Projektbericht Essay Scoring

Filter für nichtsprachliche Inhalte

Artur Begichev, Kubilay Kán Kiraz

Gliederung

Einleitung

Automatisches Essay-Scoring bezieht sich auf den Prozess, bei dem maschinelle Lernmodelle oder Algorithmen eingesetzt werden, um Essays oder schriftliche Texte automatisch zu bewerten. Dieser Ansatz wird häufig in Bildungsumgebungen, insbesondere in großen Klassen oder standardisierten Tests, angewendet, um die Bewertungseffizienz zu steigern und objektive Bewertungen zu gewährleisten. Indem bestimmte Kriterien wie Grammatik, Rechtschreibung, Kohärenz, Argumentation oder in diesem Fall auch nichtsprachliche Inhalte bewertet werden, können automatische Essay-Scoring-Systeme dazu beitragen, den Bewertungsprozess zu standardisieren und die Konsistenz zwischen verschiedenen Bewertern zu verbessern.

In diesem Bericht beschäftigen wir uns mit dem Projekt des Filters für nichtsprachliche Inhalte und evaluieren seine Ziele und Erfolge, aber auch seine Schwächen und Probleme in der Implementierung. Ebenso wird der Anwendungskontext erläutert und dieser in Zusammenhang mit dem verwendeten Korpus gesetzt. Der Korpus ist eine Essay-Sammlung von Schülern.

Außerdem werden die Prozesse und Funktionsweisen des dem Filter zugrunde liegenden Codes untersucht, vorgestellt und erläutert.

Der Filter als Teilaufgabe des AES

Der Grundgedanke des Filters für nichtsprachliche Inhalte im Kontext des automatischen Essay-Scorings, liegt in der Säuberung von Korpora in Form von Essays und zwar so, dass bestimmte Elemente des Essays identifiziert und bewertet werden, die nicht direkt mit der sprachlichen Qualität des Textes zusammenhängen.

Nichtsprachliche Inhalte und nicht textbezogene Aspekte können die Qualität eines Essays beeinträchtigen oder ihn weniger verständlich machen. Ein Filter kann helfen, solche Probleme zu erkennen und zu bewerten, um die Gesamtbewertung des Essays genauer zu gestalten. Damit kann der Bewertungsprozess objektiver gestaltet werden, indem nicht nur die sprachliche Qualität des Textes, sondern auch andere wichtige Aspekte berücksichtigt werden.

Außerdem spart ein effizient programmierter Filter Zeit- und Aufwandressourcen, indem er automatisch bestimmte nichtsprachliche Probleme identifiziert, die normalerweise viel Zeit in Anspruch nehmen würden, wenn sie manuell überprüft werden müssten.

Der Grundgedanke hinter einem Filter für nichtsprachliche Inhalte besteht also darin, sicherzustellen, dass alle relevanten Aspekte eines Essays berücksichtigt werden, um eine genaue und umfassende Bewertung zu ermöglichen und den Gesamtwert des automatischen Essay-Scorings zu verbessern.

Implementierung

Zuallererst widmeten wir uns der ersten Teilaufgabe der Emoji-Filterung und stießen auf ein Modul in Python namens ‘Emoji‘.

Dieses enthielt die Funktion ‘demojize‘, welche Emojis erkennt und sie in ihre entsprechende Textform umwandelt.

Mit der Hilfe von ‘demojize‘ entwarfen wir die Funktion

‘def remove_emojis_from_pdf(pdf_path):’,

welche die PDF-Datei des Korpus öffnet, über ihre Seiten iteriert, die Emojis in Text umwandelt und diese dann markiert.

Ein Beispiel sähe so aus: „<emoji> face_with_rolling_eyes </emoji>“.

Nachdem wir den ersten Schritt bewältigt hatten, erhielten wir die Empfehlung für ein Rechtschreiberkennungsprogramm namens ‘Hunspell‘ für unsere weiteren Ziele.

Wir versuchten dieses Programm in unseren Python Code zu integrieren bzw. zu nutzen, doch stießen auf Inkompatibilität. Hunspell erforderte für die Verwendung und Nutzung unter Python eine Installation von ‘VisualClearStudio++‘ und die korrekte Einrichtung dieses Programms. Doch ergab sich die gewünschte Synergie zwischen Python, Hunspell und VisualClearStudio++ als äußerst schwierig zu realisieren. Wir investierten sehr viel Zeit in die Implementierung von Hunspell, bis wir uns entschieden nach Alternativen zu suchen.

Unser erstes „erfolgreiches“ Rechtschreibprüfungsmodul unter Python war tatsächlich ‘Spellchecker‘, unter dem wir dann auch unseren Code programmieren konnten.

Wir schrieben unter Spellchecker folgende Funktionen:

preprocess_text(text): Diese Funktion entfernt überflüssige Leerzeichen und Zeilenumbrüche aus dem Text, um eine saubere Verarbeitung zu gewährleisten.

mark_repeated_punctuation(text): Diese Funktion markiert wiederholte Satzzeichen wie "!!!" oder "???" im Text, indem sie sie in Tags einschließt.

is_potential_noun(word, index, tokens): Diese Funktion prüft, ob ein Wort ein potenzielles Nomen sein könnte, basierend darauf, ob es großgeschrieben ist und seine Position im Satz.

check_spelling(word, spell): Diese Funktion überprüft die Rechtschreibung eines Wortes mithilfe eines Spellchecker-Objekts und einem Cache für bereits überprüfte Wörter.

mark_misspellings(text, spell): Diese Funktion markiert Rechtschreibfehler im Text, indem sie fehlerhafte Wörter in Tags einschließt.

remove_emojis_and_mark_misspellings(pdf_path): Diese Funktion verarbeitet den Text einer PDF-Datei, entfernt Emojis und markiert Rechtschreibfehler.

Doch Spellchecker zeigte gravierende Probleme, auf die wir weiter unten eingehen werden, weswegen wir uns wieder nach Alternativen umschauen mussten.

Schlussendlich stießen wir auf eine Hunspellvariante, welche kompatibel mit python war, genannt ‘phunspell‘. Es erforderte so gut wie keine Änderungen im Code, außer der Änderungen einiger Parameter, wie folgende(Änderungen unterstrichen):

check_spelling(word, phun): Diese Funktion überprüft die Rechtschreibung eines Wortes mithilfe eines Hunspell-Objekts und einem Cache für bereits überprüfte Wörter.

mark_misspellings(text, phun): Diese Funktion markiert Rechtschreibfehler im Text, indem sie fehlerhafte Wörter in Tags einschließt.

Probleme

Zuerst kam die Hürde von Hunspell, welches die Verwendung und Nutzung von ‘VisualClearStudio++‘ erforderte.

Doch Spellchecker bereitete am meisten Probleme, da dieses Programm unausgereift ist und auf einer sehr lückenhaften Datenbank basiert. Die meisten Nomen im Korpus wurden fälschlicherweise als Rechtschreibfehler markiert, obwohl sie sowohl richtig geschrieben als auch Teil des offiziellen, deutschen Wortschatzes waren. Außerdem wurden Nicht-Nomen am Satzanfang als Fehler markiert, nur weil sie großgeschrieben wurden. Dafür wurde auch die Funktion ‘is_potential_noun(word, index, tokens):‘ implementiert.

Verwendete Literatur und Module

Informationen zur Verwendung von den externen Bibliotheken wie emoji, phunspell und fitz konnten wir online in den offiziellen Dokumentationen dieser Bibliotheken finden.

re: Das integrierte Python-Modul re wird verwendet, um reguläre Ausdrücke für die Textverarbeitung zu verwenden.

emoji: Die Bibliothek emoji wird verwendet, um Emojis im Text zu verarbeiten.

phunspell: Die Bibliothek phunspell wird verwendet, um Rechtschreibprüfungen durchzuführen.

fitz: Die Bibliothek fitz (PyMuPDF) wird verwendet, um PDF-Dokumente zu öffnen und zu verarbeiten.

os: Das integrierte Python-Modul os wird verwendet, um mit dem Dateisystem zu interagieren.

gc: Das integrierte Python-Modul gc wird verwendet, um den Garbage Collector aufzurufen und Speicher freizugeben.

Fazit

Insgesamt ermöglicht der Code eine effiziente Verarbeitung von Texten in PDF-Dokumenten mit dem Ziel, nichtsprachliche Inhalte zu filtern und Rechtschreibfehler zu markieren. Durch die Kombination verschiedener Funktionen und Bibliotheken werden mehrere Aspekte der Textverarbeitung abgedeckt, angefangen bei der Entfernung überflüssiger Leerzeichen und Zeilenumbrüche bis hin zur Identifizierung und Markierung von wiederholten Satzzeichen, Emojis, nicht als Wörter identifizierbare Sequenzen von Buchstaben und Rechtschreibfehlern.