Reproducible Machine Learning
Mini Project 4 - COMP 551: Applied Machine Learning

Alexander Harris: 260688155 Abbas Yadollahi: 260680343 Le Nhat Hung: 260793376

McGill University - April 17, 2019

Abstract

Sentiment analysis is a sub-domain of opinion mining where the analysis is focused on the extraction of emotions and opinions towards a topic using textual data. In this project, we focus on the baselines used in a paper by Kim, 2014 which throws state of the art sentiment classification baselines against their own CNN models. Seeing as baselines models can sometimes out perform more complicated models, we modify their existing CNN and Linear SVM baseline implementations and design our version of a Naive Bayes SVM. Through extensive hyper-parameter tuning and feature processing, we improve the former baselines by a significant amount. Our approach using the Naive Bayes SVM classifier has the best accuracy on the MR dataset, achieving 91.4% and comes very close in the SST-2 dataset, achieving 87.3% compared to the state of the art from Kim, 2014 of 88.1%.

1 Introduction

Convolutional Neural Networks (CNNs) were originally invented for use in computer vision [4]. By exploiting grid structured inputs and extracting local features, they favor image classification, generation, or even game playing tasks [11]. However, CNNs have also proven surprisingly effective in **Natural Language Processing (NLP),** despite the sequential (and not grid) structure of sentences.

Kim, 2014 [3] further applied CNNs to NLP. CNNs were trained on top of pre-trained **word vectors** for sentence-level classification tasks. A "simple CNN with little hyperparameter tuning and static vectors" achieved excellent results, beating state of the art models at the time in 4 out of 7 tasks.

In the present work, we attempt to reproduce and improve on the results of Kim, 2014 using simple baselines. We choose to re-implement three of the baseline models: **Linear Support Vector Machine (SVM)**, **Naive Bayes SVM (NB-SVM)**, and finally the baseline CNN presented in the paper. These baselines are then evaluated on two datasets from the paper: **MR** and **SST-2**.

1.1 Preliminaries

Convolutional Neural Networks (CNNs) are a class of deep neural networks most commonly applied in computer vision, i.e. image recognition [4]. It has 3 main types of layers [2]:

1) **Convolutional layers** leverage the idea of local connectivity. Each layer neuron connects only to a local region of the preceeding layer. Used primarily for feature extraction.
2) **Pooling/subsampling layers,** commonly inserted between successive convolutional layers, these progressively reduce the spatial size of the representation and thereby the number of parameters and computations in the network. They therefore control overfitting.
3) **Fully connected layers** have full connections to all activations in the previous layer, like in regular neural networks. Typically inserted once at the end of the CNN to output the prediction as a vector of probabilities assigned to all classes.

Natural-Language Processing (NLP) refers to the ability of computers to analyze, understand, and derive meaning from human language as spoken e.g. voice assistant, or written e.g. movie review sentiment classification. **Word Vectors** are building blocks for NLP. A word vector is a row of real valued numbers, each capturing a dimension of the word's meaning. Semantically similar words have similar vectors. A popular model to learn the word vectors is **word2vec** - a 1-hidden layer neural network which learns from a corpus and assigns a word vector to each word. Indeed, our re-implementation utilizes word2vec.

Linear Support Vector Machine (SVM) is a generative classifier formally defined by a separating hyperplane. It is a perceptron for which we choose the weights such that the margin is maximized and equidistant from both classes [9].

Naive Bayes SVM (NB-SVM) replaces SVM's word count features with log-count ratios. Past findings have shown NB-SVM to be a robust model, performing on classification tasks for both long documents and snippets [13].

MR: dataset of movie reviews with one sentence per review. Classification involves detecting positive/ negative reviews [8].

SST: Stanford Sentiment Treebank - is an extension of MR but with train/dev/test splits provided and more nuanced labels (very positive, positive, neutral, negative, very negative) [12].

1.2 Important Findings

First, the improvements. We confirmed NB-SVM's robustness, as our implementation surpassed even Kim, 2014's best model, a CNN, on the MR dataset in terms of validation accuracy. Furthermore, our baseline Linear SVM outperformed Kim, 2014's CNN baseline on both datasets. These findings this pleasantly demonstrate how a simple model like NB-SVM can outperform deep neural networks with the right parameter tuning.

As for the reproduction of Kim, 2014's results, we successfully matched our accuracies to the paper's using our CNN and Linear SVM baselines, despite them unfortunately performing significantly worse compared to our best model NB-SVM.

2 Related Work

Wang and Manning, 2012 [13] presented a comprehensive comparison of non-deep learning models, of which NB-SVM interests us here. Their results demonstrated NB-SVM's robustness with regards to target documents' length (SVM alone worked better for long documents but worse for snippets; NB-SVM remedied this). This motivated our implementation of NB-SVM, which confirmed this robustness. Our implementation of NB-SVM not only performed remarkably on MR (one-sentence movie reviews), it ended up surpassing even Kim, 2014's best model in the same dataset.

Our present work also takes us full circle back to our first project. From linear regression models not even breaking 80%, we have now challenged, analyzed, and surpassed the results of another researcher's work.

3 Task Description

Our goal with this paper is to explore the baselines used in Kim, 2014 and attempt to implement our own that could possibly beat their best models. Kim, 2014 is quite a unique paper because the model they propose is quite a simple CNN with only a single convolutional layer, however it uses pre-trained word vectors from `word2vec`, which were trained on over 100 billion words from Google News. [6] Despite this, it was able to beat the previous state of the art models at the time on 4 out of the 7 datasets tested. [3] While Kim explores numerous text classification datasets, for the purposes of this paper we have decided to focus on two datasets that deal with binary sentiment classification problems, namely the Cornell Movie Review dataset (MR) [7] and the Stanford Sentiment Treebank (SST) [12]. The latter is a variant of the original dataset like the one refered to as SST2 in Kim, where neutral reviews have been removed and labels have been converted to binary. [3]

The basic model proposed in Kim, 2014 is a CNN which takes word vector embeddings as input. Then there are three convolutional layers that are concatenated together. Each has a filter size of 100, but varying kernel sizes of 3, 4 and 5, representing different n-grams. They terminate with a max pooling layer before being concatenated together and being fed into a final fully connected layer with a dropout of 0.5 and a sigmoid/softmax output (depending on the classification task). Kim proposes 4 variants of this basic model:

- **CNN-rand:** A baseline model with randomly initialized word vectors that are updated during training.
- **CNN-static:** A model that uses static pre-trained word vectors from `word2vec`.
- **CNN-non-static:** Similar to above but word vectors are updated during training.
- **CNN-multichannel:** A model with two sets of word vectors, where only one is updated at a time while the other remains static.

Since the proposed model is already fairly simple, and the baselines that are used for comparison in Kim are for the most part fairly complicated deep-learning models, we decided to attempt to beat the proposed model using non deep-learning baselines. We will also attempt to implement the basic CNN-rand from Kim, 2014 to see if more extensive hyperparameter tuning can lead to increased performance over what was reported.

4 Dataset

We decided to use the MR [7] and SST-2 [12] datasets seeing as they are both binary classification tasks and the vocabulary size is quite high, giving a wider feature set. Both datasets are represented by a raw text file containing the positive or negative label and the preprocessed sentiment sentence. We then have to choice to leave it as is or to tokenize the texts to create a different feature set. These tokens are then vectorized to produce either word occurences or TF-IDF weightings.

In the case of the MR dataset, there is no separation in between the train and test sets, as a result we are given the liberty to chose our validation pipeline. Seeing as the paper by [3] uses a 10-fold cross validation for this given dataset, we choose to follow the same for a proper comparison. As a result, we train on 9596 different sentences and test on the remaining 1066 sentences, seeing as the dataset contains 10662 movie reviews.

On the other hand, the SST-2 dataset has a very precise split, being already partitioned into a train, validation and test set. The latter sets contain 6920, 872 and 1821 sentences, respectively, for a total of 9613 samples. This makes it easier for us to mimic the process that [3] went through in their analysis.

5 Proposed Approach

5.1 Validation Pipeline

To verify our results, we decided to use both a held-out validation set and k-fold cross validation depending on the given dataset. As previously mentioned, the SST-2 dataset is already partitioned into a train, validation and test set, making the validation and testing pipeline straightforward with a 72:9:19 training/-validation/testing split, respectively. On the other hand, the MR dataset is lacking the former split, as a result we had to default to using k-fold cross validation with 10 folds.

5.2 Feature Pipelines

To test our chosen baselines, our feature pipelines was separated into 2 sections, the first being feature processing, and the second being the feature form. When it came to the feature form, we figured that the most logical separation would be word occurrences (binary) vs. TF-IDF weightings (continuous). Before so, to create our feature matrices, we used SKLearn's [10] built-in vectorizer methods, along with a custom tokenizer that was used to tokenize words, remove stopwords and perform lemmatization using NLTK [**nltk**]. We then performed grid search after these two pipelines by using a simple set of parameters such as bigram versus trigram, maximum number of word features, and by modifying the hyper-parameters used by the models. We found that the binary word occurrence pipeline with trigrams and with no cap on the maximum word frequency gave the best result using the Naive Bayes SVM model on both the MR dataset and the SST-2 dataset. Further optimization can also come from tuning model hyper-parameters, which will be discussed in the next section.

5.3 Models

We performed grid search cross validation on both the SKLearn classifiers and the Keras classifiers in order to optimize model hyper-parameters. We tuned parameters such as the maximum number of iterations, tolerance and learning rates, for the SKLearn models, and the number of epochs and batch size, for the Keras models. The optimal hyper-parameters we obtained for each model can be found in the Results section.

5.3.1 Naive Bayes SVM

As the name suggests, a Naive Bayes SVM model takes a linear model such as an SVM and infuses it with Bayesian probabilities by replacing word count features with Naive Bayes log-count ratios. The main component to implementing an NB-SVM is the Bayes' rule (Figure 1) that it uses to replace the word occurence features. Seeing as it uses binary features for classifying, we created the vectorizer using binary trigrams. Since our version of the Naives Bayes SVM was implemented as a neural network using Keras [1], the model was tested on two parts. First was the main parameters used to tune the Naives Bayes part of the classifier, which was the maximum number of features used for fitting. Afterwards was the tuning of the batch size and number of epochs hyper-parameters.

$$\delta = \ln P(y|x_j) = \ln \frac{P(x_j|y) * P(y)}{P(x)},\ where\ P(x_j|y) = \frac{\#\ instances\ with\ x_j\ and\ y+1}{\#\ instances\ with\ y+2} \tag{1}$$

Figure 1: Log-Count Ratio - Bayes' Rule with Laplace Smoothing

The main issue with Naive Bayes using binarized features is that each feature has equal weighting when in reality this is not true. Therefore, our way to remove useless features is by modifying our feature set to use a Naive Bayes log-count ratios for the words in a document. This results in less noise from otherwise meaningless features when classifying the reviews. At the same time, it substantially reduces computation time since a model accepting documents represented as sequences of word IDs trains much faster than one accepting rows from a term-document matrix. To help control the resulting probabilities when there's missing features in the training set, we use Laplace smoothing on the Naive Bayes log-count ratio. [13]

5.3.2 Linear SVC

SKLearn's Linear SVC implements an SVM using a linear kernel to allow for a more flexible loss functions for larger samples. In our case, we are using the hinge loss function which corresponds to a linear SVM. [10]

5.3.3 CNN

Our CNN is an implementation of the baseline CNN-rand model used in Kim, 2014 [3]. It has the same basic architecture as explained in Section 3, where the initial word embeddings are randomly initialized. We implemented this model using Keras [1] and the Tensorflow backend [5]. We referenced an existing implementation[1] of the models from [3] when designing our own.

6 Results

6.1 Baseline Performance

As can be seen from Table 1, our NB-SVM implementation was actually able to beat the performance of the best model from Kim, 2014 on the MR dataset, with a cross validation accuracy of 91.4%, much higher than their best model which only achieved a score of 81.5% [3]. Our two other baselines, the linear SVM and CNN, performed significantly worse than our best model, however their accuracies are quite close to Kim (2014) at 78.9% and 76.9% respectively. When it comes to the SST dataset, we found that while our NB-SVM gave the best accuracy on the test set out of our baselines at 87.3%, we were not able to beat the best model from Kim, 2014 which achieved 88.1%, however we are relatively close.

Our CNN implementation is fairly similar to the CNN-rand model from Kim, 2014, and as such we can see that they achieve similar performance, particularly on the MR dataset. We can also observe that SVMs remain a fairly good model when it comes to sentiment analysis, as they can compete quite closely and in some cases even beat much more complicated deep-learning models. For example, on the SST dataset our linear SVM achieved a score of 84.2%, higher than our CNN baseline and the CNN-rand model from Kim, 2014, all while having by far the lowest runtime of our baselines.

	Training Accuracy (%)		**Validation Accuracy (%)**		**Test Accuracy (%)**		**Runtime (s)**	
Model/Dataset	MR	SST	MR	SST	MR	SST	MR	SST
Linear SVM	78.9	98.6	78.9	83.2	-	84.2	20	9.4
NB-SVM	92.3	94.6	**91.4**	85.3	-	87.3	258	512
CNN	96.5	95.2	76.9	81.8	-	81.4	70	370
CNN-rand	-	-	76.1	-	-	82.7	-	-
CNN-static	-	-	81.0	-	-	86.8	-	-
CNN-non-static	-	-	81.5	-	-	87.2	-	-
CNN-multichannel	-	-	81.1	-	-	**88.1**	-	-

Table 1: Baseline results

6.2 Hyperparameter Tuning

For the Linear SVM model, we used Scikit-Learns's `GridSearchCV` [10] to evaluate the performance of different hyperparameters (C, tol, max_iter) on the MR and SST datasets. We found that the default model parameters of C=1, tol=1e-4 and max_iter=1000 gave the best performance on both datasets. For NB-SVM, we trained the model exhaustively testing different combinations of bath sizes and epochs. We found that a batch size of 64 and 5 epochs gave the highest accuracy on the validation set for SST, while for MR we found that a smaller batch size of 16 with the same number of epochs gave the best performance, with a cross-validation accuracy of 91.4%.

As for the CNN, we found that changing the optimizer from Adadelta to Adam helped speed up training time without impacting accuracy meaningfully. It was difficult for us to exhaustively tune hyperparameters for this model in a timely manner due to its complexity, however anecdotally we found that using the same hyperparameters as defined in Kim, 2014 appeared to give the best performance, as varying things such as

[1]https://github.com/davidsbatista/ConvNets-for-sentence-classification

batch size and number of epochs did not appear to have much of an effect on performance, although we would have liked to perform more extensive investigation if we had access to a GPU.

7 Discussion and Conclusion

While deep learning is probably one of the most popular subjects in machine learning these days, we can see from our results that simple machine learning models like SVMs can still compete with these more complicated models. Our standard Linear SVM implementation outperformed the baseline CNN from Kim, 2014 on both datasets, and our NB-SVM implementation was able to beat the best model from Kim on the MR dataset and come within 1% of the best model on the SST dataset. For our implementation of the baseline CNN from Kim, 2014, we found that additional hyper-parameter tuning did not appear to increase performance, which suggests that they may have already done this themselves when they evaluated their model. When it came to our own proposed baselines, we saw that hyper-parameter tuning did have a moderate effect on performance, and that different datasets performed better with different parameters.

This paper was fairly unique in the sense that the architecture of the model was fairly simple, and the complexity came from the specific implementation and the use of the pre-trained `word2vec` embeddings. While the use of these embeddings did provide significant improvements over the baseline CNN performance, these pre-trained models can be quite large and cumbersome, especially in an environment where memory and processing power are limited. While they might have the ability to generalize better to larger datasets, on these relatively small ones that the models were tested on our SVM-based baselines provided competitive if not better performance, all while not needing a powerful GPU to train in reasonable time frame.

Further improvements we would like to implement include performing more extensive hyper-parameter tuning on our CNN baseline to see if we can improve our performance over the baseline scores reported in [3]. We would also like to test a decision-tree or ensemble based baseline like Random Forests, as these have been shown to perform well on most text classification tasks. Another important point we believe would be interesting is to analyze our models on the CR and MPQA datasets, seeing as they have a much smaller vocabulary size than the MR and SST-2 datasets. From [13], we know that some models like our N perform better on longer reviews

8 Statement of Contributions

All team members have made significant personal contributions towards this project. The amount of work for each is described as follows:

- Alexander Harris: Linear SVM and CNN baseline implementation, training, validation and testing, write-up contribution.
- Abbas Yadollahi: NB-SVM baseline implemenation, training, validation and testing, write-up contribution
- Le Nhat Hung: Research, write-up contribution.

References

[1] Francois Chollet et al. *Keras*. `https://keras.io`. 2015.

[2] Andrej Karpathy. "CS231n Convolutional Neural Networks for Visual Recognition". In: (2018). URL: `http://cs231n.github.io/convolutional-networks/#convert`.

[3] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. DOI: `10.3115/v1/D14-1181`. URL: `https://www.aclweb.org/anthology/D14-1181`.

[4] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 0018-9219. DOI: `10.1109/5.726791`.

[5] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `http://tensorflow.org/`.

[6] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781 (2013).

[7] Bo Pang and Lillian Lee. "A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts". In: *Proceedings of the ACL*. 2004.

[8] Bo Pang and Lillian Lee. "Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales". In: *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. ACL '05. Ann Arbor, Michigan: Association for Computational Linguistics, 2005, pp. 115–124. DOI: `10.3115/1219840.1219855`. URL: `https://doi.org/10.3115/1219840.1219855`.

[9] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (Jan. 2012).

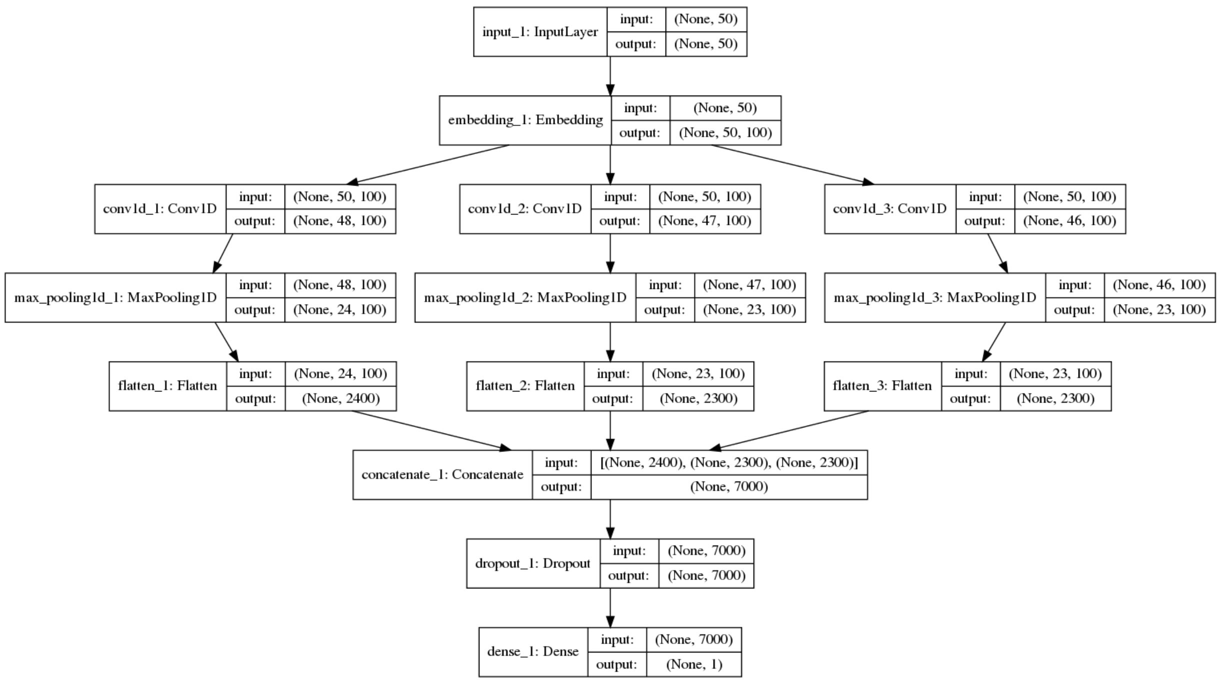
[10] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python ". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[11] David Silver et al. *Mastering the game of Go without human knowledge*. Oct. 2017. URL: `https://www.nature.com/articles/nature24270/`.

[12] Richard Socher et al. "Parsing With Compositional Vector Grammars". In: *EMNLP*. 2013.

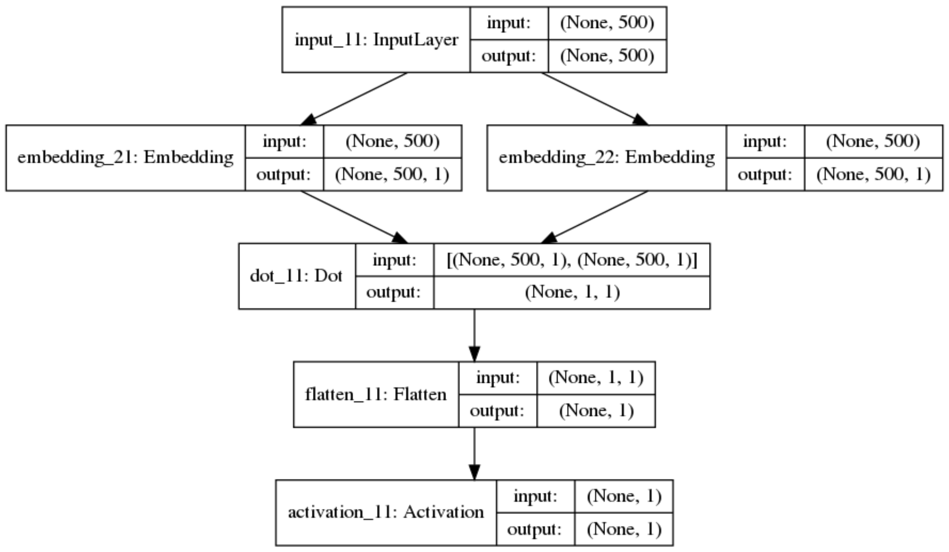
[13] Sida Wang and Christopher D. Manning. "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification". In: (2012).

Appendix



input_1: InputLayer | input: (None, 50) | output: (None, 50)
embedding_1: Embedding | input: (None, 50) | output: (None, 50, 100)
conv1d_1: Conv1D | input: (None, 50, 100) | output: (None, 48, 100)
conv1d_2: Conv1D | input: (None, 50, 100) | output: (None, 47, 100)
conv1d_3: Conv1D | input: (None, 50, 100) | output: (None, 46, 100)
max_pooling1d_1: MaxPooling1D | input: (None, 48, 100) | output: (None, 24, 100)
max_pooling1d_2: MaxPooling1D | input: (None, 47, 100) | output: (None, 23, 100)
max_pooling1d_3: MaxPooling1D | input: (None, 46, 100) | output: (None, 23, 100)
flatten_1: Flatten | input: (None, 24, 100) | output: (None, 2400)
flatten_2: Flatten | input: (None, 23, 100) | output: (None, 2300)
flatten_3: Flatten | input: (None, 23, 100) | output: (None, 2300)
concatenate_1: Concatenate | input: [(None, 2400), (None, 2300), (None, 2300)] | output: (None, 7000)
dropout_1: Dropout | input: (None, 7000) | output: (None, 7000)
dense_1: Dense | input: (None, 7000) | output: (None, 1)

Figure 2: CNN Network Architecture



input_11: InputLayer | input: (None, 500) | output: (None, 500)
embedding_21: Embedding | input: (None, 500) | output: (None, 500, 1)
embedding_22: Embedding | input: (None, 500) | output: (None, 500, 1)
dot_11: Dot | input: [(None, 500, 1), (None, 500, 1)] | output: (None, 1, 1)
flatten_11: Flatten | input: (None, 1, 1) | output: (None, 1)
activation_11: Activation | input: (None, 1) | output: (None, 1)

Figure 3: NB-SVM Network Architecture