Рубежный контроль №2

Ф.И.О.: Ергалиев Аслан
Группа: РТ5-61Б
Вариант: 4

Датасет

Данный датасет является вымышленным и предназначен для целей анализа данных и построения простых моделей машинного обучения. Он содержит информацию о 150 000 человек и включает в себя следующие признаки:

- **Number** — индекс строки (не имеет смысловой нагрузки для анализа)
- **City** — город проживания (Dallas, New York City, Los Angeles, Mountain View, Boston, Washington D.C., San Diego, Austin)
- **Gender** — пол (Male или Female)
- **Age** — возраст (от 25 до 65 лет)
- **Income** — годовой доход (от -674 до 177175)
- **Illness** — наличие заболевания (Yes или No)

Описание датасета

Данные являются синтетическими и специально сгенерированы таким образом, чтобы их распределения были удобны для статистического анализа. Целевой переменной является столбец **Illness**, который отражает наличие или отсутствие заболевания у человека. Задача — предсказать наличие заболевания на основе социально-демографических признаков.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, f1_score,
classification_report, confusion_matrix
from sklearn.metrics import precision_score, recall_score
```

Загрузка данных

```
df = pd.read_csv('toy_dataset.csv')
df.head(10)
```

```
   Number    City  Gender  Age   Income Illness
0       1  Dallas    Male   41  40367.0      No
1       2  Dallas    Male   54  45084.0      No
2       3  Dallas    Male   42  52483.0      No
3       4  Dallas    Male   40  40941.0      No
4       5  Dallas    Male   46  50289.0      No
5       6  Dallas  Female   36  50786.0      No
6       7  Dallas  Female   32  33155.0      No
7       8  Dallas    Male   39  30914.0      No
8       9  Dallas    Male   51  68667.0      No
9      10  Dallas  Female   30  50082.0      No
```

Предобработка данных

Нам не нужен столбец number, потому что какой-либо информации он не несет. Так же нужно будет обработать категориальные признаки City и Gender. Гендер можно перевести в 0 и 1. City надо будет глянуть сколько там уникальных значений и сделать вывод какой метод применить, но скорее всего one-hot

Проверить какие значения имеются в income. В описании написано, что есть отрицательный доход. Неизвестно, что это может быть значит и в жизни такого нет. Надо будет проверить сколько лиц имеет отрицательный доход. Если их много придется усреднить или удалить если их мало

Ну и естественно проверить на пропущенные значения

```
df = df.drop(columns=['Number'])

unique_cities = df['City'].nunique()
unique_genders = df['Gender'].unique()
gender_mapping = {'Male': 0, 'Female': 1}
df['Gender'] = df['Gender'].map(gender_mapping)


negative_income_count = (df['Income'] < 0).sum()
total_rows = len(df)
missing_values = df.isnull().sum()

{
    "Уникальные города (City)": unique_cities,
    "Уникальные значения в Gender": list(unique_genders),
    "Количество отрицательных доходов": negative_income_count,
    "Всего строк": total_rows,
    "Пропущенные значения по столбцам": missing_values.to_dict()
}
```

```
{'Уникальные города (City)': 8,
 'Уникальные значения в Gender': ['Male', 'Female'],
 'Количество отрицательных доходов': np.int64(1),
 'Всего строк': 150000,
```

```
 'Пропущенные значения по столбцам': {'City': 0,
  'Gender': 0,
  'Age': 0,
  'Income': 0,
  'Illness': 0}}
```

Пишет что человек с отрицательным доходом всего один. Так что испепелить его не так уж и страшно

Городов всего 8. Можно кодировать по one-hot

```
df = df[df['Income'] >= 0]
df = pd.get_dummies(df, columns=['City'], prefix='City')
df.head()
```

```
   Gender  Age   Income Illness  City_Austin  City_Boston  City_Dallas
\
0       0   41  40367.0      No        False        False         True

1       0   54  45084.0      No        False        False         True

2       0   42  52483.0      No        False        False         True

3       0   40  40941.0      No        False        False         True

4       0   46  50289.0      No        False        False         True


   City_Los Angeles  City_Mountain View  City_New York City  City_San
Diego  \
0             False               False               False
False
1             False               False               False
False
2             False               False               False
False
3             False               False               False
False
4             False               False               False
False

   City_Washington D.C.
0                 False
1                 False
2                 False
3                 False
4                 False
```

```
illness_mapping = {'No': 0, 'Yes': 1}
df['Illness'] = df['Illness'].map(illness_mapping)
```

```
X = df.drop(columns=['Illness'])
y = df['Illness']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((119999, 11), (30000, 11), (119999,), (30000,))
```

Обучаем модель

Модель 1: Дерево решений Используем метрику recall потому что она показывает сколько реально больных из больных мы нашли и F1-score, так как у нас несбалансированные классы и он неплохо подойдет

```
from sklearn.tree import DecisionTreeClassifier
tree_model = DecisionTreeClassifier(class_weight='balanced',
random_state=42)
tree_model.fit(X_train, y_train)
y_pred_tree = tree_model.predict(X_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
precision_tree = precision_score(y_test, y_pred_tree)
recall_tree = recall_score(y_test, y_pred_tree)
f1_tree = f1_score(y_test, y_pred_tree)
report_tree = classification_report(y_test, y_pred_tree,
output_dict=True)

{
    "F1-score (Дерево решений)": f1_tree,
    "Recall (Дерево решений)": recall_tree
}
```

```
{'F1-score (Дерево решений)': 0.08171687990094924,
 'Recall (Дерево решений)': 0.0815485996705107}
```

По метрикам все плохо. Потому что присутствует явный дисбаланс классов

```
# from imblearn.over_sampling import SMOTE
# smote = SMOTE(random_state=42)
# X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# tree_model = DecisionTreeClassifier(class_weight='balanced',
random_state=42)
# tree_model.fit(X_resampled, y_resampled)
# y_pred_tree = tree_model.predict(X_test)
# accuracy_tree = accuracy_score(y_test, y_pred_tree)
# precision_tree = precision_score(y_test, y_pred_tree)
# recall_tree = recall_score(y_test, y_pred_tree)
# f1_tree = f1_score(y_test, y_pred_tree)
```

```
# report_tree = classification_report(y_test, y_pred_tree,
output_dict=True)

# {
#     "F1-score (Дерево решений)": f1_tree,
#     "Recall (Дерево решений)": recall_tree
# }
```

```
{'F1-score (Дерево решений)': 0.10495626822157435,
 'Recall (Дерево решений)': 0.14827018121911037}
```

Модель 2: Градиентный бустинг

```
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train, y_train)
y_pred_gb = gb_model.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
f1_gb = f1_score(y_test, y_pred_gb)
precision_gb = precision_score(y_test, y_pred_gb)
recall_gb = recall_score(y_test, y_pred_gb)

{
    "F1-score": f1_gb,
    "Recall": recall_gb
}
```

```
{'F1-score': 0.0, 'Recall': 0.0}
```

В общем все совсем плохо по результатам оценки качества. Надо что-то делать с балансом классом явно. Попробую юзнуть SMOTE

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)
gb_smote_model = GradientBoostingClassifier(random_state=42)
gb_smote_model.fit(X_train_sm, y_train_sm)
y_pred_gb_sm = gb_smote_model.predict(X_test)
accuracy_gb_sm = accuracy_score(y_test, y_pred_gb_sm)
f1_gb_sm = f1_score(y_test, y_pred_gb_sm)
precision_gb_sm = precision_score(y_test, y_pred_gb_sm)
recall_gb_sm = recall_score(y_test, y_pred_gb_sm)

{
    "Accuracy (GB + SMOTE)": accuracy_gb_sm,
    "F1-score": f1_gb_sm,
    "Precision": precision_gb_sm,
    "Recall": recall_gb_sm
}
```

```
{'Accuracy (GB + SMOTE)': 0.8419333333333333,
 'F1-score': 0.09191880505553428,
 'Precision': 0.08589835361488905,
 'Recall': 0.09884678747940692}
```

Ну видно что метрики стали лучше, однако надо все равно играться с гиперпараметрами моделей и возможно можно будет улучшить результат. А так вся проблема в сильном дисбалансе классов