Philipps Universität Marburg

Philipps-Universität Marburg
Fachbereich Mathematik und Informatik

Prof. Gabriele Taentzer
Stefan John

Programmier-Praktikum

Automatische Analyse und Anreicherung von Chatnachrichten

In den folgenden zwei Tagen sollen Sie Ihre Chat-Applikation dahingehend erweitern, dass diese in der Lage ist, empfangene Nachrichten zu analysieren und mit zusätzlichen Informationen anzureichern. Insbesondere werden Sie sich dabei mit den folgenden Analysemethoden befassen:

Sentiment-Analysis Hierbei werden Texte daraufhin untersucht, ob sie eine positive oder negative Haltung des Autors vermitteln. Diese Informationen sollen Sie nutzen, um Nachrichten automatisiert mit passenden Emojis zu versehen.

Named-entity-Recognition Dient im wesentlichen der Erkennung von Eigennamen. In Nachrichten auftretende Eigennamen sollen um eine URL ergänzt werden, welche zu einer passenden Wikipedia-Seite führt.

1 HTTP-Anfragen in Java

Die oben genannten Analysemethoden werden durch Webservices bereitgestellt, welche durch HTTP-Anfragen angesprochen werden müssen. Im Folgenden sollen Sie sich daher damit befassen, wie das Absenden solcher Anfragen mit Java-Bordmitteln (also ohne Nutzung einer Bibliothek) realisiert werden kann. Im Web lassen sich zu diesem Thema relativ detaillierte Tutorials finden, an denen Sie sich orientieren können.

Schreiben Sie eine Klasse *HttpConnector*, mit der sowohl GET- als auch POST-Anfragen an eine gegebene URL verschickt werden können. Zu diesem Zweck soll die Klasse eine Methode *sendRequest* bereitstellen, welche zum Versenden von Anfragen genutzt werden kann und die jeweilige Antwort als *String* zurückgibt. Folgende Bedingungen sollen dabei berücksichtigt werden:

- Ein HttpConnector soll an *eine* URL gebunden sein, die bis auf Query-Parameter nicht mehr verändert werden kann. Für Anfragen zu unterschiedlichen Webservices werden daher mehrere HttpConnector-Instanzen benötigt.
- Die Verwendung URL-codierter Anfrageparameter (Query-Strings) soll für beide Anfragetypen (GET und POST) unterstützt werden. Die Query-Parameter sollen der sendRequest-Methode dabei als *Map<String,String>* übergeben werden.

- Bei POST-Anfragen soll es möglich sein, Nutzdaten in Form eines *Strings* als Inhalt der Anfrage zu verschicken. Auf andere Inhaltstypen (MIME-Type) muss hier nicht eingegangen werden. Das Spezifizieren eines Nachrichteninhalts bei GET-Anfragen soll als Fehler behandelt werden.
- Um eine Fehlerbehandlung durch die Applikation zu ermöglichen, muss bei gescheiterten Anfragen (HTTP-Statuscode $\neq$ 200, Auftreten einer Exception) ein entsprechender Fehler propagiert werden.
- HTTPS-Anfragen müssen bei Ihrer Implementierung *nicht* berücksichtigt werden.

Hinweis: Sie können Anfragen an *http://httpbin.org/get* bzw. *http://httpbin.org/post* schicken, um Ihre Implementierung zu testen. In beiden Fällen werden, sofern die Anfrage erfolgreich war, die Parameter der Anfrage als Antwort zurückgegeben.

2 Klassenstruktur

Wie bereits in der Einleitung beschrieben, sollen Sie zunächst nur zwei unterschiedliche Analysemethoden für Chat-Nachrichten implementieren. Es sind jedoch eine ganze Reihe weiterer Analyseverfahren denkbar, die in einer Chat-Anwendung zum Einsatz kommen könnten (z.B. Extraktion des aktuellen Themas einer Konversation, Grammatik und Rechtschreibprüfung, etc.). Daher ist es sinnvoll, die Struktur Ihrer Applikation so zu gestalten, dass diese leicht um zusätzliche Analysen erweitert werden kann.

Erstellen Sie eine Klassenstruktur welche folgende Elemente umfasst:

- Ein Interface *MessageAnalyzer* welches von allen Analyseverfahren implementiert werden muss. Das Interface fordert eine Methode *analyzeMessage*, welche als Parameter lediglich eine Nachricht als *String* entgegen nimmt. Zurückgeben soll die Methode Objekte vom Typ *AnnotatedMessage* (siehe unten).
- Eine Klasse *AnnotatedMessage*. Diese soll, neben der Originalnachricht, das Ergebnis einer Analyse in Form einer Liste von Annotationen beinhalten. Die zu speichernden Daten einer Annotation hängen stark vom jeweiligen Analyseverfahren ab (z.B. Sentiment der Sätze, Named Enities einer Nachricht, etc.). Verwenden Sie daher *Generics*, statt einer gemeinsamen Oberklasse für die verschiedenen Annotationsarten, um AnnotatedMessages mit unterschiedlichen Annotationstypen zu realisieren.

- Klassen zur Modellierung der analysespezifischen Annotationen. Zunächst ohne Klassenkörper. Die Ausimplementierung erfolgt später.

3 Nachrichtenanalyse

Für die im Folgenden benötigten Analysemethoden kommt die Stanford CoreNLP Bibliothek zum Einsatz. Auf einer Demo-Webseite können Sie sich zunächst ein wenig mit den unterschiedlichen Analysen vertraut machen.

Unter *http://dbsvm.mathematik.uni-marburg.de:7999* steht für die Analysen ein Webservice bereit, der POST-Anfragen entgegen nimmt. Der zu analysierende Text muss diesem als Inhalt der Anfrage (in Form eines Strings) übermittelt werden. Die Konfiguration des Webservice erfolgt über einen Query-Parameter *properties*. Dieser erwartet als Wert ein *JSON-Objekt*, welches die einzelnen Konfigurations-Optionen als Schlüssel-Wert-Paare spezifiziert. Zwei Optionen sind für Sie relevant:

- Die Option *annotators* legt fest, welches Analyseverfahren verwendet werden soll. Gültige Werte sind *sentiment* und *ner*.
- Mit der Option *outputFormat* wird das Format der Antwort eingestellt. Verwenden Sie hier *json*.

Beispiel Query-String:

```
?properties={"annotators":"sentiment","outputFormat":"json"}
```

Machen Sie sich zunächst mit dem JSON-Format vertraut. Um mit diesem Format in Java einfacher arbeiten zu können, laden Sie sich die Bibliothek JSON in Java herunter und fügen Sie diese zum *Build Path* Ihres Projekts hinzu. Dokumentation zu dieser finden Sie auf der offiziellen Github-Seite sowie in der API. Nutzen Sie diese Bibliothek, um JSON-Objekte zu erstellen und die JSON-Antworten des Webservice zu parsen.

Implementieren Sie die *Sentiment-Analysis* und *Named-entity-Recognition*. Beide Analyseverfahren sollen das oben genannte *MessageAnalyzer*-Interface implementieren. Nutzen Sie *Streams* und *Lambdas* zum Parsen der Antworten des Webservice. Lesen Sie an dieser Stelle bereits die Aufgaben *Emoji-Adder* und *Entity-Linker*, um abschätzen zu können, welche Informationen Ihre Annotationen enthalten sollten.

Hinweis: Mit einem JSON-Viewer können Sie sich schnell einen Überblick über die Struktur eines JSON-formatierten Textes verschaffen.

4 Emoji-Adder

Implementieren Sie ein Modul, welches die *Sentiment Analysis* nutzt, um die Sätze eines gegebenen *Strings* mit passenden Emojis zu versehen. Die Nutzung von Emojis wird durch die eingestellte Schriftart der gegebenen GUI-Komponenten bereits unterstützt. Emojis werden hierbei nicht durch Bilder sondern durch Unicode-Zeichen realisiert und können damit von Ihrem Modul durch einfaches Anpassen des gegebenen Strings eingefügt werden. Die Erkennung von Sätzen innerhalb eines Strings können Sie CoreNLP überlassen. Die Antwort des Webservice enthält eine entsprechende Unterteilung.

Um die Arbeit mit Emojis in Java zu erleichtern, fügen Sie die Bibliothek Emoji-Java zum *Build Path* Ihres Projekts hinzu. Durch diese kann jedes Emoji über einen Alias angesprochen und in ein Unicode-Zeichen überführt werden. Herunterladen können Sie die entsprechende jar im Abschnitt Releases der Github-Seite.

CoreNLP unterstützt 5 Sentiment-Stufen. Neutrale Sätzen sollen kein Emoji erhalten. Für die anderen Stufen können Sie folgende Emoji-Alias verwenden:

- *sehr negativ:* disappointed
- *negativ:* unamused
- *positiv:* wink
- *sehr positiv:* grin

Zurückgeben soll Ihr Modul den String der Nachricht, der um entsprechende Emojis als Unicode-Zeichen erweitert wurde.

5 Entity-Linker

Implementieren Sie ein Modul, welches die *Named-entity Recognition* nutzt, um hinter alle Eigennamen einer Nachricht die URL einer passenden Wikipedia-Seite anzufügen. Die URL soll sich dabei aus *https://en.wikipedia.org/wiki/* und dem Eigennamen zusammensetzen und in eckige Klammern eingeschlossen sein.

Beispiel:

NVIDIA [https://en.wikipedia.org/wiki/NVIDIA], stop stealing my money!

Achten Sie darauf, dass Eigennamen beliebige Zeichen enthalten können und auch hier ein URL-Encoding notwendig ist. Zudem ist für den Pfad einer URL ein reines Prozent-Encoding nötig, welches sich unter Umständen von dem Encoding unterscheidet, welches Sie für Query-Strings verwendet haben.

Zurückgeben soll Ihr Modul den um die URLs erweiterten String der Nachricht.

Hinweis: Die tatsächliche Existenz eines passenden Wiki-Eintrags müssen Sie nicht überprüfen.

6 Integration in den Chat

Erster Test

Integrieren Sie Ihren Emoji-Adder und Entity-Linker zunächst prototypisch so in die Chat-Anwendung, dass diese jede eingehende Nachrichten automatisch erweitern. Betrachten Sie dazu die Klasse *ChannelMessageListCell* im Paket *gui.chat.renderer*, welche das Erscheinungsbild der Nachrichten in einem Channel festlegt.

Vollständige Integration

Erweitern Sie das Hauptmenü der Anwendung um einen Menüpunkt *Enhancers*. Dieser soll zwei Einträge (*CheckMenuItem*) enthalten, über die Ihre beiden Module zur Erweiterung von Nachrichten aktiviert bzw. deaktiviert werden können.

Um das Hauptmenü zu erweitern, muss die Datei *chat.fxml* im package *gui.chat* angepasst werden. Sie können sich dazu an den bestehenden Menüpunkten und einem Tutorial orientieren.

Implementieren Sie in der Klasse *ChatController* im Paket *gui.chat* eine Methode

```
public void toggleEnhancer(ActionEvent e){ ... }
```

Diese soll die *Events* beider *CheckMenuItem*s verarbeiten und die Module entsprechend an- und ausschalten. Um Ihre Module über diese Methode gemeinsam ansprechen zu können, bietet sich die Nutzung eines Interfaces an. Sofern dies nicht ohnehin schon geschehen ist, finden und extrahieren Sie für die Module ein *Interface Enhancer*.

7 Langeweile? (optional)

Sollten Sie alle oben genannten Aufgaben erfolgreich bewältigt haben, können Sie sich an folgende Probleme wagen.

7.1 Verbesserung des Entity-Linkers

Wird nur die Named-entity Recognition verwendet, um Links zu Eigennamen hinzuzufügen, fällt auf, dass bspw. auch Personalpronomen als „Eigennamen“ erkannt werden. Filtern Sie Personalpronomen aus den zu verlinkenden Worten heraus. Hierzu können Sie die Tatsache nutzen, dass der Webservice für NER implizit auch eine *Part-Of-Speech*-Analyse durchführt. Im JSON-Objekt *tokens* der Antwort des Webservice können Sie auf die Ergebnisse dieser Analyse zugreifen.

7.2 Erweiterung der Nachrichtenanzeige

Im gegebenen Interface des Chats sind Nachrichten eines Kanals in *TextField*-Komponenten gekapselt. In diesen ist es nicht möglich Bilder einzufügen oder einzelne Textabschnitte unterschiedlich zu formatieren. Auch HTML wird von diesen nicht unterstützt. Erweitern Sie die Anwendung (z.B. unter Verwendung von *WebView*-Komponenten) so, dass Emojis als farbige Bilder integriert und Eigennamen direkt mit HTML-Links versehen werden können.

7.3 HTTPS-Verbindungen

Erweitern Sie die Applikation so, dass der von Ihnen implementierte *HttpConnector* auch HTTPS-Anfragen unterstützt.

7.4 Tooltips

Implementieren und integrieren Sie einen *Enhancer*, der einen Tooltip mit kurzer Beschreibung öffnet, sobald der Mauszeiger über einen Eigennamen geführt wird. Zur Beschaffung der notwendigen Tooltip-Informationen können Sie bspw. die API von Wikipedia verwenden. Beachten Sie, dass hierzu HTTPS-Anfragen benötigt werden.