Good project, sir: few-shot learning and cryptocurrency tweets

Davin Lawrence
Department of Computer Science
University of Texas
Austin, TX 78712
dhuck@cs.utexas.edu

Abstract

Tweets related to cryptocurrency provide a complex problem space to explore few-shot classification techniques on real world data. In this project, I explore meta learning and cloze questions as two methods towards making good predictions on small amounts of data.

1 Introduction

Few-shot classification is a problem of great interest because it allows models to be trained on very little classified data. Data classification is a long, arduous, and expensive task. Many solutions to few-shot, and even zero-shot, classification task have been proposed the last few years. In this project, I take aspects of two different approaches–meta-learning and Pattern-Exploitative Training–and attempt to classify real-world cryptocurrency twitter data.

Cryptocurrency was selected as the subdomain of tweets for a number of reasons. First, there is a high amount of spam associated with cryptocurrency on Twitter. Likewise, there is high prevalence of scam phishing taking place on daily basis. Third, several subjects fall under cryptocurrency, from blockchain concepts to individual coins to the entire world of NFTs which has arisen in recent years. Finally, there are myriad firmly held beliefs about cryptocurrencies, ranging from outright vitriol to religious fervor. Cryptocurrency tweets comprise a complex problem space which can be sliced in several ways. It is not uncommon, for example, for a single tweet of less than 20 words to have negative sentiment about one currency while praising another. Therefore, cryptocurrencies tweets present a rich landscape for NLP research and models.

I had two main goals in this project. The primary goal was to gain experience and learn the process of collection, classifying, and preparing a dataset for machine learning. The secondary goal is to create a neural network that could differentiate spam cryptocurrency tweets from other cryptocurrency tweets. In pursuit of these goals, I discovered the problem of few-shot learning and several proposed solutions to the problem.

2 Related works

In researching this project, I studied many approaches, a few of which I will mention here. An exciting approach was proposed recently by Kim et al. [2022] called Lexicon Guided Self Training. This approach trains on the labeled set to create a lexicon for each class. This lexicon is used to generate pseudo labels for unlabeled data and provide a scheme to calculate a confidence level for each generated label. Ultimately, this approach requires large teacher/student models and I did not have the resources to continue. However, work done while researching this approach became useful when analyzing my models.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

A big influence on the choices I made came by Hsu et al. [2018] and Lison et al. [2021]. The former introduced me to meta-learning objectives and tasks, which is how I built the pre-training pipeline as described in Section 4.2. The latter describes `skweak`, a weak classification library for python. While I did not directly use this library, the heuristic labelling techniques described in their paper informed me on how to make my own secondary labels when building tasks for my own meta training tasks.

Pattern-Exploitative Training was proposed recently by Schick and Schütze [2020]. While I did not implement their entire paper, I use their concept of using cloze questions to reframe classification tasks as NLP problems. This approach is presented in detail in Section 4.3.

3 Dataset design

3.1 Data discovery and collection

I collected cryptocurrency tweets over the course of April 15th - 24th, 2022. The code for the data pipeline can be found at `https://gitlab.com/dhuck/arexion`. Tweets were collected using the `tweepy` package alternating between two different methods. The first method filtered incoming tweets by supplying related keywords to the Twitter API. These keywords were all related to cryptocurrency and specified by the Author. The second method relied on filtering by usernames of Twitter users identified as cryptocurrency influencers. The process of discovery of these usernames was done partially by the author and Eli Good, who also significantly helped during classification. These methods alternated between set intervals throughout the time of collection.

Large amounts of the incoming data were repeat messages. With the goal of providing as diverse of a dataset as possible, I applied several heuristics to limit which tweets were added to the dataset. First, retweets were not allowed in the database by checking for the `retweeted_status` in the API's response and the phrase 'rt @' in the tweet text. Second, I created an ignore list, which employed simple regex substring matching to remove common messages. Some examples of ignored messages are 'price alert!', 'complete the puzzle', or '#coinhunterworld'. This removed a number of tweets that were generated by bots or people playing NFT-based games. The cleaning process removed all hashtags and user mentions. A simple query was run at regular intervals to remove all duplicates using full string matching on the cleaned text. To be removed from the database, a cleaned tweet needed to be present at least 5 times in the database as a means to keep the duplicate removal process from being too aggressive. During peak times, this last heuristic could remove up to 500 tweets every minute. The reason of this mass removal was to limit certain tweets from becoming overwhelmingly represented in the database.

Over the course of the collection time frame, roughly 2.3 million tweets were collected and stored.

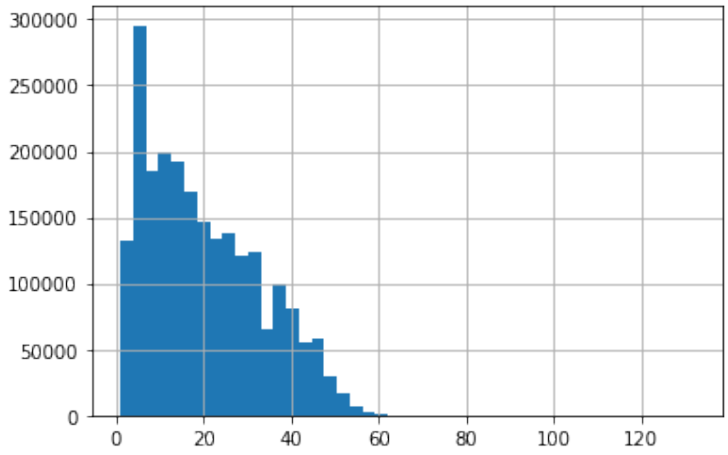
3.2 Data labelling

To label the data, a simple web application was built to display a tweet and prompt a user to classify individual tweets as one of six classes. The class choices are spam, scam, positive, neutral, negative, or off-topic. While the primary goal was to detect spam, I wanted to explore performing sentiment analysis on non-spam tweets. Furthermore, within a day of classifying, it became apparent that many tweets were off-topic, presenting a tertiary classification. The web application can be viewed at `http://vaporwav.in`, and I urge the reader to help classify some more tweets at their leisure. By the time training and model development began in earnest, only 3,600 tweets had been classified. While classification was open to many people, the majority of classification was done by myself and Eli Good, as mentioned above.

3.3 Training and test data

Both the labeled and unlabeled training data are available for download using an API in the provided notebooks. When using the labeled data, at least 10% of the data is kept back as test data. This is done at runtime in the notebooks. To prevent models from seeing training data in any of the test data, all saved and loaded checkpoints are from pre-training or objectives on the unlabeled data. Depending on the training set, the test data may end up being a significant portion of the labeled data. It is traditional in few-shot classification problems to benchmark using only n examples of each class,

Figure 1: Histogram of tweet lengths. Almost all of the tweets are less than 60 words long, keeping token strings short.



where n is typically less than 50. In these cases, the remainder of the training data is kept as the test set.

Across both the labeled and unlabeled sets, I removed all user mentions. The reason for this was two-fold. First, user mentions are not counted against tweet length, so tweets belonging in reply streams may have dozens of user mentions but only a few words of meaningful text. Second, I did not want to encourage the model to attach classifications to any one user. For example, a certain prominent billionaire's username is popular in throughout the dataset, and over exposure to this handle, among others, could influence the model in bad ways. Additionally, it is my opinion usernames should be stripped from this data for privacy concerns.

Since I am using BERT-based transformers as the root of the model, most of the word tokenization is handled by huggingface `PretrainedTokenizers`. Early experiments and analysis showed this to not work as well as I had hoped. As a result, I created a vocabulary file with all words and emoji which occur more than 500 times in the dataset. This resulted in better performance in the baseline and more meaningful analysis when investigating output attentions.

Figure 1 shows the distribution of tweet lengths. The vast majority of tweets are less than 40 words long. For data alignment reason, I chose a max length of 64, which captures all but the long tail of tweets. Originally, I was using the longest tweet as my maximum length for tokenization. Switching to 64 words per token string improved training times significantly.

3.4 Unlabeled data

Unlabeled data is used in pre-training objectives depending on the models as described in Sections 4.2 and 4.3.

4 Models

Few-shot classification is a problem with an intense amount of research currently underway. Many of the approaches I investigated were beyond my reach due to a lack of computational and time resources. Models which could be trained on available hardware took a long time and made iteration difficult. As a result, I investigated two approaches to the problem–Transfer Learning and Pattern-Exploitative Training.

4.1 Random model

To create a baseline to compare other models two, I created a random model, which guesses at answers at random with no training. This was implemented with a simple `numpy` random dice-roll for each item in the test sets.

4.2 RoBERTa classifier and transfer learning

The first model is a simple RoBERTa-LSTM Classifier. The RoBERTa transformer is fed into a bidirectional LSTM before a fully connected layer with dropout and ReLU into an output layer with the number of classes as the output. The learning rate is set initially at $8 \cdot 10^{-6}$ with a StepLR scheduler with $\gamma = 0.25$ at every four steps. Those learning rate was chosen since $1 \cdot 10^{-6}$ was common in the literature in few-shot learning. Setting it to this rate initially, however, increased training time significantly. Setting it slightly higher with a scheduler improved training time, while keeping the learning rate low enough to converge well. For the optimizer, I selected the AdamW optimizer over the Adam due to the differing implementation of weight decay [Loshchilov and Hutter, 2017]. The final version of the model does not use weight decay as regularization, however, and AdamW is left in as an artifact. Outside of dropout in the fully connected layers, early stopping is the only regularization utilized.

I chose an LSTM between the fully connected layers and the RoBERTa transformer because of the meta learning tasks. My theory was the RoBERTa transformer would learn the underlying data, while the LSTM would learn sequence information from each episode, most importantly the final training on labeled data.

While performing transfer learning, unlabeled data is used to create new labeled data for the classifier to pre-train on. This method was inspired by conversations with Dr. Huth, as well as the `skweak` [Lison et al., 2021] library. While I attempted a number of partitions of the dataset, I settled on three techniques: keyword labelling, emoji prediction, and text ablation. Each time a partition of the data with these methods were made, all corresponding tweets were removed from the unlabeled data set, exposing the model to several unique tweets.

In keyword labelling, I took keywords used for gathering tweets from the Twitter API and grouped tweets by first mention of each keyword. New datasets were created by removing every mention of the top n keywords from the unlabeled dataset, and classifying each tweet with the appropriate keyword. To keep training times down to a minimum, I took 2,500 examples for each label. Upon repeat partitions using this method, common keywords remained across all partitions, but new keywords were introduced with some regularity. Increasing the number of examples per keyword raised the number of keywords in the final set.

Emoji prediction followed a very similar pattern to the keyword labelling. I took the subset of all tweets with emoji, chose the top n emoji in the subset, and labeled data with that emoji. To my surprise, there were not many tweets with emoji, so these partitions ended up being fairly small in comparison to keyword labelling. The upshot to this problem was most partitions of emoji data had distinct labels. I aimed for 1,000 examples of each emoji, but several emoji did not have that many examples.

Text ablation takes it cue from methods used in transfer learning for images in CNNs. In this approach, text was arranged into one of four categories–unmodified, reversed, cut, and shuffled. The model was then asked to classify what had happened to each tweet. A bug was introduced in this technique which resulted in low accuracy, and it was therefore dropped in the results for this paper.

When pre-training, the datasets are partitioned and fed into the RoBERTa classifier as meta-learning objectives, one at a time in a series of episodes. At the end of each episode, the fully connected layers are removed and replaced with new fully connected layers. The optimizer is reset at the end of each episode, and the cycle repeats. By exposing the transformer to large section of the unlabeled data, my hypothesis is the model will perform significantly better than the random and classifier baselines described above.

4.3 Pattern-Exploitative Training

Pattern-**E**xploitative **T**raining (PET) is a method of classification, which reframes the classification task as a natural language generation task. It uses cloze questions to have the model predict a word over a masked token which acts as a stand in for the class of the item. The full model, as described by Schick and Schütze [2020], uses several generations of this process and predicts soft labels for unlabeled data, resulting in improved performance on few-shot learning tasks. As noted above, I did not have time to implement the full model or soft labelling of the unlabeled data. Instead, I borrowed the core idea of cloze questions and used it to classify labeled tweets.

Table 1: Mapping class names to cloze question answers. On the left are the classes that were used during classification, on the right are more natural language answers fill in the cloze sentence, "This is ____."

Class	Chosen Cloze Word
Spam	Nonsense
Scam	Untrustworthy
Positive	Wonderful
Neutral	Boring
Negative	Horrible
Off-Topic	Unrelated

For each class of tweet, I came up with words which I believe captured the essence of each class in more natural language. These are summarized in Table 1. I believe the choice of words for the cloze questions could have some effect on the accuracy of this model. Picking words which are a part of the vernacular of the domain may result in higher accuracy, but this hypothesis is left untested for now. For each tweet in the labeled dataset, the phrase "This is ____" was appended to the end of each tweet, where ____ represents a masked token. Each one of these words was added to the tokenizer's vocabulary as to make the loss calculation easier. The model was then trained to predict which one of the chosen cloze words belong in the masked position. This was handled by using a loss function as detailed in the original PET paper.

$$\text{Loss} = (1 - \alpha) \cdot L_{\text{mask}} + \alpha \cdot L_{\text{lm}} \tag{1}$$

In Equation 1, both losses L_* are Cross Entropy Loss functions. L_{mask} is the loss of the prediction of the masked word, and L_{lm} is the overall language model loss. α is a constant which sets the weight of each of these losses into the final loss. I followed the guide of the original paper and fixed $\alpha = 1 \cdot 10^{-4}$. For each example, the proper index of the logit prediction is supplied to the loss function, and the total loss is calculated.

When testing or generating inferences from this model, choosing the predictions set is key to good results. When testing, instead of choosing the highest probability result, I consider only the indices of the cloze words in the logit prediction set from the transformer. I perform a softmax on these predictions before choosing the proper prediction. Without doing this, the model would sometimes pick on other cues within the tweet and predict words we do not wish to consider for this problem.

5 Experiments

The primary experiment of this project was to investigate how well different models work on the same data set. My ability to do this was limited by both computational and time resources. I settled on the two models presented in this paper due to their potential in solving this particular problem, their ease of implementation, and their relatively low computational cost compared to other few-shot techniques. My hypothesis was the RoBERTa-LSTM Classifier would perform better on the full dataset, but would not perform well on the few-shot classification task.

The secondary experiment was to investigate the effects of pre-training on the accuracy of each model. My hypothesis is that pre-training would increase the performance of both models on the few-shot classification task, and the general classification task. For the RoBERTa-LSTM Classifier, I pre-trained the model on subsets of the unlabeled data as described in Section 4.2. I took ten sets of both generated emoji and keyword datasets, shuffled them, and trained on each until convergence. After each episode, the LSTM and fully connected layers were removed and replaced with randomized weights. This training took roughly ten hours to complete. For the PET-based model, I simply sampled 100,000 unlabeled tweets and pre-trained the RoBERTa Transformer as a language model, which took roughly four hours. As a baseline, I ran the random model, described in Section 4.1 100 times. This random model had an overall average accuracy of 16.04% with a standard deviation of 10.94%.

Table 2: Results from the RoBERTa-LSTM Classifier. The top table is with no pretraining and the second is with pretraining. Each number along the top represents the number of examples of each class shown to the model during training. Each training run was exposed to the same test set.

	10	20	30	40	50	60	70	Full
spam	60.22%	99.45%	100.%	46.41%	87.85%	41.44%	0.00%	85.64%
scam	0.00%	0.00%	0.00%	0.00%	36.36%	0.00%	0.00%	0.00%
positive	55.68%	0.00%	0.00%	42.05%	0.00%	23.86%	0.00%	39.77%
neutral	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	14.49%	47.83%
negative	4.35%	0.87%	0.00%	0.00%	0.00%	72.17%	0.00%	16.52%
off-topic	0.00%	0.00%	0.00%	50.00%	0.00%	0.00%	86.11%	0.00%
Overall	32.60%	36.20%	36.20%	27.80%	32.60%	35.80%	8.20%	48.40%

	10	20	30	40	50	60	70	Full
spam	0.55%	11.05%	34.81%	56.91%	17.13%	43.09%	58.56%	60.22%
scam	100%	100%	63.64%	9.09%	81.82%	45.45%	9.09%	27.27%
positive	9.09%	3.41%	0.00%	0.00%	28.41%	28.41%	0.00%	27.27%
neutral	2.90%	2.90%	30.43%	88.41%	59.42%	59.42%	75.36%	36.23%
negative	0.00%	0.00%	33.91%	1.74%	0.00%	1.74%	0.00%	25.22%
off-topic	11.11%	5.56%	22.22%	8.33%	0.00%	11.11%	50.00%	25.00%
Overall	5.20%	7.60%	27.60%	34.00%	21.20%	31.00%	35.40%	39.80%

Table 3: Results for the PET-based model. The top half is the results with no pre-training, and the bottom is with pre-training. Overall, these models have better accuracy for individual classes in the dataset. The pre-trained model, specifically, does better across classes, with no 0% runs when more than 40 examples of each class are shown.

	10	20	30	40	50	60	70	Full
spam	27.74%	41.94%	21.29%	36.13%	38.71%	36.77%	48.39%	69.03%
scam	0.00%	0.00%	0.00%	10.00%	0.00%	0.00%	20.00%	0.00%
positive	57.14%	57.14%	31.09%	9.24%	50.42%	56.30%	35.29%	47.06%
negative	5.00%	0.00%	15.00%	23.33%	0.00%	1.67%	10.00%	41.67%
neutral	12.90%	27.42%	14.52%	25.81%	14.52%	4.03%	8.87%	33.87%
off-topic	15.62%	3.12%	31.25%	18.75%	28.12%	34.38%	34.38%	3.12%
Overall	27.00%	33.60%	21.40%	24.00%	29.40%	28.20%	29.40%	46.20%

	10	20	30	40	50	60	70	Full
spam	56.77%	39.35%	58.06%	56.77%	44.52%	56.77%	44.52%	65.81%
scam	20.00%	60.00%	0.00%	20.00%	50.00%	30.00%	30.00%	0.00%
positive	5.88%	12.61%	0.84%	17.65%	12.61%	36.13%	30.25%	33.61%
negative	66.67%	38.33%	51.67%	28.33%	51.67%	26.67%	43.33%	48.33%
neutral	5.65%	26.61%	1.61%	28.23%	20.97%	23.39%	21.77%	54.03%
off-topic	37.50%	18.75%	46.88%	37.50%	40.62%	34.38%	65.62%	3.12%
Overall	31.20%	28.80%	27.80%	35.00%	31.80%	38.00%	36.40%	47.80%

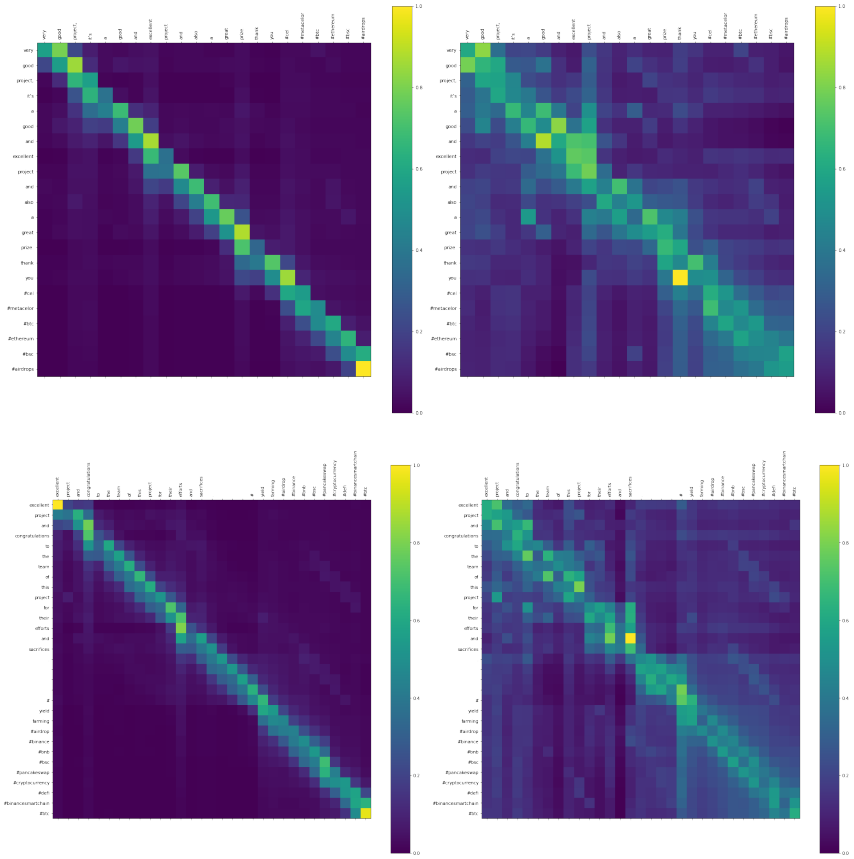


Figure 2: Attention matrices of the two models for a tweet both models predicted as being spam. On the left are the attention matrices for the PET-based model, which are much tighter. The right attention matrices are for the RoBERTa-LSTM Classifier and are much fuzzier.

Each model was tested on few-shot datasets. Datasets were chosen at random with 10, 20, 30, 40, 50, 60, and 70 examples of each class. The model was then tested on a withheld test set to calculate the final accuracy. Due to some technical difficulty in sharing data between Google Colab and my home computer, the test set for the PET-Based models was different from the RoBERTa-LSTM test set. Therefore, the results between the two architectures are not entirely conclusive, but I believe they still generalize well. Test sets were consistent between the pre-trained and nonpre-trained models of the respective models. Due to the computational complexity of the models, the results presented are from a single run for each model, rather than an average of several runs.

5.1 Results

The RoBERTa-LSTM with no pre-training had the best overall accuracy, as well as the best accuracy in spam detection. Unsurprisingly, it was not able to generalize well when trained on the few shot training sets. The pre-trained version of the RoBERTa-LSTM did not perform as well on the full dataset, which was a little shocking. It did, however, have a much easier time with the few-shot datasets, with a more even spread over its accuracies. Most of the accuracies in the 50 and 60-shot tests were well above the random threshold. In the smaller datasets, however, the accuracies fell well within the random guessing range.

While the overall accuracy is a little lower, I would argue the PET-Based models performed better in their experiments. Both the pre-trained and non-pre-trained variants have a low number of 0% accuracies in their table. The pre-trained model in particular shows impressive comparative performance in the few-shot training datasets, confirming my hypotheses above. While they are not

as confident in detecting spam, they do have overall accuracies comparable to the RoBERTa-LSTM Classifier.

5.2 Analysis

As a final analytical experiment, I took the pre-trained models trained on the full dataset and plotted the truncated attention matrices. The RoBERTa transformer has 12 attention heads and each one of these matrices is the averaged output of all 12. The matrices are normalized, so they scale from 0 to 1 across their respective minimum and maximum values. Two examples of these matrices can be seen in Figure 5. Both examples use tweets which were correctly classified as spam by both models.

In each pair of matrices I investigate, the PET-based model exhibited a tighter diagonal than the Bert-LSTM matrices. I believe the reasons for this are one of two possibilities. First, the PET-based model was pre-trained on two to three times as many tweets as the RoBERTa-LSTM model. Due to extra training data, the attention weights have become finer-tuned than the RoBERTa-LSTM model. Second, and more likely, the PET-based model consists of *only* the RoBERTa transformer, whereas RoBERTa-LSTM model has three additional layers on top of it. There are more parameters by an order of magnitude on the latter model, placing less importance on the attention heads in making a prediction. Back propagation is distributed across the transformer and the output layers.

6 Conclusion

The key takeaway from this project was the importance of data and quality datasets. One cannot code their way out of bad data. Classifications in the labeled dataset are not consistent across similar tweets. Since the amount of labeled data is so small, the networks have a difficult time learning features to differentiate the classes. Moreover, while classifying a tweet as 'positive' or 'spam' are two different labels, the number of labels I chose further complicated the problem. Being more rigorous in the classification test, or removing duplicate tweets across separate labels would have greatly improved the efficacy of my models.

I plan on continuing research into this problem. Next steps include fixing the issues outlined in the previous paragraph. Moreover, I plan on combining the ideas from Kim et al. [2022] and Schick and Schütze [2020]. I believe the PET approach could be improved by introducing a manner to measure the confidence of soft labels generated by the model. The Lexicon-guided Self Training provides an interesting way to build confidence around these labels.

References

Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. *arXiv preprint arXiv:1810.02334*, 2018.

Hazel Kim, Jaeman Son, and Yo-Sub Han. Lst: Lexicon-guided self-training for few-shot text classification. *arXiv preprint arXiv:2202.02566*, 2022.

Pierre Lison, Jeremy Barnes, and Aliaksandr Hubin. skweak: Weak supervision made easy for nlp, 2021.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Timo Schick and Hinrich Schütze. Exploiting cloze questions for few shot text classification and natural language inference. *arXiv preprint arXiv:2001.07676*, 2020.