

Online K-means Clustering and Word2Vec Model

Melodi Çalışkan *

Master Student

Department of Computational Science and Engineering

Bogazici University

Bebek, Istanbul 34432 Turkey

January 2, 2017

*minemelodicaliskan@gmail.com

1 Introduction

Many real world data are acquired sequentially over time, whether messages from social media users, time series from wearable sensors, or the firing of large populations of neurons. In these settings, rather than wait for all the data to be acquired before performing our analyses, we can use online algorithms to identify patterns over time, and make more targeted predictions and decisions.

One simple strategy is to build machine learning models on static data, and then use the learned model to make predictions on an incoming data stream. If the patterns in the data are themselves dynamic online algorithms can be used. An online algorithm is one that can only process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start. In this paper we introduce k-means clustering algorithm, word2vec algorithm and describe online k-means clustering.

2 K-means Clustering

Clustering is basically grouping similar objects together. K-means is one of the clustering algorithms which assigns each cluster with it's respective centroid, i.e, the center of the cluster. This algorithm is performed by following steps:

- Initialize K cluster centers
- Repeat until convergence:
 - Assign each data point to the cluster with the closest center.
 - Assign each cluster center to be the mean of its cluster's data points

Algorithm 1: K-means Algorithm

```
1 Initialize  $m_i, i = 1, \dots, k$ , to k random  $x^t$ ;  
2 Repeat;  
3 For all  $x^t \in \mathbb{X}$  ;  
4  $b_i^t \leftarrow 1$  if  $\|x^t - m_i\| = \min_j \|x^t - m_j\|$ ;  
5  $b_i^t \leftarrow otherwise$ ;  
6 For all  $m_i, i = 1, \dots, k$ ;  
7  $m_i \leftarrow \frac{\sum_t b_i^t x^t}{\sum_t b_i^t}$ ;  
8 Until  $m_i$  converge
```

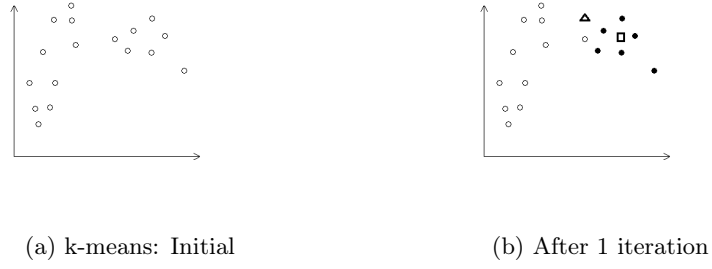
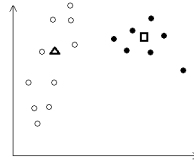


Figure 1: Evolution of k-means



(a) Convergence

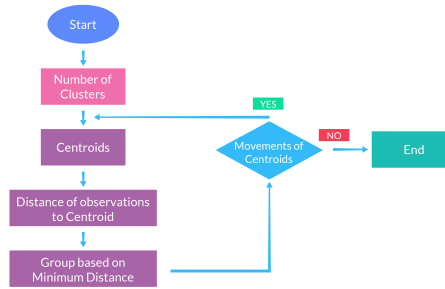


Figure 3

3 Online Algorithms

Many real world data are acquired sequentially over time, whether messages from social media users, time series from wearable sensors, or the firing of large populations of neurons.

In these settings, rather than wait for all the data to be acquired before performing our analyses, we can use online algorithms to identify patterns over time, and make more targeted predictions and decisions.

An online algorithm is one that can only process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start.

4 Online K-means

Online K-means algorithm uses a generalization of the mini-batch k-means update rule. For each batch of data, it assigns all points to their nearest cluster, compute new cluster centers, then update each cluster using:

$$c_{t+1} = \frac{c_t n_t \alpha + x_t m_t}{n_t \alpha + m_t}$$

$$n_{t+1} = n_t + m_t$$

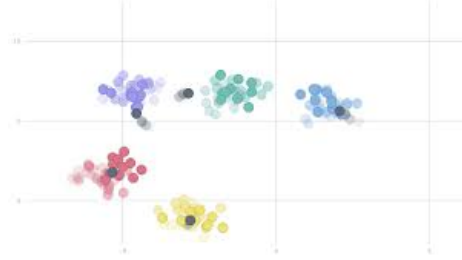


Figure 4

Where c_t is the previous center for the cluster, n_t is the number of points assigned to the cluster thus far, x_t is the new cluster center from the current batch, and m_t is the number of points added to the cluster in the current batch. The decay factor α can be used to ignore the past: with $\alpha = 1$ all data will be used from the beginning; with $\alpha = 0$ only the most recent data will be used. This is analogous to an exponentially-weighted moving average.

Algorithm 2: Generalized Online K-means Algorithm

- 1 Initialize $m_i, i = 1, \dots, k$;
 - 2 Repeat;
 - 3 For all $x^t \in \mathbb{X}$ in random order;
 - 4 $i \leftarrow \operatorname{argmin}_j \|x^t - c_j\|$;
 - 5 $c_i \leftarrow \frac{c_i n_i \alpha + x_i m_i}{n_i \alpha + m_i}$;
 - 6 $n_i = n_i + m_i$;
 - 7 Until c_i converge
-

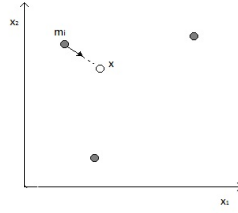


Figure 5

Algorithm 3: Online K-means Algorithm-One At a Time

- 1 Initialize $m_i, i = 1, \dots, k$;
 - 2 Repeat;
 - 3 For all $x^t \in \mathbb{X}$ in random order;
 - 4 $i \leftarrow \operatorname{argmin}_j ||x^t - m_j||$;
 - 5 $m_i \leftarrow m_i + \eta(x^t - m_i)$;
 - 6 Until m_i converge
-

5 Word2vec

Word2vec is a model that is used to produce word embeddings. It computes distributed vector representation of words. In this representation similar words are close in the vector space and this makes generalization to novel patterns easier and model estimation more robust. Distributed vector representation is showed to be useful in many natural language processing applications such as named entity recognition, disambiguation, parsing, tagging and machine translation.

Word2vec relies on either skip-grams or continuous bag of words (CBOW) to create neural word embeddings. It was created by a team of researchers led by Tomas Mikolov at Google.

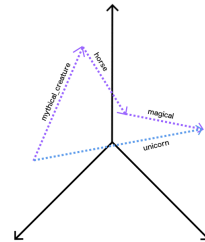


Figure 6: This is a figure caption.

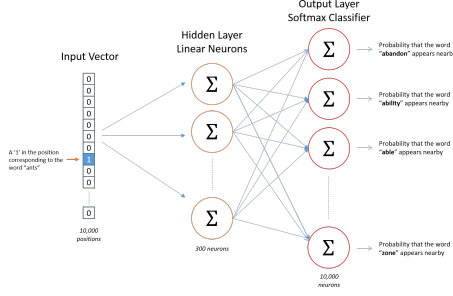


Figure 8

6 Model

In our implementation of Word2Vec, we used skip-gram model. The training objective of skip-gram is to learn word vector representations that are good at predicting its context in the same sentence. Mathematically, given a sequence of training words w_1, w_2, \dots, w_T the objective of the skip-gram model is to maximize the average log-likelihood :

$$\frac{1}{T} \sum_{t=1}^T \sum_{j=-k}^{j=k} \log p(w_{t+j}|w_t)$$

where k is the size of the training window.

In the skip-gram model, every word w is associated with two vectors u_w and v_w which are vector representations of w as word and context respectively. The probability of correctly predicting word w_i given word w_j is determined by the softmax model, which is:



$$p(w_i|w_j) = \frac{\exp(u_{w_i}^\top v_{w_j})}{\sum_{l=1}^V \exp(u_l^\top v_{w_j})}$$

where V is the vocabulary size.

The skip-gram model with softmax is expensive because the cost of computing $\log p(w_i|w_j)$ is proportional to V , which can be easily in order of millions. To speed up training of Word2Vec, we used hierarchical softmax, which reduced the complexity of computing $\log p(w_i|w_j)$ to $O(\log(V))$

7 Discussion

In this paper online and offline k-means algorithms and word2vec models are introduced. Implementation of the algorithms and source code are in the Appendix Section. Online and offline k-means perform with very similar accuracy

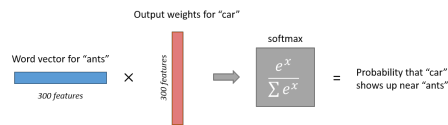
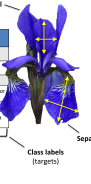


Figure 9

when the data is stable. Initial centers are important for convergence and accurate clustering. Word2vec produces homogeneous numerical data which is a good fit for k-means clustering. We will continue with an implementation to Twitter online feeds.

References

- [1] Ethem Alpaydm *Introduction to Machine Learning* The MIT Press Cambridge, Massachusetts London, England 2014
- [2] Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. *Distributed representations of words and phrases and their compositionality*. 2013
- [3] Tülay Adalı, Taehwan Kim (2003) *Approximation by Fully Complex Multilayer Perceptrons*
10.1162/089976603321891846



Samples (instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...
50	6.4	2.5	4.5	1.2	Versicolour
...
150	5.8	2.0	1.8	0.2	Virginica

Features (attributes, measurements, dimensions)

Class labels (targets)

(a) Iris Data Set Table



(b) Iris types

Figure 10: Iris Data Set

8 Appendix

8.1 Data

The Iris Dataset contains measurements for 150 Iris flowers from three different species.

8.2 Online K-means Implementation on Numerical Data

```
import numpy as np
from numpy import array
import pandas as pd
import time
import math
from math import sqrt
from pyspark import SparkContext, SparkConf
from pyspark import Row
from pyspark.sql import SQLContext
from pyspark.ml.feature import Word2Vec
#from pyspark.ml.clustering import KMeans, KMeansModel
from pyspark.mllib.clustering import KMeans, KMeansModel
from pyspark.mllib.linalg import Vectors
from pyspark.ml.feature import P
sqlContext = SQLContext(sc)
data=sc.textFile("file:///C:/Users/orhan/Desktop/irisdata.txt")
parsedData=data.map(lambda line: array([x for x in line.split(',')
]))
print(parsedData)
parsedData.collect()
params_only=parsedData.map(lambda x: array([float(x[0]), float(x
[1]), float(x[2]), float(x[3])]))
from __future__ import print_function
from math import sqrt
from pyspark import SparkContext
#from pyspark.mllib.clustering import KMeans, KMeansModel

if __name__ == "__main__":
    clusters = KMeans.train(params_only, 2, maxIterations=100,
        initializationMode="random")

    # Evaluate clustering by computing Within Set Sum of Squared
    Errors
    def error(point):
        center = clusters.centers[clusters.predict(point)]
```



```

        return sqrt(sum([x**2 for x in (point - center)]))

WSSSE = params_only.map(lambda point: error(point)).reduce(
    lambda x, y: x + y)
clusters.centers
Kcenters=clusters.centers
clusters.predict([1.01, 2.4, 0.2, 0.8])
from pyspark.mllib.clustering import StreamingKMeans,
    StreamingKMeansModel
initialCenters = Kcenters
initialWeights = [1.0, 1.0]
streamingKmeans = StreamingKMeansModel(initialCenters,
    initialWeights)
parsedData2=[[ 5.1, 3.5, 1.4, 0.2],
[ 4.9, 3. , 1.4, 0.2],
[ 4.7, 3.2, 1.3, 0.2],
[ 4.6, 3.1, 1.5, 0.2],
[ 5. , 3.6, 1.4, 0.2],
[ 5.4, 3.9, 1.7, 0.4],
[ 4.6, 3.4, 1.4, 0.3],
[ 5. , 3.4, 1.5, 0.2],
[ 4.4, 2.9, 1.4, 0.2],
[ 4.9, 3.1, 1.5, 0.1],
[ 5.4, 3.7, 1.5, 0.2],
[ 4.8, 3.4, 1.6, 0.2]]
parsedData2=sc.parallelize(parsedData2)
streamingKmeans = streamingKmeans.update(parsedData2, 1.0, u"
    batches")
streamingKmeans.centers
streamingKmeans.predict([1.01, 2.4, 0.2, 0.8])

```

8.3 Word2Vec

```

In []: import word2vec
In []: word2vec.word2phrase('C:/Users/orhan/Desktop/uygulama_cmt/
    text8', 'C:/Users/orhan/Desktop/uygulama_cmt/text8-phrases',
    verbose=True)
In []: word2vec.word2vec('C:/Users/orhan/Desktop/uygulama_cmt/text8-
    phrases', 'C:/Users/orhan/Desktop/uygulama_cmt/text8.bin', size
    =100, verbose=True)
In []: word2vec.word2clusters('C:/Users/orhan/Desktop/uygulama_cmt/
    text8', 'C:/Users/orhan/Desktop/uygulama_cmt/text8-clusters.txt
    ', 100, verbose=True)
In []: model = word2vec.load('C:/Users/orhan/Desktop/uygulama_cmt/
    text8.bin')
In []: model.vocab
In []: model.vectors.shape
In []: model.vectors
In []: model['dog'].shape
In []: indexes, metrics = model.cosine('socks')
In []: indexes, metrics
In []: model.vocab[indexes]
Out []: array(['haired', 'winged', 'jacket', 'tint', 'reddish', 'dove',
    , 'hairy',
    , 'pinkish', 'prism', 'pumpkin'],
    dtype='<U78')
In []: model.generate_response(indexes, metrics)
Out []: rec.array([( 'haired', 0.8032207248537971), ( 'winged',
    0.7694726079340833),
    ( 'jacket', 0.755904932596702), ( 'tint', 0.7518803287615149),
    ( 'reddish', 0.7500483812651513), ( 'dove', 0.7496740091306853),
    ( 'hairy', 0.7471978070385704), ( 'pinkish', 0.7457286483087218),

```

```

    ('prism', 0.7442350985067347), ('pumpkin', 0.7439536031715859)],
    dtype=[('word', '<U78'), ('metric', '<f8')])
In [ ]: model.generate_response(indexes, metrics).tolist()
In [ ]: indexes, metrics = model.analogy(pos=['paris', 'germany'], neg
    =['france'], n=10)
In [ ]: model.generate_response(indexes, metrics).tolist()
Out [ ]: [('vienna', 0.2955801714578182, 85),
    ('munich', 0.29489056838857897, 53),
    ('berlin', 0.283770446305221, 19),
    ('gymnasium', 0.28089816557455943, 4),
    ('leipzig', 0.2784708785603137, 75),
    ('edinburgh', 0.2783404687163241, 54),
    ('toronto', 0.27388616463375315, 53),
    ('st_petersburg', 0.2729529485165376, 39),
    ('basel', 0.27288353972628976, 95),
    ('trinity_college', 0.2710067027203801, 80)]
In [ ]: clusters = word2vec.load_clusters('C:/Users/orhan/Desktop/
    uygulama_cmt/text8-clusters.txt')
In [ ]: clusters.get_words_on_cluster(90)[:10]
Out [ ]:

```

8.4 K-means Clustering Word2Vec

```

from pyspark.mllib.feature import Word2VecModel
from pyspark.mllib.clustering import KMeans, KMeansModel
from pyspark.mllib.clustering import StreamingKMeans
def clean_string_to_words(line):

    line_review = (line).lower()

    for st in string.punctuation:
        line_review = line_review.replace(st, '_')

    words_list = line_review.lower().split('_')
    words_list = filter(None, words_list)

    return words_list
assert sc.version >= '1.5.1'
sqlContext = SQLContext(sc)
review = sqlContext.read.json('file:///c:/Users/orhan/
    Pet_Supplies_5.json').cache()
review=review.select("reviewText")
review_df = review.filter(review.reviewText != "").cache()
clean_words_rdd = review_df.map(lambda review:
    clean_string_to_words(review.reviewText)).cache()
word2Vec = Word2Vec()
model = word2Vec.fit(clean_words_rdd)
unique_words = word2vec_model.getVectors().keys()
vectors = []
for word in unique_words:
    vectors.append(word2vec_model.transform(word))

vectors_rdd = sc.parallelize(vectors).cache()
kmeans_model = KMeans.train(vectors_rdd, 2000)
word_index = {}
clusters = []
for word in unique_words:
    clus = kmeans_model.predict(word2vec_model.transform(word))
    if clus in clusters:
        word_index[clus].append(str(word))
    else:
        wordlist = []

```

```

        wordlist.append(str(word))
        word_index[clus] = wordlist
        clusters.append(clus)

    print(word_index)
In [ ]: cluster=1477
In [ ]: print(cluster, word_index[cluster])
Out [ ]: 1477 ['bit', 'tiny', 'lil', 'teeny', 'little', 'tad']
In [ ]: cluster=950
In [ ]: print(cluster, word_index[cluster])
Out [ ]: 950 ['fluffier', 'looser', 'drier', 'firmer', 'denser', 'softer']

```