Distinguishing Ukrainian and Russian social media

Taras Svytsun
Mykhailo Bondarenko
Yaroslav Brochenko

May 2022

Abstract

In the light of Russia's full-scale invasion of Ukraine, the need for deeper study of the Russo-Ukrainian media space became apparent. Many studies that are now concentrating on such analysis lack a reliable and scalable (to thousands of posts) method for distinguishing Russia's propaganda and/or bot-generated content from the russian-language sources in Ukraine. This paper aims to make initial advancements into making an accurate and efficient model for such classification.

In particular, this research concentrates on the utility of distinguishing Russian and Ukrainian social media paragraphs with Latent Semantic Analysis (LSA). To this end, it proposes an optimal classification model based on LSA, performance- and accuracy-testing its various configurations based on popular semantic preprocessing methods. It then compares its performance to other popular Machine Learning classifiers, and makes a verdict on whether LSA classifiers show sufficient potential for further study.

Keywords: Propaganda classification; Latent Semantic Analysis; Singular Value Decomposition; Principal Component Analysis; Logistic Regression; Naive Bayes Classifier; K-nearest neighbours; Social media moderation.

Dictionary

LSA - Latent Semantic Analysis
SVD - Singular Value Decomposition
PCA - Principal Component Analysis
kNN - k-nearest neighbours
Tf - term frequency.
Idf - inverse document frequency.
$n-gram$ - contiguous sequence of n words in the text.
m - vocabulary length (number of unique tokens)
n - number of telegram posts.
k - the parameter of truncated SVD.
X - vectorized sparse m x n matrix.
$Corpus$, $dataset$ - a set of documents.
$Document$, $text$, $post$, $paragraph$ - separate set of words with punctuation.

Introduction

In the previous studies and literature, LSA received much attention for text categorisation [KP15] and topic extraction, and seldom applied to classification tasks, such as spam detection [LHu+10]. Thus, the first objective of this paper is to study whether LSA-based classification can perform well on more sophisticated classification tasks, such as a classification task related to separating Russian propaganda from the Ukrainian sources.

Although the Russian and Ukrainian sources have developed very strongly separated narratives after the beginning of the full-scale war, it can sometimes be difficult even for humans to understand whether an article is written with an intention to serve the interests of Russian propaganda. This is largely explained by the abundance of anonymous sources and propaganda bot networks that serve a huge role for the spreading of propaganda. Various social media platforms, such as Facebook, declared the fight against Russian propaganda [Nix22], but failed to be effective, on some occasions missing more than 80% of such posts [Pau22].

Thus, a deliverable of this research is a scalable and effective method for such classification, which needs low computational power relative to large Deep Learning methods. This research will show that the proposed method won't be able to outperform the latter approach in terms of accuracy, but only one matrix multiplication by a sparse feature vector is needed to make a prediction in the LSA-based model, which has a complexity of $O(k \cdot t)$, where k is the number of the kept SVD singular values, and t is the number of terms kept in the dataset.

The second objective of this paper is to test and compare different model architectures of LSA-based classifiers, and propose a model architecture that would perform best in the scope of the discussed task. This research will try different preprocessing approaches to find their effect on the efficiency of the model, making sure that the label data doesn't leak into the feature space and the data itself is of high quality. It will also try different model architecture approaches, trying different distance measures for similarity of the paragraph-vectors and different cluster-assigning techniques. The third and final objective of this research is to compare the LSA-based classification to other popular machine learning approaches, and thus encourage more research in the topic. The primary goal was to try out as many as possible different LSA classifiers and to try overcoming such classic ML algorithms as the k-nearest neighbours, Naive Bayes and Logistic regression approaches.

This research answers the following questions:

1. What are the most efficient preprocessing techniques?
2. What is vectorization and how to choose the best?
3. How to apply standard dimensionality reduction techniques from linear algebra (SVD and PCA) to the formulated problematics?
4. What is the proposed model, how to train it and how to tune the parameters? What is the tradeoff between accuracy and time complexity?
5. What ML classification techniques worked? What were the reasons they did or did not fail?

Preprocessing and Data

Data

For this study, a labelled dataset of posts was collected from public Ukrainian/Russian Telegram channels, with the publication date ranging from the 24th February to 28th of May. The median subscriber count of the collected channels was approximately set to 300K, with a minimum of 10K subscribers and a maximum of $1,193$K. A total of $146,283$ posts were collected initially, with $138,059$ posts passing the preprocessing stage. In total, 40.1% of the posts were of Ukrainian origin and 59.9% were of Russian origin.

Cheat-words

The first task for the preprocessing stage was to remove the so-called cheat-words from the dataset. A word is labelled as a cheat-word if it immediately indicates the origin of the post, making the prediction process trivial and rendering the model unusable on previously unseen data. Such cheat-words frequently include channel titles or tags, social media links, or other marketing text unique to the source telegram channels. To automatize the exclusion process of such words an algorithm was written to count word frequencies by channel, and it was empirically established that if a word is present in more than 70% of channel's posts, then it can most probably be marked as a cheat-word. For the purposes of finer filtering, this threshold was set to 50%, and non-cheat words were manually removed from the exclusion list by a human operator.

Short posts

The length of the post can impact the accuracy of the model dramatically. If a post is short and uninformative, it can make the dataset noisier, making the resulting model less accurate. Moreover, it is easier for the model to make predictions when more information is present. Thus, we experiment with various thresholds for the shortest allowed post. We use the symbol length of the post for this purpose.

Filtering posts in Ukrainian

Since the database consists of russian-language posts from both Ukrainian and Russian sources, and the russian-language Ukrainian channels sometimes make posts in Ukrainian, an additional filtering was applied to the database to filter-out ukrainian-language posts. Since the post length was already limited from below, a simple Ukrainian charset filtering was sufficient.

Lemmatization

Lemmatization is a process of stemming words to get to their common base form by removing word endings or other grammatical constructs which don't contribute to a word's sense. Applying lemmatization reduces the number of words in two-fold, which in turn increases the performance of SVD with complexity of $O(m \cdot n \cdot k)$ by 2 times. It also positively affects the accuracy of all of the tested models by an average of 1%, with an exception made for the naive Bayes classifier.

Models

Text vectorization and transformation pipelines

After the preprocessing stage, the researcher ends up with a vector containing ordered sets of words in character form. To apply any of the numeric machine learning methods the data needs to be transformed into numerical form. This is when a task to select the most appropriate vectorizer arises. When dealing with NLP tasks it is common to create a corpus vocabulary, an unordered set of unique terms. The crucial part is to decide which value should be assigned to each vocabulary unit in each of the documents. After this operation, the expected result is m x n sparse matrix, where n marks the number of documents and m is the length of the vocabulary. Usually, $m > 50000$, $n > 10000$. The three main approaches for achieving this matrix are outlined below.

CountVectorizer

Also called Bag-of-words. Simply counts the word frequencies in each text. As mentioned before, the number of unique words in the text dataset is in the tens of thousands, but the average document contains 80 words, so the output of such vectorisation is a sparse matrix mostly filled with zeros.

Binary CountVectorizer

Works identically to the previous one, but marks word occurrences in a boolean way. Namely maps each text to a binary ordered list of features, with ones for the words that are present in the text and zeros otherwise. In our experiments, the binary vectorizer generally works better than the frequency vectorizer. This may be explained by the fact that if a word is present, say, 3 times in the text, it doesn't generally mean that it is thrice as important as if it was mentioned just once.

TfidfVectorizer

Term-frequency inverse document-frequency vectoriser consists of two parts. The first part, the term frequency tf, is the other name for CountVectorizer;

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t\prime \in d} f_{t\prime,d}}$$

The second part, inverse document frequency idf, could be thought of as some kind of normalizer;

$$idf(t,D) = log\left(\frac{\mid D \mid}{\mid \{d \in D : t \in d\} \mid}\right)$$

Its idea is based on the fact that if some arbitrary word is present in most of the documents, then it doesn't contribute much to the uniqueness of the studied paragraph, but if a word is generally infrequent, its presence is thus much more meaningful and should be associated with a higher value. Think of the words *bed*, *table*, *door* in a text paragraph about furniture. It is natural to assume that they strongly define the topic of the text, unlike common words like *the*, *you* or *and*.

The main drawback of mentioned approaches is that they disregard the word order. For explanatory purposes, consider 2 similar sentences: “I am not angry, everything is alright” and “I am angry, everything is not alright”. They bear an opposite meaning but have the same set of words, and only the position of the word “not” defines the sense behind the sentences. Another disadvantage is that using such an encoding, there is no clear way of comparing the word or paragraph meaning in their initial form (as will be shown shortly, LSA partially solves this problem). There is, however, an excellent approach, called *word2vec* [al13], which was invented and fully discovered by Mikolov et. al. in 2013, and is using a distributed representation of a word. Each term is associated with a weight vector consisting of its characteristics. The benefit of such a method is that a word is spread across all of the entries in the vector, and doesn't just happen to be a single non-negative document entry. Moreover each entry of the vector contributes to the definition of many other words, and addition/subtraction is actually meaningful. A remarkable example, using this method would be:

$$vect(King) - vect(Man) + vect(Woman) = Vect(Queen)$$

However, this method is built on neural networks, which are not in the scope of this paper, and are reserved by authors of the study for further publications.

Latent Semantic Analysis and auxiliary components

After the vectorizing part, the goal is to deal with the obtained $m \times n$ sparse matrix. Most of its elements are zeroes and thus it is not optimal to store all its components. This leads us to the idea of dimensionality reduction.

Singular Value Decomposition

Consider the following rectangular matrix X:

$$X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \cdots & x_{m,n} \end{bmatrix} \tag{1}$$

SVD can be used for approximating high dimensional data through its dominant components. Unlike eigendecomposition (also referred to as spectral decomposition), SVD always exists. In short, SVD is defined as

$$X = U \cdot \Sigma \cdot V^T,$$

where U and V are orthogonal matrices ($U^T \cdot U = U \cdot U^T = I$) usually referred to as left and right singular vectors. Σ is a $min(n,m) \times min(n,m)$ diagonal matrix with singular values on its main diagonal. The notion of singular values is tightly bound with eigenvalues. Briefly, the first, largest, singular value corresponds to the square root of largest eigenvalue of symmetric positive semidefinite nxn matrix $X^T \cdot X$ (or $X \cdot X^T$, which possesses the same set of eigenvalues as $X^T \cdot X$).

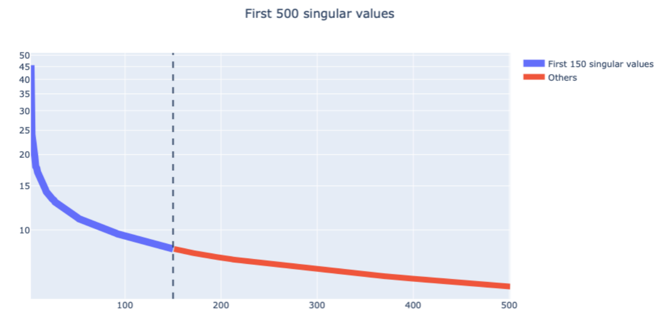
However, if X is a m x n matrix of rank r ($r < m$, $r < n$), then the diagonal matrix can be smaller. In fact, X can be represented as sum of r rank one matrices in the form of $v_j \cdot v_j^T$. In other words,

$$\begin{aligned} X &= U \Sigma V^T \\ &= \sigma_1 \cdot u_1 \cdot v_1^T + \ldots + \sigma_n \cdot u_n \cdot v_n^T \\ &= \sum_j^n \sigma_j \cdot u_j \cdot v_j^T. \end{aligned}$$

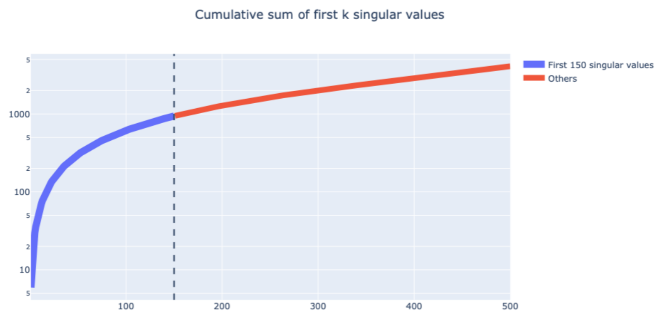
In its standard form, to make the U matrix to be of size $m \times min(m,n)$, SVD also ads “arbitrary” eigenvectors so that they are orthogonal to the present ones, but they carry less meaning than the first r ones, which correspond to the eigenvectors of $X \cdot X^T$.

The last fact leads to the notion of reduced SVD, which leaves only r columns in U and V, which are chosen in correspondence to the k largest singular values in the Σ matrix, with only k of them, correspondingly, being kept in Σ. This method saves some memory but also operates more efficiently, since the resources for the computation of all other vectors and singular values is dropped. This introduces a truncated SVD.

Usually, singular values decay rapidly, so taking the largest k of them and zeroing-out the rest is often preserving the properties of the matrix efficiently.



This plot demonstrates magnitude singular values in decreasing order



This picture provides an understanding of how much information is kept in the first few singular values

It is important to remember that, although $X \cdot X^T = I$, $X^T \cdot X$ is not the identity matrix anymore. Finally, last remark: in our context, it is convenient to call matrices U and V^T as $Terms$ and $Documents^T$ matrices, which form term and document space correspondingly.

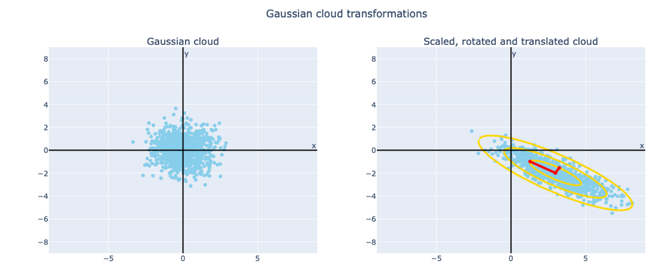
Thus, a vector can be selected from either of $Terms$ or $Documents^T$ matrices and it would correspond to a certain word or paragraph, containing information about the word's or paragraph's meaning. An important property of such vectors is that the distance between them can be measured (cosine distance), and thus it can be determined whether the documents or words are in any way similar (cosine similarity), provided a big-enough text corpus. The features of those vectors are thought of as topics, and Σ is then thought of as a topic importance matrix. The measure of paragraph vector similarity is a core notion to the LSA-based classification model built in this research.

Principal Component Analysis

Recall that Term and Document matrices are orthogonal. As a linear transformation, multiplying by orthogonal matrices preserves the norms and inner products of vectors, which could be thought of as rotation or reflection. In other words,

$$|| Ax ||^2 = (Ax)^T \cdot Ax = x^T \cdot A^T \cdot A \cdot x = x^T \cdot I \cdot x = ||x||^2.$$

In the picture below the reader may see the principal axis of the data with $\{1, 2, 3\}$ sigma confidence intervals.

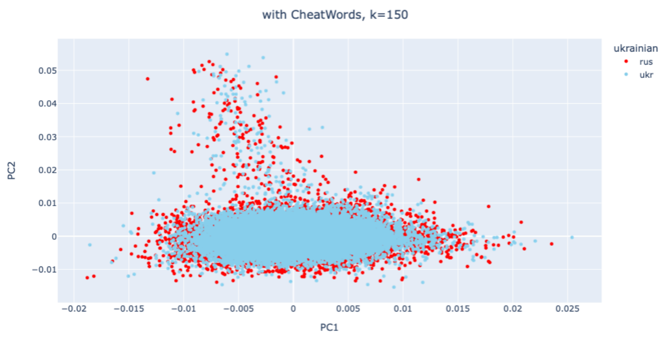


PCA with confidence intervals

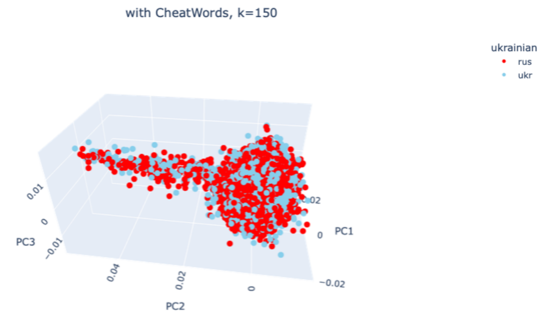
Recall the SVD decomposition equation, $X = Term \cdot \Sigma \cdot Document^T$. Let's now multiply it by $\Sigma^{-1} \cdot Term^T$:

$$\Sigma^{-1} \cdot Term^T \cdot X = \Sigma^{-1} \cdot Term^T \cdot Term \cdot \Sigma \cdot Document^T = Document^T.$$

This is a process of projecting matrix X onto the r of its principal axis. Applying $Term^T$ to X rotates X and then Σ scales axis in the direction of the biggest variance. Consider the following example of the matrix X, projected onto 2 and 3 principal exes, with $k = 150$.

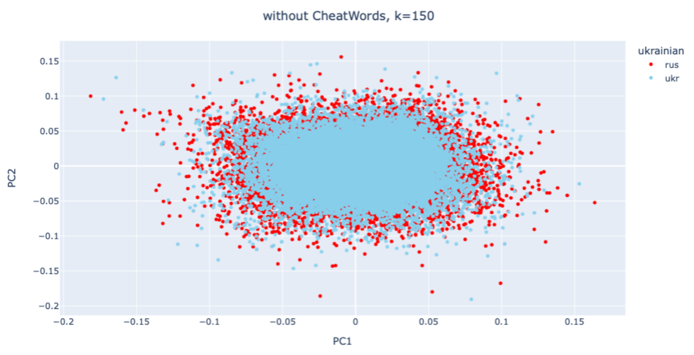


With cheat words, $k = 150$

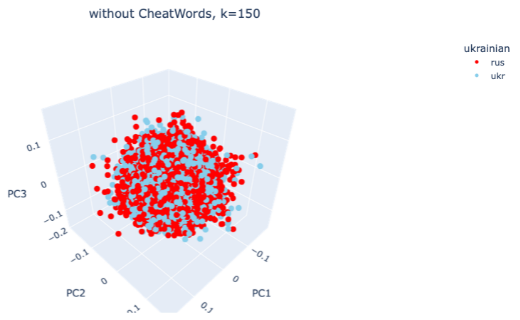


With cheat words, 3D, $k = 150$

These 2 and 3 principal components were purposefully derived from a $tfidf$ matrix without filtering-out the cheat words, to show that the data in this case can be clustered more easily than if the cheat-words are removed.



Without cheat words, $k = 150$



Without cheat words, 3D, $k = 150$

On this plot, it can be clearly seen that the data is more tightly-clustered. To sum up, PCA is used to find k orthogonal directions, in which the dataset X has the biggest deviation, take a projection on them and ignore the other $m - k$ dimensions.

LSA model

The training stage consists of calculating the *Document* matrix"

$$Document^T = \Sigma^{-1} \cdot Term^T \cdot X.$$

This matrix is obtained in a natural way after computing the SVD. Then the labelled data is used to find the paragraph-cluster centres for both of our classes: ["*Ukrainian*", "*Russian*"]. The cluster centres are k-dimensional vectors, with k being the parameter of SVD. Simply put, the resulting two vectors are the coordinates of the centres of mass of Ukrainian and Russian "clouds" correspondingly. The prediction/validation stage thus works as follows: project the X_{test} matrix of vectorised test-set documents onto terms space and scale using the Σ matrix:

$$Document^T_{pred} = \Sigma^{-1} \cdot Terms^T \cdot X_{test}.$$

Then, for each column of the $Document_{pred}$, find the distance to the Ukrainian and Russian cluster centres. This distance would then indicate whether a post is "closer" to being of Ukrainian or Russian origin. Finally, classify each column as Ukrainian if it is closer to Ukrainian centre, Russian – otherwise. There is still one uncertainty: how exactly to measure the distance between the projected document and the Ukrainian (or Russian) centre. The reader can find the discussion of this topic in the "Results" section.

Results

Best estimated parameters

After the first test run, the described model showed an accuracy of 70%. But, it's obviously not the full capability of the LSA model. Thus, in this section the reader may find explanations and comments on how the model performance can be improved.

Lemmatizing or not

The experiments showed that lemmatizing after cheat words removal upgraded the model performance by $0.5 - 1\%$. The most significant increase was for the CountVectorizer – from 70.5% to 71.5%. Without removing cheat words, however, the lemmatizing decreased the performance. Most online resources and studies claim that lemmatizing is not very efficient for LSA [LD08]. We believe that it is efficient in this study because of a relatively small text corpus size.

k parameter in the SVD

It is obvious, but still worth mentioning, that bigger k preserves more text features and thus contributes to better model accuracy. The more principal axes of the data in the X matrix are being kept, the bigger the separation of the Ukrainian and Russian clouds. Thus, it is easier for the model to guess the message origin. For instance, changing k from 150 to 500 yields an increase from 71.7% to 79% in the CountVectorizer, from 73.3% to 80% in Binary CountVectorizer, and from 75.7% to 83.7% in the TfIdfVectoriser.

Vectorizer

It wasn't a surprise that the best results were shown by Binary CountVectorizer and TfidfVectorizer. The CountVectorizer showed mediocre results.

CountVectorizer	79.2%
Binary CountVectorizer	80.1%
TfidfVectorizer	83.7%

Distance metric

There are plenty of distance metrics, but only few of them are applicable to a given problem. Determined empirically, the best distance measures were: cosine, euclidean, mahalanobis. Cosine distance is $1 - cos(\theta)$, where $cos(\theta) = \frac{\langle A,B \rangle}{\|A\| \|B\|}$. Euclidean distance between vectors $\vec{a}$ and $\vec{a}$ is $\|\vec{a} - \vec{b}\| = \sqrt{\sum_k^n (a_k - b_k)^2}$. The mahalanobis distance could be thought of as a euclidean distance normalised by the variance of a cluster's distribution.

n-gram range

This one is tricky. An n-gram is a contiguous sequence of n words in the text. E.g. “Machine” and “Learning” happen to be one with each other quite often, so it could be more convenient to treat them like one token. This approach, however, consumes lots of CPU and memory, and thus may be inconvenient. On this particular dataset, allowing the TfidfVectorizer to search for both unigrams and bigrams resulted in 1.6% accuracy improvement. But this came at a high performance cost, since the size of vocabulary grew from $85,000$ lemmatized terms to 1434042 (17x size increase). So it really depends on one’s desired objectives, whether to plug n-grams into the model or not. Changing this parameter to $(2, 2)$ or $(1, 3)$ didn’t show a significant gain of performance. However, allowing the trigrams makes a tfidf matrix grow from 85,000 terms to 4093617 (48x memory increase).

n-gram	tokens	memory
$(1, 1)$	$85,000$	x
$(1, 2)$	1434042	$17x$
$(1, 3)$	4093617	$48x$

The lowest allowed post length

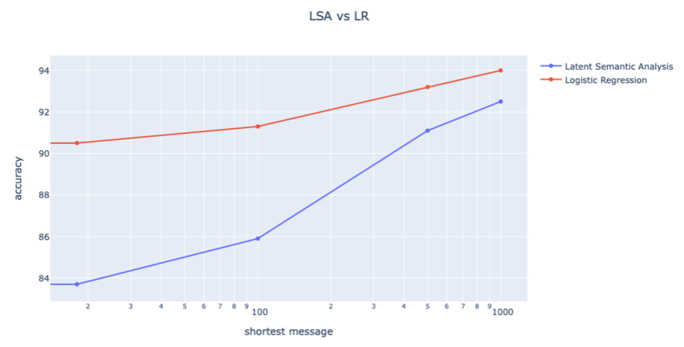
Before testing this parameter, a hypothesis was established: the longer message is – the more features the model can find to classify it. And the empirical data proved this hypothesis to be true. However, it is worth noting that if the symbol count threshold is too high, the model will become unuseful on the majority of the posts. As an example of the threshold effect on the results, a threshold of 18 symbols retains 94% of the original data and increases the best model’s performance by 0.1%, while a threshold of 100 symbols retains 73% of the data increasing the performance by 2.33%. But setting the threshold to 500 or even 1000 characters improves accuracy significantly. All in all, all being said could be summarized in the following table

threshold	% of original data	LSA, %	LR, %
0	100	83.6	89.9
18	94	83.7	90.6
100	73	85.9	91.3%
500	16	91.1	93.2%
1000	4	92.5	94%

Conclusion

1. The best preprocessing methods for the studied dataset were:
   (a) Filtering the posts in Ukrainian
   (b) Removing the cheat words
   (c) Setting the threshold for the shortest allowed post length
   (d) Lemmatization
2. Tfidf vectorizer performed the best, however there’s still room for improvement, for instance using a word2vec vectorizer. The Binary Vectorizer showed the best accuracy for the logistic regression model.
3. Truncated SVD could be applied to a low-rank approximation of X. PCA was used to project the X matrix onto its principal axes.
4. The best model in terms of accuracy can be trained using the following techniques:
   (a) Use TfidfVectorizer
   (b) Use combination of unigrams and bigrams
   (c) The best distance metric was shown to be cosine or euclidean
   (d) If the CPU allows, use as big k as possible. It only improves the performance

   In terms of time complexity it is not so obvious where the golden ratio between accuracy and time consumption is. Which k or n for n-grams to select depends on the specific objectives.
5. Naive Bayes failed dramatically. The best results (71.5%) were for unprocessed data with unigram models. The kNN attempt also failed. All results fell into the 70-76% range. The best were for 2–4 neighbours, weighted by distance. Logistic regression outperforms LSA both by time and accuracy, and works best with Binary CountVectorizer.



Comparison of logistic regression to LSA

Comments

Code with experiments, plots and other useful information could be found here:
https://github.com/taras-svystun/Distinguishing-Ukrainian-and-Russian-social-media

References

[LD08] Thomas K Landauer and Susan Dumais. “Latent Semantic Analysis”. In: *Scholarpedia* (2008). DOI: `3(11) : 4356`. URL: `http://www.scholarpedia.org/article/Latent_semantic_analysis`.

[LHu+10] Gastón L’Huillier et al. “Latent semantic analysis and keyword extraction for phishing classification”. In: (2010), pp. 129–131. DOI: `10.1109/ISI.2010.5484762`.

[al13] Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: (2013). URL: `https://arxiv.org/pdf/1301.3781.pdf`.

[KP15] Gang Kou and Yi Peng. “An Application of Latent Semantic Analysis for Text Categorization”. In: *International Journal of Computers Communications Control* 10 (Apr. 2015), p. 357. DOI: `10.15837/ijccc.2015.3.1923`.

[Nix22] Naomi Nix. “In Ukraine, Facebook fact-checkers fight a war on two fronts”. In: *Washington Post* (2022). URL: `https://www.washingtonpost.com/technology/2022/04/12/facebook-fact-checkers-misinformation-ukraine-war/`.

[Pau22] Kari Paul. “Facebook fails to label 80% of posts promoting bioweapons conspiracy theory”. In: *The Guardian* (2022). URL: `https://www.theguardian.com/technology/2022/mar/31/facebook-disinformation-war-ukraine-russia`.