LSH_from_Scratch_Assignment

February 19, 2022

1 Implement LSH from scratch

In this assignment, you will implement LSH from scratch and predict the labels of the test data. You will then verify the correctness of the your implementation using a "grader" function/cell (provided by us) which will match your implmentation.

The grader fucntion would help you validate the correctness of your code.

Please submit the final Colab notebook in the classroom ONLY after you have verified your code using the grader function/cell.

NOTE: DO NOT change the "grader" functions or code snippets written by us.Please add your code in the suggested locations.

Ethics Code: 1. You are welcome to read up online resources to implement the code. 2. You can also discuss with your classmates on the implmentation over Slack. 3. But, the code you write and submit should be yours ONLY. Your code will be compared against other stduents' code and online code snippets to check for plagiarism. If your code is found to be plagiarised, you will be awarded zero-marks for all assignments, which have a 10% weightage in the final marks for this course.

1.1 Reading the data from csv file

[1]:
```
# Code to mount google drive in case you are loading the data from your google␣
↪drive
from google.colab import drive
drive.mount('/gdrive')
%cd /gdrive
```

```
Drive already mounted at /gdrive; to attempt to forcibly remount, call
drive.mount("/gdrive", force_remount=True).
/gdrive
```

[2]:
```
# Loading data from csv file
import pandas as pd
data_path = '/gdrive/MyDrive/PGD_UOH_ASSIGNMENT/lsh_assignment_data.csv'
df = pd.read_csv(data_path)
df.head(2)
```

[2]:
```
  category                                               text
0     tech  tv future in the hands of viewers with home th…
```

```
1  business  worldcom boss  left books alone  former worldc…
```

[3]:
```
# Data Overiview
df['category'].value_counts()
```

[3]:
```
sport            509
business         508
politics         415
tech             399
entertainment    384
Name: category, dtype: int64
```

1.1.1 Creating Train and Test Datasets

Note that the labels for test data will not be present in the dataset and hence they are mentioned as NaN.

[4]:
```
# The last 10 rows in the csv file are query points, so loading them into test␣
↪data.
# And loading the reamining points to train_data for which labels are given.

train_data = df.iloc[:-10,:]
test_data = df.iloc[-10:,:]
```

[5]:
```
# For train_data here the labels are in the column named "category".
train_data.head()
```

[5]:
```
        category                                               text
0           tech  tv future in the hands of viewers with home th…
1       business  worldcom boss  left books alone  former worldc…
2          sport  tigers wary of farrell  gamble  leicester say …
3          sport  yeading face newcastle in fa cup premiership s…
4  entertainment  ocean s twelve raids box office ocean s twelve…
```

[6]:
```
test_data
```

[6]:
```
     category                                               text
2215      NaN  junk e-mails on relentless rise spam traffic i…
2216      NaN  top stars join us tsunami tv show brad pitt  r…
2217      NaN  rings of steel combat net attacks gambling is …
2218      NaN  davies favours gloucester future wales hooker …
2219      NaN  beijingers fume over parking fees choking traf…
2220      NaN  cars pull down us retail figures us retail sal…
2221      NaN  kilroy unveils immigration policy ex-chatshow …
2222      NaN  rem announce new glasgow concert us band rem h…
2223      NaN  how political squabbles snowball it s become c…
2224      NaN  souness delight at euro progress boss graeme s…
```

1.2 Custom Implementation

[7]:
```
import pandas as pd
import numpy as np
from tqdm import tqdm
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
```

1.2.1 Instructions:

1. Read in the train_data.
2. Vectorize train_data using sklearns built in tfidf vectorizer.
3. Ignore unigrams and make use of both **bigrams & trigrams** and also limit the **max features** to **4000** and **minimum document frequency** to **10**.
4. After the tfidf vectors are generated as mentioned above, next task is to generate random hyperplanes.
5. Generate **5 random hyperplanes**. And generate the hyperplanes using a random normal distribution with **mean zero and variance 1**.
6. We have set the **numpy random seed to zero**, please do not change it. And then you can make use of **np.random.normal** to generate the vectors for hyperplanes.
7. As mentioned in the course videos, compute the hash function and also the corresponding hash table for it.
8. Once the hash table is generated now take in each of the query points from the test data.
9. Vectorize those query points using the same tfidf vectorizer as mentioned above.
10. Now use the hash function on this query point and fetch all the similar data points from the hashtable.
11. Use cosine similarity to compute **11-Nearest Neighbours** from the list of data points obtained in the above step.bimport nltk from nltk.corpus import stopwords print(stopwords.words('english'))import nltk from nltk.corpus import stopwords print(stopwords.words('english'))
12. Take a majority vote among the 11-Nearest Neighbours and predict the class label for the query point in the test data.
13. **In case of a tie** in the obtained labels from nearest neighbours, you can pick a label after sorting all the labels **alphabetically**(A-Z), i.e. for example labels starting with A would get more preference than labels starting with Z.
14. Repeat steps 9 to 13 for all the points in the test data and then finally return a list with all the predicted labels.
15. Note that there are a total of 10 data points in the test data so the final list you return should be of length 10.
16. Also note that the cosine similarity function should be written from scratch, you should not directly make use of existing libraries.
17. Please use the formula of cosine similarity as explained in the course videos, you can make use of numpy or scipy to calculate dot or norm or transpose.

[8]:
```
# Please implement this fucntion and write your code wherever asked. Do NOT␣
↪change the code snippets provided by us.
```

```python
import numpy as np

def predictLabels (test_data):
    """
    Given the test_data, return the labels for all the rows in the test data.
    Follow the step by step instructions mentioned above.
    """

    np.random.seed(0)

    ##############################################################
    ####  Write YOUR CODE BELOW  as per the above instructions ###
    ##############################################################

    # tfidf-vectorizer with bi-gram and tri-gram of words having min-frequency␣
 ↪of 10 and only consider 4000 features


    vectorizer = TfidfVectorizer(ngram_range=(2,3), min_df=10,␣
 ↪max_features=4000)#, stop_words=stopwords)
    n_grams = vectorizer.fit_transform(train_data["text"])

    # create random planes
    # set the random seed to 0
    np.random.seed(0)

    def create_random_plane(a, dim):
        '''
        This will creat a number of planes of dim-dimesion'''
        w = []
        for i in range(1,a+1):
            w.append(np.random.normal(0,1,dim)) # the dimension of the each␣
 ↪plane will be same as dimension as the each data point in train_data
        return w


    m_planes = create_random_plane(5, n_grams.shape[1])


    def hash_val(vector, m_planes):
        '''
        This function return the hash values by computing the dot product of␣
 ↪vector and m_planes
        vector and m_planes belongs to d-dimension then hash values will "d"␣
 ↪number of char

        # if (w.T).(v) < 0 --> 0
```

```python
        # if (w.T).(v) > 1 --> 1
        '''
        st = ''
        for i in m_planes:
            dot_pr = (vector.dot(i))
            ## as type(vector) -->  scipy.sparse.csr.csr_matrix
            ## type(m_planes[i]) --> np.ndarray
            if dot_pr < 0:
                st += "0"
            elif dot_pr >0:
                st+="1"
        return st


    vec_label = list(range(len(list(train_data["category"]))))

    def hash_create_2(n_grams, vector_label, m_planes):
        '''
        input:
              vectors ---> vector representation of each data point
              vector_label --> vector name of corresponding vector-form in␣
↪vectors
              m_planes ---> these are the randomnly generated planes

        I will create a dict of the given vectors,and store them as hash_value:
↪vectore_name
        '''
        hash_dict = {}

        for i in tqdm(range(len(vector_label))):
            hash_value = hash_val(n_grams[i], m_planes)
            hash_dict.setdefault(hash_value, []).extend([vector_label[i]])
        return hash_dict


    hash_dict = hash_create_2(n_grams, vec_label, m_planes)


    ########################TEST################################

    test_gram = vectorizer.transform(test_data.text).toarray()

    def cosine_sim(vector1, vector2):
        '''
        The function return the cosime similarity between vector1 and vector2
        '''
```

```
        return (vector2.dot(vector1))/(np.linalg.norm(vector1)* np.linalg.
↪norm(vector2))

        new = []

        for i in test_gram:
            new.append([x for x in hash_dict[hash_val(i, m_planes)]])
            ## ## this dict contain all the vector_name that have same␣
↪hash_values as the query_points





    train_vect = n_grams.toarray()
    # convert the n_gram(spare csr.matrix) to numpy.ndarray, for easy␣
↪calculation

    knn_11 = []
    for i in range(0,len(new)):

        cosine_similarity = {}
        for j in range(len(train_vect[new[i]])):
            cos_sim = cosine_sim(test_gram[i], train_vect[new[i][j]])
            ## computing the cosine-sim between all the querry_points with the␣
↪training points(which have same hash_value as the querry point)
            cosine_similarity[new[i][j]] = cos_sim
            ## then store the value in dict PT_NAME:cosine-sim(between xq and␣
↪PT_NAME)
        knn_11.append(cosine_similarity)
        ## Finally append all the dicts in the list, which contain consine-sim␣
↪between xq and the pts which have same hash_value as xq



    def most_frequent(List):      ## source -- https://www.geeksforgeeks.org/
↪python-find-most-frequent-element-in-a-list/
        occurence_count = Counter(List)
        return occurence_count.most_common(1)[0][0]


    pred_class_labels = []
    for i in range(len(knn_11)):
        ## this line sort the dict (which contain all the pts which have␣
↪hash_value same as query point along with their cosine-sim between xq and␣
↪that point)
```

```
        ## the output of this is list, which have point in descending order of␣
↪cosine-sim (between xq and pt)
        top_11_nss = (list(v for k,v in (sorted(((value, key) for (key,value)␣
↪in knn_11[i].items()), reverse=True)[:10])))
        # print(top_11_nss)

        class_label = [ train_data.category.iloc[i] for i in top_11_nss] ##␣
↪find all the labels of the top 11-NN for each querry point
        # print(class_label)
        pred_class_labels.append(most_frequent(class_label))

    return pred_class_labels
```

1.3 Readings/references

https://www.pinecone.io/learn/locality-sensitive-hashing-random-projection/
https://santhoshhari.github.io/Locality-Sensitive-Hashing/

1.4 Grader Cell

Please execute the following Grader cell to verify the correctness of your above implementation. This cell will print "Success" if your implmentation of the predictLabels() is correct, else, it will print "Failed". Make sure you get a "Success" before you submit the code in the classroom.

[9]:
```
############################################
## GRADER CELL: Do NOT Change this.
# This cell will print "Success" if your implmentation of the predictLabels()␣
↪is correct and the accuracy obtained is above 80%.
# Else, it will print "Failed"
############################################
import numpy as np

# Predict the labels using the predictLabels() function
Y_custom = np.array(predictLabels(test_data))

# Reference grader array - DO NOT MODIFY IT
Y_grader = np.array(['tech', 'entertainment', 'tech', 'sport', 'business',␣
↪'business', 'politics', 'entertainment', 'politics', 'sport'])

# Calculating accuracy by comparing Y_grader and Y_custom
accuracy = np.sum(Y_grader==Y_custom) * 10

if accuracy >= 80:
    print("******** Success ********","Accuracy Achieved = ", accuracy,'%')
else:
    print("####### Failed #######","Accuracy Achieved = ", accuracy,'%')
    print("\nY_grader = \n\n", Y_grader)
```

```python
    print("\n","*"*50)
    print("\nY_custom = \n\n", Y_custom)
```

```
100%|      | 2215/2215 [00:00<00:00, 2777.32it/s]

******** Success ******** Accuracy Achieved =  90 %
```