HW1: Classification

Jiafeng Chen[*] Yufeng Ling Francisco Rivera[*]

February 4, 2019

1 Introduction

This note lays out our expectation for a homework submission in *CS287: Statistical Natural Language Processing*. While you do not have to follow this template to the letter, we do expect that write-ups have a very clear structure and cover all the elements described in this note. With this in mind, the burden is on the presenter to demonstrate why deviations from the standard are necessary.

All write-ups should include a short introduction. In this section you should summarize the underlying problem in high-level language and describe the extensions that you have decided to propose in your implementation. When you describe these extensions you should carefully cite the papers of interest. For instance, it will often be useful to cite the work seen in class (Murphy, 2012). Alternatively, you can also cite papers inline, for instance the work of Berger et al. (1996).

2 Problem Description—Old

In general, homeworks will be specified using informal language. As part of the assignment, we expect you to write-out a definition of the problem and your model in formal language. For this class, we will use the following notation:

- $\boldsymbol{b}, \boldsymbol{m}$; bold letters for vectors.
- $\boldsymbol{B}, \boldsymbol{M}$; bold capital letters for matrices.
- $\mathcal{B}, \mathcal{M}$; script-case for sets.
- b_i, x_i; lower case for scalars or indexing into vectors.

[*]Equal contribution

For instance in natural language processing, it is common to use discrete sets like $\mathcal{V}$ for the vocabulary of the language, or $\mathcal{T}$ for a tag set of the language. We might also want one-hot vectors representing words. These will be of the type $\boldsymbol{v} \in \{0,1\}^{|\mathcal{V}|}$. In a note, it is crucial to define the types of all variables that are introduced. The problem description is the right place to do this.

3 Problem Description

The focus of the problem set is sentiment classification. We are given *sentences* and aim to classify them as either positive or negative sentiment (a binary classification). These predictions can be made in a probabilistic fashion by writing y_i as the event that the ith sentence is positive sentiment, and calculating $p(y_i)$.

Sentences[1] are themselves sequences of words $w \in \mathcal{V}$ in some vocabulary $\mathcal{V}$. In particular, there will be two special words, $w_{\text{unk}}, w_{\text{pad}} \in \mathcal{V}$ which represent an unknown word and a padding unit respectively; we address their significance later. A particular sequence of words $w_1, \dots, w_n$ need not have a fixed length, which varies across sentences.

We represent words in two main ways. The first is as a one-hot encoded vector $\boldsymbol{v} \in \{0,1\}^{|\mathcal{V}|}$. This representation is useful for the Logistic Regression model. Alternatively, we can use a dense embedding. That is, each word gets assigned a vector $\boldsymbol{v} \in \mathbb{R}^d$ where d is the embedding dimension. We use Mikolov et al. (2013)'s pre-trained word embeddings ($d = 300$) for the Continuous Bag of Words and Convolutional Neural Network models.

4 Model and Algorithms

4.1 Naive Bayes

As we alluded to earlier, we can treat our prediction problem as probabilistic. We are interested in the probability of

$$p(y_i \mid w_1^i, \dots, w_n^i) = \frac{p(w_1^i, \dots, w_n^i \mid y_i)p(y_i)}{p(w_1^i, \dots, w_n^i)}$$

which is given by Bayes' rule. The Bayesian approach is formulated as follows:

- We first assign pseudo-counts to each word that act as priors in the Bayesian framework. This results in $\boldsymbol{m}^+, \boldsymbol{m}^- \in \mathbb{R}^{|\mathcal{V}|}$, representing the counts of words in positive/negative sentences. In our model, we initialized both to be vector of ones.

[1]We use *sentences* to mean a unit of data, which can in principle be multiple grammatical sentences.

- We train the model by updating $\boldsymbol{m}^+$ and $\boldsymbol{m}^-$. We increment m_i^+ (resp. m_i^-) by the number of occurences of word w_i in positive (resp. negative) sentences.

The predicted probabilities for a sentence with features $\boldsymbol{x}_i$ is given by softmaxing over

$$\begin{bmatrix} \log\left(\frac{\boldsymbol{m}^+}{\|\boldsymbol{m}^+\|_1}\right)^T \boldsymbol{x}_i + \log\left(\frac{N_+}{N}\right) \\ \log\left(\frac{\boldsymbol{m}^-}{\|\boldsymbol{m}^-\|_1}\right)^T \boldsymbol{x}_i + \log\left(\frac{N_-}{N}\right) \end{bmatrix}$$

where N_+ and N_- are respectively the number of positive sentences and negative sentences in the training set and $N = N_+ + N_-$. In addition, $\boldsymbol{x}_i$ is the max-over-time pooling of one-hot encoding matrix.

4.2 Logistic Regression

In this model, we consider the bag-of-words transform

$$\boldsymbol{x}_i = \phi(w_1^i, \ldots, w_n^i) = \sum_{j=1}^{n_i} \mathsf{onehot}(w_j),$$

and assume that

$$y_i \sim \text{Bern}(p_i) \quad p_i = \sigma(\boldsymbol{W}\boldsymbol{x}_i),$$

for the sigmoid function

$$\sigma(t) = \frac{1}{1+e^{-t}}.$$

The model is estimated via maximum likelihood over parameter matrix W. We maximize log likelihood via the Adam optimizer in a batched gradient descent setting, with a learning rate of 10^{-4} and weight decay of 10^{-4} for 20 epochs.

4.3 Continuous Bag of Words

A downside of logistic regression is that we have as many parameters as the size of our vocabulary $\mathcal{V}$. This means that without an extensive training set, we run the risk of overfitting. To address this, we can reduce the dimensionality of our embedding by using Mikolov et al. (2013)'s word embeddings. This model has three parts:

1. We start by using average-pooling over time on the word-embeddings to get a vector of dimension d for each sentence.
2. Then, we pass that vector through a linear transform to get another vector of dimension d_2 which then gets passed through a non-linear transformation (ReLU) to make

up the hidden layer.

3. Finally, we pass the hidden layer output through another linear layer to get a vector with two elements, and a softmax transformation to turn these values into the probabilities of positive and negative sentiment.

The model has two parameter matrices of size $d \times d_2$ and $d_2 \times 2$ respectively. We implement the model for $d_2 = 100$. We train using Adam for 20 epochs. In summary, the model's steps are visualized in Figure 1.

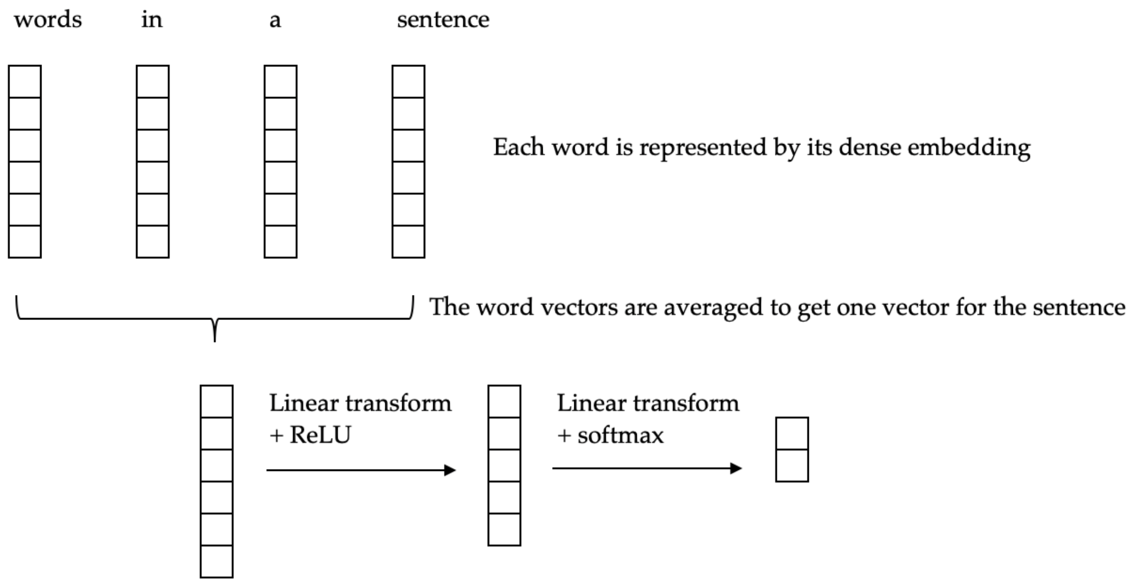
words
in
a
sentence
Each word is represented by its dense embedding
The word vectors are averaged to get one vector for the sentence
Linear transform + ReLU
Linear transform + softmax

Figure 1: CBOW model pipeline

We call this model a *bag of words* because the order of the words is lost in the average-pooling over time. In other words, any permutation of a sentence will result in the same prediction by this model.

4.4 Convolutional Neural Network

While the Continuous Bag of Words model makes use of the pre-trained dense embeddings, it suffers a limitation from the bag-of-words construction. Because the order of words is not taken into account, the sentences "it bad, not good" and "it good, not bad" cannot be distinguished.

To this end, we employ a one-dimensional convolution of the word embeddings. That is, we take a sliding window with size either 3, 4, or 5 over time, then take a linear transform to generate features for each location of the sliding window. We generate 100 features per window size.

Model	Weight
Naive Bayes	0.1755
Logistic Regression	0.3895
Convolutional Neural Network	0.4349

Table 1: Weights on each model for ensemble

Then, we max pool over time to get a fixed number of features for each sentence. These features then get passed through a ReLU and a dropout with $p = .5$ to get a hidden layer than finally gets passed through a linear transformation and a softmax to get our probabilities for each label.

The procedure is summarized in Figure 2.

4.5 Ensemble

For our generalizations, we create an ensemble of models in attempt to leverage the strengths of each. We construct our final ensemble from the Naive Bayes, Logistic Regression, and Convolutional Neural Network models. We pick these since they are our three highest-performing models and we expect each to capture something slightly different.

We depict a pair-plot of the predictions of each of the three models in Figure 3 to highlight that while the models do have a high degree of concordance, they are also not making exactly the same predictions, and so could benefit from an ensemble.

To implement the ensemble, we create a weighted average of the probability predictions of each model. We enforce that the weights are positive and sum to 1 by optimizing over log-space weights and training these hyper-parameters over our validation set. We display the weights that result from 20 epochs of training in Table 1

4.6 Ensemble voting

In addition, we also implement an even simpler form of ensembling. Instead of averaging each model's output probability, we can instead give each model a "vote" either for positive or negative sentiment (based on whether the model predicts a probability below or above 0.5). Then, of the three models' predictions, we pick the prediction that happens at least two times (the majority).

While this mechanism is less complex, we expect it to be more robust to strong convictions from any given model, thus insulating the ensemble from overconfidence.

5 Experiments

We show the accuracy and average loss (under the cross entropy criterion) for all models in Table 2. We see that the ensemble models seem to generally outperform non-ensemble models in validation accuracy, and bag-of-words based models seem to outperform embedding-based models. In particular, it is somewhat surprising that the CBOW-embedding neural network performs poorly even on the training set, potentially due to a small parameter size. On the other hand, the generalization gap for the other models is substantial.

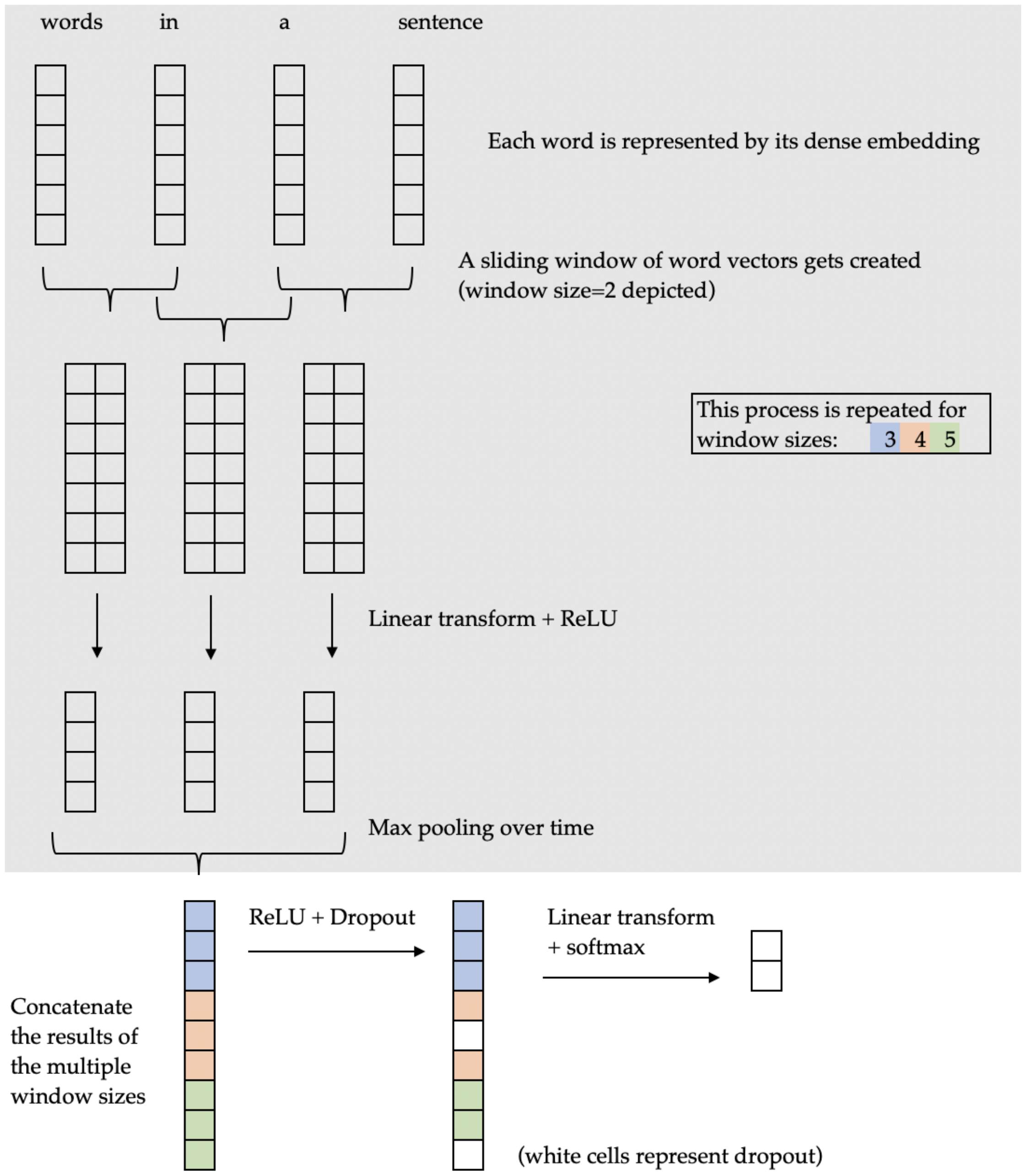

Figure 2: Convolutional Neural Network Model

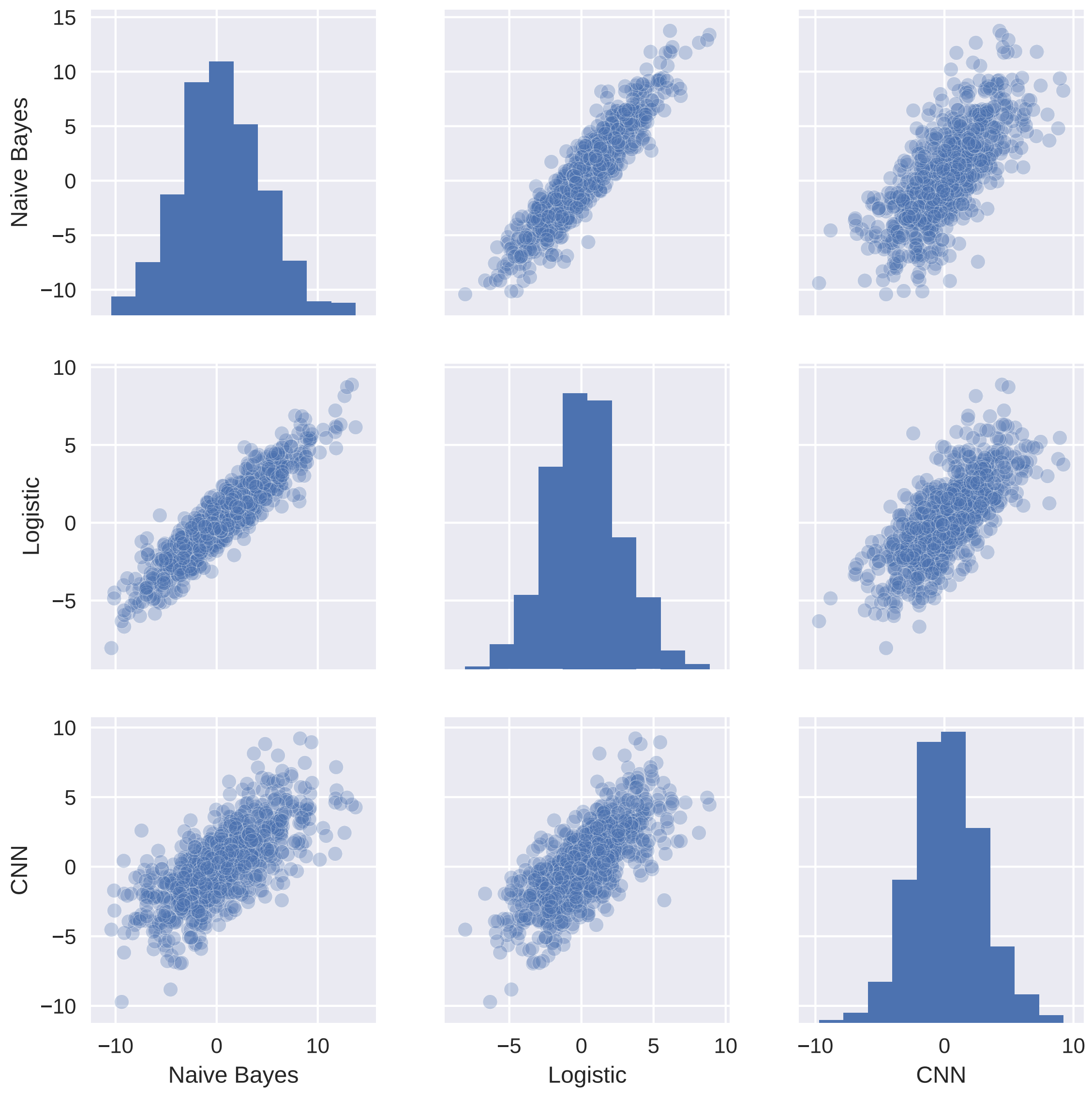

Figure 3: Correlation between Naive Bayes, Logistic, and CNN predictions on the validation set. We plot logit of the probabilities that a sentence is negative sentiment against each other (we clip $\pm\infty$ values in logit-space at ± 10).

	Training Accuracy	Training Average Loss	Validation Accuracy	Validation Average Loss
Naive Bayes	95.0%	0.0139	79.4%	0.0504
Logistic	98.8%	0.0131	78.2%	0.0487
Embedding NN	75.5%	0.0497	70.8%	0.0605
CNN	98.3%	0.0122	76.5%	0.0523
Ensemble	98.5%	0.0121	80.3%	0.0429
Ensemble-votes	98.7%	—	80.2%	—

Table 2: Accuracy and average loss (under cross-entropy loss function) for models considered. Note that for Ensemble, the ensemble weights are trained over the validation set and for the model Ensemble-votes, the loss cannot be computed since model does not output likelihood.

6 Conclusion

End the write-up with a very short recap of the main experiments and the main results. Describe any challenges you may have faced, and what could have been improved in the model.

References

Berger, A. L., Pietra, V. J. D., and Pietra, S. A. D. (1996). A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.