Build a text analysis app in 3 hours with Word2vec

Put unstructured text data to work for improved insight and decision-making

Introduction

Between the Internet of Things, customer experience and loyalty programs, social network monitoring, connected enterprise systems and other information sources, today's organizations have access to more data than they ever had before—and frankly, more than they may know what to do with. The challenge is to not just understand that data, but actualize it and use it to recognize real business value.

It's a difficult endeavor, in part because the majority of business-relevant information originates in unstructured formats, primarily text. This data is unusable until it can be "read" by analytics systems and applied by people to solve problems. Mathematical exploration of text data can make it more readable, yielding insights that translate into better decisions by marketers, entrepreneurs, doctors, teachers and others.

This ebook will walk you through a sample scenario with Albert, a data scientist who wants to put text analytics to work by using the Word2vec algorithm and other data science tools.

Along the way, you'll learn some tips and tricks for working with Word2vec and applying it to analytical models. **Let's get started.**

What is Word2vec?

Word2vec uses machine learning techniques to come up with vector representations of words. Associating vectors to words is a practical way to measure the distance between them. The angle formed between two vectors is a commonly used metric to measure the similarity of the vectors. If the two legs of the angle are close together, then the two words are frequently mentioned together. If the legs of the angle are farther apart, then the two words are not associated as frequently. The angle between vectors is called the cosine distance and it provides a numerical value that applications can analyze and interpret.

Albert’s challenge

Albert is a data scientist working for a large retailer. His job is to help his company get the most out of its data for maximum business advantage.

Albert’s company is introducing a major new product line, and the marketing department is planning a big launch campaign to promote it. But while the marketing team has traditional business intelligence solutions, it doesn’t have a way to rapidly assess how the campaign is performing as it unfolds. The marketers need to know how the company and its products are being perceived in the marketplace in near-real time so they can adjust the campaign elements for optimum results.

Also, the campaign starts in a couple of weeks—so Albert needs to come up with a solution that doesn’t take a long time to develop.

Getting started

What you'll learn

- How to select the elements needed to build a text analysis model

Problem to solve: Brand and product perception

Is the public developing a positive impression of the company and its new product line? Albert decided to use social media as a way to obtain this information, knowing that it is constantly updated and provides a large potential pool of data.

Data source: Tweets

Albert chose Twitter feeds as the specific data source because they reflect changes in public opinion and are easy to obtain in large quantities. He used the Twitter Decahose API, which continuously provides about a 10 percent sample of public tweets.

Analytics requirements: Text analysis

The analysis needed to show in a mathematically valid way what kinds of words were being associated with the new product on Twitter—good, bad and so on. Tweets take the form of unstructured text, which can't be mined in the same traditional way as structured databases. To meet the marketers' deadline, Albert needed tools he could use to rapidly build a suitable text analysis model and structure for processing tweets. Albert also wanted a model versatile enough to use for multiple purposes as the needs of his company changed or new departments wanted to analyze text data.

Data science tools: Apache Spark, Python, Word2vec

Albert chose the Apache Spark open source framework as the backbone for processing and centralizing his data. He selected Spark because he wanted to be able to perform computations in a distributed fashion, reading data from multiple different nodes.

This approach allowed him to use large training sets and perform computations in a reasonable amount of time. Because Spark is not tied to a particular storage type or format, it can handle a wide variety of data sources. This may be useful later if Albert uses his model to run different analyses with different text data.

Albert also chose the Python programming language, which is widely used with Spark, to write his application. He liked the fact that Python is easy to use yet powerful enough for professionals. His critical decision, however, was to use the Word2vec algorithm implemented in Spark to build and train his model.

What's the big deal about Word2vec?

To make data useful for building his model, Albert needed to transform it into vector arrays that show the mathematical relationships between words. Given the short time frame and the complexity of the problem he was trying to solve, Albert couldn't have done this from scratch. Fortunately, Word2vec is one of the quickest and easiest tools to use.

Behind this ease of use is some very complex mathematics: Word2vec word vectors capture linguistic similarities (Figure 1). For example: *vec('king') + vec('woman') - vec('man') = vec('queen')*

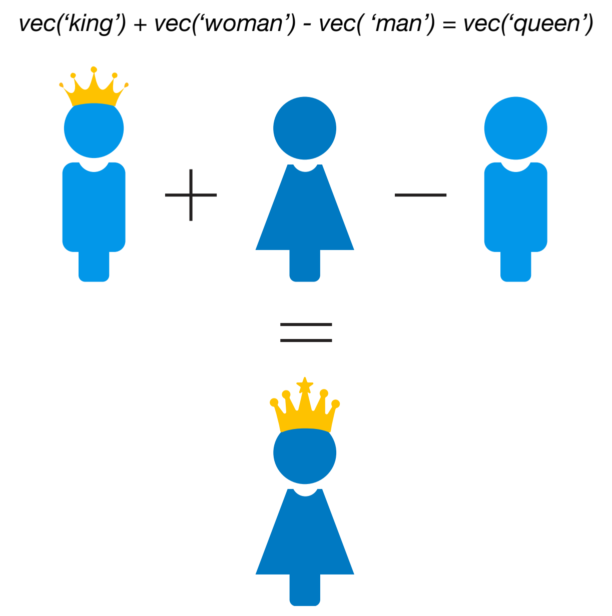
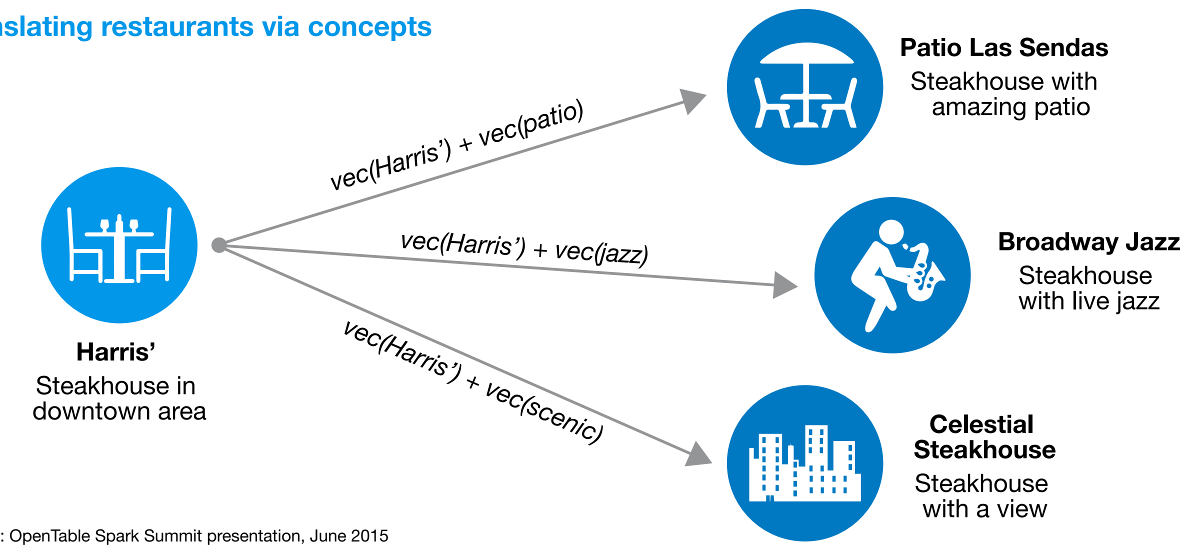


Figure 1. Example of linguistic similarities.

Given a search word, Word2vec can relate the word with other terms in Twitter. For example, the OpenTable app uses Word2vec to recommend restaurants based on reviews from users.[1] Word2vec assigns a vector to each word in the reviews, including restaurant names and other keywords such as “Steak,” “Patio,” “Jazz” or “View” (Figure 2).

It only takes a little imagination to see that Word2vec models can be useful for many similar purposes, such as associating tweeted words with the name of a product or service. Hashtags (#), handles (@) and even emoticons (☺) can be included as words.

Translating restaurants via concepts


Harris’
Steakhouse in downtown area
vec(Harris’) + vec(patio)
vec(Harris’) + vec(jazz)
vec(Harris’) + vec(scenic)
Patio Las Sendas
Steakhouse with amazing patio
Broadway Jazz
Steakhouse with live jazz
Celestial Steakhouse
Steakhouse with a view

Source: OpenTable Spark Summit presentation, June 2015

Figure 2. Word2vec can be used to recommend restaurants based on words used in reviews.

Preparing the data

What you'll learn

- How to filter, clean and parse the data to get it ready for use

To build his Word2vec model, Albert first loaded his data into the Spark framework, reading about 300,000 tweets every 15 minutes. He filtered the data in Spark and used Python to interact with Spark for cleaning and parsing the data. As he proceeded with these steps, he stored the relevant code in a github repo. The code for each step is shown in the following sections.

Filtering

Albert decided to keep his application focused on the English language for the time being, so he set the filter to retain only tweets in English. That reduced the number of tweets to about 40 percent of the original total. Albert also decided to write up a list of words that he thought were relevant for his analysis, and use it to filter out any tweets in English that did not contain at least one of the words on his list.

The relevant code:

```
# Construct SQL Command
t0 = time.time()
sqlString = "("
for substr in filter: #iteration on the list of words to filter (usually 50-100 words)
   sqlString = sqlString+"text LIKE '%"+substr+"%' OR "
   sqlString = sqlString+”text LIKE '%"+substr.upper()+"%' OR "
sqlString=sqlString[:-4]+")"
sqlFilterCommand = "SELECT lang, text FROM tweets WHERE (lang = 'en') AND "+sqlString

# Query tweets in english that contain at least one of the keywords
tweetsDF = sqlContext.sql(sqlFilterCommand).cache()
twf = tweetsDF.count()
```

Cleaning and parsing

Parsing breaks sentences into separate words. Cleaning removes dots, commas and stop words such as “the” that get in the way of processing. For example, without cleaning, the word “apple” and the word “apple,” (with a comma after it) could both appear in the same data set and be counted by the model as two different words.

The code:

```
# select the text from the tweets and map it to an rdd
tweetsRDD = tweetsDF.select(‘text’).rdd

def parseAndRemoveStopWords(text):
   t = text[0].replace(“;”,” “).replace(“:”,” “).replace(‘”’,’ ‘).replace(‘-’,’ ‘)
   t = t.replace(‘,’,’ ‘).replace(‘.’,’ ‘)
   t = t.lower().split(“ “)
   stop = stopwords.words(‘english’)
   return [i for i in t if i not in stop]

# map rdd of raw text to clean and parsed text without stop words
tw = tweetsRDD.map(parseAndRemoveStopWords)
```

Building the model

What you'll learn

- How to filter, clean and parse the data to get it ready for use

With the data loaded and prepared, it was now time for Albert to build and train the model. He began by setting important Word2vec parameters.

Data size: Generally, more data is better. Your limitations, however, include the computational power available and the time you want to spend on each experiment. Albert started with a small batch of tweets to make sure his PySpark script was working correctly. For his final model, Albert decided to train a model using about 6.2 TB of Twitter data because adding more tweets did not change the results of the model.

Vector dimensionality: Each word is represented by a vector, and the user is allowed to choose its dimensionality. Albert used the default setting of 100 dimensions because adding more dimensionality did not add accuracy to the results of the model. Also, adding more dimensionality would lengthen the time needed to train the model and to perform operations with the word vectors.

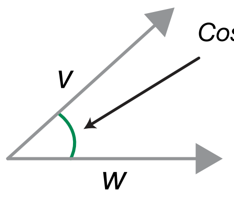
minCount: This is the number of times that a word needs to appear in the tweets to be considered in the model. A low setting such as 2 or 3 lets in unusual words that must be included in a model, but may also let in other words that are not desired. A high setting of 15 or 20 excludes words that do not appear very often.

Albert performed experiments with variations of this parameter and realized that 5 was a good choice to make sure all words of interest were shown in the final model.

The complexity of the Spark implementation of Word2vec is *O(log(V)),* where V is the number of words in the model. In other words, the fewer the words used, the faster Word2vec trains. See http://spark.apache.org/docs/latest/mllib-feature-extraction.html#model for more details.

Return closest n words: Given a search term, how many other similar terms in Twitter should be listed? Albert decided the number of words to show would depend on the final application. For visualization purposes he decided to use the 20 closest terms to avoid overwhelming the plot.

Albert then ran multiple iterations to test the parameters and tune them for optimum performance using a search word that he knew would easily produce associations. Each time he fed the data into Word2vec, it returned a data frame with words and vectors (Figure 3).



$$\text{Cosine distance } (v,w) = \frac{v^T w}{||v||_2 ||w||_2}$$

Closest words to #shopping
1. #style
2. #decor
3. #casual
4. #homedecor
5. #etsy

Figure 3. Distance between words mathematically calculated by Word2vec.

These experiments showed how the quality of results changed as Albert increased or decreased different parameters. The last iteration provided the final model.

The code:

```
# map to df
twDF = tw.map(lambda p: Row(text=p)).toDF()

# default minCount = 5 (we may need to try something larger: 20-100 to reduce the training time)
# default vectorSize = 100 (we may want to increase dimensionality when it increases model's quality)
t0 = time.time()
word2Vec = Word2Vec(vectorSize=100, minCount=5, inputCol="text", outputCol="result")
modelW2V = word2Vec.fit(twDF)
wordVectorsDF = modelW2V.getVectors()
print "Elapsed time (seconds) to train Word2Vec: ", time.time() - t0
```

Word2vec is an easy-to-use algorithm with highly complex math behind the scenes that is extremely useful in practice. And it's fast: reading the data, preparing the data and building and training the model took Albert **just 3.1 hours on a 10-node cluster using 6.2 TB of data.**

Visualizing the analysis

What you'll learn

- How to visualize Word2vec analytic results in different ways

Word2vec delivers results using numbers. But Albert's application users—the marketing teams and their clients—may not be numbers people. To help them visualize the results of the analysis, Albert used the D3 JavaScript library to show a force graph with the word of interest at the center (Figure 4). The graph uses bubbles to show how many times each word or hashtag appears (indicated by bubble size) and how close or distant they are from the original search term (located in the center).

Albert knew he could use the Principal Components Analysis (PCA) procedure to reduce the number of Word2vec dimensions. Dimensionality reduction enabled Albert to

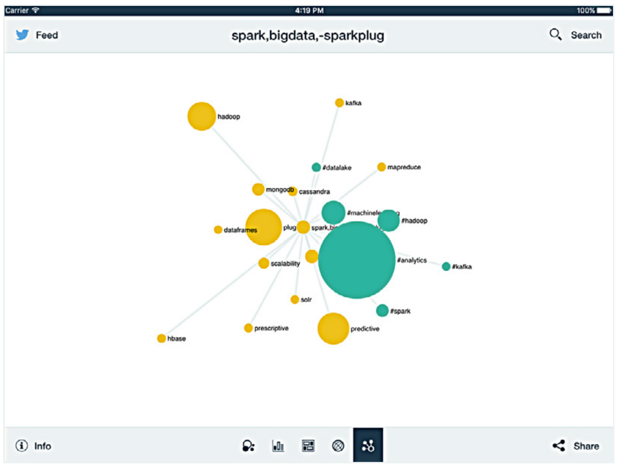


Figure 4. Word2vec force graph of nearest terms (green for hashtags, yellow for words).

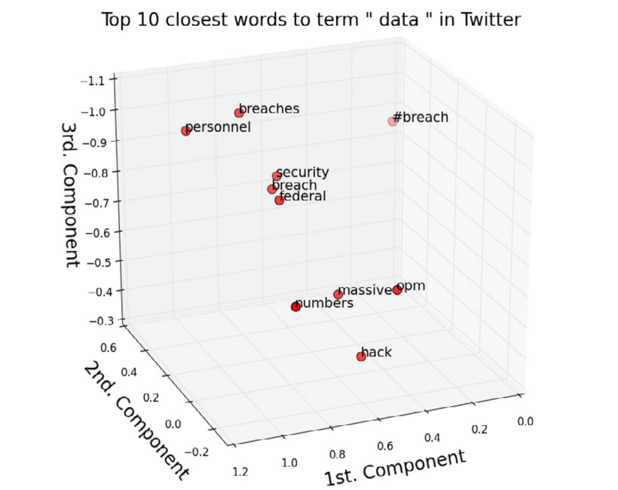


Figure 5. Three-dimensional visualization of Word2vec results.

visualize the word vectors (originally 100 dimensions) in a three-dimensional space—in this case, a 3D cube graph (Figure 5).

What if the marketers using the application want to see how tweets about one company and its products stack up against tweets about other related companies and products? To group the results in this way, Albert can employ the K-means clustering algorithm on top of Word2vec (Figure 6). The larger the circle's diameter, the more frequently the term appears.

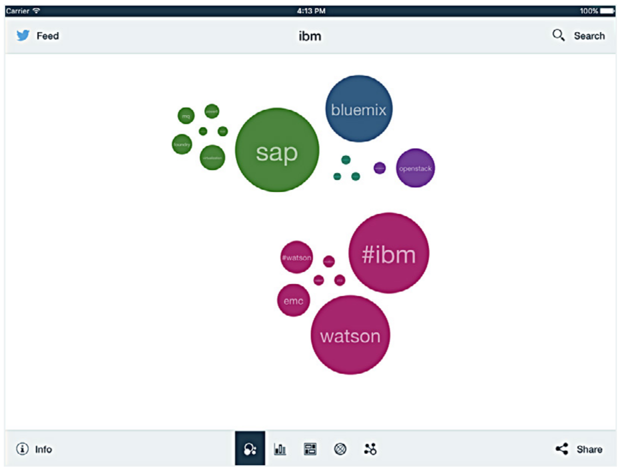
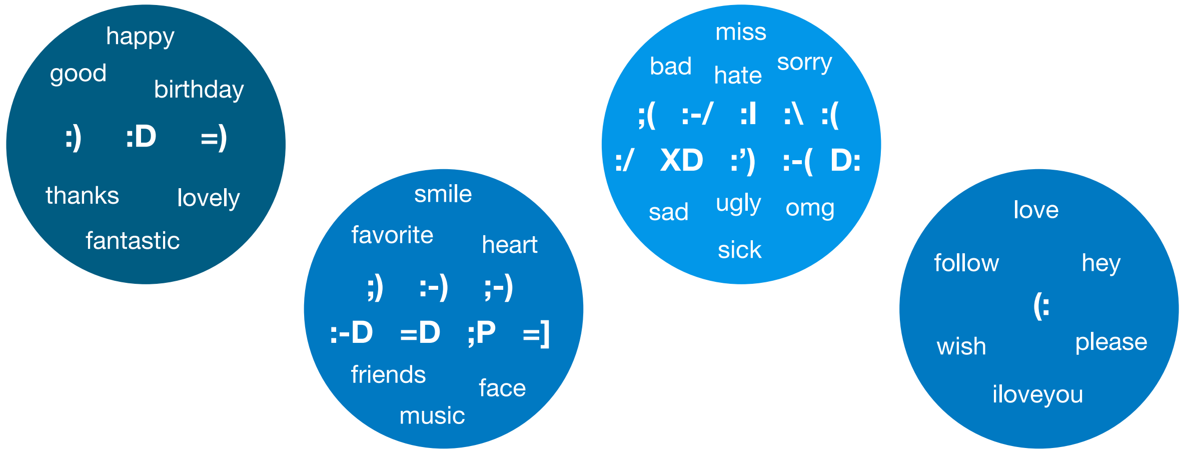


Figure 6. Grouping of Word2vec results using K-means.

K-means clustering can also show marketers how many positive, negative and neutral words are associated with a product, company or other search term. In Figure 7, the analysis of Twitter data includes one day of tweets that contain emoticons as a way to ensure terms with polarized sentiment (positive or negative) cluster together.

K-means clustering of terms with polarized sentiment



Source: Wang, H. and Castañón, J., "Sentiment Expression Via Emoticons in Social Media," IEEE Big Data Conference 2015

Figure 7. Clustering words from tweets with emoticons.

Delivering the application

What you'll learn

- How to put a model into use as an application

At this point, Albert is confident he has built an application that can be used by the marketing team and others who want to see which words Twitter users associate with their company, its products, management, advertising slogans and even strategic ideas.

In Albert's marketing application, the text used for analysis comes from tweets.

But any text document can be an input to the model, and marketing is only one of many uses for the analysis. For example:

- Human resources might use it to gauge employee acceptance of a new HR policy
- IT could use it to see how employees feel about switching to a new email solution
- Sales could use it to enhance customer service

In fact, a new use presented itself quickly once Albert finished the Twitter analysis. The sales manager saw it and told Albert she would love to have that kind of analysis in the field when she is on sales calls. Albert

did some research and found Project RedRock, which offers a variety of ways to visually present big data for iPads. He hooked his model into RedRock to provide processed text data for display (Figure 8). With the ability to view analytic results in visual displays such as force graphs and K-means clusters on her iPad, the sales manager can now compare trends and customer opinions while on the road—and without calling Albert every few days for updated reports.

Ultimately, Albert showed how quick and easy it is to train a Word2vec model with Twitter, and how simple it can be to incorporate the model into data products once you have a clear idea of your goal. Applying mathematical thinking to text data using Word2vec is especially important now, at a time when organizations of all types are experiencing an explosion of text data and are looking to gain market advantage by using it in new ways.

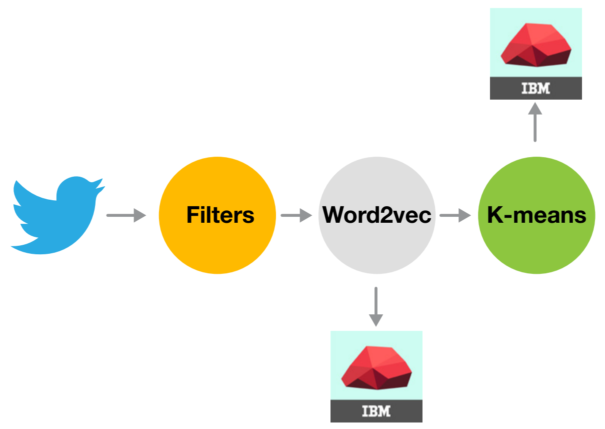


Figure 8. ML pipeline for RedRock iPad prototype.

Resources

For more information about Word2vec, see: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Interested in building your own Word2vec models? Sign up for the IBM Data Science Experience and get started with this notebook: https://github.com/IBMDataScience/word2vec

About the (real) data scientist

Jorge Castañón hails from Mexico City and received his PhD in Computational and Applied Mathematics from Rice University. He has a genuine passion for data science and machine learning applications of any kind. Since 2007, he has been developing numerical optimization models and algorithms for regularization and inverse problems. At IBM, Jorge joined the Big Data Analytics team at Silicon Valley Laboratory, where he is building the future of machine learning and text analytics tools using Apache Spark and Hadoop.

[1] Das, Sudeep, "Navigating themes in restaurant reviews with Word Mover's Distance," http://tech.opentable.com/2015/08/11/navigating-themes-in-restaurant-reviews-with-word-movers-distance/

CDM12351-USEN-01