Analyzing Cross-Domain Sentiment Classifiers: The Impact of Data Amount and Utilization

Robert Rainey
University of Memphis, Memphis TN
rarainey@memphis.edu

Abstract

Cross-domain models are important in natural language processing for their ability to train on one domain's data and apply it to other domain(s). This project seeks to look at the data aspect of cross-domain model performance, focusing on the different cases of resource availability and domain data similarity on the overall ability to accurately be transferred across domains. This project specifically trains a cross-domain classifier that is analyzed on two different domain datasets. 5 different experiments in data are tested to train the classifier, each having their own advantages and disadvantages for use in training models.

1 Introduction

In this section, I will discuss the problem that I wish to help with, the model I will be experimenting with, and how the model can help with the problem.

1.1 Introduction to the Problem

Training language models can be both computationally expensive and data resource reliant. For simpler models, these problems may not be as apparent, but larger models can take much more time, funding, and data to train. *Counting the Cost of Training Large Language Models* (Morgan, 2022) shows the time and money required to train particular LLMs. They can take upwards of $10,937,500 to fund and 110.5 days to train. Models may also require large amounts of labeled/unlabeled training data that may be limitedly available.

Identifying Transferable Information Across Domains for Cross-domain Sentiment Classification (Sharma et al., 2018) discusses that, generally, models must be retrained to learn new domain data: an impractical, expensive, and time-consuming task when there has been much work put into the previous domains' classifier. They discuss a method that allows us to reuse the previous work: cross-domain classifiers. While not a new concept, cross-domain models are designed to mitigate this expensive problem

1.2 Introduction to Cross-Domain Models

To introduce cross-domain models, let me provide an example. Suppose we are creating a sentiment classification model on product reviews. Let us say we have a model already trained on one category of product reviews, but we do not have one trained for another category. Through cross-domain models, we can reuse the resources we trained the first model on to develop a model that can identify the target domain.

Cross-domain models are models trained on one domain or data source and utilized on another domain. In this report, I often call the domain we train on the original domain and call the domain(s) we intend to apply it to the target domain(s). We can save resources by using past work on another our original domain without training an entirely new model. We can also save resources by training a model with both the original and target domain(s) in mind.

There are some obstacles to training cross-domain models, however. Most prominently is that there are special considerations you must have in data sources and planning when using a cross-domain model; applying the model to another

domain will not always have sufficient accuracy. For example, target domain-specific information will likely not be understood by the cross-domain model because the data does not exist in the original domain. How do we account for this problem and more?

1.3 Motivation

As stated, to cut down on computational costs and/or limited data samples, we can use cross-domain models to transfer past work to target domains. Still, this is often not as simple as just applying it to the other domain; we need to adapt our target data to work with the model.

In this project, I want to discuss and experiment with different cross-domain sentiment classifier training techniques. I want to find ways in which we can improve cross-domain accuracy, specifically focusing on the different ranges of data available. This will help gauge how much data quantity (and, in some cases, quality) will affect a model's performance.

I also wanted to see if domain similarity affected cross-domain performance in a significant way. Do two domains that share real-world "common attributes" perform better than dissimilar domains? Think, for example, electronics and video game product reviews. One might classify them as similar categories in our understandings of them.

To this end, I will train a sentiment classifier for one dataset and apply it two others: one more "similar" semantically to the original dataset and the other more "different."

2 Data

I use the *Amazon Review Data (2018)* dataset located at:
https://nijianmo.github.io/amazon/index.html
(Ni et al., 2019).

It includes Amazon review data from May 1996 to Oct. 2018 divided into different categories of items. The three review categories I use are the electronics, video games, and grocery/gourmet food datasets. More specifically, I use the 5-core dataset versions of these datasets, meaning that each product that has reviews in the dataset has at least 5 reviews and that each reviewer has reviewed at least 5 items. This downsizes the file size tremendously and leaves reviews that are more likely to be thorough (as the reviewers are not one-time reviewers).

The files are in JSON format and need to be extracted before they are used. I simply used a ZIP extractor like 7Zip to extract the files before use.

The electronics dataset will train the cross-domain classifier. The other two datasets will be the target domains. Models will be trained on the two target domains to test their performance against the cross-domain classifier. Electronics and video games are designed to be the two "similar" domains while grocery/gourmet is the outlier. Intuitively, we might expect an electronics classifier to work better on the video game domain than the grocery domain.

There are 6,739,590 electronics reviews, 497,577 video games reviews, and 1,143,860 grocer/gourmet food reviews. Each review has the following structure:

{"image": ["https://images-na.ssl-images-amazon.com/images/I/71eG75FTJJL._SY88.jpg",
"overall": 5.0,
"vote": "2",
"verified": True,
"reviewTime": "01 1, 2018",
"reviewerID": "AUI6WTTT0QZYS",
"asin": "5120053084",
"style": {"Size:": "Large", "Color:": "Charcoal"},
"reviewerName": "Abbey",
"reviewText": "I now have 4 of the 5 available colors of this shirt... ",
"summary": "Comfy, flattering, discreet--highly recommended!",
"unixReviewTime": 1514764800}

As I am training the model to classify the rating based on the review's text, we only need to focus on two attributes: reviewText and overall. "reviewText" is the review's text (what we identify). "overall" is the rating given to the review (the class we want to associate with the reviewText).

Now that we have the data, we need to format it for both easier use and better information extraction from the model.

3 Preprocessing

There are three types of preprocessing that need to be done: file preprocessing, splitting, and data preprocessing.

3.1 File Preprocessing

The files for the datasets are both very large and contain information that we do not want in our training process.

First, reviews with empty reviewText values need to be culled. Reviews that provide no reviewText also provide no data to train on, harming the training process. I read the JSON files line-by-line and save only the reviews with reviewText values longer than 3 words. This not only culls empty reviews, but also removes short reviews and reduces the file size.

Secondly, the star-based class system proves hard to train on for a variety of reasons: ambiguity between some classes (like 4 and 5 stars and 1 and 2 stars) and there are smaller samples of 2-star reviews than most other classes. To mitigate these problems, I convert the 5-class system into a 3-class system. Any reviews with an "overall" value of 1 or 2 gets a new "overall" of 0 (negative), 3 star reviews get a score of 1 (neutral), and 4 or 5 star reviews get a score of 2 (positive).

With these changes, our data is more informative to the model and the computational and storage load (from the original larger files) is reduced.

3.2 Splitting into Training and Testing Sets

Now that the reviews have been filtered and mapped to simpler classes, they need to be divided into training and testing sets for each dataset.

Each domain dataset has 120,000 training samples and 12,000 testing samples. We use a large amount of samples because we are training the word vectorization model and classifier model from scratch (except in one experiment). This split is the max amount we can split the smallest dataset, video games, into equal classes.

Each class for the datasets gets fair representation. There are 40,000 positive, neutral, and negative reviews each in the training set. There are 4,000 of each in the training set as well.

In making these splits, the files are shuffled, ensuring random samples in each run of the dataset initialization. This is not random per experiment though. Each experiment has the same samples.

These datasets are saved into new JSON files for quicker use in the future.

3.3 Data Preprocessing

Now that the reviews have been split for each of the datasets, the data needs to be preprocessed to increase training efficiency.

First, the extraneous attributes are removed. As stated earlier, the only attributes we need to train the classifiers on are the "reviewText" and "overall" attributes. A function resaves the data entries with only these two attributes.

Second, the reviewText itself must be preprocessed to effectively train the word vectorization and neural network models. I apply common data preprocessing techniques. The reviewText values are lowercased, scrubbed of non-alphanumeric characters, tokenized, and each word stemmed (reducing the token to a root-form). I originally implemented stop-word removal as well, but I found that it led to slightly worse model performance in my tests.

An example of a data entry currently:

{"overall": 2
"reviewText": ['thi', 'r', 'case', 'is', 'the', 'best', 'i' 'have']}

Later in the program, when the word vectorization model is trained, the data is further processed by converting the reviewTexts into their vector representations by the word vectorization model. The vectorized data is then stored in a new class, ReviewDataset, which is responsible for converting the data into their tensor representations. These ReviewDatasets are then used in PyTorch's DataLoader objects to pass them to the network model.

4 Model

To train the model, I use Word2Vec embeddings to vectorize the reviewTexts into a form that the neural network can use. Each of my experiments uses different data to train the Word2Vec model because of the simulations of different quantities of data available. I set the vector size to 300 in my Word2Vec models, and I trained them for 10 epochs.

My neural network has two fully-connected layers. The first takes a dimension of 300 (the number of embeddings I set for the Word2Vec model) and outputs a dimension of 128 (the hidden layer dimension). It uses ReLU. The second layer takes in a dimension of 128 (the hidden layer dimension) and outputs 3 dimensions, the 3 different classes. To do so, it goes through softmax.

While there are more robust and accurate models to train a sentiment classifier on (like BERT to fine-tune on or LSTMs), my project focuses more on the effects of different data qualities and quantities available from the target domains on the cross-domain classifier. While performance was important, the number of experiments and factors promoted a quicker model that captures the comparisons between experiments rather than building a slower but better model.

The model was trained and evaluated on the test data using cross-entropy loss (since it is a multi-class classifier) with the Adam optimizer and a learning rate of 0.001. Each model is trained for 10 epochs. In each epoch, the cross-domain model (electronics model) is evaluated on the target domains (video game and grocery test datasets).

5 Experiments

Now that the model and data have been defined, the project focus is turned back to the cross-domain perspective.

I divide my project into 5 main experiments, each with their own distinct use cases and advantages: the individual electronics model, the pre-trained word2vec model, the shared word2vec model, the summary model, and the limited cross-domain data model.

Each experiment alters the amount and quality of data available in some way. These give differing results on cross-domain accuracy and even original domain accuracy.

5.1 Individual Electronics Model

This model represents the results of applying no special considerations for cross-domain target classification. It uses only the electronics model data as input for both the Word2Vec model and sentiment classifier network model.

As expected, this does not work well for cross-domain classification. At epoch 10, the cross-domain sentiment classifier gets an accuracy of 70% on its own electronics reviews, ~36% on the video games reviews, and ~39% on the grocery data.

Comparing this performance to separately trained video games and grocery models, the target domain-trained models perform much better on themselves than the electronics model does on them. While the electronics model attains an accuracy of ~36% on the video games data, the video games model performs at ~68% on itself. Likewise, the grocery data on the electronics model performs at ~39% but the grocery model itself performs at ~74%.

This has a few implications.

Without consideration for cross-domain classification, the electronics model performs terribly as a cross-domain classifier. The individually trained models for video games and grocery performed much better. This was likely because there was no representation of the other two datasets in the electronics model's word2vec model. This is important going forward.

Surprisingly, the electronics model performed worse on the video games model. I think this is due to the two having perhaps contrasting meanings for words. Grocery and video games coexist better because they are likely to have less contrast. For example, a video game "running fast" is likely a good thing. An electronic "running fast" could be a good thing, but in the context, it is equally likely to be talking about "the battery running fast". The video game model is also likely harder to learn. It will consistently perform the worst going forward. This might indicate that we should not rely on our own semantics for similarity. There could be differences in ways we do not immediately recognize.

The main use case I can see for the individually trained model is for situations where there is NO unlabeled training data available for the target domain and there is enough similarity between the two domains that the accuracy would not have as much of a gap. On the plus side, this makes it the most flexible as it requires nothing from the target domain. It just might not be able to identify anything from the target either.

5.2 Pre-trained Word2Vec Model

This model represents a similar case to the last: you have insufficient target data. A pre-trained word2vec model is used to help fill in the word vectorization gaps from the different datasets. I use the GoogleNews-vectors-negative300 pretrained word2vec model (*Google*, 2013). This model is used to obtain the word embeddings for the data. The electronics reviews are still used to train the cross-domain sentiment classifier network.

This model performs much better than the individually trained model for cross-domain classification. At epoch 10, the cross-domain

sentiment classifier gets an accuracy of around 64% on its own electronics reviews which is a significant drop from the last model. But, it performs much better on the other datasets: ~59% on the video games data and ~59% on the grocery data.

This model performs much better as a cross-domain classifier than the individually trained model. This came at the cost of the original domain's accuracy, however, dropping by about 6%. Still, it works better generally due to the pre-trained model. It is evident that you lose some of the domain-specific information, especially since the pre-trained word2vec model was trained on news articles, not reviews.

Like the individually trained model, its main use case is for when there is no target-domain information for the model to use. Its better cross-domain performance may incentivize its use over the last.

5.3 Shared Word2Vec Model

This model assumes you have a significant amount of unlabeled data for the target domains. A new word2vec model is trained with a combination of all of the domain's datasets to train it. The embeddings, thus, consider all three domain's reviewText values.

This gives majorly improved performance across the board. At epoch 10, the cross-domain model still performs well on its own original domain at ~69% accuracy. It also performs better as a cross-domain classifier for the target domains: ~64% accuracy for the video game dataset and ~65% accuracy for the grocery dataset.

This model has performed the best so far as an individual classifier and a cross-domain classifier. Combining the data to create the word2vec gives it knowledge on more domain-specific language and how to classify it.

The main use case for this model is if there is a large amount of unlabeled data for the target domains. While I contributed an equal amount of data to the labeled electronics data, experiments may need to be done to gauge its effectiveness when given less unlabeled target domain data.

5.4 Summary Model

This is a more unique, data-specific model proposed in the paper, *Making the Best Use of Review Summary for Sentiment Analysis* (Yang et al., 2020). They state that using user-generated summaries work just as well as the reviewText attribute information. Our data has summaries available, so I wished to test this. This is not a cross-domain specific test, but I thought it would perform better since the summaries contained more general language like "good", "five stars", etc. This model shares similarities to the shared word2vec model as we combine the summaries of each dataset into the word2vec model.

It does perform the best of all the models if you provide its summaries as testing data. At epoch 10, it performs at ~71.5% on its original electronics domain. On the cross-domain targets, it performs ~64.5% on the video games and ~70% on the grocery dataset.

This model is very good at both cross-domain and original domain classification, but summaries on the input data are needed for it to perform well. Given raw reviews (rather than summaries), it performs at only ~62% on its own domain while the target domains score ~57% on video games and 62% on grocery. This is about as good as the pre-trained model.

This model performs the best of all the models, but it has the highest requirements, summary of the input text. This summary generation could maybe be delegated to AI models, but that could have problems. It did train much faster because it had less text but has a niche use case.

5.5 Limited Cross-Domain Data Model

The idea for this model came from the paper, *Cross-Domain Sentiment Classification with Target Domain Specific Information* (Peng et al., 2018). They state that using a small amount of domain-specific information will improve cross domain model performance. I tried this only between the electronics model and video games model as I wanted to more so focus on one cross-domain model this time. I also did not want to hinder results. I introduced a small sample of 1,200 video game review samples (1% of training dataset) to the electronics model of 120,000 samples. Now the training set has 121,200 samples.

It performs very well for such a small amount of labeled samples being introduced. At epoch 10, the original domain performs well on itself at ~70% accuracy. On the video game target domain, it performs with around 62% accuracy. This is not the best overall, but it is much more flexible than some past experiments.

The model performs surprisingly well on the video game domain despite the limited labeled training samples I gave it. The main issue is that it is labeled data. 1,200 is small in comparison to the overall model but is still a lot of data. The ratio of target domain to original domain labeled data may need to be experimented with.

The main use case is when you have available but limited labeled data available. If you have a small amount of labeled data, this model will perform much better than the individually trained alternative. If you have a small amount of unlabeled data instead, you may try experimenting using it in the shared Word2Vec model but not in the training of the neural network like done here.

6 Discussion and Takeaways

As seen from the experiments conducted in this project, cross-domain models can help reduce the computational and dataset burden that training new models can have.

However, one must make special preparations to attain an effective cross-domain classifier for the target domain. As seen in experiment 1, the model performs poor when not exposed to any of the target domains' data.

As seen in experiments 2, 3, and 5, there are various ways to improve cross-domain accuracy depending on how much and what kind of data you have available on the target domain.

2. You can use pre-trained Word2Vec models or other general resources when you have no target domain information.

3. You can train effective cross-domain classifiers if given lots of unlabeled training data for the target domains.

5. You can train effective cross-domain classifiers given a small amount of labeled training data for the target domains.

Experiment 4 shows that there are unconventional yet effective ways to make your model perform better by using the summaries.

All the experiments also show that one cannot rely on their semantic assumptions (like I did) when picking "similar" datasets. Despite video games and electronics being similar, electronics model performed better on the grocery model.

Overall, there are various ways to improve cross-domain accuracy regardless of the situation you are in.

7 Access to my Project

My project's .ipynb file can be found at the link below. It walks through the entire process.

https://drive.google.com/file/d/1ROB70gLe9JOt_9MO39oUncAm34VCbcCe/view?usp=drive_link

References

Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, Hong Kong, China. Association for Computational Linguistics.

Minlong Peng, Qi Zhang, Yu-gang Jiang, and Xuanjing Huang. 2018. Cross-Domain Sentiment Classification with Target Domain Specific Information. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2505–2513, Melbourne, Australia. Association for Computational Linguistics.

Raksha Sharma, Pushpak Bhattacharyya, Sandipan Dandapat, and Himanshu Sharad Bhatt. 2018. Identifying Transferable Information Across Domains for Cross-domain Sentiment Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 968–978, Melbourne, Australia. Association for Computational Linguistics.

Sen Yang, Leyang Cui, Jun Xie, and Yue Zhang. 2020. Making the Best Use of Review Summary for Sentiment Analysis. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 173–184, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Google. 2013. Google code archive - long-term storage for google code project hosting.

Timothy Morgan. 2022. Counting The Cost Of Training Large Language Models.