Report - spam classification project

Hannah Peuckmann

September 17, 2020

Als Modulprojekt habe ich mich für die *spam*-Klassifikation auf dem Enron Email Korpus entschieden. Die Klassifikation ist binär, unterschieden wird zwischen den Klassen *spam* und *ham*. Die Klassifikation basiert auf *features*, aus denen ein Model für jede Klasse erstellt wird. Bei den *features* habe ich hauptsächlich *features* auf Zeichenebene gewählt; Bindestriche, Ausrufungszeichen, *whitespace*, Sonderzeichen, und zusätzlich zu diesen noch ein *feature* auf Wortebene; die häufigsten Nomen der Betreffzeilen der *spam*-Mails. Mit dieser Zusammenstellung an *features* habe ich während des Projekts die beste *accuracy* erzielt. Ich habe einige weitere *features* ausprobiert, Anzahl der Tokens pro Mail, Großbuchstaben, Anzahl der Sätze, Pronomen, Stopwörter, alle haben, einzeln so wie in Kombination, die *accuracy* verschlechtert. Bis auf die Anzahl der Großbuchstaben pro Mail, dieses *feature* hat die *accuracy* nicht beeinflusst. Das Projekt könnte durch weitere *features* erweitert werden, bzw. durch das Ersetzen von *features*. Das Projekt könnte auch an einen anderen Datensatz angepasst werden oder auch für eine Klassifizierungsaufgabe mit mehr Klassen erweitert werden.

Das Projekt ist aufgeteilt in drei Klassen und eine *main*-Funktion. Die Klasse *Features* ist dafür zuständig, für eine einzelne .txt-Datei, die ausgewählten *features* zu extrahieren. Die *SpamClassifier* Klasse ist für die Klassifizierung zuständig. Sie trainiert zunächst auf einem gegebenen Korpus, und erstellt ein Model für jede Klasse, indem für jede Mail mit der *features* Klasse die *features* extrahiert werden. Anschließend kann auf Basis des erstellten Models, für eine einzelne Mail eine Vorhersage getroffen werden, oder für einen ganzen Satz an Mails. In letzterem Fall wird eine Evaluation der Ergebnisse durchgeführt, die *accuracy* wird berechnet. Die *SplitCorpus* Klasse ist für das *preprocessing* der Korpusdaten zuständig. Ich habe mich dazu entschieden, nicht die .txt-Dateine der Mails an sich in einen

train/test/val Ordner zu verschieben, sondern lediglich die Dateipfade der .txts in einer .csv zu speichern. Die *main*-Funktion ist dafür zuständig, die Benutzereingaben zu verabeiten.

Die Module, die ich genutzt habe:
os und *path* habe ich zum Navigieren in der Ordnerstruktur des Korpus gebraucht.
sklearn und *pandas* habe ich gebraucht, um die Daten in *train, val* und *test* zu unterteilen.
progress habe ich benutzt, um eine *progress bar*, bzw. einen *progress spinner* zu benutzen.
Counter habe ich benutzt, um die Nomen der Betreffzeilen zu zählen.
namedtuple habe ich benutzt, um die Vorhersage für eine einzelne Mail zu repräsentieren, sowie für das Speichern eines gesplitteten Korpus in *train, val* und *test*.
add und *sub* habe ich genutzt, um Listen elementweise zu addieren, bzw. subtrahieren.
logging, um ein *logfile* zuschreiben.
sys, damit der Benutzer dem Programm Argumente über die Kommandozeile übergeben kann.
nltk zum tokenisieren und taggen.

3. Der Aufgabenbeschreibung bin ich recht chronologisch gefolgt. Einen Teil der Aufgabenstellung hatte ich missverstanden, die *NLP tasks* die in Schritt Drei obligatorisch erledigt werden sollten. Ich habe die geforderten *NLP features* zunächst nicht gebraucht, da ich auf Zeichenebene gearbeitet habe und diesen Schritt der Aufgabenstellung nicht auslassen wollte, habe ich eine Klasse geschrieben, die den gesamten Korpus verarbeitet, tokenisiert, taggt und lemmatisiert und im *tiger*-Format in eine Datei schreibt. Im Endeffekt war diese Klasse nicht nötig, da die *NLP tasks* nicht zwingend zu erledigen waren. Zuerst habe ich mir ein Paar wenige einfach zu implementierende *features* herausgesicht, um einmal bis Schritt 8 durchzuarbeiten. Anschließend habe ich dann weitere aufwändigere *features* ausprobiert. Im Endeffekt bin ich aber fast bei diesen ersten *features* geblieben, da ich mit diesen die beste *accuracy* erreicht habe. Am schwierigsten war für mich, die Strukturierung meines Projekts, damit gemeint ist, zu entscheiden, welche Methoden sinnvoll eine Klasse bilden, wie viele Klassen es geben wird und auch, wie die Schnittstelle zum Benutzer gestaltet wird, welche Funktion-

alitäten dem Nutzer zur Verfügung stehen werden und wie diese aufzurufen sind. Die meiste Zeit habe ich damit verbracht, mein Projekt immer mal wieder zu restrukturieren. Gegen Ende habe ich noch mal etwas an der *SplitCorpus* Klasse geändert, um meine Daten anders abzuspeichern. Die *train/test/val sets*, die bis dahin für *ham* und *spam* separat gespeichert waren habe ich zusammengelegt, dadurch sind mehrere verschachtelte Funktionen nicht mehr nötig gewesen. Bei meiner *SpamClassifier* Klasse war es für mich schwierig, zu entscheiden wie ich mit den .csv Dateien arbeiten will. Ich hab es erst mit normalem, zeilenweise Einlesen versucht, dann habe ich kurz *Pandas* ausgetestet um dann doch wieder darauf zurückgekommen die Datei einfach zeilenweise Einzulesen und so zu Verarbeiten. Leider kommt es dadurch an einer Stelle dazu, dass ich eine Datei zweimal einlesen muss. Ich denke, beim nächsten Mal würde ich den Korpus anders handhaben, ich würde eventuell doch Ordner für *train test* und *validation* erstellen, die die tatsächlichen .txt-Dateien der Mails enthalten, anstatt .csv-Dateien mit Dateipfaden zu erstellen. Auch würde ich mehr Zeit einplanen, für das Implementieren und Ausprobieren von *features*. Die *review* die ich bekommen habe hat mir sehr geholfen, es sind Flüchtigkeitsfehler aufgefallen und Ideen zur Umsetzung, die ich selber nicht hatte.