Text Classification with Few Examples using Controlled Generalization

Abhijit Mahabal
Google, Pinterest
amahabal@gmail.com

Jason Baldridge
Google
jridge@google.com

Burcu Karagol Ayan
Google
burcuka@google.com

Vincent Perot
Google
vperot@google.com

Dan Roth
UPenn
danroth@seas.upenn.edu

Abstract

Training data for text classification is often limited in practice, especially for applications with many output classes or involving many related classification problems. This means classifiers must generalize from limited evidence, but the manner and extent of generalization is task dependent. Current practice primarily relies on pre-trained word embeddings to map words unseen in training to similar seen ones. Unfortunately, this squishes many components of meaning into highly restricted capacity. Our alternative begins with sparse pre-trained representations derived from unlabeled parsed corpora; based on the available training data, we select features that offers the relevant generalizations. This produces task-specific semantic vectors; here, we show that a feed-forward network over these vectors is especially effective in low-data scenarios, compared to existing state-of-the-art methods. By further pairing this network with a convolutional neural network, we keep this edge in low data scenarios and remain competitive when using full training sets.

1 Introduction

Modern neural networks are highly effective for text classification, with convolutional neural networks (CNNs) as the de facto standard for classifiers that represent both hierarchical and ordering information implicitly in a deep network (Kim, 2014). Deep models pre-trained on language model objectives and fine-tuned to available training data have recently smashed benchmark scores on a wide range of text classification problems (Peters et al., 2018; Howard and Ruder, 2018; Devlin et al., 2018).

Despite the strong performance of these approaches for large text classification datasets, challenges still arise with small datasets with few, possibly imbalanced, training examples per class. Labels can be obtained cheaply from crowd workers for some languages, but there are a nearly unlimited number of bespoke, challenging text classification problems that crop up in practical settings (Yu et al., 2018). Obtaining representative labeled examples for classification problems with many labels, like taxonomies, is especially challenging.

Text classification is a broad but useful term and covers classification based on topic, on sentiment, and even social status. As Systemic Functional Linguists such as Halliday (1985) point out, language carries many kinds of meanings. For example, words such as *ambrosial* and *delish* inform us not just of the domain of the text (*food*) and sentiment, but perhaps also of the age of the speaker. Text classification problems differ on the dimensions they distinguish along and thus in the words that help in identifying the class.

As Sachan et al. (2018) show, classifiers mostly focus on sub-lexicons; they memorize patterns instead of extending more general knowledge about language to a particular task. When there is low lexical overlap between training and test data, accuracy drops as much as 23.7%. When training data is limited, most meaning-carrying terms are never seen in training, and the sub-lexicons correspondingly poorer. Classifiers must generalize from available training data, possibly exploiting external knowledge, including representations derived from raw texts. For small training sizes, this requires moving beyond sub-lexicons.

Existing strategies for low data scenarios include treating labels as informative (Song and Roth, 2014; Chang et al., 2008) and using label-specific lexicons (Eisenstein, 2017), but neither is competitive when labeled data is plentiful. Instead, we seek classifiers that adapt to both low and high data scenarios.

People exploit parallelism among examples for generalization (Hofstadter, 2001; Hofstadter and

Proceedings of NAACL-HLT 2019, pages 3158–3167
Minneapolis, Minnesota, June 2 - June 7, 2019.

1.1	Kampuchea says **rice** crop in 1986 increased . . .	2.1	**Gamma ray** Bursters. What are they?
1.2	U.S. **sugar** policy may self-destruct . . .	2.2	Life on **Mars**
1.3	EC denies **maize** exports reserved for the U.S.. . .	2.3	Single launch **space stations**
1.4	U.S. **corn**, **sorghum** payments 50-50 cash/certs. . .	2.4	**Astronauts**—what does weighlessness feel like?
1.5	Canada **corn** decision unjustified. . .	2.5	**Satellite** around **Pluto** mission?

Table 1: **Left**: examples from the Reuters *Grains* class, showing semantic type cohesion (kinds of crops). **Right**: post headers from the *sci.space* newsgroup in 20 Newsgroups, showing topical cohesion (astronomical terms). **Bolded terms** are to draw the reader's attention to parallels among examples.

Sander, 2013). Consider Table 1, which displays five examples from a single class for two tasks. Bolded terms for each task are clearly related, and to a person, suggest abstractions that help relate other terms to the task. This helps with disambiguation: that the word *Pluto* is the planet and not Disney's character is inferred not just by within-example evidence (e.g. *mission*) but also by cross-example presence of *Mars* and *astronauts*.

Cross-example analysis also reveals the amount of generalization warranted. For a word associated with a label, word embeddings give us neighbors, which often are associated with that label. What they do not tell us is the extent this associated-with-same-label phenomenon holds; that depends on the granularity of the classes. Cross-example analysis is required to determine how neighbors at various distances are distributed among labels in the training data. This should allow us to include *barley* and *peaches* as evidence for a class like *Agriculture* but only *barley* for *Grains*.

Most existing systems ignore cross-example parallelism and thus miss out on a strong classification signal. We introduce a flexible method for controlled generalization that selects syntacto-semantic features from sparse representations constructed by Category Builder (Mahabal et al., 2018). Starting with sparse representations of words and their contexts, a tuning algorithm selects features with the relevant kinds and appropriate amounts of generalization, making use of parallelism among examples. This produces task-specific dense embeddings for new texts that can be easily incorporated into classifiers.

Our simplest model, CBC (Category Builder Classifier), is a feed-forward network that uses only CB embeddings to represent a document. For small amounts of training data, this simple model dramatically outperforms both CNNs and BERT (Devlin et al., 2018). When more data is available, both CNNs and BERT exploit their greater capacity and broad pre-training to beat CBC. We thus create CBCNN, a simple combination of CBC and the CNN that concatenates their pre-prediction layers and adds an additional layer. By training this model with a scheduled block dropout (Zhang et al., 2018) that gradually introduces the CBC sub-network, we obtain the benefits of CBC in low data scenarios while obtaining parity with CNNs when plentiful data is available. BERT still dominates when all data is available, suggesting that further combinations or ensembles are likely to improve matters further.

dataset	k	train/test/dev	size range
20NG	20	15076/1885/1885	513/810
reuters	8	6888/862/864	128/3128
spam	2	3344/1115/1115	436/2908
attack	2	10000/2000/2000	1126/8874

Table 2: Data sizes, and the disparity between the smallest and the largest class in training data. The k column indicates the number of classes in the task.

2 Evaluation Strategy

Our primary goal is to study classifier performance with limited data. To that end, we obtain learning curves on four standard text classification datasets (Table 2) based on evaluating predictions on the full test sets. At each sample size, we produce multiple samples and run several text classification methods multiple times, measuring the following:

- **Macro-F1 score**. Macro-F1 measures support for all classes better than accuracy, especially with imbalanced class distributions.
- **Recall for the rarest class**. Many measures like F1 and accuracy often mask performance on infrequent but high impact classes, such as detecting toxicity (Waseem and Hovy, 2016))
- **Degenerate solutions**. Complex classifiers with millions of parameters sometimes produce degenerate classifiers when provided very few training examples; as a result, they can skip some output classes entirely.

The datasets we chose for evaluation, while all multi-class, form a diverse set in terms of the number of classes and kinds of cohesion among examples in a single class. The former clearly affects training data needs, while the latter informs appropriate generalization.

- **20 Newsgroups** 20Newsgroups (20NG) contains documents from 20 different newsgroups with about 1000 messages from each. We randomly split the documents into an 80-10-10 train-dev-test split. The classes are evenly balanced.

- **Reuters R8.** The Reuters21578 dataset contains Reuters Newswire articles. Following several authors (Pinto and Rosso, 2007; Zhao et al., 2018, for example), we use only the eight most frequent labels. We begin with a given 80/10/10 split. Given that we focused on single-label classification, we removed items associated with two or more of the top eight labels (about 3% of examples). Classes are highly imbalanced. Of the 6888 training examples, 3128 are labeled *earn*, while only 228 examples are of class *interest* and only 128 are *ship*.

- **Wiki Comments Personal Attack.** The Wikipedia Detox project collected over 100k discussion comments from English Wikipedia and annotated them for presence of personal attack (Wulczyn et al., 2017). We randomly select 10k, 2k, and 2k items as train/dev/test. 11% are attacks.

- **Spam** The SMS Spam Collection v.1 has SMS labeled messages that were collected for mobile phone spam research (Hidalgo et al., 2012). Each of the 5574 messages is labeled as *spam* or *ham*.

3 Identifying Generalizing Features

In this section, we explicate the source of features, discuss the properties relevant to generalization by focusing on one feature in isolation, and present the overall feature selection method. The overview in Figure 1 displays the order of operations: identify *generalizing features* based on the training data (done once), and for each document to be classified, convert it to a vector, where each entry corresponds to a generalizing feature.

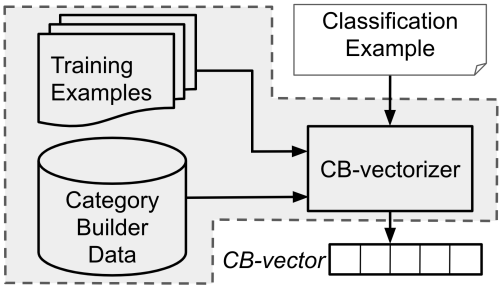

Figure 1: **Shaded Region:** We use Category Builder data (Mahabal et al., 2018) and identify generalizing features in training data, producing a *vectorizer*. This is done once. **Unshaded:** Given a document, the vectorizer produces a dense vector usable in deep networks.

Feature	Prototypical Supports
allergen as X	*pollen, dander, dust mites, soy, perfumes, milk, smoke, mildew*
liter of X	*water, petrol, milk, fluid, beer*
serve with X	*rice, sauce, salad, fries, milk*
flour mixture	*butter mixture, rubber spatula, dredged, creamed, medium speed, sifted, milk*
replacer	*colostrum, calves, whole milk, inulin, pasteurized, weaning*

Table 3: A few features (among hundreds) evoked by *milk*, with top n-grams in their support. **Above dashed line (F_S)** fit in tidy categories (here, *allergen*, *fluid*, and *food* are rough glosses). **Below dashed line (F_C)** are not describable by simple labels—the evoking terms have different parts of speech and instead display *situational coherence*, e.g. association with the process of mixing flour or with animal husbandry (a *replacer* is milk formula for calves).

3.1 Category Builder

Our source of generalizing features is Category Builder (CB) (Mahabal et al., 2018), which constructs a sparse vector space derived from parsed corpora (Erk, 2012). CB constructs features for *n-grams* (not just unigrams) that are the union of syntactic context features F_S and co-occurrence features F_C. Consider *milk*: an F_S feature is $gallon \xrightarrow{prep} of \xrightarrow{pobj} X$ and F_C features include *goat*, *cow*, *drink*, *spill*, etc. Table 3 provides other examples of features evoked by *milk*, along with other n-grams which evoke them. For present purposes, we can treat CB as a matrix with n-grams as rows and features in F_C and F_S as columns. The entries of CB are weights that give the association strength between an n-gram and a feature; these weights are an asymmetric variant of pointwise-

mutual information (Mahabal et al., 2018).

3.2 Properties of Generalizing Features

Which features generalize well depends on the granularity of classes in a task. Useful features for generalization strike a balance between *breadth* and *specificity*. A feature that is evoked by many words provides generalization potential because the feature's overall support is likely to be distributed across both the training data and test data. However, this risks over-generalization, so a feature should also be sufficiently specific to be a precise indicator of a particular class.

A key aspect of choosing good features based on a limited training set is to resolve referential ambiguity (Quine, 1960; Wittgenstein, 1953) to the extent supported by the observed uses of the words. To illustrate, consider the *grains* class in the Reuters Newswire dataset. The word *wheat* can evoke the features at different levels of the taxonimical hierarchy: *triticum* (the wheat genus), *poaceae* (grass family), *spermatophyta* (seeded plants), *plantae* (plant kingdom), and *living thing*. The first among these has low breadth and is evoked only by *wheat*. The second is far more useful: specific and yet with a large support, including *maize* and *sorghum*. The final feature is too broad. In general, the most useful features for generalization are the intermediate features, also known as Basic Level Categories (Rosch et al., 1976).

Another important aspect of generalization comprises the *facets* of meaning. For example, the word *milk* has facets relating it to other liquids (e.g., *oil*, *kerosene*), foods (*cheese*, *pasta*), white things (*ivory*), animal products (*honey*, *eggs*), and allergens (*pollen*, *ragweed*). Along these axes, generalization can be more or less conservative; e.g., both *cheese* and *tears of a phoenix* are animal products, but the former is semantically closer to *milk*. Looking back at Table 3, the utility of individual features evoked by *milk* for tasks involving related topics varies; e.g., does the classification problem pertain to *food* or *animal husbandary*?

3.3 Focus on a single feature

A single *generalizing feature* is associated with many n-grams, each of which evokes it (with different strengths). Table 4 displays n-grams that evoke the feature *co-occurrence with* ***Saturn V***, as discovered by unsupervised analysis of a large corpus of web pages. The table further displays the interaction of this unsupervised feature with supervised data, specifically, with the label *sci.space* in 20NG, when using a size 320 training sample that contain only 18 *sci.space* documents. Counts for some evoking terms are shown within and outside this class, for both training and test data.

		Training		Testing	
n-gram	wt	C	$\overline{C}$	C	$\overline{C}$
apollo	8.93	1	1	5	1
launch pad	8.52	0	0	1	0
rocket	7.32	3	1	8	0
rockets	7.27	2	0	4	1
liftoff	6.92	1	0	1	0
space shuttle	6.27	0	0	4	0
space station	6.19	0	0	4	3
payload	4.23	0	0	5	0
shuttle	2.57	2	0	15	3
kennedy	2.30	1	0	1	4
capacity	1.95	0	1	0	4

Table 4: Some evoking n-grams associated with the CB feature *co-occurrence with* ***Saturn V*** and pivoting on the class *sci.space*. Counts for n-grams in training (sample size 320) and test data are shown, within *sci.space* (C) and outside ($\overline{C}$). Bolded n-grams are not seen in training but occur in test, providing generalization. The dashed line represents a threshold; higher scoring n-grams are more cohesive, and thresholding can make a feature cleaner by decreasing semantic drift.

Notation. We introduce some notation and explicate with Table 4. We have a labelled collection of training documents T. T_l is the training examples with label l. The *positive support set* $\Psi_l(f,t)$ is the set of n-grams in T_l evoking feature f with weight greater than t, here, {*apollo, rocket, ..., shuttle*} for t=2.3. The *positive support size* $\Lambda_l(f,2.3)$=$|\Psi_l(f,2.3)|$=5 and the *positive support weight* $\lambda_l(f,2.3)$ is the sum of counts of supports of f in l with weight greater than 2.3, here 1+3+2+1+2=9. Analogously the *negative support weight* $\lambda_{\overline{l}}(f,2.3)$ is the sum of counts from outside T_l; here, 1+1=2 since {*apollo, rocket*} were seen outside *sci.space* once each.

What makes this feature (*words that have co-occurred with Saturn V*) well suited for *sci.space* is that many evoking words here are associated with the label *sci.space*. What confirms the benefit is the limited amount of negative support. Crucially, the bolded terms do not occur in the training data, but do occur in the test data. (We stress that we include these counts here only for this exam-

ple; our methods do not access the test data for feature selection in our experiments.)

That said, we must limit potential noise from such features, so we seek **thresholded features** $\langle f, t \rangle$, as suggested by the dashed line in Table 4. Items below this line are prevented from evoking f. We choose the highest threshold such that dropped negative support exceeds dropped positive support. This is determined simply by going through all the supports of a feature, sorted by ascending weight, and checking the positive and negative support of all features with smaller versus greater weight given the class. The weight of the feature at this cusp is used as the threshold of the feature for this particular class. This $\langle f, t \rangle$ pair then forms one element of the CB-vector used as a feature for classification.

Given the labeled subsets of T and this feature thresholding algorithm, we produce a vectorizer that embeds documents. The values of a document's embedding *are not* directly associated with any class. Such association happens during training. Although *sci.space* accounts for just 6% of the documents, 75% of documents that contain an n-gram evoking the *Saturn V* feature are in that class. A classifier trained with such an embedding should learn to associate this feature with that class, and an unseen document containing the unseen-in-training term *space shuttle* stands a good chance to be classified as *sci.space*.

The feature displayed in Table 4 is useful for the 20NG problem because it contains a class related to space travel. This feature has no utility in spam classification or in sentiment classification, since, for those problems, seeing *rocket* in one class does not make it more likely that a document containing *space station* belongs to that same class. This example illustrates why a generalization strategy must incorporate both what we can learn from unsupervised data as well as (limited) labeled training data.

3.4 Overall feature selection

We now describe how we use the training data T to produce a set of features-and-threshold pairs; each chosen feature-with-threshold $\langle f, t \rangle$ will be one component in the CB-vectors provide to classifiers. Calculation of features for a single class is a three step process: (i) for each feature f, choose a threshold t (as discussed above) (ii) score the resultant $\langle f, t \rangle$ (iii) filter useless or redundant $\langle f, t \rangle$.

Given a label l and a feature f, we implicitly produce a table of supporting n-grams and their distribution within and outside l (e.g. as in Table 4). This involves computing the precision of a feature at a given threshold value, comparing it to the class probability and deciding whether to keep it.

Recall the positive support $\lambda_l(f, t)$ and negative support $\lambda_{\overline{l}}(f, t)$ defined previously. The **precision** of f at threshold t is $\mu_l(f, t) = \frac{\lambda_l(f,t)}{\lambda_l(f,t)+\lambda_{\overline{l}}(f,t)}$, (this is $\frac{9}{11}$ in the example of Table 4, with t=2.3). However, since we are often dealing with low counts, we smooth the precision toward the empirical class probability of l, $p(l) = \frac{|T_l|}{|T_l|+|T_{\overline{l}}|}$.

$$\tilde{\mu}_l(f, t) = \frac{\lambda_l(f, t) + p(l)\alpha}{\lambda_l(f, t) + \lambda_{\overline{l}}(f, t) + \alpha}$$

The score $S_l(f, t)$ is reduction in error rate of the smoothed precision relative to the base rate:

$$S_l(f, t) = \frac{\tilde{\mu}_l(f, t) - p(l)}{1 - p(l)}$$

We retain a thresholded feature if it is generalizing ($\Lambda_l(\langle f, t \rangle) > 1$), has better-than-chance precision (we use $S_l(\langle f, t \rangle) > 0.01$), and is not redundant (i.e., its positive support has one or more terms not present in positive supports of higher scoring features).

3.5 Creating the CB-vector

Each vector dimension corresponds to some $\langle f, t \rangle$. The evocation level of f is the sum of its evocation for the n-grams in the document d, $e_d(f) = \sum_{w \in d} CB(w, f)$. The vector entry is $\frac{e_d(f)}{t}$ when $e_d(f) >= t$, and is clipped to 0 otherwise.

4 Models

As benchmarks, we use a standard CNN with pretrained embeddings (Kim, 2014) and BERT (Devlin et al., 2018).[1] For CNN, we used 300 filters each of sizes 2, 3, 4, 5, and 6, fed to a hidden layer of 200 nodes after max pooling. Pretrained vectors provided by Google were used.[2] For BERT, we used the `run_classifier` script from GitHub and used the BERT-large-uncased model.

We use the pre-computed vocab-to-context association matrix provided as part of the open

[1] https://github.com/google-research/bert

[2] https://code.google.com/archive/p/word2vec

source Categorial Builder repository.[3] This contains 194,051 co-occurrence features (F_C) and 954,276 syntactic features (F_S).

CBC model. The *CB-vector* containing the derived features from the training dataset and Category Builder can be exploited in various ways with existing techniques. The simplest of these is to use a feed-forward network over the *CB-vector*. This model does not encode the tokens or any word order information—information which is highly informative in many classification tasks.

CBCNN model. Inspired by the combination of standard features and deep networks in Wide-and-Deep models (Cheng et al., 2016), we pair the CBC model with a standard CNN, concatenating their pre-prediction layers, and add an additional layer before the softmax prediction. In early experiments, this combined model performed worse than the CNN on larger data sizes, as the network above the CB-vector effectively stole useful signal from the CNN. To ensure that the more complex CNN side of the network had a chance to train, we employed a block dropout strategy (Zhang et al., 2018) with a schedule. During training, with some probability, all weights in the CB-vector are set to 0.5. The probability of hiding decreases from 1 to 0 using a parameterized hyperbolic tangent function $p_k = \frac{2}{e^{Cx}+1}$. Lower values of C lead to slower convergence to zero. The effect is that the CBC sub-network is introduced gradually, allowing the CNN to train while eventually taking advantage of the additional information.

The natural strategy of replacing with 0s (instead of 0.5 as above) was tried and also works, but less well, since the network has no way to distinguish between genuine absence of feature and hiding. In CB-vector, non-zero values are at least 1, and thus 0.5 does not suffer from this problem.

5 Experiments

Our primary goal is to improve generalization for low-data scenarios, but we also want our methods to remain competitive on full data.

5.1 Experimental setup

We compare different models across learning curves of increasing the training set sizes. We use training data sizes of $40, 80, \ldots, 5120$ as well as the entire available training data. For each training size, we produce three independent samples by uniformly sampling the training data and training each model three times at each size. The final value reported is the average of all nine runs. All models are implemented in Tensorflow. Batch sizes are between 5 and 64 depending on training size. Training stops after there is no macro-F1 improvement on development data for 1000 steps.

For evaluation, we focus primarily on macro-F1 and recall of the rarest class. The recall on the rarest class is especially important for imbalanced classification problems. For such problems, a model can obtain high accuracy by strongly preferring the majority class, but we seek models that effectively identify minority class labels. (This is especially important for active learning scenarios, where we expect the CB-vectors to help with in future.)

5.2 Results: low data scenarios

Figure 2 shows learning curves giving macro-F1 scores and rarest class recall for all four datasets. When very limited training data is available, the simple CBC model generally outperforms the CNN and BERT, except for the Spam dataset. The more powerful models eventually surpass CBC; however, the CBCNN model provides consistent strong performance at all dataset sizes by combining the generalization of CBC with the general efficacy of CNNs. Importantly, CBCNN provides massive error reductions with low data for 20NG and R8 (tasks with many labels).

Table 5's left half gives results for all models when using only 320 training examples. For 20NG, CNN's macro-F1 is just 43.9, whereas CBC and CBCNN achieve 61.7 and 62.4—the same as CNN performance with four times as much data. These models outperform CNN on R8 as well, reaching 83.7 vs CNN's 74.1, and also on the Wiki-attack dataset, achieving 80.6 vs CNNs 74.0. BERT fails to produce a solution for the two datasets with >2 labels, but does produce the best result for Spam—indicating an opportunity to more fully explore BERT's parameter settings for low data scenarios and to fruitfully combine CBC with BERT.

Rarest class recall is generally much better with less data when exploiting CB-features. For example, with 320 training examples for R8, CNNs reach 36.2 whereas CBCNN scores 76.2. Prediction quality with few training examples (especially getting good balance across all labels) also inter-

[3] https://github.com/google/categorybuilder

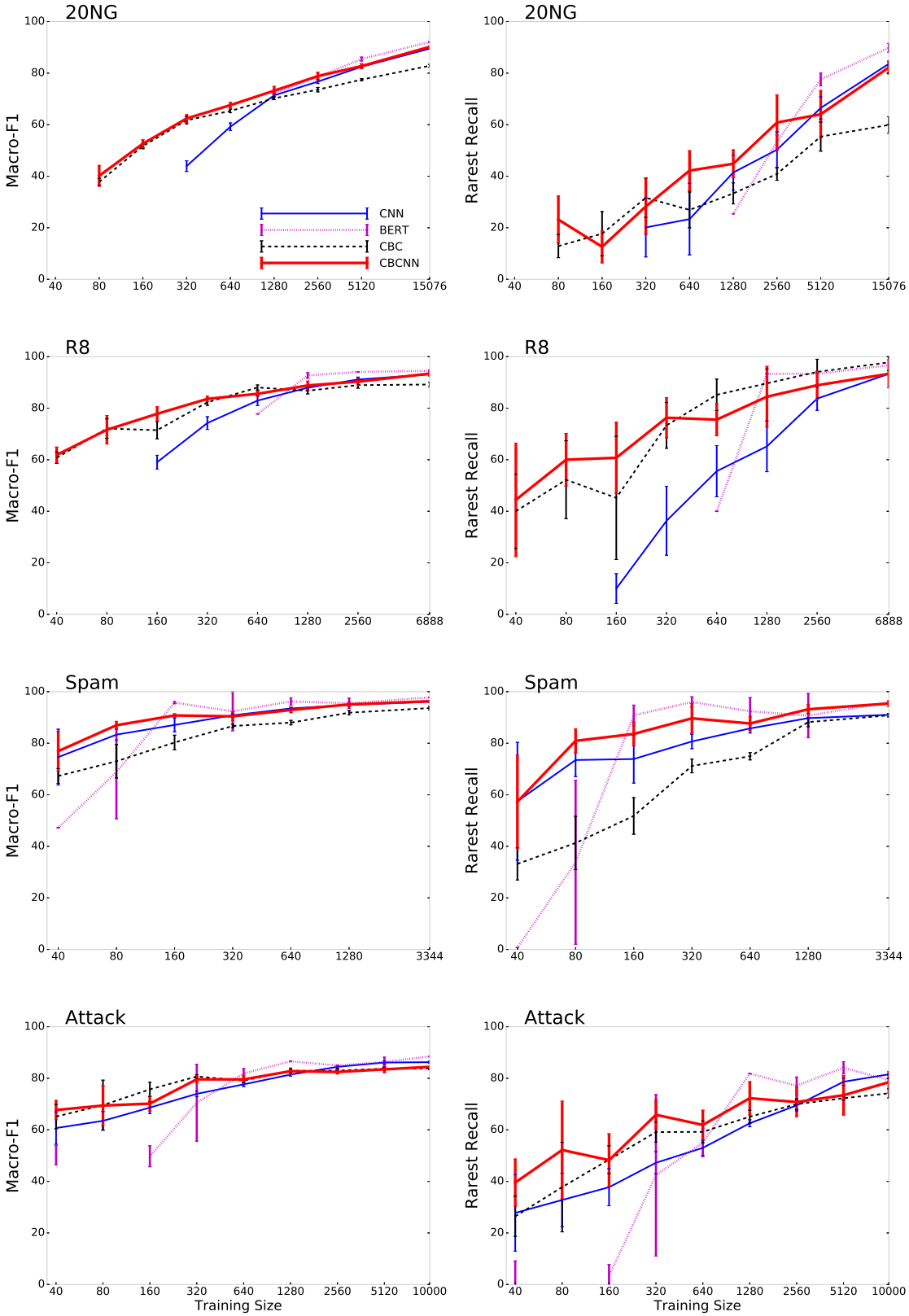

Figure 2: **Left:** F1 score by training size for 20NG, Reuters, SMS Spam, and Wiki-attack. Data is shown for non-degenerate models, and hence CNN and BERT start at higher sizes (see Table 6). **Right:** Recall for the rarest class for the same models.

		320 Training Examples				Full Training Data			
Data	k	CBC	CNN	CBCNN	BERT	CBC	CNN	CBCNN	BERT
20NG	20	61.7	43.9	**62.4**	—	82.9	89.5	90.2	**92.0**
R8	8	82.3	74.2	**83.6**	—	89.2	93.1	93.4	**94.4**
Spam	2	80.3	87.1	90.8	**95.7**	93.7	96.1	96.3	**97.8**
Attack	2	**80.7**	74.0	79.6	70.5	83.8	86.2	84.5	**88.2**

Table 5: Macro-F1 scores on all data sets when using 320 training examples (left) and when using all available training data (right). k is the number of classes. The CBCNN model provides the strongest overall performance across all data sizes. (Note that BERT produces degenerate solutions for the >2 class problems with 320 examples.)

Model	k	CBC	CNN	CBCNN	BERT
20NG	20	80	320	80	1280
R8	8	40	160	40	640
Spam	2	40	40	40	40
Attack	2	40	40	40	40

Table 6: Minimum training size at which a non-degenerate model was produced in any of 9 runs. With more classes, more data is needed by CNN and BERT to produce acceptable models. k is number of classes.

acts with other strategies for dealing with limited resources, such as active learning. For example, Baldridge and Osborne (2008) obtained stronger data utilization ratios with better base models and uncertainty sampling for Reuters text classification: better models pick better examples for annotation and thus require fewer new labeled data points to achieve a given level of performance.

Importantly, the CBC and CBCNN models take far less data to produce non-degenerate models (defined as a model which produces all output classes as predictions). CNN and BERT have a large number of parameters, and using these powerful tools with small training sets produces unstable results. Table 6 gives the minimum training set sizes at which each model produces at least one non-degenerate model. While it might be possible to ameliorate the instability of CNN and BERT with a wider parameter search and other strategies, nothing special needs to be done for CBC. It is likely that an approach which adaptively selects CBC or CBCNN and BERT would obtain the strongest result across all training set sizes.

For each dataset, among the 100 best features chosen (for training size 640), the breakdown of domain features (F_C) versus type features (F_S) is revealing. As expected, domain features are more important in a topical task such as 20NG (71% are F_C features), while the opposite is true for Spam (19%) and a toxicity dataset like Wiki Attack (23%). Reuters shows a fairly even balance between the two types of features (41%): it is useful for R8 to be topically coherent and also to hone in on fairly narrow groups of words that collectively cover a Basic Level Category.

5.3 Results: full data scenarios

Table 5 provides macro-F1 scores for all models when given all available training data. The CBC model performs well, but its (intentional) ignorance of the actual tokens in a document takes a toll when more labeled documents are available. The CNN benchmark, which exploits both word order and the tokens themselves, is a strong performer. The CBCNN model effectively keeps pace with the CNN—improving on 20NG and R8, though slipping on Wiki-Attack. BERT simply crushes all other models when there is sufficient training data, showing the impact of structured pre-training and consistent with performance across a wide range of tasks in Devlin et al. (2018).

6 Conclusion

We demonstrate an effective method for exploiting syntactically derived features from large external corpora and selecting the most useful of those given a small labeled text classification corpus. We show how to do this with the map provided by Category Builder n-grams to features, but other sources of well generalizing features have been exploited for text classification. These include topic models (Blei et al., 2003), ontologies such as WordNet (Bloehdorn and Hotho, 2004) and Wikipedia Category structure (Gabrilovich and Markovitch, 2009). It may be possible to use these other sources exactly as we use CB. Some of these sources have been manually curated, which makes them high quality but limits the size and facets. We have not yet explored their use because CB features seem to cover many of these

sources' strengths—for example, F_C features are like topics, and F_S features like nodes in ontologies. Nonetheless, a combination may add value.

Our focus is on data scarce scenarios. However, it would be ideal to derive utility at both the small and large labeled data sizes. This will likely require models that can generalize with contextual features while also exploiting implicit hierarchical organization and order of the texts, e.g. as done by CNNs and BERT. The CBCNN model is one effective way to do this and we expect there could be similar benefits from combining CBC with BERT. Furthermore, approaches like AutoML (Zoph and Le, 2017) would likely be effective for exploring the design space of network architectures for representing and exploiting the information inherent in both signals.

Finally, although we focus on multi-class problems here—each example belongs to a single class—the general approach of selecting features should work for multi-label problems. Our confidence in this (unevaluated) claim stems from the observation that we select features one class at a time, treating that class and its complement as a binary classification problem.

Acknowledgements

We would like to thank our anonymous reviewers and the Google AI Language team, especially Rahul Gupta, Tania Bedrax-Weiss and Emily Pitler, for the insightful comments that contributed to this paper.

References

Jason Baldridge and Miles Osborne. 2008. Active learning and logarithmic opinion pools for HPSG parse selection. *Natural Language Engineering*, 14(2):191222.

David M. Blei, Andrew Ng, and Michael Jordan. 2003. Latent Dirichlet Allocation. *JMLR*, 3:993–1022.

Stephan Bloehdorn and Andreas Hotho. 2004. Boosting for text classification with semantic features. In *International Workshop on Knowledge Discovery on the Web*, pages 149–166. Springer.

Ming-Wei Chang, Lev Ratinov, Dan Roth, and Vivek Srikumar. 2008. Importance of semantic representation: Dataless classification. In *AAAI*.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, pages 7–10, New York, NY, USA. ACM.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jacob Eisenstein. 2017. Unsupervised learning for lexicon-based classification. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Katrin Erk. 2012. Vector space models of word meaning and phrase meaning: A survey. *Language and Linguistics Compass*, 6(10):635–653.

Evgeniy Gabrilovich and Shaul Markovitch. 2009. Wikipedia-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research*, 34:443–498.

M. A. K. Halliday. 1985. *An introduction to functional grammar*. Arnold.

Jose Maria Gomez Hidalgo, Tiago A Almeida, and Akebo Yamakami. 2012. On the validity of a new SMS spam collection. In *Proceedings of the 11st ICMLA*, volume 2, pages 240–245. IEEE.

Douglas Hofstadter and Emmanuel Sander. 2013. *Surfaces and essences: Analogy as the fuel and fire of thinking*. Basic Books.

Douglas R Hofstadter. 2001. Analogy as the core of cognition. *The Analogical Mind: Perspectives from Cognitive Science*, pages 499–538.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar.

Abhijit Mahabal, Dan Roth, and Sid Mittal. 2018. Robust handling of polysemy via sparse representations. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 265–275.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. 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 2227–2237, New Orleans, Louisiana.

David Pinto and Paolo Rosso. 2007. On the relative hardness of clustering corpora. In *International Conference on Text, Speech and Dialogue*, pages 155–161. Springer.

Willard Van Orman Quine. 1960. *Word and object*. MIT press.

Eleanor Rosch, Carolyn B Mervis, Wayne D Gray, David M Johnson, and Penny Boyes-Braem. 1976. Basic objects in natural categories. *Cognitive psychology*, 8(3):382–439.

Devendra Sachan, Manzil Zaheer, and Ruslan Salakhutdinov. 2018. Investigating the working of text classifiers. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2120–2131, Santa Fe, New Mexico, USA.

Yangqiu Song and Dan Roth. 2014. On dataless hierarchical text classification. In *AAAI*.

Zeerak Waseem and Dirk Hovy. 2016. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL Student Research Workshop*, pages 88–93, San Diego, California.

Ludwig Wittgenstein. 1953. *Philosophical investigations*. John Wiley & Sons.

Ellery Wulczyn, Nithum Thain, and Lucas Dixon. 2017. Ex machina: Personal attacks seen at scale. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1391–1399. International World Wide Web Conferences Steering Committee.

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

Yuan Zhang, Jason Riesa, Daniel Gillick, Anton Bakalov, Jason Baldridge, and David Weiss. 2018. A fast, compact, accurate model for language identification of codemixed text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 328–337, Brussels, Belgium.

Taotao Zhao, Xiangfeng Luo, Wei Qin, Subin Huang, and Shaorong Xie. 2018. Topic detection model in a single-domain corpus inspired by the human memory cognitive process. *Concurrency and Computation: Practice and Experience*, 30(19):e4642.

Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

A Appendix

Changes to the Category Builder Matrix

Category Builder uses two matrices: one mapping items to features ($M^{\mathcal{V}\to\mathcal{F}}$), the other mapping features to items ($M^{\mathcal{F}\to\mathcal{V}}$). Two are needed since the relationship is asymmetric: the feature *X is a star sign* is more strongly associated with the term *cancer* than vice versa, and the two matrices are thus not exact transposes of each other, although they almost are. For this current work, we just use one matrix, $M^{\mathcal{V}\to\mathcal{F}}$. For syntactic features F_S, we directly use the Category Builder rows. For F_C features, however, Category Builder replaced the corresponding submatrix in $M^{\mathcal{V}\to\mathcal{F}}$ with an identity matrix as described in (Mahabal et al., 2018). We obtain that part of the matrix by copying the corresponding rows from $(M^{\mathcal{F}\to\mathcal{V}})^T$.

This new matrix will be made available as part of the Category Builder project.