L 6. Random Forest Models. Balancing techniques.

Математичні моделі в продуктовому маркетингу

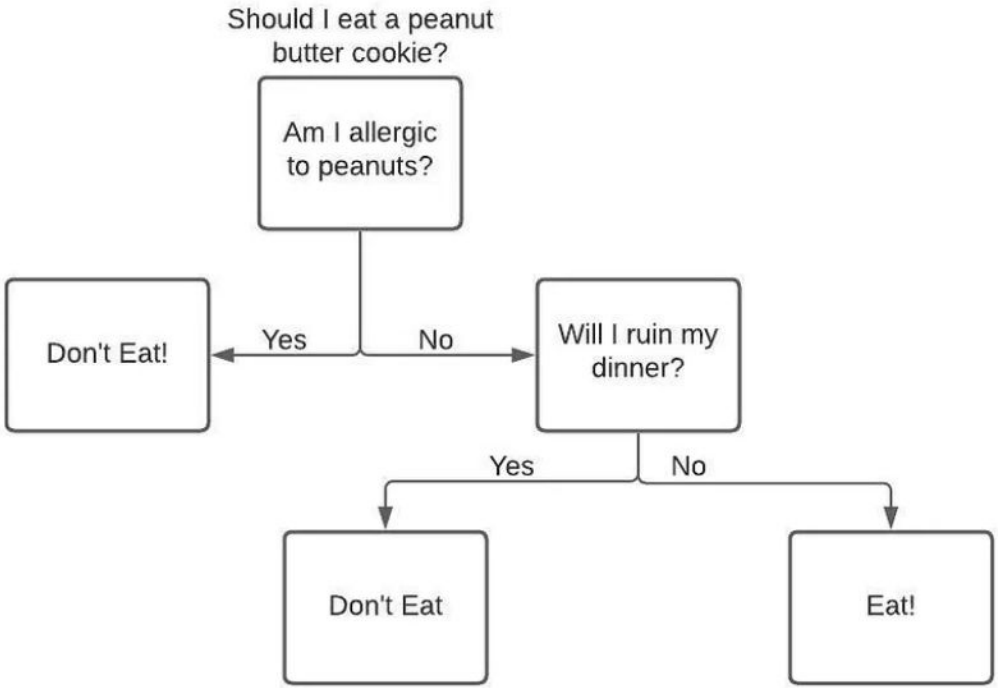
Decision Trees | Моделі дерева рішень

У науці про дані алгоритм дерева рішень — це контрольований (*supervised*) алгоритм навчання для проблем класифікації або регресії.

Алгоритм використовує історичні дані для прогнозування результату.

На відміну від лінійної регресії, дерева рішень можуть виявляти нелінійні взаємозв'язки між змінними в даних.

Приклад моделі дерева рішень: їсти чи ні тістечко



Дерева рішень : вибір змінних для групування

У прикладі з тістечком дерево рішень базується на припущенні, що ви повинні їсти печиво лише за певних критеріїв. Це кінцева мета дерева рішень. Ми хочемо продовжувати приймати рішення (поділи), доки не буде виконано певні критерії.

У цьому прикладі використовуються лише дві змінні (алергія, зіпсована вечеря). Але якщо у вас є набір даних із тисячами змінних/стовпців, як визначити, які змінні/стовпці найефективніше розділити? Популярним способом вирішення цієї проблеми, особливо якщо використовується ID3 algorithm, є використання **entropy** та **information gain**.

Ентропія

У науці про дані ентропія використовується як спосіб вимірювання того, наскільки «змішаними» є значення змінної.

Розглянемо залежну змінну в churn model, яка приймає значення 0 або 1. Чим більше змішані (1) і (0) у стовпці churn, тим вища ентропія.

Якщо кількість значень 1 = кількість значень 0, ентропія = 1.

Якщо всі значення churn = 1, ентропія = 0.

$$Entropy = -\sum_{i=1}^{c} P\left(\mathbf{X}_i\right)\log_b P\left(\mathbf{X}_i\right)$$

Ентропія: приклад розрахунку

$$Entropy = -\sum_{i=1}^{c} P\left(\mathrm{X}_i\right)\log_b P\left(\mathrm{X}_i\right)$$

Розрахуємо ентропію, якщо кількість значень 1 змінної churn = 10, а кількість 0 = 5.

$$Entropy = -\left(\frac{10}{15}\cdot\log_2\left(\frac{10}{15}\right)+\frac{5}{15}\cdot\log_2\left(\frac{5}{15}\right)\right)=-\left(-0.389975+-0.528308\right)=0.918278$$

Використання ентропії для прийняття рішень

$$IG(T,A) = Entropy(T) - \sum_{v \in A} \left| \frac{T_v}{T} \right| \cdot Entropy(T_v)$$

Тут T = Target, our “churn” column

A = атрибут (пояснювальна змінна, яку ми тестуємо для поділу

v = кожне значення в колонці A

Розрахунок Information Gain

1. Наша стартова ентропія для *T* до поділу = 0,918278.
2. Потім для кожного унікального значення (v) у змінній (A) ми обчислюємо кількість рядків, у яких (A) приймає значення (v), і ділимо його на загальну кількість рядків. Для змінної "*наявність м/н тарифу*" ми отримуємо 9/15 для унікального значення (1) і 6/15 для унікального значення (0).
3. Далі ми множимо результати на ентропію рядків, де (A) дорівнює (v). Для лівого розділення (розділити на 1 для "*наявність м/н тарифу*") ми отримуємо 9/15 * .764204. Для правого боку поділу (розділити на 0 для "*наявність м/н тарифу*") ми отримаємо 6/15 * 1.
4. Отримуємо разом 9/14*.764204 + 6/15 = .8585224.
5. **Information Gain = .918278 -.8585224 = .059754**

Розрахунок Information Gain (IG)

Наш інформаційний приріст (IG) становить 0,059754. Що це нам говорить?

Ось альтернативне пояснення. Ми знаходимо ентропію кожного набору після поділу, зважуємо її за кількістю елементів у кожному поділі, а потім віднімаємо з поточної ентропії. Якщо результат позитивний, ми зменшили ентропію за допомогою нашого поділу. Чим вищий результат, тим більше ми знизили ентропію.

У підсумку ми отримуємо 0,059754, що означає, що ми отримуємо 0,059754 біта інформації, розділяючи набір даних на змінну/стовпець «*наявність м/н тарифу*». Наш приріст інформації низький, але він все ще позитивний, тому що ми знизили ентропію в лівій частині розщеплення.

Тепер нам потрібно повторити цей процес для кожного стовпця, який ми використовуємо.

Замість того, щоб робити це вручну, давайте напишемо код Python.

```
import pandas as pd
import numpy as np
import math

midwest = pd.read_csv('midwes.csv')

def calc_entropy(column):
    """
    Calculate entropy given a pandas series, list, or numpy array.
    """
    # Compute the counts of each unique value in the column
    counts = np.bincount(column)
    # Divide by the total column length to get a probability
    probabilities = counts / len(column)

    # Initialize the entropy to 0
    entropy = 0
    # Loop through the probabilities, and add each one to the total entropy
    for prob in probabilities:
        if prob > 0:
            # use log from math and set base to 2
            entropy += prob * math.log(prob, 2)

    return -entropy
```

```
def calc_information_gain(data, split_name, target_name):
    """
    Calculate information gain given a data set, column to split on, and target
    """
    # Calculate the original entropy
    original_entropy = calc_entropy(data[target_name])

    #Find the unique values in the column
    values = data[split_name].unique()



    # Make two subsets of the data, based on the unique values
    left_split = data[data[split_name] == values[0]]
    right_split = data[data[split_name] == values[1]]

    # Loop through the splits and calculate the subset entropies
    to_subtract = 0
    for subset in [left_split, right_split]:
        prob = (subset.shape[0] / data.shape[0])
        to_subtract += prob * calc_entropy(subset[target_name])

    # Return information gain
    return original_entropy - to_subtract
```

```
columns = ['intl tarif', 'sms', 'premium']


def highest_info_gain(columns):
 #Intialize an empty dictionary for information gains
 information_gains = {}

 #Iterate through each column name in our list
 for col in columns:
   #Find the information gain for the column
   information_gain = calc_information_gain(churn, col, 'churn’)
   #Add the information gain to our dictionary using the column name as the ekey
   information_gains[col] = information_gain

 #Return the key with the highest value
 return max(information_gains, key=information_gains.get)
```

```
print(highest_info_gain(churn, columns, 'churn'))
//intl tarif
```

Висновок: перша змінна для поділу є intl tarif

Random Forest Model

Random Forest — це контрольований алгоритм машинного навчання, який широко використовується в задачах класифікації та регресії. Він будує дерева рішень на різних вибірках. При цьому результат базується на більшості голосів для задачі класифікації та на середньому значенні у випадку регресії.

Для розуміння як працює даний алгоритм ознайомтеся зі статтею:

Understand Random Forest Algorithms With Examples

Інтерпретація моделі Random Forest with SHAP

SHAP (SHapley Additive exPlanation) values використовуються для оцінки важливості поточних значень кожної пояснювальної змінної (порівняно з їх невизначеними значеннями) у прогнозуванні залежної змінної (результату) -> Feature Importance Analysis - $E[f(x)|x_s]$ (expected value of Y given X).

Для підрахунку SHAP values існує python-бібліотека shap, яка може працювати з багатьма ML-моделями (XGBoost, CatBoost, TensorFlow, scikit-learn та ін.) та має документацію з великою кількістю прикладів. За допомогою бібліотеки SHAP можна будувати різні схеми та графіки, що описують важливість ознак у моделі та їх вплив на результат Y.

Більше інформації отримайте в статті
https://habr.com/ru/companies/ods/articles/599573/#3

Покращення точності моделі: балансування класів

Незбалансована вибірка даних характеризується нерівномірним розподілом спостережень цільового класу (Y), тобто одна мітка класу має дуже велику кількість спостережень, а інша має дуже низьку кількість спостережень.

Приклади вибірок з незбалансованими класами

Приклади незбалансованої вибірки даних:

· Disease diagnosis

· Customer churn prediction

· Fraud detection

· Natural disaster

Незбалансованість класу, як правило, нормальна в проблемах класифікації. Але в деяких випадках цей дисбаланс є досить гострим, коли присутність класу більшості є набагато вищою, ніж класу меншості.

Приклад результатів моделі з незбалансованими класами

Припустимо, що ми збираємося передбачити захворювання на основі наявного набору даних, де на кожні 100 записів лише у 5 пацієнтів діагностовано захворювання.

Таким чином, більшість становить 95% без захворювання, а меншість – лише 5% хворих. Є висока ймовірність, що модель передбачає, що всі 100 із 100 пацієнтів не мають захворювання.

Використовуючи Confusion matrix, обрахуємо точність моделі (0+95)/(0+95+0+5)=95%. Це означає, що модель не в змозі визначити клас меншості, при цьому показник точності моделі доволі високий 95%.

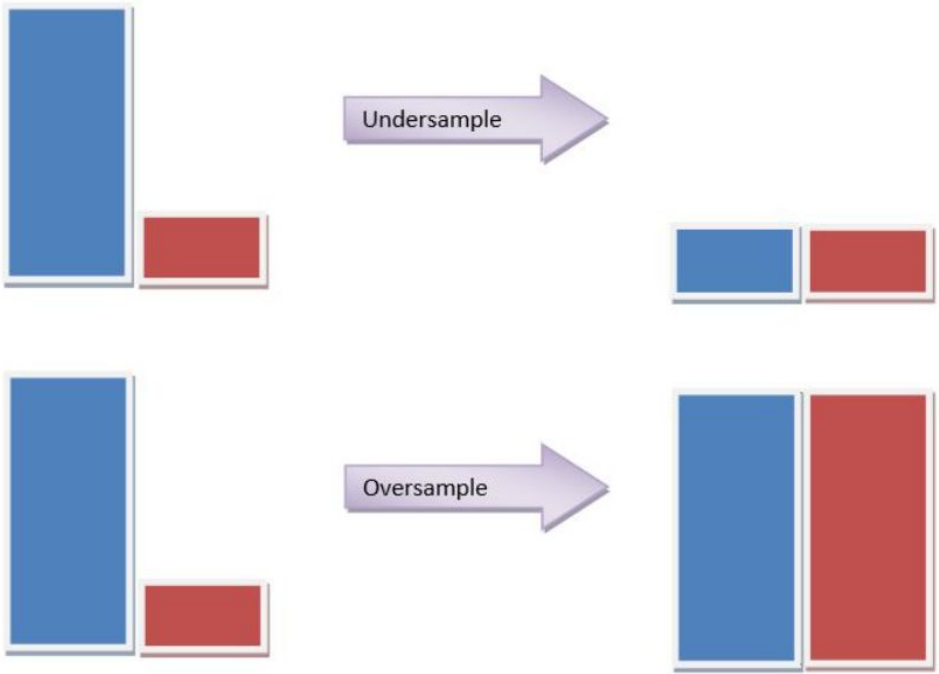
Підходи до вирішення проблеми незбалансованого набору даних

1. Виберіть правильну метрику оцінки точності

   F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

2. Resampling (Oversampling and Undersampling)
3. Synthetic Minority Oversampling Technique or SMOTE
4. Balanced Bagging Classifier
5. Threshold moving

Resampling (Oversampling, Undersampling & SMOTE)



For details on estimation see:
https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/

Threshold moving

В моделі класифікації ми прогнозуємо ймовірність приналежності до класу і відносимо ймовірності цих прогнозів до певного класу на основі порогу, який зазвичай становить 0,5, тобто якщо ймовірності < 0,5, це належить до певного класу, а якщо ні, то до іншого класу.

Для проблем незбалансованого класу цей поріг за замовчуванням може не працювати належним чином. Нам потрібно змінити порогове значення на оптимальне, щоб воно могло ефективно розділяти два класи. Ми можемо використовувати **криві ROC і криві Precision-Recall**, щоб знайти оптимальне порогове значення для класифікатора. Ми також можемо використовувати **grid method** або пошук у межах набору значень, щоб визначити оптимальне значення.

For details on estimation see:
https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/