CPSC 406: Machine learning activity 2 Due March 19

Akshdeep Sandhu SN(20665148)

In this activity, you will explore some simple data analytics for some machine learning tasks.

Word vectors are an interesting phenomenon that has recently taken the natural language community by storm. Basically, it is a way of mapping words to vectors in $\mathbf{R}^n$ that somehow semantically capture meaning. In this activity, you will download pretrained word vectors and cluster them using the K-means algorithm,

These word vectors can be trained in a number of ways. We will use a subset of a publically available set of word vectors in $\mathbf{R}^{50}$ (specifically, GloVe, from `https://nlp.stanford.edu/projects/glove/`). This dataset contains 400,000 words and their embeddings. However, there are much fewer commonly used english words, so we have pared down the list to the top $\tilde{1}$0,000 most commonly used words for this exercise.

1. **Download data** Download the files `wordlist.txt` and `embeddings.mat`. Load `embeddings.mat` into MATLAB using

```
load embeddings
```

This should load a matrix named `embeddings` of size 9964×50 into your workspace[1].

Load `wordlist.txt` into your workspace by running the commands

```
fid = fopen('wordlist.txt');
data = textscan(fid,'%s');
fclose(fid);
words = data{1};
```

This will create a cellarray called `words` of length 9964, which is an array whose ith cell is the word represented by the ith row of the embeddings matrix.

Your workspace should now look like this:

```
Name               Size              Bytes  Class     Attributes

data                1x1            1247084  cell
embeddings       9964x50           3985600  double
fid                 1x1                  8  double
words            9964x1            1246972  cell
```

2. **Visualizing data** One fun thing to do with word embeddings is to view them in 2-D space. However, our vectors are currently in 50-D space, which is challenging to visualize. One simple way to deal with this is to project it into its most prominent 2-D components, using principal component analysis (PCA).

For a matrix $X \in \mathbf{R}n \times d$, we can project each row into 2-D space by finding the appropriate matrix $V \in \mathbf{R}^{d\times 2}$ where $V^TV = I$ and XV preserves the distances between rows as much as possible. That is, if the ith and jth row of X are vectors that are close but ith and kth row of X are vectors that are far, then the exact thing should happen with the ith, jth, and kth rows of $Y = XV \in \mathbf{R}^{n\times 2}$.

Such a matrix can come from the singular value decomposition (SVD). To get the 2-D representation of the embeddings, we run a sparse SVD and multiply on the right by the right singular vectors:

```
[U,S,V] = svds(embeddings,2);
emb2d = U*sqrt(S);
```

We can now plot all the embeddings by running MATLAB's text command:

```
figure(1)
clf
plot(emb2d(:,1),emb2d(:,2),'linestyle','none')
hold on
text(emb2d(:,1),emb2d(:,2), words)
```

[1]In case your `embeddings.mat` has more rows, take the first 9964 rows

To make things slightly more interpretable, we have prepared a small list of words that fall into clear categories (numbers, locations, associated words) in `plotwords.txt`. To load and filter for these words, run the following

```
fid = fopen('plotwords.txt');
data = textscan(fid,'%s');
fclose(fid);
plotwords = data{1};
toplot = false(n,1); %n is the number of words
for k = 1:n
    word = words{k};
    toplot(k) = sum(strcmpi(word,plotwords))>0;
end

figure(1)
clf
plot(emb2d(toplot,1),emb2d(toplot,2),'linestyle','none')
hold on
text(emb2d(toplot,1),emb2d(toplot,2), words(toplot))
```

3. **Clustering** In machine learning, an important tool in understandng data structure is clustering, in which we identify the groups of data in high dimensional space that are similar. One common mechanism for doing this is K-means clustering.

   You can think of the K-means algorithm as trying to optimize the nonconvex optimization problem

$$\begin{array}{ll}\underset{C\in\mathbf{R}^{K,d},P\in\mathbf{R}^{n,K}}{\text{minimize}} & \|X-PC\|_F^2\\ \text{subject to} & P_{ij}\in\{0,1\},\quad Pe=\mathbf{1}\end{array}$$

   That is, C is a matrix with K rows, where each row is a vector representing the *center* of the kth cluster, and P is a “selection matrix” where each row tells which cluster the ith row of X belongs to. ($P_{ik}=1$ means the ith row of X belongs to cluster k.)

   The K-means algorithm solves this problem by alternatedly minimizing for C and P. Given X and C, P is found by simply going through each row x_i of X and finding the row c_k in C closest to x_i in Euclidean distance. That is, for $i=1,...,n$,

$$P_{ik}=\begin{cases}1 & \text{if } k=\underset{k'=1,\dots,K}{\text{argmin}}\ \|x_i-c_{k'}\|_2\\ 0 & \text{else.}\end{cases}$$

   Given P and X, finding C is just the solution to a least squares problem. Specifically, for each row k of C, the problem separates to

$$\underset{c_k\in\mathbf{R}^d}{\text{minimize}}\quad \|p_k^TX-n_kc_k^T\|_F^2$$

   where p_k is the kth column of P, and P_k^TX selects the columns of X in cluster k. The solution is then

$$c_k=\frac{1}{n_k}X^Tp_k$$

   and n_k are the number of points in cluster k.

   In other words, c_k is the mean of the vectors in X in cluster k.

   Picking some reasonable starting point, code up the K-means algorithm for the 9,964 word embeddings provided in `embeddings.mat`. Run the algorithm for 100 iterations, using some reasonable starting point (for example, a random sampling of K points), and for $K=500$ clusters.

   List some of the interesting clusters you find.

For example, some clusters we found (there are many more)

- giants, rangers, jets, sox, tigers, indians, kings, franchise, bears, angels, pirates, lions, eagles, saints, rays, lightning, twins...
- progress, task, strategic, achieve, consensus, objective, balanced, priorities, tasks, achieving, objectives, coordinate...
- louis, albert, architect, karl, alfred, abraham, leo, wagner, eugene, edgar, luther, isaac, milton, moses, darwin, fr
- de, la, jean, le, du, pierre, et, des, michel, les
- wearing, wear, clothes, dress, dressed, shirt, shoes, suits, shirts, worn, pants, jacket, boots, collar, jackets, jeans, dresses ...
- completed, completion, completing
- york, los, angeles, atlanta, boston, chicago, houston, miami, philadelphia, detroit, toronto, seattle, dallas, minnesota...,
- gmt, pm, edt, noon, est, sm, utc, rss, hrs, cst, pst, cdt, pdt
- toner, stylus, jpeg, gif, jpg, wav, gzip

Code used added at the end. Also in attached Matlab file.

Some clusters that were found:

{'diamond', 'pearl', 'rings', 'diamonds,' 'ruby', 'oz', 'jewel', 'amber', 'gem', 'jade', 'skins', 'beads', 'emerald', 'necklace', 'bracelet', 'earrings', 'bracelets', 'sapphire', 'pendant' }

{ 'disco', 'retro', 'funky', 'karaoke', 'samba', 'combo', 'mime', 'diy', 'mambo', 'porno', 'polyphonic', }

{ signal', 'replacement', 'switch', 'signals', 'automatically', 'switched', 'backup', 'button', 'slot', 'grid', 'rotation', 'install', 'incoming', 'switching', 'commands', 'activated', 'controllers', 'controller', 'receivers', 'manually', 'reset'}

4. Try changing the value of K, initialization tactics, and number of iterations. Discuss one of these changes on your results.

   Answer: Changing the number of iterations had huge impact no the speed of the algorithm. As the number of iterations increased so did the runtimes. An iteresting effect was that lower iterations resulted in the clusters appearing to be 'less resonable', which is to be expected as the greater the number of iterations the better clusters the algorightm is able to find.

Code used for K-means clustering

```
function [clusterCentres,P] = Kmean_Clustering(X,k,iter)
    %Function that preforms k-means clustering
    % param X: data matrix
    % param k: the number of clusters desired
    % param iter: number of iterations you wanted


    %initialize K-means with random centres
    %genrate random indices
    indices = randperm(size(X,1));
    indices = indices(1:k);
    randCentres = X(indices,:);

    %Clustering
    % Goal is to minize ||X - PC||
    %P - selection matrix n x k
    %C - cluster centres matrix k x d (2 in this case)
    % X - data matrix n x d
    n = size(X,1);
    d = size(X,2);

    %Solve once using initialization centres
    C = randCentres;
    P = solveP(X,C,n,d,k);

    for iteration = 1:iter
        %Given P solve for C
        C = solveC(X,P,n,d,k);
        %Given solved C, solve for P
        P = solveP(X,C,n,d,k);

    end

    clusterCentres = C;


end

function P = solveP(X,C,n,d,k)
    %Helper function to solve for P
    %param X: data matrix
    %param C: Cluster centres
    %param n: number of rows
    %param d: number of features
    %param k: number of cluster centres

    %Make zeros matrix for P
    P = zeros(n,d);

    for i  = 1:size(X,1)
    %Solve for P, given C and X
    %for each in X, find which cluster it is nearest to
    min_dist_idx = NaN;
    min_dist = inf;
    %get i-th row of X
    x = X(i,:);
    for j = 1:k
        %get j-th cluster centre and compute distance
        c = C(j,:);
        %get eucldian distane between two vectors
        dist = norm(c - x);
        if (dist < min_dist)
            min_dist = dist;
            min_dist_idx = j;
        end
    end

    %Add 1 to P(i,min_dist_idx).
    %i.e. which cluster centre i-th row of x is nearest to

    P(i,min_dist_idx) = 1;
    end

end


function C = solveC(X,P,n,d,k)
    %Helper function to solve for P
    %param X: data matrix
    %param P: Cluster centres
    %param n: number of rows
    %param d: number of features
    %param k: number of cluster centres

    % Given X and P, computing C
    % Each row of C, minmize the LS problem  (C = k x d)
    %ck = mean of points in cluster
    %nk = number of point in given cluster

    C = zeros(k,d);

    for j=1:k

        p = P(:,j);%get j-th column of P. i.e all points in cluster j
        nk = sum(p); %nk = number of point is j-th cluster
        px = p.*X; %get relevant rows
        sumk = sum(px);
        mean = sumk/nk;
        C(j,:) = mean ;

    end

end
```