**CE306/CE706 (Spring 2023)**

**Lab script 1 Windows**

It is now time to get some practical experience with information retrieval tools. [Elasticsearch](https://www.elastic.co/) is one of the most powerful and widely used open-source search engines. It is simple, scalable, and highly efficient and can manage structured as well as unstructured data. The examples look a bit like database examples but that is because the data comes with some structure. You will see that the pre-processing steps that you have come across in this week’s lecture will actually be performed on each field individually (check the [guide](https://www.elastic.co/guide/en/elasticsearch/reference/7.17/analysis.html) to find out more).

Please do not stop working with Elasticsearch when you finish this lab session. Keep playing around with it, install it on your own computer, use it in your own project, submit code, and join the community, among others. [Here](https://www.elastic.co/guide/en/elasticsearch/reference/7.17/getting-started.html) is a starting point to explore the framework before you approach the steps below.

**Start Elasticsearch and Kibana**

This lab assumes that you are using the Windows environment. If you start your machine with Ubuntu, feel free to leave the Ubuntu environment and reboot to Windows on your machine. If you are using the lab machine Elasticsearch and Kibana are already installed for the use of Lab 1 and Lab 2. The instructions below assume that you are using the lab machine with a Windows environment.

*NOTE: Throughout the labs, we are using version 7.17.5 of Elasticsearch (and Kibana) as a reference point because we know that this works with the CSEE Lab settings. Later versions have however been released and offer a lot of new and improved features. Feel free to install the latest release (at least when you install the software on your own computer). With every new release there might be some variations in the commands, so please feel free to explore which commands would work on your release by searching for the appropriate supported commands online.*

To start first open the file explorer and click **This PC** then **Local Disc (C:)**

Graphical user interface, application

Description automatically generated

You will find **elasticsearch** and **kibana** are already installed in the C:/elasticsearch and C:/kibana folder.

Graphical user interface

Description automatically generated

You will need first start the elasticsearch under C:/elasticsearch/bin/elasticsearch.bat

Graphical user interface, application

Description automatically generated

You will see the CMD windows pop up, let it work until it finishes all the lines, it may take about a minute.

To check if Elasticsearch is properly installed, you will need to open either Google Chrome or any web browser and access this URL: <http://localhost:9200/>

If you successfully install it, you will have it displayed like this on your browser

Text, letter

Description automatically generated

Once you checked the elasticsearh is running you can now start the kibana by clicking the C:/kibana/bin/kibana.batGraphical user interface, application

Description automatically generated

Again, you will see the CMD windows pop up, let it work until it finishes all the lines, it may take about a minute or slightly longer.

To check if Kibana is properly installed, you will need to open either Google Chrome or any web browser and access this URL: <http://localhost:5601/>

If you successfully started, you will have it displayed like this on your browser

Graphical user interface, application, website

Description automatically generated

To proceed with the LAB01 here or for your assignments, you need to do the following steps: Keep in mind that you will need to have both **elasticsearch.bat** and **kibana.bat** run on your laptop/desktop all the time when you are working on this, you can minimize them and let they run in the background.

**Indexing Documents**

First, download the **accounts.json** file data to your machine using the following link and unzip it.

<https://download.elastic.co/demos/kibana/gettingstarted/accounts.zip>

Scroll down and click on the following menu on the Kibana page <http://localhost:5601/> as the following screenshot (Dev Tools)

Graphical user interface, application, website

Description automatically generated

Then you will be promoting to this following page, to create an index with the **account.json** file simply use **POST /bank/\_bulk** command and copy and paste the contents of the whole JSON document. After that press the small arrow on the right corner of the frame.

Graphical user interface, text, application

Description automatically generated

**Searching**

Let's start with a query that matches all the documents.

GET /bank/\_search

{

"query": { "match\_all": {} }

}

You can see from the hits.total field we matched 1000 documents and, by default, the first 10 are shown.

**Pagination**

The query we performed only showed 10 documents. We can show the next 10 as follows:

GET /bank/\_search

{

  "query": { "match\_all": {} },

  "from": 10,

  "size": 10

}

from sets from which document we will start, size sets how many documents are shown.

**Querying Full Text**

A full-text query will take multiple words, and search for all of them giving each document a score based on how close it was. Let's try an example:

GET /bank/\_search

{

    "query": {

        "match" : {

            "address" : "national street"

        }

    }

}

The address bit of the query tells us which field we will be matching.

Looking at the results from this query it seems like we searched wrong. There isn't a "National Street", there's a "National Drive" though. Notice how results that contained both "national" and "street" were returned. match defaults to being an or query, so it will match documents containing either of the two terms. If we change the operator to “and” our "national street" the search will return 0 results, because the terms "national" and "street" are not present together in any address field.

GET /bank/\_search

{

  "query": {

    "match": {

      "address": {

        "query": "national street",

        "operator": "and"

      }

    }

  }

}

However, let's try "National Drive" with "and":

GET /bank/\_search

{

  "query": {

    "match": {

      "address": {

        "query": "drive national",

        "operator": "and"

      }

    }

  }

}

I've deliberately reversed the terms. Note how the search still works. It's considered the terms independently, in any order, but they must both be in the address field for the document to be a hit.

But what if we really wanted to match "drive national" exactly.

**Matching Exact Phrases**

match\_phrase matches "National Drive" exactly. This gives only 1 result.

GET /bank/\_search

{

  "query": {

    "match\_phrase": {

      "address": "national drive"

      }

   }

}

Reversing the terms as we did in the previous example does not work here. This matches the exact phrase. Sometimes, however, you only have part of the phrase.

**Matching Part of Phrases**

This type of search matches a phrase with a wildcard. An example, let's try and use this to make an autocomplete/search suggestion. When the user starts typing, we could suggest what they may want to type.

For example, try searching for a firstname:

GET /bank/\_search

{

  "query": {

    "match\_phrase\_prefix": {

      "firstname": "Jo"

      }

    }

}

You will notice you are shown lots of records with firstnames that start with Jo including **Jo**sephine, **Jo**sephina, **Jo**sie, and many others.

**Matching Multiple Fields**

It's common in a search engine, in which you would want to match multiple fields with your query type. Let's say for example we typically search by lastname when looking up customer accounts, but sometimes we get given a name and we don't know whether it is a firstname, or a lastname. To improve our recall we want to search both fields.

To achieve this we can use a multi-match:

GET /bank/\_search

{

  "query": {

    "multi\_match": {

      "query": "Francis",

      "fields": ["firstname","lastname"]

    }

  }

}

This hasn't quite worked through. "Francis Beck" came before "Kelli Francis". We can *boost* the lastname field in this search to make it more important:

GET /bank/\_search

{

  "query": {

    "multi\_match": {

      "query": "Francis",

      "fields": ["firstname","lastname^2"]

    }

  }

}

Now "Kelli Francis" comes first. To match the query in any fields simply assign an empty list to “fields”, i.e. “fields”:[].

**Sorting**

The query below sorts the results in descending order (desc) by balance.

GET /bank/\_search

{

  "query": { "match\_all": {} },

  "sort": { "balance": { "order": "desc" } }

}

Try repeating the search suggest exercise from earlier sorted alphabetically by firstname.

**Filtering**

Filtering uses bool queries. These are queries that have scores of either 0 or 1. We can extend our earlier auto-complete example.

Let's pretend we have a bank office in the state of Florida, so we are only interested in our search showing those records:

GET /bank/\_search

{

  "query": {

    "bool": {

      "must": {

        "match\_phrase\_prefix": {

          "firstname": "Jo"

        }

      },

      "filter": {

        "term": {

          "state.keyword": "FL"

        }

      }

    }

  }

}

Now there are only two results.

It isn't just the terms we can filter by, we can filter by numeric ranges. Let's pretend we are searching for someone with a name that starts with "Jo", but this time, we are in the mortgage department of the bank, and only process customers that hold a balance of over 11,000 as the bank says we aren't allowed to issue a mortgage unless they have more savings.

GET /bank/\_search

{

  "query": {

    "bool": {

      "must": {

        "match\_phrase\_prefix": {

          "firstname": "Jo"

        }

      },

      "filter": {

        "range": {

          "balance": {

            "from": 11000

          }

        }

      }

    }

  }

}

Rather than 10 results, we now have 7, excluding the results that had a balance of less than 11,000.

**Exercises**

**Exercise 1**

The LA office has bought you in as a consultant. They have lots of company accounts and most often customers call up quoting their company name/employer. They want to be able to search by that, firstname, and lastname. Then they would like all the results returned in alphabetical order by the company name. They don't want to see results from any other offices though.

**Exercise 2**

The bank HQ marketing department wants to run a promotion. They're really interested in marketing to their under-30 high-income customers. They'd like a report that shows only customers under 30, in descending order of balance.

**Exercise 3**

The customer records department is having a problem with their existing system. It keeps check of all the addresses of customers. When they search "Clay", the system does the following search:

GET /bank/\_search

{

    "query": {

        "multi\_match" : {

            "query" : "Clay",

"fields": []

        }

    }

}

But that brings up someone with the name "Clay" first. They would like to change it so that anything with the city Clay, or Clay in the address is shown before anyone with the name Clay.

**Further Ideas**

This has been a quick introduction to the use of Elasticsearch and Kibana. The next step is making Elasticsearch part of your wider application for your specific use. There is a wide range of libraries for different languages and frameworks that can assist you in passing queries to Elasticsearch and retrieving data that you can then display to your users. You may want to pick a library or framework and experiment with displaying some data as part of a web application.

**Acknowledgements**

This lab script was originally prepared by Chris Madge.