Automated Fact Checking

Information Retrieval & Data Mining Report

Giovanni Luca Favuzzi

COMPS_ENG / UCL
London England United Kingdom

ucabgfa@ucl.ac.uk

ABSTRACT

Automated Fact Checking, as well as Rumor Detection and Fake News Detection, has got a lot of traction in the recent years, due to the impact that such matter has on the society. The FEVER [6] challenge proposes to find an automated solution to these issues.

This report is going through my approach to the FEVER challenge, using data mining and machine learning techniques. The automated process can assess the veracity of a claim in three steps: Relevant Document Retrieval, Evidence Detection, and Truthfulness Classification.

KEYWORDS

Automated Fact Checking, Machine Learning, Data Mining, Logistic Regression, Deep Neural Network.

1 Introduction

The FEVER Dataset consists of more than *200,000* labelled claims along with a collection of roughly *5 million* documents and a vocabulary of more than *3 million* words. Being mainly in English, the corpus must follow the Zipf's power-law distribution.

1.1 Zipf's Law

The Zipf's law distribution $p(f)$, parametrized by rank **r**, exponent **sigma** and number of elements **N** is described by (2).

$$f(r;\sigma,N) = \frac{1/r^{\sigma}}{\sum_{i=1}^{N} 1/i^{\sigma}} \quad (1) \qquad p(f) = \alpha f^{-(1+\frac{1}{\sigma})} \quad (2)$$

As shown in Figure 1, fitting the data in logarithmic scale for both axes, with a simple straight line, using Root Mean Squared Error as Loss, leads to an exponent $\sigma = 1.0668$, which is near to the empirically expected value (for English) of *1.07*.

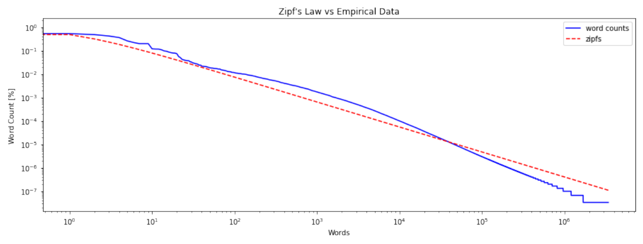

Figure 1: **Empirical distribution vs Zipf's law distribution.**

2 Document Retrieval

Having a well defined document retrieval routine is critical for the success of such project. Most of the documents are irrelevant to the provided claims in the train, dev and test sets, thus understanding which documents are relevant to which claims is key. For this purpose, three different document retrieval techniques are used: the first being a simple cosine similarity, the last two coming from a probabilistic approach.

2.1 Vector Space Document Retrieval

As already mentioned, the first approach consists of simple cosine similarity between **tf-idf** vector representations of both claims and documents. The term frequencies are computed on just the documents, while the inverse document frequencies are computed for both claims and documents.

2.*1.1 Formulation.* The Term Frequencies and Inverse Document Frequencies are formulated as in (3) and (4) respectively.

$$TF_{t,d} = \frac{f_{t,d}}{\sum_{\tau \in d} f_{\tau,d}} \quad (3) \qquad IDF_t = \frac{N_d}{n_t} \quad (4)$$

$$TF\text{-}IDF_{t,d} = IDF_t \cdot TF_{t,d} \quad (5)$$

Having as $f_{t,d}$ the count of how many times the term t is present int the document (or claim) d.
Having as N_d the number of documents in the complete collection and as n_t the number of times term t appears in the collection of documents.

The actual vector representation for one claim or document is a 1-dimensional vector long as the vocabulary length (in both docs collection and claims).

2.*1.2 Performances.* Such computation is rather fast since the derived matrices and vectors are sparse. In fact the cosine similarity between claims and documents is performed as a matrix multiplication between the claim matrix C with density of *0.0002%* and document matrix D with density of *0.0016%.*

$$S = C^T \cdot D \quad (6)$$
$$\text{Shapes:} \quad (N_c \times N_d) = (N_c \times N_w) \times (N_w \times N_d)$$

Having N_c as the number of claims, N_w as the number of words (in both claims and documents) and N_d as the number of documents.
Having as $\cdot$ the normalized inner product between vectors (which gives the same output as the cosine between vectors).

Moreover, C is composed of vector representations of the claims, which are its columns and D is made of vector representations of the documents, which are its columns.

2.2 Probabilistic Document Retrieval

The next technique derives from a different kind of approach: language models (**LM**s) and probabilities instead of vector representations in a multidimensional space.

For a LM, each document (***D***) defines a language (M_D); given a string (***s***), it is possible to define the probability of that string being generated by that document-language as $P(s \mid M_D)$.

For this project is employed a simple **unigram language model** (**ULM**): it assumes that each word in a sentence is independent from the others; such assumption is needed to save computational power, but is usually considered to be quite strong.

Moreover, a **query likelihood model** is built on top of the ULM: having claims as queries, the likelihood is defined in (7).

$$P(C \mid M_D) = P(c_1 \dots c_k \mid M_D) = \prod_{i=1}^{k} P(c_i \mid M_D) \qquad (7)$$

$$\log P(C \mid M_D) = \log P(c_1 \dots c_k \mid M_D) = \sum_{i=1}^{k} \log P(c_i \mid M_D) \qquad (8)$$

Having C as the claim and $c_1 \dots c_k$ as all the words in the claim.

The joint probability is a simple product between the single word probabilities due to the independence assumption of the UML. However, usually the logarithmic form in (8) is preferred to the original formula since it is easier dealt with.

The estimation of the probabilities follows the usual maximum likelihood estimation approach and, for a generic word ***w*** and a generic document ***D***, is given by (9).

$$M_D = P(w \mid D) = \frac{f_{w,D}}{|D|} \qquad (9)$$

Nonetheless, this model suffers from a major issue: in case in document D is absent of one single word c_i from the claim, the product is zeroed-out. Since this behavior hurts the performances of the model, a smoothing correction is required.

2*.2.1 Laplace Smoothing.* Such correction consists of adding one unit to the word count $f_{w,D}$ and then re-normalizing. The estimate, is given by the formulation in (10).

$$\text{Laplace}(w, D) = \frac{f_{w,D} + 1}{|D| + |V|} \qquad (10)$$

Having as V the vocabulary of words in the collection of documents.

The addition is interpretable as a uniform prior over every word in the vocabulary, however, usually it is considered to be too high.

2*.2.2 Lindstone correction.* To better model the estimate, the correction in (11) is applied with value $\epsilon = 0.5$.

$$\text{Lindstone}(w, D) = \frac{f_{w,D} + \epsilon}{|D| + \epsilon |V|} \quad \epsilon \in [0,1] \qquad (11)$$

2*.2.3 Performances.* Given that, now, the probability estimation for each word and for each document is non-null, a sparse implementation wouldn't have decent speed performances, since both the claim matrix C and document matrix D have density of *100%*.

For this purpose, instead, has been chosen an implementation with dictionaries. More specifically, the main idea behind the data-structures to be used is the **inverted index**, a dictionary that contains documents as keys, words as values and connects each document to the words in that document. Moreover, stop words removal has been applied, greatly increasing the speed performances of the algorithm.

Still, even with the Lindstone correction, this approach is not that performant, since it tends to favor longer texts.

2.3 Interpolation

In order to avoid giving higher relevance to longer texts, two interpolation methods are used.

2*.3.1 Jelinek-Mercer Smoothing.* Interpolating the maximum likelihood probabilities and the background probabilities of the document collection copes with such issue. Indeed, the TF-IDF scores and the ML estimates (with Laplace smoothing and Lindstone correction) are blended, parameterized by the hyper-parameter λ.

$$\text{JM}(t, d) = (1 - \lambda) \cdot \text{TF-IDF}_{t,d} + \lambda \cdot \frac{f_{t,d} + \epsilon}{|d| + \epsilon |V|} \qquad (12)$$

Still, since Jelinek-Mercer Smoothing doesn't explicitly considers the documents length, it can be optimized.

2*.3.2 Dirichlet Smoothing.* The interpolation parameter has to consider the document length N in order to nullify favoring longer documents. Thus, given a sample document, the lambda parameter is modeled as following.

$$\text{Dirichlet}(t, d) = \frac{\mu}{N + \mu} \cdot \textit{TF-IDF}_{t,d} + \frac{N}{N + \mu} \cdot \frac{f_{t,d} + \epsilon}{|d| + \epsilon |V|} \qquad (13)$$

Still there is presence of a hyper-parameter: μ is a shrinking parameter that defines the importance of longer documents. The higher the μ, the lesser the importance for longer documents. Generally, hyper-parameters are strongly dependent on the study case. For this project, a value of $\mu = 10$ is used.

3 Evidence Detection

Evidences, are needed to either prove or contradict the veracity of a claim. Moreover, document retrieval is not enough to precisely select the documents that are actually useful and, from those that aren't. That's why a proper classification is needed.

3.1 Dataset

The FEVER claim-set already contains the evidences sentences for both train-set and dev-set; such sentences represent positive samples for the task. Nevertheless, negative samples (non relevant sentences) are not provided.

To get non-relevant sentences it would be possible to use document retrieval techniques to retrieve all the useful documents and use the sentences (that are not already pointed out as evidence) as negative samples. This approach, however, is too computationally expensive.

The alternative approach, used in this project, consists of sampling as negative all the sentences, from the documents where the provided evidences are, that are not already provided as evidence. Such approach has the advantage of having negative sentences that are still somehow associated to the claim thus, leading the the creation of a dataset that is not trivial. Nonetheless, since such dataset could be too related to the claims, leading to a sort of overfitting on claim-set, another kind of negative sampling is used, called **super-negative sampling**: such sampling consists of extracting from the document collection completely random sentences.

The final negative dataset *Neg* consists of randomly subsampled sentences (with probability *p)* from the negative-samples dataset *NS* and the super-negative-samples dataset *SNS*.

$$Neg = NS \cdot p + SNS \cdot (1-p) \quad p \in [0,1] \qquad (14)$$

The final number of negative sentences is roughly equal to the number of positive sentences.

3.2 Representation

Given the positive subset and negative subset of sentences, in order to build a classifier, a proper numerical representation of the sentences is needed. Word Embedding is one of the most used approaches for such task: it is a transformation that takes as input the word string and outputs one dimensional vectors, with a fixed number of features ***k***.

GloVe [5] (Global Vectors for Word Representation) provides pre-trained word embeddings. These embeddings are trained on *2 billion* tweets, containing *27 billion* tokens, with a vocabulary size of *1.2 million* words. Moreover, GloVe provides word vectors with 25, 50, 100 or 200 features. For this project, *50*-feature vectors are used.

This said, GloVe doesn't explicitly provide sentence embeddings. In order to have an embedding for sentence ***s***, one simple idea would be averaging all the embeddings ***E(w)*** of the words in ***s***; however this way, without removing stop words, the sentence embedding would be mostly represented by such words. To cope with this issue all the embedding of the words in a sentence are summed together, weighted by their ***IDF*** (4), computed on the collection of documents and then normalized w.r.t. the length of the sentence N_s.

$$s = \frac{1}{N_s} \sum_{i:w_i \in s} E(w_i) \cdot IDF_{w_i} \qquad (15)$$

Now, given the sentence embeddings, the actual dataset for classification is computed, for each claim, as the difference between the claim embedding and the positive/negative sentence embedding related to that claim (if the sentence is super-negative, the subtraction is between claim embedding and the embedding of one of the random sentences).

3.3 Classification

Relevance classification is performed with a simple **binary Logistic Regression** model.

Logistic regression is a discriminative classification technique that focuses on learning the non-homogeneous Bernoulli distribution $P_Y(y = 1 \mid \underline{x})$, where ***y*** is the true target, coming from distribution ***Y*** and ***x*** is a data-point with ***k***-features.

Logistic regression applies the logit function, with input ***x*** and parametrized by the weight vector ***w***, to model the probability of one datapoint belonging to a certain class.

$$P_Y(y = 1 \mid \mathbf{x}) = \text{logit}_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + \exp\{-\mathbf{w} \cdot \mathbf{x}\}} \qquad (15)$$

Prediction, instead, is computed with threshold τ, which is usually put to *0.5*.

$$f_{\mathbf{w}}(\mathbf{x}) = \mathbf{1}[P_Y(y = 1 \mid \mathbf{x}) \geq \tau] \qquad (16)$$

The objective function to be maximized is given by the logarithmic likelihood estimate over all the ***n*** data-points in the train-set in (17) and (18). It is mostly known as **Cross-Entropy Function**.

$$\log(\text{likelihood}(\mathbf{w})) = \sum_{i=1}^{n} y_i \log(P_Y(y_i = 1 \mid \mathbf{x}_i)) + (1 - y_i)\log(P_Y(y_i = 0 \mid \mathbf{x}_i)) \qquad (17)$$

$$\log(\text{likelihood}(\mathbf{w})) = \sum_{n}^{i=1} y_i \mathbf{w} \cdot \mathbf{x}_i - \log(1 + \exp\{\mathbf{w} \cdot \mathbf{x}_i\}) \qquad (18)$$

Nonetheless, since it is better to work with minimization instead of maximization, the final **objective function** is expressed as:

$$\arg\min_{\mathbf{w}} \{-\log \text{likelihood}\} \qquad (19)$$

Solving logistic regression does not permit a closed form, however, the objective function is **convex**. This means that, even using a simple iterative method such as **Gradient Descent**, reaching the global optimum can be ensured.

Regularized (l_2) Gradient Descent for function $f(\mathbf{w})$ and iteration $k + 1$ is defined in (20).

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \gamma \nabla_{\mathbf{w}} f(\mathbf{w}^k) - \lambda \; \mathbf{w}^k \qquad (20)$$

Having γ as **learning rate** and λ as l_2 **regularization coefficient**.

For logistic regression, the final formulation is in (21).

$$\mathbf{w}^{k+1} = (1-\lambda)\ \mathbf{w}^{k} - \gamma \left(\frac{1}{1+\exp\{\mathbf{x}\cdot\mathbf{w}\}} - y \right) \cdot \mathbf{x} \tag{21}$$

3.*3.1 Parameter Modelling.* There are 4 hyper-parameters in this model: (1) learning rate γ, (2) regularization coefficient λ, (3) number of iterations K, (4) super-negative percentage. Due to logistic regression convexity, exists a learning rate small enough to make the iterative algorithm converge in a finite number of steps. On the other hand, if the algorithm doesn't converge due to a higher γ, the regularization coefficient is affecting the solution, as well as the number of iterations.

In order to choose the best parameters, alternated grid search is applied on all the hyper-parameters: the data-set is divided into train-set and test-set, so that the hyper-parameter configuration that affects the train-set is tested on the test-set.

3.*3.2 Performance Evaluation.* Understanding the performances of the learner is prime to assess whether the approach is correct or not. For classification there are a few popular metrics: **Accuracy**, **Precision**, **Recall**, **F1**. Each one is parametrized by the number of correctly/incorrectly, positive/negative classified points.

Each one of these metrics is easily formulated with the help of the **confusion matrix** that, in case of binary classification, is a 2 by 2 matrix.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive [TP]	False Negative [FN]
Actual Negative	False Positive [FP]	True Negative [TN]

Table 1: **Confusion Matrix for binary classification.**

$$\text{accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}} \tag{22}$$

$$\text{precision} = \frac{\text{TP}}{\text{TP}+\text{FP}} \tag{23}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{24}$$

$$\text{F}_1 = 2\ \frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}} \tag{25}$$

However, accuracy, precision and recall don't work that well when the dataset is unbalanced and F_1 is rather hard to interpret. To cope with such issue, the accuracy metric is slightly modified to get a better version named balanced accuracy.

$$\text{balanced accuracy} = \frac{1}{2}\left(\frac{\text{TP}}{\text{TP}+\text{FN}} + \frac{\text{TN}}{\text{TN}+\text{FN}}\right) \tag{26}$$

3.*3.3 Training.* The train-set contains roughly *280000* samples of which *54.5%* positive and *45.5%* negative. *80%* of the set is used for training and the remaining *20%* for validating the hyper-parameters. Training the model shows a strong dependence on the learning rate, which is shown in Figure 2.

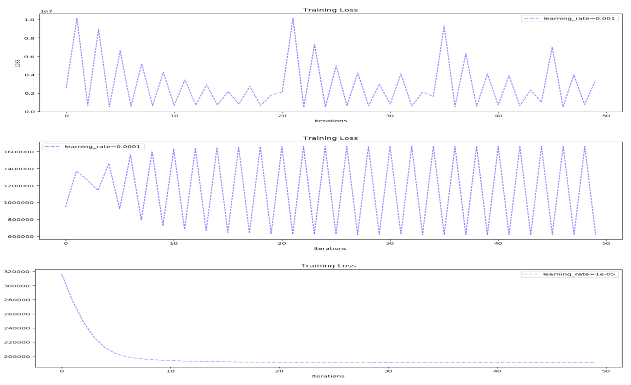

Figure 2: **Logistic regression train loss with learning rates (top to bottom) [0.001, 0.0001, 0.00001]**

Clearly, if the learning rate is not small enough, the model cannot reach the global minimum even with infinite iterations and the loss starts oscillating. Instead, with a learning rate of *0.00001* (third panel of Figure 2) the model reaches the optimum on both train and test, both with a similar fashion.

At this point, if the training converges, regularization coefficient λ and number of iterations K seem to not really impact the performances.

The super-negative percentage of the train-set, instead, seems to always hurt the performances of the model, thus is set to *0.0%*. Nonetheless, the next paragraph shows how this is not completely true. Table 2 shows the final performances on the test.

	accuracy	precision	recall	F_1	balanced accuracy
test set	0.4889	0.3334	0.7866	0.4683	0.5781

Table 2: **Logistic Regression on the test (*20%* of the train).**

3.*3.4 Testing.* Testing on part of the original train-set, however, is clearly not the correct way of assessing the true performances since the train-set is made up appositely for computational matters and doesn't express a real scenario. In order to truly understand the performances, a such scenario must be simulated: from a subset of *10* claims, from the dev-set, are retrieved the associated relevant documents using TF-IDF vector representation retrieval; finally the model is tested on the retrieved sentences.

This dataset consists of *306* samples where the *2.62%* is positive and the remaining *97.38%* is negative. The model is capable of reaching balanced accuracy $= 0.5462$; the other metrics are meaningless for such an imbalanced dataset.

This time, the percentage of super-negative samples has been increased to 50%, leading to better results. This shows how, testing on part of the train, doesn't point out the generalization loss that the learner suffers from when used against quite different kind of data. Still, the model clearly struggles classifying correctly the labelled development sub-set, but, such a result is rather expected since logistic regression is an extremely simple model.

Also, as a matter of facts, this result can be interpreted as a meaningful baseline: sentence relevance is not an easy task for a learner (given such representation of the text); more complex approaches are needed.

4 Sentence Truthfulness

Truthfulness classification is the core component of Automated Fact Checking, nonetheless it is also the hardest one.

It is still binary classification: for each claim, given the associated evidence (retrieved in the previous stage), the learner must be able to understand if the claim is true or false checking against such evidence.

4.1 Dataset

The FEVER dataset provides, for each claim, a 3-way labelling: either ‘SUPPORT’ or ‘REFUTES’ if there exist evidence in the document collection and the claim is respectively true or false; ‘NOT ENOUGH INFO’ if there is no evidence at all. Each claim corresponding to ’NOT ENOUGH INFO’ is filtered out to provide a stronger learning.

At the end of the procedure, the dataset consist of only claims with 1 or more evidences with *70.18%* positives (‘SUPPORT’) and *29.82%* negatives (‘REFUTES’), showing a strong class imbalance.

4.2 Representation

In order to have proper data-points, the chosen representation, shown in (27) for both claims and relevant sentences is similar to the representation for sentence relevance.

$$s = \frac{1}{N_s} \sum_{i:w_i \in s} E(w_i) \tag{27}$$

One substantial difference is that the IDF weighting is gone: this alternative is chosen to keep the importance to those stop words that are still meaningful for sentence truthfulness, such as *‘not’*. Moreover, instead of subtracting the relevant sentence embedding from the claim embedding, the two sentences are concatenated, resulting in a *100*-features *1*-dimensional tensor. This embedding is chosen because of the power/complexity of the learner that is used, which is better described in the next paragraph.

4.3 Classification

Neural networks are known to be the state of the art in most of the Machine Learning tasks, thanks to their huge number of possible architectures and flexibility to different problems; this is why the learner that is used is a Neural Network.

4.3.1 Architecture. The first architecture built is a simple **Multi-Layer-Perceptron** (**MLP**), needed to get baseline performances. The network is made of a first input layer with *100* units, a hidden layer with *500* nodes and the final output layer with only *1* node.

For the hidden layer, the activation function is a $\mathrm{relu}(\mathbf{x}) = \max\{0, \mathbf{x}\}$, popular choice for hidden layer thanks to its property of coping with the vanishing gradient. Nonetheless, for generalization purposes, *20%* dropout is applied on the hidden layer. The output layer, instead, is activated with a sigmoid function, special case of the logit function, which squeezes the input near 1 or 0.

The MLP is trained with **Adam** routine with learning rate of *0.0001*, no weight decay and $\beta = 0.9$, validating on *20%* of the train-set and using a mini-batch of *128* samples and optimizes the binary cross-entropy loss function.

The architecture of the model is presented in Figure 3.

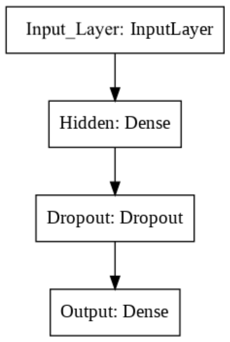

Figure 3: **MLP Architecture**

4.3.2 Performances. The MLP is clearly a simple architecture, thus reaching poor performances is rather expected. Training the model for *200* epochs results in the accuracy shown in Table 3, but most importantly in a **balanced accuracy** of *57.34%* on the dev.

	train	val	dev
accuracy	0.7657	0.3465	0.5644

Table 3: **MLP baseline performances after 200 epochs.**

5 Related Work

Nowadays, Automated Fact Checking (AFC), Rumor Detection (RD) and Fake News Detection (FND) are well known and studied problems. Indeed, seen all this traction, a lot of work has already been done in the recent years.

5.1 Fake News Detection

FND has been treated with a Hybrid model in [1], made of 3 sub-models: the first one, **Capture**, is based on the response and text of a fake news, the second one, **Score**, analyzes the source of the news and the final one, **Integrate** integrates the first two modules and computes the classification (fake of true).

Capture uses a Recurrent Neural Network (made of LSTMs) which receives the temporal user engagement on the news and a **doc2vec** representation of its text and generates a low dimensional representation of the news.

Score, instead, uses a Deep Neural Network and **Singular Value Decomposition** to build an implicit user-graph that describes the propensity of one user to take part in a group which promotes one particular news source. The model outputs a score for each user that indicates the probability of that user interacting with a promiscuous news source.

Integrate ensembles the output from the first modules, concatenating the two and then classifies as true or false. It is trained following a regularized **Cross-Entropy** loss function.

The general architecture of the model is presented in Figure 4.

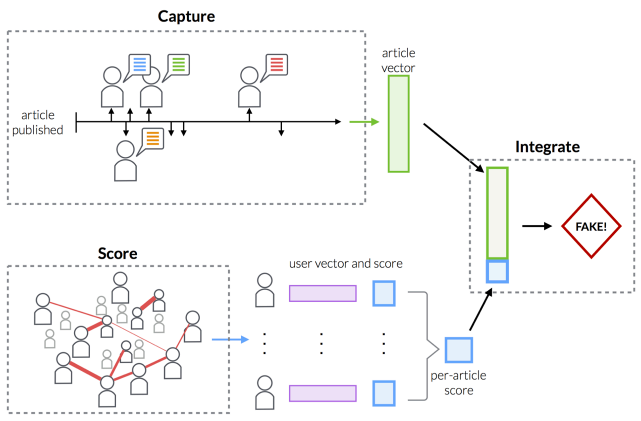

Figure 4: **Capture-Score-Integrate model architecture.**

5.2 Rumor Detection

In [2] RD was studied and tackled with Recurrent Neural Networks. The experimented **RNNs** are built using two different implementation with **LSTMs** a **GRUs**.

The models take as input the rumor text, represented as the concatenation of vocabulary terms in TF-IDF vector representation. The input is then embedded using the word-embedding matrix **E** and then goes through the recurrent layers.

The best performing model is a 2-layer GRU RNN, trained with AdaGrad, of which architecture is presented in Figure 5.

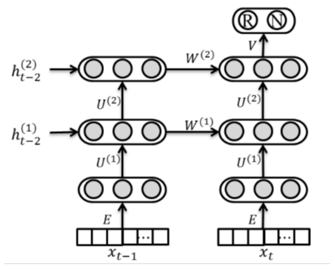

Figure 5: **Two-layer GRU + Embedding Matrix E.**

5.3 Automated Fact Checking

Pure AFC is faced in [3]. The proposed workflow consists of generating a query from a claim in input, the query is then automatically searched on the web with the help of a **crawler** and the most important snippets/sentences are retrieved from the chosen web pages. This input is fed to a **Deep NN** enhanced with **LSTM** encodings to embed both claims and relevant sentences together. Finally, the lower dimensional embedding is fed to a **SVM** on top of the net that classifies the veracity of the claim.

More specifically, *3* major features are computed to to choose the best snippets: similarity between claim and snippet, using *(i)* **tf-idf** cosine , *(ii)* **cosine** over embeddings (as averaged **Glove** word embeddings) and *(iii)* **containment**. For each claim-snippet pair, the features are averaged and the, the top-10 scoring snippets are taken. These snippets are finally fed to the LSTM layer to be embedded with the associated claim.

The architecture of such model is presented in Figure 6.

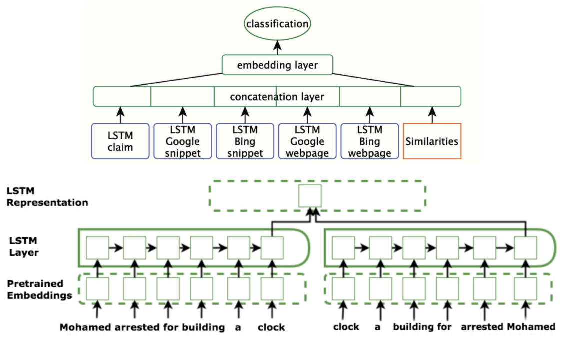

Figure 6: **LSTM Embedding Architecture + SVM Classification on Top.**

Each one of these models uses an unsupervised approach to generate embeddings and exploits the RNN ability of capturing temporal relationships to model the sentence embeddings.

Almost none of the models, uses a directly supervised approach to output the sentence embeddings. This project explores this approach using Metric Learning, recently popular technique which learns a new enhanced input space from labelled data.

6 Metric Learning

Metric or **Similarity Learning** belongs to the supervised branch of Machine Learning and can be used either for regression, classification or even learning to rank; that is why it has a variety of applications, ranging from image clustering to recommendation systems.

Its main goal is to learn a distance function, from the provided labelled samples, that best describes how distant two new samples really are. Such function should respect 4 assumptions:

1. Non-negativity
2. Symmetry
3. Triangular Inequality
4. Indistinguishable's Identity

Most of the time, though, the implemented algorithms ignore the fourth assumption, ending up learning a pseudo-metric.

Given k-dimensional input samples $\mathbf{x}_1, \mathbf{x}_2$ and a symmetric, positive-definite matrix $\mathbf{W}$, the distance is expressed in (28):

$$D_{\mathbf{W}}(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^T \mathbf{W} (\mathbf{x}_1 - \mathbf{x}_2)} \tag{28}$$

The distance function is trained to learn the covariance matrix $\mathbf{W}$ and maximize the distance between positive ($y = 1$) and negative ($y = 0$) samples. Nonetheless, learning this distance lets us transform the input space, ending up with new embeddings for a lower dimensional space.

6.1 Local Fisher Discriminant Analysis

In this paragraph is going to be explained the core concept of LFDA, however, in [4] is granted a deeper explanation.

6.1.1 Idea. **LFDA** learns the covariance matrix through a generalized eigenvalue problem, having as target the maximization of class separability as well as the preservation of the multimodality of the distribution; nonetheless, this approach keeps the results of Fisher Discriminant Analysis preserving the original local neighborhood structure.

In facts, given a matrix $\mathbf{X} \equiv (\mathbf{x}_1 | \ldots | \mathbf{x}_n)$, made of n samples, each one with label $\mathbf{x}_i \in \left\{1 \ldots C\right\}$, where C is the number of classes and each sample is d-dimensional, we can define the LDFA transformation with matrix $\mathbf{T}_{LFDA}$.

$$\mathbf{z}_i = \mathbf{T}_{\text{LFDA}}^T \cdot \mathbf{x}_i \qquad \mathbf{z}_i \in \mathbf{R}^m \,|\, m < d \tag{29}$$

The transformation matrix is obtained mixing the next two approaches and is going to output a lower-dimensional embedding for the input data.

6.1.2 Fisher Determinant Analysis. **FDA** defines the **scatter matrices**, respectively between classes and between a class, as:

$$\mathbf{S}_{cs} = \sum_{i=1}^{C} n_i (\mu_i - \mu) \cdot (\mu_i - \mu)^T \tag{30}$$

$$\mathbf{S}_c = \sum_{i=1}^{C} \sum_{j|\mathbf{x}_j \in c_i} n_i (\mathbf{x}_j - \mu_i) \cdot (\mathbf{x}_j - \mu_i)^T \tag{31}$$

Having $i \in \left\{1 \ldots C\right\}$ as the id of class c_i, as j as the id of sample $\mathbf{x}_j$ belonging to class c_i, as μ_i the mean of class c_i, as μ the mean of all the data and as n_i the number of samples in class c_i.

The general eigenvector problem can be expressed with the equality $\mathbf{S}_{cs} \cdot \phi = \lambda \mathbf{S}_c \cdot \phi$. Solving the problems leads to the eigenvectors $\left\{\phi_i\right\}_{i=1}^{d}$ and associated eigenvalues $\left\{\lambda_i\right\}_{i=1}^{d}$, which are ordered in descending order. FDA, however, is not working well with multi-modal distributions.

6.1.3 Locality-Preserving Projection. **LPP**, instead, employs the affinity matrix $\mathbf{A}$ to describe the similarity between samples, so that the similarity will be grater for nearer samples and smaller for further samples.

Now, considering a matrix $\mathbf{D}$, that respects the constraint in (32),

$$\mathbf{D} \in \text{Diagonal}(n \times n) \,|\, D_{ii} = \sum_{j=1}^{n} A_{ij} \wedge \mathbf{T}^T \mathbf{X} \mathbf{D} \mathbf{X}^T \mathbf{T} = \mathbf{1} \tag{32}$$

considering the eigenvector problem in (33), the LLP transformation matrix is given by its eigenvectors.

$$\mathbf{X}(\mathbf{D} - \mathbf{A})\mathbf{X}^T \phi = \gamma \mathbf{X}(\mathbf{D} - \mathbf{A})\mathbf{X}^T \phi \tag{33}$$

$$\mathbf{T}_{LPP} = (\phi_{d-m+1} | \ldots | \phi_d) \tag{34}$$

6.1.4 Transformation. LFDA mixes these two approaches, weighting the scatter matrices $\mathbf{S}_{cs}, \mathbf{S}_c$ with two affinity matrices $\mathbf{A}^{cs}, \mathbf{A}^{c}$.

$$\mathbf{S}_{cs} = \frac{1}{2} \sum_{i,j=1}^{C} A_{ij}^{cs} (\mathbf{x}_i - \mathbf{x}_j) \cdot (\mathbf{x}_i - \mathbf{x}_j)^T \tag{35}$$

$$\mathbf{S}_{c} = \frac{1}{2} \sum_{i,j=1}^{C} A_{ij}^{c} (\mathbf{x}_i - \mathbf{x}_j) \cdot (\mathbf{x}_i - \mathbf{x}_j)^T \tag{36}$$

Finally, the transformation matrix is obtained maximizing the objective function in (37).

$$\mathbf{T}_{\text{LFDA}} = \arg\max_{T} \left\{ \left(\mathbf{T}^T \mathbf{S}^c \mathbf{T}\right)^{-1} \mathbf{T}^T \mathbf{S}^{cs} \mathbf{T} \right\} \tag{37}$$

6.2 Input Space Embedding

Metric learning is used as a preprocessing step for several scenarios, as well as this one. This step transforms the input space into a lower dimensional space, maximizing the the distance between classes and keeping only the most important features, thus maximizing the overall partial variance. This transformation makes it easier for a neural network to classify the data, thus reaching higher performances.

The LDFA Metric Learning algorithm has been fitted on *13%* of the data (roughly *20,000* data-points), for computational issues, using the best *60* features out of *100*.

In fact, performing PCA on whole the dataset, we can see (Figure 6) how the explained variance is kept to more that *90%* using *60* features.

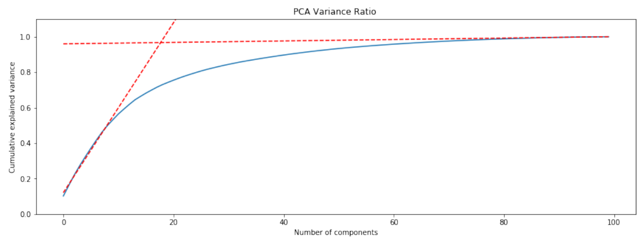

Figure 6: **PCA Variance Ratio and 10% Tail Fitting.**

Even though the knee of the function is around *20* features, that would keep less than *80%* of the variance, which could translate in important information loss.

Once fitted on *13%* of the train, the transformation is tested on another (novel) part. The result is presented in Figure 7.

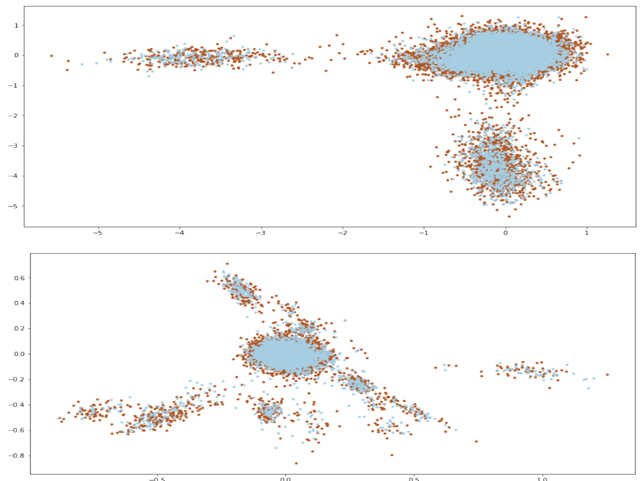

Figure 7: **Train-set input space transformation: original [TOP], transformed [BOTTOM], restricted to only 2 features.**

7 Neural Network Enhancement

7.1 Architecture

The architecture to be used along with the new embeddings is a **Dense Deep Neural Network**, with descending number of nodes. Both convolutional and recurrent architecture are avoided, since the true meaning of such operations on these embeddings is hardly interpretable. The model is composed of *10* layers starting with node-degree of *1024* and halving every two layers (except for first and last layer). All the hidden layers are activated with relu and for regularization purposes to each hidden layer is imposed a dropout of *10%* of the links.

7.2 Performance

This time too, the network is trained to optimize binary cross-entropy loss for *200* epochs using the Adam routine with learning rate of *0.0001*, no weight decay and $\beta = 0.9$, validating on *20%* of the train-set and using a mini-batch of *128* samples; nonetheless, an early stopping callback is applied to avoid overfitting. Table 4 presents the overall performances.

	train	val	dev
accuracy	0.9415	0.8979	0.5613

Table 4: **Performances on train, val (20% train) and dev sets.**

On the dev-set it reaches balanced accuracy $= 56.70\,\%$.

The new network, with new embeddings is much stronger than the MLP since it can reach far better accuracy on train and validation, however, the model is not behaving so well on the development set, where it seems to be even worse than the MLP. Such weird behavior shows clear lack of generalization.

7.3 Future Work

Seen the last performances, the input given to the LFDA algorithm must be enhanced to have proper learning.

In specific, it would be better to have each sentences (claims and relevant-sentences) represented as a matrix of concatenated word2vec embeddings (thus without averaging the embeddings); just like in [3] these embeddings must be processed by a Recurrent Neural Network, able to capture the dependency on time, thus, capturing the order of the words in the new embedding.

Now, these must be the embeddings given as input to the LFDA, which output embeddings are then given in input in the same dense multi-layer net, described here.

ACKNOWLEDGMENTS

This research was performed by Giovanni Luca Favuzzi at the University College London as a project for the Information Retrieval and Data Mining module.

REFERENCES

[1] CSI: A Hybrid Deep Model for Fake News Detection; CIKM'17 November 6-10, 2017, Singapore, University of Southern California.
[2] Detecting Rumors from Microblogs with Recurrent Neural Networks; The Chinese University of Hong Kong; Qatar Computing Research Institute, Hamad Bin Khalifa University; Graduate School of Culture Technology, Korea.
[3] Fully Automated Fact Checking Using External Sources; Sofia University "St. Kliment Ohridski", Bulgaria; Qatar Computing Research Institute, Qatar.
[4] Local Fisher Discriminant Analysis for Pedestrian Re-identification; Sateesh Pedagadi, James Orwell; Sergio Velastin; Boghos Boghossian.
[5] GloVe: Global Vectors for Word Representation; Jeffrey Pennington, Richard Socher, Christopher D. Manning Computer Science Department, Stanford.
[6] FEVER: a large-scale dataset for Fact Extraction and VERification, James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal Department of Computer Science, University of Sheffield Amazon Research Cambridge