Spellchecking in der Suche – Von Andrew Fiala und dem System Wesley

Als Händler mit mehr als 10 Millionen Artikeln, die zumeist auch noch umgangssprachlich benannt sind, muss man einiges an Herausforderungen im Bereich der Suche meistern, um den Kundenwünschen gerecht zu werden. Einer der Aspekte, mit denen man klarkommen muss, sind die vielen kreativen Schreibweisen, mit denen Kunden nach Büchern, Filmen, etc. suchen – was ihnen durch schwungvoll fremdsprachige Titel und Kunstworte auch nicht leicht gemacht wird.

Um den manuellen Aufwand bei der Pflege der Suche möglichst gering zu halten, werden Komponenten benötigt, die automatisch und möglichst intelligent die Suchbegriffe des Kunden interpretieren und das Passende heraussuchen. Dieser Artikel beschäftigt sich mit dem Mechanismen des Spellcheckings, d.h. der Korrektur falsch geschriebener Suchbegriffe.

Der Mix machts

Im Bereich der eCommerce-Suche gibt es keinen „goldenen Hammer“, der magisch alle Probleme, Missstände und Datenlücken behebt. Das gilt auch für die Korrektur von Benutzereingaben. Für den Thalia-Webshop setzen wir Lucidworks Fusion ein, das unter der Haube die bewährte OpenSource-Suchetechnologie Apache Solr verwendet.
Innerhalb von Solr gibt es gleich einen ganzen Satz an Spellchecker-Komponenten, aus denen der geneigte Verwalter sich eine für seinen Anwendungsfall am besten geeignete Lösung „zusammenstöpseln“ kann. Jede dieser Komponenten ist für einen bestimmten Anwendungsfall gedacht, eine Gesamtfunktionalität ergibt sich erst aus der Kombination der einzelnen Bausteine.

Heißt es *Lalaland*, *Lala Land* oder *La La Land*? Damit ein Kunde das nicht wissen muss, gibt es den `WordBreakSolrSpellChecker`, der zusätzliche Worttrennungen hinzufügt oder bestehende entfernt, um den Treffern im Index

näher zu kommen.
T.C. Boile wird durch durch `DirectSolrSpellChecker` zum Bestsellerautor *T.C. Boyle*, indem alternative Schreibweisen in einer gewissen Levenshtein-Distanz zum Ausgangsbegriff gebildet werden.

Diese beiden Spellchecker-Komponenten lösen – zusammen mit einem ordentlichen Synonymwörterbuch – nahezu alle falschen Schreibweisen in Ein-Wort-Suchen.

Mehr Wörter, mehr Probleme

Der Kunde hat von einer neuen Serie gehört, von Kritikern gelobt, bei Zuschauern hoch umstritten. Aber wie hieß die noch? Achja: *Tote Mädchen liegen nicht*. Nah dran, aber sofern der Kunde nicht eine mäßige Teenager-Zombie-Serie suchen wollte, müssen wir ihm hier etwas unter die Arme greifen, damit er zu seiner eigentlich gemeinten Serien *Tote Mädchen lügen nicht* kommt.

Hier kommt zunächst das Thema Collations ins Spiel, das Solr veranlasst, einen oder mehrere neue Suchbegriffe aus den „Verbesserungen" der einzelnen Tokens zu generieren. Da die oben erwähnten Komponenten nur auf Token-Ebene arbeiten, führt dies zu eher humoristisch-poetischen Korrekturvorschlägen wie *rote Märchen liehen Nacht* oder *Täte Mädchen liefen noch*, da nur jedes einzelne Token für sich mutiert wird und aus der Vielzahl von Umschreibungen dann neue Begriffe gebildet werden. Das ist langsam, ineffizient und vor allem nicht zielführend: man kann innerhalb der kurzen Zeitspanne, die ein Kunde bereit ist zu warten, nicht beliebig viele Permutationen durchrechnen und bewerten (dazu gleich mehr). Also setzt man dem ganzen ein abruptes Ende und wenn sich die richtige Version des Suchbegriffs nicht zufälligerweise unter den neu gebildeten befindet, hat man nicht nur viel Zeit aufgewendet, sondern steht auch noch mit leeren Händen da und kann dem Kunden nur den äußerst umsatzfeindlichen Satz *„Wir haben leider keine Ergebnisse zu Ihrer Suche gefunden."* anbieten.

Um auch bei Mehrwort-Suchen eine sinnvolle Verbesserung anbieten zu können, nutzen wir zusätzlich noch die `ShingleFilterFactory`, die den Suchbegriff in Wortgruppen zerlegt und diese in den weiteren Schritten als ein Token behandelt, also eine unzertrennliche Sammlung aus Zeichen. Die minimale und maximale Shingle-Länge ist konfigurierbar und wirkt sich darauf aus, wieviele Tokens mindestens oder höchstens in einem Shingle enthalten sein werden.

Nehmen wir für das Beispiel 1 als Minimum und 4 als Maximum, ergeben sich folgende „Shingle-Tokens“:

- Tote
- Mädchen
- liegen
- nicht
- Tote Mädchen
- Mädchen liegen
- liegen nicht
- Tote Mädchen liegen
- Mädchen liegen nicht
- Tote Mädchen liegen nicht

Damit diese Bildung korrekt funktioniert, muss der Suchbegriff `q=...` in einem explizit aufgeführten Parameter `spellcheck.q=...` zusätzlich an den Spellchecker übergeben werden. Nur dann funktioniert die Shingle-Bildung korrekt, da Solr andernfalls zuerst eine Aufteilung an Leerzeichen vornimmt und keine Shingles gebildet werden.

Die so erzeugten Shingles werden dann von den bereits bekannten Spellchecker-Komponenten überprüft und verändert und hier merkt der `DirectSolrSpellChecker` recht schnell, dass sich *Tote Mädchen liegen nicht* mit einer Editierdistanz von 2 in das im Index vorkommende *Tote Mädchen lügen nicht* ändern lässt.

Mit dieser Spellcheck-Konfiguration werden nicht nur sinnvollere Vorschläge bei Mehrwort-Suchen gefunden, sondern auch noch in viel kürzerer Zeit. Gegenüber dem „dummen“ Spellchecking haben wir einen Geschwindigkeitsvorteil (abhängig vom Suchbegriff) zwischen Faktor 5 und 50 gemessen.

Viel hilft viel?

Nur Vorschläge für Suchbegriffkorrekturen erzeugen alleine reicht allerdings nicht, man muss auch noch bewerten können, ob eine Korrektur sinnvoll ist oder nicht.
Out-of-the-box bietet Solr hier die Bewertung von Alternativen durch ihre Anzahl Treffer, die sie erzeugen würde, hätte man danach gesucht. Mit dem Schalter

`spellcheck.onlyMorePopular` teilt man dem System mit, dass man nur Vorschläge sehen möchte, die mehr Treffer als der eigentliche Suchbegriff erzielen.

Was zunächst sinnvoll aussieht – viel hilft viel – erweist sich bei näherer Betrachtung als Mogelpackung: da man nicht im Voraus weiß, ob der Suchbegriff des Kunden überhaupt „falsch“ ist oder nicht, führt man immer ein „brute-force“ Spellchecking durch, d.h. es werden immer Alternativen generiert. Nutzt ein Kunde den exakten Titel eines Artikels oder sucht in einer dünn besetzten Nische, findet er nur wenige (aber richtige) Treffer, der Spellchecker schlägt aber Korrekturen mit mehr Treffern vor. Diese sind allesamt unsinnig, da der Kunde schon gefunden hat, was er sucht. So wird z.B. aus dem Romantitel *Sylter Wellen*, der nur einen Treffer findet, die Alternative *System Wesley*, was etwa 50 Treffer im Bereich der Unix-Administration liefert. Mehr Treffer: ja, höhere Relevanz: nein. Ähnlich ergeht es der Autorin *Andrea Ficala*, die bislang „nur“ ein einziges, aber gefragtes Buch geschrieben hat. Die Umschreibung zu *Andrew Fiala* liefert zwar mehr Treffer, geht aber auch am Thema vorbei und sorgt für Unverständnis beim Kunden.

Bessere Vorschlagsauswahl

Der erste Schritt ist dem Kunden die Korrekturen nur als Vorschläge zu unterbreiten und ihn nur in 0-Treffer-Fällen automatisch zu korrigieren. *„Ihre Suche nach A ergab 1 Treffer. Meinten Sie vielleicht B?“*. So kann der Kunde selber entscheiden kann, ob er die Korrektur annimmt oder nicht. Jetzt wäre es noch toll, wenn dort kein Blödsinn angezeigt werden würde.

Wir nutzen für die Vorschlagsauswahl neben der reinen Trefferzahl ein zusätzliches Kriterium: wie oft Kunden nach einem Suchbegriff suchten. Für die Vervollständigung von Begriffen in der Suchbox (Suggest) speichern wir alle Suchanfragen, die zu Treffern führten (ohne Kundenbezug). Über diese Collection lässt sich ermitteln, wieviele Kundensuchen es nach dem einen oder anderen Suchbegriff gab.

Für die Vorschläge des Spellcheckers und den ursprünglichen Suchbegriff wird ermittelt, wieviele Kunden den jeweiligen Suchbegriff gesucht haben. Ist die Anzahl der Suchen für den Begriff hoch, ist auch die Wahrscheinlichkeit hoch, dass die Korrektur richtig oder zumindest sinnvoll ist. Wir bewerten Anzahl

Suchanfragen deutlich höher als Anzahl Treffer und verhindern darüber die meisten unsinnigen Korrekturvorschläge.

Über diese zusätzliche Zählung lässt sich zudem ein Sonderfall erkennen und behandeln: es gibt Artikel, die schnell ausverkauft sind, z.B. weil sie limitiert oder nicht gut beschaffbar sind. Diese generieren in einem kurzen Zeitraum eine hohe Anzahl an Suchen, liefern aber nach ihrem Ausverkauf keinerlei Treffer mehr. Gibt es zu einem Suchbegriff keine Treffer, ist aber die Anzahl an Suchen hoch (wohlgemerkt, wir speichern nur Suchen, die zum jeweiligen Zeitpunkt auch Treffer hatten), kann daraus gefolgert werden, dass der initiale Suchbegriff mit hoher Wahrscheinlichkeit auch das war, was der Kunde suchen wollte. In diesem Fall ließe sich z.B. die Anzeige eines Korrekturvorschlags unterdrücken und eine *„Es tut uns leid, der von Ihnen gesuchte Artikel ist derzeit nicht im Sortiment.“*-Nachricht ausspielen.

Fazit

Es gibt keine allgemeingültige Lösung für das Thema Spellchecking in der eCommerce-Suche, aber mit einer Kombination aus Solr-Bordmitteln und zusätzlicher, eigener Logik und entsprechenden Verkehrsdaten lässt sich ein passables Grundgerüst aufsetzen, dass viele alltägliche Situationen automatisiert abhandeln kann.

Die Autoren

Jan Marten, Developer im Team *Suche & Beraten*

Hendrik Busch, IT Analyst im Team *Suche & Beraten*