Определение ССЗ

В рамках конкурса вам нужно предсказать наличие сердечно-сосудистых заболеваний по результатам классического врачебного осмотра. Датасет сформирован из 100.000 реальных клинических анализов, и в нём используются признаки, которые можно разбить на 3 группы:

Объективные признаки:

- Возраст
- Рост
- Вес
- Пол

Результаты измерения:

- Артериальное давление верхнее и нижнее
- Холестерин
- Глюкоза

Субъективные признаки:

- Курение
- Употребление Алкоголя
- Физическая активность

Возраст дан в днях. Значения показателей холестерина и глюкозы представлены одним из трех классов: норма, выше нормы, значительно выше нормы. Значения субъективных признаков — бинарны.

Все показатели даны на момент осмотра.

Таргет - наличие сердечно-сосудистых заболеваний (ССЗ)

Ввод [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline, make_pipeline

from sklearn.metrics import precision_recall_curve, roc_curve, roc_auc_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import itertools
```

```
from sklearn.metrics import roc_auc_score, log_loss

#
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
```

Ввод [2]:

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Ввод [3]:

```
df = pd.read_csv('train_case2.csv', ';')
```

Разделим наши данные на тренировочную и тестовую выборки

Ввод [4]:
```
#разделим данные на train/test
X_train, X_test, y_train, y_test = train_test_split(df.drop('cardio', 1),
                                                    df['cardio'], random_state=0)
```

К полям:

- gender, cholesterol применим OHE-кодирование
- age, height, weight, ap_hi, ap_lo - standardScaler
- gluc, smoke, alco, active - оставим пока как есть

Ввод [5]:
```
class ColumnSelector(BaseEstimator, TransformerMixin):
    """
    Transformer to select a single column from the data frame to perform additional transformations on
    """
    def __init__(self, key):
        self.key = key

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X[self.key]

class NumberSelector(BaseEstimator, TransformerMixin):
    """
    Transformer to select a single column from the data frame to perform additional transformations on
    Use on numeric columns in the data
    """
    def __init__(self, key):
        self.key = key

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X[[self.key]]

class OHEEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, key):
        self.key = key
        self.columns = []

    def fit(self, X, y=None):
        self.columns = [col for col in pd.get_dummies(X, prefix=self.key).columns]
```

```
        return self

    def transform(self, X):
        X = pd.get_dummies(X, prefix=self.key)
        test_columns = [col for col in X.columns]
        for col_ in test_columns:
            if col_ not in self.columns:
                X[col_] = 0
        return X[self.columns]


from sklearn.preprocessing import StandardScaler


continuos_cols = ['age', 'height', 'weight', 'ap_hi', 'ap_lo']
cat_cols = ['gender', 'cholesterol']
base_cols = ['gluc', 'smoke', 'alco', 'active']

continuos_transformers = []
cat_transformers = []
base_transformers = []

for cont_col in continuos_cols:
    transfomer =  Pipeline([
                ('selector', NumberSelector(key=cont_col)),
                ('standard', StandardScaler())
            ])
    continuos_transformers.append((cont_col, transfomer))

for cat_col in cat_cols:
    cat_transformer = Pipeline([
                ('selector', ColumnSelector(key=cat_col)),
                ('ohe', OHEEncoder(key=cat_col))
            ])
    cat_transformers.append((cat_col, cat_transformer))

for base_col in base_cols:
    base_transformer = Pipeline([
                ('selector', NumberSelector(key=base_col))
            ])
    base_transformers.append((base_col, base_transformer))
```

Теперь объединим все наши трансформеры с помощью FeatureUnion

Ввод [6]:
```
from sklearn.pipeline import FeatureUnion

feats = FeatureUnion(continuos_transformers+cat_transformers+base_transformers)
```

```
feature_processing = Pipeline([('feats', feats)])

feature_processing.fit_transform(X_train)
```

Out[6]:
```
array([[-1.73391771,  0.6873301 ,  0.74843904, ...,  1.        ,
         0.        ,  1.        ],
       [-1.67343538,  0.07758923, -0.29640123, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.13738132,  1.17512278, -0.15708919, ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 1.17775864,  1.17512278, -0.15708919, ...,  0.        ,
         0.        ,  1.        ],
       [-0.47190715, -1.38578883,  0.74843904, ...,  0.        ,
         0.        ,  1.        ],
       [ 0.38174619,  0.56538192, -0.08743318, ...,  0.        ,
         0.        ,  1.        ]])
```

Ввод [7]:
```
columns = ['classifier', 'Precision', 'Recall', 'F1_score', 'roc_auc_score', 'log_loss_score']

classifiers = {'LogisticRegression': LogisticRegression(random_state=42),
               'KNeighbors': KNeighborsClassifier(n_neighbors=50),
               'RandomForest': RandomForestClassifier(max_depth=9, max_features=5, n_estimators=160),
               'AdaBoost': AdaBoostClassifier()}

result = pd.DataFrame(columns=columns)

for name_classifier in classifiers:
    classifier = Pipeline([
                    ('features',feats),
                    ('classifier', classifiers[name_classifier])
                    ])

    #запустим кросс-валидацию
    cv_scores = cross_val_score(classifier, X_train, y_train, cv=16, scoring='roc_auc')
    cv_score = np.mean(cv_scores)
    cv_score_std = np.std(cv_scores)

    #обучим пайплайн на всем тренировочном датасете
    classifier.fit(X_train, y_train)
    y_score = classifier.predict_proba(X_test)[:, 1]

    b=1
    precision, recall, thresholds = precision_recall_curve(y_test.values, y_score)
    fscore = (1+b**2)*(precision * recall) / (b**2*precision + recall)
    # locate the index of the largest f score
    ix = np.argmax(fscore)
    data = pd.DataFrame({'classifier': [name_classifier],
```

```
                            'Precision': [precision[ix]],
                            'Recall': [recall[ix]],
                            'F1_score': [fscore[ix]],
                            'roc_auc_score': [roc_auc_score(y_true=y_test, y_score=classifier.predict_proba(X_test)[:,1])],
                            'log_loss_score': [log_loss(y_true=y_test, y_pred=classifier.predict_proba(X_test)[:,1])]},
                           columns=columns)
    result = result.append(data, ignore_index=True)

result
```

Out[7]:

	classifier	Precision	Recall	F1_score	roc_auc_score	log_loss_score
0	LogisticRegression	0.647431	0.837558	0.730323	0.784035	0.577960
1	KNeighbors	0.584798	0.881912	0.703261	0.729862	0.609077
2	RandomForest	0.666421	0.831797	0.739982	0.801898	0.540746
3	AdaBoost	0.692471	0.789401	0.737766	0.794572	0.686977

Лучше других справилась модель RandomForestClassifier.

Задача 5

ROC_AUC

- TPR_1 = 90/100 = 0.9
- FPR_1 = 10/(10 + 99890) = 0.0001
- TPR_2 = 90/100 = 0.9
- FPR_2 = 910/(910 + 98990) = 0.0091

PRECISION_RECALL

- precision_1 = 90/(90+10) = 0.9
- recall_1 = 90/(90+10) = 0.9
- precision_2 = 90 / (90 + 910) = 0.09
- recall_2 = 90/(90+10) = 0.9

Первая модель лучше, поскольку precision выше на 0.81 при равных значениях recall. Легче сделать вывод позволяет precision_recall_curve, поскольку видна существенная разница в точности.

Домашнее задание

1. обучить несколько разных моделей на наборе данных CC3 (train_case2.csv): логрег, бустинг, лес и т.д - на ваш выбор 2-3 варианта
2. при обучении моделей обязательно использовать кроссвалидацию
3. вывести сравнение полученных моделей по основным метрикам классификации: pr/rec/auc/f_score (можно в виде таблицы, где строки - модели, а столбцы - метрики)
4. сделать выводы о том, какая модель справилась с задачей лучше других
5. (опциональный вопрос) какая метрика (precision_recall_curve или roc_auc_curve) больше подходит в случае сильного дисбаланса классов? (когда объектов одного из классов намного больше чем другого).

p.s.В вопросе проще разобраться, если вспомнить оси на графике roc auc curve и рассмотреть такой пример:

Имеется 100000 объектов, из которых только 100 - класс "1" (99900 - класс "0", соответственно). Допустим, у нас две модели:

- первая помечает 100 объектов как класс 1, но TP = 90
- вторая помечает 1000 объектов как класс 1, но TP такой же - 90

Какая модель лучше и почему? И что позволяет легче сделать вывод - roc_auc_curve или precision_recall_curve?

Ввод [ ]: