Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Human-Centered Computing (HCC)

QualityInsight: Explanation Interface für das articlequality Modell von ORES

Tanita Daniel

Betreuerin: Alexa Schlegel

Erstgutachterin: Prof. Dr. C. Müller-Birn

Zweitgutachter: Prof. Dr. L. Prechelt

Berlin, 30. August 2021

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 30. August 2021

Zusammenfassung

ORES ist ein Dienst, der *Machine Learning* Modelle für Wikipedia Projekte bereitstellt. Diese Modelle sollen bei der Aufrechterhaltung eines gewissen Qualitätsstandards auf Wikipedia-Seiten aushelfen und die Arbeitslast von Wikipedia Mitarbeitern verringern. Eines dieser Modelle ist das *articlequality* Modell, was eine automatische Qualitätsbestimmung von Artikeln nach den Noten und Kriterien der jeweiligen Wikipedia vornimmt. Die Ausgaben des Modells sind öffentlich zugänglich, diese liefern jedoch nur wenige Informationen darüber, warum ein Artikel eine bestimmte Note erhalten hat.

Vorhersagen des *articlequality* Modells sind, ohne technisches Wissen und *Machine Learning* Erfahrungen, schwer nachvollziehbar. Sie sind daher besonders für Editoren oder andere unerfahrene Nutzer von Wikipedia in ihrer Ursprungsform unzureichend. Um Vorhersagen und *Machine Learning* Modelle interpretierbar zu machen, bedarf es sogenannten *Explainable AIs*, die helfen sollen, Vorhersagen zu erklären.

Eine Anwendung mit einem intuitiven *human-centered* Design könnte es, mit Hilfe von Visualisierungen und Erklärungen, möglich machen, die Vorhersagen des *articlequality* Modells verständlich und nützlich zu machen. Für anstrebende oder bereits erfahrene Editoren liegt der größte Nutzen des Modells darin, zu erfahren, wie Artikel weiter verbessert werden können. Dafür müssen diese aber verstehen, welche Eigenschaften einen Artikel besser machen, was durch die Erklärung von Vorhersagen und davon abhängigen Tipps innerhalb der Anwendung realisiert werden könnte.

Um eine solche Anwendung zu erstellen, wurde zunächst ein *Interface* Prototyp erstellt, der in mehreren Iterationen von potentiellen Nutzern getestet wurde. Die Ergebnisse aus den *Usability Tests* wurden zu der Verbesserung des Prototyps vor dem nächsten Test genutzt, um am Ende ein intuitives Design zu erhalten.

Nachdem das *User Interface* festgelegt wurde, implementierte ich die Anwendung 'QualityInsight'. In dieser ist es möglich, nach Artikeln zu suchen und eine bestimmte Revision auszuwählen, für die der Nutzer eine Vorhersage und Erklärung erhalten möchte. In der Anwendung werden einem Nutzer dann Informationen zu der Vorhersage und Visualisierungen in Form von, mit LIME erstellten, Graphen zur Verfügung gestellt. Unterstützend gibt es eine Erklärung zu dem Vorhersage-Ergebnis und zusätzliche Tipps, die bei der Verbesserung des Artikels helfen sollen. Fachbegriffe und die *Feature* (Merkmale) des *articlequality* Modells stehen in der Anwendung auch zur Verfügung.

Für die Evaluierung der erstellten Anwendung wurden asynchrone *Usability Tests* durchgeführt. Diese ergaben, dass die Anwendung zwar die Vorhersagen interpretierbar machten und gut nutzbar war, aber noch nicht wirklich dabei half, zu wissen, was genau an einem Artikel verbessert werden musste.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Zielstellung . . . 2
1.3 Methode . . . 2

2 Hintergrund der Arbeit **5**
2.1 ORES . . . 5
2.1.1 Article Quality Modell . . . 6
2.1.2 Stakeholder . . . 13
2.2 Explainable AI . . . 13
2.2.1 LIME . . . 15

3 Entwicklung **17**
3.1 Prototyping . . . 17
3.1.1 Erste Analyse . . . 17
3.1.2 Erster Prototyp . . . 20
3.1.3 Erste Evaluation . . . 23
3.1.4 Zweite Analyse und Änderungen am Prototypen . . . 25
3.1.5 Zweite Evaluation . . . 27
3.1.6 Dritte Analyse und Änderungen am Prototypen . . . 29
3.2 Ergebnisse . . . 30
3.3 Implementierung . . . 30
3.3.1 Backend . . . 30
3.3.2 Frontend . . . 32

4 Evaluation **37**
4.1 Vorgehen . . . 37
4.2 Ergebnisse . . . 38

5 Diskussion **41**

6 Fazit **43**
6.1 Ausblick . . . 43

Literatur **45**

Anhang **47**
6.2 Einwilligungserklärungen . . . 48
6.3 Code . . . 51

Abbildungsverzeichnis

1.1 Schema Methodik 3

2.1 Schema Funktionalität *XAI* 14
2.2 Beispiel: Visualisierung mit LIME 15

3.1 Erster Prototyp: Startseite 21
3.2 Erster Prototyp: Artikelauswahl 21
3.3 Erster Prototyp: Revisionensauswahl 22
3.4 Erster Prototyp: Vorhersage Erklärung 23
3.5 Zweiter Prototyp: Vorhersage Erklärung 26
3.6 Zweiter Prototyp: Ausschnitt der Revisionsauswahl 27
3.7 Zweiter Prototyp: Ausschnitt der Artikelauswahl 27
3.8 Anwendung: Startseite 33
3.9 Anwendung: Artikelauswahl 33
3.10 Anwendung: Revisionen 34
3.11 Anwendung: Explanation 35
3.12 Anwendung: Hover-Effekt 35

Tabellenverzeichnis

2.1 Artikelqualitätseinstufung der englischen Wikipedia [Wik21] . . 8
2.2 *Feature* des *articlequality* Modells in der englischen Wikipedia . 9

3.1 Informationen der Teilnehmer der ersten Evaluation 24
3.2 Informationen der Teilnehmer der zweiten Evaluation 28

4.1 Ergebnisse der letzten Evaluation 38

1 Einleitung

Heutzutage können *Machine Learning* Algorithmen und künstliche Intelligenzen überall gefunden werden. Bei einigen Anwendungen und Systemen kann dies offensichtlich für die Nutzer sein, bei anderen wiederum nicht. Dies heißt auch, dass Nutzer direkt oder indirekt von diesen Algorithmen beeinflusst werden können. Dabei kommen folgende Fragen auf:

- Sollte es nicht transparenter gemacht werden, wann, wo und wie *Machine Learning* Algorithmen verwendet werden?
- Worauf basieren die Entscheidungen und Vorhersagen der Algorithmen?
- Könnte daraus profitiert werden, wenn Betroffene der Algorithmen diese Vorhersagen verstehen?
- Wie könnte man Entscheidungen und Vorhersagen so erklären, dass Betroffene die Erkenntnisse zu ihrem Vorteil nutzen können?

1.1 Motivation

Wikipedia-Seiten nutzen *Machine Learning (ML)* Modelle für die unterschiedlichsten Aufgaben. Die dafür genutzten Modelle werden von ORES, was im Abschnitt 2.1 genauer erklärt wird, zur Verfügung gestellt. Eines davon, das *articlequality* Modell, bewertet Artikel Revisionen automatisch nach ihren strukturellen Eigenschaften, um, zum Beispiel, den Artikel-Fortschritt nachverfolgbar zu machen. Mehr zu diesem Modell wird im Abschnitt 2.1.1 erklärt.

Die Modelle von ORES sind offen zugänglich und auch die Ausgaben der Vorhersagen dieser Modelle für die jeweiligen Wikipedia-Seiten können über eine Web-Schnittstelle abgerufen werden. Um diese Schnittstelle nutzen zu können, muss natürlich erst von der Existenz von ORES gewusst werden und selbst dann könnten die Ausgaben der Vorhersage nicht aufschlussreich für den Nutzer sein und das Verhalten des Modells erklären.

Für einige ORES Modelle wurden bereits Anwendungen und Visualisierungstools geschrieben, um diese Ausgaben besser aufzubereiten oder die Modelle verständlicher zu machen. Ein Beispiel wäre das Visualisierungstool PreCall

[KMG+19]. Dies wurde entwickelt, um das *damaging* Modell von ORES verständlicher zu machen. Hier wurde es durch die Manipulation der Daten innerhalb des Modells möglich gemacht, das Verhalten des Modells besser analysieren und verbessern zu können. Viele dieser Visualisierungstools sind an *Patroller* oder *Tool-Entwickler* (Abschnitt 2.1.2) gerichtet, um die Modelle anpassen zu können. Anwendungen für Editoren oder einfache Leser auf Wikipedia, im Bezug zu ORES *articlequality* Modell, sind nach meiner Recherche kaum zu finden. Die Anwendungen, die ich finden konnte, nutzen nicht das *articlequality* Modell für die Qualitätsanalyse [BAKSF13] [CHF10].

In der Bachelorarbeit von Gnanasegaram [Gna20] wurde bereits eine Anwendung für Wikidata im Bezug auf die automatisierte Qualitätsbewertung entwickelt, wobei es sich hier um das *itemquality* Modell handelte. Das Ziel dieser Anwendung überschneidet sich, trotz der unterschiedlichen Modelle, mit dem, was in dieser Arbeit verfolgt wird: Ein *ML*-Modell, welches sich um die automatisierte Qualitätsbewertung kümmert, interpretierbar machen.

1.2 Zielstellung

Machine Learning Algorithmen und deren Vorhersagen sind schwer zu verstehen und ORES *articlequality* Modell ist keine Ausnahme davon. *ML* Algorithmen sollten zugänglicher für Betroffene und Nutzer des Algorithmus gemacht werden. Nutzer, besonders auch Editoren, von Wikipedia könnten davon profitieren, zu verstehen, wie das *articlequality* Modell funktioniert und warum es bestimmte Vorhersagen trifft.

Mein Ziel ist es, ein Visualisierungstool für Editoren zu implementieren, welches die Vorhersagen des *articlequality* Modells von ORES für Artikel auf der englischen Wikipedia erklärt. Editoren sollten so einen besseren Einblick in die Bewertung von Wikipedia-Artikeln erhalten und ihnen einen neuen Weg geben, zu lernen, wie sie Artikel auf einer strukturellen Stufe verbessern können. Die Anwendung könnte ebenfalls als Einstieg für neue Editoren oder Nutzer, die an der Bearbeitung von Wikipedia-Artikeln interessiert sind, dienen.

1.3 Methode

Diese Arbeit wird einem *Human-centered* (menschenzentrierten) Designansatz [Fou21] folgen, was bedeutet, dass die Werte und Ziele der Nutzer im Mittelpunkt stehen. Es handelt sich dabei um einen iterativen Prozess, wie in Abbildung 1.1 dargestellt. Dabei soll das *User Interface* der Anwendung in mehreren Durchläufen getestet und verbessert werden.

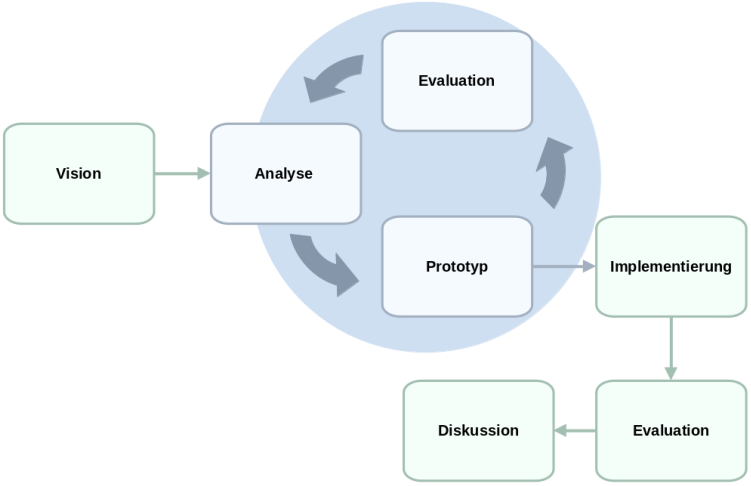

Abbildung 1.1: Schema Methodik

Eine detailliertere Erklärung der Phasen ist im Folgenden beschrieben:

1. **Vision**
   Es soll ein Konzept für die Anwendung entwickelt werden, was die Ziele der Anwendung und die Anforderung an diese zusammenfassen soll. Dies wurde im vorigen Abschnitt 1.2 beschrieben.

2. Die folgenden Phasen werden in zwei Iterationen durchgegangen:

   a) **Analyse**
   Da die hauptsächliche Zielgruppe der Anwendung die Editoren von Wikipedia sind, wird in der ersten Analyse recherchiert, welche Werte und Bedürfnisse diese haben. Um eine nutzerfreundliche Oberfläche gestalten zu können, werden dazu auch noch Informationen und Regeln zusammengetragen, die eingehalten werden sollten. In den nächsten Iterationen sollen dann die Ergebnisse aus den *Usability Tests* genutzt werden.

   b) **Prototyp**
   Aus den Ergebnissen der Analyse soll ein Prototyp erstellt werden, der dann in den nächsten Iterationen verbessert werden soll.

   c) **Evaluation**
   Die Prototypen sollen von Nutzern mit Erfahrungen bezüglich Wikipedia oder *Machine Learning* getestet werden, um mögliche Probleme aufdecken zu können.

3. **Implementierung**
   Der finale Prototyp wird als Vorlage für die Anwendung, die nun implementiert werden soll, genutzt. Für die Visualisierung soll LIME (Abschnitt 2.2.1) genutzt werden. Für die folgende Phase muss die fertige Anwendung für andere Personen zugänglich gemacht werden.

4. **Usability Test**
   Bestenfalls soll die Anwendung von Editoren der englischen Wikipedia-Seite getestet werden. Dabei werden Eindrücke und Meinungen zu der Anwendung gesammelt.

5. **Diskussion**
   Die Ergebnisse aus der letzten Evaluation werden betrachtet und eingeordnet.

In der vorliegenden Arbeit werden einige englische Bezeichnungen und Fachbegriffe genutzt, welche durch kursive Schrift erkennbar sind. Für personenbezogene Ausdrücke wird die männliche Schreibweise verwendet.

2 Hintergrund der Arbeit

Der Fokus dieser Arbeit liegt darauf, die Entscheidungen und Ausgaben von *Machine Learning (ML)* Algorithmen für Nutzer interpretierbar zu machen, die vielleicht wenig bis kein Vorwissen zu *ML* vorweisen können. Dafür muss zunächst verstanden werden, worum es sich bei dem genutzten *ML* Modell handelt, was der Zweck des Modells ist und wer davon betroffen ist.

In den folgenden Abschnitten werde ich erklären, was ORES ist und wofür es verwendet wird, außerdem wie das *articlequality* Modell von ORES funktioniert und wofür dieses verwendet wird. Danach gehe ich darauf ein, wer die *Stakeholder* von ORES sind und was *Explainable AI* in diesem Zusammenhang bedeutet.

2.1 ORES

Es muss viel Arbeit geleistet werden, um die Qualität der Artikel auf Wikipedia zu erhalten. Je mehr Anpassungen an Artikeln vorgenommen werden, desto häufiger müssen die Inhalte dieser Artikel von den freiwilligen Mitgliedern der Wikipedia-Community überprüft werden. In der Vergangenheit führte dies dazu, dass es nicht genug Menschen gab, die Artikel überprüften. Dadurch kam es dazu, dass Vandalismus an Artikeln von Lesern gesehen werden konnte. Um dies zu verhindern und die Wikipedia Community zu unterstützen, wurde ORES entwickelt. [HG20]

ORES ist ein API-basierter Webdienst, der vom *Machine Learning* Team der Wikimedia Foundation gepflegt wird und Wikimedia-Projekten *ML* als Dienst zur Verfügung stellt [Med21a] [HG20]. ORES kann als eine frei nutzbare Auswahl von *ML* Modellen betrachtet werden, die mit frei zugänglichen Daten[1] [Med21b] trainiert wurden. Jedes Modell hat dabei einen anderen Zweck und kann jederzeit auf unterschiedliche Inhalte auf Wikipedia angewandt werden [HG20].

Für jeden Inhalt auf Wikipedia können Vorhersagen über HTTP angefordert werden. ORES stellt dafür eine API[2] zur Verfügung. Als Antwort erhält der Benutzer ein JSON-Objekt mit *Scores*, das sowohl für Menschen, als auch für Maschinen lesbar ist. Es enthält die Vorhersage und ein Set von Wahrscheinlichkeiten für jede vorhandene Klasse im *ML* Modell. [HG20]

[1] `https://en.wikipedia.org/wiki/Open_data`, besucht am: 30. Juni 2021
[2] `https://ores.wikimedia.org/`, besucht am: 30. Juni 2021

Wie diese *Scores* genutzt werden, bleibt dem Benutzer überlassen, da ORES nur *Scores* liefert, aber keine Anwendung dafür vorgibt. Die *Tool-Entwickler* (Abschnitt 2.1.2) von Wikipedia stellen bereits einige Wege zur Verfügung, um die Vorhersagen der *ML* Modelle auf den Wikipedia Plattformen zu nutzen. [Med21a]

Je nach verwendetem ORES-Modell können diese *Scores* die vorhergesagte Qualität von *Edits* (Bearbeitungen der Artikel) darstellen oder dabei helfen, festzustellen, ob ein *Edit* schädlich ist und ob er gut gemeint ist. Bearbeitete Artikel, die beschädigt wurden oder zurückgesetzt werden müssen, können durch die Verwendung von ORES *Scores* leichter erkannt werden. So können die Vorhersagen helfen, den Inhalt von Wikipedia zu verbessern und ein gewisses Qualitätslevel aufrechtzuerhalten. [Med21b]

2.1.1 Article Quality Modell

Um Vandalismus und Spam zu verhindern, muss jede neue Seite und jede Bearbeitung von Artikeln überprüft und kuratiert werden. Dies kann von freiwilligen Mitgliedern einzeln erledigt werden, läuft aber ohne Automatisierung auf eine Menge Arbeit hinaus und führt zu einer inkonsistenten Abdeckung der Artikelbewertung. [Med21a]

ORES bietet das *draftquality* Modell an, was neue Artikel und Änderungen an Artikeln evaluiert und voraussagt, ob ein neuer Artikel gelöscht oder zurückgesetzt werden muss. [Med21a]

Das *articlequality* Modell hingegen evaluiert bereits existierende Artikel und wurde dazu trainiert, die Artikel-Evaluation von Menschen nachzuempfinden [Med21a]. Für jeden Artikel wird so automatisch eine Note vorausgesagt. Diese Noten sind einer Skala zu entnehmen, die größere Wikipedia-Seiten nutzen, um die Qualität der Artikel zu bewerten [Med21a].

Die Voraussagen basieren auf strukturellen Eigenschaften des Artikels wie z.B. der Anzahl an Abschnitten, ob es eine Infobox gibt oder wie viele Referenzen es gibt [Med21a]. Das heißt aber gleichzeitig auch, dass das Modell nicht den Schreibstil bewerten kann. Laut dem MediaWiki-Artikel über ORES [Med21a] beeinflusst dies jedoch nicht den Nutzen des Modells, da ein gut strukturierter Artikel oft mit einem guten Schreibstil einhergeht.

Wie zuvor erwähnt, nutzen Wikipedia-Seiten für die Bewertung der Qualität eine, für jede Wikipedia-Seite individuell, definierte Skala, um den Fortschritt eines Artikels und seine Schwachstellen zu kennzeichnen [SYS+20]. ORES *articlequality* Modell verwendet für seine Vorhersagen die Skala der jeweiligen Wikipedia-Seite. Dafür definiert es jede Note als eine eigene Klasse im *ML* Modell und berechnet die Wahrscheinlichkeit für jede Note der Skala [Med21a].

Die Note mit der höchsten Wahrscheinlichkeit gilt als die aktuelle Vorhersage des Artikels. Vorhersagen werden, wie in Abschnitt 2.1 zuvor erwähnt, als JSON-Objekt ausgegeben. Eine Vorhersage für die englische Wikipedia[3] ist im Listing 2.1 abgebildet.


```
{
  "enwiki": {
    "models": {
      "articlequality": {
        "version": "0.8.2"
      }
    },
    "scores": {
      "990999446": {
        "articlequality": {
          "score": {
            "prediction": "C",
            "probability": {
              "B": 0.22225375546918866,
              "C": 0.3443505649213038,
              "FA": 0.007961517998375519,
              "GA": 0.21463376025467823,
              "Start": 0.20420514408046725,
              "Stub": 0.0065952572759865385
            }
          }
        }
      }
    }
  }
}
```


Listing 2.1: JSON-Objekt: Vorhersage für die Revision 990999446

Wie das JSON-Objekt verrät, kann erkannt werden, dass die englische Wikipedia die Noten *Stub*, *Start*, *C*, *B*, *GA* und *FA* vergibt. In der Tabelle 2.1 werden die Noten näher erläutert [Wik21]. Unter *Prediction* (Vorhersage) kann die vorhergesagte Note abgelesen werden, während unter *Probabilities* die Wahrscheinlichkeiten der einzelnen Noten betrachtet werden können. Zusätzlich kann man sich bei einer solchen Abfrage auch die *Feature* (Merkmale) des Modells und deren Werte für den gewählten Artikel ausgeben lassen.

[3] `https://en.wikipedia.org/`, besucht am: 26. Juli 2021

Tabelle 2.1: Artikelqualitätseinstufung der englischen Wikipedia [Wik21]

Note	**Beschreibung**
Stub	Beschreibt einen sehr kurzen Artikel, einen Artikel mit nur einer rudimentären Sammlung von Informationen oder einen mit irrelevantem Inhalt. Artikel, die diese Bewertung erhalten, haben das Risiko gelöscht zu werden.
Start	Beschreibt einen kürzeren Artikel mit teilweise gutem Inhalt, was aber noch nicht ganz ausreichend ist, um das Thema abzudecken. Der Artikel kann zuverlässige und gute Quellen haben, ist aber an sich noch unvollständig.
C	Beschreibt einen Artikel, der nützliche Informationen enthält, aber dem dennoch einige wichtige Informationen fehlen. Es kann aber auch ein Artikel mit teilweise irrelevantem Inhalt sein, der diese Note erhält. Der Artikel muss bei dieser Bewertung bereits zuverlässige Quellen haben.
B	Beschreibt einen fast vollständigen Artikel, der angemessen referenziert ist, das Thema ausreichend abgedeckt hat, eine klare Struktur hat, gut geschrieben ist, Diagramme oder Bilder als unterstützendes Material enthält, eine Infobox besitzt und für ein breites Publikum verständlich formuliert ist.
GA	Beschreibt einen guten Artikel, der von einem oder mehreren *Reviewern* geprüft wurde. Der Artikel muss gut geschrieben sein, für ein breites Publikum verständlich sein, den *Style Guidelines*[4] folgen, eine Liste aller Referenzen vorweisen, keine Copyright-Verletzungen enthalten, nicht vom Thema abschweifen und neutral bleiben. Es müssen ebenfalls alle verwendeten Zitate aus verlässlichen Quellen beziehen und es sollten Bilder, Videos oder Audiodateien als unterstützendes Material vorhanden sein.
FA	Beschreibt einen Artikel, der den Status 'feature article' erreicht hat, indem er eine Prüfung durch *Reviewer* bestanden hat. Der Artikel muss gut geschrieben sein, das Thema umfangreich abdecken, gut recherchiert sein, unvoreingenommen sein, die *Wikipedia copyright policy*[5] befolgen, den *Style Guidelines* folgen, nicht vom Thema abschweifen und im *Summary Style*[6] geschrieben sein.

[4] `https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style`, besucht am: 26. Juli 2021

[5] `https://en.wikipedia.org/wiki/Wikipedia:Copyrights` , besucht am: 26. Juli 2021

[6] `https://en.wikipedia.org/wiki/Wikipedia:Summary_style`, besucht am: 26. Juli 2021

Das *articlequality* Modell von ORES verwendet *Feature*, um die strukturellen Merkmale eines Artikels einordnen zu können, und verwendet diese dann, um eine Vorhersage zu berechnen.

Nicht jede Wikipedia-Seite, die mit ORES Modellen arbeitet, verwendet das *articlequality* Modell. Es gibt 13 Wikipedia-Projekte, die das *articlequality* Modell nutzen und dazu gehören enwiki, euwiki, fawiki, frwiki, glwiki, nlwiki, ptwiki, ruwiki, simplewiki, svwiki, testwiki, trwiki und ukwiki[7]. Sie alle nutzen unterschiedliche Sets der besagten *Feature*.

In dieser Arbeit wird enwiki, das englische Wikipedia Projekt, betrachtet. Die englische Wikipedia nutzt insgesamt 25 *Feature* (Tabelle 2.2), die als Integer-Werte dargestellt werden und zur Berechnung der *articlequality* Scores verwendet werden. Einige dieser *Feature* beruhen auf der Anzahl von *Templates*. Diese *Templates* werden von Wikipedia selber für die Bearbeitung von Artikeln zur Verfügung gestellt und können damit das Einbinden von Bildern oder Referenzen vereinfachen.

Tabelle 2.2: *Feature* des *articlequality* Modells in der englischen Wikipedia

Beschreibung: Die Bedeutung der einzelnen *Feature* lässt sich bei einigen durch den Namen ganz gut ablesen, bei anderen konnte nur abgeschätzt werden, was sie bedeuten. Um klar zu machen, welche *Feature* davon betroffen sind, wird in der letzten Reihe der Grad an Zuversicht beschrieben.
H - hohe Zuversicht, M - etwas unsicher, N - unsichere Bedeutung.

Name	**Beschreibung**	**Zuversicht**
feature.english .stemmed.revision .stems_length	Die durchschnittliche Länge der Wortstämme. Genutzte Wörter in Artikeln können auf ihre Wortstämme durch Stemming reduziert werden. Stemming ist eine Form der Textnormalisierung, bei der die Textdimension reduziert wird.[8]	N
feature.enwiki .infobox_images	Die Anzahl der Bilder innerhalb von *Infobox Templates*[9].	M
	Fortsetzung auf der nächsten Seite	

[7] `https://ores.wikimedia.org/v3/scores/?model_info`, besucht am: 26. Juli 2021
[8] `https://towardsdatascience.com/stemming-corpus-with-nltk-7a6a6d02d3e5`, besucht am: 01. August 2021
[9] `https://en.wikipedia.org/wiki/Template:Infobox`, besucht am: 26. Juli 2021

Tabelle 2.2 *Feature* des *articlequality* Modells in der englischen Wikipedia

Name	Beschreibung	Zuversicht
feature.enwiki .main_article_templates	Die Menge der verwendeten *Main Article Templates*[10]. Dieses Template enthält einen Link zu einem Artikel mit einem untergeordneten Thema, bevor dieses im Hauptartikel zusammengefasst wird.	H
feature.enwiki.revision .category_links	Die Anzahl der Links zu den Wikipedia-Kategorien zu dem der Artikel gehört.	H
feature.enwiki.revision .cite_templates	Die Menge der verwendeten *Citation Templates*[11]. *Citation Templates* werden verwendet, um Referenzierungen für Bücher, Webseiten oder andere Referenzen zu erzeugen.	M
feature.enwiki.revision .cn_templates	Die Anzahl der verwendeten *Citation needed Templates*[12]. Dieses Template wird verwendet, um Anmerkungen im Artikel zu machen, wo Quellenangaben fehlen. Nach dem betreffenden Satz steht dann [citation needed].	H
feature.enwiki.revision .image_links	Die Menge an Bildern außerhalb von Templates.	M
feature.enwiki.revision .image_template	Die Anzahl an *Image Request Templates*[13]. Diese Templates sind üblicherweise auf der Talk-Seite des Artikels zu finden und werden verwendet, um ein Bild für einen Artikel anzufordern.	N
feature.enwiki.revision .images_in_tags	Die Menge an Bildern in den Tags.	N

Fortsetzung auf der nächsten Seite

[10] `https://en.wikipedia.org/wiki/Template:Main`, besucht am: 29. Juni 2021
[11] `https://en.wikipedia.org/wiki/Template:Citation`, besucht am: 29. Juni 2021
[12] `https://en.wikipedia.org/wiki/Template:Citation_needed`, besucht am: 29. Juni 2021
[13] `https://en.wikipedia.org/wiki/Template:Image_requested`, besucht am 29. Juni 2021

Tabelle 2.2 *Feature* des *articlequality* Modells in der englischen Wikipedia

Name	Beschreibung	Zuversicht
feature.enwiki.revision .images_in_templates	Die Anzahl der Bilder in Templates. Dies schließt Bilder im Infobox Template aus.	M
feature.enwiki.revision .infobox_templates	Die Menge an *Infobox Templates*[14] innerhalb eines Artikels. Infoboxen werden für einen Überblick über Details verwendet. Es gibt unterschiedliche und spezialisierte Infobox Templates.	M
feature.enwiki.revision .paragraphs_without _refs_total_length	Die Anzahl der Zeichen in allen Absätzen, die keine Referenz haben.	N
feature.enwiki.revision .shortened_footnote _templates	Die Menge der verwendeten *Shortened Footnote (sfn) Templates*[15]. Dieses Template erstellt eine Autoren- und Datumsangabe in einer Fußnote. Sie kann verwendet werden, um identische Literaturangaben zu kombinieren.	N
feature.enwiki.revision .who_templates	Die Anzahl an verwendeten *Who Templates*[16]. Dieses Template wird verwendet, um Anmerkungen im Artikel zu machen, bei denen eine genaue Person angegeben werden soll. Ein Beispiel hierfür wäre, wenn im Artikel nur steht 'Experts confirm that...'. Nach dem betreffendem Wort steht dann [who?].	H
feature.len(<datasource .english.idioms .revision.matches>)	Die Anzahl der gefundenen Redewendungen im Artikel.	N

Fortsetzung auf der nächsten Seite

[14] `https://en.wikipedia.org/wiki/Category:Infobox_templates`, besucht am 26. Juli 2021

[15] `https://en.wikipedia.org/wiki/Template:Sfn`, besucht am: 29. Juni 2021

[16] `https://en.wikipedia.org/wiki/Template:Who`, besucht am: 29. Juni 2021

Tabelle 2.2 *Feature* des *articlequality* Modells in der englischen Wikipedia

Name	Beschreibung	Zuversicht
feature.len(<datasource .english.words_to_watch .revision.matches>)	Die Menge an gefundenen Wörtern, die zu den *Words To Watch*[17] gehören. Wörter, die darunter fallen, sind solche, die einen Artikel voreingenommen erscheinen lassen können. Beispiele hierfür wären 'legendary', 'some people say' oder 'but'.	H
feature.len(<datasource .wikitext.revision.words>)	Die Anzahl der Wörter im Artikel.	H
feature.wikitext.revision .chars	Die Anzahl an Zeichen im gesamten Artikel.	M
feature.wikitext.revision .content_chars	Die Anzahl an Zeichen im inhaltlichen Teil des Artikels.	H
feature.wikitext.revision .external_links	Die Menge der Links, die im Artikel zu externen Webseiten führen.	M
feature.wikitext.revision .headings_by_level(2)	Die Anzahl der Abschnitte in einem Artikel.	H
feature.wikitext.revision .headings_by_level(3)	Die Anzahl der Unterabschnitte in einem Artikel.	H
feature.wikitext.revision .ref_tags	Die Menge der Reftags in einem Artikel. Sie werden verwendet, um Quellenangaben in Fußnoten zu definieren. Einmal definiert, können diese wiederverwendet werden.	M
feature.wikitext.revision .templates	Die Anzahl aller im Artikel verwendeten Templates.	M
feature.wikitext.revision .wikilinks	Die Menge der Links, die zu anderen Wikipedia-Seiten im inhaltlichen Teil des Artikels führen. Dies schließt Links innerhalb von Templates aus.	M

[17] `https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Words_to_watch`, besucht am: 26. Juli 2021

2.1.2 Stakeholder

Wikipedia *Stakeholder* sind vielschichtig und reichen von Personen, die Software für Wikipedia entwickeln, bis zu Personen, die versuchen, den Inhalt von Wikipedia zu verbessern oder zu erweitern. Alle diese Personen kommen direkt oder indirekt mit ORES in Kontakt und können von den Entscheidungen der ORES *ML* Modelle beeinflusst werden. Die für diese Arbeit wichtigsten *Stakeholder* werden im Folgenden erläutert.

Es gibt ***Tool-Entwickler***, die Werkzeuge und Software erstellen, um die Arbeitsabläufe auf Wikipedia zu unterstützen. Diese können ORES *ML* Modelle direkt für die Entwicklung nutzen und damit beispielsweise Software entwerfen, die automatische Vandalismus an Wikipedia-Artikeln erkennt. [SYS+20]

Wissenschaftliche ***Researcher*** können ORES gezielt nutzen, um beispielsweise die Entwicklung von Wikipedia zu untersuchen oder auch zu analysieren, wie ORES diese Entwicklung beeinflusst. [SYS+20]

Patroller oder ***Reviewer*** überprüfen Artikel in der Wikipedia manuell und sind die letzte Instanz, die entscheidet, ob eine Änderung schädlich ist und rückgängig gemacht werden muss oder ob ein neuer Artikel gelöscht werden muss. Wikipedia erhält jeden Tag viele Beiträge und Menschen alleine könnten diese Menge an Artikeln nicht regelmäßig überprüfen. Aus diesem Grund hat ORES eine unterstützende Rolle für die *Patroller*. Die Modelle von ORES können jeden Artikel automatisch überprüfen, sodass die *Patroller* die Artikel nicht stichprobenartig überprüfen müssen. Falls eine manuelle Überprüfung nötig ist, werden die Patroller informiert, in dem der Artikel markiert wird. [Med21b]

Editoren sind diejenigen, die Artikel für Wikipedia schreiben und bearbeiten. Sie sind indirekt von ORES *ML* Modellen betroffen, arbeiten aber normalerweise nicht selbst damit. Auch wenn sie ORES nicht direkt nutzen, kann es für Editoren von Vorteil sein, zu verstehen, warum bestimmte Vorhersagen getroffen werden [SYS+20], wie bei dem *articlequality* Modell von ORES. Besonders für neue Editoren kann es hilfreich sein nachvollziehen zu können, woraus sich ein guter Artikel zusammensetzt und was man unternehmen kann, um einen Artikel zu verbessern.

2.2 Explainable AI

Explainable AI (XAI) ist ein Oberbegriff für Methoden, Werkzeuge und Modelle, die *ML* Modelle für Menschen verständlich und interpretierbar machen sollen. Angewendet soll es *Explanations* (Erklärungen) dazu liefern, warum ein *ML* Modell sich für eine bestimmte Voraussage entschieden hat oder wie bestimmte Ergebnisse zustande kommen. [GDC21]

Einige Modelle können aber auch recht einfach und selbsterklärend sein, das heißt, sie sind transparent und von sich selbst aus verständlich gestaltet. Je nach Modell können verschiedene Dinge verständlicher sein als andere. Zum Beispiel sind bei einigen Modellen die Vorhersagen für den Menschen gut lesbar und in manchen Fällen auch nachvollziehbar, während die genutzten Algorithmen komplex und ohne Analyse nicht verständlich sind. [BDRD+20]

Die *ML* Modelle, die erklärt werden sollen, werden meist nicht selber verändert, sondern es werden über externe Methoden die Resultate verständlich gemacht. Um Voraussagen von Modellen zu erklären, können zwei verschiedene Ansätze verfolgt werden. Es gibt den modellunabhängigen Ansatz, in der die *XAI* das *ML* Modell als *Black Box*-Modell betrachtet, was bedeutet, dass nicht bekannt ist, was im Modell vor sich geht und daher keine inneren Vorgänge des *ML* Modells nutzt, um es zu erklären. Da in diesem Fall nicht gewusst werden muss, was sich hinter dem Modell befindet, kann dieser Ansatz bei jedem *ML* Modell genutzt werden. Der zweite Ansatz wäre die Nutzung einer modellabhängigen *XAI*. Dabei werden, im Gegensatz zur modellunabhängigen Methode, Informationen innerhalb des *ML* Modells genutzt, um eine Erklärung liefern zu können. In dem Fall müsste also beispielsweise bekannt sein, welcher Algorithmus innerhalb des Modells genutzt wird. [GDC21]

Ein weiterer Punkt, in dem sich entschieden werden muss, ist, ob man eine globale oder eine lokale *Explanation* zu dem *ML* Modell haben möchte. Bei einer *global Explanation* würde eine Erklärung des gesamten Verhaltens des *ML* Modells ausgegeben werden, während bei einer *local Explanation* nur eine Erklärung für eine bestimmte Vorhersage erhalten wird. [GDC21]

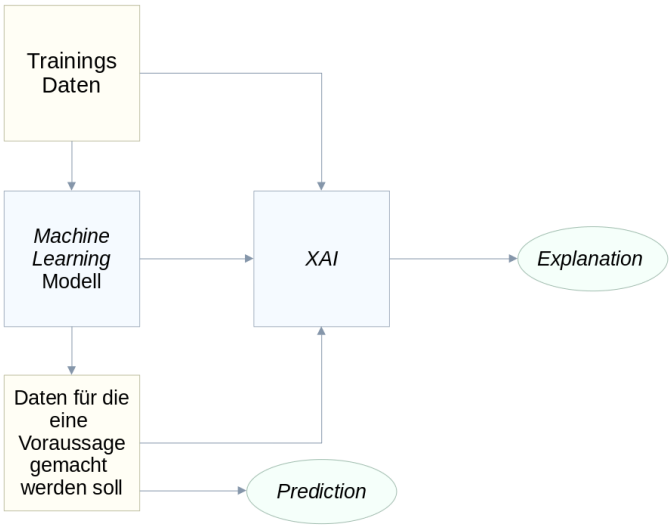

Abbildung 2.1: Schema Funktionalität *XAI*

LIME ist eine Implementierung der modellunabhängigen Methoden, bei der das *ML* Modell bereits trainiert sein muss, und wird für lokale Vorhersagen verwendet. Weitere Informationen zu LIME gibt es im nächsten Abschnitt. SHAP (SHapley Additive exPlanations) ist ebenfalls eine weitere modellunabhängige Methode, kann aber im Gegensatz zu LIME auch zur Erklärung eines gesamten Modells verwendet werden und nicht nur für einzelne lokale Vorhersagen. [Rad21]

2.2.1 LIME

LIME steht für *Local Interpretable Model-agnostic Explanations* und kann genutzt werden, um einzelne lokale Vorhersagen von *Black Box ML* Modellen zu erklären. Damit handelt es sich hier um eine modellunabhängige Methode. Das zu erklärende, nicht interpretierbare, Modell wird hierbei mit LIME ergänzt, um eine Erklärung zu generieren. [LPK20]

ML Modelle nutzen unterschiedliche Arten von Daten, wie Text, Bilder oder tabellarische Daten. Je nach Datentyp arbeitet LIME anders. Bei Text werden einzelne Wörter entfernt und wieder hinzugefügt, während Vorhersagen mit dem *ML* Modell gemacht werden. Für Bilder gilt Ähnliches, hier werden einzelne Pixel aus- und wieder angeschaltet. Wenn tabellarische Daten genutzt werden, müssen zusätzlich Trainigsdaten übergeben werden, damit können dann einzelne *Feature* geändert werden. Durch das Verändern der *Feature* kann LIME die Änderungen der Voraussage nutzen, um aufzuzeigen, wie viel Einfluss ein *Feature* auf eine bestimmte Voraussage hat. [Mol21]

Die Erklärung zu einer Vorhersage mit LIME kann aufzeigen, welches *Feature* den meisten Einfluss auf die Vorhersage einer bestimmten Klasse hat und ob *Feature* einen positiven oder negativen Einfluss darauf haben. LIME bietet hierfür auch, vom Datentyp abhängige, Visualisierungen an. Ein Beispiel mit Textdaten wird in der Abbildung 2.2 dargestellt.

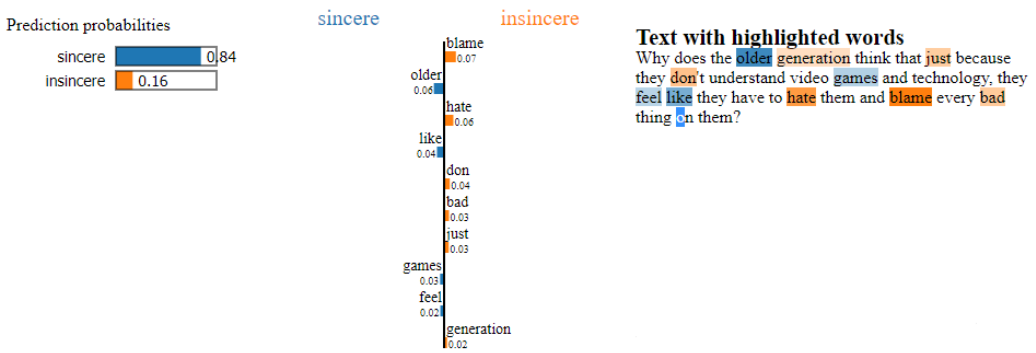

Abbildung 2.2: Visualisierung mit LIME [LPK20]

3 Entwicklung

Bei dem Visualisierungstool, welches den Namen 'QualityInsight' hat, soll es sich um eine separate Applikation handeln, die über den Webbrowser bedient werden kann. Es soll möglich sein, nach englischen Wikipedia Artikeln zu suchen und diese dann auch auswählen zu können. Weiterführend soll der Nutzer die Möglichkeit haben, eine bestimmte Revision (Version eines Artikels) auswählen zu können, um dafür die Voraussage und eine Erklärung zu erhalten. Für die Voraussage soll das *articlequality* Modell von ORES genutzt werden, wobei LIME dann als *XAI* für eine Erklärung dieser Voraussage genutzt werden soll und die Visualisierung übernimmt. Unterstützend dazu sollen die einzelnen *Feature* auch eine Beschreibung erhalten.

Die Anwendung soll mit Hilfe von *Python Flask*, für das *Backend*, und *React*, für das *Frontend*, implementiert werden. Für die spätere Evaluation soll die Anwendung dann über den Webbrowser aufrufbar sein. Das *User Interface (UI)* soll leicht zu verwenden und intuitiv sein. Bei einigen statischen Texten innerhalb der Anwendung, wie den *Feature* Beschreibungen, sollen vorbereitete JSON-Dateien genutzt werden, so können diese bei Bedarf geladen werden.

3.1 Prototyping

Wie im Abschnitt 1.3 beschrieben, werden mehrere Phasen für die Entwicklung der Anwendung durchlaufen. Diese Phasen sollten dabei helfen, die Anwendung vor der Implementation auf einen nutzerfreundlichen Stand zu bringen. Angefangen wurde mit der Betrachtung von bereits verfügbarer Lektüre. Später wurden dann die entstandenen Prototypen getestet.

3.1.1 Erste Analyse

Vor der Erstellung des ersten Prototypen, mussten erst einige wichtige Fragen geklärt werden:

- Wer wird die Anwendung nutzen?
- Was werden diese Nutzer von der Anwendung erwarten?
- Worauf muss geachtet werden, wenn man ein nutzerfreundliches Interface erstellen möchte?

Während des Designprozesses sollten immer die Wünsche und Erwartungen der Nutzer berücksichtigt werden, dafür muss geklärt werden, wer die Anwendung später nutzen wird. Im Falle meiner Anwendung kann es Nutzer geben, die sich nicht mit *ML* auskennen und denen ORES vielleicht auch unbekannt ist. Das bedeutet, dass es wahrscheinlich nicht reicht, einfach nur eine *Prediction Explanation* zu liefern. Die Anwendung sollte also auch einige Informationen zu ORES und dem genutzten *articlequality* Modell zur Verfügung stellen. Neueren Editoren könnte es an Erfahrungen mit der englischen Wikipedia mangeln, daher sollten einige Begriffe, die sich darauf beziehen, innerhalb der Anwendung kurz erklärt werden.

Im Paper 'Keeping Community in the Loop: Understanding Wikipedia Stakeholder Values for Machine Learning-Based Systems' [SYS+20] wurden die *Stakeholder* von Wikipedia und deren Werte genauer analysiert, was dieser Arbeit einige gute Anhaltspunkte geliefert hat:

Ausgaben von Algorithmen sollten replizierbar sein.
Innerhalb der Anwendung soll der Nutzer bestimmte Revisionen auswählen können. Für die einzelnen Revisionen ändern sich die Vorhersagen des *articlequality* Modells nicht, daher sind diese immer wieder replizierbar.

Existierende Algorithmische Systeme, wie ORES, sollten zum Ausprobieren und Experimentieren zur Verfügung stehen.
Meine Anwendung soll dies zum Teil möglich machen. Man soll sich von den unterschiedlichsten Artikel-Revisionen die Vorhersagen, inklusive Erklärung, ausgeben lassen können.

Anwendungen sollen Arbeitsabläufe unterstützen und dabei helfen die Ziele der Nutzer zu erreichen.
Durch die Erklärung der Vorhersage sollen Editoren dabei unterstützt werden, Artikel verbessern zu können. Das angestrebte intuitive Design soll dies leichter machen und den Nutzer nicht behindern.

Systeme oder Anwendungen sollten transparente Erklärungen ihres Verhaltens zur Verfügung stellen.
Die Visualisierung und *Explanation* mit LIME sollte die Voraussagen von ORES *articlequality* Modell transparenter machen. Die zusätzlichen Beschreibungen der genutzten *Feature* sollen dies weiter unterstützen.

Systeme oder Anwendungen sollten Wege vorschlagen, die dem Nutzer helfen, zu lernen und bessere Beiträge für die Community zu erstellen.
Die Anwendung kann, entsprechend der Voraussage, Vorschläge machen, wie der Nutzer einen Artikel verbessern könnte.

Damit wäre nun geklärt, was die Anwendung aus Nutzersicht berücksichtigen sollte und es können die allgemeinen Regeln für ein nutzerfreundliches Design

betrachtet werden. Dies ist auch wichtig, da das Design der UI die Nutzungserfahrung stark beeinflussen kann. Hierfür wurde das Buch 'Laws of UX' [Yab20] genutzt und überlegt, welche dieser Regeln wie angewendet werden können:

Nutzer schließen von Ihnen bekannten Anwendungen, auf andere ähnlich wirkende Anwendungen.

Das Design meiner Anwendung sollte an das von Wikipedia anlehnen, damit mögliche Erwartungen erfüllt werden. Anfangen kann man bei der Farbpalette der Anwendung, diese sollte den Farben, die Wikipedia nutzt, ähneln.

Nutzer können sich besser auf ihre Aufgabe konzentrieren, wenn die Anwendung ein bereits bekanntes Mentales Modell[18] nutzt.

Da in der Anwendung nach Artikeln gesucht werden soll, sollte die Suche ähnlich wie bei Wikipedia gestaltet werden. So muss sich der Nutzer an nichts Neues gewöhnen und sollte damit einen guten Einstieg in die Anwendung bieten.

Wenn dem Nutzer eine Anwendung vertraut vorkommt, hilft ihnen dies beim Verstehen, wie die Anwendung genutzt werden kann.

Editoren sollten durch ihre Arbeit auf Wikipedia gewiss damit vertraut sein, daher sollte meine Anwendung ihnen das Gefühl geben, ähnlich zu sein.

Um Nutzungsprobleme zu verhindern, sollten Seitenlayouts, Navigation und Platzierungen von Elementen von bekannten Anwendungen übernommen werden.

Die Positionierung von einigen Elementen auf Wikipedia sollten in meiner Anwendung ähnlich sein. Auch, wenn dies nicht sehr innovativ ist, hilft es Nutzern mehr, da sie die Anwendung dann ohne große Probleme und viel Überlegung nutzen können. Die Anwendung soll sich schließlich auf die Funktionalität und nutzerfreundlichen Interaktionen konzentrieren.

Auswahlmöglichkeiten sollten minimiert werden, um Entscheidungen zu beschleunigen.

Dies sollte bei der Suche von Artikeln und der Anzahl an angezeigten Revisionen berücksichtigt werden. Artikel können eine Vielzahl an Revisionen haben und sollte daher eingeschränkt werden.

Seiteninhalte und Informationen sollten dem Nutzer in kleinen, gut organisierten Teilen präsentiert werden.

Die Erklärungen der Vorhersagen sollten gut organisiert werden. Informationen sollten nicht nur an einer Stelle gesammelt stehen, sondern in kleinere Teile zerlegt werden.

Die wichtigsten und letzten Momente, die in der Anwendung verbracht werden, sollten besonders beachtet werden.

Die Vorhersage und Erklärung sind die wichtigsten Aspekte meiner Anwendung, daher sollten sich hier viele Gedanken gemacht werden und sollten das primäre Ziel der späteren *Usability Tests* sein.

Alle Prozesse haben eine Grundkomplexität, die nicht beseitigt werden kann.

Bei der Erklärung der Vorhersage, die LIME liefert, wird es nicht möglich sein, viel zu vereinfachen. In diesem Fall könnte es auch Nachteile haben, dies zu sehr zu vereinfachen.

Weiterhin sollte auch beachtet werden, dass Buttons, Formulare und andere Elemente groß genug für die Interaktion sind und dass diese genug Abstand voneinander haben. Komplexe Aufgaben sollten in kleinere Schritte aufgeteilt werden, um den Nutzer nicht zu überfordern. Der Nutzer sollte bei jeder Interaktion und Änderung eine Reaktion von der Anwendung erhalten, darunter zählen auch Ladeanimationen. Generell kann ein gut aussehendes Design auch dafür sorgen, dass Nutzer die Funktionalität der Anwendung für besser empfinden. [Yab20]

3.1.2 Erster Prototyp

Nachdem ersichtlich war, worauf bei dem Design des UIs geachtet werden sollte, wurde der erste Prototyp mit Figma[19] erstellt. Um die Gestaltung einfach zu machen, nutzte ich Komponenten des Material Designs[20]. Dies hat auch den Vorteil, dass es ein weit verbreitetes Design ist und der Nutzer daher damit vertraut sein sollte.

Ich hatte mich dazu entschieden, ein paar Informationen zu ORES und dem *articlequality* Modell auf der ersten Seite zur Verfügung zu stellen, damit auch Nutzer ohne vorhergehendes Wissen verstehen, worauf die Anwendung basiert (Abbildung 3.1). Für den ersten Prototypen hatte ich hier erst nur Platzhaltertext genutzt und keine echten Informationen.

[18] Ein *Mental Model* beschreibt wie eine Anwendung vom Nutzer verstanden wird und wie diese funktioniert. [Yab20]

[19] `https://www.figma.com`, besucht am 10. August 2021

[20] `https://material.io/design`, besucht am 10. August 2021

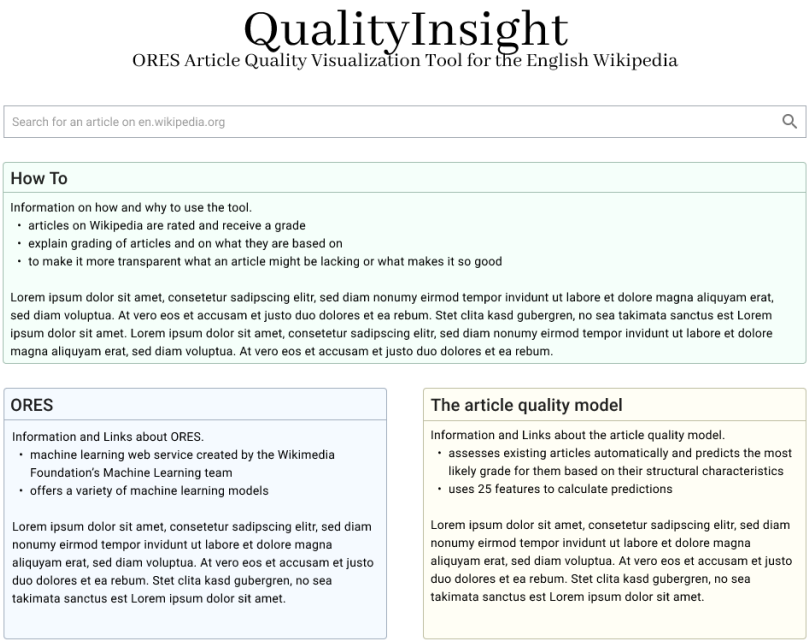

Abbildung 3.1: Startseite des ersten Prototypen

Bei der Suche nach einem Artikel hatte ich mich dazu entschieden, eine Verlinkung zu den jeweiligen Artikeln vorzunehmen. Diese Links baute ich in die Artikelnamen ein (Abbildung 3.2). Bei den Revisionen hatte ich die Revisions ID, den Editor, den Zeitstempel und den Kommentar zur Revision eingefügt gehabt (Abbildung 3.3).

Abbildung 3.2: Artikelauswahl im ersten Prototypen

Abbildung 3.3: Revisionensauswahl im ersten Prototypen

Für Nutzer, die keine Erfahrungen mit *ML* oder Wikipedia haben, hatte ich noch einige generelle Informationen über Revisionen, Vorhersagen und *Feature* dort in die Anwendung eingebaut, wo sie sich inhaltlich eignen. Diese können über '?'-Symbole aufgerufen werden.

Die Seite für die *Prediction Explanation* beinhaltet drei, von einander visuell getrennte, Hauptelemente:

- Eine gelbe Karte mit der Visualisierung, die die *Prediction probablities*, die *Feature* Werte in einer Tabelle und den *Explanation* Graphen beinhaltet. (Links in der Abbildung 3.4)
- Eine rosa Karte mit den *Feature*-Erklärungen. Dort kann über ein *Select*-Feld das gewünschte *Feature* ausgewählt werden. (Rechts oben in der Abbildung 3.4)
- Eine grüne Karte mit Tipps zu Verbesserung des Artikels. (Rechts unten in der Abbildung 3.4)

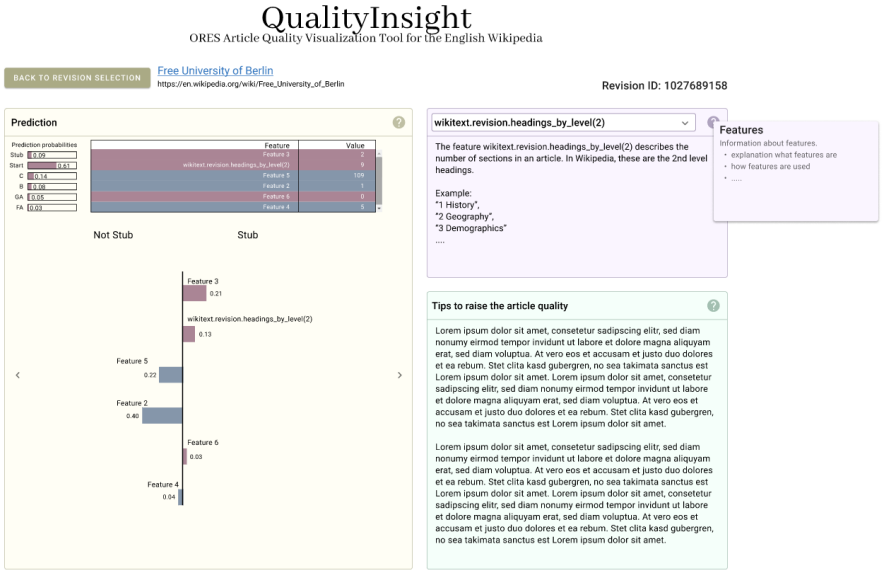

Abbildung 3.4: Seite für die Vorhersage Erklärung des ersten Prototypen

Bei diesem Prototypen entschied ich mich dafür, die Graphen mit Pfeilen wechseln zu können, denn die Graphen alle gleichzeitig anzeigen zu lassen, könnte die Anwendung unübersichtlich machen und möglicherweise auch den Nutzer abschrecken.

3.1.3 Erste Evaluation

Für die Evaluation des ersten Prototypen hatte ich zwei Teilnehmer, mit denen ich mich über Jitsi[21] für einen synchronen *Usability Test* traf. Mit Hilfe einer Bildschirmübertragung konnte ich die Interaktion mit dem Prototypen nachverfolgen. Während des Tests bekamen die Teilnehmer Aufgaben, die sie innerhalb der Anwendung erledigen mussten. Dabei sollten sie ihre Überlegungen und Eindrücke verbal ausformulieren. In Tabelle 3.1 sind die Vorkenntnisse der Teilnehmer und deren Arbeitsfeld eingetragen, nach denen ich vor dem Test fragte, um ihre späteren Anmerkungen besser einordnen zu können.

Die Personen haben vor dem Test eine Einwilligungserklärung zugeschickt bekommen, die sie unterschreiben konnten (siehe Abschnitt 6.2).

[21] `https://meet.jit.si/`, besucht am 10. August 2021

Tabelle 3.1: Informationen der Teilnehmer der ersten Evaluation

	T1	**T2**
Arbeitsfeld	Softwareentwickler im Web Frontend	Wissenschaftlicher Mitarbeiter im Bereich Human-Centered Computing
Erfahrungen mit Wikipedia	Hatte einige Artikel auf der englischen Wikipedia-Seite editiert, aber keine Neuen selber erstellt. Meistens hatte T1 grammatikalische Fehler behoben.	Hatte bis jetzt keine Artikel auf Wikipedia bearbeitet.
Erfahrungen mit *ML*	Besitzt etwas Wissen über *ML* durch einen *ML* Kurs an der Universität. Wusste auch vor dem Test, was ORES ist.	Hat Erfahrung mit *ML* und kannte ORES vom Namen her.

Ergebnisse

Beide Teilnehmer merkten gleich an, dass sie das Design an Wikipedia erinnerte und erkannten die Suchleiste auf der Startseite und ihre Funktion, nach Artikeln suchen zu können. Die Informationstexte wurden als etwas zu lang angesehen, so dass beide Teilnehmer das Gefühl hatten, zu viel lesen zu müssen.

Bei der Auswahl eines Artikels hatte T1 zur Auswahl gleich auf den nebenstehenden 'Select'-Button geklickt. T2 ging davon aus, dass der verlinkte Artikelname auch als Auswahl gilt, was aber nur den Artikel in einem anderen Tab öffnete. Dass man die Möglichkeit hatte, die Artikel öffnen zu können, empfand T2 für praktisch, nur sollte dies offensichtlicher gemacht werden. Die Revisions-Auswahl war etwas eindeutiger für die Teilnehmer, hierbei fiel es T1 nicht schwer, die neuste Revision zu wählen, da diese nach Datum sortiert waren. Der '?'-Button wurde von T2 als hilfreich empfunden, wobei die Position, die sehr nah am Zurück-Button war, bemängelt wurde.

Auf der Seite für die *Prediction Explanation* gab es einige Unklarheiten und Verbesserungsvorschläge. T1 war anfangs nicht ganz klar, dass diese Vorhersage für nur eine Revision war und meinte, dies sollte deutlicher gekennzeichnet werden. Beide Teilnehmer empfanden die *Feature*-Namen als zu kryptisch, so dass man schwer auf Anhieb ablesen konnte, wofür sie stehen. Das Element für die *Feature*-Erklärungen hatte keine Überschrift, was dazu führte, dass T1

nicht gleich erkannte, wofür die Auswahl dort war. T2 schlug an der selben Stelle auch vor, eine Überschrift für dieses Element hinzuzufügen und wünschte sich eine aussagekräftigere Überschrift für das Vorhersage-Element. Sonst fand T2 die Erklärungen zu den *Features* hilfreich. Für die Vorhersage wünschten sich die Teilnehmer mehr Erklärungen, um sie zu verstehen, besonders was die Bedeutung der Noten und *Feature* anging. Es war für T1 auch anfangs nicht ganz klar, wofür die Vorhersage Wahrscheinlichkeiten standen und was es mit der Tabelle mit den *Feature*-Werten auf sich hat. Die Aussage des Graphen war für beide Teilnehmer nicht klar und wünschten sich hierbei mehr Informationen. Besonders T1 hatte das Gefühl, dass man viel Expertenwissen benötigt, um die Diagramme und Tabellen zu verstehen. T2 hatte zu dem Design noch angemerkt, dass die Tabelle für die Vorhersage-Wahrscheinlichkeiten zu klein war, sowie dass die Pfeile, die zum Wechseln des Graphen da waren, nicht gut erkennbar waren. Die Farben der Tabellen und der Graphen empfand T2 als unpassend.

Allgemein empfanden die Teilnehmer die Navigation als gelungen, was für sie die Suche nach Artikeln und Revisionen leicht machte. T1 empfand die Zurück-Button als hilfreich und meinte, es wäre gut sichtbar, womit man in der Anwendung agieren kann.

3.1.4 Zweite Analyse und Änderungen am Prototypen

Die Anmerkungen und Ideen aus dem ersten *Usability Test* gaben eine gute Übersicht darüber, was geändert werden musste und wie der Prototyp verbessert werden kann. Zum einen hatte ich die Idee, dass man Pop-up Texte einfügen könnte, die einem mehr Informationen beim darüber hovern über bestimmte Elemente geben kann. Damit wäre es möglich, Tabellen und Graphen mit Zusatzinformationen zu ergänzen, ohne dass zu viele Informationen auf einmal angezeigt werden. Informationstexte sollten dabei nicht zu lang sein, dies müsste dann auch bei der Startseite beachtet werden, ansonsten kann es passieren, dass Nutzer die Texte als zu überwältigend ansehen.

Im neuen Prototypen hatte ich daher nun kleine Infoboxen, die beim hovern über ein Element auftauchen, eingefügt. Sie wurden für die Graphen, Tabellen und einige Buttons verwendet. Ich habe versucht, die Informationstexte auf die wichtigsten Punkte herunterzubrechen.

Bei der Darstellung der Graphen hatte man die Möglichkeit einfach alle Graphen anzeigen zu lassen oder diese individuell mit Tabs aufrufbar zu machen. Ich entschied mich für die Tabs, da, wie schon in der ersten Analyse (Abschnitt 3.1.1) bemerkt, zu viele Informationen und Elemente auf einer Seite zu einer unangenehmen Nutzererfahrung führen können. Durch die Tabs, die nach den Noten gelabelt sind, hätten die Nutzer nun auch die Möglichkeit, gezielter den Graph für eine bestimmte Note auswählen zu können (Abbildung 3.5). Die

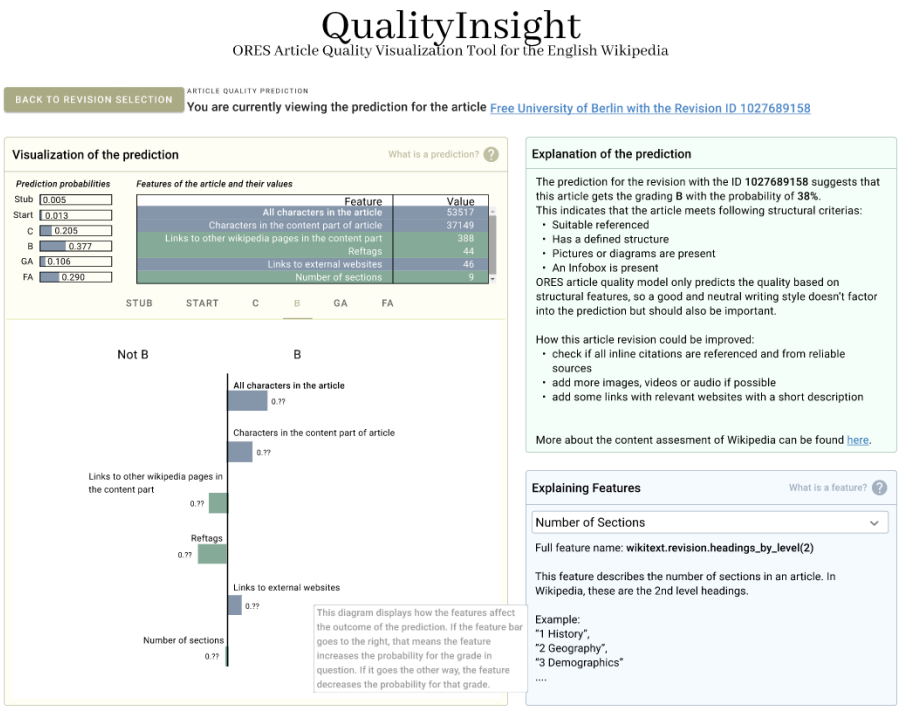

Abbildung 3.5: Seite für die Vorhersage Erklärung des zweiten Prototypen inklusive Hover-Element

Farben und die Größe des Graphen und der Tabellen hatte ich ebenfalls aktualisiert gehabt, damit das Design insgesamt stimmiger wirkt und alles gut erkennbar ist.

In der Evaluation kam heraus, dass die Elemente auf der Seite für die Vorhersage alle eindeutigere Überschriften erhalten sollten, so dass der Nutzer gleich weiß, was dort angezeigt wird. Dies gilt sowohl für die Karten, als auch für die Tabellen, die dargestellt sind. Daher erhielten die Karten neue Überschriften, die etwas ausführlicher waren. Auch die Karte für die Erklärung von *Features* bekam eine Überschrift, das Auswahlfeld verschob sich dadurch etwas weiter nach unten. Ebenfalls änderte ich die Darstellung des Artikelnamens und deren Revisions ID, damit es deutlicher wird, dass die Vorhersage für diese bestimmte Revision ist.

Mir fiel auf, dass der Inhalt des '?'-Symbols vor dem Auswählen nicht eindeutig war, daher fügte ich hier Text hinzu. Dieser sollte beschreiben, was dort bei Knopfdruck erklärt wird. Auf der Seite mit den Revisionen verschob ich das '?'-Symbol, damit dieses nicht mehr zu nah an dem Zurück-Button liegt (Abbildung 3.6).

Um *Feature* auch ohne eine extra Erklärung verständlicher zu machen, war es eindeutig, dass die Namen selbsterklärend sein müssen. Daher tauschte ich die

Abbildung 3.6: Ausschnitt der Revisionsauswahl im zweiten Prototyp

originalen Namen mit eigenen, simpleren Name aus. Die originalen *Feature*-Namen schrieb ich trotzdem noch zu der *Feature*-Erklärung, falls der Nutzer sich dafür interessieren sollte.

Da die Position der Artikel-Links bei der Auswahl nicht ideal war, fügte ich ein 'Open In New Tab'-Symbol nach dem Artikel-URL ein, der den Artikel in einem neuen Tab öffnet (Abbildung 3.7).

Abbildung 3.7: Ausschnitt der Artikelauswahl im zweiten Prototyp

Nach eigener Überlegung entschied ich mich auch dafür, die Karte für die Erklärung der *Feature* mit der für die Tipps zu tauschen. Dabei änderte sich auch der Inhalt der letzteren Karte etwas: Eine Erklärung zu der Vorhersage wird hier nun in Text ausgeschrieben, dazu gibt es noch eine kurze Erklärung, was es bedeutet, dass der Artikel diese Note erreicht hat und allgemeine Tipps, wie die Qualität gesteigert werden kann. Der Grund für diesen Tausch ist, dass die Erklärung der Vorhersage in diesem Teil wichtiger und individuell ist, es soll dem Nutzer also gleich auffallen. Die Farbe für das *Feature*-Element hatte ich ebenfalls zu einem Blau geändert.

3.1.5 Zweite Evaluation

Für die Evaluation des zweiten Prototypen hatte ich ebenfalls zwei Teilnehmer für einen synchronen *Usability Test* finden können. Wie im ersten Test gab ich den Teilnehmern Aufgaben. Der Fokus lag diesmal mehr auf das Verständnis der Vorhersage und Erklärung, daher wurde in diesem Test mehr über *Prediction Explanation* Seite geredet.

Die Personen haben vor dem Test eine Einwilligungserklärung zugeschickt bekommen, die sie unterschreiben konnten (siehe Abschnitt 6.2). In Tabelle 3.2 sind die Vorkenntnisse der Teilnehmer und deren Arbeitsfeld eingetragen.

Tabelle 3.2: Informationen der Teilnehmer der zweiten Evaluation

	T3	**T4**
Arbeitsfeld	Arbeitet mit Wikidata und ist spezialisiert auf Diversität und Bots	Masterstudent für Informatik mit dem Schwerpunkt auf *AIs* und *Human Computer Interaction*
Erfahrungen mit Wikipedia	Hat Editierungen auf Wikidata vorgenommen, aber nicht auf der englischen Wikipedia.	Hatte versucht Artikel auf Wikipedia zu bearbeiten.
Erfahrungen mit *ML*	Ist mit ORES vertraut, da ORES die Diversität beeinflussen kann.	Hat Vorwissen zu *ML* und kannte ORES durch ein Seminar an der Universität.

Ergebnisse

Bei der Navigation durch die Anwendung gab es bei beiden Teilnehmer keinerlei Probleme, die Suchleiste wurde wie beim letzten Mal gleich erkannt und auch die Auswahl eines Artikels und einer Revision stellten keine Probleme dar. Da bei der Artikelauswahl der Link zu dem Artikel ein eigenes Symbol bekam, gab es diesmal keine Verwechselung und T4 erkannte gleich, dass dort der Artikel geöffnet werden kann.

Die Startseite wurde von T3 als informativ bezeichnet und T4 empfand sie als gut strukturiert. T4 hatte angemerkt, dass es deutlicher sein sollte, dass das *articlequality* Modell Bestandteil von ORES ist und dass es eines von vielen Modellen ist.

Ähnlich wie bei der ersten Evaluation nutzte T3 die Zeitstempel, um die neuste Revision zu suchen, dabei machte die Formatierung des Zeitstempels ein paar Probleme. T4 wusste durch den '?'-Button, wie die Revisionen sortiert waren und fand es gut, dass Revisionen erklärt wurden, da hier keine Vorkenntnisse vorlagen. Generell fand T4 die Informationen, die man durch das Klicken von Buttons oder vom Hovern über Elemente bekam, hilfreich.

Welche Note vorhergesagt wurde, erkannte T3 gleich. Wie die Noten gewertet werden, war beiden Teilnehmern nicht ganz klar, sie nahmen aber richtigerweise an, dass FA die beste Note war. Auch die Interpretation des Graphen war für die Teilnehmer etwas schwer. T3 meinte, dass eine Erklärung dazu fehlt, warum ein *Feature* nach rechts oder links im Graphen ausschlägt. Beide Teilnehmer hätten es auch hilfreich gefunden, wenn zu den *Feature* dazu gestanden hätte, ob diese besser einen niedrigen oder hohen Wert haben sollten.

Die selbst definierten *Feature*-Namen haben bei dem Test gut funktioniert. T4 fand, dass, bis auf ein paar Ausnahmen, die meisten *Feature* dadurch gleich verständlich waren und dass es gut war, die originalen *Feature*-Namen mit in die Beschreibung zu schreiben. Die Beispiele zu den *Features* wurden auch positiv aufgenommen. Die Erklärung der Vorhersage war für beide Teilnehmer gut verständlich, auch welche Kriterien der Artikel bereits trifft. Die Auswahl der Graphen fiel dieses Mal um einiges leichter, die Tabs wurden von beiden Teilnehmer intuitiv zum Wechseln benutzt.

Generell fand T3, dass es mehr Informationen zu ORES geben sollte, da Nutzer ohne Vorwissen sonst Verständnisprobleme bekommen könnten. T4 wünschte sich noch mehr Informationen dazu, warum ein *Feature* eine Vorhersage positiv oder negativ beeinflusst.

3.1.6 Dritte Analyse und Änderungen am Prototypen

An der Art der Navigation in der Anwendung wurde nichts geändert, da diese ganz gut bei beiden Evaluationen funktioniert hatte. Die Strukturierung der Seiten wurde ebenfalls gut aufgenommen, daran wurde also auch nichts mehr geändert. Informationen durch ein Hovern über ein Element zu bekommen, kam gut an, nur der Inhalt der Informationen wurde erweitert, um die Graphen und Noten besser zu erklären. Auch in den Infoboxen der Startseite wurden neue Informationen hinzugefügt. Dadurch, dass die Informationen in kleinere Teile auf den Seiten verteilt wurden, waren die Teilnehmer eher motiviert zu lesen, dies wurde also versucht beizubehalten. Tabs schienen bei der Auswahl der Graphen eine gute Lösung gewesen zu sein, sowie auch die Umbenennung der *Feature*.

Die *Timestamps* bei den Revisionen waren nicht so gut lesbar für die Nutzer, daher änderte ich hier die Formatierung, so sollte es einfacher sein zu erkennen, welche Revision die neuste ist.

Es wurde sich gewünscht, dass erklärt wird, welcher Wert für die *Feature* am idealsten wäre. Dies hatte ich nicht mit in die Änderungen aufgenommen, da es zeitlich nicht mehr gepasst hätte, für jedes einzelne *Feature* die besten Werte zu ermitteln. Um das herauszufinden, hätte man entweder mit Hilfe der ORES API und der Manipulation von Werten einer Vorhersage untersuchen können,

wie sich die Vorhersage bei einer Manipulation eines bestimmten *Feature* ändert oder man hätte dazu einen Algorithmus schreiben können, der dies für einen übernimmt.

Im Grunde wurde das meiste am Inhalt der Anwendung geändert, die Struktur und Navigation blieb die gleiche.

3.2 Ergebnisse

Während dieser Iterationen fiel mir am Ende auf, dass es sich als schwer herausstellte, die Graphen auf eine simple Art und Weise erklären zu können. Die Evaluationen waren weitreichend hilfreich, um zu ermitteln, wo noch Arbeit nötig war und was bereits gut funktionierte.

Grundlegend zeigte sich, dass die Navigation durch die Anwendung nach der Behebung einiger kleiner Störfaktoren, wie der Verlinkung der Artikel, nach der zweiten Evaluation schon nicht mehr geändert werden musste. Die erste Analyse hatte dabei den meisten Einfluss genommen.

3.3 Implementierung

Um die Anwendung umzusetzen, wurde Python für das *Backend* und React für das *Frontend* genutzt.

3.3.1 Backend

Folgende grundlegende Python Pakete wurden verwendet:

flask wird für die Kommunikation zwischen dem Backend und dem Frontend verwendet. Primär werden Daten aus Berechnungen an das Frontend weitergeleitet.

json wird zum Auslesen von JSON-Dateien genutzt.

datetime wird benötigt, um die *Timestamps* der Revisionen formatieren zu können.

requests kann dazu verwendet werden, Anfragen an bestimmte Wikipedia-Seiten zu senden. Im Falle der Anwendung wird hiermit die Suche nach Artikeln und die Ausgabe von Artikel Revisionen implementiert.

lime wurde im Abschnitt 2.2.1 beschrieben und wird hier für die *Explanation* der Vorhersagen verwendet. Die *Explanation* lässt sich über LIME auch als Graph, für einzelne Klassen, ausgeben.

matplotlib wird für die Anpassung der LIME Graphen benötigt, damit sich diese besser in die Anwendung integrieren.

base64 wird verwendet, um die Graphen in Strings zu enkodieren, da diese dann direkt im Frontend eingebunden werden können.

pandas wird nur verwendet, um eine csv-Datei auszulesen. Diese enthält die Trainingsdaten des *articlequality* Modells der englischen Wikipedia, die LIME für die *Explanation* benötigt.

numpy Die tabellarischen Daten müssen LIME als Numpy-Array übergeben werden. Daher wird dieses Modul zur Umwandlung genutzt.

revscoring wird verwendet, um das *articlequality* Modell zu laden und dieses wird dann für den Erhalt von Vorhersagen benutzt.

mwapi wird benötigt, um eine Verbindung zur MediaWiki API herzustellen. Diese Verbindung wird für revscoring benötigt.

In Listing 6.3 befinden sich alle Funktionen, die sich um die Suche nach Artikeln und Revisionen kümmern. Für die Suche nach Artikeln muss man den Suchbegriff an die Funktion `searchArticle` übergeben. Es wird eine Liste mit zehn Objekten zurückgeben, die Objekte beinhalten die Artikelnamen, den Artikel-URL und einen Schlüsselwert. Die Menge der Objekte wurde auf zehn begrenzt, da es wahrscheinlich ist, dass der gesuchte Artikel bereits bei den vordersten Ergebnissen mit dabei ist. Ansonsten kann der Nutzer seine Suchanfrage auch genauer eingeben. Um die Revisionen eines Artikels zu erhalten, kann die Funktion `getArticleVersions` genutzt werden. Der Artikelname wird benötigt und es wird ein Objekt, dass die zehn neusten Revisionen inklusive *Timestamp*, Editor, Kommentar und ID beinhaltet, zurückgegeben.

Für die *Explanation* und Visualisierung mit LIME werden Trainingsdaten des *articlequality* Modells benötigt. Diese wurden mit dem Code in Listing 6.1 vorbereitet. Datensätze[22] mit Revision IDs für die Trainigsdaten wurden der Github-Seite des *articlequality* Modells entnommen. Für diese IDs wurden die *Feature*-Werte ausgelesen und dann in mehrere csv-Dateien geschrieben. Diese kleineren csv-Dateien wurden später zu einer großen csv-Datei zusammengefasst.

Listing 6.4 beinhaltet die Funktion `getVersionScore`, was die Revisions ID benötigt und für diese dann die Vorhersage ausführt. Die Funktion gibt die Vorhersage mit den *Prediction probablities*, den *Feature*-Werten und den originalen *Feature*-Namen zurück. Um eine *Explanation* einer Vorhersage zu erhalten, wird die Funktion `getExplanation` genutzt. Diese benötigt die Vorhersage und die *Feature*-Werte. Es wird der `LimeTabularExplainer` verwendet, diesem übergibt man die vorbereiteten Trainingsdaten, die Namen der *Feature* in einer

[22] `https://github.com/wikimedia/articlequality`, besucht am: 05. August 2021

Liste und die Bezeichnung der Klassen in einer weiteren Liste. Da für die *Explanation* die alternativen *Feature*-Namen genutzt werden sollen, wurden diese übergeben. Um nun eine *Explanation* zu erhalten, muss noch eine zusätzlich Funktion `score` angegeben werden. Das Ergebnis dieser Funktion muss für LIME lesbar sein, daher kann nicht einfach die `score`-Funktion von *revscoring* genutzt werden. Es wurde daher eine eigene `score`-Funktion geschrieben, welche aber die von *revscoring* nutzt. Sonst werden noch die *Feature*-Werte der Vorhersage benötigt. Für die *Explanation* hat man noch die Möglichkeit, die Anzahl der Samples anzugeben. Diese hatte ich auf 500 gesetzt, andernfalls hätte die Berechnung zu lange dauert. Idealer wäre es, eine größere Sample-Anzahl zu nutzen, da bei nur 500 Samples die *Explanation*-Ausgabe und der daraus resultierende Graph bei einer Wiederholung der Berechnung deutlich anders aussehen kann. Dies liegt daran, dass LIME nur eine zufällige Menge der Trainingsdaten nutzt. Nachdem LIME mit der Berechnung fertig ist, kann ein Graph für jede Note erstellt werden. Für die spätere Nutzung im Frontend werden die Graphen noch in Strings enkodiert. Die Funktion `getVersionScore` übergibt dann zuletzt die enkodierten Graphen in einer Liste.

Damit diese ganzen Daten zum Frontend gelangen, wird eine Flask Applikation erstellt (Listing 6.2). Je nachdem, welche Route aufgerufen wird, werden die nötigen, vorher besprochenen, Funktionen aufgerufen. Über `return` werden dann JSON-Objekte an das Frontend übergeben.

3.3.2 Frontend

Für die *UI* wurden primär Komponenten der Material UI Bibliothek[23] genutzt und angepasst. Als Beispiel ist in Listing 6.5 die Implementierung der Startseite abgebildet. Mit Hilfe der *axios*-Bibliothek konnten Anfragen vereinfacht erstellt werden. Entscheidungen des Nutzers werden über die URL-Parameter angezeigt, diese können auch leicht vom Backend ausgelesen und als Parameter für die Funktionen eingesetzt werden.

Auf der Startseite (Abbildung 3.8) liegt die einzige Interaktionsmöglichkeit auf der Suchleiste. Der Rest der Seite stellt nur Informationen zur Anwendung, ORES, dem *articlequality* Modell und einige damit zusammenhängende Links zur Verfügung.

[23] `https://material-ui.com/`, besucht am: 06. August 2021

Abbildung 3.8: Startseite der Anwendung

Nachdem die Eingabe in der Suchleiste mit einem 'Enter' oder dem 'Lupen'-Symbol bestätigt wird, werden einem maximal zehn dazu passende Artikel angezeigt (Abbildung 3.9). Diese kann man durch einen Klick auf den 'Select'-Button auswählen oder über das Symbol neben dem URL in einem neuen Tab öffnen lassen. Beim Hovern über den 'Select'-Button taucht eine Information auf.

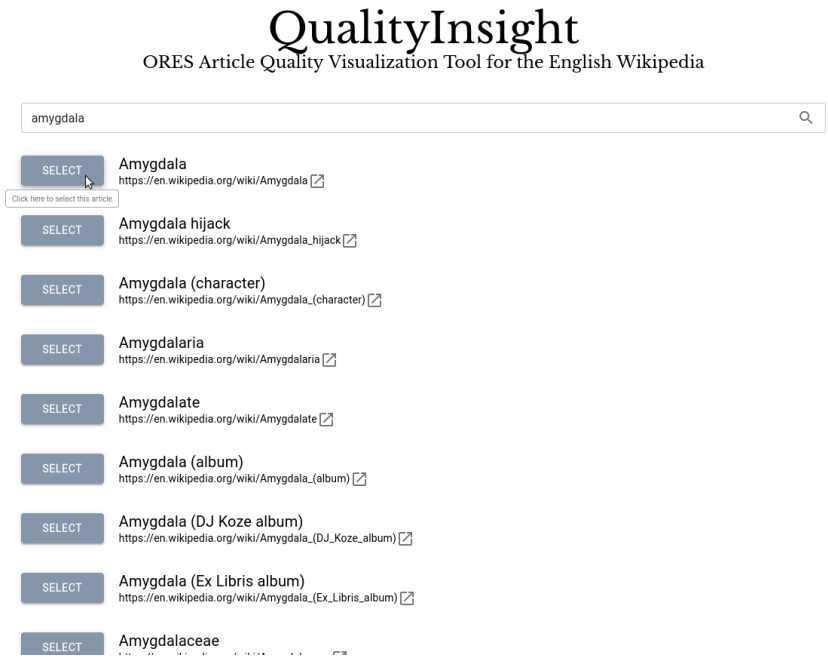

Abbildung 3.9: Artikelauswahl für den Suchbegriff 'amygdala'

Wenn ein Artikel ausgewählt wurde, werden dem Nutzer die zehn neusten Revisionen angezeigt (Abbildung 3.10). Diese sind von der neusten bis zur ältesten Revision sortiert. Zu den Revisionen wird die ID, der Editor, der *Timestamp* und der Kommentar angezeigt. Falls nicht bekannt sein sollte, was eine Revision ist, kann durch das Anklicken des '?'-Symbols eine Information dazu aufgerufen werden.

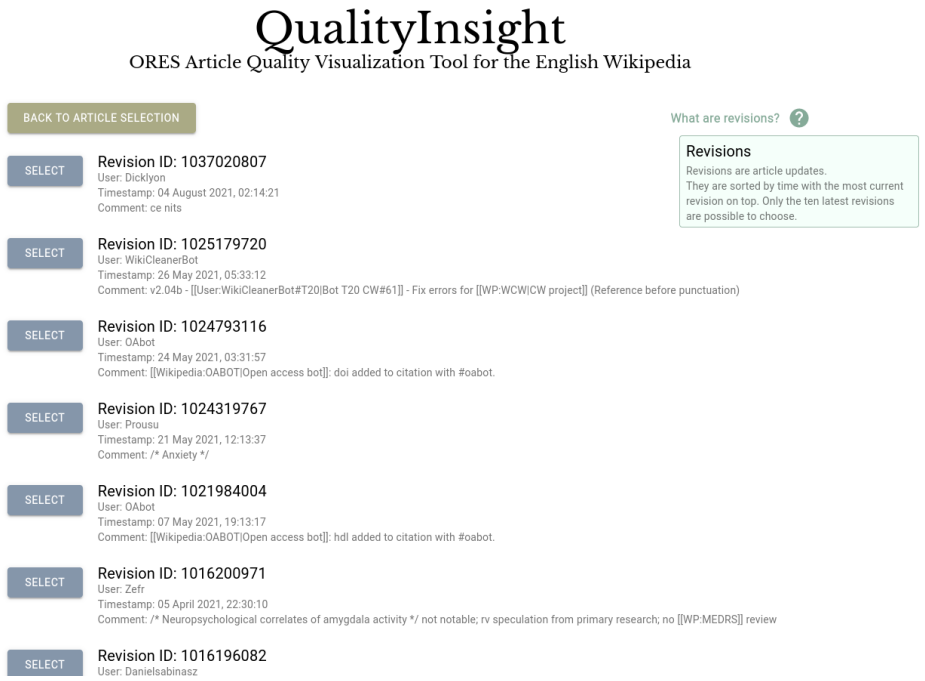

Abbildung 3.10: Die zehn neusten Revisionen des Artikels 'Amygdala'

Nach der Auswahl einer Revision wird die Berechnung der Vorhersage und der *Explanation* gestartet. Da dies einiges an Zeit braucht, wurden Lade-Animationen eingebaut, um dem Nutzer deutlich zu machen, dass die Seite arbeitet und nicht hängen geblieben ist. Wenn die Berechnung abgeschlossen wurde, laden sich die Tabellen, die Graphen und Erklärung zur Vorhersage automatisch. Es gibt drei Arten von Karten, die unterschiedliche Informationen beinhalten (Abbildung 3.11). Zum einen gibt es die Karte 'Visualization of the prediction', in der eine Anzeige zu den *Prediction probabilities*, eine Tabelle zu den *Feature*-Werten und die Graphen für jede Note, die mit LIME erstellt wurden, liegen. Bei jedem dieser drei Elemente können mehr Informationen durch ein Hovern über sie erhalten werden (Abbildung 3.12).

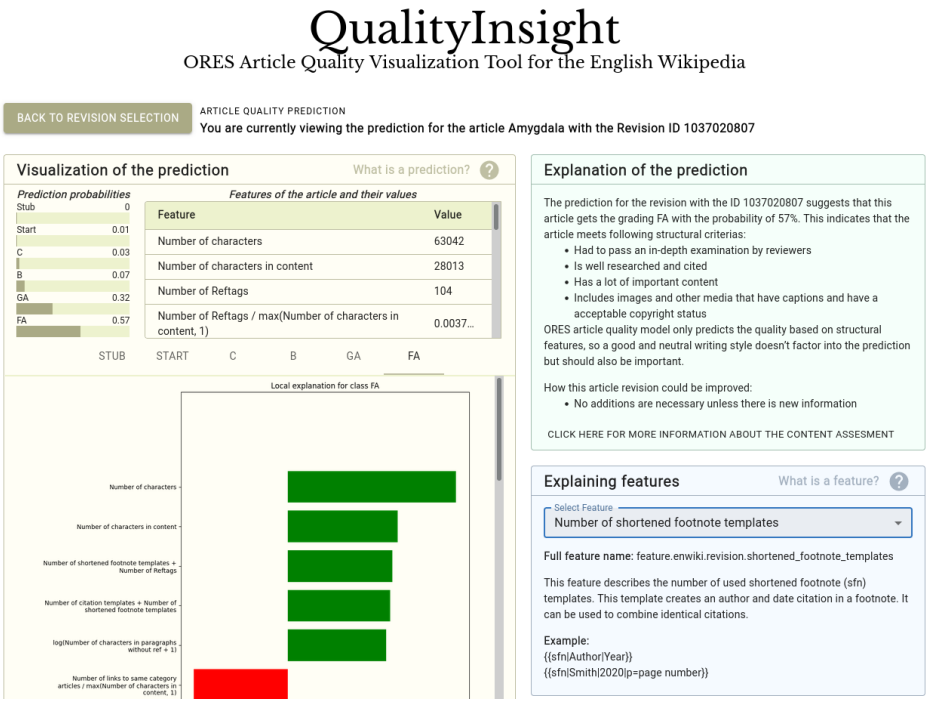

Abbildung 3.11: *Explanation* der Revision 1037020807 des Artikels 'Amygdala'

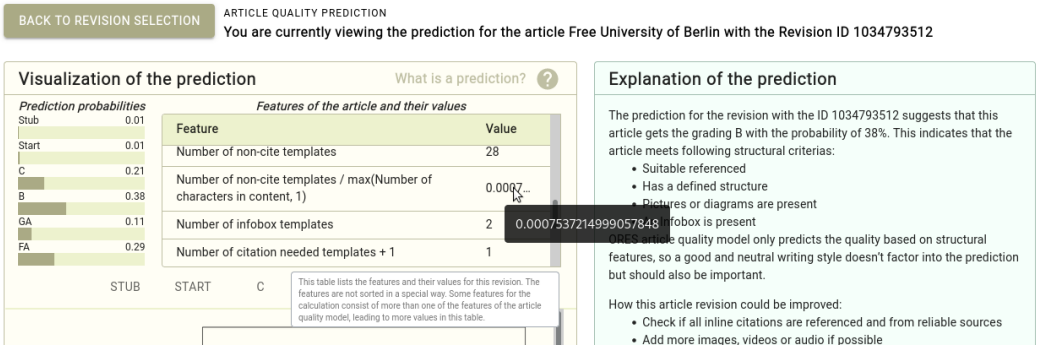

Abbildung 3.12: Weitere Informationen durch hovern über die Tabelle

Über ein '?'-Symbol neben der Überschrift können mehr Informationen zu Vorhersagen abgerufen werden. Die Karte 'Explanation of the prediction' hebt die vorhergesagte Note hervor, erklärt allgemein, welche Kriterien dieser Artikel schon erfüllt und wie der Artikel verbessert werden kann. Die Verbesserungsvorschläge sind in diesem Fall nicht abhängig von den *Feature*-Werten, sondern nur von der vorhergesagten Note. Um mehr Informationen zu den *Features* zu bekommen, gibt es die 'Explaining features' Karte. Dort kann ein einzelnes *Feature* ausgewählt und eine genauere Erklärung zu ihrer Bedeutung erhalten werden. Dazu steht auch der originale *Feature*-Name und ein Beispiel, wie diese Elemente aussehen können. Allgemeine Informationen zu *Features* können über das '?'-Symbol neben der Überschrift abgerufen werden.

Durch die Zurück-Button auf jeder Seite können die Nutzer ihre Entscheidung jederzeit Schritt für Schritt rückgängig machen. Durch das Löschen des Eintrages in der Suche und ein erneutes Bestätigen können auch die Informationstexte der Startseite wieder angezeigt werden.

4 Evaluation

Für die Evaluation der Anwendung waren synchrone Usability-Tests angedacht, doch aufgrund mangelnder Teilnehmer wurden asynchrone Tests durchgeführt. Der Schwerpunkt dieser Tests lag darauf, ob die Nutzer die Vorhersage und *Explanation* verstehen.

4.1 Vorgehen

Als erstes wurde die Anwendung über einen Ubuntu 18.04 Server aufrufbar gemacht. Dann wurde ein Google Docs Formular erstellt, welches eine Einleitung in das Thema, generelle Fragen und den *Usability Test* enthielt. Am Anfang des Test wurden die Teilnehmer darauf hingewiesen, wofür die gesammelten Daten verwendet werden. Ich hatte die Umfrage so eingestellt, dass keine E-Mail Adressen von Google gespeichert werden, daher waren diese Tests anonym.

Die generellen Fragen beinhalteten:

1. In which field do you work?
2. Have you ever edited an article on https://en.wikipedia.org or other Wikipedia pages?
3. Do you know (a little) about machine learning algorithms?
4. Did you know about ORES before this Usability test?

Für diese Fragen konnten die Teilnehmer aus verschiedenen Optionen auswählen.

Für den asynchronen Test wurde den Teilnehmern eine grobe Anleitung dazu gegeben, was sie innerhalb der Anwendung machen sollen. Nachdem die Teilnehmer die Seite für eine Vorhersage offen hatten, sollten sie folgende Fragen beantworten:

5. Do you fell like you understand the given graphs? If not, what would help to make it more understandable?
6. Is there anything in the prediction and explanation that you can't understand with the information given in the application?

7. Does the explanation help you to understand why the article received the calculated grade? If it does help, please elaborate what helped you the most. If it doesn't help, please elaborate what could be added to make it more understandable.
8. Would the information given by the explanation help you to know how to improve this article? If it does help, please elaborate what helped you the most. If it doesn't help, please elaborate what could be added.

Am Ende des Tests wurden noch drei allgemeinere Fragen zur Anwendung gestellt:

9. What is your overall impression of the application?
10. What worked well for you?
11. What could be improved?

Der Link zu dem Test wurde auf mehreren Plattformen geteilt, auf denen sich Personen befanden, die Erfahrungen mit Wikipedia oder mit *ML* haben.

4.2 Ergebnisse

An dem asynchronen Test hatten drei Personen teilgenommen gehabt. Die Fragen sowie die Antworten waren auf englisch. Für die Auswertung habe ich sie in die deutsche Sprache übersetzt.

Die Ergebnisse wurden in Tabelle 4.1 zusammengefasst. Die Nummern hinter den Kategorien beziehen sich auf die Nummerierung der Fragen, die zuvor im Abschnitt 4.1 definiert wurden.

Tabelle 4.1: Ergebnisse der letzten Evaluation

	A1	**A2**	**A3**
Arbeitsfeld (1)	Wissenschaft und Technologien	Wissenschaft und Technologien	Studium
Erfahrungen mit Editierungen (2)	Nein	Ja, auf der englischen Wikipedia	Nein
Erfahrungen mit *ML* (3)	Ja	Ja	Ja

Fortsetzung auf der nächsten Seite

Tabelle 4.1 Informationen der Teilnehmer der letzten Evaluation

	A1	**A2**	**A3**
Erfahrungen mit ORES (4)	Nein	Ja	Nein
Verstand die Graphen (5)	Ja	Ja	Nein, es ist nicht klar warum ein *Feature* einen roten oder grünen Balken hat und wie das *Feature* die Höhe des Balken beeinflusst.
Unklarheiten (6)	Definition der Klassen/Noten	Keine	Kein Vorwissen zu Wikipedias Benotungssystem.
Hilft die *Explanation* um die Vorhersage zu verstehen (7)	Ja	Ja, der Text hatte es gut erklärt.	Der Graph gibt eine gute Übersicht über die positiven und negativen *Feature* in einem Artikel, aber es ist schwer einzuschätzen, wie viele Änderungen gemacht werden müssen, um ein *Feature* im grünen Bereich zu haben.
Hilft die *Explanation* um den Artikel zu verbessern (8)	Ja, es war hilfreich, die Definitionen der Feature abrufen zu können.	Nein, die Vorschläge waren zu allgemein gehalten.	Gibt eine gute Übersicht darüber, welche Aspekte des Artikels noch Verbesserungen brauchen.

Fortsetzung auf der nächsten Seite

Tabelle 4.1 Informationen der Teilnehmer der letzten Evaluation

	A1	**A2**	**A3**
Gesamteindr (9)	Es war interessant zu sehen, wie sehr die Vorhersage von den *Features* abhängt.	*¨grade 'start'¨*	Sehr positiv.
Hat gut funktioniert (10)	Das Interface ließ sich gut nutzen.	*Prediction probabilities* und die *Explanation* waren leicht verständlich.	*¨Easy to use, hard to master¨*
Verbesserungsvorschläge (11)	Die Klassen könnten besser erklärt werden.	*Feature* in der Tabelle sollten nach Relevanz für die vorhergesagte Note sortiert werden, damit man nicht so viel scrollen muss.	Sehr technisch.

5 Diskussion

Bei dieser letzten Evaluation und auch den Evaluationen der Prototypen wünschten sich mehrere Testteilnehmer genauere Angaben dazu, was ein guter Wert für die einzelnen *Feature* wäre. Dies konnte, wie ich zuvor schon in Abschnitt 3.1.6 erwähnte, aus zeitlichen Gründen nicht mehr realisiert werden, aber ich kann verstehen, warum dies gefordert wurde. Bei der Analyse der *Feature* für den Abschnitt 2.1.1 fragte ich mich auch, ob es bei der Anzahl der Abschnitte eine Grenze gibt. Beispielsweise scheinen mehr Abschnitte in einem Artikel positiv gewertet zu werden, aber würde das auch bei einer sehr großen Anzahl von Abschnitten gelten?

Wenn die idealen Werte für die *Feature* ermittelt werden, könnten diese bei den *Feature*-Beschreibungen stehen und auch dazu genutzt werden, auf die *Explanation* zugeschnittene Tipps, für Verbesserungen am Artikel, zu geben. Einige Teilnehmer hatten diesbezüglich auch angesprochen gehabt, dass ihnen die allgemeinen Vorschläge nicht genug waren.

Die Teilnehmer, die Erfahrungen mit der Editierung auf Wikipedia hatten, hatten oft weniger Probleme, die Definition der Klassen zu verstehen. Im Gegensatz zu den anderen Teilnehmern, brauchten diese keine weiteren Erklärungen oder nur ein wenig mehr Informationen. Wenn man aber bedenkt, dass vielleicht auch neue Editoren oder Nutzer, die am Editieren interessiert sind, die Anwendung als Einstieg nutzen wollen, sollten die Informationen dort noch etwas verbessert werden. Dass besonders auch unerfahrene Editoren mit Hilfe von ORES Algorithmen unterstützt werden sollten, schlugen bereits befragte Wikipedia *Stakeholder* aus dem Paper 'Keeping Community in the Loop: Understanding Wikipedia Stakeholder Values for Machine Learning-Based Systems' [SYS+20] vor. Um die Anwendung im Bezug darauf zu verbessern, sollte statt einer allgemeinen Erklärung jede Note ihre eigenen Informationen erhalten. Umsetzen könnte man dies entweder durch das Auftauchen von Informationen beim Hovern oder durch das Anklicken der jeweiligen Note in der *Prediction probabilites* Anzeige.

Die Tabelle der *Feature*-Werte folgte in der Implementierung der Reihenfolge, die von der Vorhersage angegeben wurde. Diese folgte keiner besonderen Sortierung und war für jede Vorhersage die selbe. Einer der Teilnehmer der letzten Evaluation merkte dies auch an und hätte es besser gefunden, wenn diese nach der Relevanz der *Feature* für die vorhergesagte Note sortiert gewesen wäre. So musste man in einigen Fällen etwas länger scrollen, um die wirklich relevanten

Feature-Werte zu finden. LIME hat die Möglichkeit, eine nach Relevanz sortierte Tabelle im HTML-Format ausgeben zu lassen, was ich auch am Anfang versuchte, in das Frontend einzubinden. Da es Probleme bei der Einbindung gab, wurde stattdessen die unsortierte Tabelle gewählt. Dabei hatte ich aber nicht bedacht, dass Ziele der Nutzer möglicherweise dadurch schlechter und langsamer zu erreichen sind. Wenn dies der Fall ist, könnte es passieren, dass Nutzer ihre eigentlichen Ziele nicht weiterverfolgen wollen [SYS+20].

Die Struktur der Seite kam bei den Testern gut an. Es schien nicht so, als ob sie sich mit Informationen überladen fühlten und kleine Informationen überall ein wenig zu verteilen, schien eine gute Entscheidung gewesen zu sein.

6 Fazit

Mit der Nutzererfahrung im Vordergrund war das Ziel dieser Arbeit, eine Anwendung zu entwickeln, die Ausgaben und Vorhersagen des *ML* Modells *articlequality* von ORES für Editoren der englischen Wikipedia interpretierbar zu machen. Die zur Verfügung gestellten Visualisierungen und Erklärungen sollten nicht nur das *articlequality* Modell transparenter machen, sondern auch bei der Verbesserung der Artikel helfen.

Für die Entwicklung des *User Interfaces* wurden an den entstandenen Prototypen *Usability Tests* mit potentiellen Nutzern der Anwendung und Personen, die sich mit nutzerfreundlichen *UIs* auskennen, durchgeführt. Diese waren in der Hinsicht nötig und hilfreich, um zu verstehen, was an der Gestaltung und an der Navigation der Anwendung verbessert werden musste, bevor diese implementiert wird. Die Visualisierungen und Inhalte konnte bei diesen Tests schlecht beurteilt werden, da diese nur nachgebaut wurden und noch keine realen Daten beinhalteten. Zu dem Zeitpunkt war auch noch nicht ganz klar, was später bei der Implementierung realisierbar war.

Die resultierend programmierte Anwendung konnte Visualisierungen der Vorhersagen, die mit LIME erstellt wurden, ausgeben. Dazu kam noch eine Erklärung der Vorhersage, inklusive allgemeiner Tipps für die Verbesserung des Artikels und die Möglichkeit, sich die Definitionen der Feature anzeigen zu lassen. Auch, wenn es relativ schwer war, Editoren für die folgenden *Usability Tests* begeistern zu können, gaben Testpersonen mit *ML* Erfahrungen und Personen, die an der Editierung von Artikeln interessiert waren, einen guten Einblick, was an der Anwendung für neue und unerfahrende Editoren fehlt. Die letzte Evaluation zeigte, dass durch die Anwendung verstanden werden konnte, was die ausschlaggebenden Feature für eine Vorhersage waren. Entscheidungen des *articlequality* Modells konnten transparenter gemacht werden, aber es machte auch deutlich, dass der jetzige Stand der Anwendung noch keine ausreichenden Tipps zur Verbesserung von Artikeln geben kann.

6.1 Ausblick

Die gewonnen Erkenntnisse aus der letzten Evaluation deckten drei nötige Änderungen auf:

- Die Tabelle, die die *Feature*-Werte beinhaltet, muss sinnvoll sortiert werden.

- Die Noten, die von der englischen Wikipedia vergeben werden, müssen besser erklärt werden.
- Die Tipps sollten auf die Vorhersage zugeschnitten werden.

Eine Sortierung der *Feature* innerhalb der Tabelle ist dahingehend nötig, um den wahrscheinlich interessantesten *Feature*-Wert schnell finden zu können. Die Reihenfolge sollte sich daher an die der Graphen halten. Das heißt, die Tabelle müsste sich dynamisch an die, in den Tabs ausgewählte, Note anpassen. Ganz oben sollten also die einflussreichsten *Feature* der Note stehen.

Da sich nicht jeder neue Editor von Anfang an mit der Bewertung von Artikeln auf der englischen Wikipedia auskennen wird, sollten die Noten innerhalb der Anwendung besser erklärt werden. Durch Anklicken der einzelnen Noten könnte man Informationen zu ihnen anzeigen lassen. Die Informationen zu den Noten sollten hierfür von der englischen Wikipedia selber beziehen.

Die Tipps, die die Anwendung gibt, sollten sich an den *Feature*-Werten orientieren, dazu müsste aber erst analysiert werden, welche Werte die *Feature* anstreben sollten.

Literaturverzeichnis

[BAKSF13] Robert Biuk-Aghai, Ck Kit, Yain Whar Si, and Simon Fong. Visualizing recent changes in wikipedia. *SCIENCE CHINA Information Sciences*, 56:1–15, 05 2013.

[BDRD+20] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115, 2020.

[CHF10] Fanny Chevalier, Stéphane Huot, and Jean-Daniel Fekete. Wikipediaviz: Conveying article quality for casual wikipedia readers. pages 49–56, 03 2010.

[Fou21] Interaction Design Foundation. `https://www.interaction-design.org/literature/topics/user-centered-design`, besucht am 10. August 2021.

[GDC21] Leonida Gianfagna and Antonio Di Cecco. *Explainable AI with Python*, pages 1–25. Springer International Publishing, Cham, 2021.

[Gna20] Sajeera Gnanasegaram. Konzept und implementierung einer visuellen methode zur verbesserung der interpretierbarkeit der automatisierten qualitatsbewertung mit ores in wikidata, 2020.

[HG20] Aaron Halfaker and R. Stuart Geiger. Ores: Lowering barriers with participatory machine learning in wikipedia, 2020.

[KMG+19] Christoph Kinkeldey, Claudia Müller-Birn, Tom Gülenman, Jesse Josua Benjamin, and Aaron Halfaker. PreCall: A Visual Interface for Threshold Optimization in ML Model Selection. *arXiv e-prints*, page arXiv:1907.05131, July 2019.

[LPK20] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable ai: A review of machine learning interpretability methods. *Entropy*, 23:18, 12 2020.

[Med21a] MediaWiki. `https://www.mediawiki.org/wiki/ORES`, besucht am 21. Juni 2021.

[Med21b] MediaWiki. https://www.mediawiki.org/wiki/ORES/FAQ, besucht am 21. Juni 2021.

[Mol21] Christoph Molnar. https://christophm.github.io/interpretable-ml-book/lime.html, besucht am 30. Juli 2021.

[Rad21] Dario Radečić. https://python-bloggers.com/2020/12/lime-vs-shap-which-is-better-for-explaining-machine-learning-models/, besucht am 29. Juli 2021.

[SYS+20] C. Estelle Smith, Bowen Yu, Anjali Srivastava, Aaron Halfaker, Loren Terveen, and Haiyi Zhu. Keeping community in the loop: Understanding wikipedia stakeholder values for machine learning-based systems. New York, NY, USA, 2020. Association for Computing Machinery.

[Wik21] Wikipedia. https://en.wikipedia.org/wiki/Wikipedia:Content_assessment, besucht am 28. Juni 2021.

[Yab20] Jon Yablonski. *Laws of UX: Design Principles for Persuasive and Ethical Products*. O'Reilly Media, 2020.

Anhang

6.2 Einwilligungserklärungen

Einverständniserklärung
zur Durchführung von Audio- und Video-Aufzeichnung und der Verarbeitung personenbezogener Daten.

Hiermit erteile ich meine Einwilligung, dass im Rahmen eines Usability-Tests eines Prototypen die von mir erstellten Audio- und Video-Aufzeichnungen für die Evaluation der Bachelorarbeit von Tanita Daniel verarbeitet werden dürfen.

Folgende personenbezogene Daten werden verarbeitetet:

- Biographische Daten (z.B. Ausbildung, Beruf)
- Inhaltsdaten aus den Audio- und Video-Aufzeichnungen (z.B. die aufgerufene Webseite, das gesprochene Wort)
- Daten, die ein Nachweis Ihrer Einwilligung der Datenverarbeitung sind (z.b. diese Einwilligungserklärung)

Die genannten personenbezogen Daten werden ausschließlich zu Forschungszwecken verwendet.

Diese Einwilligung ist freiwillig und ich kann sie ohne Angabe von Gründen verweigern, ohne dass ich deswegen Nachteile zu befürchten hätte. Zum Widerruf ist eine formlose E-Mail an tanid96@zedat.fu-berlin.de ausreichend.

Berlin, den 17.06.2021

Ort, Datum

Unterschrift

Einverständniserklärung
zur Durchführung von Audio- und Video-Aufzeichnung und der Verarbeitung personenbezogener Daten.

Hiermit erteile ich meine Einwilligung, dass im Rahmen eines Usability-Tests eines Prototypen die von mir erstellten Audio- und Video-Aufzeichnungen für die Evaluation der Bachelorarbeit von Tanita Daniel verarbeitet werden dürfen.

Folgende personenbezogene Daten werden verarbeitet:

- Biographische Daten (z.B. Ausbildung, Beruf)
- Inhaltsdaten aus den Audio- und Video-Aufzeichnungen (z.B. die aufgerufene Webseite, das gesprochene Wort)
- Daten, die ein Nachweis Ihrer Einwilligung der Datenverarbeitung sind (z.b. diese Einwilligungserklärung)

Die genannten personenbezogen Daten werden ausschließlich zu Forschungszwecken verwendet.

Diese Einwilligung ist freiwillig und ich kann sie ohne Angabe von Gründen verweigern, ohne dass ich deswegen Nachteile zu befürchten hätte. Zum Widerruf ist eine formlose E-Mail an tanid96@zedat.fu-berlin.de ausreichend.

28.06.2021, Berlin
Ort, Datum

Unterschrift

Einverständniserklärung
zur Durchführung von Audio- und Video-Aufzeichnung und der Verarbeitung personenbezogener Daten.

Hiermit erteile ich meine Einwilligung, dass im Rahmen eines Usability-Tests eines Prototypen die von mir erstellten Audio- und Video-Aufzeichnungen für die Evaluation der Bachelorarbeit von Tanita Daniel verarbeitet werden dürfen.

Folgende personenbezogene Daten werden verarbeitetet:

- Biographische Daten (z.B. Ausbildung, Beruf)
- Inhaltsdaten aus den Audio- und Video-Aufzeichnungen (z.B. die aufgerufene Webseite, das gesprochene Wort)
- Daten, die ein Nachweis Ihrer Einwilligung der Datenverarbeitung sind (z.b. diese Einwilligungserklärung)

Die genannten personenbezogen Daten werden ausschließlich zu Forschungszwecken verwendet.

Diese Einwilligung ist freiwillig und ich kann sie ohne Angabe von Gründen verweigern, ohne dass ich deswegen Nachteile zu befürchten hätte. Zum Widerruf ist eine formlose E-Mail an tanid96@zedat.fu-berlin.de ausreichend.

Berlin, 28.06.2021

Ort, Datum

Unterschrift

Einverständniserklärung
zur Durchführung von Audio- und Video-Aufzeichnung und der Verarbeitung personenbezogener Daten.

Hiermit erteile ich meine Einwilligung, dass im Rahmen eines Usability-Tests eines Prototypen die von mir erstellten Audio- und Video-Aufzeichnungen für die Evaluation der Bachelorarbeit von Tanita Daniel verarbeitet werden dürfen.

Folgende personenbezogene Daten werden verarbeitetet:

- Biographische Daten (z.B. Ausbildung, Beruf)
- Inhaltsdaten aus den Audio- und Video-Aufzeichnungen (z.B. die aufgerufene Webseite, das gesprochene Wort)
- Daten, die ein Nachweis Ihrer Einwilligung der Datenverarbeitung sind (z.b. diese Einwilligungserklärung)

Die genannten personenbezogen Daten werden ausschließlich zu Forschungszwecken verwendet.

Diese Einwilligung ist freiwillig und ich kann sie ohne Angabe von Gründen verweigern, ohne dass ich deswegen Nachteile zu befürchten hätte. Zum Widerruf ist eine formlose E-Mail an tanid96@zedat.fu-berlin.de ausreichend.

Berlin, 02.07.2021
Ort, Datum

Unterschrift

6.3 Code

```
import os.path
import mwapi
import numpy as np
import csv
```

```
5  from revscoring import Model
6  from revscoring.extractors import api
7  from revscoring.utilities.util import read_observations # Get training set
8
9  # model and observation data taken from https://github.com/wikimedia/
       articlequality/
10 model_path = os.path.realpath("functions/articlequality/models/enwiki.nettrom_wp10
       .gradient_boosting.model")
11 sm = Model.load(open(model_path), error_on_env_check=False)
12 observations = list(read_observations(open(os.path.realpath("functions/articlequality/
       datasets/enwiki.labeling_revisions.nettrom_30k.json"))))
13
14 revids = []
15 for o in observations:
16     revids.append(o['rev_id'])
17 #revids = revids[0:12] # less data for test
18
19 # chunk data (has RAM problems with csv.writer)
20 # later: head -n 1 train0.csv > combined.out && tail -n+2 -q *.csv >> combined.
       out
21 chunkSize = 30
22 print(len(revids) / chunkSize)
23 revChunks = [revids[i:i + int(len(revids) / chunkSize)] for i in range(0, len(revids),
       int(len(revids) / chunkSize))]
24
25 # extract features from observed data and safe
26 extractor = api.Extractor(mwapi.Session("https://en.wikipedia.org", user_agent="
       ORES-LIME demo"))
27
28 for i in range(0,len(revChunks)):
29     rawTrain = list(extractor.extract(revChunks[i],sm.features))
30     train = []
31     train.append(sm.features)
32     for t in rawTrain:
33         if isinstance(t[1], list):
34             train.append(t[1])
35     with open("functions/articlequality/datasets/train" + str(i) + ".csv", "w+") as
           my_csv:
36         csvWriter = csv.writer(my_csv, delimiter=';')
37         csvWriter.writerows(train)
38     print("Finished Chunk" + str(i))
39
40 print("Finished!")
```

Listing 6.1: Python-Code um die Trainingsdaten vorzubereiten

```
1  from flask import Flask, send_from_directory, request, render_template
2  import flask
3  from flask_restful import Api, Resource, reqparse
4  from flask_cors import CORS #comment this on deployment
5  from api.ApiHandler import ApiHandler
6
```

```python
import json
from functions.wikiFunctions import *
from functions.visualization import *

app = Flask(__name__, static_url_path='', static_folder='qualityinsight-frontend/build')
CORS(app) #comment this on deployment
api = Api(app)

@app.route("/", methods = ['POST','GET'])
def home():
    searchTerm = request.args.get('search')
    searchRes = searchArticle(searchTerm)

    return flask.jsonify(searchRes)

@app.route("/found-revisions", methods = ['POST','GET'])
def foundRevisions():
    article = request.args.get('article')
    searchRes = getArticleVersions(article)
    return flask.jsonify(searchRes)

@app.route("/explanation", methods = ['POST', 'GET'])
def explanation():
    revid = request.args.get('revid')
    prediction, feature_values, features = getVersionScore(int(revid))
    #print(prediction)
    fig = getExplanation(prediction, feature_values)

    with open('data/feature-names.json') as json_file:
        altFeatureNames = json.load(json_file)
    featureList = []
    for i in range(0, len(features)):
        featureList.append([altFeatureNames[features[i]], list(feature_values)[i]])
    #print(featureList)
    return {"prediction": prediction, "fig": fig, "featureValues": featureList}

api.add_resource(ApiHandler, '/')

if __name__ == '__main__':
    app.run(host="85.214.159.67", port=5000, debug = False)
```

Listing 6.2: Python-Code der Flask Applikation

```python
from datetime import datetime
import requests # https://www.mediawiki.org/wiki/API:Revisions#Example_1:_Get_revision_data_of_several_pages

S = requests.Session()
URL = "https://en.wikipedia.org/w/api.php"

# searchs for articls that fit the search-term
```

```python
# and returns the titles and links (max. 10)
def searchArticle(text):
    params = {
        "action": "opensearch",
        "namespace": "0",
        "search": text,
        "limit": "10",
        "format": "json"
    }
    r = S.get(url=URL, params=params)
    data = r.json()

    # results in a list of dicts
    articles = []
    for i in range(0, len(data[1])):
        articles.append({"key": "a" + str(i), "articleName": data[1][i], "articleURL": data[3][i]})

    return articles

# gets the ten most recent revisions
def getArticleVersions(article):
    # https://stackoverflow.com/questions/34411896/how-to-get-full-wikipedia-revision-history-list-from-some-article
    params = {
        "action": "query",
        "prop": "revisions",
        "titles": article,
        "rvlimit": "10",
        "rvprop": "timestamp|user|comment|ids",
        "rvdir": "older",
        "rvexcludeuser": "SSethi (WMF)",
        "rvslots": "main",
        "formatversion": "2",
        "format": "json"
    }
    r = S.get(url=URL, params=params)
    data = r.json()
    revisions = data['query']['pages'][0]['revisions']

    # format timestamp
    for i in range(0, len(revisions)):
        timestamp = datetime.strptime(revisions[i]['timestamp'], '%Y-%m-%dT%H:%M:%SZ')
        revisions[i]['timestamp'] = timestamp.strftime("%d %B %Y, %H:%M:%S")
        revisions[i]['key'] = "r" + str(i)

    return revisions
```

Listing 6.3: Python-Code um Informationen von Wikipedia abzurufen

```python
import os.path
```

```
import base64
from io import BytesIO
from matplotlib.pyplot import show
from matplotlib.pyplot import figure
import matplotlib.pyplot as plt #, mpld3

import pandas as pd
import numpy as np
import json
import mwapi
from textwrap import wrap

#import articlequality
from revscoring import Model
from revscoring.extractors import api
from revscoring.utilities.util import read_observations # Get training set

from lime.lime_tabular import LimeTabularExplainer

class_names = ["B","C","FA","GA","Start","Stub"]

# model and observation data taken from https://github.com/wikimedia/articlequality/
model_path = os.path.realpath("functions/articlequality/models/enwiki.nettrom_wp10.gradient_boosting.model")
observations = list(read_observations(open(os.path.realpath("functions/articlequality/datasets/enwiki.labeling_revisions.nettrom_30k.json"))))

# load extractor for revscoring
extractor = api.Extractor(mwapi.Session("https://en.wikipedia.org", user_agent="ORES-LIME demo"))
# load model for enwiki
sm = Model.load(open(model_path), error_on_env_check=False)
# get featurenames
features = [str(f) for f in sm.features]
with open('data/feature-names.json') as json_file:
    altFeatureNames = json.load(json_file)
altFeatures = []
for f in features:
    altFeatures.append(altFeatureNames[f])
altFeatures = ['\n'.join(wrap(l, 40)) for l in altFeatures]

# based on tabular data -> prepared training set for LIME in ./articlequality/training/train.csv
# load training data
data = pd.read_csv(os.path.realpath("functions/articlequality/trainingdata/train.csv"),delimiter=";")
train = data.to_numpy()

def getVersionScore(revID):
    #print(revID)
    feature_values = np.array(list(extractor.extract(revID, sm.features)))
    #print(feature_values)
```

```
49      return sm.score(feature_values), feature_values, features
50
51  def getExplanation(prediction, feature_values):
52      print("Explanation start!")
53      explainer = LimeTabularExplainer(
54          train,
55          mode="classification",
56          feature_names=altFeatures,
57          class_names=class_names,
58          discretize_continuous=False
59      )
60
61      def score(samples):
62          raw_results = [np.array([sm.score(v)["probability"][t] for t in ["B","C","FA","
                GA","Start","Stub"]]) for v in samples]
63          return np.array(raw_results)
64
65      exp = explainer.explain_instance(
66          np.array(feature_values),
67          score,
68          num_features=40,
69          top_labels=6,
70          num_samples=500
71      )
72      print("LIME is done!")
73
74      # get graphs and info for table for every label
75      figures = []
76      #tables = []
77      with open('data/feature-names.json') as json_file:
78          altFeatureNames = json.load(json_file)
79      for i in range(0,6):
80          # graph
81          plt.rcParams.update({'font.size': 10})
82          fig = exp.as_pyplot_figure(label=i)
83          fig.set_size_inches(10, 40)
84          plt.tight_layout()
85          #fig.savefig('lime' + str(i) + '.png', transparent=False, dpi=100)
86
87          tmpfile = BytesIO()
88          fig.savefig(tmpfile, format='png', transparent=True)
89          encoded = base64.b64encode(tmpfile.getvalue()).decode('utf-8')
90          figures.append(encoded)
91          plt.close('all')
92
93      #exp.save_to_file('lime.html',labels=[0])
94      return figures
```

Listing 6.4: Python-Code für die Vorhersage von Noten und der Explanation Visualisierung

```
import "./Home.css";
import React, { useEffect, useState } from "react";
import axios from "axios";
import { useHistory, useLocation } from "react-router-dom";

import Textfield from "@material-ui/core/TextField";
import InputAdornment from "@material-ui/core/InputAdornment";
import IconButton from "@material-ui/core/IconButton";
import SearchIcon from "@material-ui/icons/Search";
/* import SearchBar from '../components/SearchBar'; */

import SearchResult from "../components/SearchResult";
import Information from "../components/Information";

// submit search function
let startSearch = (search, setResults) => {
  if (search.length === 0) {
    setResults((results) => []);
    return;
  }

  axios
    .get("http://85.214.159.67:5000/?search=" + search)
    .then((response) => {
      setResults((results) => response.data);
    })
    .catch((error) => {
      console.log(error);
    });
};

export default function Home() {
  let initValue = new URLSearchParams(useLocation().search).get("search");
  let history = useHistory();
  let [search, setSearch] = useState(initValue || "");
  const [results, setResults] = useState([]);

  // search term via query
  useEffect(() => {
    startSearch(search, setResults);
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, []);

  // (update on) update search term
  useEffect(() => {
    history.push({
      pathname: "/",
      search: "?search=" + search,
      state: {
        search,
      },
    });
  }, [search, history]);
```

```
54
55    return (
56      <div id="home">
57        <form
58          onSubmit={(ev) => {
59            ev.preventDefault();
60            startSearch(search, setResults);
61          }}
62        >
63          <Textfield
64            selected
65            classes={{ root: "searchbar" }}
66            placeholder="Search for an article on en.wikipedia.org"
67            variant="outlined"
68            defaultValue={search}
69            size="small"
70            onInput={(ev) => {
71              setSearch(ev.target.value);
72            }}
73            InputProps={{
74              endAdornment: (
75                <InputAdornment position="end">
76                  <IconButton type="submit" edge="end">
77                    <SearchIcon />
78                  </IconButton>
79                </InputAdornment>
80              ),
81            }}
82          />
83        </form>
84        {results.length > 0 ? results.map(SearchResult) : <Information />}
85      </div>
86    );
87  }
```

Listing 6.5: Frontend der Startseite