**Oppositional Thinking Analysis**

This is Short Title of the paper, used in page headers

This is the subtitle of the paper, this document both explains and embodies the submission format for authors using Word

Dimitrij Schulz

TU Berlin, dmtschulz@gmail.com

This report addresses oppositional thinking analysis through text classification, focusing on distinguishing critical from conspiratorial narratives in a dataset of 4000 email instances. Tasks include data insights extraction, pre-processing steps selection, text classification using Naïve Bayes and Neural Network models as well as semantic textual similarity computation. Results underscore the effectiveness of different approaches in classifying and analyzing discourse, contributing insights into misinformation detection.

CCS CONCEPTS • Insert your first CCS term here • Insert your second CCS term here • Insert your third CCS term here

**Additional Keywords and Phrases:** Insert comma delimited author-supplied keyword list, Keyword number 2, Keyword number 3, Keyword number 4

ACM Reference Format:

First Author’s Name, Initials, and Last Name, Second Author’s Name, Initials, and Last Name, and Third Author’s Name, Initials, and Last Name. 2018. The Title of the Paper: ACM Conference Proceedings Manuscript Submission Template: This is the subtitle of the paper, this document both explains and embodies the submission format for authors using Word. In Woodstock ’18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA, 10 pages. NOTE: This block will be automatically generated when manuscripts are processed after acceptance.

1. Extracting insights from data

Understanding the distribution and characteristics of text data is foundational in any NLP endeavor. In this initial task, we aim to extract meaningful insights from our dataset through statistical analysis and visual representations. Key objectives include examining variations in text length across different categories, identifying the number of unique words per category, and exploring other pertinent metrics that shed light on the nature of oppositional thinking within the corpus.

In this task, we begin by loading and extracting data from the 'Oppositional\_thinking\_analysis\_dataset.json' file, which contains 4000 instances categorized into spam and normal email texts. Using Python and Pandas, we convert the extracted data into a DataFrame format for structured analysis. The initial step involves exploring fundamental characteristics of the dataset, such as text length distributions and the average number of unique words per category. Through visualizations like histograms and bar charts, we aim to uncover key insights that differentiate oppositional thinking narratives from other types of discourse. This foundational analysis not only sets the stage for subsequent tasks but also provides essential context for understanding the dataset's composition and variability.

My analysis of the dataset reveals several critical distinctions between conspiracy and critical thinking narratives. Conspiracy texts exhibit richer lexical diversity, averaging 80 unique words compared to 60 in critical texts. Notably, "5G" or "5 G" terms are significantly more prevalent in conspiracy texts, appearing in 6% of instances compared to nearly 0% in critical texts. Conversely, mentions of "Bill Gates" are markedly higher in conspiracy texts at 4%, contrasting sharply with a mere 1% in critical texts. Moreover, conspiracy texts tend to employ more uppercase words and exclamation marks on average, indicative of heightened emphasis or emotion. Another striking difference is the prevalence of sentences around 100 characters in length, with conspiracy texts containing approximately 1000 such instances compared to 300 in critical texts. These findings underscore distinct linguistic patterns and thematic focuses that differentiate oppositional thinking narratives within the dataset.

1. Preprocessing

Effective pre-processing of textual data is essential for optimizing our dataset of 4000 email instances categorized into conspiracy and critical thinking narratives. In this task, we focus on key pre-processing steps like tokenization, stopwords removal and lemming and others. These techniques are selected to standardize text representations and enhance the clarity of distinctions between narrative categories. Our choices are informed by their ability to reduce noise and improve data quality, aligning with the specific goals of our oppositional thinking analysis. We acknowledge the importance of these steps in preparing the dataset for subsequent modeling tasks, while considering their relevance over other available pre-processing techniques.

For this project, we implemented a series of pre-processing steps tailored to enhance the quality and relevance of our dataset of 4000 email instances categorized into conspiracy and critical thinking narratives. Our approach includes essential techniques such as lowercase conversion, removal of URLs, punctuation normalization (excluding exclamation marks), and tokenization using NLTK's word\_tokenize function. We opted for lemmatization using WordNetLemmatizer to reduce words to their base forms, which aids in standardizing word representations across texts. Additionally, we selectively removed stopwords from the English language using NLTK's stopwords corpus to eliminate common but non-informative words. These chosen steps are aimed at improving the dataset's coherence and facilitating clearer distinctions between narrative categories. Conversely, we decided against stemming techniques, which can overly simplify word forms and potentially reduce interpretability in our specific context. Our approach prioritizes maintaining semantic integrity while enhancing the dataset's suitability for subsequent analysis and modeling tasks.

1. text classification

In this task, we embark on a comprehensive evaluation of models designed for binary classification of narratives within our dataset. Our primary objectives include preparing the data by splitting it into training and test sets, ensuring balanced representation through under sampling technique. We initiate the process by training a Naïve Bayes model on the training data and subsequently evaluating its performance on the designated test set. A critical aspect of this task involves exploring the impact of various vectorization models, such as Count Vectorizer and TF-IDF, on enhancing the Naïve Bayes model's predictive accuracy. Additionally, we scrutinize different pre-processing pipelines, including strategies with and without stopwords and with and without lemming, to gauge their influence on model performance. Error analysis will be conducted to pinpoint areas where the Naïve Bayes model may falter, elucidating potential shortcomings in distinguishing between critical discourse and conspiratorial narratives. Furthermore, we augment our investigation by training a feed-forward neural network model and comparing its performance, particularly focusing on metrics like the F1 score. This comparative analysis between the Naïve Bayes and neural network models aims to derive insights into their respective strengths and limitations, culminating in a nuanced evaluation of their efficacy for this binary classification task.

* 1. Dataset Preparation

The original dataset was initially divided into an 80% train set comprising 3200 instances and a 20% test set comprising 800 instances. In the original train set, class counts were unevenly distributed with 1103 instances labeled as class 1 (conspiracy messages) and 2097 instances as class 0 (critical messages). Similarly, the test set had 524 instances of class 0 and 276 instances of class 1. To address this class imbalance, undersampling was applied, resulting in a reduced train set size of 2450 instances and an expanded test set size of 1550 instances. Post-undersampling, the train set now consists of 1103 instances of class 1 and 1347 instances of class 0, while the test set comprises 1274 instances of class 0 and 276 instances of class 1. These adjustments were crucial for ensuring a more balanced representation of classes in both training and testing phases, enhancing the reliability and generalizability of subsequent model training and evaluation processes.

* 1. Naïve Bayes Model Training

For the Naïve Bayes training, we initialized dictionaries for results and misclassifications and created a directory to save results. Data was loaded and split into train and test sets, with class distributions recorded. To address class imbalance, we applied undersampling, balancing the datasets by moving samples of the 'CRITICAL' category (labeled as 0)

Each data file and vectorizer combination was used to train and evaluate the Naïve Bayes model. This process enabled a comprehensive comparison of pre-processing and vectorization strategies, highlighting their impact on model performance. Table 1 summarizes the results of various vectorization models and pre-processing pipelines on the performance of the Naïve Bayes model. Table 1 summarizes the results of various vectorization models and pre-processing pipelines on the performance of the Naïve Bayes model.

Table 1: Performance Comparison of Naïve Bayes Model Using Different Pre-processing Pipelines and Vectorization Methods

| Data File | Vectorizer | F1 Score (weighted) | Misclassified Test Examples |
| --- | --- | --- | --- |
| With stopwords with lemmatization | TfidfVectorizer1 | 0.8942 | 165 |
| With stopwords with lemmatization | TfidfVectorizer2 | 0.8688 | 208 |
| With stopwords with lemmatization | TfidfVectorizer3 | 0.8478 | 229 |
| With stopwords with lemmatization | CountVectorizer1 | 0.8307 | 288 |
| With stopwords with lemmatization | CountVectorizer2 | 0.8255 | 298 |
| With stopwords with lemmatization | CountVectorizer3 | 0.8082 | 324 |
| Without stopwords with lemmatization | TfidfVectorizer1 | 0.8963 | 160 |
| Without stopwords with lemmatization | TfidfVectorizer2 | 0.8616 | 208 |
| Without stopwords with lemmatization | TfidfVectorizer3 | 0.8134 | 262 |
| Without stopwords with lemmatization | CountVectorizer1 | 0.8612 | 230 |
| Without stopwords with lemmatization | CountVectorizer2 | 0.8432 | 257 |
| Without stopwords with lemmatization | CountVectorizer3 | 0.7941 | 324 |
| With stopwords without lemmatization | TfidfVectorizer1 | 0.8869 | 178 |
| With stopwords without lemmatization | TfidfVectorizer2 | 0.8677 | 212 |
| With stopwords without lemmatization | TfidfVectorizer3 | 0.8482 | 230 |
| With stopwords without lemmatization | CountVectorizer1 | 0.8338 | 283 |
| With stopwords without lemmatization | CountVectorizer2 | 0.8179 | 312 |
| With stopwords without lemmatization | CountVectorizer3 | 0.7982 | 344 |
| Without stopwords without lemmatization | TfidfVectorizer1 | 0.8956 | 162 |
| Without stopwords without lemmatization | TfidfVectorizer2 | 0.8576 | 217 |
| Without stopwords without lemmatization | TfidfVectorizer3 | 0.8186 | 261 |
| Without stopwords without lemmatization | CountVectorizer1 | 0.8584 | 236 |
| Without stopwords without lemmatization | CountVectorizer2 | 0.8379 | 266 |
| Without stopwords without lemmatization | CountVectorizer2 | 0.8002 | 316 |

TfidfVectorizer1 is a unigram model, TfidfVectorizer2 includes bigrams, and TfidfVectorizer3 includes trigrams. Similarly, CountVectorizer1 is a unigram model, CountVectorizer2 includes bigrams, and CountVectorizer3 includes trigrams.

* + 1. Comparison of the Impact of different vectorization Models

The comparison of different vectorization models revealed that TF-IDF vectorizers consistently outperformed Count Vectorizers in terms of the F1 score. Among the TF-IDF vectorizers, TfidfVectorizer1, which uses unigrams, achieved the highest F1 score of 0.8963 with the least number of misclassified examples. In contrast, TfidfVectorizer3, using a combination of n-grams, showed a lower F1 score and higher misclassification rate. For Count Vectorizers, CountVectorizer1, which uses unigrams, performed better than other configurations but still lagged behind the TF-IDF vectorizers.

* + 1. Compare the Impact of different pre-processing Pipelines

The impact of different pre-processing pipelines was significant on the model’s performance. Models trained on data preprocessed without stop words and with lemmatization generally performed better, as seen with the highest F1 score of 0.8963 achieved using TfidfVectorizer1. Conversely, the pipeline including stop words and without lemmatization resulted in lower performance, indicating that removing stop words and applying lemmatization can enhance the model's ability to accurately classify messages.

* + 1. Naïve Bayes Error Analysis

The best performance was achieved using the dataset preprocessed without stopwords and with lemmatization in conjunction with TfidfVectorizer1 (unigram). This configuration resulted in the highest F1 score of 0.8963 and the fewest misclassified examples, with 160 out of 1550. This suggests that removing stopwords and applying lemmatization effectively reduces noise and enhances the model's ability to generalize better, especially when using a TF-IDF representation of unigrams.

The worst performance was observed with the dataset preprocessed with stopwords and without lemmatization paired with CountVectorizer3 (trigram). This combination yielded the lowest F1 score of 0.7982 and the highest number of misclassified examples, with 344 out of 1550. The inclusion of stopwords and lack of lemmatization likely introduced significant noise and redundancy, exacerbating the challenge of accurately capturing the semantic content of the messages.

* 1. Feedforward Neural Network Model Training

For training a Feedforward Neural Network (FFNN) model, we defined several key hyperparameters: two hidden layers with sizes 64 and 32, an output size of 1, a dropout rate of 0.4, a learning rate of 0.001, a batch size of 32, and 10 epochs for training. The model was configured to run on a GPU if available, otherwise on a CPU. Each data file was preprocessed using different vectorizers, and the resulting datasets were split into training and testing sets, which were then loaded into PyTorch DataLoader objects to facilitate batch processing during training. The FFNN model architecture consisted of two fully connected layers with ReLU activations and dropout layers for regularization, followed by a final fully connected layer with a sigmoid activation function to output the prediction. The model was trained and evaluated using the binary cross-entropy loss function and the Adam optimizer on a GTX 1060 with 6 GB of RAM graphic card. Table 2 summarizes the results of various vectorization models and pre-processing pipelines on the performance of the FFNN model.

Table 2: Performance Comparison of FFNN Model Using Different Pre-processing Pipelines and Vectorization Methods

| Data File | Vectorizer | F1 Score (weighted) | Misclassified Test Examples |
| --- | --- | --- | --- |
| With stopwords with lemmatization | TfidfVectorizer1 | 0.8424 | 268 |
| With stopwords with lemmatization | TfidfVectorizer2 | 0.8424 | 267 |
| With stopwords with lemmatization | TfidfVectorizer3 | 0.8159 | 310 |
| With stopwords with lemmatization | CountVectorizer1 | 0.8652 | 223 |
| With stopwords with lemmatization | CountVectorizer2 | 0.8707 | 211 |
| With stopwords with lemmatization | CountVectorizer3 | 0.8556 | 228 |
| Without stopwords with lemmatization | TfidfVectorizer1 | 0.8471 | 257 |
| Without stopwords with lemmatization | TfidfVectorizer2 | 0.8455 | 251 |
| Without stopwords with lemmatization | TfidfVectorizer3 | 0.7910 | 324 |
| Without stopwords with lemmatization | CountVectorizer1 | 0.8698 | 213 |
| Without stopwords with lemmatization | CountVectorizer2 | 0.8527 | 237 |
| Without stopwords with lemmatization | CountVectorizer3 | 0.8112 | 264 |
| With stopwords without lemmatization | TfidfVectorizer1 | 0.8497 | 254 |
| With stopwords without lemmatization | TfidfVectorizer2 | 0.8226 | 305 |
| With stopwords without lemmatization | TfidfVectorizer3 | 0.8122 | 317 |
| With stopwords without lemmatization | CountVectorizer1 | 0.8492 | 254 |
| With stopwords without lemmatization | CountVectorizer2 | 0.8669 | 216 |
| With stopwords without lemmatization | CountVectorizer3 | 0.8429 | 232 |
| Without stopwords without lemmatization | TfidfVectorizer1 | 0.8597 | 234 |
| Without stopwords without lemmatization | TfidfVectorizer2 | 0.8366 | 267 |
| Without stopwords without lemmatization | TfidfVectorizer3 | 0.8009 | 314 |
| Without stopwords without lemmatization | CountVectorizer1 | 0.8489 | 254 |
| Without stopwords without lemmatization | CountVectorizer2 | 0.8554 | 231 |
| Without stopwords without lemmatization | CountVectorizer3 | 0.8121 | 290 |

TfidfVectorizer1 is a unigram model, TfidfVectorizer2 includes bigrams, and TfidfVectorizer3 includes trigrams. Similarly, CountVectorizer1 is a unigram model, CountVectorizer2 includes bigrams, and CountVectorizer3 includes trigrams.

* + 1. Comparison of the Impact of different vectorization Models

Across the datasets analyzed, the choice of vectorization method—TfidfVectorizer versus CountVectorizer—shows distinct impacts on classification performance. When using TfidfVectorizer, which weighs terms based on their frequency-inverse document frequency, the misclassification rates tend to vary. For instance, in datasets with stopwords and lemmatization, TfidfVectorizer consistently resulted in higher misclassifications compared to CountVectorizer, particularly noticeable with n-gram sizes of 2 and 3. In contrast, CountVectorizer, which counts term frequencies directly, generally exhibited lower misclassification rates across all datasets, showcasing its robustness in capturing meaningful distinctions in text data. This comparison underscores the importance of considering the inherent characteristics of each vectorization model when optimizing text classification tasks.

* + 1. Compare the Impact of different pre-processing Pipelines

Comparing different preprocessing pipelines reveals significant impacts on text classification results. Datasets with stopwords and lemmatization generally exhibit higher misclassification rates with TfidfVectorizer compared to CountVectorizer. The choice of n-gram range within each vectorization model also influences performance, with larger n-grams sometimes increasing misclassification errors.

* 1. Comparing Naïve Bayes and feed-forward Model Performance

The Naïve Bayes model consistently outperforms the Feed-Forward Neural Network across all vectorization strategies and preprocessing pipelines in terms of F1 score and misclassification rates. Specifically, Naïve Bayes achieves higher F1 scores with generally fewer misclassified examples compared to FFNN. For instance, with stopwords and lemmatization, Naïve Bayes using TfidfVectorizer1 achieves an F1 score of 0.8942 with 165 misclassified examples, whereas FFNN achieves an F1 score of 0.8424 with 268 misclassified examples. Similarly, across other configurations, Naïve Bayes consistently demonstrates superior performance metrics.

One possible reason for the FFNN's comparatively poorer performance could be attributed to the size of the training dataset. FFNNs typically require larger amounts of data to effectively learn complex patterns and generalize well to unseen examples. In this context, the dataset size may not have been sufficient to fully capitalize on FFNN's potential, leading to suboptimal performance when compared to Naïve Bayes. These results underline the importance of dataset size and model suitability in achieving robust performance in text classification tasks. Therefore, when deciding between Naïve Bayes and FFNN for similar tasks, practitioners should consider both model performance and the adequacy of the training dataset.

1. Textual similarity

In this task, the objective is to assess textual similarity using distributional semantics at the sentence level. Specifically, 15 random instances will be selected, and the average of word vectors will be employed as a measure of semantic similarity between messages. The cosine similarity metric will be computed to quantify the degree of similarity between pairs of randomly chosen sentences. This approach provides insights into how effectively distributional semantics can capture semantic relationships between texts based on their word embeddings. The findings from this analysis will be detailed in the report, highlighting the computed cosine similarities and their implications for understanding textual coherence and similarity.

1. Select that table’s row, then right-click the row and select “Table Properties”;
2. In the *Table Properties* window, click the *Row* tab and select the box that says “Repeat as header row at the top of each page.”

Or

Apply the “table head” style by highlighting the respective row and applying the “**TableHead**” style found in the “Body Element” section of the ACM Master Article Template.

* 1. Figures

Figures are “float elements” which should be inserted after their first text reference, and have specific styles for identification. Insert a figure and apply the “**Image**” paragraph style to it. For the figure caption, apply the style “**FigureCaption.**”

To accommodate readers with color vision differences, figures should still be usable when printed in grayscale. Refer to elements of the figure with non-color terms, for example “indicated as squares” instead of “indicated in blue”. Use different patterns in bar charts, different line patterns in graphs, and different shapes in plots to distinguish groups of elements and reinforce color differences.

* + 1. Half Width Figures.

Figure 1 is an example of a figure and caption spanning the half-page width (one column in a two column format) with the styles applied. If your figure contains third-party material, you must clearly identify it as such, as shown in the example below.



Figure 1: 1907 Franklin Model D roadster. Photograph by Harris & Ewing, Inc. [Public domain], via Wikimedia Commons. (https://goo.gl/VLCRBB)

* + 1. Full Width Figures.

Figure 2 is an example of a figure and caption spanning the full-page width with the styles applied. If your figure contains third-party material, you must clearly identify it as such, as shown in the examples.



Figure 2: Mockup of a bombe machine at Bletchley Part. Photograph by Sarah Hartwell. [Public domain], via Wikimedia Commons. (<https://commons.wikimedia.org/wiki/File:TuringBombeBletchleyPark.jpg>)

* + 1. Multi-part figure.

Authors can also insert a multi-part figure above a single caption. Every inserted figure must have the “Image” style applied. Below are instructions regarding how to insert a multi-part figure in your paper.

* If the author wants to insert two multi-part images, they must draw a one row and one column table and insert the images one-by-one in the cells.
* If the author wants to insert three multi-part images, they must draw a one-row and three-column table and insert the images one by one in all three cells.
* If the author wants to insert four multi-part images, they must draw a two-row and two-column table and insert the images one-by-one in all four cells. (see the following example):

| Figure 2: The layout of multipart images should be as per the above example within the table in image 1. | Figure 2: The layout of multipart images should be as per the above example within the table in image 2. |
| --- | --- |
| Figure 2: The layout of multipart images should be as per the above example within the table in image 3. | Figure 2: The layout of multipart images should be as per the above example within the table in image 4. |

Figure 3: The layout of multipart images should be as per the above example within the table. All images must have the “Image” style applied.

* + 1. Figure Descriptions.

Every figure should have a figure description unless it is purely decorative. These descriptions convey what’s in the image to someone who cannot see it. They are also used by search engine crawlers for indexing images, and when images cannot be loaded.

A figure description must be unformatted plain text less than xxx characters long. Figure descriptions should not repeat the figure caption – their purpose is to capture important information that is not already provided in the caption or the main text of the paper. For figures that convey important and complex new information, a short plain text description may not be adequate. More complex alternative descriptions can be placed in an appendix and referenced in a short figure description. For example, provide a data table capturing the information in a bar chart, or a structured list representing a graph. For additional information regarding how best to write figure descriptions and why doing this is so important, please see [https://www.acm.org/accessibility.](https://www.acm.org/accessibility)

The instructions below describe the required steps authors need to follow in order to insert descriptive text for figures (alt-txt value) in **MS Word 2019 on Windows or Word 2016 and later on Mac**:

1. Insert a picture in the document.
2. Right-click the image and select “Edit Alt Text”.
3. In the “alt text” section, provide your text description of the image.

Below are the steps to insert figure descriptions in **MS Word 2013 and 2016**:

1. Insert a picture in the document.
2. Right click on the inserted picture and select the **Format Picture** option.
3. In the settings at the right side of the window, click on the “Layout & Properties” icon (3rd option).
4. Expand **Alt Txt** option.
5. In the “Title” and “Description” text boxes, type the text you want to represent the figure, and then click “Close.”

Below are steps to insert the alt-txt value in **MS Word 2010/2011 for Windows\***:

1. Insert a picture in the document.
2. Right click on the inserted picture and select the **Format Picture** option.
3. Select the **Alt Txt** option from the left-side panel options.
4. In the “Title” and “Description” text boxes, type the text you want to represent the picture, and then click “Close.”  
   \* The Mac 2011 version 14.0.0 and later allows the option for inserting “alt-text.” In the MAC version of Word 2016, right-click on the image and select “Edit Alt Text” from the pop-up menu and then enter the description for the alt text.
   1. Quotations and Extracts

There are styles for block quotations, which should be used for quotes that are separated from in-line text. Below is an example.

“Microsoft tried to revive the idea of an assistant with Clippy, who began popping up in Microsoft Office in 1997. Its creator, Kevan Atteberry, was actually contracted by Microsoft to design Clippy, which, funnily enough, he did on a Mac … Sure, people could disable Clippy, but the fact he was on by default angered people.” [10]

* 1. Equations

There are two types of math equations: the *numbered display math equation* and the *un-numbered display math equation*. Below are examples of both.

* + 1. DisplayFormula.

The **DisplayFormula** style is applied in the numbered math equation. A numbered display equation always has an equation number (label) on the right.

(1)

* + 1. DisplayFormula.Unnum.

The **DisplayFormulaUnnum** style is applied only in unnumbered equations. An unnumbered display equation never contains an equation number Bertot and Grimes (2012) on the right—this element distinguishes it from the numbered equation.

Please note: the subsequent text after the **DisplayFormula** (numbered equation) or **DisplayFormulaUnnum** (unnumbered equation) must have the paragraph style **ParaContinue** applied.

* 1. Math statements

Math statements should have the “Statement” style applied.

**Theorem/Proof/Lemma.** Math statements should have the “**Statement**” style applied. This paragraph is an example of the “**Statement**” style.

* 1. Algorithms

Algorithms use the styles “AlgorithmCaption” and “Algorithm”.

ALGORITHM 1: Iterative Algorithm

current\_position center

current\_direction up

current\_position is inside circle

while current\_position is inside circle, do

neighborhood all grid hexes within two hexes from current\_position

for each hex in neighborhood, do

for each neuron in hex do

convert neuron\_orientation to vector

scale vector by neuron\_excitation

vector\_sum vector\_sum + vector

end

end

normalize vector\_sum

end

1. COMPUTER CODE

Display Computer codes can be inserted using “ComputerCode” style.

CHAT Start

SAY Welcome to my world

WAIT 1.2

SAY Thanks for Visiting

ASK Do you want to play a game?

OPT Sure

OPT No Thanks

Similary, this is an example of intext code text.

Similary, this is an example of intext code text.

1. Citing Related Work

This section cites a variety of journal [5, 15], conference [1, 6, 8, 12, 13], and magazine [3] articles to illustrate how they appear in the references section. It also cites books [9, 10], a technical report [7], a PhD dissertation [4], an online reference [14], a software artifact [11], and a dataset [2].

As you build your article, you should note where you will be placing citations. If you are using numbered citations and references, the reference number - "...as shown in [5]..." is sufficient. If you are using the "author year" style, a reasonable placeholder is the primary author's last name and the year of publication - "...as shown in [Harel 1978]..." - we will be updating this placeholder later in the process with the citation label as generated by the Word macros in the "master template.

ACKNOWLEDGMENTS

Acknowledgments are placed before the references. Add information about grants, awards, or other types of funding that you have received to support your research. Author can capture the **grant sponsor information**, by selecting the grant sponsor text and apply style ‘GrantSponsor’. After this, select grant no and apply ‘GrantNumber’ from style panel. Example of Grant sponsor: Competitive Research Programme and example of Grant no: CRP 10-2012-03.

1. HISTORY DATES

In case of submissions being prepared for Journals or PACMs, please add history dates after References as (*please note revised date is optional*):

Received November 2019; revised August 2020; accepted December 2020

REFERENCES

1. Atul Adya, Paramvir Bahl, Jitendra Padhye, Alec Wolman, and Lidong Zhou. 2004. A multi-radio unification protocol for IEEE 802.11 wireless networks. In Proceedings of the IEEE 1st International Conference on Broadnets Networks (BroadNets’04) . IEEE, Los Alamitos, CA, 210–217. https://doi.org/10.1109/BROADNETS.2004.8
2. Sam Anzaroot and Andrew McCallum. 2013. UMass Citation Field Extraction Dataset. Retrieved May 27, 2019 from <http://www.iesl.cs.umass.edu/data/data-umasscitationfield>
3. Martin A. Fischler and Robert C. Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM 24, 6 (June 1981), 381–395. https://doi.org/10.1145/358669.358692
4. Chelsea Finn. 2018. Learning to Learn with Gradients. PhD Thesis, EECS Department, University of Berkeley.
5. Jon M. Kleinberg. 1999. Authoritative sources in a hyperlinked environment. J. ACM 46, 5 (September 1999), 604–632. https://doi.org/10.1145/324133.324140
6. Matthew Van Gundy, Davide Balzarotti, and Giovanni Vigna. 2007. Catch me, if you can: Evading network signatures with web-based polymorphic worms. In Proceedings of the first USENIX workshop on Offensive Technologies (WOOT ’07) . USENIX Association, Berkley, CA, Article 7, 9 pages.
7. James W. Demmel, Yozo Hida, William Kahan, Xiaoye S. Li, Soni Mukherjee, and Jason Riedy. 2005. Error Bounds from Extra Precise Iterative Refinement. Technical Report No. UCB/CSD-04-1344. University of California, Berkeley.
8. David Harel. 1979. First-Order Dynamic Logic. Lecture Notes in Computer Science, Vol. 68. Springer-Verlag, New York, NY. <https://doi.org/10.1007/3-540-09237-4>
9. Jason Jerald. 2015. The VR Book: Human-Centered Design for Virtual Reality. Association for Computing Machinery and Morgan & Claypool.
10. Prokop, Emily. 2018. The Story Behind. Mango Publishing Group. Florida, USA.
11. R Core Team. 2019. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/
12. Brian K. Reid. 1980. A high-level approach to computer document formatting. In Proceedings of the 7th Annual Symposium on Principles of Programming Languages. ACM, New York, 24–31. <https://doi.org/10.1145/567446.567449>
13. John R. Smith and Shih-Fu Chang. 1997. Visual Seek: a fully automated content-based image query system. In Proceedings of the fourth ACM international conference on Multimedia (MULTIMEDIA ’96). Association for Computing Machinery, New York, NY, USA, 87–98. https://doi.org/10.1145/244130.244151
14. TUG 2017. Institutional members of the LaTeX Users Group. Retrieved May 27, 2017 from <http://wwtug.org/instmem.html>
15. Alper Yilmaz, Omar Javed, and Mubarak Shah. 2006. Object tracking: A survey. ACM Comput. Surv. 38, 4 (December 2006), 13–es. https://doi.org/10.1145/1177352.1177355

Patricia S. Abril and Robert Plant. 2007. The patent holder's dilemma: Buy, sell, or troll? Commun. ACM 50, 1 (Jan. 2007), 36-44. DOI: <https://doi.org/10.1145/1188913.1188915>

Sarah Cohen, Werner Nutt, and Yehoshua Sagic. 2007. Deciding equivalences among conjunctive aggregate queries. J. ACM 54, 2, Article 5 (April 2007), 50 pages. DOI: https://doi.org/10.1145/1219092.1219093

David Kosiur. 2001. Understanding Policy-Based Networking (2nd. ed.). Wiley, New York, NY.

Ian Editor (Ed.). 2007. The title of book one (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago. DOI: https://doi.org/10.1007/3-540-09237-4

Donald E. Knuth. 1997. The Art of Computer Programming, Vol. 1: Fundamental Algorithms (3rd. ed.). Addison Wesley Longman Publishing Co., Inc.

Sten Andler. 1979. Predicate path expressions. In Proceedings of the 6th. ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '79), January 29 - 31, 1979, San Antonio, Texas. ACM Inc., New York, NY, 226-236. <https://doi.org/10.1145/567752.567774>

Joseph Scientist. 2009. The fountain of youth. (Aug. 2009). Patent No. 12345, Filed July 1st., 2008, Issued Aug. 9th., 2009.

David Harel. 1978. LOGICS of Programs: AXIOMATICS and DESCRIPTIVE POWER. MIT Research Lab Technical Report TR-200. Massachusetts Institute of Technology, Cambridge, MA.

Kenneth L. Clarkson. 1985. Algorithms for Closest-Point Problems (Computational Geometry). Ph.D. Dissertation. Stanford University, Palo Alto, CA. UMI Order Number: AAT 8506171.

David A. Anisi. 2003. Optimal Motion Control of a Ground Vehicle. Master's thesis. Royal Institute of Technology (KTH), Stockholm, Sweden.

Harry Thornburg. 2001. Introduction to Bayesian Statistics. (March 2001). Retrieved March 2, 2005 from http://ccrma.stanford.edu/~jos/bayes/bayes.html

ACM. Association for Computing Machinery: Advancing Computing as a Science & Profession. Retrieved from http://www.acm.org/.

Wikipedia. 2017. Wikipedia: the Free Encyclopedia. Retrieved from https://www.wikipedia.org/.

Dave Novak. 2003. Solder man. Video. In ACM SIGGRAPH 2003 Video Review on Animation theater Program: Part I - Vol. 145 (July 27-27, 2003). ACM Press, New York, NY, 4. DOI: https://doi.org/99.9999/woot07-S422

Barack Obama. 2008. A more perfect union. Video. (5 March 2008). Retrieved March 21, 2008 from http://video.google.com/videoplay?docid=6528042696351994555

Martha Constantinou. 2016. New physics searches from nucleon matrix elements in lattice QCD. arXiv:1701.00133. Retrieved from <https://arxiv.org/abs/1701.00133>