Implementation 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 referred to as "/Learning to Rank"* [28]) that began development at Microsoft from 2004 onwards [7], although previous work in this area had already been undertaken as early as the 90s (see generally [13, 12, 13, 14, 51]). These earlier models did not, however, perform well compared to more modern machine learning techniques [31, §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) and 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 [47]).

2 Motivation

Search Engines implement more sophisticated techniques to rank results, one such example being TF-IDF weighting [32] , well established search engines such as *Apache Lucene* [2] and *Xapian* [20], 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 younger search engines in more modern languages (i.e. `Go` and `Rust`). This is important because these more modern languages are more accessible [19] and memory safe [38] 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 §6.1 of the appendix.

3 Implementation

3.1 Neural Networks

The Ranknet method is typically implemented using a Neural Network[1], although other machine learning techniques can also be used [7, 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

[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 §4.3

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

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

the *Gradient Descent Technique* (although there are other optimisation algorithms such as RMSProp and AdaGrad [33] that can be shown to perform better [4]). The specifics of Neural Networks are beyond the scope of this paper (see [17] or more generally [41]).

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 section (§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 §6.4 for more specific guidance.

Code listings The code listings provided will use a standard set of import statements (see §6.2) and so they will be omitted from the listings, for more comprehensive guidance on implementing this code refer to the documentation page[4] that accompanies the `git` repo.

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), this can be implemented using sci kit learn as shown below and visualized[5] in figure 1.[6]

In order to fit a Neural Network the *PyTorch* package can be used [37], 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.

[4] crmds.github.io/CRMDS-HDR-Training-2020/

[5] See §6.3 for the specific method definition used to export the data to a `csv`.

[6] Visualisations for this Report were implemented using `org-babel` [11] inside *Emacs* [44] to call ***R*** [43] with *GGPlot2* [49] (and *Tidyverse* [50] generally), the source code for this is avaliable in the report manuscript available in the `git` repository available at github.com/RyanGreenup/ranknet/blob/main/Report/Report.org

```python
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 != "":
        export_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):
        # Create the Scatter Plot
        plt.scatter(X[:, 0], X[:, 1], c=y)
        plt.title("Sample Data")
        plt.show()

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

    return torch_data

# 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")
```

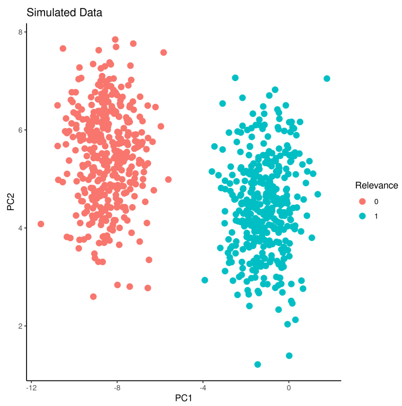

Figure 1: Generated data, output classes denote document relevance and the axis features or principle components

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:

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

        self.bi = torch.randn(hidden_size,
                              dtype=dtype,
                              requires_grad=True,
                              device=dev)
        self.bo = torch.randn(output_size,
                              dtype=dtype,
                              requires_grad=True,
                              device=dev)

        self.σ = torch.randn(1, dtype=dtype, requires_grad=True, device=dev)

        self.losses = []        # List of running loss values
        self.trainedQ = False   # Has the model been trained yet?

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

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

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

A model can then be instantiated, a 2-3-1 model has, arbitrarily, been implemented in this case:[7]

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

# Set Torch Parameters
```

[7]note that the model has not yet been trained, the weights are random and the model output is not related to the data at all.

```
dtype = torch.float
dev = test_cuda()

# Make the Data
X_train, X_test, y_train, y_test = make_data(
    n=100, create_plot=True, dtype=dtype, dev=dev)

# Create a model object
model = three_layer_classification_network(
    input_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")
```

```
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 [8]:

```
class three_layer_classification_network(nn.Module):
    # __init__ method goes here, see above
    # ...
    # ...

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

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

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

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

            # Calculate the Gradients with Autograd
            loss.backward()

            with torch.no_grad():
                # Update the Weights with Gradient Descent
```

[8]This class definition is incomplete and serves only to show the method definition corresponding to the original class shown in §3.3

```
                self.wi -= η * self.wi.grad; self.wi.grad = None
                self.bi -= η * self.bi.grad; self.bi.grad = None
                self.wo -= η * self.wo.grad; self.wo.grad = None
                self.bo -= η * self.bo.grad; self.bo.grad = None
                self.σ  -= η * self.σ.grad;  self.σ.grad = None
            bar.next()
        bar.finish()
                # ; Zero out the gradients, they've been used

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

With this definition the model can hence be trained in order to produce meaningful classifications, as shown below, due to the simplicity of the data set, this model classifies the points perfectly on the testing set, the training error over time is shown in figure 2.

```
# Make the Data
X_train, X_test, y_train, y_test = make_data(
    n=100, create_plot=True, dtype=dtype, dev=dev)

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

# Train the Model
model.train(X_train, y_train, η=1e-2, iterations=10000)

# Plot the Losses
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))
```

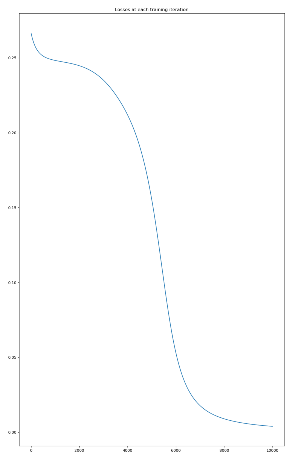

Figure 2: Training error, given by $l(x) = \sum_{i=1}^{n}\left[(x_i - f(x_i))^2\right]$, at each iteration of training

3.5 Implement Ranknet F25F376:05DF04F

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

- Measure loss using a BCE function which is reported in the literature [7, 6] to perform better for Ranking problems.
- Modify the model so that it operates pairwise, such that:
  1. Two points are identified, sent through the neural network and two values returned:

$$s_i = n(\mathbf{X}_i) \tag{1}$$
$$s_j = n(\mathbf{X}_j) \tag{2}$$

     The network previously created can be adapted for this and hence the method will be renamed to `forward_single` and this will represent function $n()$ implemented in (1) and (2)
  2. These values will be combined to give a single value which is intended to measure the model confidence:[9]

$$\hat{P}_{ij} = \text{sig}\left(\sigma, (s_i - s_j)\right), \quad \exists \sigma \in \mathbb{R} \tag{3}$$
$$= \frac{1}{1 + e^{\sigma \cdot (s_i - s_j)}} \tag{4}$$

  3. The range of (4) is the interval $\hat{P}_{ij} = [0, 1]$, let $\bar{P}_{ij}$ be the known probability[10] that $\mathbf{X}_i \triangleright \mathbf{X}_j$, the simulated data has a boolean range of $\bar{P}_{ij} \in \{0, 1\}$, this can be recast to $\{-1, 0, 1\}$ and then linearly scaled to $[0, 1]$ like so:

$$\bar{P}_{ij} \leftarrow p_i - p_j \tag{5}$$
$$\bar{P}_{ij} \leftarrow \frac{1 + \bar{P}_{ij}}{2} \tag{6}$$

These modifications only need to be made to the neural network class like so:

```
class three_layer_ranknet_network(nn.Module):
    # __init__ method
    # ...
    # ...

    def forward(self, xi, xj):
```

[9] This value is a measurement of the models "confidence" but could be extended to represent the "measured probability" of one item being ranked higher than an other (e.g. the probability that a person would rank one type of wine as better than the other in a random sample).

[10] Note the convention to that $\triangleleft$, $\triangleright$ denote the ranking of two observations [6]

```
        si = self.forward_single(xi)
        sj = self.forward_single(xj)
        out = 1 / (1 + torch.exp(-self.σ * (si - sj)))
        return out

    def forward_single(self, x):
        x = torch.matmul(x, self.wi).add(self.bi)
        x = torch.sigmoid(x)
        x = torch.matmul(x, self.wo).add(self.bo)
        x = torch.sigmoid(x)

        return x

    def loss_fn(self, xi, xj, y):
        y_pred = self.forward(xi, xj)
        loss = torch.mean(-y * torch.log(y_pred) -
                          (1 - y) * torch.log(1 - y_pred))
        return loss

   def pairwise(iterable):
       "pairwise([1,2,3,4]) --> [(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]"
       s = list(iterable)
       pair_iter = chain.from_iterable(combinations(s, r) for r in [2])
       return pair_iter
```

The training method must be adapted to interact with these changes like so:[11]

```
class three_layer_ranknet_network(nn.Module):
    # __init__ method
    # ...
    # ...
    def train(self, x, target, η=1e-2, iterations=4e2):
        self.trainedQ = True
        # Create a progress bar
        bar = Bar('Processing', max=iterations)
        # Train for a number of iterations
        for t in range(int(iterations)):
            sublosses = []
            # Loop over every pair of values
            for pair in pairwise(range(len(x) - 1)):
                xi, yi = x[pair[0], ], target[pair[0]]
                xj, yj = x[pair[1], ], target[pair[1]]

                # encode from {0, 1} to {-1, 0, 1}
                y = yi - yj

                # Scale between {0,1}
                y = 1 / 2 * (1 + y)

                # Calculate y, forward pass
                y_pred = self.forward(xi, xj)

                # Measure the loss
                loss = self.loss_fn(xi, xj, y)
                sublosses.append(loss.item())

```

[11]Note the definition of the `pairwise` function, this was incorrectly implemented initially (`f25f376`) and rectified shortly after (`05df04f`). see §6.4.1

```
                # Calculate the Gradients with Autograd
                loss.backward()

                # Update the Weights with Gradient Descent
                # ; Zero out the gradients, they've been used
                with torch.no_grad():
                    self.wi -= η * self.wi.grad; self.wi.grad = None
                    self.bi -= η * self.bi.grad; self.bi.grad = None
                    self.wo -= η * self.wo.grad; self.wo.grad = None
                    self.bo -= η * self.bo.grad; self.bo.grad = None
                    self.σ  -= η * self.σ.grad ; self.σ.grad  = None

            self.losses.append(np.average(sublosses))
            bar.next()
        bar.finish()
        self.threshold_train(x, target, plot=False)
```

This can then be implemented as before, the loss function is provided at figure 3.

```
# Make the Data
X_train, X_test, y_train, y_test = make_data(
    n=30, create_plot=True, dtype=dtype, dev=dev)

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

# Train the Model
model.train(X_train, y_train, η=1e-1, iterations=1e2)

# Save the Losses
np.savetxt(fname="/tmp/losses.csv", X=model.losses, delimiter=',')
```

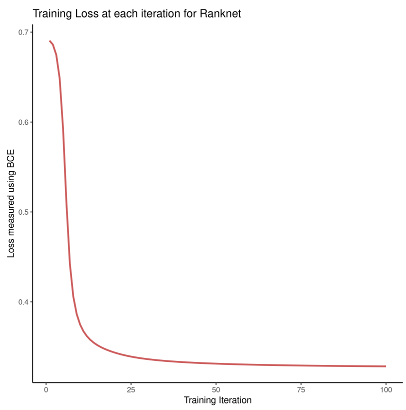

Figure 3: BCE training loss at each iteration for the Ranknet method.

3.6 Implement sorting 99B390A:7D46636

One of the difficulties in implementing this, however, is that it is not simple to determine whether or not the model has classified the data well,[12] In order to address this the model can be implemented to sort the data by ranked values and then visualised. To implement this a derivative of the *quicksort* algorithm was chosen as a sorting function [18], this was implemented by adapting code already available in the literature [24, §4.10] and online [42]:

```python
def split(values, left, right, data, model):
    # Define the leftmost value
    l = (left-1)
    # Set the right value as the pivot
    pivot = values[right]  # TODO The pivot should be random

    for q in range(left, right):
        # Only move smaller values left
        if leq(values[q], pivot, data, model):
            # +1 next left element
            l = l+1
            # Swap the current element onto the left
            values[l], values[q] = values[q], values[l]

    # Swap the pivot value into the left position from the right
    values[l+1], values[right] = values[right], values[l+1]
    return (l+1)


def qsort(values, left, right, data, model):
    if len(values) == 1:
        return values
    if right > left:
        # pi is the index of where the pivot was moved to
        # It's position is now correct
        pi = split(values, left, right, data, model)

        # Do this again for the left and right parts
        qsort(values, left, pi-1, data, model)
        qsort(values, pi+1, right, data, model)

import random

def leq(a, b, data, model):
    score = model.forward(data[a, :], data[b, :])
    if score <= 0.5:
        return True
    if score > 0.5:
        return False

        if (a < b):
            return True
        else:
            return False


```

[12]A naive misclassification method was implemented (`f25f376`), but it was not very insightful and so was omitted from this report.

```python
if DEBUG:
    for i in range(3):
        import random
        values = random.sample(range(9), 7)
        n = len(values)
        print(values)
        qsort(values, 0, n-1, data, model)
        print("==>", values)
```

The data can then be plotted, as in figure 4 and exported like so:[13]

```python
# Main Function
def main():
    # Make the Data
    X_train, X_test, y_train, y_test = make_data(n=100,
                                                 create_plot=True,
                                                 dtype=dtype,
                                                 dev=dev)

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

    # Train the Model
    model.train(X_train, y_train, η=1e-1, iterations=1e2)

    # Visualise the Training Error
    plot_losses(model)

    # Misclassification won't work for ranked data
    # Instead Visualise the ranking
    plot_ranked_data(X_test, y_test, model)


def plot_losses(model):
    plt.plot(model.losses)
    plt.title("Cost / Loss Function for Iteration of Training")
    plt.show()


def plot_ranked_data(X, y, model):
    # Create a list of values
    n = X.shape[0]
    order = [i for i in range(n)]
    # Arrange that list of values based on the model
    quicksort(values=order, left=0, right=(n - 1), data=X, model=model)
    print(order)

    ordered_data = X[order, :]
    y_ordered = y[order]

    np.savetxt("/tmp/ordered_data.csv", X=ordered_data.numpy(), delimiter=',')

    p = plt.figure()
```

[13]In this case the plot has been generated by *GGPlot2* [6]

```
    for i in range(len(ordered_data)):
        plt.text(ordered_data[i, 0], ordered_data[i, 1], i)
    plt.scatter(ordered_data[:, 0], ordered_data[:, 1], c=y_ordered)
    plt.title("Testing Data, with ranks")
    plt.show()


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

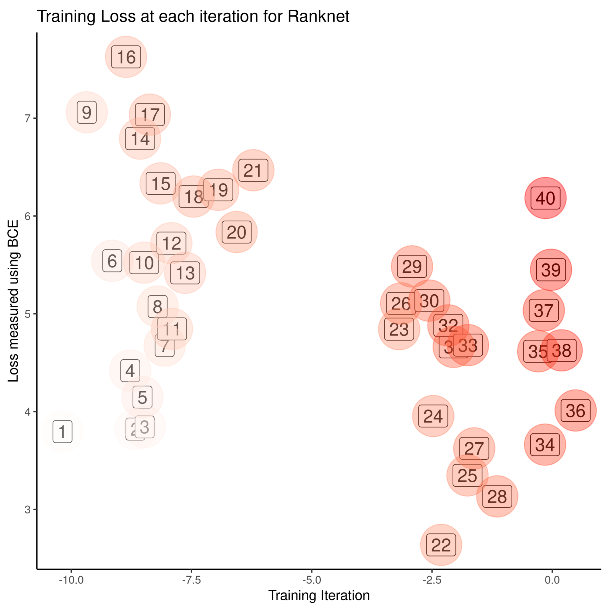

Figure 4: Using Machine Learned Ranking to order the points from most to least relevant

3.6.1 Visualising Model Performance

Although the results in figure 4 appear very encouraging at first, if the same sorting approach is implemented for an untrained model (i.e. a model with random weights), an uncomfortably similar, ordered pattern emerges, this is shown[14] in figure 5.

This pattern could be explained by the fact that the model, trained or untrained, is continuous and so the rank of nearby points could lead the emergence of an ordered pattern.

Investigating types of data where the Ranknet model will produce an ordered pattern only when trained would represent a logical next step. One approach would be to consider the rating of a product (e.g. the quality of wine, see [9]) and train a Ranknet model based only on whether or not one wine is better than the next. The order of the

[14]For Clarity sake the colours have been inverted.

returned results could be compared with the order of the original dataset as well as a random untrained control model in order to evaluate the efficacy of the model.

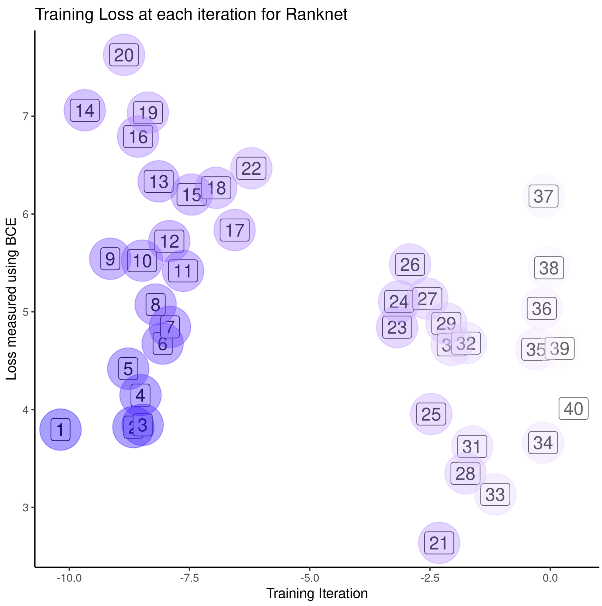

Figure 5: Using Machine Learned Ranking to order the points from most to least relevant

3.7 Enhancing the Model

Further enhancements (such as random batches and alternative optimisers) were made to the model and alternative datasets implemented (see `473dce` → `d11e607`) however the results were mixed and not encouraging, rather an alternative type of data should first be considered and evaluated, as already discussed in §3.6.1.

4 Further Research

The implementation of this technique has proved considerably more difficult than first perceived, more work is required. In this section particular points of improvement are identified.

4.1 Improve the sorting algorithm

The "Quicksort" algorithm likely needs a random pivot to be efficient [45].

4.2 Evaluate Model Performance

It is still not clear how the performance of Ranknet compares to traditional approaches implemented by search engines (see §6.1), 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 [46] using a cummulated-gain cost function [22] as demonstrated in previous work [47].

It is not yet clear:

1. How this could be applied for a broad range of queries
2. How to effectively measure the "correctness" of ranked documents

4.3 Evaluate alternative machine learning models

An open question is whether or not alternative machine learning algorithms such as trees and SVMs could be used to fit the Ranknet model in an effective fashion, this could be ideal for performance concerns as deep neural networks can be resource intensive.

Another question is how Ranknet compares to simply using classification approaches to identify documents that might be relevant.

5 Conclusion

The Ranknet approach to machine-learned ranking looks promising, but it is difficult to implement and more study is needed to evaluate whether or not it brings significant advantages over traditional approaches and whether or not there are effective ways to apply the model to search problems for a broad scope of queries.

6 Appendix

6.1 Search Engines

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

- Apache lucene/Solr (Java) [2]
    - Implemented by DocFetcher [10]
- Sphinx (C++) [52]
- Xapian (C++) [36]
    - Implemented by Recoll [23]
- Zettair (C) [21]

More Modern Search engines include:

- Bleve Search (`Go`) [32]
- Riot (`Go`) [48]
- LunrJS (`JS`) [35]
- SimSearch (`Rust`) [29]
- Tantivy (`Rust`) [8]

6.1.1 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` [5]
- `fzy` [16]
- `go-fuzzyfinder` [26]
- `peco` [27]
- Skim [53]
- Swiper [25]

6.2 Import Statements

The following import statements were included, where used, [15] separate scripts were used to make the model as modular as possible, such corresponding inputs have also been listed:

```python
# Import Packages
from itertools import chain
from itertools import combinations
from itertools import tee
from progress.bar import Bar
import math as m
import matplotlib.pyplot as plt
import numpy as np
import random
import sys
import sys
import torch
import torch
from torch import nn

# Sepereate Scripts lcated below main
```

[15]Including `import` statements where they are not used is fine, other than complaints from a *linter* following *PEP* [34] (e.g. autopep [15]) the code will function just fine.

```
from ranknet.test_cuda import test_cuda
from ranknet.make_data import make_data
from ranknet.neural_network import three_layer_ranknet_network
from ranknet.quicksort import quicksort
```

6.3 Export Data Method

The data was exported by printing the values to a text file like so:

```python
def export_data(X, y, export):
    try:
        os.remove(export)
        print("Warning, given file was over-written")
    except:
        pass

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

6.4 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 tools like magit [30] and git-timemachine [40] (or GitLens [1] and git-temporal [3] in VsCode) in order to effectively preview the changes at each step, alternatively a pager like bat [39] can also be used with something like fzf [5] like so:

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

6.4.1 Version Control Log for Walkthrough

Commit ID	***Message***
`ed5f4cf`	*Initial Commit*
`075acf9`	*Walkthrough Initial Commit*
`cf9ab26`	*Generate data to use for classification*
`7291112`	*Create a Neural Network Model*
`7d46636`	*Implement gradient descent to train neural network*
`f25f376`	*Adapt Neural Network to perform Ranking*
`42509ab`	*Implement sorting algorithm to visualise ranking order*
`05df04f`	*Adapt Neural Network to perform Ranking*
`99b390a`	*Implement sorting algorithm to visualise ranking order*
`473dce3`	*Implement optimizer to replace mere gradient descent*
`4141e92`	*Train Model using Batches not entire dataset*
`a2671a6`	*Format code to make it more readable*
`d11e607`	*plot and only train on different ranked pairs*

References

[1] Eric Amodio. *Eamodio/Vscode-Gitlens*. Nov. 6, 2016. URL: https://github.com/eamodio/vscode-gitlens (visited on 02/20/2021) (cit. on p. 17).

[2] 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. 2, 15).

[3] Bee Wilkerson. *Git-Temporal/Git-Temporal: The Mono Repo for Components, Data Layer, and Interfaces of Git-Temporal*. Sept. 12, 2018. URL: https://github.com/git-temporal/git-temporal (visited on 02/20/2021) (cit. on p. 17).

[4] 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. 3).

[5] Junegunn Choi. *Junegunn/Fzf*. Feb. 20, 2021. URL: https://github.com/junegunn/fzf (visited on 02/20/2021) (cit. on pp. 16, 17).

[6] 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. 8).

[7] 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. 2, 8).

[8] 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. 16).

[9] A Cortez et al. "Modeling Wine Preferences by Data Mining from Physicochemical Properties. In Decision Support Systems". In: *Elsevier* 47.4 (2009), pp. 547–553 (cit. on p. 13).

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

[11] Carsten Dominik. *Org Mode Reference Manual*. Place of publication not identified: 12TH MEDIA SERVICES, 2018. ISBN: 978-1-68092-165-6 (cit. on p. 3).

[12] 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. 2).

[13] 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. 2).

[14] 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. 2).

[15] Hideo Hattori. *Autopep8: A Tool That Automatically Formats Python Code to Conform to the PEP 8 Style Guide.* Version 1.5.5. URL: https://github.com/hhatto/autopep8 (visited on 02/20/2021) (cit. on p. 16).

[16] John Hawthorn. *Jhawthorn/Fzy.* Feb. 20, 2021. URL: https://github.com/jhawthorn/fzy (visited on 02/20/2021) (cit. on p. 16).

[17] HMKCode. *Backpropagation Step by Step.* URL: https://hmkcode.com/ai/backpropagation-step-by-step/ (visited on 02/16/2021) (cit. on p. 3).

[18] C. A. R. Hoare. “Algorithm 64: Quicksort”. In: *Communications of the ACM* 4.7 (July 1, 1961), p. 321. ISSN: 0001-0782. DOI: 10.1145/366622.366644. URL: http://doi.org/10.1145/366622.366644 (visited on 02/21/2021) (cit. on p. 11).

[19] 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. 2).

[20] James Aylett. *GSoCProjectIdeas/LearningtoRankStabilisation – Xapian.* Oct. 20, 2019. URL: https://trac.xapian.org/wiki/GSoCProjectIdeas/LearningtoRankStabilisation (visited on 02/20/2021) (cit. on p. 2).

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

[22] 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. 15).

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

[24] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language.* 2nd ed. Englewood Cliffs, N.J: Prentice Hall, 1988. 272 pp. ISBN: 978-0-13-110370-2 978-0-13-110362-7 (cit. on p. 11).

[25] Oleh Krehel. *Abo-Abo/Swiper.* Feb. 20, 2021. URL: https://github.com/abo-abo/swiper (visited on 02/20/2021) (cit. on p. 16).

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

[27] lestrrat. *Peco/Peco.* peco, Feb. 18, 2021. URL: https://github.com/peco/peco (visited on 02/20/2021) (cit. on p. 16).

[28] 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. 2).

[29] Andy Lok. *Andylokandy/Simsearch-Rs*. Feb. 15, 2021. URL: https://github.com/andylokandy/simsearch-rs (visited on 02/20/2021) (cit. on p. 16).

[30] *Magit/Magit*. Magit, Oct. 19, 2008. URL: https://github.com/magit/magit (visited on 02/20/2021) (cit. on p. 17).

[31] 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. 2).

[32] Marty Schoch and Abhinav Dangeti. *Bleve Search Documentation*. URL: http://blevesearch.com/ (visited on 02/20/2021) (cit. on pp. 2, 16).

[33] 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. 3).

[34] Nick Coghlan. *PEP 8 – Style Guide for Python Code*. Python.org. Aug. 1, 2001. URL: https://www.python.org/dev/peps/pep-0008/ (visited on 02/20/2021) (cit. on p. 16).

[35] Oliver Nightingale. *Olivernn/Lunr.Js*. Feb. 20, 2021. URL: https://github.com/olivernn/lunr.js (visited on 02/20/2021) (cit. on p. 16).

[36] 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. 15).

[37] 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. 3).

[38] 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. 2).

[39] David Peter. *Sharkdp/Bat*. Apr. 21, 2018. URL: https://github.com/sharkdp/bat (visited on 02/20/2021) (cit. on p. 17).

[40] Peter Stiernström and Basti. *Emacsmirror/Git-Timemachine*. Emacsmirror, June 29, 2014. URL: https://github.com/emacsmirror/git-timemachine (visited on 02/20/2021) (cit. on p. 17).

[41] Philip Picton. *Neural Networks*. Basingstoke, Hampshire ; New York: Palgrave, 1994. 195 pp. ISBN: 978-0-333-94899-6 (cit. on p. 3).

[42] *Python Program for QuickSort*. GeeksforGeeks. Jan. 7, 2014. URL: https://www.geeksforgeeks.org/python-program-for-quicksort/ (visited on 02/15/2021) (cit. on p. 11).

[43] R Core Team. *R: A Language and Environment for Statistical Computing*. manual. R Foundation for Statistical Computing. Vienna, Austria, 2021. URL: https://www.R-project.org/ (cit. on p. 3).

[44] Richard Stallman. *GNU Emacs Manual.* 15.ed., updated for Emacs version 21.2. Boston, Mass: Free Software Foundation, 2002. 602 pp. ISBN: 978-1-882114-85-6 (cit. on p. 3).

[45] 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. 14).

[46] 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. 15).

[47] 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. 2, 15).

[48] vz. *Go-Ego/Riot.* ego, Feb. 20, 2021. URL: https://github.com/go-ego/riot (visited on 02/20/2021) (cit. on p. 16).

[49] Hadley Wickham. *Ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org (cit. on p. 3).

[50] Hadley Wickham et al. "Welcome to the tidyverse". In: *Journal of Open Source Software* 4.43 (2019), p. 1686. DOI: 10.21105/joss.01686 (cit. on p. 3).

[51] 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. 2).

[52] 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. 15).

[53] Jinzhou Zhang. *Lotabout/Skim.* Feb. 19, 2021. URL: https://github.com/lotabout/skim (visited on 02/20/2021) (cit. on p. 16).