lab-6-Naive-Bayes-KNN

April 8, 2023

0.1 Naive Bayes

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

$$P(Class|Sample) = \frac{P(Sample|Class) \times P(Class)}{P(Sample)}$$

$$posterior = \frac{likelihood \times prior}{evidence}$$

$$P(c_i|x_j) \propto P(x_j|c_i) \times P(c_i)$$

For categorical features, we can use a Multinouilli distribution, where μ_{ic} is an histogram over the possible values for x_i in class c :

$$P(\mathbf{x}|c) = \prod_{i=1}^{D} Cat(x_j|\mu_{jc})$$

During Training: - Compute the prior probability i.e $p(c_i)$ the proportion of samples inside each class of the whole training set. - for each feature: - if the feature is categorical, compute $p(x_j|c_i)$ for $j = 1, 2 \dots D$ and $i = 1, 2 \dots C$ - for each possible values of this feature in the training samples of class c_i compute the probability that this feature appear in class c_i

To Predict - Compute $p(c_i|\mathbf{x})$ - Multiply the prior of each class $p(c_i)$ by - for each feature **k**: - if categorical, multiply by the probabilities calculated earlier, $p(x_k|c_i)$ where x_k is the value of the input on feature k - return the highest probability $p(c_i|x)$ of all classes

[2]:
```
df = pd.read_csv("PlayTennis.csv")
msk = np.random.rand(len(df)) < 0.8
train = df[msk]
test = df[~msk]
train.head()
```

[2]:

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No

```
2  Overcast         Hot     High    Weak         Yes
4      Rain        Cool   Normal    Weak         Yes
5      Rain        Cool   Normal  Strong          No
```

[3]:
```
test.head()
```

[3]:
```
   Outlook Temperature Humidity    Wind Play Tennis
3     Rain        Mild     High    Weak         Yes
7    Sunny        Mild     High    Weak          No
9     Rain        Mild   Normal    Weak         Yes
13    Rain        Mild     High  Strong          No
```

[4]:
```
# create the dict data structure
_class = 'Play Tennis'
di = {}
for class_label in train[_class].unique():
    di[class_label] = {}
    for feature in train.columns:
        if feature != _class:
            di[class_label][feature] = {}
            for item in train[feature].unique():
                di[class_label][feature][item] = 0
```

[5]:
```
di
```

[5]:
```
{'No': {'Outlook': {'Sunny': 0, 'Overcast': 0, 'Rain': 0},
  'Temperature': {'Hot': 0, 'Cool': 0, 'Mild': 0},
  'Humidity': {'High': 0, 'Normal': 0},
  'Wind': {'Weak': 0, 'Strong': 0}},
 'Yes': {'Outlook': {'Sunny': 0, 'Overcast': 0, 'Rain': 0},
  'Temperature': {'Hot': 0, 'Cool': 0, 'Mild': 0},
  'Humidity': {'High': 0, 'Normal': 0},
  'Wind': {'Weak': 0, 'Strong': 0}}}
```

[6]:
```
# code to mimic training
classLabel = 'Yes'

for classLabel in ['Yes', 'No']:
    for feature in train.columns:
            if feature != _class:
                for item in train[feature].unique():
                    numr = len(train[ (train[feature] == item) & (train[_class]␣
 ↪== classLabel) ])
                    denr = len(train[train[_class] == classLabel])
                    di[classLabel][feature][item] = numr/denr
```

[7]:
```
di
```

[7]:
```
{'No': {'Outlook': {'Sunny': 0.6666666666666666,
   'Overcast': 0.0,
   'Rain': 0.3333333333333333},
  'Temperature': {'Hot': 0.6666666666666666,
   'Cool': 0.3333333333333333,
   'Mild': 0.0},
  'Humidity': {'High': 0.6666666666666666, 'Normal': 0.3333333333333333},
  'Wind': {'Weak': 0.3333333333333333, 'Strong': 0.6666666666666666}},
 'Yes': {'Outlook': {'Sunny': 0.2857142857142857,
   'Overcast': 0.5714285714285714,
   'Rain': 0.14285714285714285},
  'Temperature': {'Hot': 0.2857142857142857,
   'Cool': 0.42857142857142855,
   'Mild': 0.2857142857142857},
  'Humidity': {'High': 0.2857142857142857, 'Normal': 0.7142857142857143},
  'Wind': {'Weak': 0.5714285714285714, 'Strong': 0.42857142857142855}}}
```

[8]:
```
# Code to Mimic Testing
P_Yes = len(train[train[_class] == 'Yes'])/len(train)
P_No = len(train[train[_class] == 'No'])/len(train)
print(P_Yes, P_No)
```

```
0.7 0.3
```

[9]:
```
yes = []
no = []

for Label in ['Yes', 'No']:
    for index, row in test.iterrows():
        value = P_Yes if Label == 'Yes' else P_No
        for feature in test.columns:
            if feature != _class:
                value*=di[Label][feature][row[feature]]
        yes.append(value) if Label == 'Yes' else no.append(value)
```

[10]:
```
yes
```

[10]:
```
[0.00466472303206997,
 0.00932944606413994,
 0.011661807580174925,
 0.0034985422740524776]
```

[11]:
```
no
```

[11]:
```
[0.0, 0.0, 0.0, 0.0]
```

[12]:
```
print("Predicted Labels")
predicted_labels = []
for item in zip(yes, no):
    predicted_labels.append("Yes" if item[0]>item[1] else "No")
    #print("Yes") if item[0]>item[1] else print("No")
predicted_labels
```

```
Predicted Labels
```

[12]:
```
['Yes', 'Yes', 'Yes', 'Yes']
```

[13]:
```
print("True Labels")
list(test[_class])
```

```
True Labels
```

[13]:
```
['Yes', 'No', 'Yes', 'No']
```

[14]:
```
# Accuracy
sum(1 for x,y in zip(list(test[_class]),predicted_labels) if x == y) /␣
 ↪float(len(predicted_labels))
```

[14]:
```
0.5
```

0.1.1 Lets compare our result with sklearn

[15]:
```
## with sklearn
from sklearn import preprocessing
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import plot_confusion_matrix
```

[16]:
```
le = preprocessing.LabelEncoder()
data_train_df = pd.DataFrame(train)
data_train_df_encoded = data_train_df.apply(le.fit_transform)

data_test_df = pd.DataFrame(test)
data_test_df_encoded = data_test_df.apply(le.fit_transform)
```

[17]:
```
x_train = data_train_df_encoded.drop(['Play Tennis'],axis=1)
y_train = data_train_df_encoded['Play Tennis']

x_test = data_test_df_encoded.drop(['Play Tennis'],axis=1)
y_test = data_test_df_encoded['Play Tennis']
```

[18]:
```
model = GaussianNB()
nbtrain = model.fit(x_train, y_train)

y_pred = nbtrain.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5
```

0.2 K-NN

[22]:
```
# data
df = pd.read_csv("PlayTennis.csv")
df.head()
_class = 'Play Tennis'
```

Overview: https://www.cs.hmc.edu/~yjw/teaching/cs158/lectures/03_kNN.pdf

During Training: - Find k nearest neighbors of x. - Choose as label the majority label within k nearest neighbors.

How do we measure **nearest**? - Euclidean distance metric? But wont work for categorical features? - Hamming distance? See: https://www.ibm.com/topics/knn

[4]:
```
df['Outlook'].unique()
```

[4]:
```
array(['Sunny', 'Overcast', 'Rain'], dtype=object)
```

[5]:
```
df['Temperature'].unique()
```

[5]:
```
array(['Hot', 'Mild', 'Cool'], dtype=object)
```

[6]:
```
df['Humidity'].unique()
```

[6]:
```
array(['High', 'Normal'], dtype=object)
```

[7]:
```
df['Wind'].unique()
```

[7]:
```
array(['Weak', 'Strong'], dtype=object)
```

[ ]:
```
## Convert "Wind" data to 0 and 1
```

[8]:
```
df['Wind'] = np.where(df['Wind'] == 'Weak', 0, 1)
```

[9]:
```
df
```

[9]:

	Outlook	Temperature	Humidity	Wind	Play Tennis
0	Sunny	Hot	High	0	No
1	Sunny	Hot	High	1	No
2	Overcast	Hot	High	0	Yes

```
3       Rain        Mild     High     0         Yes
4       Rain        Cool   Normal     0         Yes
5       Rain        Cool   Normal     1          No
6   Overcast        Cool   Normal     1         Yes
7      Sunny        Mild     High     0          No
8      Sunny        Cool   Normal     0         Yes
9       Rain        Mild   Normal     0         Yes
10     Sunny        Mild   Normal     1         Yes
11  Overcast        Mild     High     1         Yes
12  Overcast         Hot   Normal     0         Yes
13      Rain        Mild     High     1          No
```

[ ]:
```
## Convert "Humidity" data to 0 and 1
```

[10]:
```
df['Humidity'] = np.where(df['Humidity'] == 'Normal', 0, 1)
```

[11]:
```
df
```

[11]:
```
     Outlook Temperature  Humidity  Wind Play Tennis
0      Sunny         Hot         1     0          No
1      Sunny         Hot         1     1          No
2   Overcast         Hot         1     0         Yes
3       Rain        Mild         1     0         Yes
4       Rain        Cool         0     0         Yes
5       Rain        Cool         0     1          No
6   Overcast        Cool         0     1         Yes
7      Sunny        Mild         1     0          No
8      Sunny        Cool         0     0         Yes
9       Rain        Mild         0     0         Yes
10     Sunny        Mild         0     1         Yes
11  Overcast        Mild         1     1         Yes
12  Overcast         Hot         0     0         Yes
13      Rain        Mild         1     1          No
```

[12]:
```
outlook = pd.crosstab(index = df.index,columns=df.Outlook)
```

[13]:
```
df = df.join(outlook)
```

[14]:
```
df
```

[14]:
```
     Outlook Temperature  Humidity  Wind Play Tennis  Overcast  Rain  Sunny
0      Sunny         Hot         1     0          No         0     0      1
1      Sunny         Hot         1     1          No         0     0      1
2   Overcast         Hot         1     0         Yes         1     0      0
3       Rain        Mild         1     0         Yes         0     1      0
4       Rain        Cool         0     0         Yes         0     1      0
5       Rain        Cool         0     1          No         0     1      0
```

```
6   Overcast        Cool          0     1          Yes          1     0      0
7      Sunny        Mild          1     0           No          0     0      1
8      Sunny        Cool          0     0          Yes          0     0      1
9       Rain        Mild          0     0          Yes          0     1      0
10     Sunny        Mild          0     1          Yes          0     0      1
11  Overcast        Mild          1     1          Yes          1     0      0
12  Overcast         Hot          0     0          Yes          1     0      0
13      Rain        Mild          1     1           No          0     1      0
```

[15]:
```
temperature = pd.crosstab(index = df.index,columns=df.Temperature)
```

[16]:
```
df = df.join(temperature)
```

[17]:
```
df
```

[17]:
```
     Outlook Temperature  Humidity  Wind Play Tennis  Overcast  Rain  Sunny  \
0      Sunny         Hot         1     0          No         0     0      1
1      Sunny         Hot         1     1          No         0     0      1
2   Overcast         Hot         1     0         Yes         1     0      0
3       Rain        Mild         1     0         Yes         0     1      0
4       Rain        Cool         0     0         Yes         0     1      0
5       Rain        Cool         0     1          No         0     1      0
6   Overcast        Cool         0     1         Yes         1     0      0
7      Sunny        Mild         1     0          No         0     0      1
8      Sunny        Cool         0     0         Yes         0     0      1
9       Rain        Mild         0     0         Yes         0     1      0
10     Sunny        Mild         0     1         Yes         0     0      1
11  Overcast        Mild         1     1         Yes         1     0      0
12  Overcast         Hot         0     0         Yes         1     0      0
13      Rain        Mild         1     1          No         0     1      0

    Cool  Hot  Mild
0      0    1     0
1      0    1     0
2      0    1     0
3      0    0     1
4      1    0     0
5      1    0     0
6      1    0     0
7      0    0     1
8      1    0     0
9      0    0     1
10     0    0     1
11     0    0     1
12     0    1     0
13     0    0     1
```

[ ]:
```
# drop "outlook" and "temperature" column since we have croostab-ed those features
```

[18]:
```
df = df.drop(['Outlook','Temperature'], axis=1)
```

[19]:
```
df
```

[19]:

	Humidity	Wind	Play Tennis	Overcast	Rain	Sunny	Cool	Hot	Mild
0	1	0	No	0	0	1	0	1	0
1	1	1	No	0	0	1	0	1	0
2	1	0	Yes	1	0	0	0	1	0
3	1	0	Yes	0	1	0	0	0	1
4	0	0	Yes	0	1	0	1	0	0
5	0	1	No	0	1	0	1	0	0
6	0	1	Yes	1	0	0	1	0	0
7	1	0	No	0	0	1	0	0	1
8	0	0	Yes	0	0	1	1	0	0
9	0	0	Yes	0	1	0	0	0	1
10	0	1	Yes	0	0	1	0	0	1
11	1	1	Yes	1	0	0	0	0	1
12	0	0	Yes	1	0	0	0	1	0
13	1	1	No	0	1	0	0	0	1

[20]:
```
msk = np.random.rand(len(df)) < 0.8
train = df[msk]
test = df[~msk]
train.head()
```

[20]:

	Humidity	Wind	Play Tennis	Overcast	Rain	Sunny	Cool	Hot	Mild
0	1	0	No	0	0	1	0	1	0
1	1	1	No	0	0	1	0	1	0
2	1	0	Yes	1	0	0	0	1	0
3	1	0	Yes	0	1	0	0	0	1
4	0	0	Yes	0	1	0	1	0	0

[23]:
```
train_x = train.loc[ : , train.columns!=_class]
train_y = train[['Play Tennis']]
test_x = test.loc[ : , test.columns!=_class]
test_y = test[['Play Tennis']]
```

[24]:
```
def hammingDistance(x1, x2):
    '''
    x1 and x2 are vectors about which the hamming distance is calculated
    '''
    return np.sum(np.abs(x1-x2))
```

How to design the KNN algorithm? To use the distance measure in KNN to segregate

datapoints, follow these steps:

- Choose a distance metric: There are several distance metrics that can be used in KNN, such as Euclidean distance, Manhattan distance, and Minkowski distance. Choose the one that is most suitable for your data.
- Determine the value of k: The value of k is the number of nearest neighbors to consider. This can be determined through cross-validation or other methods.
- Calculate the distance between the test datapoint and each training datapoint: Using the chosen distance metric, calculate the distance between the test datapoint and each training datapoint.
- Select the k-nearest neighbors: Select the k-nearest neighbors to the test datapoint based on the calculated distances.
- Assign a label to the test datapoint: Assign a label to the test datapoint based on the majority class among the k-nearest neighbors.
- Repeat steps 3-5 for each test datapoint: Repeat steps 3-5 for each test datapoint to classify the entire test dataset.

[25]:
```
k = 3
for i in range(len(test_x)):
    distance = []
    test_point = np.array(test_x.iloc[i])
    for j in range(len(train_x)):
        train_point = np.array(train_x.iloc[j])
        distance.append(hammingDistance(test_point, train_point))

    # show only the k minumum distances
    idx = np.argpartition(distance, k)

    # get the majority vote
    values, counts = np.unique(np.array(train_y.iloc[idx[:k]]).flatten(),␣
↪return_counts=True)
    ind = np.argmax(counts)
    print(values[ind])  # prints the most frequent element
```

```
Yes
Yes
```

[26]:
```
test_y
```

[26]:

	Play Tennis
6	Yes
13	No

[ ]:
```

```