 performance is consistent even as augmentation accuracy and the unlabeled fraction change by 10%. One reason for this is that we use RIPPED only as an augmentation tool prior to classifier training: by training a general classifier on the augmented labeled dataset (instead of classifying the test data directly), the learned classifier is able to see enough training examples to ignore the noise generated by a few incorrectly-labeled examples.

7 Discussion

In this section we provide some additional discussion around the results presented in §6, relating our findings to the original hypotheses and motivations behind this research.

The results presented comprehensively demonstrate the benefit of incorporating RIPPED into the intent classification pipeline in limited labeled data domains. Our method outperforms a range of competitive semi-supervised baselines on all datasets tested, and is robust to linguistic complexity, the number of classes in a given classification domain, and noisy class boundaries, especially given that all datasets tested have extremely unbalanced classes. These are all typically challenges that semi-supervised approaches struggle to overcome. We believe that both core components of RIPPED – the incorporation of pretrained embeddings and the lp-R algorithm – were significant in achieving this performance.

7.1 Recursive Label Propagation

As demonstrated by Figure 6.3, lp-R is able to maintain the highest augmentation accuracy of all methods while utilising a relatively high fraction of the unlabeled data. This corresponds to our hypotheses in designing this algorithm: 1) we suspected that attempting to make use of *all* unlabeled data in such complex decision spaces would degrade performance, and 2) we suspected that a thresholded, recursive implementation would be able extend propagation further from the initial labeled set than a non-recursive approach, while still being able to stop when labeling confidence decreases. Both hypotheses have been validated as a result of our experiments. However, we would like to conduct a more thorough investigation into point 2) in the future, exploring in detail the propagation process, the areas where RIPPED makes mistakes, and the relative propagation sequences of the lp-R, lp-T and lp-B variants.

7.2 Pre-trained Embedding Models

We believe that the incorporation of pretrained sentence representations into the label propagation setting was the most important factor underlying the strong classification results. In our approach, we utilised transfer learning in the few-shot domain by incorporating pretrained representations only in terms of pairwise distance values between them. There was no upfront guarantee that these distance calculations would even preserve the semantic knowledge contained in the embeddings, let alone accurately model the intent class semantics of various classification tasks. However, our results in §6.1 and §6.2.2 demonstrate that there is enough relevant semantic information encoded by these pairwise computations to understand complex, subtle class boundaries in a variety of linguistic domains.

We believe both ELMo and InferSent to be strong embeddings for use in our framework *in general*, as these express pairwise semantic information with no fine-tuning, but it's clear using the Chatbot dataset as an example that other embedding models can be substantially more effective in specific domains. A better understanding of why this is the case is one of the most important areas for future work. In addition, there is an extensive list of untested universal sentence embedding models that could be suitable for use in our framework. It is clear that the success of RIPPED is heavily dependent on the representation graph created by the untuned embeddings for a given task, and as such the choice of embedding model is the most important step in applying our method to new domains. As outlined in the next section, this is the most pressing area for future research.

7.3 Future Work

The most important area for future work is the development of a more principled theory around selecting the optimal sentence embedding model for a given problem domain. The core challenge in this task is that, when given very few labeled examples, it is challenging to predict whether a specific embedding model will be effective at representing the intent class semantics. Simple heuristics could be tested, such as a measurement similar to I that operates over clusters generated in an unsupervised manner. One approach that we are especially interested in is the development of strong embedding models for a given linguistic domain. SciBERT (Beltagy et al., 2019) is a recent example demonstrating the performance benefits of pretraining for a particular set of downstream tasks using carefully-selected training data that reflects the task domain. We also believe that our STS-training approach could be extended to yield better results in the future. For example, we could train a universal embedding model in a multi-task framework using the datasets compiled in our embedding evaluation in Table 3.1, thereby explicitly teaching a model to succeed at the tasks we consider salient indicators of its ability to represent semantic relations between sentences.

Secondly, there are several extensions to the recursive label propagation algorithm used in RIPPED that we are interested in exploring. These include using domain knowledge to perform class rebalancing over the output Y_U matrix (Zhu, 2002) and improving the core iterative algorithm to be more computationally efficient, thus allowing RIPPED to be easily applied to larger datasets. Perhaps the most interesting extension we are considering for future work is the use of different σ values for each embedding dimension. We could use σ_i, $i = 1, \ldots, d$, where all σ_i would be experimentally determined using a variation of the entropy minimisation process. In high-dimensional pretrained embeddings, it seems likely that many dimensions are irrelevant to the semantics of intent class boundaries in a specific downstream task, and therefore σ_i could be learned to let RIPPED only consider the most important dimensions.

In general, we are interested in applying RIPPED to other domains. Our choice of intent classification as an initial task by which to evaluate our method was motivated in part by the recent success of universal sentence embedding models, as well as by the challenges specific to this problem outlined in §3.1. As we have shown, constructing the representation graph using pretrained embedding knowledge dramatically improves performance, despite these embeddings not experiencing any domain-specific tuning. There doesn't seem to be any comparable 'universal' embeddings for document representation as of yet, though many approaches could be attempted in order to test RIPPED on the more general task of text classification.

Finally, there are several human-in-the-loop approaches that we believe could further improve the performance of RIPPED, an especially relevant area for future work given that our method is a tool for bootstrapping NLU systems. For example, the unlabeled examples that lp-R doesn't use (after all iterations) could be annotated by a human, or a human could be asked to select representative examples of each class from a larger unlabeled dataset to use as our initial labeled data (similar to (Bailey and Chopra, 2018)).

8 Conclusion

In order to work towards general conversational intelligence, we require the ability to develop natural language understanding systems in novel, data-poor domains. Based on the assumption that the meaning stored in pretrained representations of text contain enough knowledge to understand the semantic structure of a specific intent classification domain, we have proposed RIPPED, a method for substantially improving few-shot intent classification performance. In designing our approach to specifically address the unique challenges of intent classification, we have developed that can be used for boostrapping NLU in new domains, beginning with only a few labeled examples per class.

We have demonstrated the effectiveness of our method in substantially improving intent classification performance in a variety of data-poor domains, and we have presented exploratory results indicating the method's robustness to novel settings. The empirical results are strong, but we are excited about the future work that needs to be done in order to better understand the conditions in which RIPPED will thrive. In particular, we need to develop a better theoretical understanding of the optimal embedding-selection process, as this will allow RIPPED to be easily applied to new domains with no expert knowledge.

9 Bibliography

S. Arora, Y. Liang, and T. Ma. A simple but tough-to-beat baseline for sentence embeddings. In *ICLR*, 2017.

D. Arthur and S. Vassilvitskii. K-means++: the advantages of careful seeding. In *SODA*, 2007.

K. Bailey and S. Chopra. Few-shot text classification with pre-trained word embeddings and a human in the loop. *CoRR*, abs/1804.02063, 2018.

I. Beltagy, A. Cohan, and K. Lo. Scibert: Pretrained contextualized embeddings for scientific text. *CoRR*, abs/1903.10676, 2019.

D. Braun, A. Hernandez-Mendez, F. Matthes, and M. Langen. Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 174–185, SaarbrÃijcken, Germany, August 2017. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W17-3622`.

D. M. Cer, M. T. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia. Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *SemEval@ACL*, 2017.

O. Chapelle and A. Zien. Semi-supervised classification by low density separation. In *AISTATS*, 2005.

C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, and P. Koehn. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH*, 2013.

A. Chronopoulou, C. Baziotis, and A. Potamianos. An embarrassingly simple approach for transfer learning from pretrained language models. *CoRR*, abs/1902.10547, 2019.

A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*, 2017.

A. Conneau, G. Kruszewski, G. Lample, L. Barrault, and M. Baroni. What you can cram into a single vector: Probing sentence embeddings for linguistic properties. In *ACL*, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

E. Ferrara, O. Varol, C. A. Davis, F. Menczer, and A. Flammini. The rise of social bots. *Commun. ACM*, 59:96–104, 2016.

A. Gammerman, V. Vovk, and V. Vapnik. Learning by transduction. In *UAI*, 1998.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011.

H. S. Gowda, M. Suhil, D. S. Guru, and L. N. Raju. Semi-supervised text categorization using recursive k-means clustering. In *RTIP2R*, 2016.

T. Jebara, J. Wang, and S.-F. Chang. Graph construction and b-matching for semi-supervised learning. In *ICML*, 2009.

T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *ECML*, 1998.

R. Johnson and T. Zhang. Semi-supervised convolutional neural networks for text categorization via region embedding. *Advances in neural information processing systems*, 28:919–927, 2015.

R. Johnson and T. Zhang. Supervised and semi-supervised text categorization using lstm for region embeddings. In *ICML*, 2016.

S. S. Keerthi, S. Sellamanickam, and S. K. Shevade. Extension of tsvm to multi-class and hierarchical text classification problems with general losses. In *COLING*, 2012.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In *Proceedings of the American Mathematical Society, 7*, 1956.

X. Li and D. Roth. Learning question classifiers. In *COLING*, 2002.

Y.-F. Li and Z.-H. Zhou. Improving semi-supervised support vector machines through unlabeled instances selection. In *AAAI*, 2011.

B. Liu, W. S. Lee, P. S. Yu, and X. Li. Partially supervised classification of text documents. In *ICML*, 2002.

M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. Bernardi, and R. Zamparelli. A sick cure for the evaluation of compositional distributional semantic models. In *LREC*, 2014.

P.-E. Mazaré, S. Humeau, M. Raison, and A. Bordes. Training millions of personalized dialogue agents. In *EMNLP*, 2018.

G. A. Miller. Wordnet: A lexical database for english. In *Proceedings of the Workshop on Speech and Natural Language*, 1992.

K. Nigam, A. T. McCallum, and T. M. Mitchell. Semi-supervised text classification using em. In *Semi-Supervised Learning*, 2006.

S. Pawar, N. Ramrakhiyani, S. Hingmire, and G. K. Palshikar. Topics and label propagation: Best of both worlds for weakly supervised text classification. In *CICLing*, 2016.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

C. S. Perone, R. Silveira, and T. S. Paula. Evaluation of sentence embeddings in downstream and linguistic probing tasks. *CoRR*, abs/1806.06259, 2018.

M. Peters, S. Ruder, and N. A. Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. *CoRR*, abs/1903.05987, 2019.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. S. Zettlemoyer. Deep contextualized word representations. In *NAACL-HLT*, 2018.

J. K. Pritchard, M. Stephens, and P. Donnelly. Inference of population structure using multilocus genotype data. *Genetics*, 155 2:945–59, 2000.

P. K. Pushp and M. M. Srivastava. Train once, test anywhere: Zero-shot learning for text classification. *CoRR*, abs/1712.05972, 2017.

A. Radford. Improving language understanding by generative pre-training. 2018.

S. Ruder. *Neural Transfer Learning for Natural Language Processing*. PhD thesis, National University of Ireland, Galway, 2019.

S. Ruder and J. Howard. Universal language model fine-tuning for text classification. In *ACL*, 2018.

R. Sennrich, B. Haddow, and A. Birch. Improving neural machine translation models with monolingual data. *CoRR*, abs/1511.06709, 2016.

V. Sindhwani and S. S. Keerthi. Large scale semi-supervised linear svms. In *SIGIR*, 2006.

I. Triguero, S. García, and F. Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, 42:245–284, 2013.

A. M. Turing. Computing machinery and intelligence. 1950.

O. Vinyals and Q. V. Le. A neural conversational model. *CoRR*, abs/1506.05869, 2015.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461, 2018.

B. Wang, Z. Tu, and J. K. Tsotsos. Dynamic label propagation for semi-supervised multi-class multi-label classification. In *ICCV*, 2013.

Z. Wang, H. Mi, and A. Ittycheriah. Semi-supervised clustering for short text via deep representation learning. In *CoNLL*, 2016.

J. W. Wei and K. Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *CoRR*, abs/1901.11196, 2019.

J. Wieting, M. Bansal, K. Gimpel, and K. Livescu. Towards universal paraphrastic sentence embeddings. *CoRR*, abs/1511.08198, 2016.

A. Williams, N. Nangia, and S. Bowman. 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, 2018. URL `http://aclweb.org/anthology/N18-1101`.

Z. Yang, W. W. Cohen, and R. R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

M. Yu, X. Guo, J. Yi, S. Chang, S. Potdar, Y. Cheng, G. Tesauro, H. Wang, and B. Zhou. Diverse few-shot text classification with multiple metrics. In *NAACL-HLT*, 2018.

X. Zhang, J. J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *NIPS*, 2015.

L. Zhou, J. Gao, D. Li, and H. Shum. The design and implementation of xiaoice, an empathetic social chatbot. *CoRR*, abs/1812.08989, 2018.

X. Zhu. Learning from labeled and unlabeled data with label propagation. In *CMU CALD tech report CMU-CALD-02-107, 2002*, 2002.

X. Zhu, T. Li, and G. Melo. Exploring semantic properties of sentence embeddings. In *ACL*, 2018.

Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

10 Supplementary Material

This section contains a variety of supplementary information referenced in the paper. Information appears in the order it was referred to.

10.1 Pretrained Embedding Evaluation Sources

Emb	STS		Inf			Para		Probe
	SICK-R	**STS-B**	**SICK-E**	**QNLI**	**MNLI**	**MRPC**	**QQP**	**SOMO**
GloVe	Per	Per	Per	Glu	Glu	Glu	Glu	Per
ELMo	Pet	Pet	Pet	Elm	Pet	Pet	Glu	Pet
BERT	Pet	Pet	Pet	Ber	Pet	Pet	Glu	N/A
InfSnt	Per	Per	Per	Glu	Glu	Glu	Glu	Per

Table S1: Sources for the results compiled in our comparison of universal embeddings. Per=(Perone et al., 2018), Pet=(Peters et al., 2019), Glu=(Wang et al., 2018), Elm=(Peters et al., 2018), Ber=(Devlin et al., 2018).

10.2 Dataset Intent Statistics

Dataset	Intent	# Train	# Test
Chatbot	'DepartureTime'	43	35
	'FindConnection'	57	71
	$\sum$	100	106
AskUbuntu	'MakeUpdate'	10	37
	'SetupPrinter'	10	13
	'ShutdownComputer'	13	14
	'SoftwareRecom'	17	40
	'None'	3	5
	$\sum$	53	109
Webapps	'ChangePassword'	2	6
	'DeleteAccount'	7	10
	'DownloadVideo'	1	0
	'ExportData'	2	3
	'FilterSpam'	6	14
	'FindAlternative'	7	16
	'SyncAccount'	3	6
	'None'	2	4
	$\sum$	30	54
Sied	'assist'	16	2
	'bot'	20	3
	'how'	9	2
	'justDetails'	4	1
	'name'	14	2
	'notGiving'	8	2
	'query'	5	1
	'wait'	5	1
	'contact'	108	14
	'aadhMissing'	28	4
	'addressProof'	47	6
	'applicationProcess'	162	21
	'applyRegister'	49	7
	'approvalTime'	49	7
	'badService'	11	2
	'bankOptionMissing'	43	6
	'bizCategoryMissing'	32	4
	'bizNew'	54	7
	'bizSampler'	75	10
	'borrowLimit'	60	8
	'borrowUse'	82	11
	$\sum$	881	121

Table S2: Dataset Statistics - Intent Breakdown

10.3 Additional Training Details

Supervised Classifier Grid Search

We performed a standard grid search, using k-fold cross validation with $k = 5$. For both the multinomial Naive Bayes and the Stochastic Gradient Descent classifiers we searched over the following parameters for the input representation: input format {word,char,char+wb}; max document frequency in [0.25,1]; all ngram ranges combinations between 1 and 5; use tfidf in {True,False}; tfidf normalisation in {l1,l2}. We searched over a wide range of α values both classifiers, as well as all different loss functions, and normalisation for SGD.

For the MLP classifier, we tested using either zero (logistic regression) or one hidden layer, in the latter case with sizes from 5 to 100. We tried BoW and word embedding input formats, as well as a range of non-linearities. For the RNN, we tested a range of values for each of the standard parameters: hidden dim, input representation, embedding initialisation, output representation etc.

The optimal classifiers for each dataset we use are as follows:

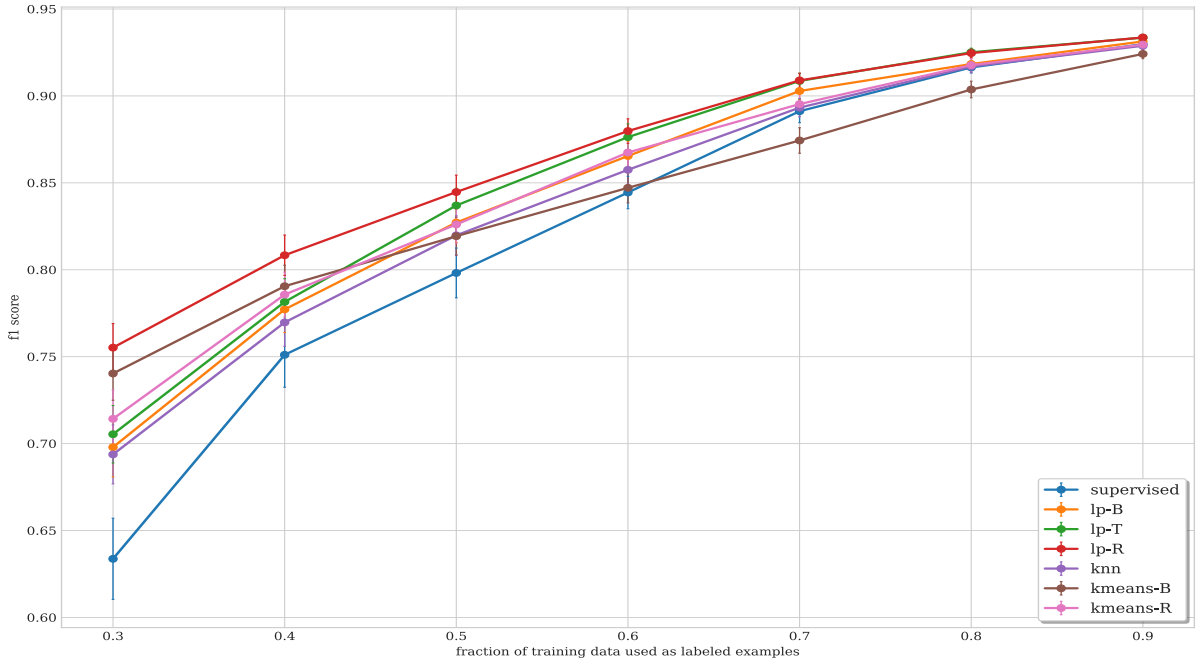
- **Chatbot** Tf-idf over character n-grams of range (2,3) with l2 norm and max doc frequency of 0.5. Naive Bayes classifier with $\alpha = 1$
- **AskUbuntu** Tf-idf over character n-grams of range (2,3) with l2 norm and max doc freq of 0.5. Naive Bayes classifier with $\alpha = 0.1$
- **Webapps** Tf-idf over character n-grams of range (2,3) with l2 norm and max doc freq of 0.5. SGD classifier with hinge loss, l2 normalisation, optimal learning rate schedule, and $\alpha = e^{-2}$.
- **Sied** Tf-idf over character n-grams of range (2,5) with l2 norm and max doc freq of 0.5. SGD classifier with hinge loss, l2 normalisation, optimal learning rate schedule, and $\alpha = 1e - 4$

These were used in all experiments as both the supervised baselines and the classifiers used in conjunction with the various semi-supervised methods.

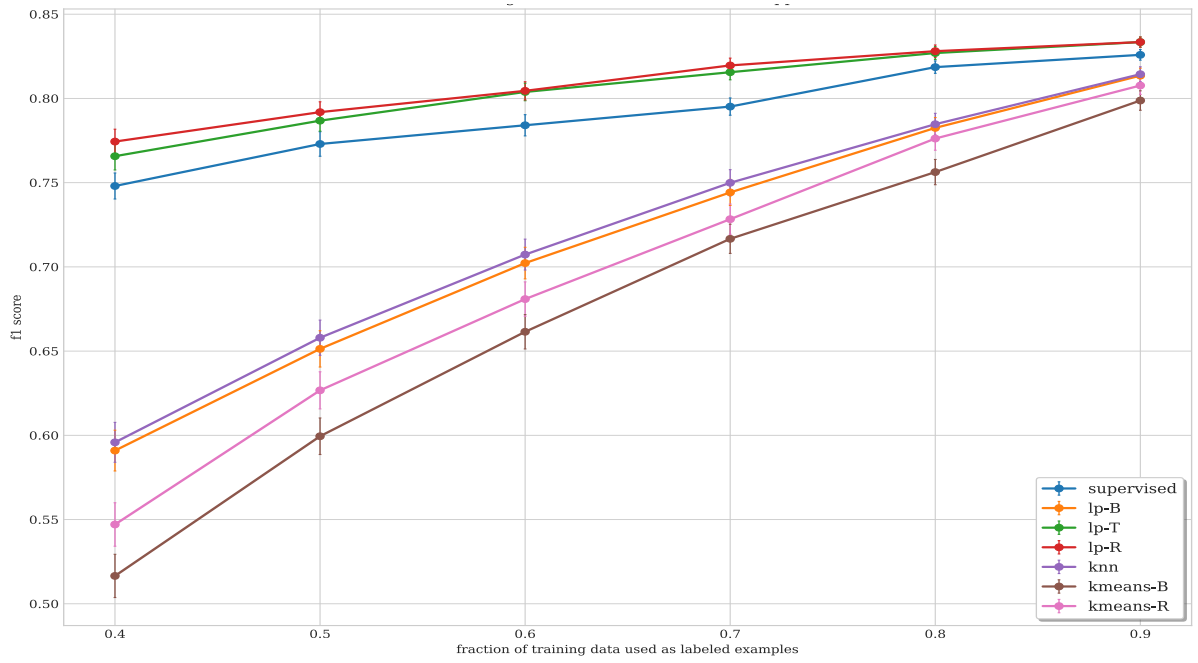
KNN Grid Search

The parameter values searched over for each dataset were: $k \in [1, 10]$, weights $\in \{\text{uniform}, \text{distance}\}$, algorithm $\in \{\text{auto}, \text{ball-tree}, \text{brute}\}$. In all cases, the optimal KNN-classifier used $k = 1$, uniform weights, and the default algorithm.

10.4 Semi-Supervised Algorithm Comparison - Extra Examples



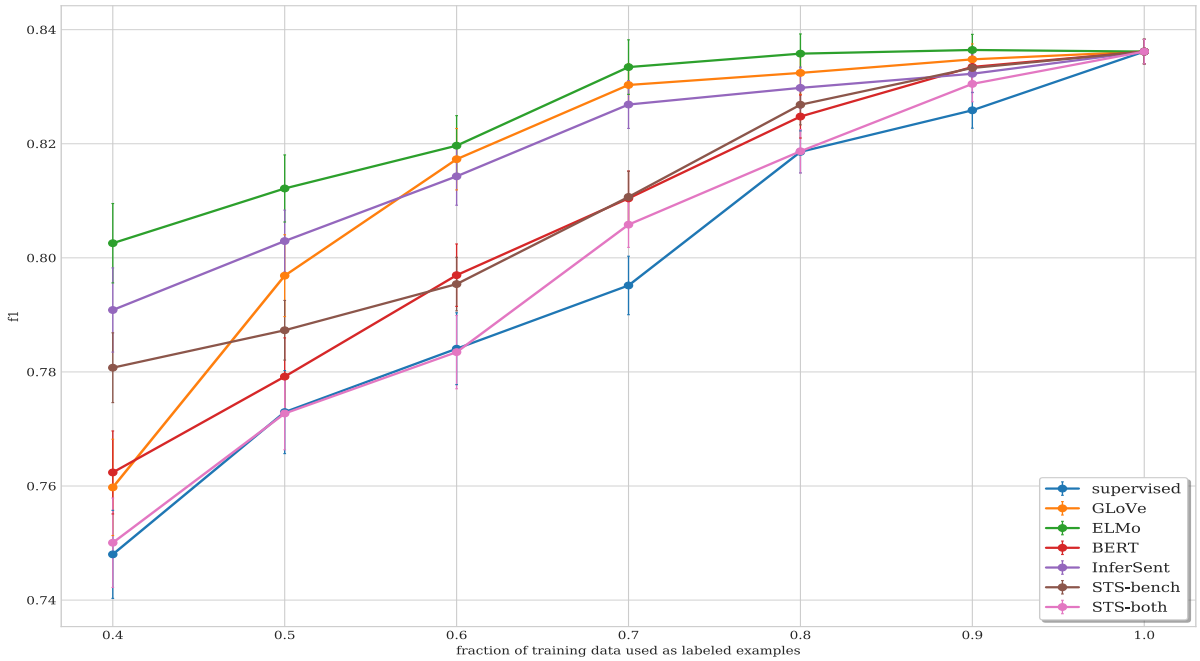
(a) AskUbuntu



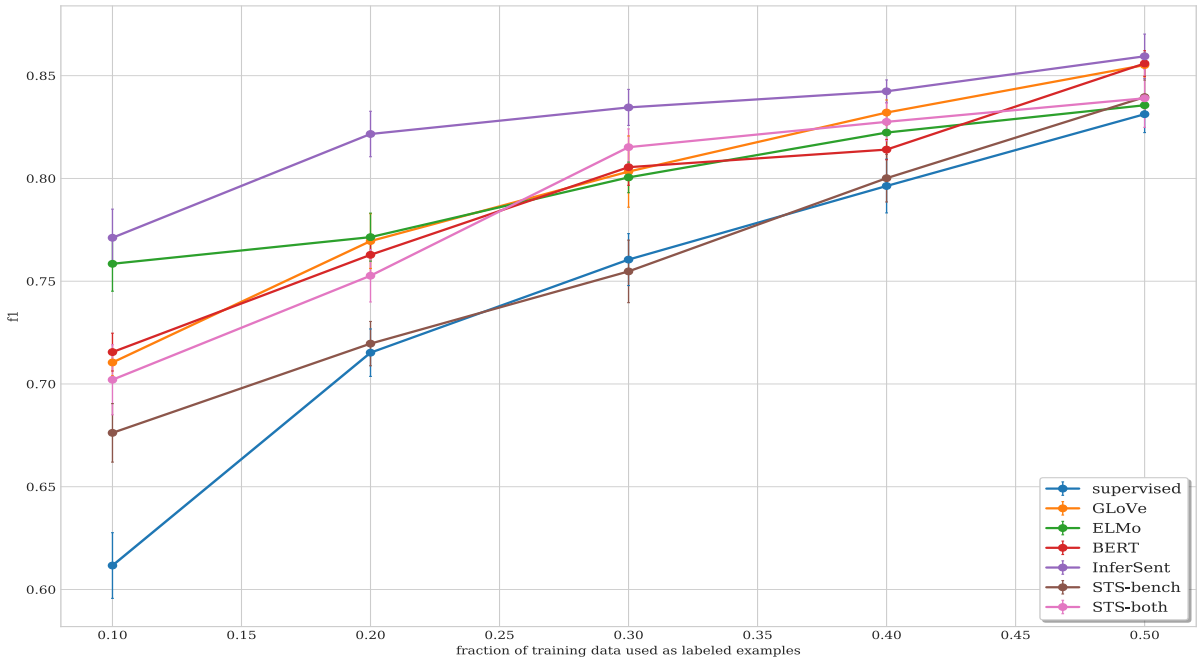
(b) Webapps

Figure S1: f1 scores for each semi-supervised algorithm on the AskUbuntu and Webapps datasets, averaged across embedding types. All results obtained are the mean of 200 trials; 95% confidence intervals are shown.

10.5 Embedding Type Comparison - Extra Examples



(a) Webapps



(b) Sied

Figure S2: F1 scores using lp-R with each embedding type on the Webapps and Sied datasets. STS-sick is excluded from both because of v. poor performance. All results obtained are the mean of 200 trials; 95% confidence intervals are shown.