Robust Text Classifier on Test-Time Budgets

Md Rizwan Parvez
University of California Los Angeles
rizwan@cs.ucla.edu

Tolga Bolukbasi
Boston University
tolgab@bu.edu

Kai-Wei Chang
University of California Los Angeles
kwchang@cs.ucla.edu

Venkatesh Saligrama
Boston University
srv@bu.edu

Abstract

We design a generic framework for learning a robust text classification model that achieves high accuracy under different selection budgets (a.k.a selection rates) at test-time. We take a different approach from existing methods and learn to dynamically filter a large fraction of unimportant words by a low-complexity selector such that any high-complexity classifier only needs to process a small fraction of text, relevant for the target task. To this end, we propose a data aggregation method for training the classifier, allowing it to achieve competitive performance on fractured sentences. On four benchmark text classification tasks, we demonstrate that the framework gains consistent speedup with little degradation in accuracy on various selection budgets.

1 Introduction

Recent advances in deep neural networks (DNNs) have achieved high accuracy on many text classification tasks. These approaches process the entire text and encode words and phrases in order to perform target tasks. While these models realize high accuracy, the computational time scales linearly with the size of the documents, which can be slow for a long document. In this context, various approaches based on modifying the RNN or LSTM architecture have been proposed to speed up the process (Seo et al., 2018; Yu et al., 2017). However, the processing in these models is still fundamentally sequential and needs to operate on the whole document which limits the computational gain. In contrast to previous approaches, we propose a novel framework for efficient text classification on long documents that mitigates sequential processing. The framework consists of a *selector* and a *classifier*. Given a selection budget as input, the *selector* performs a coarse one-shot selection deleting unimportant words and pass the remainder to the *classifier*. The *classifier* then takes the sentence fragments as an input and performs the target task. Figure 1 illustrates the procedure. This framework is general and agnostic to the architecture of the downstream *classifier* (e.g., RNN, CNN, Transformer).

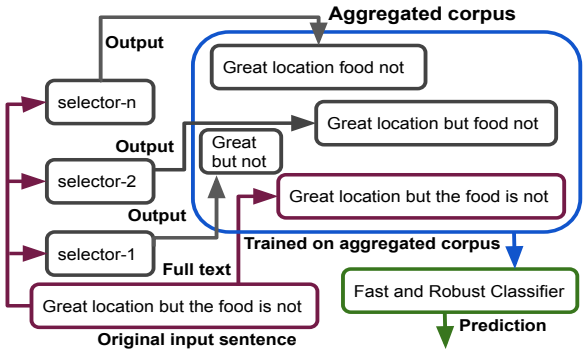


Figure 1: Our proposed framework. Given a selection rate, a *selector* is designed to select relevant words and pass them to the *classifier*. To make the classifier robust against fractured sentences, we aggregate outputs from different *selectors* and train the *classifier* on the aggregated corpus.

However, three challenges arise. First, to build a computationally inexpensive system, the *selector* must have negligible overhead. We adopt two effective yet simple architectures to design *selectors* based on word embeddings and bag-of-words. Second, training multiple distinct models for different budgets is unfeasible in practice, especially when model size is large. Hence, our goal is to learn a single *classifier* that can adapt to the output of any *selector* operating at any budget. Consequently, this *classifier* must be robust so that it can achieve consistent performance with different budgets. Third, the input to the *classifier* in our framework is a sequence of fractured sentences which is incompatible with a standard *classifier* that trained on the full texts, causing its performance degrades significantly. One potential but unfeasible solution is to train the *classifier* with a diverse collection of sentence fragments which is

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 1167–1172,
Hong Kong, China, November 3–7, 2019.

combinatorially numerous. Another approach is to randomly blank out text (a.k.a. blanking-noise), leads to marginalized feature distortion (Maaten et al., 2013) but this also leads to poor accuracy as DNNs leverage word combinations, sentence structure, which this approach does not account for. To mitigate this problem, we propose a data aggregation framework that augments the training corpus with outputs from *selector*s at different budget levels. By training the *classifier* on the aggregated *structured* blank-out text, the *classifier* learns to fuse fragmented sentences into a feature representation that mirrors the representation obtained on full sentences and thus realizes high-accuracy. We evaluate our approach through comprehensive experiments on real-world datasets. [1]

2 Related Work

Several approaches have been proposed to speed up the DNN in test time (Wu et al., 2017; Choi et al., 2017). LSTM-jump (Yu et al., 2017) learns to completely skip words deemed to be irrelevant and skim-RNN (Seo et al., 2018) uses a low-complexity LSTM to skim words rather than skipping. Another version of LSTM-jump, LSTM-shuttle (Fu and Ma, 2018) first skips a number of words, then goes backward to recover lost information by reading some words skipped before. All these approaches require to modify the architecture of the underlying classifier and cannot easily extend to another architecture. In contrast, we adopt existing *classifier* architectures (e.g., LSTM, BCN (McCann et al., 2017)) and propose a meta-learning algorithm to train the model. Our framework is generic and a *classifier* can be viewed as a black-box. Similar to us, Lei et al. (2016) propose a *selector-classifier* framework to find text snippets as justification for text classification but their *selector* and *classifier* have similar complexity and require similar processing times; therefore, it is not suitable for computation gain. Various feature selection approaches (Chandrashekar and Sahin, 2014) have been discussed in literature. For example, removing predefined stop-words (see Appendix A), attention based models (Bahdanau et al., 2015; Luong et al., 2015), feature subspace selection methods (e.g., PCA), and applying the L1 regularization (e.g., Lasso (Tibshirani, 1996) or Group Lasso (Faruqui et al., 2015), BLasso (Gao et al., 2007)). However, these approaches either cannot obtain sparse features or cannot straightforwardly be applied to speed up a DNN *classifier*. Different from ours, Viola and Jones (2001); Trapeznikov and Saligrama (2013); Karayev et al. (2013); Xu et al. (2013); Kusner et al. (2014); Bengio et al. (2016); Leroux et al. (2017); Zhu et al. (2019); Nan and Saligrama (2017); Bolukbasi et al. (2017) focus on gating various components of existing networks. Finally, aggregating data or models has been studied under different contexts (e.g., in context of reinforcement learning (Ross et al., 2011), Bagging models (Breiman, 1996), etc.) while we aggregate the data output from *selectors* instead of models.

[1] Our source code is available at: https://github.com/uclanlp/Fast-and-Robust-Text-Classification

3 Classification on a Test-Time Budget

Our goal is to build a robust *classifier* along with a suite of *selectors* to achieve good performance with consistent speedup under different selection budgets at test-time. Formally, a *classifier* $C(\hat{x})$ takes a word sequence $\hat{x}$ and predicts the corresponding output label y, and a *selector* $S_b(x)$ with selection budget b takes an input word sequence $x = \{w_1, w_2, \ldots, w_N\}$ and generates a binary sequence $S_b(x) = \{z_{w_1}, z_{w_2}, \ldots, z_{w_N}\}$ where $z_{w_k} \in \{0, 1\}$ represents if the corresponding word w_k is selected or not. We denote the sub-sequence of words generated after filtering by the *selector* as $I\big(x, S_b(x)\big) = \{w_k : z_{w_k} = 1, \forall w_k \in x\}$. We aim to train a *classifier* C and the *selector* S_b such that $I\big(x, S_b(x)\big)$ is sufficient to make accurate prediction on the output label (i.e., $C\Big(I\big(x, S_b(x)\big)\Big) \approx C(x)$). The selection budget (a.k.a selection rate) b is controlled by the hyper-parameters of the *selector*. Higher budget often leads to higher accuracy and longer test time.

3.1 Learning a *Selector*

We propose two simple but efficient *selector*s. **Word Embedding (WE) *selector*.** We consider a parsimonious word-selector using word embeddings (e.g., GloVe (Pennington et al., 2014)) as features to predict important words. We assume the informative words can be identified independently and model the probability that a word w_k is selected by $P(z_{w_k} = 1|w_k) = \sigma(\theta_S^T \vec{w}_k)$, where θ_S is the model parameters of the *selector* S_b, $\vec{w}_k$ is the corresponding word vector, and σ is the sigmoid function. As we do not have explicit anno-

tations about which words are important, we train the *selector* S_b along with a *classifier* C in an end-to-end manner following Lei et al. (2016), and an L1-regularizer is added to control the sparsity (i.e., selection budget) of $S_b(x)$.

Bag-of-Words *selector*. We also consider using an L1-regularized linear model (Zou and Hastie, 2005; Ng, 2004; Yuan et al., 2010) with bag-of-words features to identify important words. In the bag-of-words model, for each document x, we construct a feature vector $\vec{x} \in \{0,1\}^{|V|}$, where $|V|$ is the size of the vocabulary. Each element of the feature vector $\vec{x}_w$ represents if a specific word w appearing in the document x. Given a training set $\mathcal{X}$, the linear model optimizes the L1-regularized task loss. For example, in case of a binary classification task (output label $y \in \{1, -1\}$),

$$J(x_t, y_t) = \log\left(1 + \exp(-y_t \theta^T \vec{x_t})\right)$$

$$\theta^* = \arg\min_\theta \sum_{(x_t,y_t)\in\mathcal{X}} J(x_t, y_t) + \frac{1}{b}\|\theta\|_1,$$

where $\theta \in R^{|V|}$ is a weight vector to be learned, θ_w corresponds to word $w \in V$, and b is a hyper-parameter controlling the sparsity of θ^* (i.e., selection budget). The lower the budget b is, the sparser the selection is. Based on the optimal θ^*, we construct a *selector* that picks word w if the corresponding θ^*_w is non-zero. Formally, the bag-of-words *selector* outputs $S_b(x) = \{\delta(\theta_w \neq 0) : w \in x\}$, where δ is an indicator function.

3.2 The Data Aggregation Framework

In order to learn to fuse fragmented sentences into a robust feature representation, we propose to train the *classifier* on the aggregated corpus of structured blank-out texts.

Given a set of training data $\mathcal{X} = \{(x_1, y_1), .., (x_t, y_t), .., (x_m, y_m)\}$, we assume we have a set of selectors $\mathcal{S} = \{S_b\}$ with different budget levels trained by the framework discussed in Section 3.1. To generate an aggregated corpus, we first apply each *selector* $S_b \in \mathcal{S}$ on the training set, and generate corresponding blank-out corpus $\mathcal{I}(\mathcal{X}, S_b) = \{I(x_t, S_b(x_t)), \forall x_t \in \mathcal{X}\}$. Then, we create a new corpus by aggregating the blank-out corpora: $\mathcal{T} = \bigcup_{S_b\in\mathcal{S}} \mathcal{I}(\mathcal{X}, S_b)$.[2] Finally, we train the *classifier* $C_\mathcal{T}$ on the aggregated corpus $\mathcal{T}$. As $C_\mathcal{T}$ is trained on documents with distortions, it learns to make predictions with different budget levels. The training procedure is summarized in Algorithm 1. In the following, we discuss two extensions of our data aggregation framework.

Algorithm 1: **Data Aggregated Training Schema**

Input: Training corpus $\mathcal{X}$, a set of *selectors* with different budget levels $\mathcal{S} = \{S_b\}$, *classifier* class C

Output: A robust *classifier* $C_\mathcal{T}$

```
Initialize the aggregated corpus: T ← X
for S_b ∈ S do
    S_b ← Train a selector S_b ∈ S with budget level b on X
    Generate a blank-out dataset I(X, S_b)
    Aggregate data: T ← T ∪ I(X, S_b)
C_T ← Train a classifier C on T
return C_T
```

First, the blank-out data can be generated from different classes of *selectors* with different features or architectures. Second, the blank-out and selection can be done in phrase or sentence level. Specifically, if phrase boundaries are provided, a phrase-level aggregation can avoid a *selector* from breaking compound nouns or meaningful phrases (e.g., "Los Angeles", "not bad"). Similarly, for multi-sentenced documents, we can enforce the *selector* to pick a whole sentence if any word in the sentence is selected.

4 Experiments

To evaluate the proposed approach, we consider four benchmark datasets: *SST-2* (Socher et al., 2013), *IMDB* (Maas et al., 2011), *AGNews* (Zhang et al., 2015), and *Yelp* (Conneau et al., 2016) and two widely used architectures for classification: *LSTM*, and *BCN* (McCann et al., 2017). The statistics of the datasets are summarized in Table 2. We evaluate the computation gain of models in terms of overall test-time, and the performance in terms of accuracy. We follow Seo et al. (2018) to estimate the test-time of models on CPU[3] and exclude the time for data loading.

In our approach, we train a *classifier* with both WE and bag-of-words *selector*s with 6 selection budgets[4] {50%, 60%, ..., 100%} by the word-

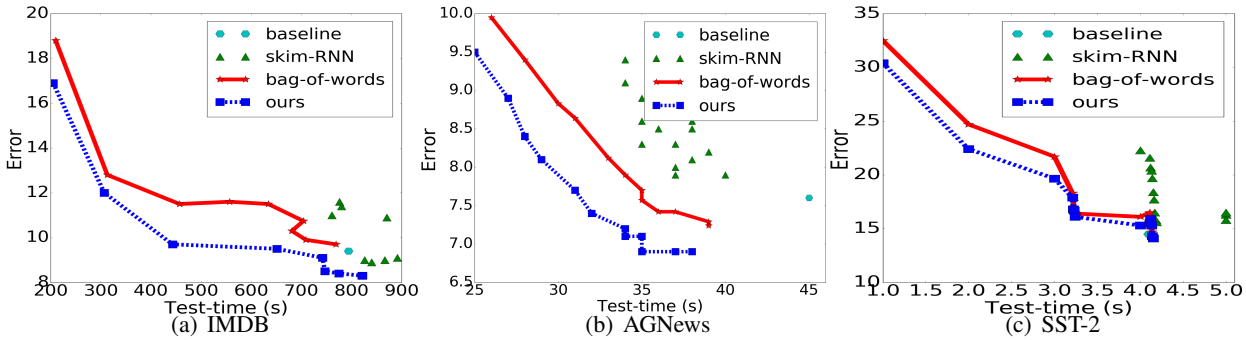
[2] Note that, the union operation is used just to aggregate the train instances which does not hinder the model training (e.g., discrete variables).

[3] Machine specification is in Appendix C.

[4] For the very large Yelp dataset, 3 selection budgets {50%, 60%, 70%} are used.

Model	SST-2				IMDB				AGNews				Yelp			
	acc.	selection(%)	time	speedup	acc.	selection(%)	time	speedup	acc.	selection(%)	time	speedup	acc.	selection(%)	time	speedup
Baseline	**85.7**	100	9	1x	91.0	100	1546	1x	92.3	100	59	1x	**66.5**	100	3487	1x
Bag-of-Words	78.8	75	5.34	1.7x	91.5	91	1258	**1.2x**	92.9	97	48	1.2x	59.7	55	2325	**1.6x**
Our framework	82.6	65	4.6	**2x**	92.0	91	1297	**1.2x**	93.1	91	46	**1.3x**	64.8	55	2179	**1.6x**
	85.3	0	9	1x	**92.1**	0	1618	1x	**93.2**	0	57	1x	66.3	0	3448	1x

Table 1: Accuracy and speedup on the test datasets. Test-times are measured in seconds. The speedup rate is calculated as the running time of a model divided by the running time of the corresponding baseline. For our framework, top row denotes the best speedup and the bottom row denotes the best test accuracy achieved. Overall best accuracies and best speedups are boldfaced. Our framework achieves accuracies better than baseline with a speedup of **1.2x** and **1.3x** on IMDB, and AGNews respectively. With same or higher speedup, our accuracies are much better than Bag-of-Words.



(a) IMDB (b) AGNews (c) SST-2

Figure 2: Performance under different test-times on IMDB, AGNews, and SST-2. All the approaches use the same LSTM model as the back-end. Bag-of-Words model and our framework have the same bag-of-words *selector* cascaded with this LSTM *classifier* trained on the original training corpus and aggregated corpus, respectively. Our model (blue dashed line) significantly outperform others for any test-time budget. Also its performance is robust, while results of skim-RNN is inconsistent with different budget levels.

Dataset	#class	Vocabulary	Size (Train/Valid/Test)	Avg. Len
SST	2	13,750	6,920/872/1,821	19
IMDB	2	61,046	21,143/3,857/25,000	240
AGNews	4	60,088	101,851/18,149/7,600	43
Yelp	5	1,001,485	600k/50k/50k	149

Table 2: Dataset statistics.

level data aggregation framework. We evaluate the computation gain of the proposed method through a comparative study of its performance under different test-times by varying the selection budgets[5] in comparison to the following approaches: (1) *Baseline*: the original *classifier* (i.e., no *selector*, no data aggregation) (2) *skim-RNN*: we train a skim-RNN model and vary the amount of text to skim (i.e., test-time) by tuning θ parameter as in Seo et al. (2018). (3) Bag-of-Words: filtering words by the bag-of-words *selector* and feeding the fragments of sentences to the original *classifier* (i.e., no data aggregation). This approach serves as a good baseline and has been considered in the context of linear models (e.g., Chang and Lin (2008)). For a fair comparison, we implement all approaches upon the same framework using AllenNLP library[6], including a re-implementation of the existing state-of-art speedup framework skim-RNN (Seo et al., 2018)[7]. As skim-RNN is designed specifically for accelerating the LSTM model, we only compare with skim-RNN using LSTM *classifier*. Each corresponding model is selected by tuning parameters on validation data. The model is then frozen and evaluated on test-data for different selection budgets.

Figure 2 demonstrates the trade-off between the performance, and the test-time for each setting. Overall, we expect the error to decrease with a larger test-time budget. From Figure 2, on all of the IMDB, AGNews, and SST-2 datasets, LSTM *classifier* trained with our proposed data aggregation not only achieves the lowest error curve but also the results are robust and consistent. That is our approach achieves higher performance across different test-time budgets and its performance is a predictable monotonic function of the test-time budget. However, the performance of skim-RNN exhibits inconsistency for different budgets. As a matter of fact, for multiple budgets, none of the skim-RNN, and LSTM-jump address the prob-

[5] In Appendix B, we discuss how to vary these budgets.

[6] https://allennlp.org/

[7] The official skim-RNN implementation is not released.

World News	.. **plant** searched. Kansai Electric Power's **nuclear** power **plant** in Fukui .. **was searched by police** Saturday ..
Business	Telecom Austria taps the Bulgarian **market**. **Telecom** Austria, Austrias **largest** telecoms operator, obtained ..
Sci/Tech	.. Reuters - **Software security** companies **and** handset makers, including Finland's **Nokia** (NOK1V.HE), **are** ..

Table 3: Examples of the WE *selector* output on AGNews. Bold words are selected.

lem of different word distribution between training and testing. Therefore, similar to skim-RNN, we anticipate that the behavior of LSTM-jump will be inconsistent as well[8]. Additionally, since LSTM-jump has already been shown to be outperformed by skim-RNN, we do not further compare with it. Next, we show that our framework is generic and can incorporate with other different *classifier*s, such as BCN (see Table 1).[9] When phrase boundary information is available, our model can further achieve **86.7** in accuracy with **1.7x** speedup for BCN on SST-2 dataset by using phrase-level data aggregation. Finally, one more advantage of the proposed framework is that the output of the *selector* is interpretable. In Table 3, we present that our framework correctly selects words such as "Nokia", "telecom", and phrases such as "searched by police", "software security" and filters out words like "Aug.", "users" and "products".

Note that nevertheless we focus on efficient inference, empirically our method is no more complex than the baseline during training. Despite the number of training instances increases, and so does the training time for each epoch, the number of epochs we require for obtaining a good model is usually smaller. For example, on the Yelp corpus, we only need 3 epochs to train a BCN *classifier* on the aggregated corpus generated by using 3 different *selector*s, while training on the original corpus requires 10 epochs.

5 Conclusion

We present a framework to learn a robust *classifier* under test-time constraints. We demonstrate that the proposed *selector*s effectively select important words for *classifier* to process and the data aggregation strategy improves the model performance. As future work we will apply the framework for other text reading tasks. Another promising direction is to explore the benefits of text classification model in an edge-device setting. This problem naturally arises with local devices (e.g., smart watches or mobile phones), which do not have sufficient memory or computational power to execute a complex *classifier*, and instances must be sent to the cloud. This setting is particularly suited to ours since we could choose to send only the important words to the cloud. In contrast, skim-RNN and LSTM-jump, which process the text sequentially, have to either send the entire text to the server or require multiple rounds of communication between the server and local devices resulting in high network latency.

6 Acknowledgments

We thank the anonymous reviewers for their insightful feedback. We also thank UCLA-NLP group for discussion and comments. This work was supported in part by National Science Foundation grants IIS-1760523 and CCF-1527618.

[8] As an example, from Table 6 in Yu et al. (2017), the performance of LSTM-jump drops from 0.881 to 0.854 although it takes longer test-time (102s) than the baseline (81.7s).

[9] Because of the inherent accuracy/inference-time trade-off, it is difficult to depict model comparisons. For this reason, in Figure 2, we plot the trade-off curve to demonstrate the best speedup achieved by our model for achieving near state-of-art performance. On the other hand, test results are tabulated in Table 1 to focus attention primarily on accuracy.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *ICLR*.

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. 2016. Conditional computation in neural networks for faster models. *ICLR*.

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. In *ICML*.

Leo Breiman. 1996. Bagging predictors. *Machine learning*, 24(2):123–140.

Girish Chandrashekar and Ferat Sahin. 2014. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28.

Yin-Wen Chang and Chih-Jen Lin. 2008. Feature ranking using linear SVM. In *Causation and Prediction Challenge*, pages 53–64.

Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. 2017. Coarse-to-fine question answering for long documents. In *ACL*.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. 2016. Very deep convolutional networks for natural language processing. *arXiv preprint arXiv:1606.01781*.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A Smith. 2015. Sparse overcomplete word vector representations. In *ACL-ICJNLP*.

Tsu-Jui Fu and Wei-Yun Ma. 2018. Speed reading: Learning to read forbackward via shuttle. In *EMNLP*.

Jianfeng Gao, Galen Andrew, Mark Johnson, and Kristina Toutanova. 2007. A comparative study of parameter estimation methods for statistical natural language processing. In *ACL*.

Sergey Karayev, Mario Fritz, and Trevor Darrell. 2013. Dynamic feature selection for classification on a budget. In *ICML*.

Matt J Kusner, Wenlin Chen, Quan Zhou, Zhixiang Eddie Xu, Kilian Q Weinberger, and Yixin Chen. 2014. Feature-Cost Sensitive Learning with Submodular Trees of Classifiers. In *AAAI*.

Tao Lei, Regina Barzilay, and Tommi S. Jaakkola. 2016. Rationalizing neural predictions. In *EMNLP*.

Sam Leroux, Steven Bohez, Elias De Coninck, Tim Verbelen, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. 2017. The cascading neural network: building the internet of smart things. *KAIS*.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *EMNLP*.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *ACL*.

Laurens Maaten, Minmin Chen, Stephen Tyree, and Kilian Weinberger. 2013. Learning with marginalized corrupted features. In *ICML*.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *NeurIPS*.

Feng Nan and Venkatesh Saligrama. 2017. Adaptive classification for prediction under a budget. In *NeurIPS*.

Andrew Y Ng. 2004. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *ICML*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. 2011. No-regret reductions for imitation learning and structured prediction. *AISTATS*.

Min Joon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2018. Neural speed reading via skim-rnn. *ICLR*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

Kirill Trapeznikov and Venkatesh Saligrama. 2013. Supervised sequential classification under budget constraints. In *AISTATS*.

Paul Viola and Michael Jones. 2001. Robust real-time object detection. *IJCV*.

Felix Wu, Ni Lao, John Blitzer, Guandao Yang, and Kilian Q. Weinberger. 2017. Fast reading comprehension with convnets. *CoRR*.

Zhixiang Xu, Matt Kusner, Minmin Chen, and Kilian Q Weinberger. 2013. Cost-Sensitive Tree of Classifiers. In *ICML*.

Adams Wei Yu, Hongrae Lee, and Quoc Le. 2017. Learning to skim text. In *ACL*.

Guo-Xun Yuan, Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. 2010. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *JMLR*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NeurIPS*, pages 649–657.

P. Zhu, A. Acar, F. Nan, P. Jain, and V. Saligrama. 2019. Cost-aware inference for iot devices. In *AISTATS*.

Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.