Sandra Sanchez Paez

Olha Svezhentseva

Projektbericht

Korpus

Wir haben den Korpus (IBM_Debater_(R)_CE-EMNLP-2015.v3) ausgewählt, der im Rahmen des Seminars präsentiert wurde.

Das Thema vom dazugehörigen Paper *Show me your Evidence* ist ***Automatic Method for Context Dependent Evidence Detection***. Gegeben ein Debattenthema, ein Claim und ein Text muss das Programm eine oder mehrere Evidenzen finden, die diesen Claim unterstützen. Das ist ein zweistufiger Prozess, zuerst muss das Programm alle Kandidaten finden (kontext-unabhängiger Teil) und dann prüfen, welche der Kandidaten dem gegebenen Thema und Claim entsprechen (kontext-abhängiger Teil).

Da die Aufgabe ziemlich komplex ist, haben wir später entschieden, Claims statt Evidenzen zu identifizieren. Außerdem hatten wir Schwierigkeiten, weil der Korpus nicht sehr “benutzerfreundlich” annotiert wurde. Claims und Evidenzen wurden nicht im Text annotiert, sondern es sind mehrere txt-Dateien, wo zum Beispiel das Thema und dazugehörende Claims stehen. Deshalb müssen Texte durchgesucht werden, wenn man auf Satzebene Claims von Nicht-Claims unterscheiden möchte. Nachdem wir mit Dictionaries experimentiert hatten, haben wir unseren Ansatz geändert und uns für Dataframes entschieden, weil es intuitiver war. So haben wir gelabelte Daten bekommen.

Preprocessing

Während des Preprocessing werden Texte zuerst tokenisiert, dann werden Stoppwörter entfernt und anschließend werden Tokens lemmatisiert.

Extract Features

Da es unsere erste Erfahrung mit ML Modellen war, gab es ein paar wichtige Details, die für uns neu waren. So haben wir zum Beispiel einen besseren Einblick bekommen, wie TF-IDF Vectorizer funktioniert und was der Unterschied zwischen *fit_transform* und *transform* ist. Nach ein paar Versuchen haben wir die Daten erfolgreich in Train, Dev und Test Sets gesplittet, um erst danach *fit_transform* auf Train Set aufzurufen und *transform* auf Dev/Test set. So werden die Ergebnisse nicht manipuliert und das Modell lernt nur das Vocabulary von Train Set während der Umgang mit ungesehenen Tokens intern implementiert ist und wir glauben, dass es hier auch möglich wäre, unterschiedliche Ansätze auszuprobieren, was aber im Rahmen dieses Projekts nicht unser primäres Ziel war.

Außerdem haben wir uns ein paar Features überlegt, die auf einen Claim hinweisen könnten. Und es ist wichtig hinzuzufügen, dass die Daten in einem numerischen Format kodiert werden müssen, damit das Modell lernen kann. Die Features sind wie folgt:

1. Opinion-Wörter
2. Subjektivität des Satzes
3. Länge des Satzes

4. Entities

Wir haben eine kleine Liste von **Verben** zusammengestellt, die oft benutzt werden, um die Meinung auszudrücken, nämlich "claim", "state", "believe", "think", "suggest". Funktion check_opinion_verbs prüft ob es mindestens 1 solches Wort im Satz vorkommt und gibt dann 1 zurück, sonst 0. Sie könnte auch umgebaut werden, so dass sie einen normalisierten Wert ausgeben würde, wie viele Wörter aus dem Satz ein Opinion Wort sind, aber wir waren uns nicht sicher, inwiefern es für einen Claim typisch ist, mehrere Opinion Wörter zu enthalten.

Das zweite Feature ist **Subjektivität**, die Funktion gibt einen Wert zwischen 0 (nicht subjektiv) und 1(subjektiv) zurück. Dafür wurde Pysentiment benutzt. Die Annahme war, dass Claims oft über ein gewisses Maß der Subjektivität verfügen.

Das dritte Feature ist die **Länge des Satzes**, wir sind davon ausgegangen, dass Claims normalerweise mindestens 4 Wörter enthalten.

Das vierte Merkmal sind **Entities**, die wir als relevant für die Identifizierung von Evidenzen erachtet und programmiert hatten, die aber auch bei Claims eine wichtige Rolle spielen. Hierfür verwenden wir spacy. In diesem Fall gibt die Funktion eine 1 zurück, wenn eine zur Gruppe "Person" oder "Organisation" gehörende Entität identifiziert wird, ansonsten eine 0.

In unserem Fall war es eine interessante und hilfreiche Erfahrung über Features nachzudenken aber es hat Ergebnisse nicht wirklich beeinflusst,
TF-IDF beschäftigt sich mit Relevanz von Wörtern und kann daher als eine viel komplexere Version von unserem ersten Feature (Opinion-Wörter) betrachtet werden. Da die Subjektivität des Satzes meistens durch die Wortwahl ausgedrückt wird, bringt es auch nicht viel. Die Länge des Satzes

mit der Grenze von 4 Wörtern scheint auch keine große Rolle zu spielen, weil die meisten Sätze in unserem Korpus länger sind. Da die Texte aus Wikipedia stammen und nicht zum Beispiel aus Twitter, ist es nicht überraschend. Ein weiterer Grund, warum die Ergebnisse durch die o.g. Features nicht beeinflusst wurden, ist die Größe der Matrix. Nach TF-IDf wird eine Matrix erstellt, die für jedes Token des Vocabulary eine Spalte hat. Nachdem zusätzliche Features hinzugefügt werden, hat die Matrix nur 3 Spalten mehr als zuvor. Natürlich tragen deshalb neue Features nicht viel bei.

Ergebnisse von Dev Set

Nachdem wir TF-IDF und die o.g. Features implementiert hatten und ein paar Modelle (MLP, SVM) laufen lassen haben, haben wir festgestellt, dass Accuracy sehr hoch war (97%) , während Recall von Klasse Claims 0 war. Unser Datensatz ist ein gutes Beispiel dafür, wie aussagekräftig oder nicht einige Metriken in bestimmten Fällen sein können. Der Datensatz ist sehr unbalanciert.
97% der Sätze sind keine Claims. Deshalb haben wir folgende Strategien benutzt, um Recall von Claims zu verbessern.

- Downsampling/Undersampling
- Oversampling

Beim **Downsampling** haben wir aus 75620 Sätzen alle Claims extrahiert (2199) und 3000 Nicht Claims, die zufällig ausgewählt wurden. Das hat den Recall deutlich verbessert. Wir wissen aber nicht, inwiefern das als eine gültige Lösung gilt, weil wenn das Modell auf unvorbereitete (unbalancierte) Daten aus der Welt angewendet wird, wird der Recall wieder viel niedriger.

Eine andere Möglichkeit, dieses Problem zu lösen, ist **Oversampling**: die Generierung neuer Samples in den Klassen, die unterrepräsentiert sind. Die naivste Strategie besteht darin, neue Samples zu generieren, indem die aktuell verfügbaren Samples durch zufällige Samples ersetzt werden. Der RandomOverSampler, aus dem Modul imbalanced_learn von sklearn, ermöglicht eine solche Vorgehensweise.

Obwohl man die Anzahl der erzeugten Samples sowie andere Parameter manuell auswählen kann, haben wir den Oversampler mit seinen Standardwerten verwendet und somit mit etwa doppelt so vielen Samples gearbeitet, was die Prozesse natürlich deutlich verlangsamt hat. Dadurch verbesserten sich die Ergebnisse, aber auch hier wissen wir nicht, inwieweit es sich um eine legitime Strategie handelt.

Classifying

Wir haben folgende Modelle trainiert:

- ☐ DummyClassifier als Baseline
- ☐ MLP
- ☐ SVC mit/ohne weights

Bezüglich der **Tuning der Hyperparameter** haben wir einige Tests mit der Anzahl der Iterationen, der Verwendung von n-Grammen und Stoppwörtern und vor allem der Gewichte durchgeführt, und wir haben gesehen, dass die Ergebnisse auch vom Typ des verwendeten Classifiers abhängen. Außerdem haben wir Grid Search benutzt, um die besten Parameter Values zu identifizieren.

In jedem Fall ist es wichtig, Informationen nicht doppelt zu verwalten (wenn beispielsweise dem Modell Gewichte gegeben werden, ergibt es keinen Sinn, mit Oversamplern zu arbeiten, da das Modell bereits nach einer Lösung für die Imbalance der Klassen sucht), und die Ergebnisse sind in diesem Fall deutlich schlechter (Recall bei Klasse Claim ist beim SVC mit Weights und RandomOversampler nur 0.09).

Evaluation

Comparison of F-scores

	Downsampling		Oversampling		Original	
	Not claim	Claim	Not claim	Claim	Not claim	Claim
SVC	0.77	**0.74**	0.98	0.33	0.99	0.08
MLP	0.76	**0.71**	0.98	0.33	0.99	0.24
Dummy classifier	0.72	0.0	0.99	0.0	0.99	0.0

	Not claim	Claim
SVC with weights	0.99	0.12