Advanced Machine Learning Project Report

João Braz , fc60419
Joel Oliveira, fc59442

***Abstract*—This report presents the work performed for the task of sentiment polarity classification for both small sentences and big sentences movie reviews. The results are analyzed and discussed in the different scenarios. We could observe that they were very different.**

A pre-defined pipeline with word filtering and word embedding is followed thoroughly. This pipeline enumerates the steps from the beginning (raw data) to the end (review classification) of the process, which uses CNN for the binary classification.

I. Introduction

Text sentiment analysis is a common task in natural language processing where a text sequence is converted into a text category [1]. This categories, usually, are a reference to whether the text had a positive or a negative sentiment. For example, through the analysis of a product review on a e-commerce platform, enterprises can quickly know if consumers like a certain product, or if someone enjoyed a certain movie [2]. The companies can find which products are needing improvements [1], or use the results as input to recommender systems.

Different communities/people share their views in different ways. This leads to the fact that data used for the process of sentiment analysis or emotion recognition is very heterogeneous, which makes the task hard. [2]
This problem lays in two sub-problems

1) Represent each word with the proper features;
2) Select a Machine Learning (ML) model that can extract meaningful knowledge from the representation.

A. Word Embedding

Word embedding is a way of representing words in a continuous vector space. The vectors of words with similar meaning tend to be close to one another, capturing the correlation between those words [3]. Word embedding has been commonly used in Deep Learning (DL) as it achieves the best results, in comparison to one-hot encoding, which is a sparse representation [4].

B. Model

There are many different models that can be used for the task of sentiment analysis, such as Naive Bayes (NB) or Long-short Term Memory (LSTM) models [2]. Another one that can be used is the Convolutional Neural Networks (CNN). The latest is a deep feed-forward neural network that has been originally designed for computer vision. They contain multiple features that convolve with the original data, ”specializing” each filter in some aspect that relates to the desired output. In this manner, silent features can be extracted for the classification. The usage of this network for NLP tasks has grown. For NLP processing only 1 dimension is required for the data representation, instead of 2.

II. Approach

The approach followed in this work is similar to the one presented in [2], which can be seen in Figure 1. A few changes were made in order to improve the reliability and reproducibility of the study. For starters, instead of separating the data into training and testing sets, we created 3 sets. One used for **training**, one used as the **validation** set and finally the **test** set used only once to perform a final evaluation. Moreover the evaluation performed on the paper was solely using accuracy score. In this work we use the Matthew Correlation Coefficient (MCC), as it is a robust statistic for when the dataset is imbalanced.

Regarding the datasets used, for the small sentences the dataset used was the same as in in [2]. The large dataset was obtained through *Kaggle* [**?**].

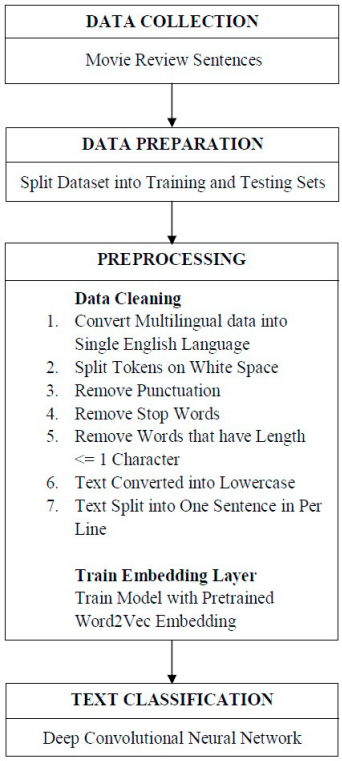

Fig. 1. Processing Pipeline

A. Data Collection

Like just stated, the work was performed in parallel for both a dataset with small sentences, from now on "*Dataset A*" and another one with more complex and elaborated phrases, from now on "*Dataset B*".

1) Dataset A: This sentence polarity dataset was retrieved from IMDb. It is available for research purposes, namely in NLP. It contains 5331 positive sentences and 5331 negative ones [2]. It is a dataset balanced with 50% of the elements for each class.

2) Dataset B: This dataset was obtained through Kaggle [5]. This is a much bigger dataset, with 320 thousand movie reviews. It is also imbalanced, as it contains around 30% of it's labeled with the negative polarity and the other 70% labeled with the positive class.

B. Data Preparation

In order to access if the trained model can have a good performance in unseen data, some samples must be removed from the training set. For both the datasets the sets created were one for **training**, another for the model **validation** and finally an **independent test** set to evaluate the final results of the model.

C. Data Preprocessing

Every sentence was converted to English, for two reasons: 1) The dataset had Multilingual text; 2) Fix minor errors in human writings that can be detected easily such as, e.g.: "didnt" → "didn't".
Afterwards each document from the corpus was split into tokens where the punctuation and english stopwords were removed. Small words with only one character were also filtered. Finally the text was parsed to lowercase.

D. Training Embedding Layer

One popular method to generate these word embeddings is *Word2Vec* [6]. It is a neural network model that learns in a self-supervised way the dense representations of words. The usual architecture used for this process is the *Skip-gram* [Fig. 2]. It involves training a model on a large corpus of text to predict the context of a target word within a sliding window of neighboring words. By learning to predict nearby words, the model generates dense vector representations of words that capture semantic and syntactic relationships. The resulting embeddings can be used to enhance the performance of natural language processing (NLP) tasks, including sentiment analysis.

Another popular choice is the *Glove* [7] embedding, created by Standford's university. This methodology uses the global co-occurrence statistics of words in a corpus as the core information for the embedding generation.

It is common practice to use pre-trained representations as a starting point for NLP tasks. In the work done, it was used a Word2Vec pre-trained on Google News dataset, that contains 3 million words/phrases. It contains embeddings of 300 dimensions. These embeddings will be the input for the classifier [2].

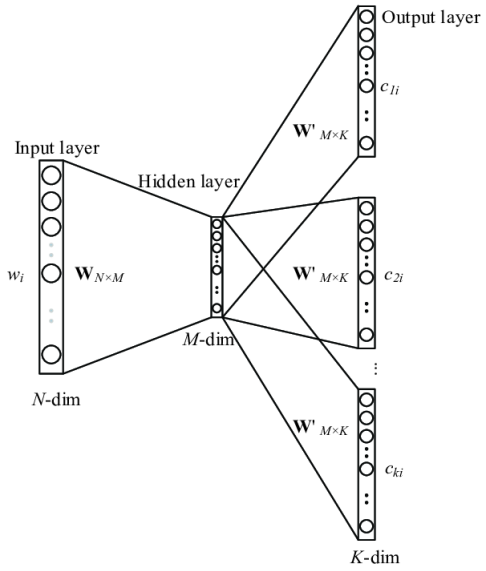

Fig. 2. Skip-gram Model

E. Text Classification using CNN

CNN models use kernels to filter the data. Each filter convolves with windows of a certain number of words, interpreting the data like a *n-gram*. Afterwards a pooling is performed to undersample the data, with the goal of reducing noise and keep the most relevant data.

III. Implementation

The approach described in the last chapter was implemented using Python 3 [8]. This programming language is very powerful and provides a wide set of packages with tools for multiple tasks. The tools used for the development were:

- *Nltk* [9] - Package with multiple NLP tools.;
- *NumPy* [10] - Package for fast processing of large quantities of data;
- *Tensorflow* [11] - Software library for machine learning and artificial intelligence, with focus on deep neural networks;
- *Gensim* [12] - Library for unsupervised topic modeling, document indexing, retrieval by similarity, and other NLP functionalities;
- *Scikit-learn* [13] - Software library for machine learning;
- *googletrans* - Package that allows to make request to google translator REST API.

A. Data Preparation

The preparation of the data, as mentioned in section II-B was performed using *Scikit-learn*, since it has methods implemented to ease this task. As mentioned three sets were created.

A first split was performed by separating the training test and the test set with the following ratios:

- Training Set - 90%
- Test Set - 10%

Then the training set was again split into the data to train the model and the data to validate the model:

- Training Set - 80%
- Validation Set - 20%

The total rates of each split were:

- Training Set - 72%
- Validation Set - 18%
- Test Set - 10%

B. Data Preprocessing

The translation of each sentence was made using the Google Translator REST API with the methods already implemented in *googletrans* package.

Nltk provides methods to perform the tokenization of documents. Then from each token the punctuation is removed, the stopwords (commonly used words in a language, e.g, "The", "is", "and") and words with length one are filtered. Finally the word is parsed to lower case. In this process there is a boolean flag that indicates whether or not we intend to build a vocabulary from the corpus. This "vocabulary" is nothing more than a dictionary with the distinct words mapping to unique index.

For the *dataset A*, in the training set there were 15899 distinct words. In all the dataset there were a total of 17491. In the *dataset B* there were a lot more words, as expected, having the training set a total of 183992 distinct words.

C. Embedding Layer

For the embedding layer, as we said, we started the vectors from a pre-trained model with 300 dimensions, in a non-static format, so that they can be optimized for the task at hand

. *Gensim* is a known tool for this purpose, and it has an API implemented that allows the download of these pre-trained models, namely the "word2vec-google-news-300", which contains 3 million words.

A problem that was found in this part of the process was that even though the pre-trained model had a lot of words, due to the fact that humans can mistype words and use abbreviations. This led to a lot of words that did not exist in the pre-trained model. For each word that was not in the word2vec model a random vector was generated from a uniform distribution. Both a padding and an unknown token were also added to the model with a vector with zeros.

After all the vectors were added to the word2vec model, each word in the corpus was mapped to the respective index. To set the data to a tabular format every document was padded or truncated to the maximum token length of the training set.

Finally, to incorporate the word2vec embeddings in the *Tensorflow* pipeline an embedding layer was initialized having it's vectors assigned to the vectors of the *Gensim* model.

D. CNN model

The CNN model was implemented with the use of the *Sequential* model from *Tensorflow*. The implementation can be seen in Figure 3. A dropout layer was included after the embedding layer with a dropout rate of 40%. Another was inserted before the fully connected layer, with a dropout rate of 20%.

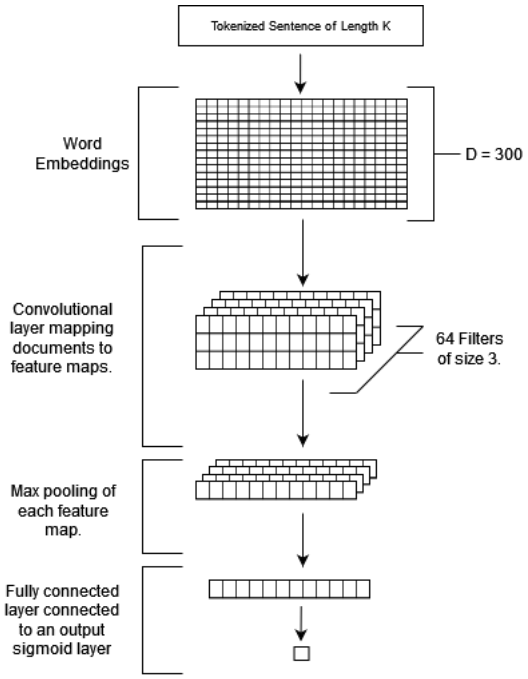

Fig. 3. CNN model used in the work being presented.

This model was then compiled using "adam" as the optimization function, binary cross-entropy as the loss function and accuracy as the metric. The model was fit early stopping, with patience 3.

The model trained with the *dataset A* used batches of size 32. The model trained with the *dataset B* used batches of size 128. This was the maximum batch size that could be selected in the experiments due to memory limitations.

IV. Results

A. Dataset A

The results obtained in the training process of the *dataset A* can be observed in Figures 4 and 5. Comparing to the reference paper [2] we could not reach the performance obtained. In the original paper the model scored 82% of accuracy. However, the trend of the learning process was similar, since the best score was also obtained in the second epoch.

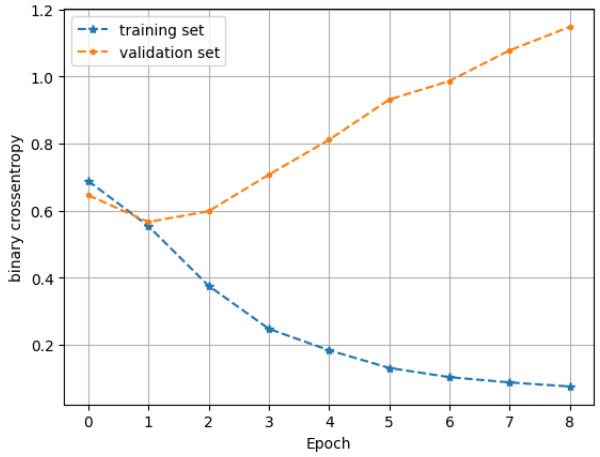

Fig. 4. Loss Scores for model training using *Dataset A*

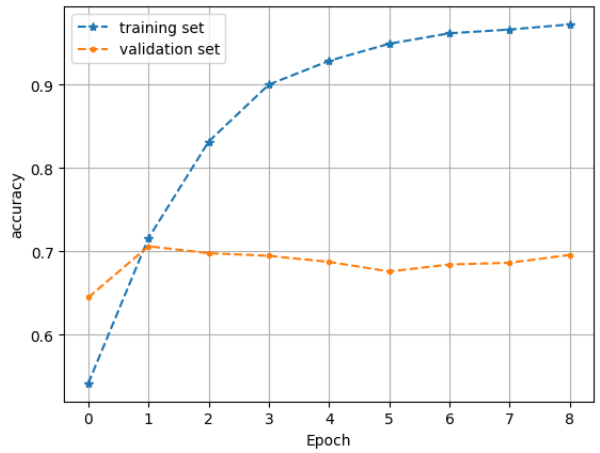

Fig. 5. Accuracy Scores for model training using *Dataset A*

It could be verified that one of the most important aspects to obtain good results was the word embedding representation. This embeddings were set to be non-static are therefore were updated with the gradient. As can be seen in table I which shows the most similar words to the word "hate". In the pre-trained model we can see that closest word is in fact "love" which makes since they are both verbs. However, for this context they have opposite meanings, since "love" is probably related to the positive polarity while "hate" is related to the negative polarity. After the training process we can see that this is the case, since "love" is no longer the most similar word, not even one of the closest 5.

To understand the model a Lime was used. A sentence from the test set, where the model performed really poorly, was selected in order to understand the model limitations. For this sentence, of the positive class, the model classified it as one belonging to the negative class with a probability of 99%. The sentence is the following:

- *maybe it is formula film making, but there's nothing wrong with that if the film is well-crafted and this one is*

Rank	Pre-Trained Word2Vec	CNN Embeddings After Training
1	love	loath
2	hated	dislike
3	hating	jonah
4	loathe	hated
5	dislike	fuhgeddaboutit

TABLE I
TOP 5 WORDS WITH THE EMBEDDING REPRESENTATION MOST SIMILAR TO THE WORD "HATE" EMBEDDING BEFORE AND AFTER TRAINING THE CNN MODEL.

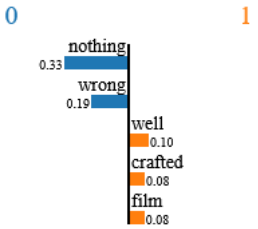

Fig. 6. Top 5 most important words for each label prediction, according to Lime.

In Figure 6 we can see words with most relevance to the classification. Looking at it, we can see that the model indeed related words commonly used to express negative opinions to the negative class. Another thing we can see by looking at the results and at the sentence is that the model does not capture the context in which the word is inserted. Context that in this case changes the meaning of the word in the sentence altogether.

B. Dataset B

The *Dataset B* differed from the *Dataset A* in the sense that it is imbalanced, with 30% of the values from the negative class and 70% from the positive class.

Similarly to the scenario of *Dataset A* the model doesn't improve much after finishing the first epoch, as can be seen in Figures 7 and 8, which could mean that the model is powerful enough to learn the core information for polarity classification in only one epoch. However, it can be noticed that the accuracy scores are higher than before, in the validation set, and the loss score is also lower.
Due to the fact that the dataset is imbalanced the accuracy measure is not a good indicator of the performance of the model, therefore the Matthews Correlation Coefficient (MCC) was calculated for the validation set after the training process, obtaining a value of 0.75.

We can see that only by increasing the size of dataset the model results improved considerably. The dataset used in paper [2] seems to not have enough data to train a model with a larger complexity as is the case of Convolutional Neural Networks. The model architecture could probably be tuned to

improve the results as well, so for corpus with larger datasets this type of model seems to be promising.

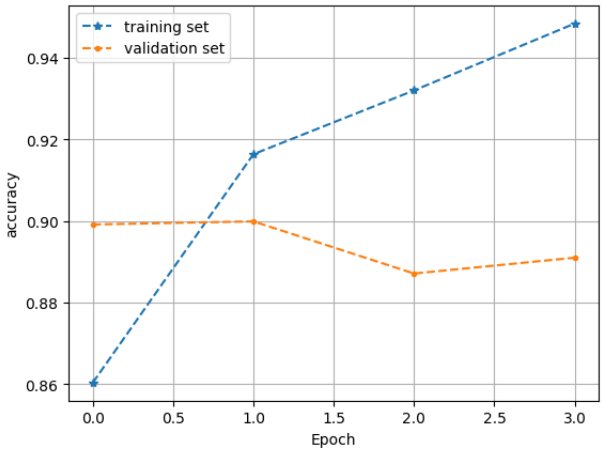

Fig. 7. Loss Scores for model training using *Dataset B*

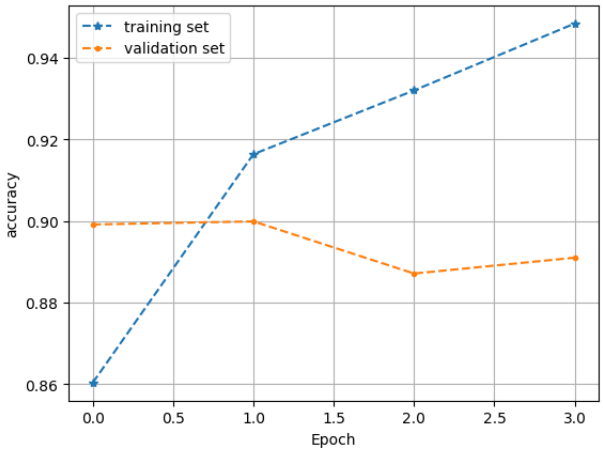

Fig. 8. Accuracy Scores for model training using *Dataset B*

Regarding the word embedding, similar results were obtained. The training process clustered the words regarding negative class and words that led to a positive class.

To perform a final evaluation in this model, the test set was used. The Matthews Correlation Coefficient obtained was similar to the one obtained in the validation set, around 0.75. We can see the confusion matrix generated by the model results in Figure 9. Although class distribution was not balanced we can see that the model adjusted well to the least present class, as it miss classified more elements from the frequent class than from the infrequent one.

V. Final Comments

The results obtained in the different datasets showed some variations. For *Dataset A*, which consisted of smaller sentences, the accuracy achieved was not as high as in the reference paper. However, the trend of the learning process was similar. One possible limitation of the model was the fact

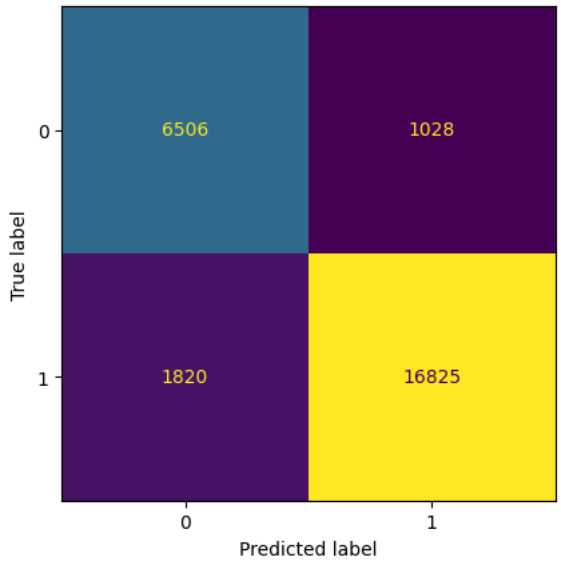

Fig. 9. Confusion Matrix with the results from the test set

that there were a significant amount of relevant words in the validation set that did not appear at all in the training set.

For *Dataset B*, that comprised of more complex and elaborated phrases, the model showed improved performance.

While the chosen architecture for the model chosen has demonstrated effectiveness in obtaining the raw meaning of words, by using the Lime model to understand the model. It could also be observed that the model has difficulties in capturing the meaning of a word when in inserted in a context that, given the previous word dependencies completely changes the meaning of the word. The model is also likely to be unable to capture sarcasm in the sentences.

A way of improving the results could be by using some embedding model that uses context dependencies to represent the words such as BERT.

References

1 Lu, R., Li, Y., and Yan, Y., "Construction of text emotion classification model based on convolutional neural network," in *2021 International Conference on Intelligent Computing, Automation and Applications (ICAA)*, 2021, pp. 271–274.

2 Sharma, A. K., Chaurasia, S., and Srivastava, D. K., "Sentimental short sentences classification by using cnn deep learning model with fine tuned word2vec," *Procedia Computer Science*, vol. 167, pp. 1139–1147, 2020, international Conference on Computational Intelligence and Data Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920308826

3 Lu, R., Li, Y., and Yan, Y., "Construction of text emotion classification model based on convolutional neural network," in *2021 International Conference on Intelligent Computing, Automation and Applications (ICAA)*, 2021, pp. 271–274.

4 Ombabi, A. H., Lazzez, O., Ouarda, W., and Alimi, A. M., "Deep learning framework based on word2vec and cnnfor users interests classification," in *2017 Sudan Conference on Computer Science and Information Technology (SCCSIT)*, 2017, pp. 1–7.

5 Fragkis, N., "Imdb 320.000 movie reviews," https://www.kaggle.com/datasets/nikosfragkis/imdb-320000-movie-reviews-sentiment-analysis, [Online; last accessed 17-May-2023].

6 Mikolov, T., Chen, K., Corrado, G., and Dean, J., "Efficient estimation of word representations in vector space," 2013.

7 Pennington, J., Socher, R., and Manning, C. D., "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

8 Van Rossum, G. and Drake, F. L., *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

9 Bird, S., Klein, E., and Loper, E., *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

10 Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

11 Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

12 Rehurek, R. and Sojka, P., "Gensim–python framework for vector space modelling," *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, vol. 3, no. 2, 2011.

13 Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.