Machine Learning Engineer Nanodegree

Capstone Project

John Loutzenhiser January 20, 2019

Evaluating Pretrained Language Models for Short Text Classification on Small Datasets

I. Definition

Project Overview

Text classification is a well-studied and widely used practical application of machine learning technologies ([Giorgino 2004][]). In a text classification task, one starts with a corpus of documents or texts which have been categorized or labeled according to some criteria. This pre-categorized corpus is used as training data to train a machine learning model which is then able to predict the category or class of new, unseen documents.

Examples of text classification tasks that I have recently been involved with are:

- automatic quality control of vocational course descriptions uploaded from vocational schools from all over Germany into a central course description database
    - Are the documents plausible as a vocational course description? (yes/no classification)
    - Which type of qualifications does a course description offer (multiclass, or one of many classification)
- automatic recognition of university enrollment certificates scanned and uploaded by citizens applying for government benefits.
    - is an uploaded document plausibly an authentic university enrollment certificate? (yes/no classification)

For these classification tasks, there were relatively large amounts of training data available - in the range of many thousands up to 1 million examples. With this amount of training data, it was relatively easy to achieve good prediction results quickly by training classifier models using the "Bag of Words" (BOW) technique using a Support Vector Machine, Naive Bayes, or Logistic Regression algorithms. (see [Wang and Manning 2012][] for a discussion of these strong baseline text classifiers)

However training robust text-classifiers that can generalize well and effectively classify a wide variety of input texts is a challenge when large training datasets are not available, which is often the case when developing a chatbot from scratch, for example. In my experience, a lack of readily available domain-specific training data is a significant hurdle to bootstrapping chatbots and getting them robust enough for production without significant data-collection an annotation effort.

It seems that when only small datasets are available, the Bag-of-Words approach might have some limitations. This is because Bag-of-words only considers the surface lexical aspects of a text (essentially the arrangement of letters into words) and no way of encoding lexical semantics (e.g. similar words), phrase structure (for example, negation "not a good book" means the opposite of "a very good book") or context. A toy dataset can demonstrate why this could be a limitation. Given the following labeled texts as training data:

Text	label
dogs should be walked every day	pet_care
indoor cats need activities to avoid boredom	pet_care
brush your teeth twice a day to keep your teeth clean	hygiene

How would a BOW classifier classify the following?

"hamsters need a clean cage"

Most likely "hygiene", as the word "clean" is shared and "hamster" is out-of-vocabulary for the training set. However, "pet_care" seems to be the correct class.

Although it might appear as if BOW classifiers might have limitations on small datasets, classifiers such as those in [Wang and Manning 2012][] still represent very strong benchmarks for text classification. So the question is, are more modern, sophisticated methods more powerful than these simple, well known models on small datasets?

Pretrained Language Models

A recent advancement in NLP has been the development of pretrained *language* models which can be used in transfer learning for a variety of NLP tasks including text classification. ([UlmFit][], [Elmo][], [BERT][])

These models represent not only lexical features (as is the case with Bag-of-Words) but also lexical and contextual semantics as well. Because of this, the potential impact of these language models for NLP is being compared to the impact pretrained image-recognition models have had for the field of computer vision (NLPs Imagenet Moment has Arrived)

The promise of pretrained language models is that they make it possible to fine-tune pretrained models with relatively small training sets, creating classifiers that might generalize well by leveraging features and context they learned through pre-training on huge datasets.

Problem Statement

I would like to determine how pretrained language models fine-tuned on small datasets measure up for text classification on short texts. The focus on relatively small datasets and rather short texts should simulate the real-world problem of bootstrapping a dialog system/chatbot when large training datasets are unavailable.

Can pretrained language models result in classifiers that can generalize well when fine-tuned on small datasets? Can they:

- classify texts with features that were not explicitly seen in the (fine-tuning) training set?
- "understand" out-of-vocabulary words?

Can these classifiers perform better than well-known benchmark classifiers?

In order to answer these questions, an evaluation is proposed where two modern pretrained language model-based classifiers will be compared to a common benchmark text classifier. In order to perform this evaluation, the following tasks were undertaken:

- Identify and acquire an appropriate dataset for training and evaluation. The dataset should consist of short texts taken from a real-world conversational or "chat" scenario. Each text must have a label or class assigned so that it can be used for supervised learning
- Using a supervised learning paradigm, train 3 text classifiers using training data in the dataset. The 3 classifiers are based on the following implementations:
    - **TF-IDF weighted word-grams fed into a Naive Bayes Classifier**. This implementation is based on `scikit-learn`, and represents a widely-used and strong baseline implementation for comparison
    - **BERT** - implementation based on the open-source release available at https://github.com/google-research/bert
    - **ULMFiT** - implementation based on the open source release found at https://github.com/fastai/fastai
- Each classifier is trained (or fine-tuned) on samples out of the total dataset of various sizes. The focus is on small samples, to simulate the situation of bootstrapping a chatbot from scratch, and to test the hypothesis that pretrained language models might indeed require less training (fine-tuning) than other classifiers to achieve similar or better results.
- Evaluate the performance of each classifier on training or holdout data in each dataset sample using the accuracy metric (percent of correctly classified texts).
- Examine the classification results more closely in order to determine to what extent pretrained language models were indeed able to "infer" the correct class in cases where training features were not present in the input sample (out-of-vocabulary)

Metrics

All models are evaluated on test data according to the accuracy metric. As **balanced datasets** are used in the training samples, the accuracy metric can be used to reliably predict the overall performance of the classifiers.

Accuracy is defined by the following:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

where:

- tp = true positives - number of instances with correctly predicted class label
- tn = true negatives - for a particular class, number of instances correctly predicted as not belonging to that class
- fp = false positives - for a class, number of instances incorrectly predicted as belonging to that class
- fn = false negatives - for a particular class, number of instances actually belonging to the class which were incorrectly predicted as not belonging to that class

II. Analysis

Data Exploration

For this evaluation I used the [amazon question/answer dataset][]. ([Wan and McAuley 2016][]), ([McAuley and Yan 2016][]) This dataset contains around 1.4 million relatively short question/answer pairs taken from amazon product reviews. This dataset is an excellent dataset for this evaluation, because:

- they texts are short in a question/answer format - similar to a single "turn" in a conversational chatbot
- they are real-world transcribed data, full of all of the wonderful noisy ambiguities and complexities of natural language

For the classification task, I have downloaded a total of 227106 question/answer pairs belonging to 10 different categories. The following is an example of one question/answer pair taken from the "baby" category

```
{'questionType': 'open-ended',
'asin': '177036417X',
'answerTime': 'Apr 16, 2015',
'unixTime': 1429167600,
'question': "Does this book contain any vaccination/immunization pages? Or pages about
school? (Most do, yet I don't vax and I homeschool). Thx so much! :)",
'answer': 'Immunization page, yes. School, no.'}
```

As this is not a question answering evaluation but a simple text classification, I split the question/answer pairs into two separate short texts, for a total of 454212 short texts.

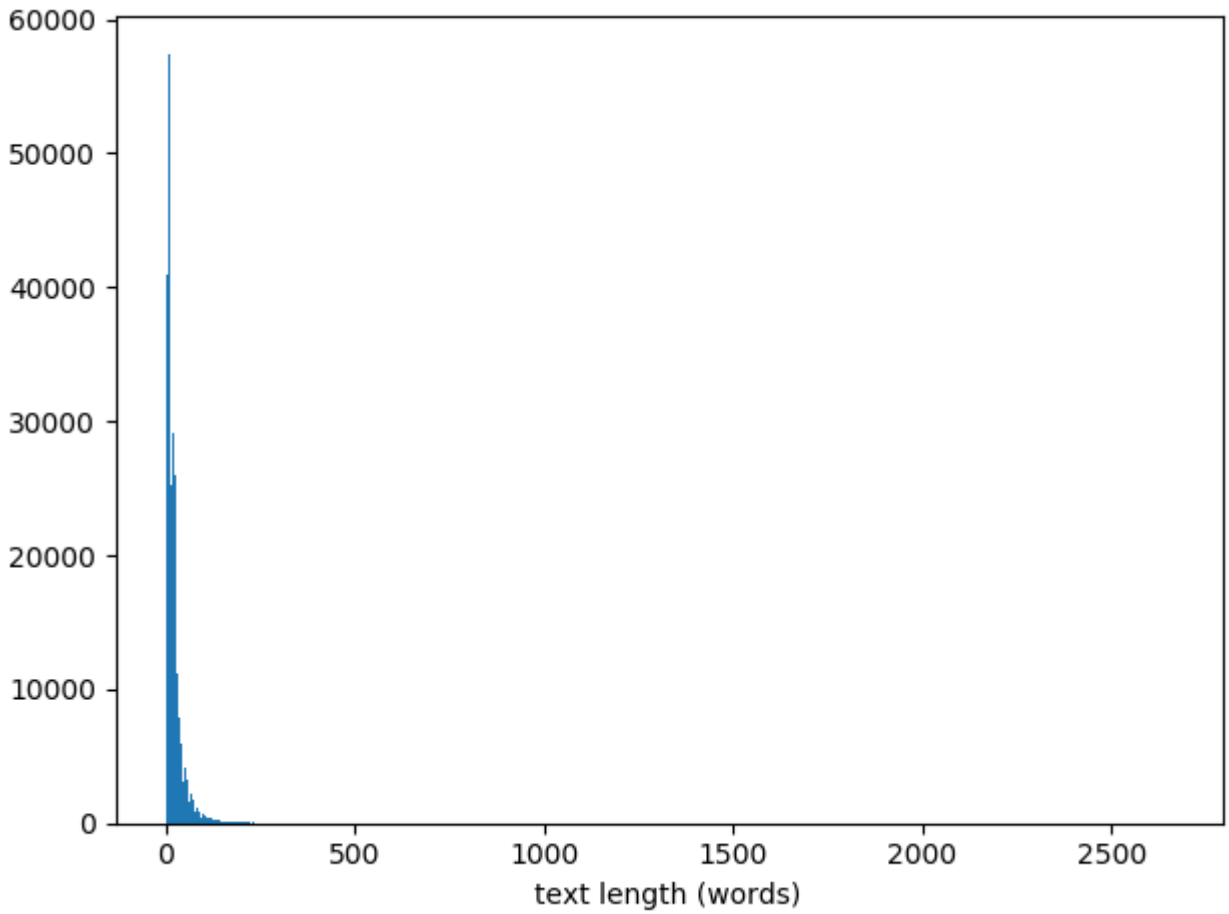
The category names, the amount of texts per category, and the relative size of the category are:

Category	Count	Percent
appliances	18022	3.97
arts_crafts_and_sewing	42524	9.36
baby	57866	12.74
beauty	84844	18.68
clothing_shoes_and_jewelry	44136	9.72
grocery_and_gourmet_food	39076	8.60
musical_instruments	46644	10.27
pet_supplies	73214	16.12
software	21272	4.68
video_games	26614	5.86

As we can see, the sizes of each category range from about ~4% to about 19% of the total dataset size. This unbalanced category distribution will be corrected in order to produce balanced classes. This step is particularly important, as the accuracy metric will be used to evaluate classifier performance.

Exploratory Visualization

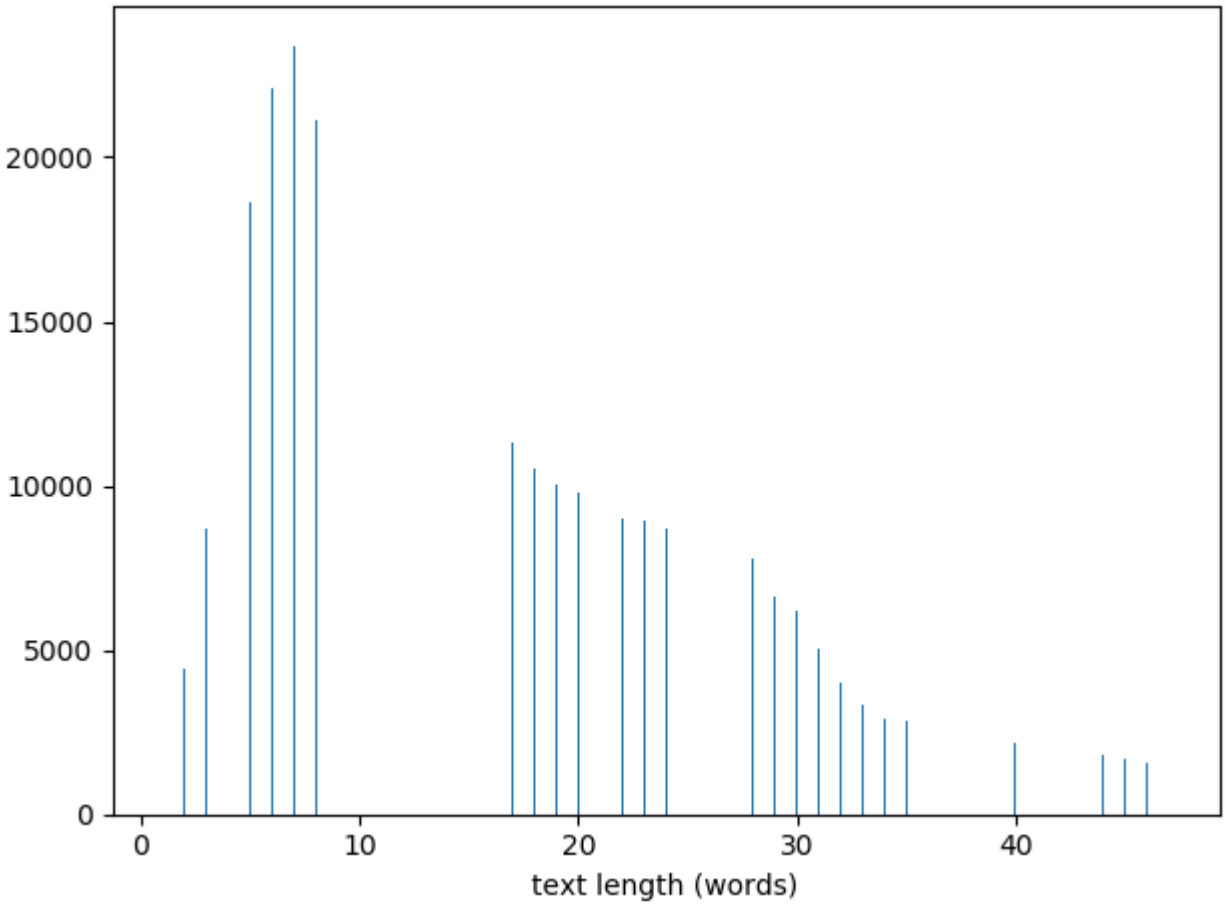
Since this evaluation is concerned with short text classification, it is interesting to see just how short the texts in the total dataset actually are. The following histogram shows the distribution of text lengths:




```
'doc_count': 454212, 'mean': 22.59106320396643,
'mode': ModeResult(mode=array([7]), count=array([23396])),
'quantile_90': 47.0, 'max': 2660, 'min': 0, 'num_empty': 11
```

As we can see by the graphic and the stats above, the vast majority of the texts are somewhere between 1 and 100 words long, with an average length of 22.6 words. However there are small numbers of much longer texts, up to 2660 words long (not visible as bars in the histogram, but apparent on the scale of the x-axes). As well, there are 11 empty texts in the dataset. Because we are interested in short text classification, we can discard the small amount of very long texts. Additionally, empty texts should be discarded.

The following shows the distribution of text lengths after discarding all texts longer than 47 words (the 90th percentile of all text lengths) and empty texts.




```
'doc_count': 409768, 'mean': 15.86339343238125,
'mode': ModeResult(mode=array([7]), count=array([23396])),
'quantile_90': 31.0, 'max': 47, 'min': 1, 'num_empty': 0
```

After discarding those texts, 90% of the dataset remains, and consists of more appropriate texts for a short text classification task. Discarding long and empty texts is one step taken in data preprocessing for this reason.

The new distribution of text categories after discarding long and empty texts is:

Category	Count	Percent
appliances	16349	3.99
arts_crafts_and_sewing	38639	9.43
baby	52163	12.73
beauty	76992	18.79
clothing_shoes_and_jewelry	40775	9.95
grocery_and_gourmet_food	35539	8.67
musical_instruments	42084	10.27
pet_supplies	64127	15.65
software	18441	4.50
video_games	24659	6.02

The distribution did not change significantly compared to the dataset before trimming, and balancing the dataset is still a requirement.

Algorithms and Techniques

Transfer Learning

Transfer Learning is a technique for improving the performance of task-specific machine learning models by incorporating existing models pretrained on very large, general datasets into those task-specific models. During so-called *fine-tuning*, task-specific data pertaining to the specific problem to be solved is used to further train an existing model, adjusting or fine-tuning the model to better fit the task at hand.

Transfer learning has been used extensively in recent years in the field of computer vision and image classification (Donahue et al. 2013). In this application, deep neural convolutional networks are trained on a large dataset, such as the popular imagenet. Through training, the layers in these networks come to represent a hierarchy of low-level (lines, edges) to higher-level (shapes, object contours, scenes) features of the images. The intuition of transfer learning is that these layers of learned feature representations can be re-used for a wide variety of novel tasks where the amount of training data might be insufficient for training a deep model from scratch.

Language Models

Transfer learning in NLP (Natural Language Processing) works by the same principle as transfer learning in Computer Vision , but applied to language tasks instead of image-related tasks. First, a *language model* is trained on a large dataset of text which thereby learns representations of the features of the language/dataset which can then be re-used for specific, novel NLP tasks. The models evaluated here (BERT, ULMFiT) are examples of such pretrained models.

A *language model* in the most exact sense refers to a model trained on the prediction task of *predicting the next word in a sequence of words*. This task is difficult because accurate prediction of the next word in a sequence must involve some representation of syntactic, semantic, and real-world knowledge as is demonstrated by the following sentences:

Have you ever really fallen in ___

Have you ever fallen down the ___

A distinguishing feature of the language models evaluated here as opposed to word embeddings such as word2vec, GloVe, or Fasttext, is that these models learn *context-sensitive representations* of words, whereas word embeddings will learn one embedding for each word in the training vocabulary. In contrast to word embeddings, these models can be used for all but the output layer in transfer learning, whereas word embeddings can only provide the first layer of a task-specific network in transfer learning.

Learning context-sensitive representations presumably entails that these models are learning and representing abstract and higher-level features of language. The exact nature of these higher-level features must remain speculative however, as they are represented in the weights of very large neural networks and as such, not transparently interpretable.

The abstractions learned by these models are useful to the extent that they can generalize well. Such models have been applied to a wide variety of language related tasks such as question-answering, entailment and sentiment analysis, grammaticality judgments and more, as evidenced by competitive performance of these models on difficult benchmarks such as GLUE.

Language modeling of this type has the additional advantage of being an *unsupervised learning task*. This is fundamentally important, as in NLP, the amount of unlabeled training data is practically unlimited (raw text), whereas labeled training data for a specific task is usually hard to come by and involves significant human effort to collect and annotate. The ability to use freely available and abundant text for pretraining is key in the usefulness of pretrained language models

ULMFiT

ULMFiT itself is less of a language model implementation as a general framework for transfer learning and fine-tuning language models. Indeed, the suggestion that word-prediction-from-context language models are the right task to train for NLP transfer learning is a proposal of the ULMFiT framework. The reference implementation uses the AWD LSTM language model under the hood which has the following properties:

- implements innovations in regularization and hyperparameter tuning, but otherwise:
- is a "Vanilla" LSTM with no custom modifications to the LSTM architecture such as attention or short-cut connections
- It is a 3-layer architecture with 1150 units in the hidden layer and an embedding size of 400

ULMFiT claims to be a general framework in that the same language model and hyperparameter tune can be used for transfer learning in a wide variety of tasks.

ULMFiT proposes a 3-Step process for training and fine-tuning

1. **LM pretraining**

   The model is pretrained on a large corpus. The reference implementation is pretrained on the Wikitext 103 dataset

2. **LM fine-tuning**

   During LM fine-tuning, the weights of all layers of the network are updated using task-specific training data. LM fine-tuning uses Discriminative fine-tuning and Slanted triangular learning rates (see below)

3. **Classification fine-tuning** (for classification) - for classification fine-tuning, an intermediate layer is added to the model with ReLU activation, and the classifier output layer is added with softmax activation outputting the probability distribution of predictions for each class in the target variable.

ULMFiT proposes 3 innovations for fine-tuning:

1. **Discriminative fine-tuning** assigns a *unique, dedicated learning rate parameter to each individual layer in the model*. The reasoning is that because each layer encodes different information, they should be fined tuned differently or *discriminatively*. In contrast to normal Gradient Decent, where all model parameters at a time-step are updated at once,

   $$\theta_t = \theta_{t-1} - \mu \nabla_{\theta_t} J(\theta)$$

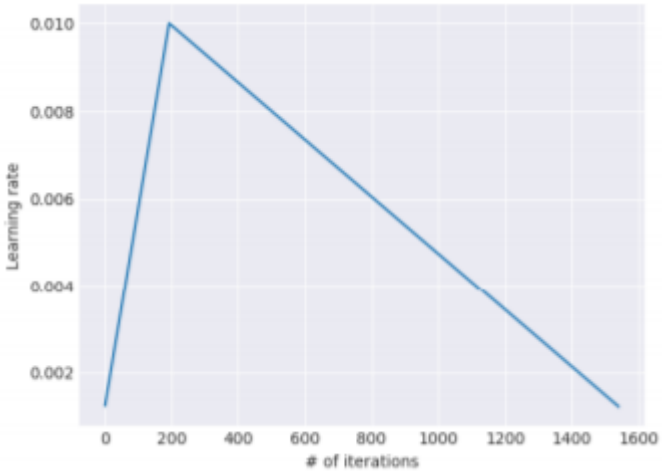
   where $\theta = parameters$, $\mu = learningrate$ and $\nabla = gradient$

   discriminative fine-tuning splits the model parameters into one set per layer, and updates them according to their own learning rate as follows:

   $$\theta_t^l = \theta_{t-1}^l - \mu^l \nabla_{\theta_t} J(\theta)$$

   where *l* refers to each layer.

2. **Slanted Triangular learning rates** - the authors want the learning at each layer to converge quickly to the approximate correct region in feature space, and then iteratively refine the parameters. To accomplish this, they propose a slanted triangular learning rate which increases rapidly, and then declines slowly.

   (taken from Howard and Ruder 2018, page 4)

3. **Gradual Unfreezing** is a technique applied at the classification fine-tuning step. Rather than "unfreeze" or unlock all layers in the model at once during fine-tuning, making them all available for updating, the authors propose unfreezing the top layer, fine-tuning for one epoch, and then unfreezing the next layer, and repeating the process until convergence. Gradual unfreezing is claimed to help the model resist "catastrophic forgetting" - that is, forgetting much of what it has learned in steps 1 and 2 and making transfer learning pointless. The rational for starting at the top layer is that this layer contains the least general information, or in other words the information containing features which are nearest to the final classification step.

These techniques are claimed to help ULMFiT generalize well to a broad range of specific tasks, while at the same time preserving as much pretrained knowledge as possible.

BERT

BERT (Bidirectional Encoder Representations from Transformers) differs from other language model implementations by incorporating **bi-directional context**. The idea is intuitive: train a language model which can predict a word from context, but use context from both directions, both before and after the word:

{She went to the} {and bought groceries}

While ULMFiT and other pretrained LMs can be trained forwards (predicting next word) and backwards (predicting previous word) independently and either concatenate forward and backward context features (Peters et al 2018) or average LM classifier results (ULMFiT), BERT is unique in that it learns bi-directional contextual representations at once.

The technique used for bi-directional learning is called the **"masked language model"** or MLM. This technique is inspired by the Cloze Test where a paragraph is presented with a proportion of words replaced by a blank, and subjects have to try to guess the missing words. In the MLM, a small percentage of words are hidden or "masked out" during training (as in the example sentence above) and the prediction task is to predict the masked tokens. The learned hidden vectors representing the masked tokens are feed to a softmax output layer to predict the output token (in the space of all tokens in the training set vocabulary). In the reference implementation, 15% of input tokens were masked out.

BERT is based on a multi-layer bidirectional transformer encoder architecture (Vaswani et al 2017) . The details of the underlying implementation can be taken from that paper, important here is to note that it is a fundamentally different architecture than ULMFiT. The Transformer contains no recurrence (as in LSTM) nor convolutions (as in CNNs), but uses rather a unique encoder-decoder with stacked attention architecture.

Benchmark

There is no established benchmark for classifiers using this dataset that I am aware of, however to provide a baseline for the performance of the language models on this data, a baseline Bag-of-Words classifier is used. This baseline tokenizes input text creating normalized word-unigrams and bigrams which are then weighted with tf-idf. The tf-idf feature vectors are used as input to a Multinomial naive Bayes classifier. This simple baseline represents a strong benchmark for short text classification, as already mentioned above and discussed in Wang and Manning 2012.

III. Methodology

Data Preprocessing

The training data was downloaded from the amazon question/answer dataset site. The downloaded files were tarballs (*.tar.gz), one tarball per category. The extracted data consists of one *.json file for each category. These files consist of question-answer data and metadata, one line for each question/answer pair.

```
{'questionType': 'open-ended',
'asin': '177036417X',
'answerTime': 'Apr 16, 2015',
'unixTime': 1429167600,
'question': "Does this book contain any vaccination/immunization pages? Or pages about
school? (Most do, yet I don't vax and I homeschool). Thx so much! :)",
'answer': 'Immunization page, yes. School, no.'}
```

The first real step of preprocessing is to extract the question and answers from these text lines, discarding the other metadata. For each question/answer pair in each category, the pair is extracted and appended to an array of texts. This array represents all texts in a category, for example "baby". Additionally, an array of category names (labels) is built.

```
{'labels': ['baby', '.....,
'texts': [['"Does this book contain any vaccination/immunization pages? Or pages about
school? (Most do, yet I don't vax and I homeschool). Thx so much! :)","Immunization
page, yes. School, no.", .....],[....]]
```

The class `AmazonQADataLoader` takes over the above tasks of tarball extraction, sentence pair extraction, and label and text array construction. The extracted text is saved in the data loader "data" attribute. This data is then used to initialize instances of *`DataProvider`* class as needed. The `DataProvider` works on the underlying total extracted dataset, and does the following:

- **normalizes text lengths** - as discussed above, long and empty texts are removed from data from which samples are later taken

```
    def _normalize_lengths(self, data, quantile=90):
        ### remove empty texts and texts of len > quantile (percentile)
        def lengths(dat):
            ## get lengths
            lengths = []
            for texts in dat:
                lengths.extend([len(text.split()) for text in texts])
            return lengths
        newdata = []

        def _check_len(text, max_incl):
            x = len(text.split())
            return x > 0 and x <= max_incl
        for texts in data:
            newtexts = [text for text in texts if _check_len(text, quantile_count)]
            newdata.append(newtexts)

        return newdata,orig_stats,new_stats
```

- **provides samples** of the data corresponding to the size of the passed-in train, eval, and test size parameters. This is important as this evaluation is carried out on samples of various sizes of the underlying dataset. As the focus of this evaluation is on fine-tuning pretrained models with small datasets, small samples were taken from the total dataset to create the train and test datasets. **The sample sizes used for the evaluation** are:

Sample name	training set size	evaluation set size	test set size
small-150	150	50	100
small-300	300	100	100
small-450	450	150	100
small-600	600	200	100
med-900	900	300	100
med-1500	1500	500	100
lrg-3000	3000	1000	100
lrg-12k	12000	4000	100
lrg-30k	30000	10000	100
full	~110k	~50k	100

- ensures that the **provided samples are balanced across the target classes**. Balanced classes are important so as to not skew the accuracy score of the classifier. Balanced classes have the effect of giving the classifiers no a-priori reason for preferring one class over the other. As there is much more data available in the dataset than needed for training/fine-tuning, creating balanced sample datasets is relatively easy.

```
    def _sample_from_data(self,data:dict,balance_classes = True):
        ...

        ## if balance classes, downsample to smallest class size
        downsampleto = min([len(x) for x in X_])

        for (i,label) in enumerate(y_):
            vals = X_[i]
            class_sample_size = downsampleto if balance_classes else len(vals)
            y.extend([label for i in range(class_sample_size)])
            X.extend(vals[:class_sample_size])

        ....

        ## train test split with stratify true if balance classes
        dostrat = y if balance_classes else None
        self.x_train, self.x_eval, self.y_train, self.y_eval = \
train_test_split(X,y,train_size=train_size,
                test_size=eval_size,stratify=dostrat)
```

- **Exposes the samples** in a format appropriate for the different classifiers in the evaluation. For example, BERT requires a list of Instances of `TrainingExample` objects, whereas the `scikit-learn` based baseline classifier requires two arrays, one "X" array of training texts, and a "y" array of the corresponding labels.

Implementation

A particular coding challenge for this evaluation was in analyzing and understanding the open source code for ULMFiT and BERT, and getting that code to work with this dataset. Because of the complexity of this code, I decided to do some object oriented analysis and design as a method of understanding the code and incorporating it into this evaluation.

The BERT code used as a starting point for the BERT implementation here is available at:

https://github.com/google-research/bert

The script `run_classifier.py` provides the boilerplate for implementing a custom classifier with BERT. This code was cloned and added as a source dependency to the project.

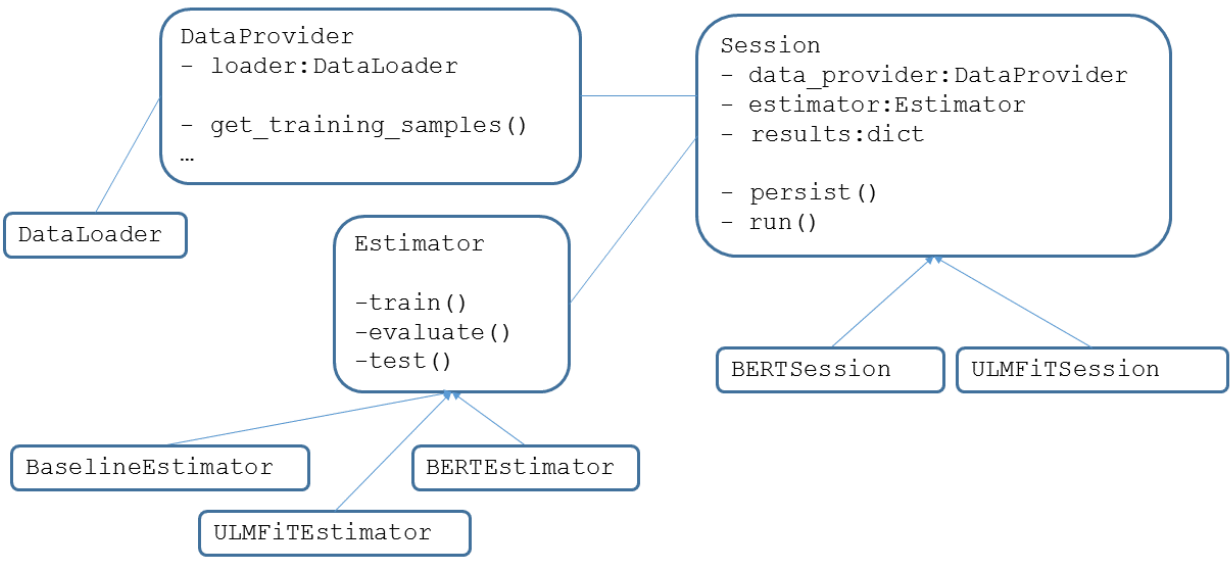
For ULMFiT, the fastai wrapper for ULMFiT was used.

https://docs.fast.ai/text.html

the `fastai` library was added as a python dependency with pip.

The goal of the analysis and design was to produce a suite of classes, interfaces, and implementations that would provide the functionality of evaluating all three classifiers (baseline, ULMFiT, BERT) on the same set of dataset samples, and record the results. I wanted to do this in a way that provided a consistent API, was manageable and readable, and had a minimum of "spaghetti code", and encapsulated the complexities of the apis for BERT and ULMFiT.

A simplified UML of the primary classes and collaborators is as follows:



As already covered above, the `DataLoader` and `DataProvider` handles loading, extraction, sampling, balancing and length normalization.

For the evaluation, one `DataProvider` was initialized for each sample size (see above).

```
## initialize data provider for dataset with 900 train, 300 eval, and 100 test samples
dp = DataProvider(data, "med-900",900,300,100)
```

The `Estimator` class encapsulates the configuration and logic for training, evaluating, and testing for each of the classifiers being evaluated. The train, evaluate, and test methods take data prepared by the `DataProvider` as parameters, and return evaluation or prediction results as needed.

```
## create an estimator for bert
bert_estimator=BertEstimator(config)
```

The `Session` class has a `DataProvider` and an `Estimator` as collaborators, and encapsulates the process of running an `Estimator` with a specific dataset sample prepared from the `DataProvider`, and then recording the results in its `evaluation_results` and `prediction_results` variables respectively. After a session is run, it can be persisted so that the data and the results can be retrieved for final evaluation.

```
bert_session = BertSession(dp, bert_estimator,name="bertsession")
bert_session.train()
bert_session.evaluate()
bert_session.predict()
print(bert_session.evaluation_results)
print(bert_session.prediction_results)
bert_session.persist()
```

The above example is for BERT, the ULMFiT and Baseline implementations follow the same pattern.

A note on text preprocessing

The text data provided by the `DataProvider` class are not yet fully preprocessed for the estimators. Much text preprocessing happens "under the hood", is implementation specific, and is initiated by the calls to train, evaluate, predict respectively.

The baseline estimator uses the default tokenization and normalization provided by `scikit-learn` `TfIdfVectorizer` with the exception that unigrams and bigrams are used (`ngram_range(1,2)`). The relevant part of that configuration is reproduced here:

```
    encoding='utf-8',
    strip_accents=None, lowercase=True,
    analyzer='word',
    stop_words=None, token_pattern=r"(?u)\b\w\w+\b"
```

The BERT estimator uses the BERT-internal `FullTokenizer`. This tokenizer converts to unicode and then to lower case, removes punctuation and normalizes whitespace, just as the baseline does. It also segments the input into "word-pieces" with the BERT `WordpieceTokenizer` This tokenizer is unique to BERT and creates tokens based on word segments. The documentation for this is provided here:

```
  def tokenize(self, text):
    """Tokenizes a piece of text into its word pieces.

    This uses a greedy longest-match-first algorithm to perform tokenization
    using the given vocabulary.

    For example:
      input = "unaffable"
      output = ["un", "##aff", "##able"]

    Args:
      text: A single token or whitespace separated tokens. This should have
        already been passed through `BasicTokenizer.

    Returns:
      A list of wordpiece tokens.
    """
```

Text preprocessing for ULMFiT takes place when creating a `TextDataBunch` object

```
## create text bunches out of label, text dataframes.
# Language model data
self.lm_databunch = TextLMDataBunch.from_df(
    train_df=df_trn, valid_df=df_val, test_df=df_test, path="")
# Classifier model data
self.clf_databunch = TextClasDataBunch.from_df(path="",
    train_df=df_trn, valid_df=df_val, test_df=df_test,
                  vocab=self.lm_databunch.train_ds.vocab, bs=32)
```

The `Tokenizer` under the hood normalizes whitespace, converts to lowercase, and does other ULMFiT-specific transformations. The details are found in the `fastai.transform` module

Estimator Hyperparameters

The baseline estimator uses an `alpha` value of 1 for `MultinomialNB`. Trying other values for `alpha` did not result in any improvement

The hyperparameters for BERT are extensive. It was prohibitive to do an extensive parameter tune due to the time and expense of running BERT on cloud TPU machines, so the defaults recommended by the BERT team were used. The relevant hyperparameters used are as shown below:

```
class BertEstimatorConfig:
```

```
    def __init__(self,
                 bert_pretrained_dir,
                 output_dir ,
                 use_tpu = True,
                 tpu_name = None
                 ):
        ...

        self.do_lower_case = (bert_pretrained_dir.find("uncased")>-1)
        "The maximum total input sequence length after WordPiece tokenization. "
        "Sequences longer than this will be truncated, and sequences shorter "
        "than this will be padded."
        self.max_seq_length = 128
        self.train_batch_size = 32
        self.eval_batch_size = 8
        self.predict_batch_size = 8
        self.learning_rate = 5e-5
        self.num_train_epochs = 3.0
        "Proportion of training to perform linear learning rate warmup for. "
        "E.g., 0.1 = 10% of training."
        self.warmup_proportion = 0.1
        "How often to save the model checkpoint."
        self.save_checkpoints_steps = 1000
        "How many steps to make in each estimator call."
        self.iterations_per_loop = 1000
        ....
```

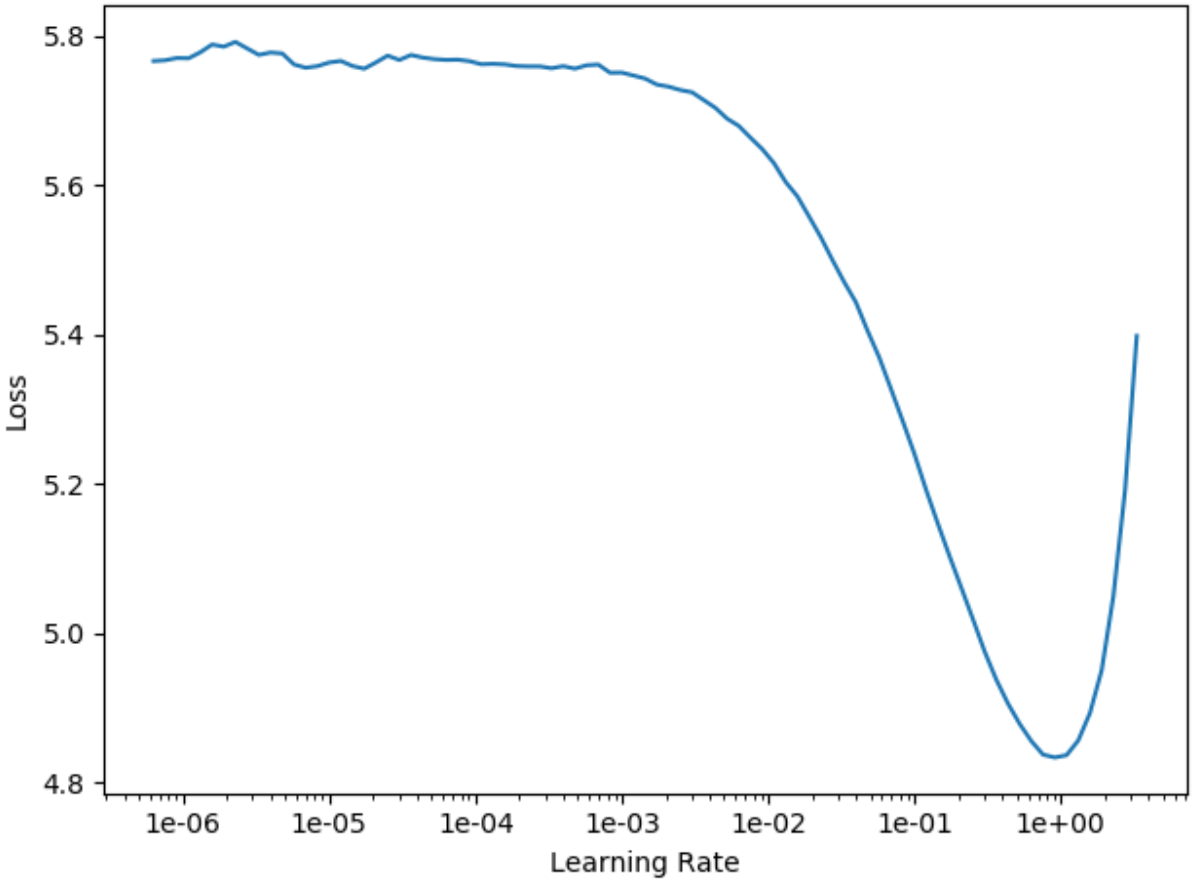
The BERT Base model was used for this evaluation

```
BERT_BASE_MODEL = "gs://cloud-tpu-checkpoints/bert/uncased_L-12_H-768_A-12"
```

The BERT Base model utilizes a Transformer architecture with 12-layers of size 768 each, 12 attention heads, and 110 million trainable parameters. It is pretrained on Wikipedia text.

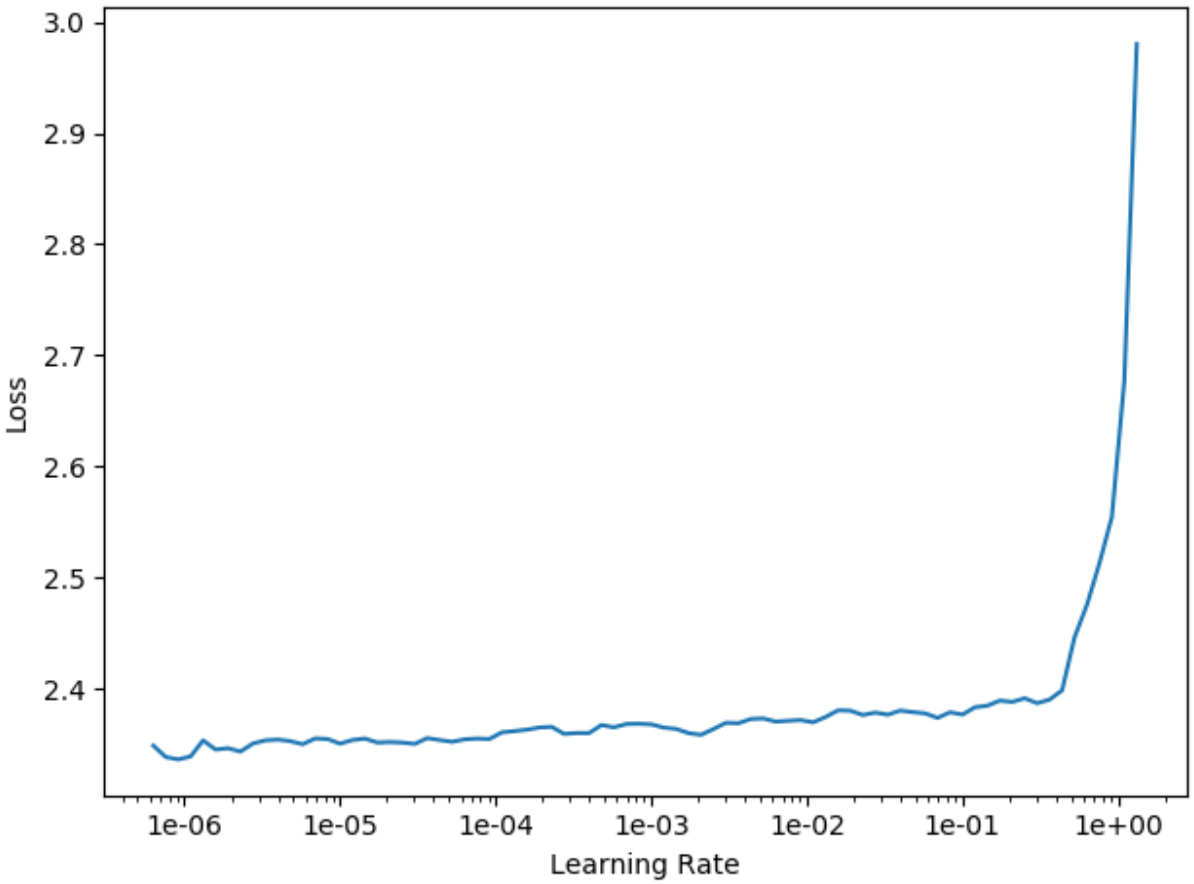
For ULMFiT, the primary hyperparameters to choose were

- dropout factor - what proportion of the preconfigured dropout rate should be applied. Values of 0.5, 0.7, and 1 were tried. A value of 0.5 provided marginally better results than the others.
- learning rate - choosing an appropriate learning rate for the language model and classifier models was accomplished by means of the `fastai lr_find` function, which mock-trains a model and then provides a scatterplot of the learning rates and losses. The scatterplot for the language model learner is as follows:



The ULMFiT team recommends choosing a learning rate around the middle of the downward curve, but not the lowest value. For this reason a learning rate of $1e-01$ was selected.

The scatterplot for the classifier learner is somewhat less interpretable:



Here, any learning rate before the loss starts to increase seems a plausible choice. A learning rate of $1e-03$ was chosen.

Refinement

BERT

Initially, the evaluation was run with the BERT large model.

```
BERT_LARGE_MODEL = "gs://cloud-tpu-checkpoints/bert/uncased_L-24_H-1024_A-16"
```

This model has a deeper architecture with 340 million trainable parameters as opposed to the base model's 110 million.

It was found that running the large model often resulted in inconsistent evaluation and training results. Specifically, often the predicted class returned during evaluation would be the same for all examples, resulting in an accuracy score of 10%, or exactly the score of a naive classifier. After extensive debugging, I was convinced this error was not in my code, but coming from BERT somewhere. Switching to the base model fixed the problem, and consistent results could be acquired.

ULMFiT

A challenge for ULMFiT was in implementing *discriminative fine tuning* and *gradual unfreezing* as described above and in the original ULMFiT paper. Introductory tutorials and quick start guides for ULMFiT do not provide any indication if this functionality is implemented under the hood, or if it must be implemented extra in client code. It turns out, one must dig a little deeper in the documentation and api to find the

answers.

discriminative fine-tuning

Discriminative fine-tuning is implemented by providing a list of learning rates to the fit method of the learner. The list needs to be of the same length as the number of layers in the model. To simplify this, the method `lr_range` is provided by `fastai`, which provides such a list based on start and end values:

```
lrr = self.lm_learn.lr_range(slice(1e-1, 1e-3))
```

Gradual unfreezing

Gradual unfreezing must be implemented by client code. There is no method or flag in the wrapper code which automatically activates this feature. This is a bit of a surprise, as this feature is one of the central innovations of ULMFiT. Be that as it may, the ULMFiT paper recommends starting with the top layer, training for one epoch, and then incrementally working ones way down before starting over again for as many epochs as one wishes to train. This functionality is partially realized with the `freeze` and `freeze_to` methods. The implementation in my code for the `train()` method of the language model learner is reproduced here:

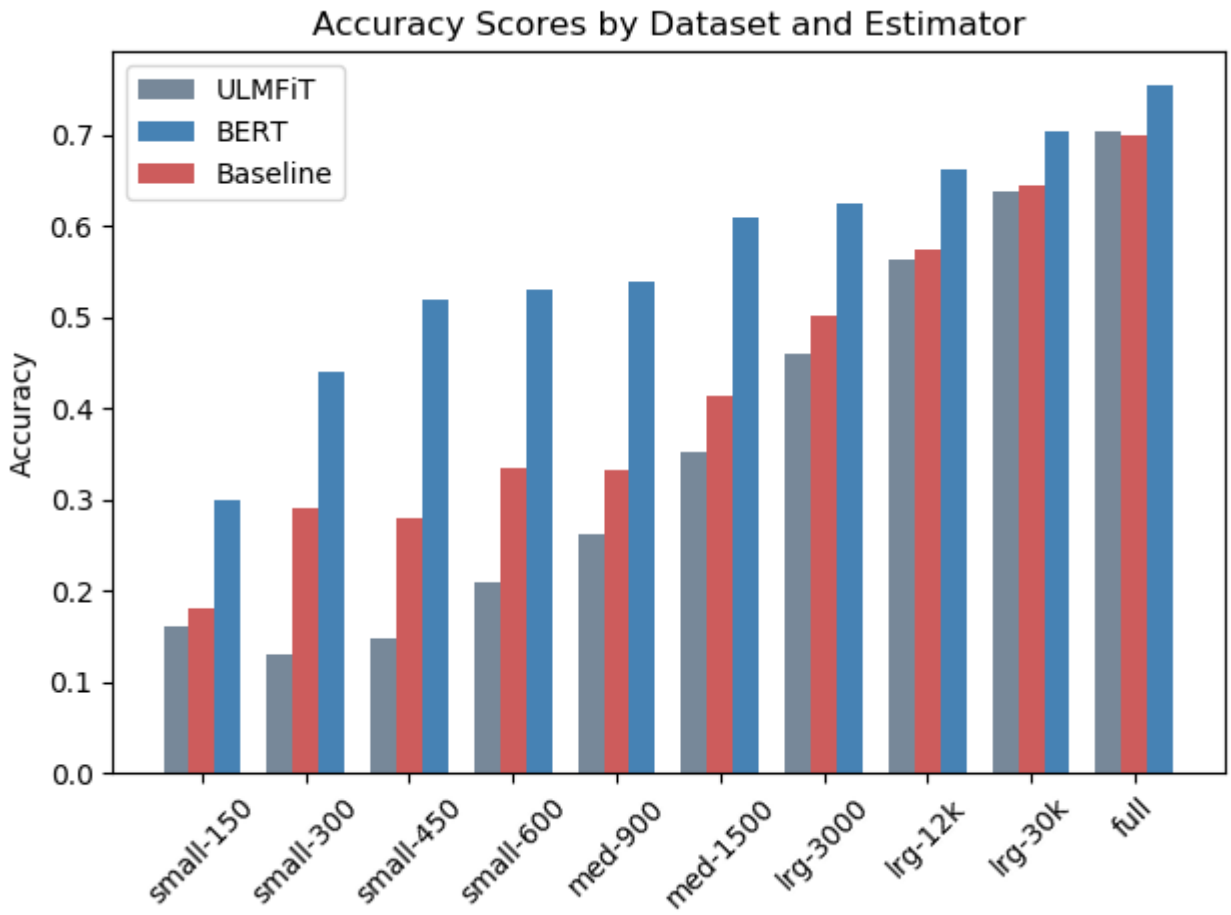
```
## learn using discriminative training and gradual unfreezing
numlayers = len(self.lm_learn.layer_groups)
print("lm learner has {} layers ",numlayers)
stop = 0-(numlayers)
## 1e-1 previously found as acceptable learning rate with learn.lr_find
## lr_range creates learning rates for each layer (discriminative training)
lrr = self.lm_learn.lr_range(slice(1e-1, 1e-3))
for i in range(self.epoch1):
    print("Training for Epoch ",i+1)
    self.lm_learn.freeze()
    ## now start layer unfreezing
    ##only last layer trainable
    self.lm_learn.fit_one_cycle(1,lrr)
    layer_num_inverted = -2 ## not second layer, but second from last
    ## train next layers
    while layer_num_inverted > -numlayers:
        print("freeze to ",layer_num_inverted)
        self.lm_learn.freeze_to(layer_num_inverted)
        self.lm_learn.fit_one_cycle(1, lrr)
        layer_num_inverted = layer_num_inverted-1
```

Implementing discriminative fine-tuning and (especially) gradual unfreezing improved ULMFiT performance by a few percent points (accuracy) compared to an initial naive implementation (following quick-start guides).

IV. Results

Model Evaluation

Without any further ado, I present preliminary results for each classifier, for each sample in the total dataset.



	Baseline	ULMFiT	BERT
small-150	0.18000	0.16000	0.30000
small-300	0.29000	0.13000	0.44000
small-450	0.28000	0.14670	0.52000
small-600	0.33500	0.21000	0.53000
med-900	0.33333	0.26300	0.54000
med-1500	0.41400	0.35200	0.61000
lrg-3000	0.50200	0.46000	0.62500
lrg-12k	0.57500	0.56400	0.66200
lrg-30k	0.64430	0.63800	0.70500
full	0.70003	0.70300	0.75361

These results were obtained using one dataset, containing one sample for each sample size, and one pass through each estimator for each sample size.

The accuracy results reflect the score on the "evaluation" held-out data for each sample, which was consistently 1/3 of the training set size.

These results are preliminary, because they represent one pass through each classifier using the same total dataset, and do not take sensitivity to other datasets into account.

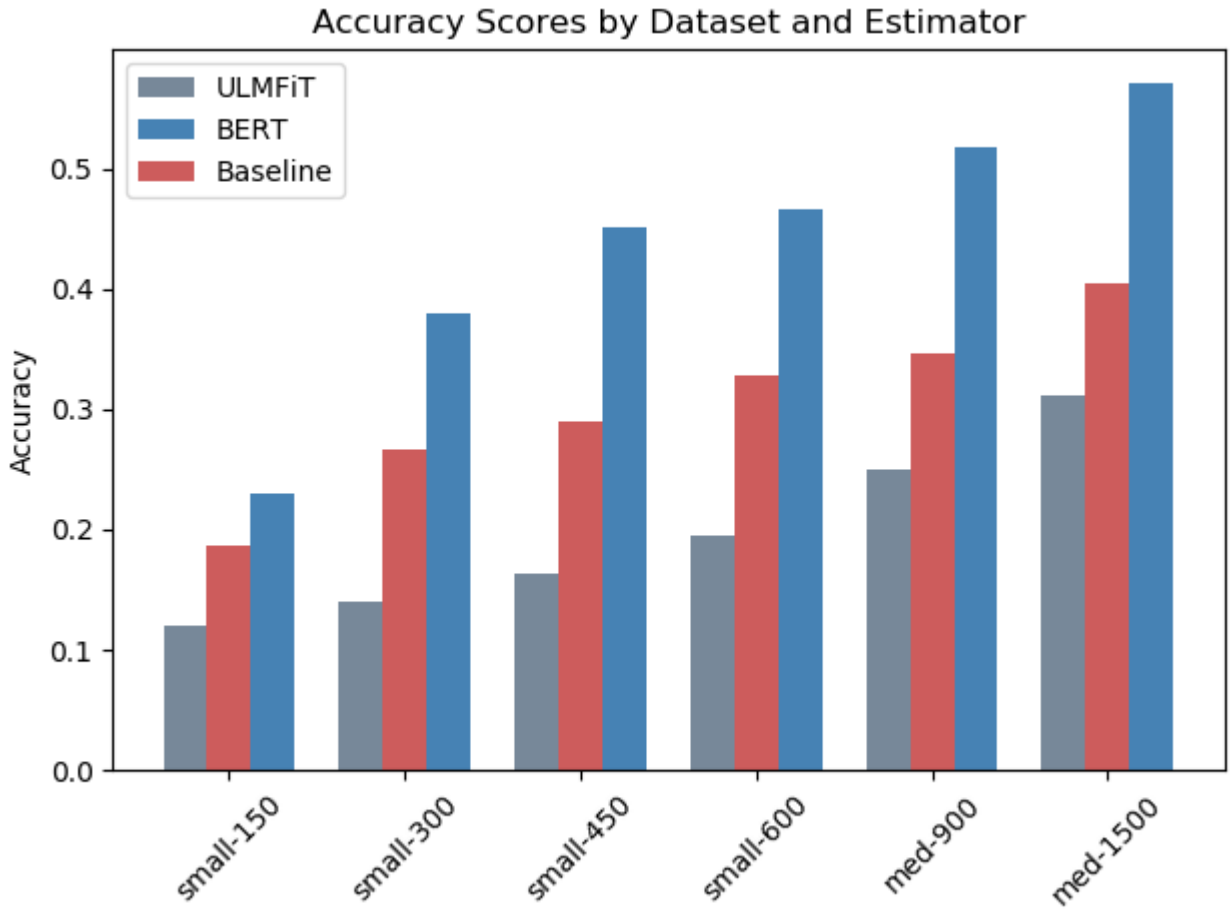
However, based on these results, we can make some preliminary conclusions:

- ULMFiT performed poorly, barely better than a naive classifier on small data. This is surprising and disappointing, particularly because the ULMFiT authors report that ULMFiT performs well on small data on an IMDB sentiment analysis dataset. It does not appear as if ULMFiT was able to learn useful abstractions which would enable it to generalize well to new examples.
- BERT is the strongest classifier in this evaluation, but less so than the others with larger data, where the performance of the 3 is comparable
- The best accuracy score in this evaluation was 75%. This was obtained by the BERT classifier on the full dataset with ~110k training examples.
- BERT is particularly strong on small and medium data (150-1500 training examples), precisely the samples of interest in this evaluation. In this range, BERT consistently outperforms the baseline estimator.
- BERT reaches an accuracy of 50% with only 450 training examples. The baseline needs 3000 training examples to reach 50%, and ULMFiT still more. BERT reaches 50 and 60% accuracy with ~10x less training data than is required for the other classifiers.

Results of small data samples over 5 different datasets

However, it is still unclear how robust these results are, and if similar results are reproducible if different samples from the dataset are used. In order to check how sensitive the classifiers are to different data, 4 more datasets were produced. The datasets contain samples of the sizes `small-150` to `med-1500.` This is to focus our attention on the small datasets which are the focus of the evaluation.

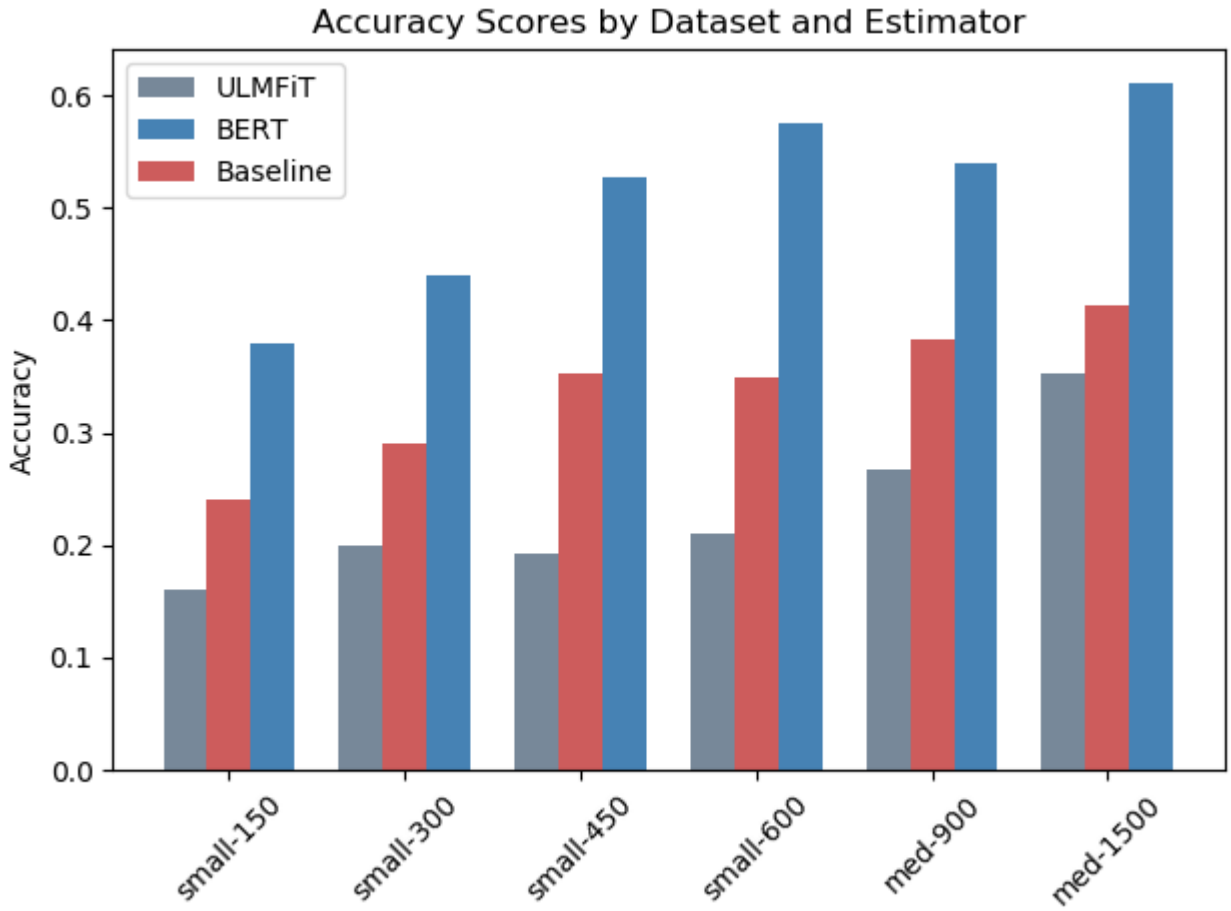
After running an evaluation on a total of 5 datasets of small samples, an **average (mean) evaluation score** was taken for each. The results are as follows:



As we can see, the contour is similar, but more consistently linear than the preliminary results.

	BERT	Baseline	ULMFiT
small-150	0.23000	0.18667	0.12000
small-300	0.38000	0.26667	0.14000
small-450	0.45111	0.28889	0.16267
small-600	0.46583	0.32833	0.19500
med-900	0.51833	0.34667	0.24933
med-1500	0.57000	0.40400	0.31160

The **maximum (best) score** achieved by each over samples from the 5 datasets were:



	BERT	Baseline	ULMFiT
small-150	0.38000	0.24000	0.16000
small-300	0.44000	0.29000	0.20000
small-450	0.52667	0.35333	0.19333
small-600	0.57500	0.35000	0.21000
med-900	0.54000	0.38333	0.26667
med-1500	0.61000	0.41400	0.35200

Here, we see that BERT has some pretty high highs, but is much more sensitive to the particular dataset used than are the others. Of particular note is the **score of 38% accuracy for 150 samples**, which is pretty spectacular, even if not consistently reproduceable.

Zooming in

As can be seen by the preliminary results, the best classifier accuracy achieved was ~75% on the full dataset. This indicates that about 25% of the examples of the dataset might have insufficient information content or be too ambiguous to be clearly classified at all. In order to examine some of the challenges and ambiguities of this data, it is useful to examine some of the examples the strongest classifier failed to classify correctly. On a test dataset of 100 examples, the BERT-full classifier missed only 11 examples (89% accuracy on the test dataset!). Those examples are shown below

true	pred	text
baby	grocery_and_gourmet_food	Hi I would like to enquire on the dimension of the packaging as i am calculating freight charges as I am using 3rd party forwarder to export to spore.
appliances	pet_supplies	how do you open the top when you have no power?
musical_instruments	baby	How do you get it to stop swinging from side to side (as in the hanger is able to move easily from the base)?
grocery_and_gourmet_food	arts_crafts_and_sewing	Is the center in a clear liquid?
baby	clothing_shoes_and_jewelry	Would a small or medium work better if I'm in between sizes? I'm 5' 4" and wear size 8-10 depending on clothes so size chart is not helping...
video_games	software	I recently installed them and they open independently.
musical_instruments	baby	what is the weight limit?
appliances	grocery_and_gourmet_food	The blue, not the purple are the real ones, right? Also, the labels of the purple misspelled "Disposable" as "Disopsable"
appliances	pet_supplies	Its a 110 v or 220/240v?
clothing_shoes_and_jewelry	baby	I have a four year old child that sits on the back of my tandem with a "kiddy crank" and needs some padded shorts. Are your shorts padded and will th
pet_supplies	beauty	I have very fine, curly hair and want to try this shampoo because of the positive reviews. Will this weigh down my hair?

Text such as "what is the weight limit" or "is the center a clear liquid" are pretty ambiguous. Additionally, some predictions that BERT made seem more plausible than the true label (see the last two texts for example).

Classifying using Out-of-Vocabulary words

It seems as if BERT is learning to generalize by using abstractions learned through a modest amount of training data, abstractions that are more powerful than simply the presence or absence of lexical word forms. In order to determine if this is indeed the case, an closer examination of some of the specific predictions BERT made is needed.

On a test set of 100 examples, the BERT classifier trained on the `small-450` sample had a 52% accuracy. Some of the correctly guessed examples are:

pet_supplies	pet_supplies	Is it large enough for Guinea pig?
video_games	video_games	DOES PS3 GRAN TURISMO 5 PLAY WELL ON A PS4?
pet_supplies	pet_supplies	if you can attatch a brass name plate, attatched with rivets, I'll purchase the collar and nameplate. Brass Name Tag For Dog Collar With 3 Set Rivet P
appliances	appliances	Where do you get the griddle and can you swap out for their other knobs? Love it so far.
baby	baby	Can an adult open the door from the outside? Can it be used to keep kids outside?
beauty	beauty	Is this product for African American hair only
video_games	video_games	yes it will work without it
video_games	video_games	Is this the slim model? How do I tell if my xbox, which came in a box just like this, is slim or not? Thanks!
arts_crafts_and_sewing	arts_crafts_and_sewing	The case doesn't create a seal and therefore the pads dry out very quickly. Also the composition of the inks might have something to do with it. Poor Product
grocery_and_gourmet_food	grocery_and_gourmet_food	what is the benefit of using this product
video_games	video_games	does it come with power cord to charge controller
arts_crafts_and_sewing	arts_crafts_and_sewing	yeh that,s the point of it I believe. I have made gobs of ruffles wih it.
musical_instruments	musical_instruments	A high E string
clothing_shoes_and_jewelry	clothing_shoes_and_jewelry	It is about 12 1/4 inches wide and 12 inches long.
grocery_and_gourmet_food	grocery_and_gourmet_food	Why does it say 100% beef then list all these ingredients? Water, Sugar, Less Than 2% Salt, Corn Syrup Solids, Dried Soy Sauce (Soybeans, Salt, Wheat)
pet_supplies	pet_supplies	I have a malti-poo that measures chest - 17 ", length - 14" Neck - 13/14" - weighs 10.5 lbs. Will the small fit him.
baby	baby	Does this pillow work well on a twin sized bed? The dimensions are the exact size of my mattress and I'm wondering if it's going to be too big.
beauty	beauty	can I use hair spray after applying rogaine
baby	baby	Cleaning this chair: Does the seat have a washable removable cover?
pet_supplies	pet_supplies	We have the Giant Kennel (65+ lb dog). The Giant Kennel weighs approximately 40 lbs and takes up a bunch of space but if you need a kennel for a larger dog, this works really well and h
appliances	appliances	Pull knob out, turn upside down, put back on. It's worked great, no problems
baby	baby	Hi I bought this gate some time ago. But don't remember if it's safe to have up at the stairs for toddlers or small children
musical_instruments	musical_instruments	Does this mic come with a cord?
arts_crafts_and_sewing	arts_crafts_and_sewing	Does this machine sew leather and multiple layers of fabrics without jamming??
arts_crafts_and_sewing	arts_crafts_and_sewing	Would this burn a bamboo cutting board
musical_instruments	musical_instruments	My toddler grandson bangs his head to go to sleep. It was suggested to try a metronome. Would it be loud enough?

One of the questions posed by this evaluation is, ***can pretrained language models infer using out-of-vocabulary words (with respect to the fine-tuning training set)***. In order to provide some insight to this question, it is useful to examine correctly guessed instances such as those above, and examine if presumably high-information words such as "Guinea" and "pig" (for the class pet_supplies) were present in the training data. If not, it is reasonable to conclude that the correct classification of a text was due to transfer learning and not the presence of a lexical word in the training data.

- ***"is it large enough for Guinea pig"*** Correctly classified as "pet_supplies" with 75% confidence.
- Neither "Guinea" nor "pig" was present in the training data for the dataset `small-450`
- ***"do any of the under armours come with a fly you can open"*** Correctly classified as "clothing_shoes_and_jewelry" (48% confidence, followed by "arts_crafts_sewing" with 26% confidence)
    - neither "armours" nor "fly" were present in the training data. How could BERT have inferred the category "clothing" from a sentence "do any of the under xxx come with a xxx you can open"? ***Here it seems an intriguing possibility that BERT has learned a feature for "opening a fly" (as opposed to swatting a fly), and applied this to the input text*, although "fly" was not present in the training data.** This might be an instance of transfer learning ("fly" not in training data) and word sense disambiguation through context ("fly you can open" is different than "fly you can

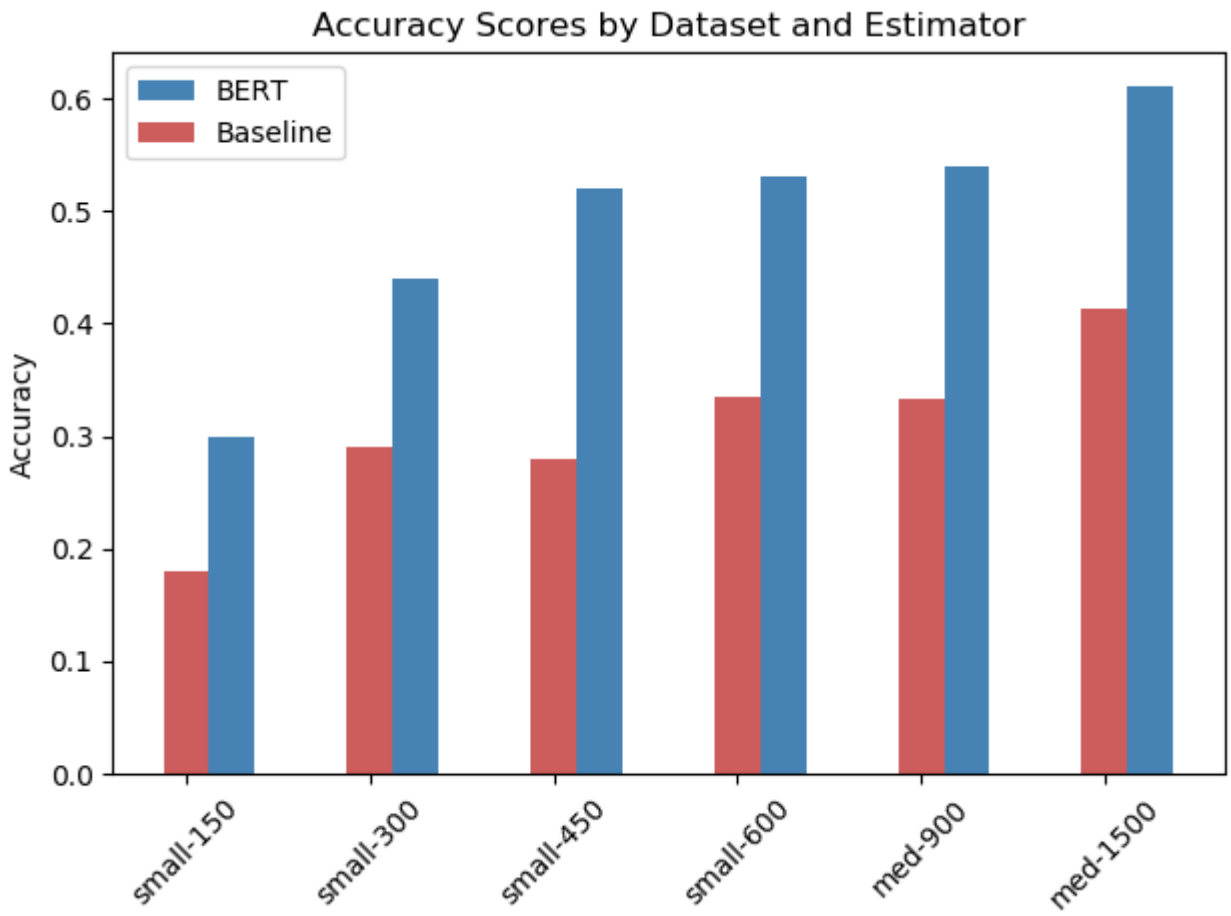
swat"). Furthermore, the other words which might be a hint for "clothing" - "under" and "open" are not at all exclusive to the clothing category. Below is a listing for every category every time the words "under" and "open" appear in the training data. They appear not that often, and in several categories.

```
under   baby
open    baby
open    arts_crafts_and_sewing
under   musical_instruments
under   clothing_shoes_and_jewelry
under   musical_instruments
open    pet_supplies
under   musical_instruments
open    software
under   clothing_shoes_and_jewelry
under   baby
```

- **"yes that is the point of it I believe, I have made gobs of ruffles with it"** Correctly classified as "arts_crafts_and_sewing" (with only 26%, almost evenly split with "clothing" at 25% confidence).
    - neither "gobs" nor "ruffles" were present in the training data.

It seems as if BERT is indeed successfully leveraging transfer learning to achieve the results it does on small datasets.

V. Conclusion



The histogram above of the preliminary results reduced to BERT and Baseline helps focus our attention on the primary positive finding from this evaluation which is that ***BERT consistently outperforms the baseline significantly on small datasets***.

The conclusion to be drawn must be that that ***yes, pretrained language models have the potential to help us bootstrap NLP applications such as chatbots by leveraging transfer learning when only small training datasets are available*** , however this statement has two very large caveats:

- Pretraining on a large corpus and fine-tuning is in and of itself no guarantee of improved performance. The many, details of implementation, pretraining and fine-tuning can make the difference between success and failure. For some reason, ULMFiT performed very badly although the principles of pretraining and fine-tuning are the same.
- Transfer learning using deep neural models is a technique that is many orders of magnitude more complex than "traditional" techniques like the baseline used here in terms of

- **time** - pretraining these models can take days and even weeks, whereas training the baseline on the 110k full dataset took less than two minutes. Of course pretraining is not something that is done often, maybe never by users of the models, however even fine-tuning the BERT and ULMFiT models for each of the sample datasets takes several hours (on big machines, see below), whereas the baseline classifier is done in around two minutes on my commodity business laptop.
- **computational power** - the several hours mentioned above for fine-tuning BERT (ULMFiT took also several hours) is the time it took for training on *expensive cloud-hosted machines fitted with multiple tpu accelerators or gpus*. The baseline classifier with Naive Bayes requires no special computing power.
- **code/api complexity**- using a Naive Bayes from scikit-learn to achieve the baseline results above involves a handful of lines of code. The scikit learn code is also well-documented easy to use and understand, and works as expected with an absolute minimum of "going under the hood". Understanding and using the code for BERT and especially ULMFiT was a real challenge. I have around 20 years experience as a coder, but found finding the correct documentation and actually working with the code to be an at times frustrating experience of trail-and-error and unpleasant surprises. With ULMFiT, because the results were as poor as they were, I have the nagging feeling I may have used the API wrong or forgotten some key detail, however nothing was apparent even after long and repeated research and trail and error.

These practical considerations, while not the primary focus of this evaluation, weigh heavily in the chose of technology in a practical application. At the moment, although BERT did deliver some surprising and interesting results, it would be impractical and prohibitive in terms of time, cost, and effort to actually incorporate BERT into a real-world chatbot project.

Reflection

The design of this evaluation was to simulate a real-life scenario of trying to bootstrap a dialog-system when large amounts of high-quality training data are not available. This scenario is not only theoretical or purely academic, but is exactly the situation I find myself in in my work, where we are entering the field of designing dialog systems (chatbots) from scratch. Our clients are not generally able to provide us with training data (indeed, most do not know anything about NLP and supervised learning), so we are left to thinking up or collecting examples on our own.

Often when designing and training these dialog systems, and using classifiers not too terribly different from the baseline classifier used here, you get the sense that - in the beginning at least - you must really hit the system over the head with training data before it demonstrates even stable behavior. The "wow" effect of providing a few examples and having the system draw "intelligent" conclusions or generalizations from those examples is simply not yet there. This missing "wow" effect can really cause some disillusionment amongst those new to the field who might be expecting a bit more "intelligence", especially given the recent AI-hype.

My hopes were high going into this evaluation, as transfer learning with pretrained language models seemed exactly the type of technique that could provide exactly the "wow" effect I had been missing.

Were my hopes met? Well, partially. BERT does demonstrate some behavior that indicates it is using transfer learning to make intelligent generalizations beyond what is seen in the fine-tuning training data, but at what cost? At the moment, it is a real question if the computational requirements of running a BERT model are too high for using in real-life projects where GPU-enhanced hardware is still extremely rare, and a dialog system might be required to handle many parallel dialogs at once.

Also, although the BERT results are indeed interesting, it is an open question - also for myself - if the results represent a real breakthrough in NLP or simply a significant incremental improvement over more commonplace solutions like the baseline classifier here.

Of course language models like BERT and ULMFiT can do more than simply classify text. Sequence prediction, grammaticality judgments, question-answering are all very interesting applications for these models, applications that cannot be approached with commonplace BOW-based classifiers. But although text classification is not necessarily the most sophisticated task for a modern NLP system, we saw that the task here presented some real challenges for the models evaluated. The texts are short and they are "dirty", taken from real-life exchanges.

The fact that the classifiers here were not able to get much better than 70% accuracy even on ~110k training examples highlights the challenge of classifying this kind of ambiguous and "dirty" real-life text accurately. And of course the real challenge is in training a model that can generalize well from small training data, and we saw mixed results in this evaluation.

In the end, I am left wondering if the improvements shown by BERT are really a result of a system that is efficiently learning relevant abstractions of language, and re-using these abstractions "intelligently" , or if they are rather a result of brute-force, considering the size and complexity of the BERT models.

Improvement

The most salient negative result of this evaluation was the poor performance of the ULMFiT model. This is surprising, as one of the claims of the authors is that ULMFiT requires much less training data than other models to achieve state-of-the-art results. As we have seen, I could not reproduce this effect at all, and ULMFiT performed even poorer than Naive-Bayes with TF/IDF. There seem to be two possibilities for this result:

- That I did not find an optimal hyperparameter tune and training strategy. Although I spent some time with trail-and-error and followed the documentation I could find, It is possible that the poor results are a result of my implementation, and not ULMFiT. The only way to find out it seems would be to ***present this evaluation to the ULMFiT team, and request a code-review and discussion of the results.***
- That ULMFiT simply does not perform as well on the dataset here as it does on datasets and benchmarks reported by the authors. It could be that the evaluation here - multi-class classification of short texts on very small datasets simply exposes weaknesses in the ULMFiT model that were not exposed by previous benchmarks.

References

Giorgino 2004 : https://pdfs.semanticscholar.org/99e1/3b7ac59b3b82956b26fb5fb964b2c69f4338.pdf "An Introduction to Text Classification"

Wang and Manning 2012 https://www.aclweb.org/anthology/P12-2018 " "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification""

ULMFiT: https://arxiv.org/abs/1801.06146

Elmo: https://allennlp.org/elmo

Peters et al. 2018: https://arxiv.org/abs/1802.05365 "Deep contextualized word representations"

BERT: https://github.com/google-research/bert

Howard and Ruder 2018: https://arxiv.org/pdf/1801.06146.pdf "Universal Language Model Fine-tuning for Text Classification"

Devlin et al. 2018: https://arxiv.org/pdf/1810.04805.pdf "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"

NLPs-Imagenet-Moment: https://thegradient.pub/nlp-imagenet/ "NLPs ImageNet Moment has Arrived"
amazon question/answer Dataset: http://jmcauley.ucsd.edu/data/amazon/qa/

Wan and McAuley 2016: http://cseweb.ucsd.edu/~jmcauley/pdfs/icdm16c.pdf "Modeling ambiguity, subjectivity, and diverging viewpoints in opinion question answering systems"

McAuley and Yang 2016: http://cseweb.ucsd.edu/~jmcauley/pdfs/www16b.pdf "Addressing complex and subjective product-related queries with customer reviews"

Donahue et. al. 2013: https://arxiv.org/abs/1310.1531 "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition"

Mikolov et. al. 2013: https://arxiv.org/abs/1301.3781 " Efficient Estimation of Word Representations in Vector Space"

Pennington et. al: https://nlp.stanford.edu/pubs/glove.pdf "GloVe: Global Vectors for Word Representation"

Joulin et al. 2016: https://arxiv.org/pdf/1607.01759.pdf "Bag of Tricks for Efficient Text Classification"

GLUE: https://gluebenchmark.com/tasks

Merity et al. 2017: https://arxiv.org/abs/1708.02182 "Regularizing and Optimizing LSTM Language Models "

Merity et al. 2017b: https://www.salesforce.com/products/einstein/ai-research/the-wikitext-dependency-language-modeling-dataset/

Vaswani et al.: https://arxiv.org/abs/1706.03762 "Attention is all you need"

fastai ulmfit wrapper: https://docs.fast.ai/text.html "fastai documentation for ulmfit"