**Recognizing Textual Entailment (final project)**

Sofia Timofeev, Arie Korzhevski

1. **Approach to the task**

One of the first things that we thought about, was the similarity between current task and homework #2, where we needed to train the model, based on Amazon products reviews, and given a review, to predict which rating it was given from 1 to 5.

Relationship can be seen as rating, but the model would predict based on how close the hypothesis was to the original text. And so we needed to find a way to do it.

* **BERT model**

While searching on the internet for different approaches and ideas for the task, **BERT** model was the most popular solution for the RTE problems. On the Stanford NLP group website, results for BERT like models were much higher than the traditional machine learning implementations, and so we decided to try it as well.

But from the start we understood that it’s a much more complicated model to implement, since it had a completely different approach from the previous models that we learned. And even though we had some basic knowledge on BERT itself from the lecture, we couldn’t make it work as we wanted, getting lower results than with the word2vec version, and that’s why we went with the latter one.

* **Word2Vec model**

During the lecture, we learned about distributional semantics, and how to represent a word with a vector. One of the tools that were presented was **word2vec**. Vectors are learned by understanding the context in which words appear, and it will have similar numeric representation. We thought it would give us a solution, so we decided to use it as a base for our model.

After additional search on the internet, we saw that word2vec is indeed widely used for solving similar tasks that were presented to us.

We found out about **Gensim** - an open-source library for unsupervised topic modeling and natural language processing, that includes the word2vec model.

It has a lot of pretrained models, but it also has an option to make a custom one, based on your own dataset. And that’s exactly what we did.

We used *The Stanford Natural Language Inference (SNLI) Corpus* for our training.

First, we cleaned up the corpus (lowering letters, disabling punctuation, etc.).  
One of the debates that we had was, if it would be better to clean up the corpus from stopwords, in order to achieve higher accuracy. But, since sentences could heavily rely on words that are usually included in stopwords, it could change the relationship, so we decided to leave the original text.

Also, the corpus that we based our training on, had an excessive label “-”, that wasn’t presented on the official website, and we didn’t find any explanation about it. Instead, we decided to remove sentences that had this label during the cleaning of the corpus, so it won’t mess up with our results in the end.

After cleaning both the train and the test set, we created 2 models using gensim’s word2vec (model for 2-way and model for 3-way). Then with models in our hands, we needed to decide how our model would make a prediction.

In one of the previous homework, we were using Logistic Regression of the sklearn library, so we decided to use it in our current task as well, since it was familiar to us and fulfilled the requirement of our needs.

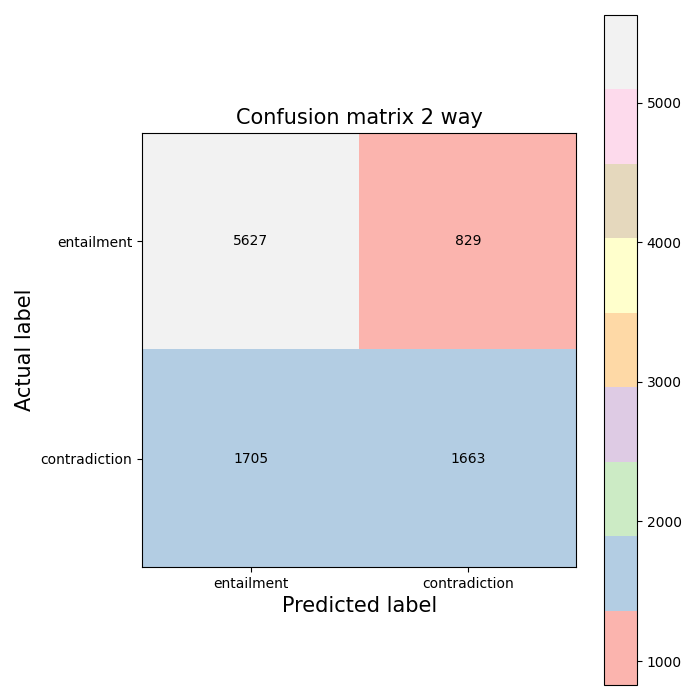
Logistic Regression needs an array of vectors(*train set*), labels for fitting, and an array of vectors(*test set*) for prediction.

Our idea was to concatenate original text and prediction, sum up vectors that represent each word, and add the final vector to an array. We did it for both word2vec models (train and test), and used it as parameters for Logistic Regression.

1. **Results**

**Two-way:** (1)Entailment, (2)Non-entailment

End result: **~ 74%**



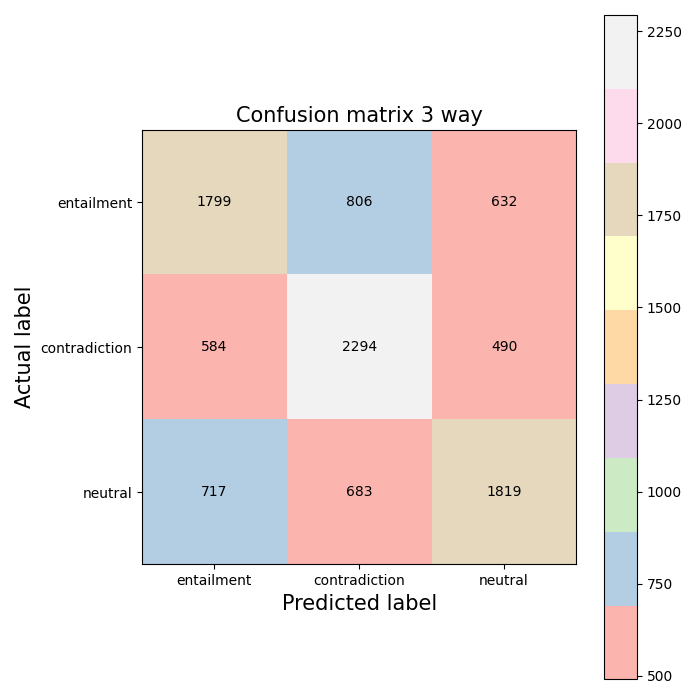
In 2-way, we got a higher discrepancy in non-entailment.

Since it consisted of contradiction + neutral relationships, many sentences that were classified falsely as an entailment were once tagged as neutral.

This happened because the neutral sentiment hypothesis sentences contain a number of semantically similar words to the original text; by having more of these words, the sentence was falsely classified as entailment, even though the meaning wasn’t close to the original sentence.

**Three-way:** (1) Entailment, (2) Contradiction, (3) Neutral

End result: **~ 60%**



In 3-way, contradiction got the best result, since it’s easier to understand when the hypothesis is completely wrong, i.e a sentence containing different words semantically.

Entailment was the worst out of the bunch, as our model only consists of word semantic similarity it misses the nuance of the meaning. Thus, even though two sentences had the same meaning, the vector wasn’t similar enough to the rest of the training vector examples.

Overall, as expected, 2-way scored much better than 3-way, also because it’s easier to predict when you have less choices, and the choices are more straightforward.

1. **Analysis**

1. We tried different approaches in which way to send vectors to Logistic Regression.

* Summing up vectors of each word per sentence, where one sentence has both original text and it’s hypothesis.
* Summing up vectors, but first we divide between text and hypothesis. Then, we sum vectors separately, and concatenate both sums with a numpy concatenate function. We got improved results using this method, instead of simply summing up vectors.
* Cosine Similarity: we got worse results than before, and had problems with implementation of it, and so we didn’t continue with it.

2. We started with vectors of size 100, and then increased size to 300, improving the overall result, but increasing the overall runtime.

3. We tried to change parameters of Logistic Regression, for example increasing max\_iter to 300, and it also improved the result. Increasing to a higher number of iterations improved it a bit, but it also increased runtime significantly.

1. **Instructions**

Our code runs both 2-way and 3-way.

We tried to save vectors with a pickle function, but the files were big, so we couldn’t include them in our assignment. Thus, we removed the function itself from the code.

Please put corpus files (train and test files) into the same directory with the code.

Estimated time is: ~15 minutes