Bericht zum Praktikum
„Named Entity Recognition mit Wikipedia (Orte)“

Teilnehmer:
- Kevin Gomez
- Moritz Engelmann
- Yves Annanias
- Stefan Faulhaber

Aufgabenstellung:

Im Rahmen des Text-Mining Praktikums hat unsere Gruppe die Aufgabe „Named Entity Recognition mit Wikipedia (Orte)“ bearbeitet. Dazu sollten wir zuerst Ortsbeschreibungen auf Wikipedia identifizieren, um dann die Ortsnamen mit Hilfe der Überschriften zu annotieren. Außerdem sollten wir alle Sätze extrahieren, die Ortsnamen enthalten um einen Klassifikator mit diesen annotierten Daten zu trainieren. Der trainierte Klassifikator sollte dann automatisch alle Wikipedia Texte klassifizieren. Mit diesen gesamten klassifizierten Wikipedia Daten sollte dann letztendlich ein neuer, besserer Klassifikator erzeugt werden.

Methodik & Vorgehen:

Für die Ermittlung der Orte ist es natürlich viel zu aufwendig, sämtliche Wikipedia Seiten online nach dem Normdaten-Tag „Geografikum“ zu durchsuchen. Jedoch bietet Wikipedia eine xml-Datei mit allen Artikeln an. Wir haben einen Parser geschrieben, der sämtliche Artikel mit dem Normdaten-Tag „Geografikum“ in einer neuen xml-Datei speichert und alle Titel dieser Artikel in eine andere. Damit haben wir nun eine komplette Liste aller Orte, die wir nutzen können um alle Orte in den einzelnen Artikeln zu markieren. Als nächstes haben wir ein Programm geschrieben, mit dem wir diese Markierungen durchführen können. Einfache String-Vergleiche sind hier aber nicht sehr effektiv, da einzelne Orte in verschiedener Form vorkommen können, z.B. „Leipzig” und „Leipziger Land“. Daher bietet es sich an die Editierdistanz zwischen zwei Wörtern zu bestimmen, um solche Abweichungen zu erkennen. Für die Bestimmung der Editierdistanz gibt es für Java eine Bibliothek namens „Symmetrics“. Sie bietet verschiedene Algorithmen (z.B. Levenshtein, CosineSimilarity, EuclideanDistance oder MongeElkan) für die Bestimmung der Änlichkeit von zwei Wörtern an. Wir haben alle Algorithmen mit verschiedenen Daten getestet. Dabei brachte Levenshtein die besten Ergebnisse. Als nächstes galt es herauszufinden, welche Distanz am besten geeignet ist, um möglichst viele falsche Wörter zu überspringen, aber auch viele Richtige zu markieren. Auch hier haben wir wieder mit mehreren Daten getestet und am Ende 0.8333 als optimalen Wert ermittelt.
Da die kompletten Daten sehr groß sind dauert es sehr lange, alle Orte in allen Artikeln zu markieren. Daher haben wir uns überlegt, dass eine Parallelisierung sinnvoll wäre.

Dafür haben wir Hadoop, eine Java-Implementierung des MapReduce-Frameworks, verwendet. MapReduce teilt die xml-Datei mit den Artikeln nach Orten auf. Das eingegebene Key-Value-Paar besteht aus dem Namen des Ortes und dem Artikel zu diesem Ort. Die Map-Funktion überprüft nun, ob Orte aus der Liste aller Orte in dem Artikel vorkommen und markiert diese mit den Highlight-Tags. Am Ende wird ein Key-Value-Paar mit dem Namen des Ortes und dem markierten Text ausgegeben. Die Reduce-Funktion ist dafür zuständig, eine xml-Datei mit allen markierten Artikeln zu erstellen.
Nachdem diese xml-Datei erstellt wurde, liegt ein großer Datensatz vor, indem alle Vorkommen und auch Abweichungen von Ortschaften (so gut wie möglich) markiert wurden. Nun können alle Sätze extrahiert werden, in denen ein Wort gehighlightet wurde, also alle Sätze in denen ein Ort vorkommt. Diese Sätze dienen dann als Trainingsdaten für den Klassifikator (Stanford NER). Nachdem der Klassifikator trainiert wurde, kann er dazu genutzt werden, um alle Texte in der Wikipedia Sammlung zu klassifizieren. Dieses bessere Ergebnis kann wiederum genutzt werden, um einen neuen Klassifikator zu trainieren.

Algorithmen:

Die Methode *readPlacesCompareAndWrite* erfüllt grundsätzlich folgendes: Sie liest die einzelnen Wörter des Korpus ein, führt einige Vorbearbeitungen durch, vergleicht dann die Wörter mit allen Titeln und markiert gefundene Treffer. Die einzelnen Schritte werden nun im Detail betrachtet und erläutert.

Vorverarbeitung der Wörter aus dem Korpus

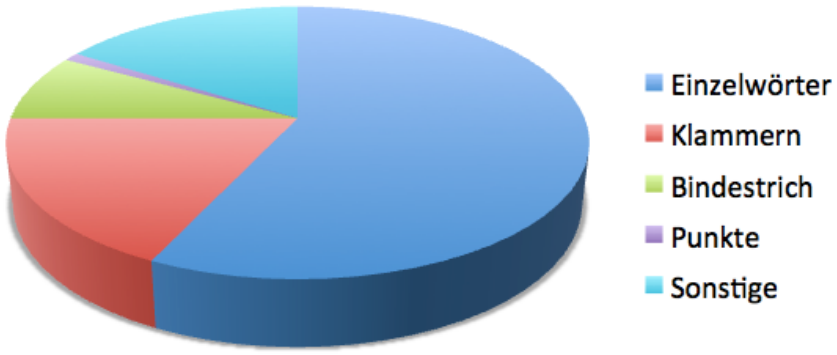
Bei der Durchsicht der Daten hat sich ergeben, dass an vielen Tokens noch bestimmte Sonderzeichen anhängen. So tritt zum Beispiel häufig ein ‚/' auf, gefolgt von einer Zeichenkette (z.B. „Halle/Saale"). Dies kommt natürlich so nicht als ein Titel vor, daher wird aus allen Tokens das ‚/' zusammen mit dem nachfolgendem Rest abgeschnitten. Es gibt noch weitere Sonderzeichen, die entfernt werden müssen. Dies geschieht durch *replaceAll("[\\Q][{}()\"“',.;!?<>%\\E]", "")*. Wenn die Vorverarbeitung durchgeführt wurde, kann mit dem Vergleichen begonnen werden.

Vergleich der Tokens mit den Titeln

Da nicht alle Titel aus nur einem Wort bestehen, ist es notwendig bei mehreren Wörtern auch mehrere Wörter aus dem Korpus zu vergleichen. Dazu wird erst ermittelt, aus wie vielen Wörtern der Titel besteht. Besteht ein Titel aus mehren Wörtern, wird aus dem Korpus ein String mit ebenso vielen Wörtern gebaut. Nun wird dieser String solange mit allen Titeln verglichen, bis eine Übereinstimmung gefunden wurde. Dann wird der entsprechende String im Korpus markiert und der Vergleich beginnt von vorn mit dem nächsten String (bei *n* Wörtern im Titel eben *n* Wörter später im Korpus).

Vergleich und Abstandsmaß

Zum Vergleich der Wörter wird der Levenshtein-Algorithmus aus der Simmetrics Bibliothek verwendet, da mit diesem in verschiedenen Tests die besten Ergebnisse erzielt werden konnten. Aus weiteren Tests ergab sich, dass der Vergleichswert 0.8333 das beste Ergebnis liefert. Mit ihm werden fast nur noch korrekte Übereinstimmungen geliefert und die meisten falschen Übereinstimmung fallen weg. Auf diese Weise werden alle Titel die nur aus einem Wort bestehen gefunden, was schon einen Großteil ausmacht (siehe Diagramm).



Wieso MapReduce?

Wir testen jedes Wort eines Wikipedia-Artikels mit jedem Wort aus dem Titel-Korpus. Dies ist im Grunde das Aufbauen einer sehr großen m x n – Matrix. Dieses große Datenaufkommen lässt sich gut mit MapReduce bewältigen. Die vorangestellte Überlegung und Implementierung wurde deshalb in einen MapReduce-Job übernommen und ausgeführt. Um den MapReduce-Job so schnell wie möglich ausführen zu können, wurden zur Beschleunigung der Berechnung noch verschiedene Techniken verwendet, welche im Folgenden erklärt sind.

1. Großbuchstaben:

Im Titel-Korpus befinden sich nur Eigennamen, Ortsnamen und Bezeichnungen. Deshalb ist es unnötig, kleingeschriebene Wörter aus dem Text-Korpus zu überprüfen.

2. Stopp-Wort-Liste:

Die 15 am häufigsten im Deutschen verwendeten Wörter werden am Anfang mit jedem Wort verglichen, sodass bei einer Übereinstimmung direkt mit dem nächsten Wort fortgefahren werden kann.

3. Blocking:

Blocking ist der Versuch, die zu erstellende Matrix zu verkleinern. Unser Titel-Korpus ist eine alphabetisch sortierte Liste. Wir wollen nur Wörter aus dem Text-Korpus mit Titeln aus dem Titel-Korpus vergleichen, wenn der erste Buchstabe übereinstimmt. Nach mehreren Durchläufen wissen wir den Start-Index von vielen Anfangsbuchstaben, sodass nicht alle Titel verglichen werden müssen.

Die Erzeugung der Trainingsdaten geschieht, nachdem alle Sätze extrahiert wurden, die eine Ortschaft enthalten. In diesen Sätzen werden nun alle Sonderzeichen sowie das Highlight-Tag entfernt, dabei wird das Wort im Highlight-Tag durch *LOCATION* ergänzt, alle anderen Wörter mit einem *O*. So entsteht eine Ausgabedatei, in der in der ersten Spalte ein Wort steht, gefolgt von einem Tabulator und dem Symbol *O* oder *LOCATION*. Einzelne Sätze werden durch eine Leerzeile getrennt.

Ergebnisinterpretation:

Grundsätzlich kann man sagen, dass unsere Gruppe die Aufgabe erfolgreich bearbeitet hat. Trotzdem gibt es noch einige Probleme, die beseitigt werden könnten. Bei der Annotation der Orts-Artikel ist der Umgang mit Sonderzeichen zur Zeit noch nicht hinreichend implementiert. Titel, die aus mehreren Wörtern bestehen werden ebenfalls noch nicht richtig markiert. Die Herausforderung bei der Implementation dieser Anpassungen liegt im Gleichgewicht aus der Korrektheit der Markierungen und der Erfassung aller Vorkommen eines Titels. Dieses Gleichgewicht ist immer ein Kompromiss, der die Implementation erschwert. Deswegen werden zur Zeit auch einige Wörter markiert, die nicht als Titel vorhanden sind (z.B. „Stadt“).

Durch die Lösung der Probleme und Erweiterung des Projekts, kann die Effizienz und Nützlichkeit noch weiter verbessert werden. Die Klassifikatoren konnten von uns nicht mit allen Trainingsdaten trainiert werden, da unsere Ressourcen beschränkt sind. Dies kann aber auf einem leistungsstarken Rechner in Zukunft problemlos nachgeholt werden, was die Klassifikatoren nochmals verbessert. Weiterhin könnten die Trainingsdaten (und somit auch die Klassifikatoren) verbessert werden, indem die Wörter der Sätze in den Trainingsdaten in der richtigen Reihenfolge angeordnet werden. Aktuell sind zwar alle Sätze in den Trainingsdaten, die Wörterreihenfolge ist jedoch nicht immer korrekt.