Naive Bayes classifiers

Naive Bayes with categorical feature from scratch

The features are assumed to be generated from a simple multinomial distribution. The multinomial distribution describes the probability of observing counts among a number of categories, and thus multinomial naive Bayes is most appropriate for features that represent counts or count rates. We aim at modeling the data distribution with a best-fit multinomial distribution.

In [ ]:

```
"""

First let us design a simple dataset taking into account weather related features:
    - outlook in {'sunny', 'overcast', 'rainy'}
    - temp: level of temperature {'hot','cool','mild'}
    - humidity: level of humidity {'high', 'normal'}
    - windy: either it is windy are not {'yes','no'}
Then we aim at predicting either a tennis match can be played or not. "play" in {'yes','no'}.


"""
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn import preprocessing

outlook = ["sunny", "overcast", "rain"]
temperature = ["hot", "mild", "cold"]
humidity = ["high", "normal"]
wind = ["yes", "no"]
play = ["yes", "no"]

dataset = [
    ["outlook", "temp", "humidity", "windy", "play"],
    ["sunny", "hot", "high", "no", "no"],
    ["sunny", "hot", "high", "yes", "no"],
    ["overcast", "hot", "high", "no", "yes"],
    ["rainy", "mild", "high", "no", "yes"],
    ["rainy", "cool", "normal", "no", "yes"],
    ["rainy", "cool", "normal", "yes", "no"],
```

```
    ["overcast", "cool", "normal", "yes", "yes"],
    ["sunny", "mild", "high", "no", "no"],
    ["sunny", "cool", "normal", "no", "yes"],
    ["rainy", "mild", "normal", "no", "yes"],
    ["sunny", "mild", "normal", "yes", "yes"],
    ["overcast", "mild", "high", "yes", "yes"],
    ["overcast", "hot", "normal", "no", "yes"],
    ["rainy", "mild", "high", "yes", "no"],
]

df = pd.DataFrame(dataset[1:], columns=dataset[0])
df.head()
```

Out[ ]:

	outlook	temp	humidity	windy	play
0	sunny	hot	high	no	no
1	sunny	hot	high	yes	no
2	overcast	hot	high	no	yes
3	rainy	mild	high	no	yes
4	rainy	cool	normal	no	yes

[To do Students]: Complete the following functions

In [ ]:

```
def prior_probability(play_outcome):
    """
    input:
        - play_outcome: string taking values in ['yes','no']
    output:
        - prior probability P(play = play_outcome)
    """
    return (df["play"] == play_outcome).mean()


def likelihood(feature_name, feature_value, play_outcome):
    """
    inputs:
        feature_name: string with values in df column names
        feature_value: given value of the variable corresponding to feature_name
        play_outcome: outcome of target variable "play"
    output:
```

```
        Compute the conditional probability P(feature_name = feature_value|play= play_outcome)
    """
    df_play_outcome = df[df["play"] == play_outcome]
    return (df_play_outcome[feature_name] == feature_value).mean()


def predict_play_outcome(outlook, temp, humidity, windy):
    """
    inputs:
        outlook: value of outlook for a given observation
        temp: value of outlook for a given observation
        humidity: value of outlook for a given observation
        windy: value of outlook for a given observation
    Outputs:
        predicted label by multinomial naive bayes for the given observation (outlook,temp,humidity,windy)"""

    P_yes = (
        prior_probability("yes")
        * likelihood("outlook", outlook, "yes")
        * likelihood("temp", temp, "yes")
        * likelihood("humidity", humidity, "yes")
        * likelihood("windy", windy, "yes")
    )
    P_no = (
        prior_probability("no")
        * likelihood("outlook", outlook, "no")
        * likelihood("temp", temp, "no")
        * likelihood("humidity", humidity, "no")
        * likelihood("windy", windy, "no")
    )
    return dict(zip(["yes", "no"], [P_yes, P_no]))


predict_play_outcome(
    outlook="sunny",
    temp="cool",
    humidity="high",
    windy="yes",
)
```

Out[ ]:
```
{'yes': 0.005291005291005291, 'no': 0.02057142857142857}
```

The same using sklearn library

All features are categorical, we need to use Multinomial Naive Bayes.

- Step 1: encode the feature in categories (MultinomialNB doesn't work with string)
- Step 2: fit MultinomialNB
- Step 3: predict with MultinomialNB

[To do Students]:

- Learn multinomial Naive Bayes using sklearn without and with Laplace smoothing. Plus detail available model hyperparameters of the sklearn implementation.
- Compare predicted probabilities and label on df_test
- Design an example to highlight importance of Laplace smoothing.

In [ ]:

```
from sklearn.preprocessing import LabelEncoder

# Build new item for prediction
df_test = pd.DataFrame(data=[['sunny', 'cool', 'high', 'yes']],
                    columns=['outlook', 'temp', 'humidity', 'windy'])

# instantiate labelencoder object
le = {col: LabelEncoder() for col in df.columns}
print(le)

df_enc = pd.DataFrame()
df_test_enc = pd.DataFrame()
for col in df.columns:
    df_enc[col] = le[col].fit_transform(df[col])
    # Encode test df
    if col != "play":
        df_test_enc[col] = le[col].transform(df_test[col])
# fit_transform the dataset
display(df.head())
display(df_enc.head())
display(df_test)
display(df_test_enc)
```

```
{'outlook': LabelEncoder(), 'temp': LabelEncoder(), 'humidity': LabelEncoder(), 'windy': LabelEncoder(), 'play': Labe
lEncoder()}
```

	outlook	temp	humidity	windy	play
0	sunny	hot	high	no	no
1	sunny	hot	high	yes	no
2	overcast	hot	high	no	yes
3	rainy	mild	high	no	yes
4	rainy	cool	normal	no	yes

	outlook	temp	humidity	windy	play
0	2	1	0	0	0
1	2	1	0	1	0
2	0	1	0	0	1
3	1	2	0	0	1
4	1	0	1	0	1

	outlook	temp	humidity	windy
0	sunny	cool	high	yes

	outlook	temp	humidity	windy
0	2	0	0	1

In [ ]:

```
from sklearn.naive_bayes import MultinomialNB

for laplace_smoothing in [0, 1, 2, 5, 10]:
    print(f"\n--> Laplace smoothing = {laplace_smoothing}")
    mnb = MultinomialNB(alpha=laplace_smoothing)
    mnb.fit(df_enc[[col for col in df_enc.columns if col != "play"]], df["play"])
    print(dict(zip(mnb.classes_, *mnb.predict_proba(df_test_enc))))
    print(f"prediction: {(mnb.predict(df_test_enc))[0]}")
```

```
--> Laplace smoothing = 0
{'no': 0.6862080038678134, 'yes': 0.3137919961321867}
prediction: no

--> Laplace smoothing = 1
{'no': 0.6406632947915922, 'yes': 0.35933670520840805}
prediction: no

--> Laplace smoothing = 2
{'no': 0.6053306056287737, 'yes': 0.39466939437122645}
prediction: no

--> Laplace smoothing = 5
{'no': 0.53630212527358, 'yes': 0.4636978747264203}
prediction: no

--> Laplace smoothing = 10
{'no': 0.4785545044821437, 'yes': 0.5214454955178565}
prediction: yes
/home/joris/.local/lib/python3.10/site-packages/sklearn/naive_bayes.py:555: UserWarning: alpha too small will result
in numeric errors, setting alpha = 1.0e-10
  warnings.warn(
```

We notice that the prediction changes when we set the Laplace Smoothing Parameter to 10.

Multinomial Naive Bays: text classification

One place where multinomial naive Bayes is often used is in text classification, where the features are related to word counts or frequencies within the documents to be classified. We discussed the extraction of such features from text in Feature Engineering; here we will use the sparse word count features from the 20 Newsgroups corpus to show how we might classify these short documents into categories.

Let's download the data and take a look at the target names:

In [ ]:
```python
from sklearn.datasets import fetch_20newsgroups

data = fetch_20newsgroups()
data.target_names
```

Out[ ]:
```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

For simplicity here, we will select just a few of these categories, and download the training and testing set:

In [ ]:
```
categories = ['talk.religion.misc', 'soc.religion.christian',
              'sci.space', 'comp.graphics']
train = fetch_20newsgroups(subset='train', categories=categories)
test = fetch_20newsgroups(subset='test', categories=categories)
```

In [ ]:
```
print(train.data[5])
```

```
From: dmcgee@uluhe.soest.hawaii.edu (Don McGee)
Subject: Federal Hearing
Originator: dmcgee@uluhe
Organization: School of Ocean and Earth Science and Technology
Distribution: usa
Lines: 10


Fact or rumor....?  Madalyn Murray O'Hare an atheist who eliminated the
use of the bible reading and prayer in public schools 15 years ago is now
going to appear before the FCC with a petition to stop the reading of the
Gospel on the airways of America.  And she is also campaigning to remove
Christmas programs, songs, etc from the public schools.  If it is true
then mail to Federal Communications Commission 1919 H Street Washington DC
20054 expressing your opposition to her request.  Reference Petition number

2493.
```

In order to use this data for machine learning, we need to be able to convert the content of each string into a vector of numbers. For this we will use the well-known TF-IDF vectorizer, and create a pipeline that attaches it to a multinomial naive Bayes classifier:

In [ ]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import make_pipeline

model = make_pipeline(TfidfVectorizer(), MultinomialNB())
```

With this pipeline, we can apply the model to the training data, and predict labels for the test data:

In [ ]:

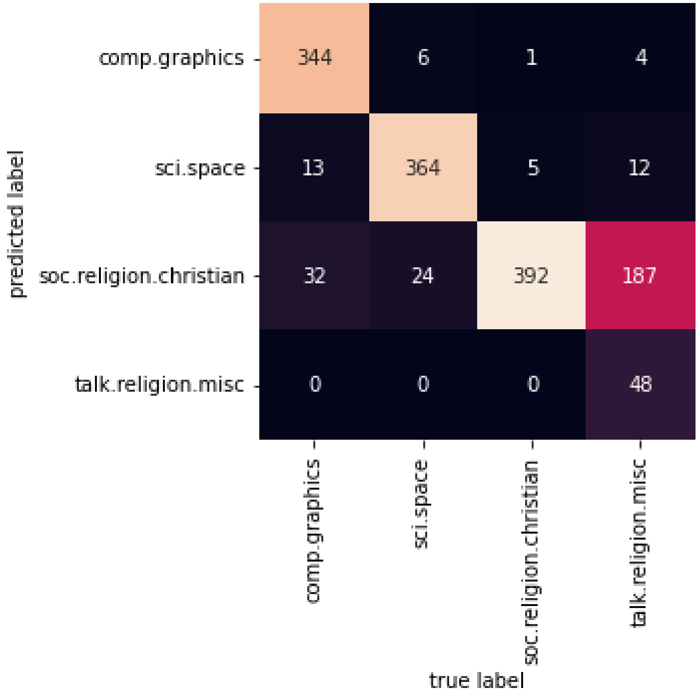
```
model.fit(train.data, train.target)
pred = model.predict(test.data)
```

Now that we have predicted the labels for the test data, we can evaluate them to learn about the performance of the estimator. For example, here is the confusion matrix between the true and predicted labels for the test data:

In [ ]:

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
mat = confusion_matrix(test.target, pred)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
```

```
            xticklabels=train.target_names, yticklabels=train.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



Evidently, even this very simple classifier can successfully separate space talk from computer talk, but it gets confused between talk about religion and talk about Christianity. This is perhaps an expected area of confusion!

The very cool thing here is that we now have the tools to determine the category for *any* string, using the `predict()` method of this pipeline. Here's a quick utility function that will return the prediction for a single string:

In [ ]:

```
def predict_category(s, train=train, model=model):
    pred = model.predict([s])
    return train.target_names[pred[0]]
```

In [ ]:

```
predict_category('sending a payload to the ISS')
```

Out[ ]:

```
'sci.space'
```

In [ ]:
```
predict_category('determining the screen resolution')
```

Out[ ]:
```
'comp.graphics'
```

Remember that this is nothing more sophisticated than a simple probability model for the (weighted) frequency of each word in the string; nevertheless, the result is striking. Even a very naive algorithm, when used carefully and trained on a large set of high-dimensional data, can be surprisingly effective.

Naive Bayes with continuous data from scratch

For continuous data we assume the conditional probabilities P(X|Y) to follow gaussian distributions.

In [ ]:
```
""" Define the dataset """
import numpy as np
from scipy import stats
import plotly.express as px
import plotly.graph_objects as go

Yes = [25.2, 19.3, 18.5, 21.7, 20.1, 24.3, 22.8, 23.1, 19.8]
No = [27.3, 30.1, 17.4, 29.5, 15.1]

df_yesno = (
    pd.DataFrame(
        data=[[val, "Yes"] for val in Yes] + [[val, "No"] for val in No],
        columns=["val", "label"]
    )
)

px.histogram(df_yesno, color="label", opacity=.7, barmode="overlay", nbins=5)
```

[TO DO STUDENTS]

- Implement the computation of first and second moments of the continuous feature for each class.
- Implement the computation of the posterior probability P(Y|X)

In [ ]:
```
""" Calculate the posterior probability P(Y|X) """
```

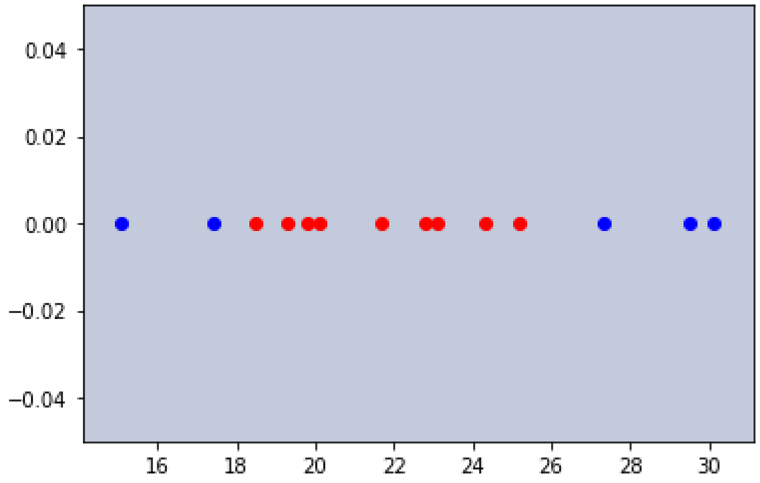
```python
def P(x, y=True):
    yes_mean = np.mean(Yes)
    yes_var = np.var(Yes)
    proba_yes = stats.norm.pdf(
        x=x,
        loc=yes_mean,
        scale=yes_var,
    )

    no_mean = np.mean(No)
    no_var = np.var(No)
    proba_no = stats.norm.pdf(
        x=x,
        loc=no_mean,
        scale=no_var,
    )
    if y == True:
        return "yes"
    elif y == False:
        return "no"
    return dict(zip(["yes", "no"], [proba_yes, proba_no]))


P(22)
```

Out[ ]: 'yes'

In [ ]:

```python
''' Plot the boundaries '''
import matplotlib.pyplot as plt
import math
%matplotlib inline

x_min, x_max = min(Yes+No)-1, max(Yes+No)+1

plt.plot(Yes, np.zeros_like(Yes),'ro')
plt.plot(No, np.zeros_like(No),'bo')

xx=np.linspace(x_min,x_max,100)
zz=[1 if P(x, True)>=P(x, False) else 0 for x in xx]

plt.contourf(xx, [-0.05, 0.05], [zz, zz], alpha=0.3)
```

Out[ ]:
```
<matplotlib.contour.QuadContourSet at 0x7f14fe535c90>
```

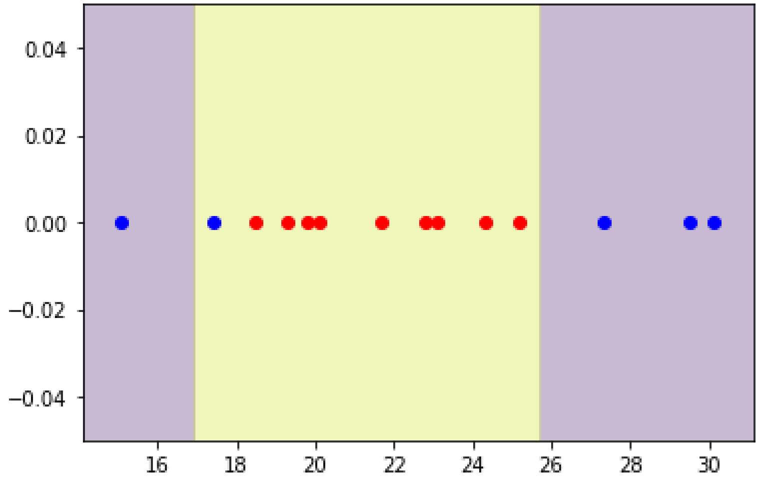


Naive Bayes with continous using sklearn library

Data are continous, we use Gaussian Naive Bayes

In [ ]:
```
from sklearn.naive_bayes import GaussianNB

X = np.array([Yes + No]).reshape(-1, 1)
y = [1]*len(Yes)+[0]*len(No)

"""
[TO DO STUDENTS]
Learn a Gaussian Naive Bayes classifier using sklearn implementation on this data (X,y)
"""
nb = GaussianNB().fit(X, y)
```

In [ ]:
```
xx=np.linspace(x_min,x_max,1000).reshape(-1, 1)
zz = nb.predict(xx).tolist()

plt.plot(Yes, np.zeros_like(Yes),'ro')
plt.plot(No, np.zeros_like(No),'bo')
plt.contourf(xx.ravel().tolist(), [-0.05, 0.05], [zz, zz], alpha=0.3);
```


0.04
0.02
0.00
−0.02
−0.04
16
18
20
22
24
26
28
30