Naive Bayes

Introdução

Vamos pensar em um caso muito comum de mineração de dados: spams. Um spam nada mais é que alguma coisa indesejada pelo usuário. Seu nome surgiu de um esquete do programa Monty Python.

A aplicação do modelo Naive Bayes nesse tipo de análise é muito comum. Outros exemplos de possíveis aplicações são:

- Classificação de textos (tweets, artigos em uma revista, etc.);
- Análise de sentimentos;
- Sistemas de recomendação.

O modelo Naive Bayes é, portanto, um modelo supervisionado de classificação, assim como o modelo dos k-vizinhos. Basicamente, o modelo aprende a probabilidade de um objeto com certos atributos pertencer a um determinado grupo.

Dentre as vantagens desse modelo, podemos citar:

- É um algoritmo relativamente simples de entender e construir;
- Tende a ser mais rápido prever classes usando esse algoritmo do que outros algoritmos de classificação;
- Ele pode ser facilmente treinado com um conjunto de dados pequeno.

Porém, ele também possui algumas desvantagens:

- Se uma determinada classe ou atributo possuem frequência zero, então a probabilidade associada a essa classe ou atributo também vai ser zero (suponha que nenhum dos emails usados no treinamento tenha a palavra "hoje"). Esse é conhecido como "Problema de Probabilidade Condicional Zero". Ele pode ser contornado com algumas técnicas, como a correção de Laplace (que vamos ver mais à frente);
- O modelo parte de uma premissa de que a ocorrência de um determinado evento é independente da ocorrência de outros eventos. Sabemos que isso nem sempre é o caso, mas aceitamos essa premissa ainda assim para a utilização do modelo.

Essa segunda desvantagem, por sinal, é o que dá o nome "naive" ao nosso modelo.

Já a parte "bayes" do nome do modelo vem do teorema de Bayes, que é a base do algoritmo do modelo. Esse teorema provê uma forma de calcular a probabilidade condicional, ou seja, a probabilidade de um evento baseado em conhecimentos prévios dos eventos.
Matematicamente, o teorema se apresenta da seguinte forma:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} = \frac{P(B|A) \times P(A)}{P(B|A) \times P(A) + P(B|\sim A) \times P(\sim A)}$$

Exemplo de aplicação do teorema

Vamos pegar um exemplo. Suponha que tenhamos que escolher uma carta de um baralho comum de 52 cartas. A probabilidade da carta ser uma dama é:

$$P(dama) = \frac{4}{52}$$

Se você possui evidências de que a carta que você tirou é uma carta de face (valete, dama ou rei), a probabilidade P(dama|face) pode ser calculada pelo teorema de Bayes:

$$P(dama|face) = \frac{P(face|dama) \times P(dama)}{P(face)}$$

Como P(face|dama) = 1 e P(face) = 12 / 52, podemos calcular:

$$P(dama|face) = \frac{1 \times \left(\frac{4}{52}\right)}{\frac{12}{52}} = \frac{1}{3}$$

Sobre o algoritmo

Vamos agora expandir o teorema de Bayes para o nosso algoritmo. Nós vamos ter múltiplos atributos (ou classes), vamos chamar C_1, C_2, ..., C_k. O objetivo principal do algoritmo é calcular a probabilidade de um objeto que tenha atributos x_1, x_2, ..., x_n pertencer a uma determinada classe C_i.

$$P(C_i|x_1, x_2, \dots, x_n) = \frac{P(x_1, x_2, \dots, x_n|C_i)}{P(x_1, x_2, \dots, x_n)} \times P(C_i), para\ 1 \leq i \leq k$$

Assumindo aqui que os eventos (atributos) são independentes (o "naive" do nosso algoritmo), podemos então chegar à seguinte conclusão:

$$P(C_i|x_1, x_2, \dots, x_n) = \frac{\prod_{j=1 \to n} P(x_j|C_i)}{P(x_1, x_2, \dots, x_n)} \times P(C_i), para\ 1 \leq i \leq k$$

Exemplo prático

Vamos fazer um exemplo prático na mão para entender melhor o algoritmo. Suponha que tenhamos a tabela abaixo como nosso conjunto de dados de treino:

Nome	Amarela?	Doce?	Comprida?	Total
manga	350	450	0	650
banana	400	300	350	400
outras	50	100	50	150
total	800	850	450	1200

Vamos supor que desejamos classificar uma fruta que possui como características (atributos) o fato de ser amarela, doce e comprida.

O primeiro passo é montar uma tabela de frequências para cada um dos atributos:

Nome	Amarela?	Doce?	Comprida?	Total
manga	350/650 = P(amarela|manga)	450/650	0/650	650/1200 = P(manga)
banana	400/400	300/400	350/400	400/1200
outras	50/150	100/150	50/150	150/1200
total	800/1200 = P(amarela)	850/1200	450/1200	1200/1200

Vamos calcular, então, as probabilidades condicionais:

$$P(manga|amarela, doce, comprida)$$
$$= \frac{P(amarela|manga) \times P(doce\backslash manga) \times P(comprida|manga) \times P(manga)}{P(amarela, doce, comprida)} = 0$$

$$P(banana|amarela, doce, comprida)$$
$$= \frac{P(amarela|banana) \times P(doce|banana) \times P(comprida|banana) \times P(banana)}{P(amarela, doce, comprida)}$$
$$= \frac{\frac{400}{400} \times \frac{300}{400} \times \frac{350}{400} \times \frac{400}{1200}}{P(evidência)} = \frac{0.21875}{P(evidência)}$$

$$P(outras|amarela, doce, comprida)$$
$$= \frac{P(amarela|outras) \times P(doce|outras) \times P(comprida|outras) \times P(outras)}{P(amarela, doce, comprida)}$$
$$= \frac{\frac{50}{150} \times \frac{100}{150} \times \frac{50}{150} \times \frac{150}{1200}}{P(evidência)} = \frac{0.00926}{P(evidência)}$$

Vemos, então, que dadas as três características (amarela, doce e comprida), baseado no nosso conjunto de dados, a fruta tem mais probabilidade de ser uma banana.

Observem que, nesse caso, temos um atributo com frequência igual a zero. Nesse caso, poderíamos fazer uma suavização simples no cálculo da frequência, como sendo:

$$P(A|B) = \frac{k + eventos\ de\ B\ que\ contêm\ A}{2k + eventos\ de\ B}$$

Esse valor de k pode arbitrado pelo cientista de dados, baseado no peso que ele deseja incluir a esses atributos com frequência zero. Adotando k = 1, teríamos, por exemplo:

$$P(comprida|manga) = \frac{k + 0}{2k + 650} = \frac{1}{2 + 650} = 0.0015$$

Para aplicar essa suavização (suavização de Laplace), precisaríamos aplicar esse cálculo a todos os atributos da nossa tabela, para em seguida recalcular as três probabilidades anteriores.

Existem algumas variações do algoritmo Naive Bayes dependendo da distribuição da frequência $P(x_j|C_i)$. As três mais comuns são:

- Gaussiana: os atributos possuem uma distribuição gaussiana ou normal;
- Multinomial: é usado quando múltiplas ocorrências de um mesmo atributo é relevante para a análise;
- Bernoulli: é usado quando os atributos no conjunto de dados são binários, por exemplo, se uma determinada palavra aparece ou não em um email.

Implementação do modelo

Para a implementação programática do modelo Naive Bayes, vamos continuar usando a base de dados das flores íris, que já vimos na última aula. Da mesma forma que na última aula, vamos eliminar também o passo de transformação dos dados, que não será necessário para essa base de dados em particular.

Passo 1: import e análise preliminar dos dados

```
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
data = iris.data
target = iris.target
```

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Passo 2: divisão dos dados e treino do modelo

```
# Dividir os dados nos conjuntos de treino e de teste
data_train, data_test, target_train, target_test = train_test_split(
  data,
  target,
  test_size=0.2,
  random_state=0
)

gnb = GaussianNB()
mnb = MultinomialNB()

target_pred_gnb = gnb.fit(data_train, target_train).predict(data_test)
```

Passo 3: análise dos resultados

Vamos agora determinar uma matriz de confusão do problema. Por definição, uma matriz de confusão C é tal que C(i, j) é igual ao número de observações conhecidas por pertencerem ao grupo i e foram previstas no grupo j.

```
cnf_matrix_gnb = confusion_matrix(target_test, target_pred_gnb)

print(cnf_matrix_gnb)
```

```
[[11  0  0]
 [ 0 13  0]
 [ 0  1  5]]
```

Isso significa, portanto, que no nosso conjunto de teste, tivemos uma previsão errada: tivemos uma observação que é conhecida pertencer à espécie virginica, porém a prevemos como sendo versicolor.

Vamos aproveitar e ver mais algumas métricas do nosso modelo.

```
class_report = classification_report(target_test, target_pred_gnb)

print(class_report)
```

```
              precision    recall  f1-score   support
```

```
              0       1.00      1.00      1.00        11
              1       0.93      1.00      0.96        13
              2       1.00      0.83      0.91         6

       accuracy                           0.97        30
      macro avg       0.98      0.94      0.96        30
   weighted avg       0.97      0.97      0.97        30
```

Vamos relembrar as definições:

- Precisão: é a razão `tp / (tp + fp)`, onde `tp` é o número de positivos verdadeiros e `fp` é o número de positivos falsos. Essa métrica indica a habilidade do modelo em não nomear uma amostra negativa como sendo positiva.
- Sensibilidade: é a razão `tp / (tp + fn)`, com `fn` sendo o número de negativos falsos, e mede a fração dos positivos que nosso modelo identificou.
- F1 score: é um indicador que relaciona a precisão e a sensibilidade, sendo calculada como `2 * p * r / (p + r)`, com `p` e `r` sendo a precisão e a sensibilidade, respectivamente.
- Acurácia: compara quantos rótulos de uma amostra foram previstos exatamente como os dados de teste.
- Média macro: calcula a média aritmética simples das métricas.
- Média peso: calcula a média aritmética ponderada das métricas, baseada nos suportes.

Fazendo a mesma análise de novas predições, vamos incluir um novo conjunto de dados e identificar quais classes são previstas.

```
labels = {0: "setosa", 1: "versicolor", 2: "virginica"}

# Vamos montar a previsão em cima de dados novos
# Prevê os rótulos das duas observações aleatórias abaixo
x_new = [
    [3, 4, 5, 2],
    [5, 4, 2, 2]
]
y_predict = gnb.predict(x_new)

print(labels[y_predict[0]])
print(labels[y_predict[1]])
```

```
virginica
versicolor
```

Se compararmos com os resultados da aula passada, vemos um possível problema: as previsões segundo esse modelo são diferentes com relação às previsões do modelo dos k-vizinhos próximos. Algumas possíveis razões para isso ter ocorrido estão apresentadas abaixo:

- Devido à aleatoriedade da função `train_test_split`, os dados de treino usados nessa análise foram diferentes dos dados usados na análise dos k-vizinhos. Isso pode afetar a forma como esses modelos chegam às suas respectivas conclusões;
- Temos apenas 120 entradas de dados pra o treino dos algoritmos. Para termos um melhor desempenho dos nossos modelos, seria importante termos uma base de dados muito superior a essa. O fato do modelo ter desempenhado bem com esse volume de dados não necessariamente implica que ele teria um bom resultado para qualquer dado;
- Comparar os resultados de apenas duas entradas de teste e assumir que um dos modelos está incorreto é uma falácia. Existe uma chance de entrarmos com outros 100 dados e todos eles apresentarem a mesma previsão em ambos os modelos. Devemos evitar tirar esse tipo de conclusão com uma base de dados tão pequena.

Caso queira se aprofundar, elabore a matriz de confusão e imprima as métricas do modelo dos k-vizinhos que vimos na última aula. Faça também uma comparação entre os resultados dos dois modelos. Veja que, em termos de métricas, os dois modelos estão muito próximos.

Esse resultado é interessante pois ressalta a importância da comparação de modelos por parte do cientista de dados. Assumir que um modelo será sempre o correto a se usar é um erro que não deve ser cometido. Um bom projeto de análise de dados deve levantar vários possíveis modelos e compará-los, de forma a adotar aquele que apresenta um melhor desempenho.

Caso tivéssemos mais dados disponíveis para essa base, uma possível forma de tentar esclarecer qual dos dois modelos tem um melhor desempenho para esses dados seria, ao invés de dividir a base em duas partes (treino e teste), dividir em três. Com as duas primeiras partes implementamos o aprendizado de cada modelo e o validamos; e com a terceira parte comparamos os desempenhos entre os modelos. Assim utilizamos uma mesma base comparativa para analisar os resultados de todos os modelos.

✓ 0s conclusão: 19:56