Bayes

April 26, 2020

1 Bayes

1.1 Theory

The formula for Bayes rule in a sample is the following:

$$p(\{Y\}|\{X\}) = \frac{p(\{X\}|\{Y\})p(\{Y\})}{p(\{X\})}, \tag{1}$$

where $\{X\}$ denotes a sample of random variables and X denotes a single unit in the sample. Because the unit in a sample is assumed to be iid, given the realziation of the sample $\{x, y\}$, the conditional probability of obeserving the predictor-outcome pair in the current sample is

$$p(\{y\}|\{x\}) \propto \prod_i p(x^i|y^i)p(y^i), \tag{2}$$

where i is the label for units in the sample.

Assumptions are made about individual likelihood $p(X|Y)$ with some unkonwn parameter θ, e.g. gaussian distribution with standard deviation and mean as the parameter. Then some estimation method is used to fix the parameter.

1.2 MAP in Bayes

The method used in sklearn is MAP. It is to maximize the posteriori:

$$\begin{aligned}
\theta &= \arg\max_\theta \log p(\{y\}|\{x\}) && (3)\\
&= \arg\max_\theta \log \prod_i p(x^i|y^i)p(y^i) && (4)\\
&= \arg\max_\theta \sum_i [\log p(x^i|y^i) + \log p(y^i)]. && (5)
\end{aligned}$$

When the mariginal distribution $p(Y)$ does not contains the parameter in the likelihood function, $p(Y)$ can be estimated by the sample distribution.

[1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import sklearn.naive_bayes as NB
```

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

[2]:
```
# read data to pandas format
iris_data=pd.read_csv('Iris.csv')
# drop the id column to make the hist paragraph look better
iris_data=iris_data.drop(columns='Id')
```

[3]:
```
X_p, y_p = iris_data.iloc[:,:4],iris_data.iloc[:,[4]]
X=X_p.to_numpy()
y=y_p.to_numpy()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0)
y_train=y_train.reshape(y_train.shape[0],)
y_test=y_test.reshape(y_test.shape[0],)
```

1.3 Gaussian Bayes

The assumption of likelihood function reads:

$$p(X|Y) = \frac{1}{\sqrt{(2\pi)^d|\Sigma_Y|}}\exp(-\frac{1}{2}(X-\mu_Y)_i\Sigma_Y^{-1ij}(X-\mu_Y)_j). \tag{6}$$

There is no need to apply numerical method e.g. SGD to calculate the covariance matrix Σ and mean μ as can be proved by hand they simply given by the sample covariance matrix and sample mean for each class of Y.

Given X_{test}, if we only want to achieve the prediction and not care about the probability distributio on each class, we can iterate among all classes to find the largest value of

$$p(X_{test}|Y)p(Y) = \frac{1}{\sqrt{|\Sigma_Y|}}\exp(-\frac{1}{2}(X_{test}-\mu_Y)_i\Sigma_Y^{-1ij}(X_{test}-\mu_Y)_j)p(Y). \tag{7}$$

[4]:
```
## this is the GaussianBayes code written by myself. I didn't find a␣
 ↪GaussianBayes package
class GaussianBayes():
    def __init__(self):
        pass
    def fit(self,X,Y):
        data=pd.DataFrame(data=X)
        data['class']=Y
        self.sigmas=[]
        self.mus=[]
        self.classes=data.iloc[:,-1].unique()
        self.p_y=data.iloc[:,-1].value_counts()

        for y in self.classes:
```

```
            # calculate covariance matrix for each class
            self.sigmas.append(data.loc[data.iloc[:,-1]==y].corr())
            # calculate mean vector for each class
            self.mus.append(data.loc[data.iloc[:,-1]==y].mean())
    def predict(self,X):
        # sigmas.shape=(#_class,#_predictor,#_predictor)
        # mus.shape=(#_class,#_predictor)
        Y=[]
        p_y=np.array([self.p_y[y] for y in self.classes])
        sigmas=np.array([i.to_numpy() for i in self.sigmas])
        mus=np.array([i.to_numpy() for i in self.mus])
        for x in X:
            # power.shape=(#class,1,1)
            power=-np.matmul(np.subtract(mus,x)[:,np.newaxis,:],np.matmul(np.
↪linalg.inv(sigmas),np.subtract(mus,x)[:,:,np.newaxis]))
            power=power.reshape(power.shape[0])
            p_xy=np.multiply(np.multiply(np.reciprocal(np.sqrt(np.linalg.
↪det(sigmas))),np.exp(power)),p_y)
            Y.append(self.classes[np.argmax(p_xy)])
        return Y
```

[5]:
```
GB=GaussianBayes()
GB.fit(X_train, y_train)
y_pred = GB.predict(X_test)
print("Number of mislabeled points out of a total %d points : %d"
     % (X_test.shape[0], (y_test != y_pred).sum()))
```

```
Number of mislabeled points out of a total 45 points : 3
```

1.3.1 Naive Bayes

Naive Bayes is based on the conditional independence assumption between features such that

$$p(x_j^i|y^i) = \prod_j p(x_j^i|y^i), \tag{8}$$

where j is the label of feature. So MAP in the case of naive Bayes can be further written as:

$$\theta = \arg\max_\theta \sum_i [\sum_j \log p(x_j^i|y^i) + \log p(y^i)]. \tag{9}$$

Gaussian Naive Bayes

$$p(x_j|y) = \frac{1}{\sqrt{2\pi\sigma_y}} \exp(-\frac{(x_j-\mu_y)^2}{2\sigma_y^2}). \tag{10}$$

[6]:
```
gnb = NB.GaussianNB()
y_pred = gnb.fit(X_train, y_train).predict(X_test)
print("Number of mislabeled points out of a total %d points : %d"
      % (X_test.shape[0], (y_test != y_pred).sum()))
```

```
Number of mislabeled points out of a total 45 points : 0
```

Multinomial Naive Bayes

[7]:
```
mnb = NB.MultinomialNB()
y_pred = mnb.fit(X_train, y_train).predict(X_test)
print("Number of mislabeled points out of a total %d points : %d"
      % (X_test.shape[0], (y_test != y_pred).sum()))
```

```
Number of mislabeled points out of a total 45 points : 18
```

Categorical Naive Bayes

[8]:
```
mnb = NB.CategoricalNB()
y_pred = mnb.fit(X_train, y_train).predict(X_test)
print("Number of mislabeled points out of a total %d points : %d"
      % (X_test.shape[0], (y_test != y_pred).sum()))
```

```
Number of mislabeled points out of a total 45 points : 4
```