KAPITEL 3

Die Ähnlichkeit von Texten mithilfe von Worteinbettungen berechnen

Dies ist das erste Kapitel mit richtigem Code. Eventuell sind Sie direkt hierher gesprungen, und wer würde es Ihnen verübeln? Um den Rezepten zu folgen, ist es wirklich hilfreich, den dazugehörigen Code nebenher laufen zu lassen. Dies können Sie ganz einfach erreichen, indem Sie die folgenden Befehle in einer Shell ausführen:

```
git clone \
  https://github.com/DOsinga/deep_learning_cookbook.git
cd deep_learning_cookbook
python3 -m venv venv3
source venv3/bin/activate
pip install -r requirements.txt
jupyter notebook
```

Eine etwas detailliertere Erklärung findet sich im Abschnitt »Voraussetzungen« auf Seite XI.

In diesem Kapitel beschäftigen wir uns mit Worteinbettungen und wie diese uns dabei helfen können, Ähnlichkeiten zwischen Textstücken zu berechnen. Worteinbettungen sind eine effektive Technik in der Sprachverarbeitung, um Wörter als Vektoren in einem *n*-dimensionalen Raum darzustellen. Das Interessante an diesem Raum ist, dass Wörter mit ähnlicher Bedeutung nahe beieinanderliegen.

Das Hauptmodell, das wir hier verwenden, ist eine Variante von Googles Word2vec. Dies ist kein Deep-Learning-Modell. Tatsächlich ist es nicht mehr als eine große Nachschlagetabelle von Wort zu Vektor und daher eigentlich kaum als Modell zu bezeichnen. Die Word2vec-Einbettungen entstehen als Nebenprodukt beim Trainieren eines Netzwerks, das das nächste Wort aus dem Kontext von Sätzen aus Google News vorhersagen soll. Darüber hinaus ist es wahrscheinlich das bekannteste Beispiel für Worteinbettungen, und diese sind ein wichtiges Instrument im Deep Learning.

Sobald Sie danach Ausschau halten, werden Ihnen hochdimensionale Räume mit semantischen Eigenschaften in allen Bereichen des Deep Learnings begegnen. Wir können ein Empfehlungssystem für Filme erstellen, indem wir die Filme in einen

hochdimensionalen Raum projizieren (Kapitel 4), oder wir können eine Karte handgeschriebener Ziffern auf nur zwei Dimensionen darstellen (Kapitel 13). Bilderkennungsnetzwerke projizieren Bilder so in einen Raum, dass ähnliche Bilder nahe beieinanderliegen (Kapitel 10).

In diesem Kapitel konzentrieren wir uns nur auf Worteinbettungen. Zunächst berechnen wir mit einem vortrainierten Worteinbettungsmodell Wortähnlichkeiten und zeigen dann einige interessante Wort2vec-Recheneigenschaften auf. Danach ergründen wir, wie wir diese hochdimensionalen Räume visualisieren können.

Als Nächstes schauen wir, wie wir die semantischen Eigenschaften von Worteinbettungen wie Word2vec für domänenspezifische Rangordnungen nutzen können. Wir behandeln die Wörter und deren Einbettungen wie die Objekte, die sie repräsentieren, und kommen damit zu einigen interessanten Ergebnissen. Wir beginnen mit der Suche nach Objektklassen in Word2vec-Einbettungen – in diesem Fall Ländern. Dann zeigen wir, wie man Begriffe im Vergleich zu diesen Ländern einordnet und wie man die Ergebnisse auf einer Karte visualisieren kann.

Worteinbettungen sind ein mächtiges Instrument, um Wörter auf Vektoren abzubilden, und haben eine Vielzahl von Einsatzmöglichkeiten. Oft werden sie auch als Vorverarbeitungsschritt für Text verwendet.

Zu diesem Kapitel gehören die folgenden zwei Python-Notebooks:

```
03.1 Using pretrained word embeddings
03.2 Domain specific ranking using word2vec cosine distance
```

3.1 Wortähnlichkeiten mithilfe vortrainierter Worteinbettungen finden

Problem

Sie wollen herausfinden, ob zwei Wörter ähnlich, aber nicht gleich sind, zum Beispiel wenn Sie Benutzereingaben überprüfen und dabei nicht vom Benutzer verlangen möchten, genau das erwartete Wort einzugeben.

Lösung

Sie können ein vortrainiertes Worteinbettungsmodell verwenden. Im folgenden Beispiel verwenden wir `gensim`, eine nützliche Python-Bibliothek für Topic-Modelling.

Zuerst müssen wir uns ein vortrainiertes Modell holen. Eine Reihe vortrainierter Modelle steht im Internet zum Download bereit, von denen wir uns für das von Google News entscheiden. Dieses umfasst Einbettungen für 3 Millionen Wörter und wurde mit etwa 100 Milliarden Wörtern aus dem Google-News-Archiv trainiert. Das Herunterladen dauert eine Weile, daher speichern wir die Datei lokal:

```
MODEL = 'GoogleNews-vectors-negative300.bin'
path = get_file(MODEL + '.gz',
    'https://s3.amazonaws.com/dl4j-distribution/%s.gz' % MODEL)
unzipped = os.path.join('generated', MODEL)
if not os.path.isfile(unzipped):
    with open(unzipped, 'wb') as fout:
        zcat = subprocess.Popen(['zcat'],
                                stdin=open(path),
                                stdout=fout
                               )
        zcat.wait()
```

```
Downloading data from GoogleNews-vectors-negative300.bin.gz
1647050752/1647046227 [==============================] - 71s 0us/step
```

Nach dem Downloaden des Modells können wir es in den Speicher laden. Das Modell ist ziemlich groß und benötigt an Speicher etwa 5 GByte RAM:

```
model = gensim.models.KeyedVectors.load_word2vec_format(MODEL, binary=True)
```

Sobald das Modell fertig geladen ist, können wir es verwenden, um ähnliche Wörter zu finden:

```
model.most_similar(positive=['espresso'])
```

```
[(u'cappuccino', 0.6888186931610107),
 (u'mocha', 0.6686209440231323),
 (u'coffee', 0.6616827249526978),
 (u'latte', 0.6536752581596375),
 (u'caramel_macchiato', 0.6491267681121826),
 (u'ristretto', 0.6485546827316284),
 (u'espressos', 0.6438628435134888),
 (u'macchiato', 0.6428250074386597),
 (u'chai_latte', 0.6308028697967529),
 (u'espresso_cappuccino', 0.6280542612075806)]
```

Diskussion

Worteinbettungen ordnen jedem Wort aus dem Vokabular einen *n*-dimensionalen Vektor zu, sodass ähnliche Wörter nahe beieinanderliegen. Ähnliche Wörter können dann einfach per Nächste-Nachbarn-Suche gefunden werden, für die es selbst in hochdimensionalen Räumen effiziente Algorithmen gibt.

Einfach gesagt, werden die Word2vec-Einbettungen beim Trainieren eines neuronalen Netzes zum Vorhersagen eines Worts aus dem Kontext erzeugt. Wir lassen also das Netzwerk vorhersagen, welches Wort es für X in einer Reihe von Fragmenten verwenden würde, zum Beispiel »das Café servierte mir einen X, der mich so richtig aufweckte«.

Auf diese Weise erhalten Wörter, die in ähnliche Muster eingesetzt werden können, Vektoren, die nahe beieinanderliegen. Wir kümmern uns hier nicht um die eigentliche Aufgabenstellung des Netzwerks, sondern nur um die zugewiesenen Gewichte, die wir als Nebeneffekt beim Trainieren dieses Netzwerks erhalten.

Später in diesem Buch erfahren Sie auch, wie Sie Wörter mithilfe von Worteinbettungen in ein neuronales Netz einspeisen können. Ein 300-dimensionaler Einbettungsvektor ist wesentlich geeigneter, um in ein Netzwerk eingespeist zu werden, als ein One-Hot-kodierter Vektor mit 3 Millionen Dimensionen. Darüber hinaus muss ein Netzwerk, das mit vortrainierten Worteinbettungen versorgt wird, nicht zuerst die Beziehungen zwischen den Wörtern erlernen, sondern kann sofort mit der eigentlichen Aufgabenstellung beginnen.

3.2 Word2vec-Mathematik

Problem

Wie können Sie Fragestellungen der Form »A verhält sich zu B wie C zu ...« automatisch beantworten?

Lösung

Verwenden Sie die semantischen Eigenschaften des Word2vec-Modells. Mit der `gensim`-Bibliothek geht das ziemlich einfach:

```
def A_is_to_B_as_C_is_to(a, b, c, topn=1):
  a, b, c = map(lambda x:x if type(x) == list else [x], (a, b, c))
  res = model.most_similar(positive=b + c, negative=a, topn=topn)
  if len(res):
    if topn == 1:
      return res[0][0]
    return [x[0] for x in res]
  return None
```

Wir können dies nun auf beliebige Wörter anwenden – zum Beispiel um herauszufinden, was sich zu »king« so verhält wie »man« zu »woman«:

```
A_is_to_B_as_C_is_to('man', 'woman', 'king')
```

```
u'queen'
```

Dieser Ansatz ist ebenfalls dafür geeignet, Hauptstädte bestimmter Länder abzufragen:

```
for country in 'Italy', 'France', 'India', 'China':
    print('%s is the capital of %s' %
          (A_is_to_B_as_C_is_to('Germany', 'Berlin', country), country))
```

```
Rome is the capital of Italy
Paris is the capital of France
Delhi is the capital of India
Beijing is the capital of China
```

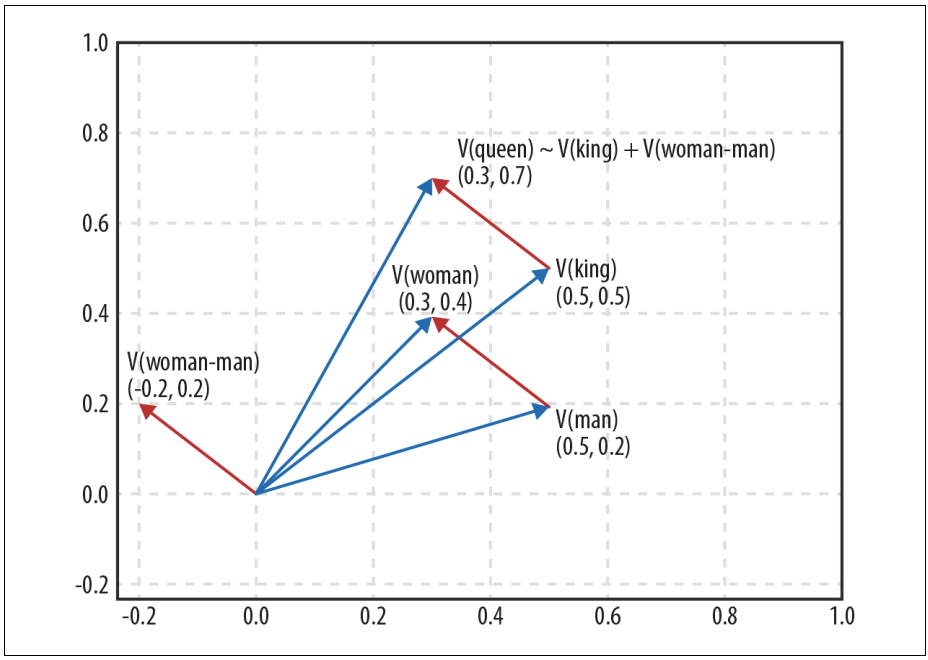
oder um die Hauptprodukte von Unternehmen zu finden (beachten Sie den Platzhalter ###, der in dieser Einbettung für sämtliche Zahlen verwendet wird):

```
for company in 'Google', 'IBM', 'Boeing', 'Microsoft', 'Samsung':
  products = A_is_to_B_as_C_is_to(
    ['Starbucks', 'Apple'], ['Starbucks_coffee', 'iPhone'], company, topn=3)
  print('%s -> %s' %
        (company, ', '.join(products)))
```

```
Google -> personalized_homepage, app, Gmail
IBM -> DB2, WebSphere_Portal, Tamino_XML_Server
Boeing -> Dreamliner, airframe, aircraft
Microsoft -> Windows_Mobile, SyncMate, Windows
Samsung -> MM_A###, handset, Samsung_SCH_B###
```

Diskussion

Wie im vorherigen Schritt gesehen, kodieren die den Wörtern zugeordneten Vektoren die Bedeutung der Wörter – ähnliche Wörter haben Vektoren, die nahe beieinanderliegen. Es stellt sich heraus, dass der Unterschied zwischen Wortvektoren auch den Unterschied zwischen Wortbedeutungen darstellt. Wenn wir vom Vektor für das Wort »man« den Vektor für das Wort »woman« abziehen, erhalten wir mit der Differenz etwas, das wie »von männlich zu weiblich wechseln« interpretiert werden kann. Wenn wir diese Differenz zum Wortvektor für »king« addieren, landen wir in der Nähe des Vektors für das Wort »queen«:



Die `most_similar`-Funktion nimmt ein oder mehrere positive Wörter und ein oder mehrere negative Wörter als Argumente. Die Funktion sucht dann die entspre-

chenden Vektoren heraus, zieht die negativen von den positiven ab und gibt die Wörter aus, die dem Ergebnisvektor am nächsten liegen.

Um also die Frage »A verhält sich zu B wie C zu ...?« zu beantworten, ziehen wir A von B ab und addieren C, oder wir rufen die `most_similar`-Funktion auf mit `positive = [B, C]` und `negative = [A]`. Das Beispiel `A_is_to_B_as_C_is_to` erweitert dieses Verhalten um zwei kleine Eigenschaften. Wenn wir nur ein Beispiel abfragen, wird lediglich ein Item ausgegeben statt einer Liste mit einem Item. Des Weiteren können wir entweder Listen oder einzelne Items für A, B und C angeben.

Die Möglichkeit, Listen in die Funktion stecken zu können, erweist sich in einem Beispiel mit Produkten als nützlich. Wir fragen drei Produkte pro Unternehmen ab, wodurch es wichtiger ist, den Vektor genau richtig zu bestimmen, als wenn wir nur ein Produkt abfragen würden. Indem wir mit »Starbucks« und »Apple« der Funktion zwei Beispiele geben, erhalten wir einen genaueren Vektor für das Konzept »ist ein Produkt von«.

3.3 Worteinbettungen visualisieren

Problem

Sie möchten einen Einblick darin erhalten, wie Worteinbettungen eine Objektmenge aufteilen.

Lösung

Ein 300-dimensionaler Raum ist nur schwer zu erkunden, aber wir können einen Algorithmus namens *t-distributed Stochastic Neighbor Embedding* (t-SNE) dazu verwenden, einen höherdimensionalen Raum in etwas Anschaulicheres wie zwei Dimensionen umzuwandeln.

Angenommen, wir möchten sehen, wie drei Gruppen von Begriffen aufgeteilt werden. Wir nehmen Länder, Sportarten und Getränke.

```
beverages = ['espresso', 'beer', 'vodka', 'wine', 'cola', 'tea']
countries = ['Italy', 'Germany', 'Russia', 'France', 'USA', 'India']
sports = ['soccer', 'handball', 'hockey', 'cycling', 'basketball', 'cricket']

items = beverages + countries + sports
```

Nun betrachten wir die dazugehörigen Vektoren:

```
item_vectors = [(item, model[item])
                    for item in items
                    if item in model]
```

Wir können t-SNE verwenden, um die Cluster im 300-dimensionalen Raum zu finden:

```
vectors = np.asarray([x[1] for x in item_vectors])
lengths = np.linalg.norm(vectors, axis=1)
norm_vectors = (vectors.T / lengths).T
tsne = TSNE(n_components=2, perplexity=10,
            verbose=2).fit_transform(norm_vectors)
```

Mithilfe von matplotlib stellen wir die Ergebnisse schön in einem Streudiagramm dar:

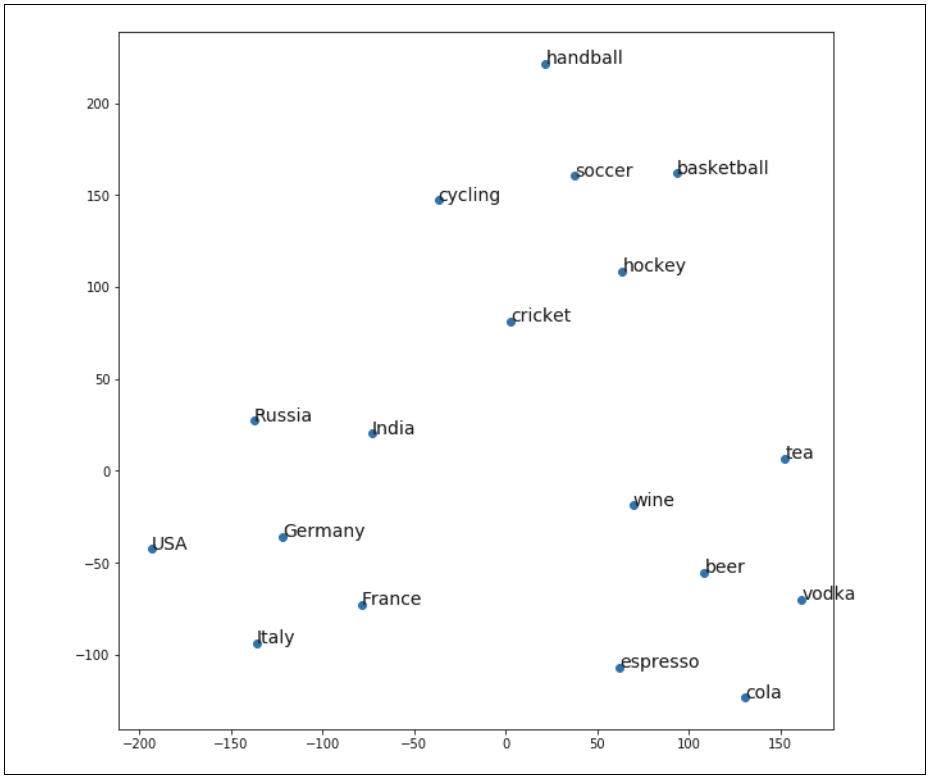
```
x=tsne[:,0]
y=tsne[:,1]

fig, ax = plt.subplots()
ax.scatter(x, y)

for item, x1, y1 in zip(item_vectors, x, y):
    ax.annotate(item[0], (x1, y1))

plt.show()
```

Hier ist das Ergebnis:



Diskussion

t-SNE ist ein ausgeklügelter Algorithmus; man gibt ihm eine Menge an Punkten in einem hochdimensionalen Raum, und er versucht iterativ, die beste Projektion auf einen niedriger dimensionalen Raum (meist eine Ebene) zu finden, der die Abstände zwischen den Punkten so gut wie möglich erhält. t-SNE eignet sich daher sehr gut zur Visualisierung höherer Dimensionen wie bei (Wort)Einbettungen.

In etwas komplexeren Situationen kann man mit dem `perplexity` Parameter etwas experimentieren. Dieser Parameter bestimmt grob das Verhältnis zwischen lokaler Genauigkeit und Gesamtgenauigkeit. Ein niedriger Wert erzeugt kleine Cluster, die lokal genau sind; ein hoher Wert führt lokal zu mehr Verzerrungen, aber dafür zu besseren Gesamtclustern.

3.4 Objektklassen in Einbettungen finden

Problem

In hochdimensionalen Räumen gibt es oft Unterräume, die nur Objekte einer bestimmten Klasse enthalten. Wie kann man diese Räume finden?

Lösung

Wenden Sie eine *Support Vector Machine* (SVM) auf eine Menge von Beispielen und Gegenbeispielen an. Lassen Sie uns beispielsweise die Länder im Word2vec-Raum suchen. Zunächst laden wir wieder das Modell und suchen dann die einem bestimmten Land ähnlichen Dinge, in diesem Fall Deutschland:

```
model = gensim.models.KeyedVectors.load_word2vec_format(MODEL, binary=True)
model.most_similar(positive=['Germany'])
```

```
[(u'Austria', 0.7461062073707581),
 (u'German', 0.7178748846054077),
 (u'Germans', 0.6628648042678833),
 (u'Switzerland', 0.6506867408752441),
 (u'Hungary', 0.6504981517791748),
 (u'Germnay', 0.649348258972168),
 (u'Netherlands', 0.6437495946884155),
 (u'Cologne', 0.6430779099464417)]
```

Wie Sie sehen können, gibt es eine Reihe von Ländern in der Nähe, aber es tauchen auch Wörter wie »German« oder Namen deutscher Städte in der Liste auf. Wir könnten versuchen, einen Vektor zu erstellen, der bestmöglich das Konzept »Land« repräsentiert, indem wir die Vektoren vieler Länder zusammenaddieren, statt nur Deutschland zu verwenden. Aber auch das bringt uns nicht sehr viel weiter. Das Konzept »Land« ist im Einbettungsraum kein Punkt, sondern eine Figur. Wir benötigen daher einen richtigen Klassifikator.

Für solche Klassifikationsaufgaben haben sich Support Vector Machines als sehr effektiv erwiesen. scikit-learn bietet dafür eine sehr leicht implementierbare Lösung. Der erste Schritt ist das Erstellen eines Trainingsdatensatzes. Positive Beispiele für dieses Rezept zu finden, ist einfach, da es nur eine gewisse Anzahl an Ländern gibt:

```
positive = ['Chile', 'Mauritius', 'Barbados', 'Ukraine', 'Israel',
  'Rwanda', 'Venezuela', 'Lithuania', 'Costa_Rica', 'Romania',
  'Senegal', 'Canada', 'Malaysia', 'South_Korea', 'Australia',
  'Tunisia', 'Armenia', 'China', 'Czech_Republic', 'Guinea',
  'Gambia', 'Gabon', 'Italy', 'Montenegro', 'Guyana', 'Nicaragua',
  'French_Guiana', 'Serbia', 'Uruguay', 'Ethiopia', 'Samoa',
  'Antarctica', 'Suriname', 'Finland', 'Bermuda', 'Cuba', 'Oman',
  'Azerbaijan', 'Papua', 'France', 'Tanzania', 'Germany' ... ]
```

Je mehr positive Beispiele, desto besser, aber 40 bis 50 sollten für dieses Beispiel genügen, um uns eine gute Vorstellung davon zu geben, wie die Lösung funktioniert.

Wir benötigen ebenfalls einige negative Beispiele. Diese entnehmen wir direkt dem allgemeinen Wortschatz des Word2vec-Modells. Mit etwas Pech könnten wir dabei ein Land erwischen und es in die negativen Beispiele aufnehmen, aber da es 3 Millionen Wörter in dem Modell und nur knapp 200 Länder auf der Welt gibt, müssten wir dafür schon sehr viel Pech haben.

```
negative = random.sample(model.vocab.keys(), 5000)
negative[:4]
```

```
[u'Denys_Arcand_Les_Invasions',
 u'2B_refill',
 u'strained_vocal_chords',
 u'Manifa']
```

Jetzt erstellen wir basierend auf den positiven und negativen Beispielen einen Trainingsdatensatz mit Labels. Wir verwenden `1` als Label für Länder und `0` für alles andere. Wir folgen der üblichen Vorgehensweise, die Trainingsdaten in der Variablen `X` und die Labels in der Variablen `y` zu speichern:

```
labelled = [(p, 1) for p in positive] + [(n, 0) for n in negative]
random.shuffle(labelled)
X = np.asarray([model[w] for w, l in labelled])
y = np.asarray([l for w, l in labelled])
```

Jetzt geht es ans Trainieren des Modells. Wir legen einen Teil der Daten zur Evaluation unseres Modells zur Seite:

```
TRAINING_FRACTION = 0.7
cut_off = int(TRAINING_FRACTION * len(labelled))
clf = svm.SVC(kernel='linear')
clf.fit(X[:cut_off], y[:cut_off])
```

Das Trainieren sollte selbst auf einem nicht sehr rechenstarken Computer blitzschnell geschehen, da unser Datensatz recht klein ist. Wir können die Performance

unseres Modells überprüfen, indem wir einen Blick darauf werfen, wie viele Vorhersagen auf dem Evaluationsdatensatz richtig sind.

```
res = clf.predict(X[cut_off:])

missed = [country for (pred, truth, country) in
          zip(res, y[cut_off:], labelled[cut_off:]) if pred != truth]
100 - 100 * float(len(missed)) / len(res), missed
```

Ihre Ergebnisse hängen ein wenig davon ab, welche Länder Sie als positive Beispiele gewählt haben und welche negativen Beispiele zufällig gezogen wurden. Ich erhalte meistens einige Länder, die das Modell übersehen hat – typischerweise weil der Ländername darüber hinaus eine andere Bedeutung hat, wie etwa »Jordan«, aber es gibt auch einige echte Fehler. Die Genauigkeit sollte etwa bei 99,9 % liegen.

Jetzt können wir den Klassifikator über alle Wörter laufen lassen, um die Länder zu ermitteln:

```
res = []
for word, pred in zip(model.index2word, all_predictions):
  if pred:
    res.append(word)
    if len(res) == 150:
      break
random.sample(res, 10)
```

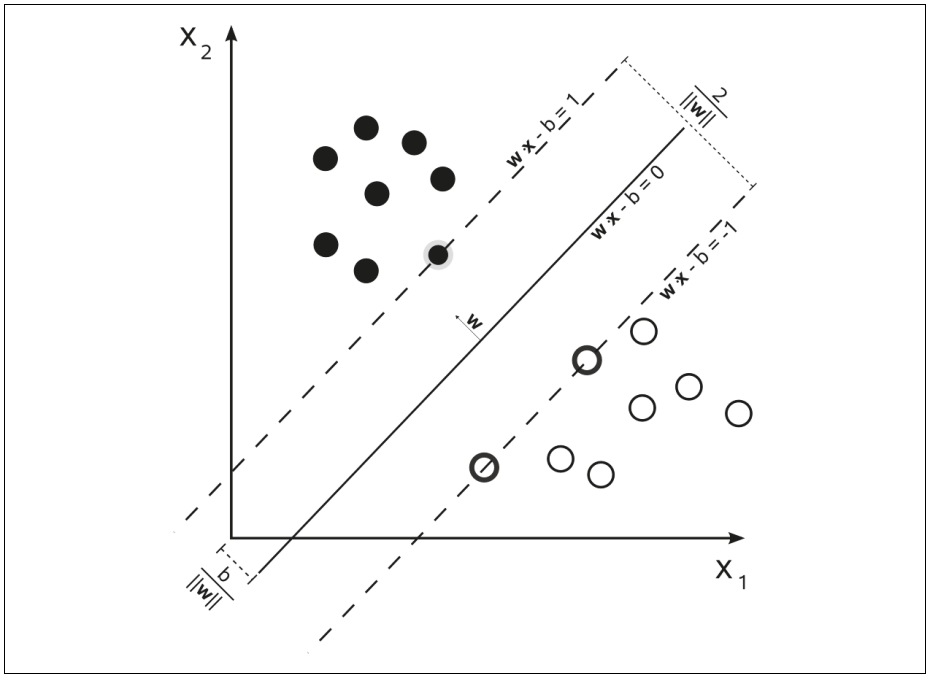
```
[u'Myanmar',
 u'countries',
 u'Sri_Lanka',
 u'Israelis',
 u'Australia',
 u'Pyongyang',
 u'New_Hampshire',
 u'Italy',
 u'China',
 u'Philippine']
```

Die Ergebnisse sind ziemlich gut, aber nicht perfekt. Beispielsweise wird das Wort »countries« selbst als Land klassifiziert, ebenso Objekte wie Kontinente oder US-Bundesstaaten.

Diskussion

Support Vector Machines sind sehr effektiv darin, Klassen in einem höherdimensionalen Raum wie Worteinbettungen zu finden. Sie tun dies, indem sie versuchen, Hyperebenen zu finden, die die positiven von den negativen Beispielen trennen.

Länder in Word2vec sind alle einigermaßen nahe beieinander, da sie semantische Gemeinsamkeiten haben. SVMs helfen uns dabei, die Ansammlung der Länder zu finden und dafür Grenzen zu definieren. Das folgende Diagramm veranschaulicht das im zweidimensionalen Raum:



SVMs können für alle Arten von Ad-hoc-Klassifikatoren im Machine Learning verwendet werden, da sie wirkungsvoll sind, selbst wenn die Anzahl an Dimensionen größer ist als die Anzahl an Daten, wie in diesem Fall. Mit den 300 Dimensionen könnte das Modell die Daten auswendig lernen (Overfitting). Aber da die SVM versucht, ein einfaches Modell an die Daten anzupassen, können wir selbst von einem Datensatz mit nur ein paar Dutzend Beispielen gut verallgemeinern.

Die erzielten Ergebnisse sind ziemlich gut, obwohl man anmerken muss, dass eine Genauigkeit von 99,7 % bei 3 Millionen Negativbeispielen immer noch zu 9.000 falsch Positiven führen würde, die die eigentlichen Länder weit übersteigen würden.

3.5 Semantische Abstände innerhalb einer Klasse berechnen

Problem

Wie findet man die nach einem bestimmten Kriterium relevantesten Elemente einer Klasse?

Lösung

In einer Klasse, zum Beispiel Länder, können wir die Elemente dieser Klasse mithilfe der relativen Abstände nach einem bestimmten Auswahlkriterium ordnen:

```
country_to_idx = {country['name']: idx for idx, country in enumerate(countries)}
country_vecs = np.asarray([model[c['name']] for c in countries])
country_vecs.shape
```

```
(184, 300)
```

Wir können nun, wie zuvor, ein numpy-Array mit den Vektoren für die Länder erstellen:

```
countries = list(country_to_cc.keys())
country_vecs = np.asarray([model[c] for c in countries])
```

Zur raschen Überprüfung schauen wir, welche Länder Kanada am ähnlichsten sind:

```
dists = np.dot(country_vecs, country_vecs[country_to_idx['Canada']])
for idx in reversed(np.argsort(dists)[-8:]):
    print(countries[idx], dists[idx])
```

```
Canada 7.5440245
New_Zealand 3.9619699
Finland 3.9392405
Puerto_Rico 3.838145
Jamaica 3.8102934
Sweden 3.8042784
Slovakia 3.7038736
Australia 3.6711009
```

Die karibischen Länder überraschen hier etwas, und viele Nachrichten über Kanada müssen etwas mit Eishockey zu tun haben, da die Slowakei und Finnland in der Liste auftauchen. Aber ansonsten schaut die Liste relativ plausibel aus.

Probieren wir nun etwas anderes aus, nämlich die Länder nach der Nähe zu einem beliebigen Begriff zu ordnen. Dafür berechnen wir den Abstand von jedem Landesnamen zu dem Begriff, nach dem wir die Länder ordnen möchten. Länder, die dem Begriff »näher« sind, sind im Zusammenhang mit dem Begriff relevanter:

```
def rank_countries(term, topn=10, field='name'):
    if not term in model:
        return []
    vec = model[term]
    dists = np.dot(country_vecs, vec)
    return [(countries[idx][field], float(dists[idx]))
            for idx in reversed(np.argsort(dists)[-topn:])]
```

Zum Beispiel:

```
rank_countries('cricket')
```

```
[('Sri_Lanka', 5.92276668548584),
 ('Zimbabwe', 5.336524486541748),
 ('Bangladesh', 5.192488670349121),
 ('Pakistan', 4.948408126831055),
 ('Guyana', 3.9162840843200684),
 ('Barbados', 3.757995128631592),
 ('India', 3.7504401206970215),
 ('South_Africa', 3.6561498641967773),
 ('New_Zealand', 3.642028331756592),
 ('Fiji', 3.608567714691162)]
```

Da das Word2vec-Modell auf Google News basiert, werden die Länder am höchsten eingeordnet, die im Zusammenhang mit dem angegebenen Begriff am häufigsten in den Nachrichten erschienen sind. Indien ist womöglich im Zusammenhang mit Cricket am meisten erwähnt worden, aber solange auch jede Menge anderes über Indien geschrieben wurde, kann Sri Lanka trotzdem als am relevantesten eingestuft werden.

Diskussion

In Räumen, in denen Elemente verschiedener Klassen auf die gleichen Dimensionen abgebildet werden, können wir Kreuzklassenabstände als Ähnlichkeitsmaß verwenden. Word2vec kann nicht wirklich einen konzeptionellen Raum darstellen (das Wort »Jordan« kann sich beispielsweise auf den Fluss, das Land oder eine Person beziehen), aber es ist immerhin gut genug, um Länder nach ihrer Relevanz für verschiedene Begriffe sinnvoll zu ordnen.

Ein ähnlicher Ansatz wird oft beim Erstellen von Empfehlungssystemen verfolgt. Beim Netflix-Wettbewerb war beispielsweise die Strategie beliebt, mithilfe der Filmratings von Usern Filme und User in einen gemeinsamen Raum abzubilden. Von Filmen, die nahe bei einem User liegen, wird dann erwartet, dass sie vom User hoch bewertet werden.

Wenn wir zwei Räume haben, die nicht gleich sind, können wir diesen Trick trotzdem anwenden, falls wir die Projektionsmatrix berechnen können, um von einem Raum zum anderen zu gelangen. Dies ist dann möglich, wenn wir genügend Elemente haben, deren Position wir in beiden Räumen kennen.

3.6 Länderdaten auf einer Landkarte visualisieren

Problem

Wie kann man die Länderrangliste eines Projekts auf einer Landkarte visualisieren?

Lösung

GeoPandas ist das perfekte Hilfsmittel, um numerische Daten auf einer Landkarte zu visualisieren.

Diese ausgefeilte Bibliothek kombiniert die Stärke von Pandas mit geografischen Elementen und bietet einige vorinstallierte Karten an. Im Folgenden laden wir die Karte der gesamten Welt:

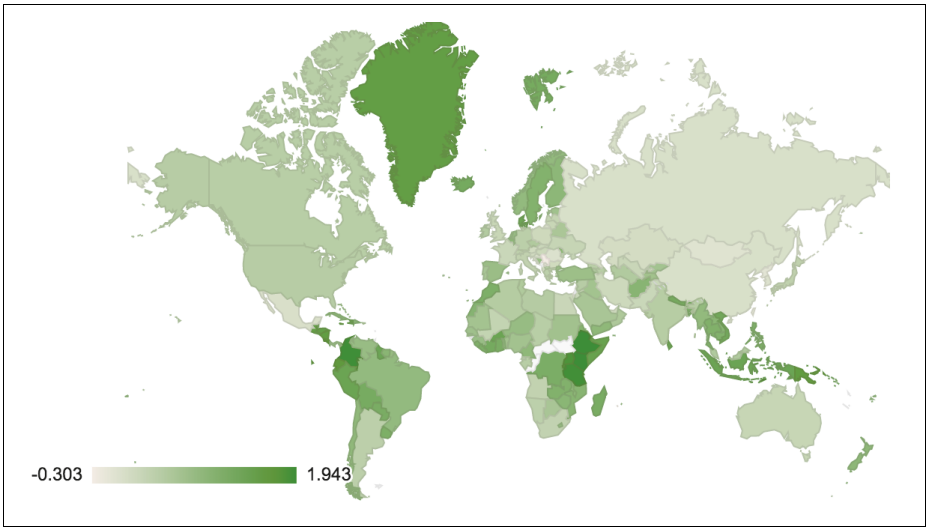
```
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
world.head()
```

Dies zeigt uns einige grundlegende Informationen über eine Reihe von Ländern. Wir können dem `world`-Objekt eine Spalte basierend auf unserer `rank_countries`-Funktion hinzufügen:

```
def map_term(term):
    d = {k.upper(): v for k, v in rank_countries(term,
                                                 topn=0,
                                                 field='cc3')}
    world[term] = world['iso_a3'].map(d)
    world[term] /= world[term].max()
    world.dropna().plot(term, cmap='OrRd')

map_term('coffee')
```

Dies zeichnet beispielsweise eine recht anschauliche Karte für Kaffee, in der die Kaffee konsumierenden Länder und die Kaffee produzierenden Länder hervorgehoben werden.



Diskussion

Das Visualisieren von Daten ist eine wichtige Aufgabe im Machine Learning. Indem wir die Daten näher betrachten – seien es die Eingabedaten oder die Ergebnisse eines Algorithmus –, können wir schnell Anomalien erkennen. Trinken Menschen in Grönland wirklich so viel Kaffee? Oder sehen wir hier vielleicht eine Verzerrung aufgrund des »Greenlandic Coffee« (einer Variation des Irish Coffee)? Was ist mit den Ländern in der Mitte Afrikas – stimmt es, dass sie weder Kaffee trinken noch produzieren? Oder haben wir vielleicht einfach keine Daten für diese Länder, weil sie nicht in unseren Einbettungen vorkommen?

GeoPandas ist das perfekte Hilfsmittel, um geografische Informationen zu analysieren. Es baut auf den allgemeinen Funktionen zur Datenverarbeitung von Pandas auf, mit denen wir uns näher in Kapitel 6 beschäftigen werden.