4_Pradeep_Lecture_4_Machine_learning_clustering

March 7, 2020

1 Machine learning Turorial : Classification/ Clustering

[7]:
```
# Libraries
import numpy as np
import pandas as pd
from sklearn import preprocessing, neighbors
from sklearn.model_selection import train_test_split
import pickle

import math
import random

import matplotlib.pyplot as plt
from matplotlib import style
from collections import Counter
import warnings

style.use('fivethirtyeight')
%matplotlib inline
```

2 K nearest neighbors

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

suppose we have two classes and for a new data point which class it belongs:

We find the distances between the nearest k neighbors and the classify the data point based on the number of neighbors that are close to the given point. we can also define confidence interval: suppose if out of three (K=3), two point belongs to class A and 1 to class B. Then the confidence of the point belonging to class A is 66% confidence.

Confience is different from the model accuracy

WE measure simple Euclidean distance. And since the data set is very large this claculation takes lot of time. Larger this dataset this algorithm is not good as compared to other algorithm such as SVM.

We are using dataset from university of califorina: UCI machine learning repository: *archive.ics.uci.edu/ml/datasets.html*

Breast cancer data : downloaded

Data

[8]:
```
# Sklearn   neighbors.KNeighborsClassifier

df = pd.read_csv('./data/breast-cancer-wisconsin.data')
df.tail()
```

[8]:

	id	clump_thickness	unif_cell_size	unif_cell_shape	marg_adhesion	\
694	776715	3	1	1	1	
695	841769	2	1	1	1	
696	888820	5	10	10	3	
697	897471	4	8	6	4	
698	897471	4	8	8	5	

	single_epith_cell	bare_nuclei	bland_chrom	norm_nucleoli	mitosis	class
694	3	2	1	1	1	2
695	2	1	1	1	1	2
696	7	3	8	10	2	4
697	3	4	10	6	1	4
698	4	5	10	4	1	4

[9]:
```
set(list(df['class'].values))
```

[9]:
```
{2, 4}
```

Missing data and Filtering

[3]:
```
# data summary downladed states that there are missing data
# we can also drop the missing data but in most cases we lose significant␣
 ↪amount of information
# most algoritm treats -99999 as an outlier and treat it as an outlier

# replacing missing data (denote by '?' in df) with a -99999 as an outlier

df.replace('?', -99999, inplace=True)
# df.head()
```

[4]:
```
# removing the columns that are not useful
df.drop(['id'],1, inplace=True)
df.head()
```

[4]:

	clump_thickness	unif_cell_size	unif_cell_shape	marg_adhesion	\
0	5	1	1	1	
1	5	4	4	5	

```
2                3                1                1                1
3                6                8                8                1
4                4                1                1                3

   single_epith_cell bare_nuclei  bland_chrom  norm_nucleoli  mitosis  class
0                  2           1            3              1        1      2
1                  7          10            3              2        1      2
2                  2           2            3              1        1      2
3                  3           4            3              7        1      2
4                  2           1            3              1        1      2
```

[5]:
```
# In this problem we are trying to predict whether the cancer is benign or␣
 ↪malignant  given by
# class (2 for benign, 4 for malignant).
# so every thing except the class column is our features
# and label is the class

X = np.array(df.drop(['class'],1))
y = np.array(df['class'])

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)
```

[6]:
```
# defining classifier
clf = neighbors.KNeighborsClassifier()

# fitting classifier
clf.fit(X_train,y_train)

# accuracy
accuracy = clf.score(X_test,y_test)
print(accuracy)

# saving classifier

with open('kneigh_breast.pickle','wb') as f:
    pickle.dump(clf,f)

# # to use classifier  from file
pickle_in = open('kneigh_breast.pickle','rb')
clf_loaded = pickle.load(pickle_in)
```

```
0.9785714285714285
```

[7]:
```
# pridiction for unknown values
example_measures = np.array([[3,1,2,1,2,1,1,2,3],[20,4,4,3,3,1,1,2,3]])
# example_measures = example_measures.reshape(len(example_measures),-1)
prediction = clf_loaded.predict(example_measures)
```

```
print(prediction)
```

```
[2 4]
```

Writing our own Knearest neighbors algorithm <center

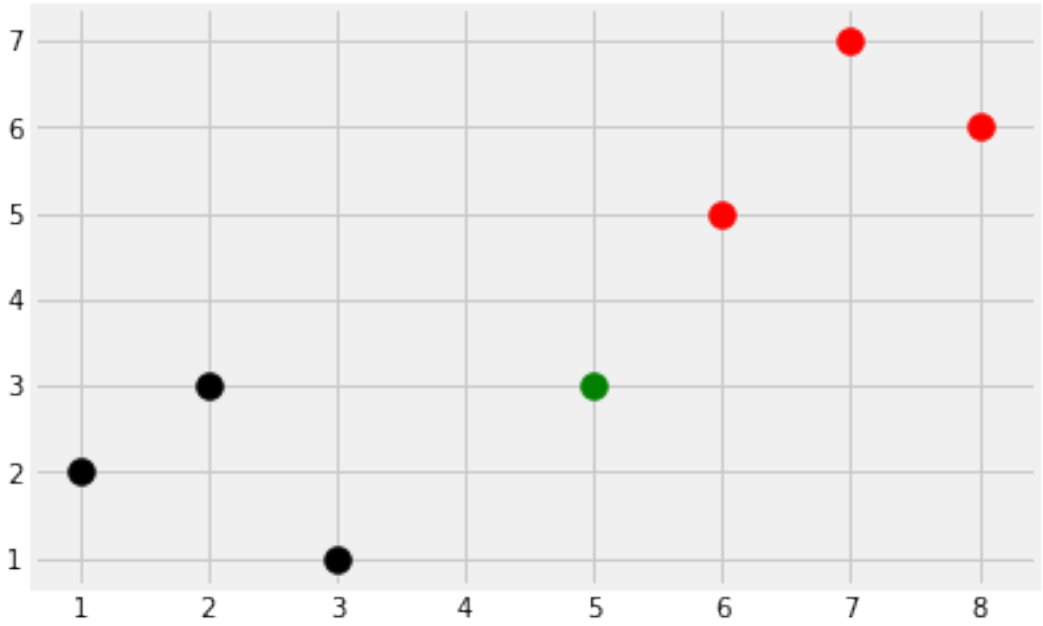
[10]:
```
def K_nearest_neighbor(data_in, data_to_predict, k =3):
    '''calculate the nearest neighbor distance
    data_in:  actual data with classes (has to be a dictionary)
    data_to_predict: data point/ points to be predicted or classifed into␣
↪classes
    k : K value
    '''
    if len(data_in) >= k:  # classes should not be less than K
        warnings.warn('K is set to a value less than total  classes')

    #to do this we have to compare and calculate the distance of the given␣
↪point with all the other data
    #points. An efficient way to do it is check the points in the  radius of␣
↪the given point and then calculate the
    # eculidian distance

    # we calcuialte the list of all distance from the datapoint
    distances = []
    for group in data_in:
        for features in data_in[group]:
            euclidean_distance = np.linalg.norm(np.array(features)-np.
↪array(data_to_predict))

            distances.append([euclidean_distance, group])

    votes = [i[1] for i in sorted(distances)[:k]] # sort the euclidean distance␣
↪and take class for first k values
    #print(Counter(votes))
    vote_result= Counter(votes).most_common(1)[0][0]  # using counter to count␣
↪the max values for a class

    # confidence of the outcome  (by calculating number of votes in favor)␣
↪divided by total votes or k values
    confidence = 1.0*Counter(votes).most_common(1)[0][1]/k
    #print("result is %d with confidence %f" %(vote_result,confidence))

    return vote_result, confidence
```

[11]:
```
# define dataset  as a dictionary with two classes "k" and "r" with the␣
↪features.
```

```
dataset = {'k':[[1,2],[2,3],[3,1]], 'r':[[6,5],[7,7],[8,6]]}
new_features = [5,7]  # which this point belongs to either k or r
```

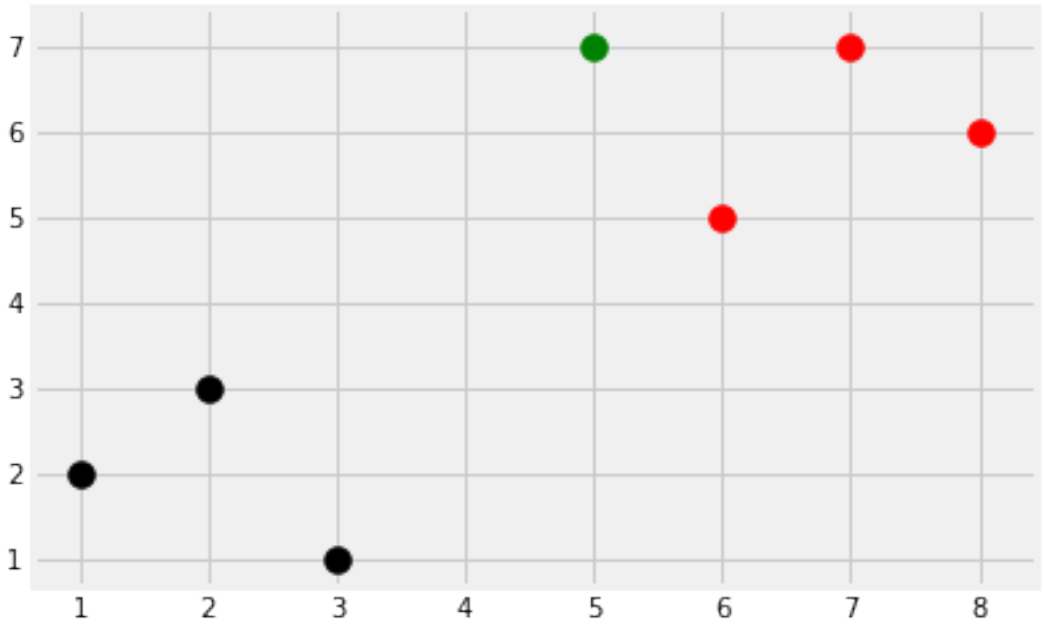
[13]:
```
# define dataset  as a dictionary with two classes "k" and "r" with the features.

dataset = {'k':[[1,2],[2,3],[3,1]], 'r':[[6,5],[7,7],[8,6]]}
new_features = [5,3]  # which this point belongs to either k or r


result, confidence = K_nearest_neighbor(dataset, new_features, k=3)
print("Class predicted as %s" %(result))
print("confidence %f" %(confidence))
for i in dataset:
    for ii in dataset[i]:
        plt.scatter(ii[0],ii[1], s=100, color = i)
plt.scatter(new_features[0], new_features[1], color ='g', s =100)
plt.show()
```

```
Class predicted as k
confidence 0.666667
```



[11]:
```
# one line code
[[plt.scatter(ii[0],ii[1], s =100, color =i) for ii in dataset[i]] for i in dataset]
plt.scatter(new_features[0], new_features[1], color ='g', s =100)
```

[11]: <matplotlib.collections.PathCollection at 0x7f327fb78b38>



[14]:
```
# using our self algorithm in breast cancer dataset

full_data = df.astype(float).values.tolist()  # some data are represented as␣
↪string (like '1') and
#therefore converting all points to float

random.shuffle(full_data)  # shuffling order of the list to remove biases

test_size = 0.2

train_data = full_data[:-int(test_size*len(full_data))]  # slicing data for␣
↪training  first 80 percent data
test_data = full_data[-int(test_size*len(full_data)):]

train_set = {2:[], 4:[]}  # to create dictionary for training set
test_set = {2:[],4:[]}


for i in train_data:
    train_set[i[-1]].append(i[:-1])  # appending train set based on the value␣
↪of last column
    #of train data  upto second last column
```

```
for i in test_data:
    test_set[i[-1]].append(i[:-1])  # appending train set based on the value of␣
 ↪last column
    #of train data  upto second last column

correct = 0.0  # to count the correct prediction
total = 0.0

for group in test_set:
    for data in test_set[group]:
        vote, confidence = K_nearest_neighbor(train_set, data , k=5)

        if group == vote:
            correct += 1
        else:
            print(confidence)
        total += 1
accuracy_our =  (correct/total)

print("the accuracy of our method is %f" %(accuracy_our))
```

```
      ␣
↪---------------------------------------------------------------------------

       ValueError                                Traceback (most recent call␣
↪last)

       <ipython-input-14-5c7973f789a9> in <module>
         1 # using our self algorithm in breast cancer dataset
         2
   ----> 3 full_data = df.astype(float).values.tolist()  # some data are␣
↪represented as string (like '1') and
         4 #therefore converting all points to float
         5


       /usr/local/lib/python3.6/dist-packages/pandas/core/generic.py in␣
↪astype(self, dtype, copy, errors)
      5696         else:
      5697             # else, only a single dtype is given
   -> 5698             new_data = self._data.astype(dtype=dtype, copy=copy,␣
↪errors=errors)
      5699             return self._constructor(new_data).__finalize__(self)
      5700
```

```
       /usr/local/lib/python3.6/dist-packages/pandas/core/internals/managers.py␣
↪in astype(self, dtype, copy, errors)
       580
       581     def astype(self, dtype, copy: bool = False, errors: str =␣
↪"raise"):
   --> 582         return self.apply("astype", dtype=dtype, copy=copy,␣
↪errors=errors)
       583
       584     def convert(self, **kwargs):


       /usr/local/lib/python3.6/dist-packages/pandas/core/internals/managers.py␣
↪in apply(self, f, filter, **kwargs)
       440                 applied = b.apply(f, **kwargs)
       441             else:
   --> 442                 applied = getattr(b, f)(**kwargs)
       443             result_blocks = _extend_blocks(applied, result_blocks)
       444


       /usr/local/lib/python3.6/dist-packages/pandas/core/internals/blocks.py␣
↪in astype(self, dtype, copy, errors)
       623             vals1d = values.ravel()
       624             try:
   --> 625                 values = astype_nansafe(vals1d, dtype, copy=True)
       626             except (ValueError, TypeError):
       627                 # e.g. astype_nansafe can fail on object-dtype of␣
↪strings


       /usr/local/lib/python3.6/dist-packages/pandas/core/dtypes/cast.py in␣
↪astype_nansafe(arr, dtype, copy, skipna)
       895     if copy or is_object_dtype(arr) or is_object_dtype(dtype):
       896         # Explicit copy, or required since NumPy can't view from /␣
↪to object.
   --> 897         return arr.astype(dtype, copy=True)
       898
       899     return arr.view(dtype)


       ValueError: could not convert string to float: '?'
```

[13]:
```
# doining exp multiple times and taking average accuracies

accuracies = []
```

```
for sam_i in range(20):

    full_data = df.astype(float).values.tolist()  # some data are represented␣
↪as string (like '1') and
    #therefore converting all points to float

    random.shuffle(full_data)  # shuffling order of the list to remove biases

    test_size = 0.2

    train_data = full_data[:-int(test_size*len(full_data))]  # slicing data for␣
↪training  first 80 percent data
    test_data = full_data[-int(test_size*len(full_data)):]

    train_set = {2:[], 4:[]}  # to create dictionary for training set
    test_set = {2:[],4:[]}


    for i in train_data:
        train_set[i[-1]].append(i[:-1])  # appending train set based on the␣
↪value of last column
        #of train data  upto second last column



    for i in test_data:
        test_set[i[-1]].append(i[:-1])  # appending train set based on the␣
↪value of last column
        #of train data  upto second last column

    correct = 0.0  # to count the correct prediction
    total = 0.0

    for group in test_set:
        for data in test_set[group]:
            vote, confidence = K_nearest_neighbor(train_set, data , k=5)

            if group == vote:
                correct += 1
#             else:
#                 print(confidence)
            total += 1
    accuracy_our =  (correct/total)

    print("the accuracy of our method is %f" %(accuracy_our))

    accuracies.append(correct/total)
```

```
print("Averace accuracy is %f" %(sum(accuracies)/len(accuracies)))
```

```
the accuracy of our method is 0.985612
the accuracy of our method is 0.978417
the accuracy of our method is 0.978417
the accuracy of our method is 0.971223
the accuracy of our method is 0.956835
the accuracy of our method is 0.978417
the accuracy of our method is 0.964029
the accuracy of our method is 0.935252
the accuracy of our method is 0.964029
the accuracy of our method is 0.956835
the accuracy of our method is 0.971223
the accuracy of our method is 0.978417
the accuracy of our method is 0.992806
the accuracy of our method is 0.942446
the accuracy of our method is 0.971223
the accuracy of our method is 0.978417
the accuracy of our method is 0.964029
the accuracy of our method is 0.949640
the accuracy of our method is 0.978417
the accuracy of our method is 0.978417
Averace accuracy is 0.968705
```

[ ]: