Naive-Bayes

Inspired by machinelearningmastery blog on Naive Bayes classifier

Introduction

Classification is a predictive modelling problem that assigns class label to a given input data sample.

Bayes theorem provides a principled way for classification through Bayes optimal classifier but in practice, it requires an enormous amount of data to calculate the conditional probability of class label given the dataset and is computationally expensive.
However, certain simplifications to this approach like assuming the variables or features in input data to be conditionally independent, makes the calculation of conditional probability tractable(feasible) and results in an effective way of classification called Naive Bayes.

Thus, to simplify the calculations involved in Bayes theorem, we tend to assume all the variables to be independent of each other.

In a conditional predictive modelling, we calculate

$P(yi \mid x1, x2, \ldots, xn) = P(x1, x2, \ldots, xn \mid yi) * P(yi) / P(x1, x2, \ldots, xn)$
Where,
yi : ‘i’th class or label
x1,x2,.....,xn : features or variables in an input example to be classified

As P(x1,x2,...,xn) is a constant and has an effect of normalisation, we can remove it from our calculation.

$P(yi \mid x1,x2,....,xn) = P(x1,x2,...,xn \mid yi) * P(yi)$
Where,
P(x1,x2,...,xn | yi) : Conditional probability of independent variables given a class yi

Since the variables are independent,

$P(yi \mid x1, x2, \ldots, xn) = P(x1|yi) * P(x2|yi) * \ldots P(xn|yi) * P(yi)$ ...**Eq1**

This calculation is performed for each class label and the label with maximum conditional probability can be selected as the classification for the given instance.
This decision rule is referred to as *MAP(Maximum a posteriori)* rule.
This simplification of Bayes theorem is widely used for classification predictive modelling problem and is referred to as *Naive Bayes*.

Calculation of terms involved in Eq1

Lets start with generating a small binary classification dataset using make_blobs function in scikit learn API.

```
from sklearn.datasets.samples_generator import make_blobs
#Generating a small binary classification dataset
X,y=make_blobs(n_samples=100,n_features=2,centers=2,random_state=1)
```

This will create a 100-examples dataset with two classes ‘0’ and ‘1’. Each example would consist of two features or variables and random_state=1 is used so that the values reproduce itself exactly the same on every function call.

- P(yi) can be calculated by dividing the frequency of data samples of the ‘i’ th class by the total number of examples as given below:

```
#To differentiate samples of both classes and hence, calculate prior probabilities, that is the probability of two classes
Xy0=X[y==0]
Xy1=X[y==1]

#Prior probability
Py0=len(Xy0)/len(X)
Py1=len(Xy1)/len(X)
```

The above code separates the samples of two classes(‘0’ and ‘1’) and calculates prior probability of the two classes.

- To calculate the conditional probability of a feature or variable given a class label, that is
  P(x1|yi), P(x2|yi), …, P(xn|yi)
  we need K*n probability distributions where, ‘K’ is the number of classes and ‘n’ is the number of features or variables in the input data.
  In our case, we have two classes and two features in each sample, that is we need four probability distributions.

  The data for a specific feature of a given class is used to determine the parameters of one of the three standard probability distributions.

  - If the feature or variable is binary(True or False), binomial probability distribution is used.
  - If the feature or variable is a label or counts(categorical), multinomial probability distribution is used.
  - If the feature or variable is numerical, often Gaussian probability distribution is used.

  In our case, the features of our data are numerical, hence to find individual conditional probability of a feature given a class, we fit the data pertaining to a feature and a class in a Gaussian probability distribution.

```
from scipy.stats import norm
from numpy import mean,std
```

norm function provides the normal distribution for a given data.

```
def fit_dist(data):
    return norm(mean(data),std(data))
```

mean and standard deviation(*std*) are the parameters of the normal distribution for the given data.

```
#feature #1 for class 0
X1y0=Xy0[:,0]
#feature #2 for class 0
X2y0=Xy0[:,1]
#feature #1 for class 1
X1y1=Xy1[:,0]
#feature #2 for class 1
X2y1=Xy1[:,1]

#fitting probability distribution(gaussian)
#For class 0
PX1y0=fit_dist(X1y0)
PX2y0=fit_dist(X2y0)
#For class 1
PX1y1=fit_dist(X1y1)
PX2y1=fit_dist(X2y1)
```

The above calculates the four probability distribution of every pair of feature given a class.

Now, calculate the probability of a class given a data sample from these probability distributions, in order to classify the data sample.

```
def prob(X,prior,dist1,dist2):
    return prior*dist1.pdf(X[0])*dist2.pdf(X[1])
```

The above python code is equivalent to the below mathematical equation,
For class ‘0’,
P(class ‘0’ | x1,x2)=P(x1 | class ‘0’)*P(x2 | class ‘0’)*P(class ‘0’)
For class ‘1’,
P(class ‘1’ | x1,x2)=P(x1 | class ‘0’)*P(x2 | class ‘1’)*P(class ‘1’)

```
X_sample,y_sample=X[1],y[1]

#Probability of class 0 given the dataset
Py0=prob(X_sample,Py0,PX1y0,PX2y0)
#Probability of class 1 given the dataset
Py1=prob(X_sample,Py1,PX1y1,PX2y1)

print('X_sample:',X_sample)
print('P(y=0/',X_sample,')=%.3f'%Py0)
print('P(y=1/',X_sample,')=%.3f'%Py1)
print('predicted class:',0 if Py0>Py1 else 1)
print('y_sample(The ground Truth):',y_sample)
```

The above code classifies the sample into two classes ‘0’ or ‘1’ using the Naive-Bayes algorithm.

The result looks like,

```
X_sample: [-9.15155186 -4.81286449]
P(y=0/ [-9.15155186 -4.81286449] )=0.000
P(y=1/ [-9.15155186 -4.81286449] )=0.053
predicted class: 1
y_sample(The ground Truth): 1
```

In practice, it is a good idea to use optimized implementations of the Naive Bayes algorithm. The scikit-learn library provides three implementations, one for each of the three main probability distributions; for example, BernoulliNB, MultinomialNB, and GaussianNB for binomial, multinomial and Gaussian distributed input variables respectively.

To know how it is implemented and some tips on Naive Bayes which I describe briefly below, I would suggest you read the original blog by Jason Brownlee from which this reading is derived.

5 tips when using Naive-Bayes

- Use a KDE(Kernel Density Estimator) for a complex distribution
  A KDE helps estimating the probability distribution for a data with unknown or complex distribution using the data samples itself.

- Decreased performance with increasing variable dependency
  Naive-Bayes is based on the assumption of independence of variables.This assumption works well even when some or most of the variables are dependent.Nevertheless, the performance of the algorithm degrades.

- Avoid Numerical Underflow with Log
  Multiplication of probabilities lead to a numerically unstable values.Thus, to overcome that problem, we take the sum of natural logarithm of the independent conditional probabilities.

- Update Probability Distribution
  It is a good idea to update the probability distribution as the new data comes in.

- Use as a Generative model
  In the words of Jason Brownlee,
  The prepared probability distributions can be randomly sampled in order to create new plausible data instances. The conditional independence assumption assumed may mean that the examples are more or less plausible based on how much actual interdependence exists between the input variables in the dataset.

Conclusion

Thus,we learnt about the Naive-Bayes algorithm and its implementation from scratch.

I tried to deliver the things in a simple and lucid manner.

Also, I suggest you to read blogs by Jason Brownlee who is amazing at delivering knowledge on Machine Learning algorithms.

If you want my help in any of your project, I will be more than happy to help you.
Mail me:nigamharshit1712@gmail.com