Systemy Sztucznej Inteligencji

dokumentacja projektu

Obliczanie prawdopodobieństwa wystąpienia udaru na podstawie danych pacjenta

Wiktor Machoń, Piotr Kołodziejski

29 maja 2022

Część I

Opis programu

Celem projektu jest zbudowanie klasyfikatora w oparciu o Naiwny Klasyfikator Bayesa. Klasyfikator ten ma za zadanie ocenić czy dany pacjent, ze swoimi parametrami jest w grupie zagrożonej dostaniem udaru.

Instrukcja obsługi

Program został napisany w języku programowania Python przy użyciu Jupyter notebook. Program (plik z rozszerzeniem .ipynb) należy uruchomić z poziomu aplikacji Jupyter Notebook. Plik .csv z danymi powinien znajdować się w tym samym folderze, co plik rozwiązania.

Dodatkowe informacje

Aby klasyfikator uznać za działający, jego trafność predykcji musi być na poziomie co najmniej 85 procent.

Część II

Opis działania

Naiwny klasyfikator Bayesa tworzymy w oparciu o Twierdzenie Bayesa, które opisujemy poniższym równaniem:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \tag{1}$$

W naszym programie używamy zmiennych X (dla danych wejściowych) oraz y (dla danych wyjściowych), więc nasz wzór przyjmuje postać:

$$P(y|X) = \frac{P(X|y) * P(y)}{P(X)} \tag{2}$$

Przyjmujemy naiwne założenie, że zmienne są od siebie niezależne, co implikuje, że:

$$P(X|y) = P(x_1|y) * P(x_2|y) * ... * P(x_n|y) \tag{3}$$

Prawdopodobieństwo obliczamy dla y, więc P(X) jest stałą, co oznacza, że możemy usunąć ją z równania otrzymując przybliżenie:

$$P(X|y) = P(y) * \prod_{i=1}^{n} P(x_i|y) \tag{4}$$

Korzystając z tego wzoru musimy wyznaczyć maksymalną wartość y.

Algorytm

Implementacja Naiwnego Klasyfikatora Bayesa w oparciu o Twierdzenie Bayesa, opisane wyżej. Dane z pliku .csv wczytujemy korzystając w biblioteki pandas, tworząc tym samym dataframe. Zamieniamy wartości tekstowe na liczbowe oraz dzielimy zbiór na treningowy i testowy, tak że ten pierwszy to 70 procent zbioru. Przed podziałem policzone zostały średnie i odchylenia stadardowe dla każdej klasy.

Data: Dane wejściowe dataframe X
Result: Klasy kolejnych wierszy - lista Predictions
Wczytaj dataframe. $Predictions = [...]$;
foreach $i \in X.index$ **do**
  Zainicjuj listę $ClassLikelihood$;
  Przypisz obecny wiersz jako obiekt do zmiennej $instance = X.loc[i]$;
  foreach $cls \in classes$ **do**
    Utwórz listę $FeutureLikelihoods$;
    Dodaj do listy $FeutureLikelihoods$ $log(prior[cls])$;
    foreach $col \in xTrain.columns$ **do**
      $data = instance[col]$;
      $mean = means[col].loc[cls]$;
      $variance = var[col].loc[cls]$;
      Obliczyć gęstość prawdopodobieństwa
       $Likelihood = Normal(data, mean, variance)$
       if $Likelihood! = 0$ **then**
        $Likelihood = np.log(Likelihood)$;
      else
        $Likelihood = 1/len(xTrain)$;
      end
    end
    $TotalLikelihood = sum(FeatureLikelihoods)$;
    $ClassLikelihood.append(TotalLikelihood)$;
  end
  $MaxIndex = ClassLikelihood.index(max(ClassLikelihood))$;
  $Prediction = classes[MaxIndex]$;
  $Predictions.append(Prediction)$;
end

Algorithm 1: Algorytm drukowania informacji o liczbie parzystej/nieprarzystej.

Algorytm odrzucania najmniej istotnych kolumn

Data: Dane wejściowe dataframe y, X
Result: Nazwy dwóch kolumn, których odrzucenie będzie skutkowało zwiększeniem dokładności
Utwórz pusty słownik $colsAndAcc$;
foreach $col1 \in X.columns$ **do**
    $x1 = X.drop(col1, axis = 1)$;
    foreach $col2 \in x1.columns$ **do**
        $x2 = x1.drop(col2, axis = 1)$;
        $prediction = Predict(x2)$;
        $colsAndAcc[str(col1 +' and' + col2)] = round(Accuracy(y, prediction), 5)$;
    end
end
Zwróć maksymalną wartość ze słownika $colsAndAc$ oraz jej klucz

Algorithm 2: Algorytm odrzucania najmniej istotnych kolumn

Implementacja

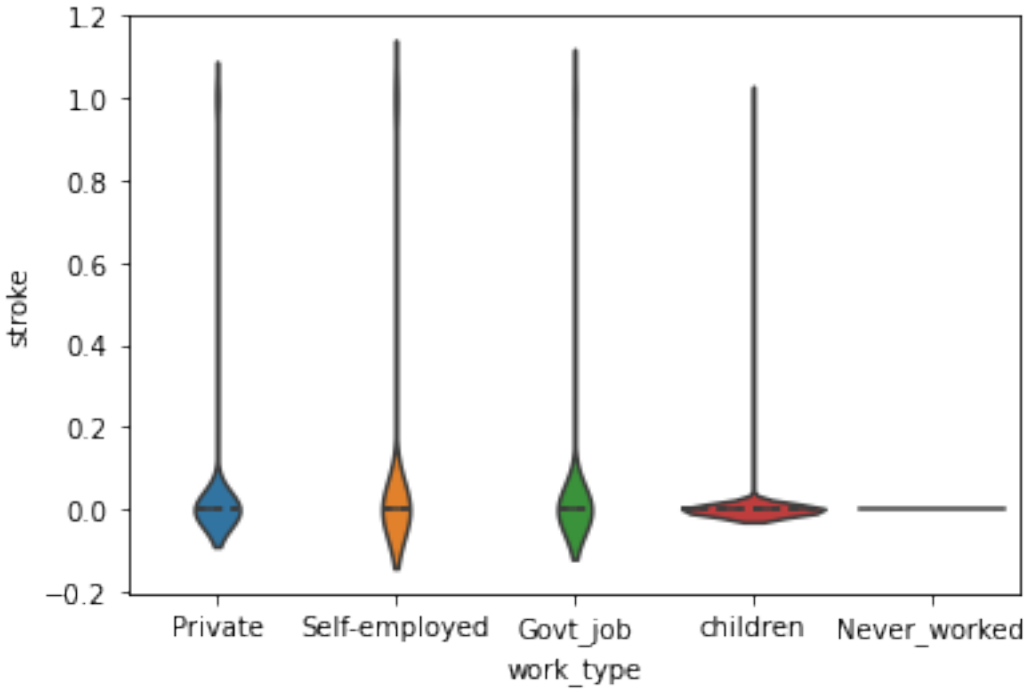
W pliku z danymi występowały dane tekstowe, więc żeby móc je wykorzysztać zostały one przekszałcone w dane numeryczne. Gdy dla danej klasy występowały tylko dwie wartości to zostały one przekształcone na wartości 0 lub 1. Gdy wartości było więcej to dodawano kolejne liczby naturalne. Jeżeli nie ma konkretnej informacji (jak np. *Other* w klasie *gender*) wartości takiej zostało przypisane -1.

```
def ToNumData(X):
  for col in X.columns:
      if col == "gender":
          for row_nr in range(len(X[col])):
              if(X[col].iloc[row_nr] == "Male"):
                  X[col].iloc[row_nr] = 0
              elif (X[col].iloc[row_nr] == "Female"):
                  X[col].iloc[row_nr] = 1
              elif (X[col].iloc[row_nr] == "Other"):
                  X[col].iloc[row_nr] = -1
      elif col == "ever_married":
          for row_nr in range(len(X[col])):
              if(X[col].iloc[row_nr] == "Yes"):
                  X[col].iloc[row_nr] = 1
              elif (X[col].iloc[row_nr] == "No"):
                  X[col].iloc[row_nr] = 0
      elif col == "Residence_type":
          for row_nr in range(len(X[col])):
              if(X[col].iloc[row_nr] == "Rural"):
                  X[col].iloc[row_nr] = 1
              elif (X[col].iloc[row_nr] == "Urban"):
                  X[col].iloc[row_nr] = 0
      elif col == "smoking_status":
          for row_nr in range(len(X[col])):
              if(X[col].iloc[row_nr] == "formerly smoked"):
                  X[col].iloc[row_nr] = 1
              elif (X[col].iloc[row_nr] == "never smoked"):
                  X[col].iloc[row_nr] = 0
              elif (X[col].iloc[row_nr] == "smokes"):
                  X[col].iloc[row_nr] = 2
              elif (X[col].iloc[row_nr] == "Unknown"):
                  X[col].iloc[row_nr] = -1
      elif col == "work_type":
          for row_nr in range(len(X[col])):
              if(X[col].iloc[row_nr] == "Never_worked"):
                  X[col].iloc[row_nr] = 0
              elif (X[col].iloc[row_nr] == "Govt_job"):
                  X[col].iloc[row_nr] = 1
              elif (X[col].iloc[row_nr] == "Private"):
                  X[col].iloc[row_nr] = 2
              elif (X[col].iloc[row_nr] == "children"):
                  X[col].iloc[row_nr] = -1
              elif (X[col].iloc[row_nr] == "Self-employed"):
                  X[col].iloc[row_nr] = 3

  return X
```

Testy

Sprawdzenie jak różne formy zatrudnienia mają się do możliwości wystąpienia udaru.



Rysunek 1: Forma zatrudnienia a możliwość wystąpienia udaru

Eksperymenty

W ramach sprawdzenia i głębszej analizy danych sprawdzono jakie kolumny mają wpływ na wynik klasyfikacji. Gdy kolumna ta nie miała wpływu została odrzucania. W ten sposób efektywność wzrosła z 86 do 92 procent.

```
1
2 def colSelection(y,X):
3     i = 1
4     cols_and_acc = dict()
5     for col1 in X.columns:
6         x1 = X.drop(col1, axis = 1)
7         for col2 in x1.columns:
8             x2 = x1.drop(col2, axis = 1)
9             prediction = Predict(x2)
10            cols_and_acc[str(col1 + " and " + col2)] = round(Accuracy(y,
                  prediction), 5)
11            i+=1
12    print("Accuracy without columns "+max(cols_and_acc, key=cols_and_acc
          .get)+": "+str(cols_and_acc[max(cols_and_acc, key=cols_and_acc.
          get)]))
```

```
colSelection(yVal,xVal)
```

```
Accuracy without columns hypertension and avg_glucose_level: 0.92532
```

Rysunek 2: Po odrzuceniu powyższych kolumn klasyfikator ma wyższą skuteczność

Pełen kod aplikacji

```
import numpy as np
import random as rd
import pandas as pd
import seaborn as sns
import matplotlib as plt

strokes = pd.read_csv('healthcare-dataset-stroke-data.csv')
strokes.info()
strokes.describe()
strokes.head()

sns.violinplot(y='stroke', x='work_type', data=strokes, inner='quartile')

sns.pairplot(strokes, hue='stroke', markers='+')

def ToNumData(X):
    for col in X.columns:
        if col == "gender":
            for row_nr in range(len(X[col])):
                if(X[col].iloc[row_nr] == "Male"):
                    X[col].iloc[row_nr] = 0
                elif (X[col].iloc[row_nr] == "Female"):
                    X[col].iloc[row_nr] = 1
                elif (X[col].iloc[row_nr] == "Other"):
                    X[col].iloc[row_nr] = -1

        elif col == "ever_married":
            for row_nr in range(len(X[col])):
                if(X[col].iloc[row_nr] == "Yes"):
                    X[col].iloc[row_nr] = 1
                elif (X[col].iloc[row_nr] == "No"):
                    X[col].iloc[row_nr] = 0

        elif col == "Residence_type":
            for row_nr in range(len(X[col])):
                if(X[col].iloc[row_nr] == "Rural"):
                    X[col].iloc[row_nr] = 1
                elif (X[col].iloc[row_nr] == "Urban"):
                    X[col].iloc[row_nr] = 0

        elif col == "smoking_status":
```

```
            for row_nr in range(len(X[col])):
                if(X[col].iloc[row_nr] == "formerly smoked"):
                    X[col].iloc[row_nr] = 1
                elif (X[col].iloc[row_nr] == "never smoked"):
                    X[col].iloc[row_nr] = 0
                elif (X[col].iloc[row_nr] == "smokes"):
                    X[col].iloc[row_nr] = 2
                elif (X[col].iloc[row_nr] == "Unknown"):
                    X[col].iloc[row_nr] = -1

        elif col == "work_type":
            for row_nr in range(len(X[col])):
                if(X[col].iloc[row_nr] == "Never_worked"):
                    X[col].iloc[row_nr] = 0
                elif (X[col].iloc[row_nr] == "Govt_job"):
                    X[col].iloc[row_nr] = 1
                elif (X[col].iloc[row_nr] == "Private"):
                    X[col].iloc[row_nr] = 2
                elif (X[col].iloc[row_nr] == "children"):
                    X[col].iloc[row_nr] = -1
                elif (X[col].iloc[row_nr] == "Self-employed"):
                    X[col].iloc[row_nr] = 3

    return X

#Dropping rows where bmi is NaN
strokes.dropna(subset= ["bmi"], inplace=True)

#Changing non-numeric data to numeric types
strokes = ToNumData(strokes)
strokes["gender"] = pd.to_numeric(strokes["gender"])
strokes["ever_married"] = pd.to_numeric(strokes["ever_married"])
strokes["Residence_type"] = pd.to_numeric(strokes["Residence_type"])
strokes["smoking_status"] = pd.to_numeric(strokes["smoking_status"])
strokes["work_type"] = pd.to_numeric(strokes["work_type"])

strokesTrain = strokes.sample(frac=0.7, random_state=1)
strokesVal = strokes.drop(strokesTrain.index)

yTrain = strokesTrain["stroke"]
xTrain = strokesTrain.drop("stroke", axis = 1).drop("id", axis = 1)

yVal = strokesVal["stroke"]
xVal = strokesVal.drop("stroke", axis = 1).drop("id", axis = 1)

means = strokesTrain.groupby(["stroke"]).mean()
var = strokesTrain.groupby(["stroke"]).var()
classes = np.unique(strokesTrain["stroke"].tolist())
prior = (strokesTrain.groupby(["stroke"]).count()/len(strokesTrain)).
    iloc[:,0]

def Normal(n, mu, var):
    sd = np.sqrt(var)
    pdf = (np.e ** (-0.5 * ((n - mu)/sd) ** 2)) / (sd * np.sqrt(2 * np.
        pi))
```

```
95      return pdf # pdf - probability density function
96
97  def Predict(X):
98      Predictions = []
99
100     for i in X.index: # Loop through each instances
101         ClassLikelihood = []
102         instance = X.loc[i]
103
104         for cls in classes: # Loop through each class
105
106             FeatureLikelihoods = []
107             FeatureLikelihoods.append(np.log(prior[cls])) # Append log
                    prior of class 'cls'
108
109             for col in X.columns: # Loop through each feature
110
111                 data = instance[col]
112
113                 mean = means[col].loc[cls] # Find the mean of column '
                        col' that are in class 'cls'
114                 variance = var[col].loc[cls] # Find the variance of
                        column 'col' that are in class 'cls'
115
116                 Likelihood = Normal(data, mean, variance)
117
118                 if Likelihood != 0:
119                     Likelihood = np.log(Likelihood) # Find the log-
                            likelihood evaluated at x
120                 else:
121                     Likelihood = 1/len(X)
122
123                 FeatureLikelihoods.append(Likelihood)
124
125             TotalLikelihood = sum(FeatureLikelihoods) # Calculate
                    posterior
126             ClassLikelihood.append(TotalLikelihood)
127
128         MaxIndex = ClassLikelihood.index(max(ClassLikelihood)) # Find
                largest posterior position
129         Prediction = classes[MaxIndex]
130         Predictions.append(Prediction)
131
132     return Predictions
133
134 PredictTrain = Predict(xTrain)
135 PredictVal = Predict(xVal)
136
137 def Accuracy(y, prediction):
138
139     # Function to calculate accuracy
140     y = list(y)
141     prediction = list(prediction)
142     score = 0
143
```

```
    for i, j in zip(y, prediction):
        if i == j:
            score += 1

    return score / len(y)

round(Accuracy(yTrain, PredictTrain), 5)

round(Accuracy(yVal, PredictVal), 5)

def ColumnsSelection(y,X):
    for col in X.columns:
        x = X.drop(col, axis = 1)
        prediction = Predict(x)
        print(f"Accuracy without column {col}: {str(round(Accuracy(y,
            prediction)*100, 5))}%")

ColumnsSelection(yTrain,xTrain)

ColumnsSelection(yVal,xVal)

def colSelection(y,X):

    i = 1
    cols_and_acc = dict()
    for col1 in X.columns:
        x1 = X.drop(col1, axis = 1)
        for col2 in x1.columns:
            x2 = x1.drop(col2, axis = 1)
            prediction = Predict(x2)
            cols_and_acc[str(col1 + " and " + col2)] = round(Accuracy(y,
                prediction), 5)
            i+=1
    print(f"Accuracy without columns {max(cols_and_acc, key=cols_and_acc
        .get)}: {str(cols_and_acc[max(cols_and_acc, key=cols_and_acc.get)
        ])}")

colSelection(yVal,xVal)
```