Clustering Semantically Similar Words [1]

Data Science Weekend Camp & Jam 2016

26 November 2016

[1] Author: Bayu Aldi Yansyah
bay@artificialintelligence.id

—
Revisi:
Saturday 3rd December, 2016 02:30
Versi terbaru:
https://github.com/pyk/talks
—

Dokumen ini berisi materi yang akan kita bahas bersama pada sesi "Clustering Semantically Similar Words" di Data Science Weekend Camp & Jam [2]. Para peserta di asumsikan sudah familiar dengan dasar-dasar *Neural Networks (NN)*, *Natural Language Processing (NLP)* dan *Machine Learning (ML)*.

[2] 5 Desember 2016 di Universitas Islam Indonesia, Yogyakarta.
http://datascienceweekend.id

Kita akan mulai pembahasan dari pengenalan word clustering dan apa aja yang kita butuhin untuk melakukan word clustering. Lalu di sesi ini kita juga akan membahas penerapan *Deep Learning* untuk NLP khususnya untuk *Word Embedding* yaitu model neural networks *Feed-forward Neural Net Language Model (NNLM)* dari Bengio et al (2003).

Kita juga akan membahas 2 model dari Mikolov et al. (2013) yang terinspirasi oleh NNLM, model ini populer dengan nama *Word2Vec*.

Setelah itu kita akan bahas gimana cara mengukur kesamaan semantik antara dua kata menggunakan beberapa *Similarity metrics* dan yang terakhir kita bahas algoritma clustering *Consensus Clustering*.

Di bagian akhir akan ada juga studi kasus masalah apa yang bisa di selesaikan oleh tim Data Science di Sale Stock [3] menggunakan word clustering ini.

[3] Sale Stock Engineering
https://careers.salestock.io/

Untuk koreksi dan masukan bisa dikirimkan langsung ke email saya yah. [1]

Introduction to Word Clustering

Word Clustering adalah teknik yang digunakan untuk mencari kelompok-kelompok kata dimana setiap kata dalam satu kelompok memiliki kesamaan secara semantik.[4]

[4] Antonio Sanfilippo et al. Eagles le3-4244: Preliminary recommendations on lexical semantic encoding final report. pages 171–175, 1999

Misalkan kita punya himpunan kata:

$$W = \{w_1, w_2, ..., w_n\}, n \in \mathbb{N}$$

Tujuan kita adalah mencari kelompok-kelompok kata:

$$C = \{C_1, C_2, ..., C_k\}, k \in \mathbb{N}$$

Dengan setiap kelompok kata C_i kita definisikan sebagai:

$$C_i = \{w \mid \forall w \in W \text{ yang } similarity(w_c, w) \geq t\}$$

Jadi C_i adalah kelompok kata yang memiliki kesamaan satu dengan yang lain. Di tiap-tiap C_i terdapat kata utama w_c atau sering disebut *Centroid* dari kelompok kata tersebut.[5] Kemudian $similarity(w_c, w)$ adalah suatu fungsi yang menghitung nilai kesamaan antara kata w_c dan kata w. t adalah nilai batas kesamaan

[5] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*, chapter Cluster Analysis: Basic Concepts and Algorithms. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005

dimana jika nilai fungsi tersebut lebih besar dari t maka w_c dan w dekat secara semantik.

Untuk $w_1 \in C_a$ dan $w_2 \in C_b$ berlaku $similarity(w_1, w_2) < t$, sehingga:

$$C_a \cap C_b = \emptyset, \forall C_a, C_b \in C$$

atau dengan kata lain, tidak ada satu kata yang masuk dalam dua kelompok kata yang berbeda.

Sebenernya fungsi $similarity(w_c, w)$ berguna untuk mencari nilai kesamaan bisa berdasarkan 2 kriteria: kesamaan berdasarkan leksikal dan kesamaan berdasarkan semantik. Di sesi ini kita akan bahas fungsi $similarity(w_c, w)$ berdasarkan semantik aja. Bagi yang tertarik tentang yang leksikal bisa baca papernya Gomaa dan Fahmy (2013).

Untuk melakukan mengelompokkan kata yang sama secara semantik, hal-hal harus kita lakukan adalah

1. Merepresentasikan kata menjadi sebuah semantik vektor[6] yang bisa kita hitung nilai kesamaan dan ketidaksaman semantiknya.
2. Menemukan w_c untuk tiap cluster di C.
3. Menentukan metrik kesamaan $similarity(w_c, w)$ dan nilai batas t.

[6] D. Jurafsky and J.H. Martin. *Speech and Language Processing (3rd ed. draft)*, chapter Vector Semantics

Untuk yang langkah pertama kita bisa menggunakan teknik yang dinamakan *word embedding*. Untuk yang nomor dua kita bisa memakai *Consensus Clustering* dan yang terakhir kita bisa menggunakan *Cosine similarity*, *Jaccard similarity* atau *Euclidean distance*.

Perlu diketahui bahwa **semantik bukanlah sinonim**, dua kata yang sama secara semantik bukan berarti bahwa dua kata tersebut memiliki makna yang sama.

Dua kata dikatakan sama secara semantik, jika dua kata tersebut sering muncul **di konteks yang sama**. Jadi ada kemungkinan bahwa kata yang berlawanan (antonim) sangatlah dekat secara semantik, karena sering muncul di konteks yang sama.[7]

[7] Wael H Gomaa and Aly A Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13), 2013

Pada bagian berikutnya akan dibahas lebih detail tentang teknik word embedding, dilanjutkan dengan bagaimana cara mengukur kesamaan antara dua kata dan algoritma-algoritma clustering yang bisa kita pakai.

Word Embedding

Word embedding adalah teknik di untuk memetakan kata menjadi vektor bilangan real berdimensi k. Hasil dari *word embedding* ini sering disebut *word vector* atau *distributed representation of words*.

Beberapa pendekatan yang bisa kita gunakan untuk melakukan pemetaan kata menjadi vektor bilangan real antara lain pendekatan berdasarkan neural networks, pendekatan berdasarkan reduksi dimensionalitas[8] dan pendekatan berdasarkan model probabilistik.[9]

[8] Rémi Lebret and Ronan Lebret. Word emdeddings through hellinger PCA. *CoRR*, abs/1312.5542, 2013

[9] Amir Globerson et al. Euclidean embedding of co-occurrence data. *Journal of Machine Learning Research*, 8(Oct):2265–2295, 2007

Disini kita akan fokus pada pendekatan berdasarkan neural networks. Pendekatan ini menggunakan model neural networks

Figure 1: Visualisasi kata dengan representasi $k = 5$ dimensional vektornya

untuk mempelajari representasi vektor tiap kata yang ada di dalam korpus.

Kita akan bahas 3 model neural networks, yaitu *Feedforward Neural Net Language Model*, *Continuous Bag-of-Words Model* dan *Continuous Skip-gram Model*.

Model-model yang akan kita bahas menggunakan data training urutan kata

$$w_1, w_2 \dots w_T \text{ untuk } w_t \in V$$

dengan x_t adalah representasi vektor 1-of-$|V|$ dari tiap kata w_t yang ada di *vocabulary* V.

Vektor 1-of-$|V|$ adalah vektor dimana dari sebanyak $|V|$ unit , $w_1, w_2 \dots w_{|V|}$ hanya ada satu unit aja yang bernilai 1 lainnya bernilai 0.

Untuk mempermudah membandingkan modelnya, kita akan memakai notasi dari Collobert et al. (2011), jadi setiap neural networks dengan jumlah layer L bisa kita tulis sebagai komposisi fungsi:

$$f_\theta(\cdot) = f_\theta^L\left(f_\theta^{L-1}\left(\dots f_\theta^1(\cdot)\dots\right)\right)$$

dengan parameter untuk masing-masing layernya $1 \leq l \leq L$:

$$\theta = \left(\theta^1, \theta^2, \dots, \theta^L\right)$$

pada umumnya, setiap layer punya weight dan bias $\theta^l = (W^l, b^l)$.

Feedforward Neural Net Language Model (NNLM)

NNLM kita bahas pertama karena model inilah yang menginspirasi dua model neural networks yang akan kita bahas berikutnya.

Figure 2: NNLM mencoba memprediksi kata apa yang akan muncul berikutnya w_t berdasarkan 4 kata sebelumnya "bisa", "stock", "sale" dan "keren".

Model ini di usulkan oleh Bengio et al. (2003) untuk mengatasi masalah **curse of dimensionality**[10]. Ide sederhananya adalah model ini ingin memprediksi kata yang akan muncul berikutnya w_t berdasarkan data n kata sebelumnya $w_{t-1} \dots w_{t-n}$.

[10] Ini adalah istilah dari masalah yang sering muncul ketika kita sedang mengolah data *high-dimensional*

Model ini terdiri dari 4 layer: *Input layer*, *Projection layer*, satu atau lebih *Hidden layer* dan *Output layer* (Figure 3). Layer-layer ini bisa kita tulis sebagai komposisi fungsi seperti berikut:

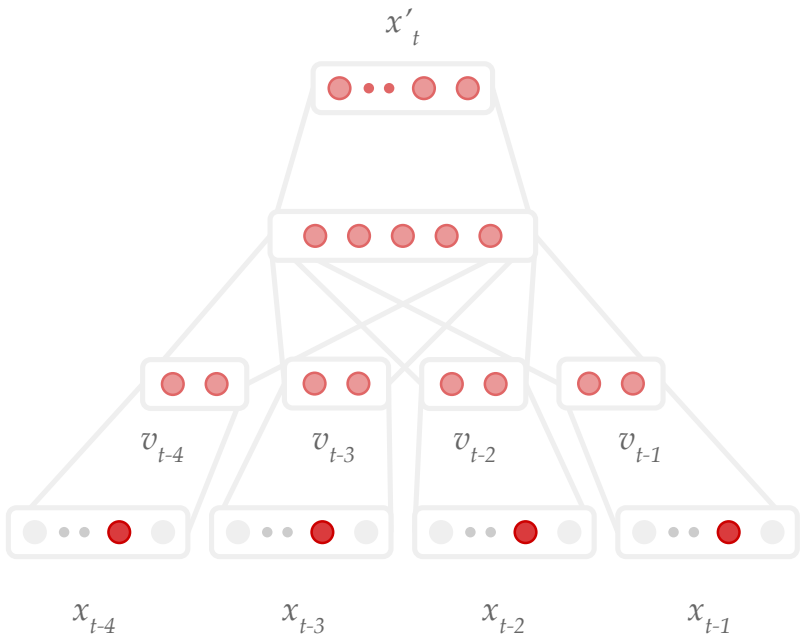
W^{iT} adalah matrix transpose dari matrix W^i

$$x_t^{'} = f(x_{t-1}, \ldots, x_{t-n}; \theta)$$

untuk detailnya:

$$\begin{aligned}
\text{output: } & \left[f_\theta^4\right]_i = x_t^{'} = \sigma\left(W^{4T}\left[f_\theta^3\right]_i + b^4\right) \\
\text{hidden: } & \left[f_\theta^3\right]_i = \tanh\left(W^{3T}\left[f_\theta^2\right]_i + b^3\right) \\
\text{projection: } & \left[f_\theta^2\right]_i = \left(W^{2T}x_{t-1}, W^{2T}x_{t-2}, \ldots, W^{2T}x_{t-n}\right) \\
\text{input: } & \left[f_\theta^1\right]_i = x_{t-1}, \ldots, x_{t-n}
\end{aligned}$$

Figure 3: Visualisasi *Feedforward Neural Net Language Model* dengan $|V| = 6$ dan hyperparameter $n = 4$, $m = 2$ dan $h = 5$.



Oke, kita bahas mulai dari input layer. Di layer ini setiap

$$x_{t-1}, x_{t-2}, \ldots, x_{t-n} \in \mathbb{R}^{|V|\times 1}$$

adalah vektor 1-of-$|V|$ dari n kata sebelumnya $w_{t-1}, w_{t-2}, \ldots, w_{t-n}$.

Selanjutnya untuk projection layer. Di paper Bengio et al. (2003), layer ini disebut *The shared word features layer* karena di layer ini ada parameter word embedding $W^2 \in \mathbb{R}^{|V|\times m}$ atau matrix $|V| \times m$ dimana setiap barisnya adalah vektor yang berdimensi m. Vektor-vektor inilah yang nantinya kita pakai untuk merepresentasikan kata-kata yang akan kita clustering.

Hal yang menarik di layer ini adalah tidak adanya fungsi aktivasi non-linear seperti tanh yang ada di hidden layer. Jadi disini setiap $x_{t-1}, x_{t-2}, \ldots, x_{t-n}$ langsung di proyeksikan oleh $W^2 \in \mathbb{R}^{|V|\times m}$

ke dimensi yang lebih kecil $v_i \in \mathbb{R}^{m\times 1}$. $\left[f_\theta^2\right]_i$ merupakan matrix hasil kontatenasi dari vektor $v_{t-1}, v_{t-2}, \ldots, v_{t-n}$.

Kemudian output dari projection layer $\left[f_\theta^2\right]_i \in \mathbb{R}^{nm\times 1}$ kita operasikan di hidden layer yang mempunyai parameter $W^3 \in \mathbb{R}^{h\times nm}$ dan $b^3 \in \mathbb{R}^h$ dengan h adalah jumlah *hidden units*. Fungsi aktivasi yang ada di layer ini adalah *hyperbolic tangent*. Di layer ini kita dapat $\left[f_\theta^3\right]_i \in \mathbb{R}^{h\times 1}$.

Yang terakhir, output layer. Di layer ini kita menggunakan fungsi aktivasi $softmax$, sebagai klasifikasi log-linear, untuk mendapatkan nilai probabilitas kata apa yang akan muncul berikutnya. Di layer ini kita punya parameter $W^3 \in \mathbb{R}^{h\times|V|}$ dan $b^3 \in \mathbb{R}^{|V|}$. Di layer ini kita dapat $\left[f_\theta^4\right]_i \in \mathbb{R}^{|V|\times 1}$ dimana $\left[f_\theta^4\right]_i(t)$ adalah probabilitas kata w_t yang akan muncul berikutnya.

Model ini di training menggunakan *stochastic gradient ascent* dengan tujuan untuk memaksimalkan fungsi tujuan berikut:

$$L = \frac{1}{N}\sum_{i=1}^{N} \log\left[f(x_{t-1}, \ldots, x_{t-n}; \theta)\right]_i$$

dengan N adalah jumlah training data.

Untuk cara update parameternya:

$$\theta_{baru} \leftarrow \theta_{lama} + \frac{\partial L}{\partial \theta}$$

Btw, pada prakteknya model ini sangat jarang sekali digunakan karena memerlukan daya komputasi yang besar. Terutama pada komputasi di hidden layernya. Untuk itu Mikolov et al. (2013), mengusulkan model yang lebih baik yaitu *Continuous Bag-of-Words Model* dan *Continuous Skip-gram Model* atau sangat populer dengan nama *Word2Vec*. Dua model inilah yang akan kita bahas berikutnya.

Continuous Bag-of-Words Model (CBOW)

Perbedaan mendasar dari CBOW dengan NNLM terletak pada konteks input katanya, projection layer dan tidak adanya hidden layer.

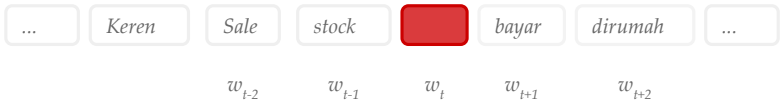


Figure 4: CBOW mencoba memprediksi kata yang akan muncul w_t berdasarkan konteks 2 kata sebelumnya: "stock" dan "sale", dan 2 kata setelahnya "bayar" dan "dirumah".

Ide sederhana dari CBOW adalah model ini ingin memprediksi kata yang akan muncul w_t berdasarkan informasi konteks n kata sebelum dan n kata setelahnya a $w_{t-n}, \ldots, w_{t-1}, w_{t+1}, \ldots w_{t+n}$.

CBOW punya 3 layer, yaitu *Input layer*, *Projection layer* dan *Output layer* (Figure 5). Kita bisa tulis model ini sebagai komposisi fungsi seperti berikut:

$$x_t^{'} = f(x_{t-n}, \ldots, x_{t-1}, x_{t+1}, \ldots, x_{t+n}; \theta)$$

untuk detailnya:

$$\text{output: } \left[f_\theta^3\right]_i = x_t' = \sigma\left(W^{3T}\left[f_\theta^2\right]_i + b^3\right)$$
$$\text{projection: } \left[f_\theta^2\right]_i = \frac{1}{2n}\left(W^{2T}x_{t-n} + \ldots + W^{2T}x_{t-1} + W^{2T}x_{t+1} + \ldots + W^{2T}x_{t+n}\right)$$
$$\text{input: } \left[f_\theta^1\right]_i = x_{t-n}, \ldots, x_{t-1}x_{t+1}, \ldots, x_{t+n}$$

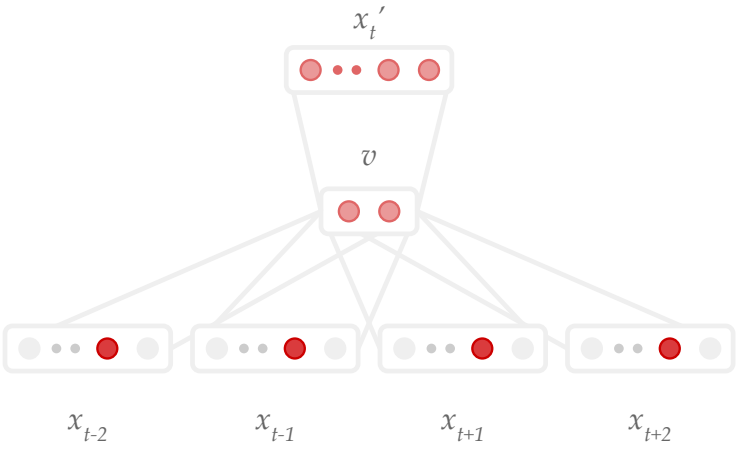


Figure 5: Visualisasi *Continuous Bag-of-Words Model* dengan vocabulary berjumlah $|V|$ dan hyperparameter $n = 2$, $m = 2$ dan $h = 5$.

Oke, kita bahas mulai dari input layer nya. Di layer ini

$$x_{t-n}, \ldots, x_{t-1}x_{t+1}, \ldots, x_{t+n} \in \mathbb{R}^{|V|\times 1}$$

merupakan vektor 1-of-$|V|$ dari konteks n kata sebelum $w_{t-n}, \ldots, w_{t-1}$ dan n kata setelahnya $w_{t+1}, \ldots, w_{t+n}$.

Di projection layer, vektor $|V|x1$ tersebut di proyeksikan ke dimensi yang lebih kecil. Perbedaan model ini dengan model NNLM adalah, model ini memproyeksikan $x_{t-n}, \ldots, x_{t-1}x_{t+1}, \ldots, x_{t+n}$ ke satu vektor $v \in \mathbb{R}^{m\times 1}$ saja dengan cara mengambil rata-ratanya.

$$v = \frac{1}{2n}\left(W^{2T}x_{t-n} + \ldots + W^{2T}x_{t-1} + W^{2T}x_{t+1} + \ldots + W^{2T}x_{t+n}\right)$$

Di model ini tidak ada hidden layer.

Untuk output layer, metode training dan cara update parameternya sama dengan model NNLM.

Untuk fungsi tujuannya ada sedikit modifikasi:

$$C = \frac{1}{N}\sum_{i=1}^{N} \log\left[f(x_{t-n}, \ldots, x_{t-1}, x_{t+1}, \ldots, x_{t+n}; \theta)\right]_i$$

dengan N adalah banyaknya training data.

Sama seperti model NNLM, hasil akhir dari model ini yang kita pakai adalah parameter $W^2 \in \mathbb{R}^{|V|\times m}$ dimana setiap barisnya merupakan representasi vektor m-dimensional dari kata-kata yang ada di V.

Continuous Skip-gram Model

Berbeda dengan CBOW yang mau coba prediksi suatu kata berdasarkan konteksnya, ide sederhana dari model Skip-gram ini adalah dia mau mencoba prediksi konteks berdasarkan satu kata aja.

Figure 6: Skip-gram mencoba memprediksi konteks n kata setelah dan n kata sebelum berdasarkan kata "bisa".

Sama seperti CBOW, Skip-gram punya 3 layer. Ada *Input layer*, *Projection layer* dan *Output layer* (Figure 7). Kita bisa tulis model ini sebagai komposisi fungsi seperti berikut:

$$x^{'} = f(x_t; \theta)$$

untuk detailnya:

$$\text{output: } \left[f_\theta^3\right]_i = x_t^{'} = \sigma\left(W^{3T}\left[f_\theta^2\right]_i + b^3\right)$$
$$\text{projection: } \left[f_\theta^2\right]_i = v = \left(W^{2T} x_t\right)$$
$$\text{input: } \left[f_\theta^1\right]_i = x_t$$

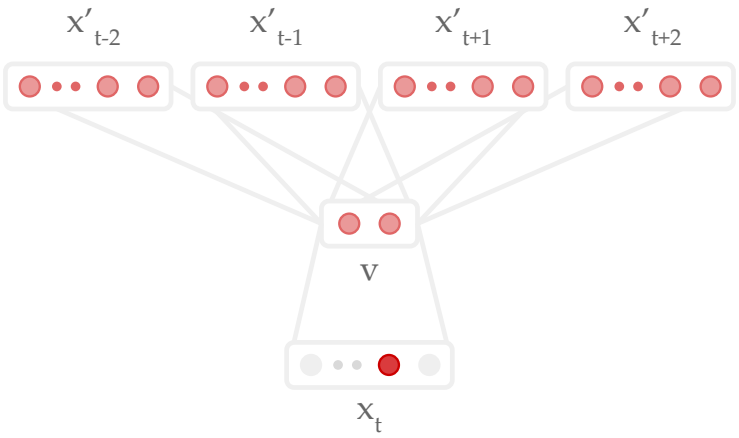


Figure 7: Visualisasi *Continuous Skip-gram Model* dengan vocabulary berjumlah $|V|$ dan hyperparameter $n = 2$, $m = 2$ dan $h = 5$.

Oke kita mulai dari yang input layer, model ini inputnya cuma vektor 1-of-$|V|$ dari kata w_t aja.

Kemudian untuk projection layernya, persis sekali dengan CBOW. Di layer ini ada parameter $W^2 \in \mathbb{R}^{|V| \times m}$ dan $b^2 \in \mathbb{R}^m$. Di layer ini x_t di proyeksikan ke vektor dengan dimensi yang lebih kecil $\left[f_\theta^2\right]_i \in \mathbb{R}^m$.

Untuk output layernya kita punya parameter $W^3 \in \mathbb{R}^{m \times 2n|V|}$ dan $b^3 \in \mathbb{R}^{2n|V|}$. Output dari layer ini merupakan vektor yang berukuran $2n|V|$ kata dengan tiap-tiap $|V|$ elemen merupakan vektor dari konteks kata yang di prediksi.

Output vektornya bisa kita tulis sebagai:

$$x^{'} = \left(p\left(w_{t-n} \mid w_t\right), \ldots, p\left(w_{t-1} \mid w_t\right), p\left(w_{t+1} \mid w_t\right), \ldots, p\left(w_{t+n} \mid w_t\right)\right)$$

Sehingga kita bisa tulis fungsi tujuan kita sebagai berikut:

$$C = \frac{1}{N}\sum_{i=1}^{N} \log \left[\sum_{j=1}^{-n \leq j \leq n, j \neq 0} p(w_{t+j} \mid w_t) \right]_i$$

Dan yang terakhir, sama seperti NNLM dan CBOW, yang kita gunakan dari model ini adalah parameter di projection layernya $W^2 \in \mathbb{R}^{|V| \times m}$ dimana setiap baris adalah vektor dari kata-kata yang ada di vocabulary $|V|$.

Mengukur Nilai Kesamaan antara 2 Vektor Kata

Sebenernya ada banyak cara untuk mengukur nilai kesamaan antara 2 vektor kata. Kita disini akan fokus pada *Term-based Similarity* yaitu Cosine similarity dan Euclidean similarity. [11]

Setelah kita dapat representasi vektor dari kata yang ada di vocabulary, tentunya melalui teknik word embedding yang sebelumnya kita bahas, langkah berikutnya yang kita butuhkan untuk melakukan pengelompokan kata secara semantik adalah mengukur nilai kesamaan semantik antara 2 vektor kata.

Formula yang bisa kita pakai adalah Cosine similarity. Misalkan kita punya vektor kata $v_1, v_2 \in \mathbb{R}^m$, maka:

$$simlarity\left(v_1, v_2\right) = \frac{v_1 \cdot v_2}{\|v_1\|\|v_2\|}$$

dimana $-1 \leq simlarity\left(v_1, v_2\right) \leq 1$ semakin mendekati nilai 1 semakin similar 2 vektornya.

Untuk yang kedua, kita bisa pake Euclidean distance untuk mencari nilai similarity antara 2 vektor kara.

Untuk mencari Euclidean distance:

$$distance\left(v_1, v_2\right) = \sqrt{\sum_{i=1}^{n}\left(v_{1i} - v_{2i}\right)^2}$$

hanya dengan sedikit perubahan kita bisa dapet Euclidean similarity-nya:

$$similarity\left(v_1, v_2\right) = \frac{1}{1 + distance\left(v_1, v_2\right)}$$

Consensus Clustering

Ide dasarnya adalah kita mau menemukan w_c berdasarkan konsensus.

Disini kita akan fokus pada salah satu metode clustering yang disebut *Consensus clustering*[12]. Sebenarnya ada beberapa pendekatan untuk Consensus clustering ini diantaranya *Iterative Voting Consensus*, *Iterative Probabilistic Voting Consen-sus* dan *terative*

[11] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul MB Vitányi. The similarity metric. *IEEE transactions on Information Theory*, 50(12):3250–3264, 2004

[12] Nam Nguyen and Rich Caruana. Consensus clusterings. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, ICDM '07, pages 607–612, Washington, DC, USA, 2007. IEEE Computer Society

Pairwise Consensus tapi disini kita akan bahas Iterative Voting Consensus dengan sedikit modifikasi sesuai kasus kita.

Oke, ide sederhana dari metode ini adalah kita memilih *Centroid* atau pusat clusternya berdasarkan nilai kesamaan semantik antara tiap-tiap vektor katanya. Untuk psuedocode nya seperti ini:

Algorithm 1: Iterative Voting Consensus with slightly modification

Input : Word vector $v_1, \ldots, v_{|V|} \in \mathbb{R}^m$
Set of cluster $C = \{C_1, \ldots, C_k\}, k \in \mathbb{N}$
Threshold similarity t
Output: Updated set of cluster C
foreach v_i **do**
  set createNewCluster = True;
  foreach C_i *in* C **do**
    set $w_c = w_c$ of C_i;
    if $similarity(w_c, v_i) \geq t$ **then**
      $C_i = C_i \cup \{v_i\}$;
      set createNewCluster = False;
      break;
    end
  end
  if *createNewCluster* **then**
    $C = C \cup \{w_c = v_i\}$
  end
end

References

[1] Yoshua Bengio et al. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155, 2003.

[2] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011.

[3] Amir Globerson et al. Euclidean embedding of co-occurrence data. *Journal of Machine Learning Research*, 8(Oct):2265–2295, 2007.

[4] Wael H Gomaa and Aly A Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13), 2013.

[5] D. Jurafsky and J.H. Martin. *Speech and Language Processing (3rd ed. draft)*, chapter Vector Semantics.

[6] Rémi Lebret and Ronan Lebret. Word emdeddings through hellinger PCA. *CoRR*, abs/1312.5542, 2013.

[7] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul MB Vitányi. The similarity metric. *IEEE transactions on Information Theory*, 50(12):3250–3264, 2004.

[8] Tomas Mikolov et al. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[9] Tomas Mikolov et al. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*. Association for Computational Linguistics, May 2013.

[10] Nam Nguyen and Rich Caruana. Consensus clusterings. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, ICDM '07, pages 607–612, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Xin Rong. word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014.

[12] Antonio Sanfilippo et al. Eagles le3-4244: Preliminary recommendations on lexical semantic encoding final report. pages 171–175, 1999.

[13] Martin Sundermeyer, Ilya Oparin, J-L Gauvain, Ben Freiberg, Ralf Schlüter, and Hermann Ney. Comparison of feedforward and recurrent neural network language models. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8430–8434. IEEE, 2013.

[14] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*, chapter Cluster Analysis: Basic Concepts and Algorithms. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.