ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

WikiMining - Finding gems in Wikipedia using submodular function maximisation

Master Thesis

Victor Ungureanu

April 17, 2014

Advisors: Prof. Dr. A. D. Visor, Dr. P. Ostdoc

Department of Computer Science, ETH Zürich

Abstract

This example thesis briefly shows the main features of our thesis style, and how to use it for your purposes.

Contents

Chapter 1

Introduction

Dummy text.

1.1 Contributions

Dummy text.

Chapter 2

Related work

Dummy text.

Chapter 3

Preliminaries

Dummy text.

3.1 Term frequency - inverse document frequency

Dummy text.

3.2 Cosine similarity

Dummy text.

3.3 Locality-sensitive hashing

Dummy text.

Chapter 4

Submodularity, coverage, summarisation

In this chapter we only provide an overview of what submodularity is, why it is useful and how it can be applied to summarisation. If you are interested in a deeper understanding of submodular functions and their many other use-cases, you can read survey [**?**] on *submodular function maximisation*.

4.1 Submodular functions

In this section we introduce what are submodular functions and why they are important. We also offer a couple of examples to shed some light on how these functions behave. Note that we will refer to these examples from some of the other sections.

4.1.1 Definitions

Definition 4.1 (Submodularity) *A set function $f : 2^D \rightarrow \mathbb{R}$ is submodular iff $\forall S, T$ such that $S \subseteq T \subseteq D$ and $\forall d \in D \setminus T$ we have*

$$f(S \cup d) - f(S) \geq f(T \cup d) - f(T).$$

Intuitively this means that a new element's impact can never be higher in the future than it currently is, an effect also knows as *diminishing returns*.

Definition 4.2 (Monotonicity) *A set function $f : 2^D \rightarrow \mathbb{R}$ is monotone iff $\forall S, T$ such that $S \subseteq T \subseteq D$ we have*

$$f(S) \leq f(T).$$

From Definition 4.1 and Definition 4.2 we derive [**?**]:

Proposition 4.3 (Monotone submodular) *A set function $f : 2^D \to \mathbb{R}$ is monotone submodular iff $\forall S, T$ such that $S \subseteq T \subseteq D$ and $\forall d \in D$ we have*

$$f(S \cup d) - f(S) \geq f(T \cup d) - f(T).$$

Note that in this case we also allow $d \in T$.

Missing figure

Submodularity visual example

4.1.2 Examples and properties

In this subsection we will discuss only monotone submodular functions – used in the other sections – and some of the submodular functions' properties.

Let $\mathcal{D}$ be a universe, $A_1, A_2, \ldots, A_n \subseteq \mathcal{D}$ and $D = 1, 2, \ldots, n$. We can define several functions $f : 2^D - > \mathbb{R}$ that are monotone submodular.

Citation needed / Stanford-lec16.pdf

Definition 4.4 (Set coverage function) $f(S) := |\bigcup_{i \in S} D_i|$.

More generally we can extend the above function as follows.

Definition 4.5 (Weighted set coverage function) *Let $w : \mathcal{D} \to \mathbb{R}_+$ be a non-negative weight function.*
Then $f(S) := w(|\bigcup_{i \in S} D_i|)$.

This differs from Definition 4.4 in that we can sum non-constant weights that depend on the selected elements. From this we can define a more complex, but very useful monotone submodular function that we will use in Section 4.3.

Definition 4.6 (More general weighted set coverage function) *Then $f(S) := w(|\bigcup_{i \in S} D_i|)$.*

Simplify these functions

A very useful property of submodular functions is that the class of submodular functions is closed under non-negative linear combinations.

Citation needed

Proposition 4.7 (Closedness under non-negative linear combinations) *Let $g_1, g_2, \ldots, g_n : 2^D \to \mathbb{R}$ be submodular functions and $\lambda_1, \lambda_2, \ldots, \lambda_n \in \mathbb{R}_+$.*

Then

$$f(S) := \sum_{i=1}^{n} \lambda_i g_i(S)$$

is submodular.

This property is important because it allows us to easily construct new submodular function by combining multiple simpler submodular functions.

4.2 Submodular function maximisation

4.2.1 Problem statement

Given a submodular function f we are interested in maximising its value on set S given some constraints on S. A common constraint on S is the *cardinality constraint* which limits the size of set S. Formally, we are interested in computing:

$$\max_{S \subseteq D} f(S) \text{ subject to } |S| \leq k, \text{ for some } k \tag{4.1}$$

Most of the time we are actually interested in computing the set S that maximises our function f, so Equation 4.1 becomes:

$$\arg\max_{S \subseteq D} f(S) \text{ subject to } |S| \leq k, \text{ for some } k \tag{4.2}$$

4.2.2 Greedy maximisation

Optimally solving Equations 4.1, 4.2 for some function f is *NP-hard* . Fortunate

Citation needed / Feige, 1998

Algorithm 4.1 Greedy submodular function maximisation

```
$S \leftarrow \emptyset$
while $|S| < k$ do
    $d^* \leftarrow \arg\max_{d \in D} f(S \cup d) - f(S)$
    $S \leftarrow S \cup d^*$
end whileAnswer $S$
```

we can devise a *greedy algorithm* that is at most $1 - 1/e$ worse than the best solution for maximising a fixed monotone submodular function f [?]. We present the required steps in Algorithm 4.1.

Discuss about lazy vs. non-lazy + cite

4.2.3 GreeDi protocol

Given that in this thesis we are interested in applying submodular function maximisation to a large corpus we need to find a way to transform the sequential Algorithm 4.1 to run distributively. Fortunately, there exists a *greedy distributed submodular maximisation* protocol described in Algorithm 4.2 that partitions the data into subsets and then runs Algorithm 4.1 on each individual partition. This approach has the benefit that it gracefully degrades the approximation guarantees based on the number of partitions and, more importantly, for many types of data it offers approximation guarantees close to the ones offered by the sequential version, also with great experimental results [**?**] – results that are almost identical or very similar to the sequential algorithm.

Algorithm 4.2 Greedy Distributed Submodular Maximisation (GreeDi). Adapted from [**?**] with $l = k$

```
$D$ :=set of all elements
$p$ :=number of partitions
$k$ :=number of selected elements
1: Partition $D$ into $p$ sets: $D_1, D_2, \ldots, D_p$
2: Run Greedy Algorithm 4.1 on each set $D_i$ to select $k$ elements in $T_i$
3: Merge the answers: $T = \bigcup_{i=1}^{p} T_i$
4: Run Greedy Algorithm 4.1 on T to select the final $k$ elements in $S$
5: Answer $S$
```

4.3 Word coverage

One of the basic ways of finding significant multi-document summaries is to find a good measure for a document's *information coverage*. One such proposed metric is *word coverage*, a method which argues that covering words is a good indication of covering information. This method has been used before in document summarisation and it extends naturally to multi-document summarisation [**?**].

4.3.1 Definition

Sipos et al [**?**] propose a way to adapt a known submodular function to use word coverage as a information measure. We define this function in Definition 4.8.

Definition 4.8 (Word coverage) *Let:*

D be a set of documents,

W be a set of all words from all documents in D.

Then we define:

$$f : 2^D \to \mathbb{R}_+$$
$$f(S) := \sum_{w \in W} \theta(w) \max_{d \in S} \varphi(d, w) \; (\forall S \subseteq D)$$

where:

$\theta : W \to \mathbb{R}_+$ *represents the importance of word* w,
$\varphi : D \times W \to \mathbb{R}_+$ *represents the coverage of word* w *in document* d *and it is usually chosen to be tf-idf*(d, w).

Remember that we described term-frequency–inverted-document-frequency (tf-idf) in Section .

Preliminaries

Proposition 4.9 *Function* f *from Definition 4.8 is monotone submodular.*

proof: paper / paper ref [8] ?

4.3.2 Rationale

The word coverage function defined in Definition 4.8 on the facing page promotes word diversity while trying to minimize eccentricity. Intuitively, maximising $\varphi(d, w$, seen as tf-idf, prefers selecting documents that have a lot of words rarer in the other documents. This promotes diversity, but also increases the eccentricity of the selected documents. To dampen the eccentricity of picked documents we introduce $\theta(w)$ so that we include the importance of the word in the selection process. This will counterbalance the eccentricity of words by preferring more common significant words[1].

4.4 Document influence

Dummy text.

[1]Not to be confused with stop words.

Chapter 5

Massive corpus summarisation

In this chapter we present novel submodular functions that can be applied to find the most important documents out of millions of interconnected documents such as parts of the web (or even the whole web). Our approach expands on the ideas of *multi-document summarisation using submodular word coverage* [**?**] presented in Chapter 4 on page 7. This chapter presents the authors' novel contributions to the problem of *massive corpus summarisation*.

5.1 Scaling from thousands to millions

Sipos et al [**?**] present an interesting way to summarise a document corpus using *submodular functions*. However, their focus is on the different ways one can view summarisation and their methods are hard or even impossible to scale beyond tens of thousand of documents. In this thesis we concentrate on finding the most important articles out of a massive corpus of interconnected documents. This can be viewed as a multi-document summarisation task, but given the small ratio between the number of selected documents and the total size of the corpus it also bears some differences from the classical view on multi-document summarisation. Concretely, we aim to select tens of *Wikipedia* articles from about 1.3 million *Wikipedia* pages[1].

This problem is two-folded. On one hand, we have to adapt the submodular functions so that they yield satisfying results for such a large *'compression' ratio* of about 1/50000 (as it is in the case of Wikipedia). On other hand, we can no longer run sequentially; so, in turn, we will employ having all algorithms written in a *MapReduce* [**?**] distributed fashion. If you are interested in more details about the framework we implemented please refer to Chapter **??** on page ?? .

Implementation chapter

[1]This is the total number of human-written articles in *Wikipedia* before Oct 2012

5.2 Graph coverage

In this section we present a submodular function defined on the *inlinks* graph structure of the considered set of documents. Instead of only evaluating the selected document we argue that each node expands its influence among its neighbouring articles.

5.2.1 Definition

Definition 5.1 (Graph coverage) *Let $G = (D, E)$ be a graph where:*

$$D := \text{set of document vertices,}$$
$$E := \text{set of edges.}$$

Then we define:

$$f : 2^D \to \mathbb{R}_+$$
$$f(S) := |\bigcup_{d \in S} V_d|$$

where:

$$V_d := \{d\} \cup \{v \in D | (v, d) \in E\}.$$

Proposition 5.2 *Graph coverage from Definition 5.1 is monotone submodular. Proof in .*

Appendix

5.2.2 Rationale

Graph coverage is a simple function that aims to measure a document's coverage in terms of vertices it covers, while keeping the selected documents diversified. We argue that each document manifest an aura of influence among the articles that link to it. Consequently, we conjecture that adding the source document – the page that adds the highest number of new inlinks – captures a topic that is both important and sufficiently different from the already selected articles. As a simple extension, Definition 5.1 can be modified to consider neighbours at a radius larger than one (for example, to take into account all nodes that are at most two edges away from the selected document).

5.3 Beyond word coverage

In Section 4.3 on page 10 we introduced *word coverage* as a measure of information coverage based on words that tries to avoid eccentric documents. However, as you can see from our results in Chapter **??** on page ?? , *word*

Results chapter

coverage performs poorer than expected when scaled to *Wikipedia*'s size (millions of documents). In this, chapter we modify the function such that it takes into account more information about documents when comparing them.

5.3.1 Definitions

We remind the reader that in Definition 4.8 on page 10 we described a submodular function that uses word coverage as follows:

$$f : 2^D \to \mathbb{R}_+ \qquad f(S) := \sum_{w \in W} \theta(w) \max_{d \in S} \varphi(d, w) \; (\forall S \subseteq D)$$

For details about the notations used we advise the reader to return to Definition 4.8 on page 10.

In this section we propose explicit definitions for functions $\theta(w)$ and $\varphi(d, w$ such that they yield desirable results even for large-scale corpora. Let:

$$\begin{aligned} D &:= \text{set of documents,} \\ W &:= \text{set of all words from all documents in } D, \\ (D, E) = G &:= \text{ documents' graph.} \end{aligned}$$

Word importance

Word importance is the weight we give to a word in terms of its meaning. As we mentioned before, there has been significant work done on how to properly define it []. Here we use some of the basic metrics such that it easily scales to a massive corpus.

Citation needed

Definition 5.3 (Word importance) *We define word importance*

$$\theta : W \to \mathbb{R}_+$$

in one of the following four (non-equivalent) ways:

1. $\theta(w) := \sqrt{wc(w)}$
2. $\theta(w) := \log_{10}(wc(w))$
3. $\theta(w) := \sqrt{dc(w)}$
4. $\theta(w) := \log_{10}(dc(w))$

where:

$$wc, dc : W \to \mathbb{R}_+$$

$$\begin{aligned} wc(w) &:= \sum_{d \in D} count(w, d), \\ count(w, d) &:= \textit{number of times word } w \textit{ appears in document } d, \\ dc(w) &:= |\{w \in d | d \in D\}|. \end{aligned}$$

tf-idf paper

Functions $wc(w)$ and $dc(w)$ are functions needed to define tf-idf []:

- $wc(w)$ stands for *word count* and represents the number of times word w appears in the whole corpus;
- $dc(w)$ stands for *document count* and represents the number of documents (from the corpus) in which word w appears.

Coverage of a word

The coverage of a word in a document is a measure of how much a word covers a specific concept in that document. In our case we view the coverage of a word as the *meaningfulness* of that word in a document in the context of all other words and documents from the corpus. Here we define novel functions that consider some of the different metrics one can use for this task.

Definition 5.4 (Coverage of a word) *We define coverage of a word*

$$\varphi : D \times W \to \mathbb{R}_+$$

Find a better notation

in one of the following (non-equivalent) ways :

$$\begin{aligned}\varphi(d,w) = \textit{tf-idf}(d,w)\, & [\cdot \#\textit{inlinks}(d)] \\ & [\cdot \#\textit{revisions}(d)] \\ & [\cdot \textit{revisions-volume}(d)],\end{aligned}$$

where:

$$\#\textit{inlinks}, \textit{revisions-volume}, \#\textit{revisions} : D \to \mathbb{R}_+$$

$$\begin{aligned}\#\textit{inlinks}(d) &:= |\{v \in D | (v,d) \in E\}|, \\ \#\textit{revisions}(d) &:= \textit{number of edits of document } d, \\ \textit{revisions-volume}(d) &:= \textit{total size of edits of document } d.\end{aligned}$$

Note that $[\cdot]$ marks an optional term. As such we get twelve different possible functions in total.

Also note that using any explicit forms for *word importance* from Definition 5.3 and for *coverage of a word* from Definition 5.4 maintains the *word coverage* function from Definition 4.8 on page 10 *monotone submodular*.

5.3.2 Rationale

We were unsatisfied with the results of using only tf-idf for the *coverage of a word in a document*, so we thought about natural ways to extend $\varphi(d,w)$ to capture more of the available information. One of the main problems was that although we take into account the *word importance* in Definition 4.8 this is not enough to dampen the eccentricity of the articles. In the following paragraphs we explain how each added term from Definition 5.4 fits in and what is the reasoning behind it. We warn the reader that we offer only intuitive explanations that were empirically confirmed, but no strict formalisms about their validity.

Inlinks

The idea for inlinks has two justifications behind it that are similar, but distinct concepts.

One of them is merging *graph coverage* presented in Section 5.2 with *word coverage*. This can be achieved, and we present the way it can be done it Section , but we could not find appropriate parameters to use that approach.

Combining multiple submodular functions

The other is seeing inlinks as a citations measure. This has been used somewhat successfully in web ranking and it has since evolved into *PageRank* and other more sophisticated, hybrid approaches, but here we are using it in its plain form viewing Wikipedia different from a (competitive) game. However, using *PageRank* or more advanced measure might prove to be a useful tweak to our current approach.

Revisions

While our goal is to find time-agnostic important articles, and as such we would like to stay away from transitional popular articles, we consider that adding a measure of *popularity* and/or *controversy* into our functions improves our results considerably.

While combining *number of revisions* and *revision volume* - total size of all revisions of an articles - together is validated entirely in an empiric fashion, through experiments, each individual term has a interpretation behind it.

Number of revisions We can view *number of revisions* as a foremost *controversy* measure as articles with high edit count were changed a lot of times (by different persons); this means there existed different users with diverging opinions on the given topic. For example, this can happen in case of a celebrities death.

Revision volume On the other side, we can interpret *revision volume* as a measure of popular interest. Articles with high revision volumes have both a sizeable length and a considerable percentage of big changes made by users passionate about the given topic.

5.4 Beyond influential documents

Dummy text.

5.4.1 Definitions

Dummy text.

5.4.2 Rationale

Dummy text.

Chapter 6

WikiMining framework design

The framework is in an *alpha* development state. While all the described functionality is implemented and works accordingly, you might sometimes find yourself needing to change the code in order to tweak some of the behaviour or that it might not be straight-forward to extend the framework with some specific complex functionality. However, I intend to bring it to a quality standard that will allow you to perform both of the above tasks with ease.

6.1 External libraries

Dummy text.

6.1.1 Java Wikipedia Library

Java Wikipedia Library (JWPL) is an open-source library that facilitates access to a lot of information concerning Wikipedia including, but not limited to:

- page link graph
- category link graph
- page ↔ category link graph
- page content (MediaWiki and plain-text formats)

How it works? Given a Wikipedia dump - that can be downloaded from - it generates 11 text files that can then be imported into a MySQL databased and accessed through the JWPL Application Programming Interface (API). You can find more details about using JWPL on their website .

http://dumps.wikimedia.

https://code.google.com/

The library is publicly available and its authors wrote a paper [?] providing

https://code.google.com/

further insights into its inner workings. For the purposes of this thesis we use *JWPL 0.9.2*.

6.1.2 Hadoop

http://hadoop.apache.org/

Hadoop is an open-source project that aims to offer solutions for 'reliable, scalable, distributed computing' . Although the whole system is more complex we mainly use two of its components:

- MapReduce
- Hadoop Distributed File System

MapReduce

MapReduce (MR) is a two-phase process that borrows two concepts from functional programming:

- map;
- reduce;

and applies them to distributed computing. The main steps are as follows:

1. In the first stage - the map stage - the data is partitioned and spread across multiple processes that perform the same task and output multiple (key, value) pairs.
2. In-between the two phases, the results from the first stage are collected and sorted by key in a distributed fashion.
3. In the second stage - the reduce stage - all pairs that share the same key are sent to the same process and this task outputs a results based on the received pairs (that share the same key).

MapReduce (MR) is a very powerful technique for doing distributed computations because it provides an easy way to write distributed code that is both fault-tolerant and free from other distributed computing problems (such as synchronisation concerns, deadlocks etc.). If you are interested in learning more about MapReduce (MR) we recommend reading the paper [**?**] written by its proponents.

Missing figure

MapReduce stages

Hadoop Distributed File System

Hadoop Distributed File System (HDFS) is, as it name says, a distributed file-system that offers the file-system back-end for writing MapReduce jobs and using other Hadoop-based frameworks. Its purpose is to store large-scale data reliably and distributively such that it can be easy to stream this data at a high bandwidth to applications running on multiple machines [**?**].

For the purposes of this thesis we use *Hadoop 1.0.4*.

6.1.3 Mahout

Mahout is a scalable machine learning library designed to run over very large data sets using Hadoop MapReduce. While it offers distributed algorithms for a lot of machine learning problems such as:

- classification
- clustering
- recommendations

we are only using Mahout for indexing Wikipedia and creating the corresponding tf-idf vectors.

Missing figure

Architecture from http://www.slideshare.net/VaradMeru/introduction-to-mahout-and-machine-learning ?

Generating tf-idf vectors To create tf-idf vectors from a set of plain-text documents [1] is being done using *mahout seq2sparse*. This tool has parameters that allow us to filter out rare words, stop-words and even perform stemming if needed.

Citation needed

https://mahout.apache.org/

Used for XML -¿ plain conversion

If you are interested in finding out more about Mahout you can read the book *Mahout in Action* or visit their website . For the purposes of this thesis we use *Mahout 0.9*.

Cloud9

6.2 System architecture

In this section we present how we designed and implemented the WikiMining library for summarising Wikipedia using submodular function maximisation.

6.2.1 Base data types

We use the following data type classes, defined in package *ch.ethz.las.wikimining.base*:

ArrayWritable Different subclasses for serialiasing an array of different types:

- *IntArrayWritable* - for integers: *IntWritable*;
- *TextArrayWritable* - for strings: *Text*;

or offer more functionality such as print its content in plain text using a *toString()* method.

DocumentWithVector An (Integer document id, Vector) pair used in a lot of MapReduces to partition the data as part of:

- the GreeDi protocol (see Section 4.2 on page 9);
- Locality Sensitive Hashing (see Section).

Preliminaries

It also comes in a *Writable* form so that it can be serialisable by Hadoop MapReduce.

HashBandWritable A (hash, band) pair used for Locality Sensitive Hashing (LSH).

SquaredNearestNeighbour A collection that gets the nearest neighbour of an article (based on cosine similarity) from among the documents that were written before the input article. This is used to compute *document influence*.

[1]Stored in a HDFS-specific format called Sequence Files.

Vector Class from Mahout that represents a *double*-valued mathematical vector. It comes in two flavours:

- *DenseVector* - stored as a normal array
- *RandomAccessSparseVector* - stored as a hash-map of indexes to values.

In addition we have two classes for:

- storing the default parameters - *Defaults*
- defining the program arguments - *Fields*

6.2.2 Submodular functions

Given that the *GreeDi* protocol has only general, high-level restrictions (such as being able to partition the data for example), we can easily implement submodular functions similarly to implementing them in other programming language. Also the *submodular function maximisation* algorithms are implemented such that they can correctly work with any submodular function that extends the *ObjectiveFunction* interface - which is very non-restrictive.

WikiMining provides the following submodular functions package *ch.ethz.las.wikimining.functions*:

CombinerWordCoverage Implements the function described in Section 5.3 on page 14, whose *coverage of a word* function uses *tf-idf*, *number of inlinks*, *number of revisions* and *revision volume*.

CutFunction Given a graph split, it compute the maximum cut value. It is not used with Wikipedia, but instead we used it to test our implementation correctness against the *SFO Matlab toolbox* [**?**].

DocumentInfluence Computes the *influence of a document* as described in Sections , given the *document creation dates*, a *document novelty* index and a *yearly word spread* index.

4.d.Doc Infl, 5.e.Scaling doc infl

GraphCoverage Computes the *graph coverage* described in Section 5.2 on page 14, given the graph as an *adjacency list*.

LSHBuckets Computes the *LSH-buckets* function described in Section , given the LSH buckets.

5.d Lsh buckets

ObjectiveCombiner Linearly combines multiple objective functions as described in Section , given a list of functions and weights.

5.f. Combining multiple subm fn

RevisionsSummer A *modular* function that just sums the number of revisions of each selected document.

WeightedWordCoverage Implements the *word coverage* function where:

- *word importance* is defined as in Section 5.3 on page 14;
- *coverage of a word* uses just the *tf-idf* as described in Section 4.3 on page 10;

WordCoverageFromMahout Implements the same *word coverage* function as above (WeightedWordCoverage), except that it ignores the *word importance*, considered to be a *constant* 1.

All functions that need the *tf-idf* vectors expect them as a *HashMap* from *document ids* to *Mahout Vectors*.

If you are interested in implementing your own submodular functions, you just need to implement the *ObjectiveFunction* interface such that your class provides a *compute* method which returns the *double* score of a set of *document ids*.

6.2.3 Submodular function maximisation

WikiMining provides three types of greedy submodular function maximisation (SFO) algorithms in package *ch.ethz.las.wikimining.sfo*:

SfoGreedyLazy Implements a lazy greedy SFO algorithm that uses a non-stable sorting mechanism, that is a heap, to speed-up the computations as described in section/paper . This is the only maximisation algorithm that we use in all our non-testing code. It has a complexity of $O(k \cdot N$, where k is the average number times the current maximum score is not the real maximum score .

4.b.sf max

Check this complexity

SfoGreedyNonLazy Implements a brute-force (non-lazy) - it tries all possible documents at each iteration - greedy SFO algorithm as described in section/paper .

4.b.sf max

SfoGreedyStableLazy Implements a stable-sort lazy greedy SFO algorithm. The complexity of this algorithm is the same as of the non-lazy version: $O(N)$, so it isn't really useful, but we used it for testing and debugging.

If you need to implement your own submodular function maximisation algorithm you would need to implement the *SfoGreedyAlgorithm* interface such that your class provides the two synonymous *run* methods which take a *list of ids* and the *number of elements to select* and return the *selected document ids*. Instead, if you are interested in using more of our already implemented code and structure you can consider extending the *AbstractSfoGreedy* abstract class that provides you with *score, document id* pair.

6.2.4 Coverage MapReduces

This part refers to the three categories of classes that deal with coverage, found in package *ch.ethz.las.wikimining.mr.coverage.h104*:

1. preprocessing classes;
2. GreeDi protocol classes;
3. reducer classes.

The preprocessing classes are as follows:

GraphEdgesConvertor Converts a plain-text *list of edges* to an *adjacency list* saved in a sequence file.

GraphInlinksCounter Given the inverted adjacency lists it computes the number of inlinks of each node.

RevisionsConvertor Converts a list of *revision statistics* (number of revisions, revision volume from plain-text to sequence files.

The two stages of the *GreeDi* protocol are implemented as classes *GreeDiFirst* and *GreeDiSecond*. These two classes are very similar in implementation and they deal with:

- parsing the program arguments;
- reading the main input files - the tf-idf vectors;
- setting up the desired reducer.

They also include the *mapper* implementations which are simple as their only job is to partition the data.

While the *mappers* for the GreeDi protocol are simple and are almost always the same, independent of the used submodular functions being used, the *reducers* have to deal with reading all the additional data for the more complex submodular functions (for example, the graph, the revision statistics). We implemented the following reducers:

CombinerGreeDiReducer Read multiple statistics - such as word count, LSH-buckets, graph, revisions - and uses submodular functions that combines them together [2];

CoverageGreeDiReducer Reads as less as possible to apply either the simple word coverage with a *word importance* of 1 or with *word importance* as word count or document count [3];

GraphGreeDiReducer Reads the graph and applies *graph coverage* by itself [4];

LshBucketsGreeDiReducer Reads the LSH buckets and applies the *LSH buckets function*. [5] .

Add to footnote 5.d. LSH buckets

[2]See Section 5.3 on page 14
[3]See Section 5.3 on page 14
[4]See Section 5.2 on page 14
[5]See Section

6.2.5 Influence MapReduces

Dummy text.

6.2.6 Selection and evaluation

Dummy text.

6.2.7 Input, output

Dummy text.

Chapter 7

Experiments

In this chapter we present our results and compare them to existing approaches.

7.1 Datasets and metrics

Dummy text.

7.1.1 Datasets

To get the Wikipedia pages, we use the dump from *01 October 2012* that has only the latest revision for each article . The dump version we use should not impact the results considerably. We happen to use the 2012 version just because was easily available at the time. For this Wikipedia version we extract articles corresponding to a category (or multiple categories recursively) recursively – we parse the category graph downwards starting from the given *root* category. After extracting the articles we keep only non-stub, human-written articles and we exclude all meta pages. For more information about the extraction process see .

Footnote for where to get it

coverage.WikiToPlainText section

We tested our approach on the following recursively extracted categories:

Classical composers Contains 7246 articles representing all classical music composers – this is our biggest category apart from using all Wikipedia;

Game artificial intelligence Contains 471 articles about the use of artificial intelligence (AI) in games (for example, chess, go, video games);

Machine learning Contains 735 articles about the machine learning field;

Vectors Contains 341 articles about mathematical vectors – this our most abstract category;

and merges of them:

Last 3 Contains the 1547 articles of the last three categories described above (Game AI, Machine learning, Vectors);

All 4 Contains the 8793 articles of all above categories (Composers, Game AI, Machine learning, Vectors).

double check this number

For the final results we used the **2012 Wikipedia** which has a total of 1.3 million non-stub, human-written articles (excluding meta pages).

In addition to the Wikipedia subset, we use the *neural Information Processing Systems (NIPS)* dataset – contains 1955 papers – and the *Association for Computational Linguistics (ACL)* dataset – contains 18041 papers – to test our baseline implementations against their authors [**?**].

7.1.2 Metrics

We have looked to find metrics suitable to our task of finding the most important, valuable Wikipedia articles, but we could not find a reliable method that also fits well to our use case. Some of the possible metrics are:

citation needed

Number of inlinks This metric is similar to the use of *number of citations* presented in [**?**], but extended to web pages. While it does not have the same limitations as citations, it is hard to guarantee its validity .

citation needed

PageRank A common way to extend the use of just inlinks which solves a lot of the problems with inlinks. .

citation needed

Condensed Wikipedia Count the number of selected documents that appear in a condensed Wikipedia version used in education or in emph-Wikispeedia .

For the purposes of this thesis we use *number of inlinks* as we view Wikipedia as a non-competitive environment (different from the game between search engines and malevolent web masters).

7.2 Baselines

Dummy text.

7.2.1 Random

Dummy text.

7.2.2 Word coverage

Dummy text.

7.2.3 Document influence

Dummy text.

7.3 Graph coverage

Dummy text.

7.4 Beyond word coverage

Dummy text.

7.5 Scaling document influence

Dummy text.

Chapter 8

Conclusion

Dummy text.

8.1 Future work

Dummy text.

Appendix A

Dummy Appendix

You can defer lengthy calculations that would otherwise only interrupt the flow of your thesis to an appendix.

Bibliography

[1] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[2] Andreas Krause. Sfo: A toolbox for submodular function optimization. *The Journal of Machine Learning Research*, 11:1141–1144, 2010.

[3] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 3, 2012.

[4] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems*, pages 2049–2057, 2013.

[5] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

[6] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[7] Ruben Sipos, Adith Swaminathan, Pannaga Shivaswamy, and Thorsten Joachims. Temporal corpus summarization using submodular word coverage. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 754–763. ACM, 2012.

[8] Torsten Zesch, Christof Müller, and Iryna Gurevych. Extracting lexical semantic knowledge from wikipedia and wiktionary. In *Proceedings of the 6th International Conference on Language Resources and Evaluation*, Marrakech, Morocco, May 2008. electronic proceedings.