

# Description of the System Developed by Team Athene in the FNC-1

Andreas Hanselowski, Avinesh PVS, Benjamin Schiller, Felix Caspelherr  
UKP-Lab, TU-Darmstadt, Germany

June 15, 2017

## 1 System installation:

The code can be downloaded from github: [athene\\_system](#)

Details of the required packages and installation are described in the *README.md*

In order to be able to reproduce our last submission for the FNC-1 competition from June second 2017, the serialized model and the features must be downloaded from [google drive](#) and placed in the corresponding folders.

For this purpose,

```
⇒ unzip models.zip athene_system/fnc-1/data/fnc-1/mlp-models
```

```
⇒ unzip features.zip athene_system/fnc-1/data/fnc-1/features
```

Moreover, in order to be able exactly reproduce our submission, the system configuration should be the same as it is described in the *README.md*. In particular, one should use *python 3.4* and *tensorflow 0.9.0* (GPU version). Any other configuration might lead to different results.

## 2 How to run the system:

In order to run the system, the file *pipeline.py* in the directory *athene\_system/fnc-1/fnc* must be executed.

```
⇒ python pipeline.py -p ftest
```

For more details:

```
⇒ python pipeline.py --help
```

```
⇒ python pipeline.py -p crossv holdout ftrain ftest
```

*crossv*: Runs 10-fold cross validation on train/validation set

*holdout*: Trains classifier on train and validation, tests it on holdout set

*ftrain*: Trains classifier on the whole dataset and saves it to *data/fnc-1/mlp-models*

*ftest*: Predicts stances of unlabeled test set based on the model

The resulting file with the predicted labels is stored in *athene\_system/fnc-1/data/fnc-1/fnc-results*.

### **To reproduce the last submission from June 2.**

```
⇒ python pipeline.py -p ftest
```

The serialized features and model are loaded, and the predictions are made on the official testing set provided on June 1. The predictions must exactly correspond to the submission on June 2.

### **To Train the model and predict on the testing set**

```
⇒ python pipeline.py -p ftrain ftest
```

In this setting, the serialized features are loaded, the model is retrained and the predictions are made on the official testing set provided on June 1. Since the weights are initialized differently, there might be some variance to that of our official submission.

### **To generate features, train the model and predict on the testing set**

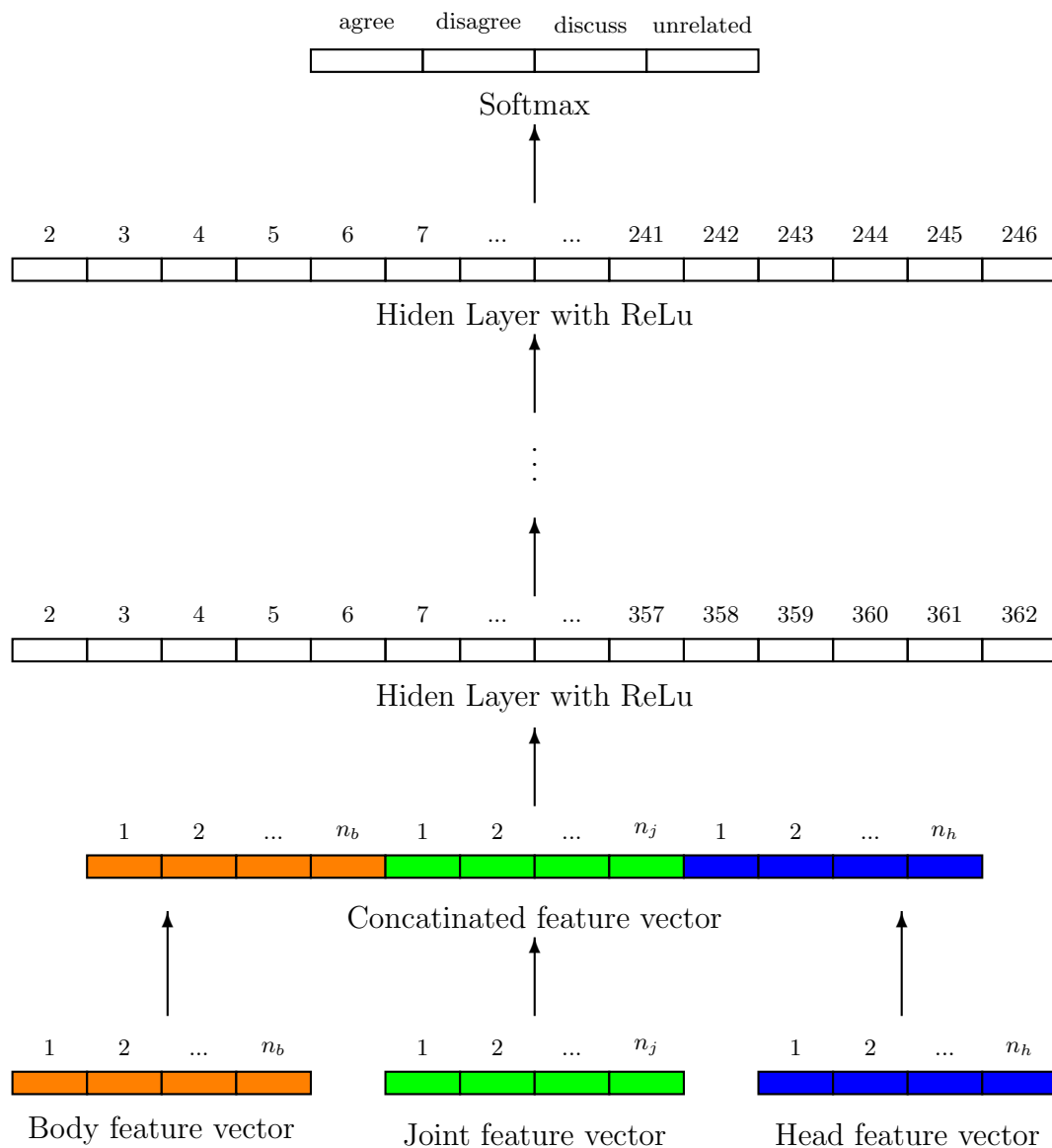
Delete all the features from the directory *athene\_system/data/fnc-1/features* and retrain the system as mentioned above.

```
⇒ rm -rf athene_system/data/fnc-1/features/*
```

```
⇒ python pipeline.py -p ftrain ftest
```

In this setting, the serialized features are recalculated and the model is retrained. Finally, the predictions are made on the official testing set provided on June 1.

### 3 Architecture of the multilayer peceptron



## 4 Approach

1. As a baseline, we used the multilayer perceptron with bag-of-words features, as suggested by [Davis and Proctor(2017)].
2. In order to optimize the hyper parameters, we carried out a random search [Bergstra and Bengio(2012)]. The structure of the resulting multilayer perceptron is displayed in Section (3).
3. In addition to the bag-of-words features, the following features have been implemented: FNC-1 baseline features, topic models and features for stance detection as suggested in [Ferreira and Vlachos(2016)]
4. Depending on the feature type, either individual feature vectors for the article body and the headline are created and then concatenated, or a joint feature vector. Both types of input features vectors to the multilayer perceptron are illustrated in Section (3).
5. Some of the features require a vocabulary, which has been generated on the basis of the training set, development set, holdout data and the official testing set from June 1.
6. In order to further improve performance, an ensemble method consisting of 5 multilayer perceptrons has been used, whereby the labels have been predicted by hard voting.
7. The model has been trained on the basis of the training set, development set and the holdout data, in order to make prediction for the official testing set from June 1.

## 5 Detailed description of the approach

The system setting described in the following resulted from a large number of experiments and the validation of the performance using the 10-fold cross-validation provided with the baseline code. In the final weeks of the challenge, the performance has also been tested on the holdout data, which yielded similar results as the cross-validation. All changes, which have been made to the model, turned out to improve performance even if only to a small extend.

In the following, the model and the used features are discussed in a little more detail.

**Multilayer perceptron** (after the hyper parameter optimization of the model described in [Davis and Proctor(2017)]):

1. Number of layers: 7
2. Number of unit per layer: 362, 942, 1071, 870, 318, 912, 247

3. Configurations: Learning rate = 0.001, Batch size = 188, number of epochs = 70, optimizer = Adam, learning rate decay = on, keep\_prob\_const = 1.0, weight initialization = sqrt\_n, bias initialization = 0.001, activation function = relu

**Features:**

1. Baseline features:  
Original features from the fnc-1 baselines: *overlap*, *refuting*, *polarity*, *hand*  
Keys in python: `overlap`, `refuting`, `polarity`, `hand`
2. Bag-of-words unigram features:  
Description: Simple bag of words n-gram features represented by a vector, whereby the entries represent the frequency of the n-gram (length 5000 head + 5000 body). Concatenation of head and body, l2 norm and bleeding (BoW = train+development+holdout+unlabeled test set)  
Feature type: concatenated feature vector  
Configurations: Vectorizer = tf-idf, stop words removal = on, vector length = 5000, use idf = off, norm = L2  
Key in python: `word_ngrams_concat_tf5000_l2_w_holdout_and_test`
3. Non-negative matrix factorization cosine distance:  
Description: Implements non negative matrix factorization. Calculates the cosine distance between the resulting head and body topic models vectors.  
Feature type: joint feature vector  
Configurations: Vectorizer = tf-idf, stop words removal = on, vector length = 5000, use idf = off, norm = L2  
Key in python: `NMF_fit_all_incl_holdout_and_test`
4. Non-negative matrix factorization concatenated:  
Description: Implements non negative matrix factorization. The topic model vectors of the headline and the body are concatenated.  
Feature type: concatenated feature vector  
Configurations: number of topics = 300, vocabulary based on: train, development, hold out, official test data set  
Key in python: `NMF_fit_all_concat_300_and_test`
5. Latent Dirichlet Allocation:  
Description: Sklearn LDA implementation based on the 5000 most important words (based on train+test+holdout+ unlabeled test data's term freq =  $\chi^2$  bleeding). Returns feature vector of cosine distances between the topic models of headline and bodies.  
Feature type: joint feature vector  
Configurations: n\_topics=25, use\_idf=False, term\_freq=True  
Key in python: `latent_dirichlet_allocation_incl_holdout_and_test`
6. Latent Semantic Indexing:  
Description: Takes all the data (holdout+test+train) and interpreters the

headlines and bodies as different documents. Instead of combining them, they are appended. Then it tokenizes these 50k headline-docs and 50k body-docs, builds a TfIdf-Matrix out of them and creates a LSI-Model out of it. In the next step the headlines and bodies for the feature generation are also treated as different documents and merely appended. Also, they are tokenized and a TfIdf-Matrix is built. This matrix is passed to the learned LSI-Model and a Matrix is being returned. In this matrix, each document is represented as a vector with length(topics) of (topic-id, distance of this doc to the topic). The probabilities are then taken as a feature vector for the document. The first half of the matrix represent the headline docs, the latter half represent the body docs. In the end, the feature vectors of the headlines get concatenated with its body feature vector.

Feature type: concatenated feature vector Configurations: n\_topics=300  
Key in python: `latent_semantic_indexing_gensim_holdout_and_test`

#### 7. Word similarity:

Description: Comparison of the embeddings of the nouns and verbs of the headline and the body (motivated by paraphrase detection).

Feature type: concatenated feature vector

Configurations: -

Key in python: `stanford_wordsim_1sent`

#### Ensemble modeling:

1. Approach: Voting
2. Voting type: hard
3. Number of models 5 (multilayer perceptrons described above randomly initialized)

## References

- [Bergstra and Bengio(2012)] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13(Feb):281–305.
- [Davis and Proctor(2017)] Richard Davis and Chris Proctor. 2017. Fake news, real consequences: Recruiting neural networks for the fight against fake news .
- [Ferreira and Vlachos(2016)] William Ferreira and Andreas Vlachos. 2016. Emergent: a novel data-set for stance classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. ACL.