NLP – Algorithms, Terms, Maths

1. Term Document matrix – frequency of word in each document
2. **TF-IDF** – raw word count / document count
3. Frequency of word in each document becomes the vector (list of numbers)
4. **Word embedding is nothing but a vector representing a word**
5. VxD = Vocabulary size x Vector dimensionality. Eg:
   5.1. Counting word occurrence in a set of books. D = Total no of books
6. **Vectorization – turn collection of text documents into numerical feature vectors**
   6.1. Tokenize – int id to each token of string, using space/punctuation as separator
   6.2. Count – occurrence of token in each document
   6.3. Normalize – mean weighting with diminishing importance of tokens in samples
   6.4. Feature –
   6.5. Sample –
   6.6. CountVectorizer – tokenizes a collection of text documents, builds a vocabulary of known words and encodes new documents using that vocabulary
7. **Word analogies** – Magnitude and direction are very close to each other
   7.1. King-Queen ~= Prince-Princess, Walk-Walking ~= Swim-Swimming
   7.2. LeftVector ~= Right vector
   7.3. There are 4 words in analogies. Given 3 find the 3rd
   7.4. Use vector distance to find closest matching word
   7.5. Euclidean or Cosine distance
   7.6. Word2Vec and GloVe have no concept of analogies
   7.7. Raw word count and TF-IDF don't give good analogies
8. **Dimensionality** – TF-IDF / word count across documents creates high dimensionality
   8.1. **t-SNE** reduces dimensionality
   8.2. TF-IDF is not good for analogies
9. Pretrained word vectors from GloVe, word2vec – benefit in using them for word embeddings
10. **Text Classification** – using word vectors. Also, Bag of words
11. **N-grams. Bigram** model = p(wt|wt-1) = count(wt-1 -> wt) / count(wt-1)
    11.1. Bayes rule = chain rule of probability
    11.2. Trigram: P(A->B->C) = p(C|A->B)*p(B|A)*p(A) = count(A->B->C)/count(A->B)
    11.3. P(A) = count(A) / corpus length
    11.4. Psmooth(B/A) = count(A->B)+1/count(A)+V. V = vocabulary, distinct words
       11.4.1. **Add 1 smoothing**. V is added to ensure probabilities sum to 1
       11.4.2. Handles the 0 probability case where sentence is valid but dependent word does not occur in that sentence
12. **Markov assumption** = what you see now is only dependent on previous step
    12.1. P(E|ABCD) = p(E|D). First order
    12.2. Convert multi-word sentence to bigram only model.
       12.2.1. P(ABCDE) = P(E|D)*P(D|C))*P(C|B))*P(B|A))*P(A)
    12.3. Longer sentences are infrequent. Shorter phrases are common, more samples so make sentences more probable
    12.4. $P(w_1,\ldots,w_T) = p(w_1)*\pi_{t=2toT}\ P(w_t|w_{t-1})$
13. **Underflow problem** - Probabilities are between 0 and 1. If you multiply too many, gets close to numerical precision 0, so rounded to 0.

13.1. So you use log function. Increasing function. Multiplication becomes addition
13.2. Log $P(w_1,\ldots,w_T) = \log p(w_1) + \sum_{t=2toT} \log P(w_t|w_{t-1})$
14. **Normalize each sentence**. Using raw probabilities, Bias towards shorter sentences
14.1. So normalize using sentence size T
14.2. Log $P(w_1,\ldots,w_T)/T = \log p(w_1)/T + \sum_{t=2toT} \log P(w_t|w_{t-1})/T$
14.3. Log probabilities are -ve (between 0 and 1)
14.4. Longer sentences will have more -ve numbers
14.5. So, logp(shorter sentences) > logp(longer sentences)

15. Bigrams – 2 words in sequence
15.1. Probability = count no of occurrences
15.2. P(a|b) = P(b|a)/P(b)
15.3. The quick brown fox jumps over the lazy dog
15.4. P(brown|quick) = Count(quick->brown))/Count(quick) = 1/1 = 1

16. **Vector norm – for n dimensional vector** x = [x1, x2, x2, …xn]
16.1. |x| is the general vector norm, absolute norm
16.2. ||x|| is matrix norm
16.3. $|x|_p$ for p = 1 to n, = $(\sum|x_i|^p)$^1/p
16.4. Most common vector norm is **L^2 norm, $|x|_2 = \text{Sqrt}(x_1^2+x_2^2+\ldots+x_n^2)$**
16.5. X= (1,2,3). L_2 norm = Sqrt(14) = 3.742
16.6. Euclidean norm = L^2 norm = Frobenius norm

17. Pairwise distance = Euclidean or cosine distance between 2 vectors
17.1. Euclidean distance between a,b = sqrt(sum(abs(a-b)^2))
17.1.1. (2,3,4,2) and (1,-2,1,3) = sqrt(1+25+9+1) = 6
17.2. **Manhattan distance = ∑(|a-b|).** Difference between coordinates. City grid layout. Sum of each grid point distance
17.2.1. A=[[1,2], [3,4], [5,6]], b=[[1,2], [3,4]].
17.2.2. Md = [|a1-b1|+|a2-b2|, |a1-b3+a2-b4|]
17.2.3. |a3-b1|+a4-b2,]
17.2.4. |a5-b1|+|a6-b2|,]]
17.3. Manhattan can be better for high dimension ML problems due to lower cost function compared to Euclidean which requires sqrt, pow
17.4. Cosine distance = 1 – cos theta = angular distance = 1-angular similarity
17.4.1. Cosine similarity = cos ø or cos theta
17.4.2. **Cos ø = $\sum$AiBi/SQRT($\sum Ai^2$)* SQRT($\sum Bi^2$),** i = 1 to n
17.4.3. Angular distance = $\cos^{-1}$(cosine similarity)/π
17.4.4. The advantage of the angular similarity coefficient is that, when used as a difference coefficient (by subtracting it from 1) the resulting function is a proper distance metric
17.4.5. Vectors are similar if they are parallel and dissimilar if they are orthogonal
17.4.6. Similarity ranges from -1 exact opposite to 1 exactly same, 0 means decorrelation
17.4.7. For text matching A,B are term frequency vectors, cos ø normalises document length during comparison
17.4.8. For information retrieval cos ø is from 0 to 1 as term frequencies tf-idf cannot be -ve
17.5. Linear kernel – kernel computes dot product of 2 vectors x,y in higher dimension feature space.

17.5.1. Linear kernel is the length of projection of one vector on another. Simple dot product.

17.5.2. $K(x,y) = x^T*y$. sklearn did x*y

18. **Word embedding dimensionality** - Glove 50/100/300 dimensions meaning
    18.1. Dimensionality represents no of features it encodes
    18.2. Using hidden layer in training. Most useful features are selected
    18.3. Each dimensions features combined in non simple/orthogonal way
    18.4. Word embedding maps word to higher dimension vector using a function
    18.5. Function is a lookup table, parameterized by matrix T with a row for each word $W_m(w_n)$ = Tn. W: words -> R^n
19. **Glove** is based on co-occurrence matrix and trains word vectors so that their differences predict co- occurrence ratios
    19.1. Word2vec considers only local context, glove considers global context
    19.2. Word2vec simple word analogies can be expressed as simple vector math
    19.3. Vectors capture dimension of meaning
    19.4. Glove uses both global context and learning dimensions of meaning
    19.5. Co-occurrence ratios between 2 words in a context are strongly connected to meaning
    19.6. Probabilities show count of word 'k' when a specific word appears in the context
20. **Word embedding design – word vector encodes semantic relationship among words**
    20.1. CBOW – uses n words before and after target word 'w', to predict w
    20.2. Skipgram – uses target word 'w' to predict n words before and after it
    20.3. Negative sampling – used small sample of -ve training record to train model
    20.4. Word2vec – trained on 100bn google news words, 300 dimensions, skip-gram and negative sampling
    20.5. **Glove** – 25-300 dimensions, 2-840 bn tokens, applied word-word co-occurrence probability to build the embedding. If 2 words co-exist many times, both words may have similar meaning so matrix will be closer
21. **Vector product** – dot product, multiply 2 vectors, result is a scalar
    21.1. Euclidean space – (x1,x2,…xn), (y1,y2,…yn) = (x1y1+x2y2+…xnyn)
    21.2. Euclidean space = cartesian space = n-space or real numbers = n points
22. **Numpy.reshape** – give new shape to array
    22.1. Np.arange(6).reshape((3,2)) = [[0,1], [2,3],[4,5]]
23. **Numpy.ravel** – returns contiguous flattened array
    23.1. X = np.array([1,2,3],[4,5,6]). Np.ravel(x) = [1,2,3,4,5,6] = np.reshape(-1)
24. **^** - denotes estimator like mean. Unit vector (vector of length 1).
    24.1. Growth rate of x in calculus x^ = d lnx = dx/x
25. **∇- inverted delta – Nabla or del**. It means gradient (slope) in vector analysis
    25.1. Generally applied to function of 3 variables ƒ(x1, x2, x3)
    25.2. Differential operator in cartesian coordinates (x,y,z) on 3-d Euclidean space
    25.3. **∇ø(x,y,z) = ∂ø/∂x x^ + ∂ø/∂y y^ + ∂ø/∂z z^**
26. **Logistic regression** – logit model
    26.1. Model probability of a certain class or event - pass/fail. Can be extended to more events as probabilities from 0 to 1
        26.1.1. Log-odds (log of the odds). Function to convert log-odds to probability
        26.1.2. Independent variables can be binary or continuous (multinomial)

26.1.3. Y = f(x1,x2). l = logb(p/1-p) = $B_0 + B_1x_1 + B_2x_2$

26.1.4. p = $b^{B0 + B1x1 + B2x2}$ / $b^{B0 + B1x1 + B2x2}$ +1

26.1.5. p = 1 / (1+ $b^{-(B0 + B1x1 + B2x2)}$)

26.1.6. B_0 – y-intercept. Log-odds when Y=1

26.1.7. B_1 = 1 means increasing x1 by 1 increases log odds by 1. Odds of Y=1 increase by a factor of 10^1

26.1.8. B2 = 2 means increasing x2 by 1 increases log odds by 2. Odds of Y=1 increase by a factor of 10^2

26.2. Binary classification

26.2.1.

26.3. Multinomial classification

26.3.1.

27. **Softmax function – softargmax. Normalized exponential function**

27.1. **$\sigma(z)_i = e^{zi}$ / $\sum_{j=1tok} e^{zj}$.** i = 1 to k. z = (z1, z2, ..., zk) $\in$ R^k (vector space)

27.2. Input vector K, normalizes it to a probability distribution of K probabilities proportional to exponentials of the input numbers

27.3. After applying softmax components will add to 1 and be in range (0,1)

27.3.1. Before they could be -ve and not add up to 1

27.4. Apply exponential function to each element of input vector and normalize by dividing by the sum of all the exponentials

27.5. It is not a smooth maximum. Can be used to represent categorical distribution, multiclass classification, logistic regression, neural networks, naïve bayes

27.6.

28. **Gradient descent – Cost function**

28.1. Say cost function is MSE = $1/n\sum_{i=1 \text{ to } n}(yi-y\hat{})^2$. Minimise this

28.1.1. Y pred = y^ = mx + b. m is slope of line, b is y-intercept

28.2. You go down and up the curve, in fixed steps, find the global minima

28.3. Fixed steps use learning rate and slope at each point to determine direction

28.3.1. Now get partial derivative of m and b at each point

28.3.2. $\partial/\partial m = 2/n\sum_{i=1 \text{ to } n} -x_i\ (y_i-(mx_i + b))$

28.3.3. $\partial/\partial b = 2/n\sum_{i=1 \text{ to } n} -(y_i-(mx_i + b))$

28.4. Steps use learning rate lr

28.4.1. m = m – lr * ∂/∂m

28.4.2. b = b – lr * ∂/∂b

29. **Derivative**. One variable function. Y = f(x) = f(x+Δx)

29.1. Slope between 2 points is (y2-y1)/(x2-x1)

29.2. Slope at a point is small. Use slope = Δy/Δx = derivative. Then shrink to 0

29.3. Slope = f(x+Δx) – f(x)/x+Δx – x

29.4. Then shrink Δx to 0

29.4.1. $f'(x^2) = d/dx\ x^2 = 2x$. Using the power rule

29.4.2. Means slope or rate of change at any point is 2x

29.4.3. x changes to x + Δx. $f(x + \Delta x) = (x + \Delta x)^2 = x^2+2x\Delta x+\Delta x^2$

29.4.4. you get 2x + Δx. as Δx approached 0, we get 2x

30. **Partial derivative.** More than one variable function. Hold some variables constant

30.1. Eg – function for surface depends on x,y

30.1.1. Keep y constant, find slope in x direction

30.1.2. Keep x constant, find slope in y direction

30.2. $f(x,y) = x^2 + y^3$

30.2.1. partial derivative = $\partial f/\partial x = f'x = 2x$. 'del'

30.2.2. $f'y = 3y^2$

30.2.3. derivative of a constant is 0

31. **Exponential Smoothing** –

31.1.

32. Tanh – Hyperbolic tan function

32.1. Analogs of ordinary trigonometric functions. Defined for hyperbola rather than on the circle

32.2. Just as the points (cos *t*, sin *t*) form a circle with a unit radius, the points (cosh *t*, sinh *t*) form the right half of the equilateral hyperbola.

32.3. Tan z = Sin z / Cos z

32.4. Tanh z = Sinh z / Cosh z