# Implementing of RankNet

Ryan Greenup

February 21, 2021

## Contents

# #+TODO: TODO IN-PROGRESS WAITING DONE

# 1 Introduction

Ranknet is an approach to *Machine-Learned Ranking (often refered to as "*/Learning to Rank" [21]) that began development at Microsoft from 2004 onwards [5], although previous work in this area had already been undertaken as early as the 90s (see generally [9, 8, 9, 10, 37]) these earlier models didn't perform well compared to more modern machine learning techniques [23, §15.5].

Information retrieval is an area that demands effective ranking of queries, although straight-forward tools such as `grep`, relational databases (e.g. sqlite, MariaDB, PostgreSql) or NoSQL (e.g. CouchDB, MongoDB) can be used to retrieve documents with matching characters and words, these methods do not perform well in real word tasks across large collections of documents because they do not provide any logic to rank results (see generally [35]).

# 2 Motivation

Search Engines implement more sophisticated techniques to rank results, one such example being TF-IDF weighting [24] , well established search engines such as *Apache Lucene* [1] and *Xapian* [14], however, are implementing Machine-Learned Ranking in order to improve results.

This paper hopes to serve as a general introduction to the implementation of the Ranknet technique to facilitate developers of search engines in more modern languages (i.e. `Go` and `Rust`) in implementing it. This is important because these more modern languages are more accessible [13] and memory safe [30] than C/C++ respectfully, without significantly impeding performance; this will encourage contributors from more diverse backgrounds and hence improve the quality of profession-specific tooling.

For a non-comprehensive list of actively maintained search engines, see §9.1 of the appendix.

# 3 Implementation

Neural Networks

Ranking/ is the process of applying machine learning algorithms to ranking problems, it .

This implementation will first apply the approach to a simple data set so as to clearly demonstrate that the approach works, following that the model will be extended to support wider and more complex data types before finally being implemented on a corpus of documents.

## 3.1 Neural Networks

The Ranknet method is typically implemented using a Neural Networks [1], although other machine learning techniques can also be used [5, p. 1], Neural Networks are essentially a collection of different regression models and classifiers that are fed into one another to create a non-linear classifier, a loss function is used to measure the performance of the model with respect to the parameters (e.g. RMSE [2] or BCE [3]) and the parameters are adjusted so as to reduce this error by using the *Gradient Descent Technique* (although there are other optimisation algorithms such as RMSProp and AdaGrad [25] that can be shown to perform better, see [2]). The specifics of Neural Networks are beyond the scope of this paper (see [12] or more generally [31]).

### 3.1.1 The Ranknet Method

The Ranknet method is concerned with a value $p_{ij}$ that measures the probability that an observation $i$ is ranked higher than an observation $j$.

A Neural Network ($n$) is trained to return a value $s_k$ from a feature vector $\mathbf{X}_k$:

$$n(\mathbf{X}_i) = s_i \quad \exists k$$

So as to minimise the error of:

$$p_{ij} = \frac{1}{1 + e^{\sigma \cdot (s_i - s_j)}} \quad \exists \sigma \in \mathbb{R}$$

**Version Control** The implementation in this paper corresponds to the `walkthrough` branch of the `git` repository used in production of this work, id values (e.g. `:08db5b0:`) will be appended to titles to denote specific changes made in that section. See §9.2 for more specific guidance.

## 3.2 Creating Data cf9ab26

The first step is to create a simple data set and design a neural network that can classify that data set, the data set generated should have two classes of data (this could be interpreted as relevant and irrelevant documents given the features or principle components of a data set).

In order to fit a Neural Network the *PyTorch* package can be used [29], this will allow the gradients of the neural network to be calculated numerically without needing to solve for the partial derivatives, hence the data will need to be in the form of tensors.

This can be implemented like so:

[1] An early goal of this research was to evaluate the performance of different machine learning algorithms to implement the Ranknet method, as well as contrasting this with simple classification approaches, this research however is still ongoing, see §5.3

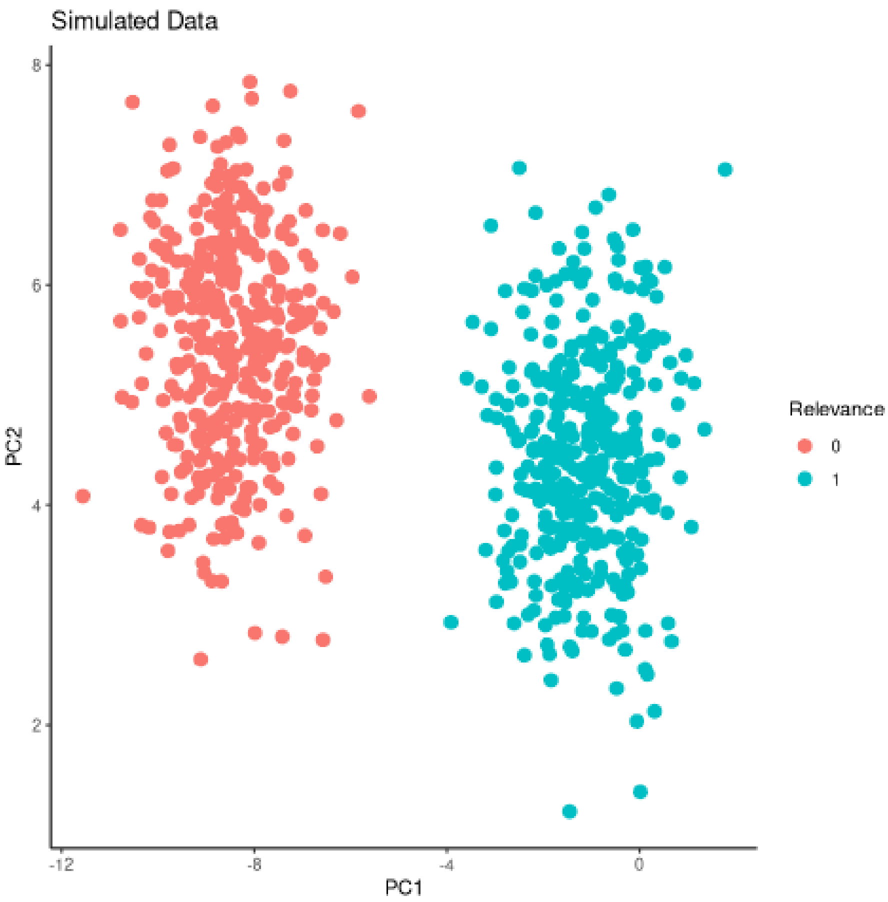
[2] **RMSE** *Root Mean Square Error*

[3] **BCE** *Binary Cross Entropy*

```python
import torch                    #
import os
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split


def make_data(create_plot=False, n=1000, dtype=torch.float, dev="cpu", export=""):
    X, y = datasets.make_blobs(n, 2, 2, random_state=7)
    # X, y = datasets.make_moons(n_samples=n, noise=0.1, random_state=0) # Moons Data for later

    # Save the data somewhere if necessary
    if export != "":
^^Iexport_data(X, y, export)

    # Reshape the data to be consistent
    y = np.reshape(y, (len(y), 1))  # Make y vertical n x 1 matrix.

    # -- Split data into Training and Test Sets --------------------
    data = train_test_split(X, y, test_size=0.4)

    if(create_plot):
^^I# Create the Scatter Plot
^^Iplt.scatter(X[:, 0], X[:, 1], c=y)
^^Iplt.title("Sample Data")
^^Iplt.show()

    # Make sure we're working with tensors not mere numpy arrays
    torch_data = [None]*len(data)
    for i in range(len(data)):
^^Itorch_data[i] = torch.tensor(data[i], dtype=dtype, requires_grad=False)

    return torch_data


def export_data(X, y, export):
    try:
^^Ios.remove(export)
^^Iprint("Warning, given file was over-written")
    except:
^^Ipass

    with open(export, "a") as f:
^^Iline = "x1, x2, y \n"
^^If.write(line)
^^Ifor i in (range(X.shape[0])):
^^I    line = str(X[i][0]) + ", " + str(X[i][1]) + ", " + str(y[i]) + "\n"
^^I    f.write(line)
    print("Data Exported")

# Set Torch Parameters
dtype = torch.float
dev = test_cuda()

# Generate the Data
X_train, X_test, y_train, y_test = make_data(
    n=int(300/0.4), create_plot=True, dtype=dtype, dev=dev, export = "/tmp/simData.csv")
```

And will produce a dataset like so:

```r
library(tidyverse)
data  <- read_csv("/tmp/simData.csv")
myplot <-  ggplot(data, aes(x = x1, y = x2, col = factor(y))) +
    geom_point(size = 3) +
    theme_classic() +
    labs(col = "Relevance", x = "PC1", y = "PC2",
^^I title = "Simulated Data")

myplot
```




## 3.3 Creating a Neural Network 7291112

A Neural Network model can be designed as a class, here a 2-layer model using Sigmoid functions has been described, this design was chosen for it's relative simplicity:

```python
import torch
import numpy as np
from torch import nn


class three_layer_classification_network(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, dtype=torch.float,
    ↪   dev="cpu"):
```

```
^^Isuper(three_layer_classification_network, self).__init__()
^^Iself.wi = torch.randn(input_size, hidden_size, dtype=dtype, requires_grad=True)
^^Iself.wo = torch.randn(hidden_size, output_size, dtype=dtype, requires_grad=True)

^^Iself.bi = torch.randn(hidden_size, dtype=dtype, requires_grad=True)
^^Iself.bo = torch.randn(output_size, dtype=dtype, requires_grad=True)

^^Iself.losses = []

    def forward(self, x):
^^Ix = torch.matmul(x, self.wi).add(self.bi)
^^Ix = torch.sigmoid(x)
^^Ix = torch.matmul(x, self.wo).add(self.bo)
^^Ix = torch.sigmoid(x)
^^Ireturn x

    def loss_fn(self, x, y):
^^Iy_pred = self.forward(x)
^^Ireturn torch.mean(torch.pow((y-y_pred), 2))

    def misclassification_rate(self, x, y):
^^Iy_pred = (self.forward(x) > 0.5)
^^Ireturn np.average(y != y_pred)
```

A model can then be instantiated, here a 2-3-1 model has been implemented, this choice was arbitrary (note that the model has not yet been trained, the rates are random):

```
#!/usr/bin/env python

# Import Packages
import numpy as np
import matplotlib.pyplot as plt
import torch
import sys
import random
from ranknet.test_cuda import test_cuda
from ranknet.make_data import make_data
from ranknet.neural_network import three_layer_classification_network

# Set Seeds
torch.manual_seed(1)
np.random.seed(1)

# Set Torch Parameters
dtype = torch.float
dev = test_cuda()

# Set personal flags
DEBUG = True


# Main Function

def main():
    # Make the Data
    X_train, X_test, y_train, y_test = make_data(
^^In=100, create_plot=True, dtype=dtype, dev=dev)

    # Create a model object
    model = three_layer_classification_network(
```

```
^^Iinput_size=X_train.shape[1], hidden_size=2, output_size=1, dtype=dtype, dev=dev)


    # Send some data through the model
    print("\nThe Network input is:\n---\n")
    print(X_train[7,:], "\n")
    print("The Network Output is:\n---\n")
    print(model.forward(X_train[7,:]).item(), "\n")


if __name__ == "__main__":
    main()
```

This outputs the following:

```
The Network input is:
---

tensor([-1.5129,  2.9332])

The Network Output is:
---

0.22973690927028656
```

## 3.4 Train the Model with Gradient Descent 7D46636

Now that the model has been fit, a method to train the model can be implmented:

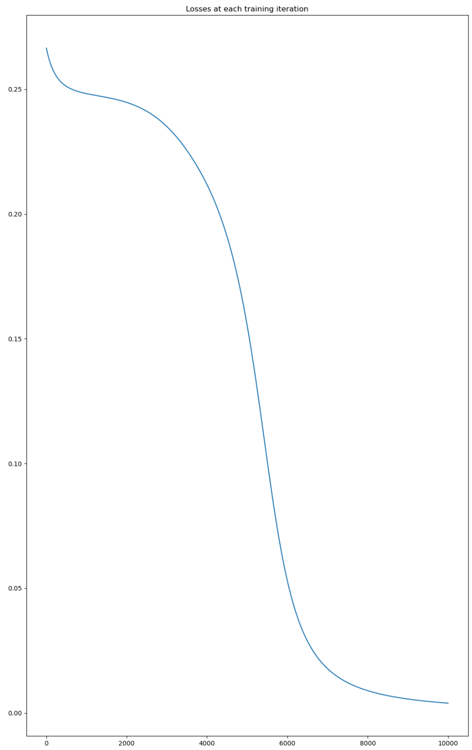
```
class three_layer_classification_network(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, dtype=torch.float,
     ↪  dev="cpu"):
^^Isuper(three_layer_classification_network, self).__init__()
^^Iself.wi = torch.randn(input_size, hidden_size, dtype=dtype, requires_grad=True)
^^Iself.wo = torch.randn(hidden_size, output_size, dtype=dtype, requires_grad=True)

^^Iself.bi = torch.randn(hidden_size, dtype=dtype, requires_grad=True)
^^Iself.bo = torch.randn(output_size, dtype=dtype, requires_grad=True)
^^Iself.σ = torch.randn(output_size, dtype=dtype, requires_grad=True)

^^Iself.losses = []

    def train(self, x, target, η=30, iterations=2e4):
^^Ibar = Bar('Processing', max=iterations) # progress bar
^^Ifor t in range(int(iterations)):

^^I    # Calculate y, forward pass
^^I    y_pred = self.forward(x)

^^I    # Measure the loss
^^I    loss = self.loss_fn(x, target)

^^I    # print(loss.item())
^^I    self.losses.append(loss.item())

^^I    # Calculate the Gradients with Autograd
^^I    loss.backward()

^^I    with torch.no_grad():
```

```
^^I^^I# Update the Weights with Gradient Descent
^^I^^Iself.wi -= η * self.wi.grad; self.wi.grad = None
^^I^^Iself.bi -= η * self.bi.grad; self.bi.grad = None
^^I^^Iself.wo -= η * self.wo.grad; self.wo.grad = None
^^I^^Iself.bo -= η * self.bo.grad; self.bo.grad = None
^^I^^Iself.σ  -= η * self.σ.grad;  self.σ.grad = None
^^I    bar.next()
^^Ibar.finish()
^^I^^I# ; Zero out the gradients, they've been used

    # Rest of the Class Definition Below ...VVV...
```

So now the model can be trained in order to produce a meaningful classification:

```
def main():
    # Make the Data
    X_train, X_test, y_train, y_test = make_data(
^^In=100, create_plot=True, dtype=dtype, dev=dev)

    # Create a model object
    model = three_layer_classification_network(
^^Iinput_size=X_train.shape[1], hidden_size=2, output_size=1, dtype=dtype, dev=dev)

    model.train(X_train, y_train, η=1e-2, iterations=10000)
    plt.plot(model.losses)
    plt.title("Losses at each training iteration")
    plt.show()

    print("The testing misclassification rate is:\n")
    print(model.misclassification_rate(X_test, y_test))


if __name__ == "__main__":
    main()
```

This model classifies the points perfectly, even on the testing data, the loss function at each iteration of training shown below:




## 3.5 Implement Ranknet

Now that the model can classify the data, the implementation will be modified to:

... describe ranknet ...

this is shown below:

misclassification rate isn't a meaningful measure though

## 3.6 Implement sorting

So instead of ranking, sort the values, this produces the output.

but this is the problem, did it work? it's not clear, because even if the model was not trained we get the following (put them side by side).

So this is definitely one of the hard issues.

what would be better would be to classify data with a rating (i.e. wine scores), only show the model whether the wine is good/bad and compare the output order with the input order, that would be an effective way to see that it works. This was not yet effectively implemented.

## 3.7 Moons

## 3.8 Optimisers

## 3.9 Batches

## 3.10 Wine

### 3.11 Rank Wiki Articles

## 4 Difficulties

- Don't use torch
  - Do it by hand first because it can be hard to see if the correct weights are being updated sensibly, making debugging very difficult.
  - R or Julia would be easier because counting from 0 get's pretty confusing when dealing with {1, 0}, {-1, 0, 1}.
- Don't use misclassification rate to measure whether the ranking
  - In hindsight this is obvious, but at the time misclassification was a tempting metric because of it's interpretability

  was correct

  Very difficult to see if the model is working
- A continuous function will still produce an ordered pattern in the ranking of results, even if the model hasn't been trained, so visualising isn't helpful either.
- Implement it on a data set that already has order, obfuscate the order and then contrast the results
  - or use a measurement
- Plot the loss function of the training data live, the model is slow to train and waiting for it to develop was a massive time drain.

## 5 Further Research

### 5.1 Practical Improvements

- Apply this to documents to get a sorted list, like the wine data
- The "Quicksort" algorithm likely needs a random pivot to be efficient [32]

### 5.2 Evaluate performance improvements

It is still not clear how the performance of Ranknet compares to traditional approaches implemented by search engines (see §9.1), further study would ideally:

- Write a program to query a corpus of documents using an existing search engine.
  - Or possibly just implement TF-IDF weighting in order to remove variables.
- Extend the program to implement machine learned ranking
- Measure and contrast the performance of the two models to see whether there are any significant improvements.

This could be implemented with TREC datasets [34] using a cummulated-gain cost function [16] as demonstrated in previous work [35].

### 5.3 Evaluate alternative machine learning models

i.e. can SVM's or trees be used instead of neural networks?

## 6 Conclusion

## 7 Text and References

Fractals are complex shapes that often occur from natural processes, in this report we hope to investigate the emergence of patterns and complex structures from natural phenomena. We begin with an investigation into fractals and the concept of dimension and then discuss links between fractal patterns and natural processes.

This is a Reference [33] and another [26] and yet another [4].

## 8 Fractals

Images are shown in figure .

## 9 Appendix

### 9.1 Search Engines

There are many open source search engines available , a cursory review found the following popular projects:

- Zettair (`C`) [15]
- Apache lucene/Solr (`Java`) [1]
  - Implemented by DocFetcher [7]
- Sphinx (`C++`) [38]
- Xapian (`C++`) [28]
  - Implemented by Recoll [17]

More Modern Search engines include:

- LunrJS (`JS`) [27]
- Bleve Search (`Go`) [24]
- Riot (`Go`) [36]
- Tantivy (`Rust`) [6]
- SimSearch (`Rust`) [22]

### 9.1.1 Fuzzy String Match

Somewhat related are programs that rank string similarity, such programs don't tend to perform well on documents however (so for example these would be effective to filter document titles but would not be useful for querying documents):

- `fzf` [3]
- `fzy` [11]
- `peco` [20]
- Skim [39]
- `go-fuzzyfinder` [19]
- Swiper [18]

## 9.2 Version Control Repository

The `git` repository used in production of this code is currently available on *GitHub* at github.com/CRMDS/CRMDS-HDR-Training-2020, in order to get a local copy, execute the following commands (`bash`):

```bash
# Clone the repository
git clone https://github.com/CRMDS/CRMDS-HDR-Training-2020

# Change to the subdirectory
cd CRMDS-HDR-Training-2020/ranknet

# Checkout the Walkthrough branch
git checkout walkkthrough

# list the changes
git log
```

Consider the use of a tool like magit and git-timemachine (or GitLens and git-temporal in VsCode) in order to effectively preview the changes at each step, alternatively a pager like bat can also be used with something like fzf like so:

```bash
git log | grep '^commit' | sed 's/^commit\ //' |\
    fzf --preview 'git diff {}^! |\
     bat --color always'
```

## References

[1] Apache Software Foundation. *Learning To Rank | Apache Solr Reference Guide 6.6*. Solr Reference Guide. 2017. URL: https://lucene.apache.org/solr/guide/6_6/learning-to-rank.html (visited on 02/20/2021) (cit. on pp. 3, 12).

[2] Vitaly Bushaev. *Understanding RMSprop — Faster Neural Network Learning*. Medium. URL: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a (visited on 02/16/2021) (cit. on p. 4).

[3] Junegunn Choi. *Junegunn/Fzf*. Feb. 20, 2021. URL: https://github.com/junegunn/fzf (visited on 02/20/2021) (cit. on p. 13).

[4] Christopher Burges. *From RankNet to LambdaRank to LambdaMART: An Overview (MSR-TR-2010-82*. Jan. 1, 2010. URL: https://www.microsoft.com/en-us/research/uploads/prod/2016/02/MSR-TR-2010-82.pdf (cit. on p. 12).

[5] Christopher Burges. *RankNet: A Ranking Retrospective*. Microsoft Research. July 7, 2015. URL: https://www.microsoft.com/en-us/research/blog/ranknet-a-ranking-retrospective/ (visited on 02/15/2021) (cit. on pp. 3, 4).

[6] Clement Renault, Marin, and Quentin de Quelen. *Meilisearch/MeiliSearch*. MeiliSearch, Feb. 20, 2021. URL: https://github.com/meilisearch/MeiliSearch (visited on 02/20/2021) (cit. on p. 12).

[7] Docfetcher Development Team. *DocFetcher - Fast Document Search*. URL: http://docfetcher.sourceforge.net/en/index.html (visited on 02/15/2021) (cit. on p. 12).

[8] Norbert Fuhr. "Optimum Polynomial Retrieval Functions Based on the Probability Ranking Principle". In: *ACM Transactions on Information Systems* 7.3 (July 1, 1989), pp. 183–204. ISSN: 1046-8188. DOI: 10.1145/65943.65944. URL: https://doi.org/10.1145/65943.65944 (visited on 02/15/2021) (cit. on p. 3).

[9] Norbert Fuhr. "Probabilistic Models in Information Retrieval". In: *The Computer Journal* 35.3 (June 1, 1992), pp. 243–255. ISSN: 0010-4620. DOI: 10.1093/comjnl/35.3.243. URL: https://doi.org/10.1093/comjnl/35.3.243 (visited on 02/15/2021) (cit. on p. 3).

[10] Fredric C. Gey. "Inferring Probability of Relevance Using the Method of Logistic Regression". In: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '94. Berlin, Heidelberg: Springer-Verlag, Aug. 1, 1994, pp. 222–231. ISBN: 978-0-387-19889-7 (cit. on p. 3).

[11] John Hawthorn. *Jhawthorn/Fzy*. Feb. 20, 2021. URL: https://github.com/jhawthorn/fzy (visited on 02/20/2021) (cit. on p. 13).

[12] HMKCode. *Backpropagation Step by Step*. URL: https://hmkcode.com/ai/backpropagation-step-by-step/ (visited on 02/16/2021) (cit. on p. 4).

[13] Daniel Hunt. "A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science". In: (), p. 30 (cit. on p. 3).

[14] James Aylett. *GSoCProjectIdeas/LearningtoRankStabilisation – Xapian*. Oct. 20, 2019. URL: https://trac.xapian.org/wiki/GSoCProjectIdeas/LearningtoRankStabilisation (visited on 02/20/2021) (cit. on p. 3).

[15] Dr Rolf Jansen. *Cyclaero/Zettair*. Dec. 22, 2020. URL: https://github.com/cyclaero/zettair (visited on 02/20/2021) (cit. on p. 12).

[16] Kalervo Järvelin and Jaana Kekäläinen. "Cumulated Gain-Based Evaluation of IR Techniques". In: *ACM Transactions on Information Systems* 20.4 (Oct. 1, 2002), pp. 422–446. ISSN: 1046-8188. DOI: 10.1145/582415.582418. URL: http://doi.org/10.1145/582415.582418 (visited on 02/20/2021) (cit. on p. 12).

[17] Jean-Francois Dockes. *Recoll User Manual*. URL: https://www.lesbonscomptes.com/recoll/usermanual/usermanual.html (visited on 02/15/2021) (cit. on p. 12).

[18] Oleh Krehel. *Abo-Abo/Swiper*. Feb. 20, 2021. URL: https://github.com/abo-abo/swiper (visited on 02/20/2021) (cit. on p. 13).

[19] ktr. *Ktr0731/Go-Fuzzyfinder*. Feb. 16, 2021. URL: https://github.com/ktr0731/go-fuzzyfinder (visited on 02/20/2021) (cit. on p. 13).

[20] lestrrat. *Peco/Peco*. peco, Feb. 18, 2021. URL: https://github.com/peco/peco (visited on 02/20/2021) (cit. on p. 13).

[21] Tie-Yan Liu. "Learning to Rank for Information Retrieval". In: *Foundations and Trends® in Information Retrieval* 3.3 (June 26, 2009), pp. 225–331. ISSN: 1554-0669, 1554-0677. DOI: 10.1561/1500000016. URL: https://www.nowpublishers.com/article/Details/INR-016 (visited on 02/15/2021) (cit. on p. 3).

[22] Andy Lok. *Andylokandy/Simsearch-Rs*. Feb. 15, 2021. URL: https://github.com/andylokandy/simsearch-rs (visited on 02/20/2021) (cit. on p. 12).

[23] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. New York: Cambridge University Press, 2008. 482 pp. ISBN: 978-0-521-86571-5 (cit. on p. 3).

[24] Marty Schoch and Abhinav Dangeti. *Bleve Search Documentation*. URL: http://blevesearch.com/ (visited on 02/20/2021) (cit. on pp. 3, 12).

[25] Mahesh Chandra Mukkamala and Matthias Hein. "Variants of RMSProp and Adagrad with Logarithmic Regret Bounds". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, Aug. 6, 2017, pp. 2545–2553 (cit. on p. 4).

[26] Olympia Nicodemi, Melissa A. Sutherland, and Gary W. Towsley. *An Introduction to Abstract Algebra with Notes to the Future Teacher*. Upper Saddle River, NJ: Pearson Prentice Hall, 2007. 436 pp. ISBN: 978-0-13-101963-8 (cit. on p. 12).

[27] Oliver Nightingale. *Olivernn/Lunr.Js*. Feb. 20, 2021. URL: https://github.com/olivernn/lunr.js (visited on 02/20/2021) (cit. on p. 12).

[28] Olly Betts and Richard Boulton. *Xapian/Xapian*. Xapian, Feb. 19, 2021. URL: https://github.com/xapian/xapian (visited on 02/20/2021) (cit. on p. 12).

[29] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on p. 4).

[30] Jeffrey M. Perkel. "Why Scientists Are Turning to Rust". In: *Nature* 588.7836 (7836 Dec. 1, 2020), pp. 185–186. DOI: 10.1038/d41586-020-03382-2. URL: https://www.nature.com/articles/d41586-020-03382-2 (visited on 02/20/2021) (cit. on p. 3).

[31] Philip Picton. *Neural Networks*. Basingstoke, Hampshire ; New York: Palgrave, 1994. 195 pp. ISBN: 978-0-333-94899-6 (cit. on p. 4).

[32] Tim Roughgarden, director. *Quicksort Overview*. Jan. 28, 2017. URL: https://www.youtube.com/watch?v=ETo1cpLN7kk&list=PLEAYkSg4uSQ37A6_NrUnTHEKp6EkAxTMa&index=25 (visited on 02/15/2021) (cit. on p. 11).

[33] Enmei Tu et al. *A Graph-Based Semi-Supervised k Nearest-Neighbor Method for Nonlinear Manifold Distributed Data Classification*. June 3, 2016. arXiv: 1606.00985 [cs, stat]. URL: http://arxiv.org/abs/1606.00985 (visited on 12/02/2020) (cit. on p. 12).

[34] US National Institute of Standards and Technology. *Text REtrieval Conference (TREC) Home Page*. URL: https://trec.nist.gov/ (visited on 02/20/2021) (cit. on p. 12).

[35] Vik Singh. *A Comparison of Open Source Search Engines*. Vik's Blog. July 6, 2009. URL: https://partyondata.com/2009/07/06/a-comparison-of-open-source-search-engines-and-indexing-twitter/ (visited on 02/20/2021) (cit. on pp. 3, 12).

[36] vz. *Go-Ego/Riot*. ego, Feb. 20, 2021. URL: https://github.com/go-ego/riot (visited on 02/20/2021) (cit. on p. 12).

[37] S. K.M. Wong and Y. Y. Yao. "Linear Structure in Information Retrieval". In: *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '88. New York, NY, USA: Association for Computing Machinery, May 1, 1988, pp. 219–232. ISBN: 978-2-7061-0309-4. DOI: 10.1145/62437.62452. URL: https://doi.org/10.1145/62437.62452 (visited on 02/14/2021) (cit. on p. 3).

[38] Yuri Schapov, Andrew Aksyonoff, and Ilya Kuznetsov. *Sphinxsearch/Sphinx*. Sphinx Technologies Inc, Feb. 18, 2021. URL: https://github.com/sphinxsearch/sphinx (visited on 02/20/2021) (cit. on p. 12).

[39] Jinzhou Zhang. *Lotabout/Skim*. Feb. 19, 2021. URL: https://github.com/lotabout/skim (visited on 02/20/2021) (cit. on p. 13).