Задача 4

1. Обработайте набор данных набор данных, указанный во втором столбце таблицы 1, подготовив его к решению задачи классификации. Выделите целевой признак, указанный в последнем столбце таблицы, и удалите его из данных, на основе которых будет обучаться классификатор. Разделите набор данных на тестовую и обучающую выборку. Постройте классификатор типа, указанного в третьем столбце, для задачи классификации по параметру, указанному в последнем столбце. Оцените точность построенного классификатора с помощью метрик precision, recall и F1 на тестовой выборке.

Таблица 4.1. Набор данных.

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Набор данных | Тип классификатора | Классификация по столбцу |
| 18 | Credit card customers | DecisionTreeClassifier | Customer Age (выше среднего значения – класс 0, ниже или совпадает – класс 1) |

После удаления столбца с целевым признаком (см. Таблица 4.1) необходимо привести данные к «хорошему» виду. Для начала, такие столбцы как “Gender”, “Attrition\_Flag”, “Education Level”, “Marital Status”, “Income Category” и “Card Category” были приведены к бинарному виду с помощью one hot encoding. Затем, столбцы с действительными числами “Customer Age”, “Months on book”, “Total Revolving Bal”, “Avg Open To Buy”, “Total Amt Chng Q4 Q1”, “Total Trans Amt”, “Total Trans Ct”, “Total Ct Chng Q4 Q1”, “Credit Limit” были также ортнормированы.

Далее, столбец Customer Age был промаркирован в соответствии с требованием задачи – если значение больше 0, т.е. больше среднего значения в неотнормированной выборке, то данная строка маркируется как 0, иначе маркируется как 1.

Наконец, выборка была поделена на тренировочную (77%) и тестовую (33%), а на ее основе с помощью пакета sklearn был обучен первый классификатор без подбора параметров. Данный классификатор имеет глубину 4 и 16 листьев.

Рисунок 4.1. Обученное дерево решений высотой 4.

Судя по полученному классификатору (см. Рисунок 4.1), если человек зарегистрирован в банковской системе был недавно, или же у него зарегистрировано много доходов на иждивении и у него меньше транзакций, то данный человек имеет возраст меньше среднего. Проверим точность построенного классификатора.

Таблица 4.2. Точность полученного классификатора.

|  |  |
| --- | --- |
| Метрика | Оценка |
| f1 | 78% |
| precision | 76% |
| recall | 82% |

Точность полученного классификатора (см. Таблица 4.2) приемлемая.

Попробуем подобрать гиперпараметры с помощью Grid Search чтобы получить более точную модель. Пускай минимальная глубина варьируется от 2 до 30, а выборку можно будет разбить от 2 до 10 подвыборок.

После обучения было получено дерево высотой 3.

Рисунок 4.2. Обученное дерево с помощью Grid Search высотой 4.

Судя по полученному классификатору (см. Рисунок 4.2), если человек был зарегистрирован недавно, или он имеет много доходов на иждивении, то его возраст меньше среднего. Также проверим точность классификатора.

Таблица 4.3. Точность полученного классификатора с использованием Grid Search.

|  |  |
| --- | --- |
| Метрика | Оценка |
| f1 | 79% |
| precision | 74% |
| recall | 85% |

Полученный классификатор высотой 3 (см. Таблица 4.3) имеет схожую точность с классификатором высотой 4 (см. таблица 4.2).

1. Постройте классификатор типа Случайный Лес (Random Forest) для решения той же задачи классификации. Оцените его качество с помощью метрик precision, recall и F1 на тестовой выборке. Какой из классификаторов оказывается лучше?

Обучим случайный лес со следующими параметрами:

* Возможное число деревьев – от 5 до 400.
* Максимальная глубина – от 2 до 20
* Критерий обучения – критерий Джини.

После обучения была получена модель из 100 деревьев. Проверим точность данной модели.

Таблица 4.3. Точность полученного классификатора с использованием случайного леса.

|  |  |
| --- | --- |
| Метрика | Оценка |
| f1 | 80% |
| precision | 75% |
| recall | 85% |

Точность данной модели (см. таблица 4.3) из 100 деревьев практически не превосходит модель из одного дерева с 3 вершинами.

Вывод

После обработки данных построили три различных классификатора – обычный классификатор высотой 4, классификатор с подбором гиперпараметров в Grid Search и классификатор с использованием случайного леса. Все три классификатора имеют примерно одинаковую точность, поэтому разумнее всего использовать наименьший по высоте и количеству деревьев классификатор, а именно один классификатор высотой 3.

Судя по данному классификатору, с вероятностью в ~79% можно определить, имеет ли человек возраст меньше среднего лишь по двум переменным – как долго существует его банковский аккаунт и по количеству доходов на иждивении.

Приложение

import pandas

import numpy as np

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

data = pandas.read\_csv('BankChurners.csv')

data = data.dropna()

data = data.drop('CLIENTNUM', 1)

pandas.unique(data["Gender"].values)

pandas.unique(data["Attrition\_Flag"].values)

pandas.unique(data["Education\_Level"].values)

pandas.unique(data["Marital\_Status"].values)

pandas.unique(data["Income\_Category"].values)

pandas.unique(data["Card\_Category"].values)

data['Gender'] = np.where(data['Gender'] == 'M', 0, 1)

data['Attrition\_Flag'] = np.where(data['Attrition\_Flag'] == 'Existing Customer', 0, 1)

def conditions(x):

if x == 'High School':

return 1

elif x == 'College':

return 1

elif x == 'Doctorate':

return 1

else:

return 0

func = np.vectorize(conditions)

data['Education\_Level'] = func(data['Education\_Level'])

data['Marital\_Status'] = np.where(data['Marital\_Status'] == 'Married', 0, 1)

def conditions(x):

if x == 'Less than $40K':

return 0

elif x == 'Unknown':

return 0

else:

return 1

func = np.vectorize(conditions)

data['Income\_Category'] = func(data['Income\_Category'])

data['Card\_Category'] = np.where(data['Card\_Category'] == 'Blue', 1, 0)

data.head()

ages = data["Customer\_Age"]

data["Customer\_Age"] = (ages - ages.mean()) / np.sqrt(ages)

month = data["Months\_on\_book"]

data["Months\_on\_book"] = (month - month.mean()) / np.sqrt(month)

month = data["Total\_Revolving\_Bal"]

data["Total\_Revolving\_Bal"] = (month - month.mean()) / np.sqrt(month)

month = data["Avg\_Open\_To\_Buy"]

data["Avg\_Open\_To\_Buy"] = (month - month.mean()) / np.sqrt(month)

month = data["Total\_Amt\_Chng\_Q4\_Q1"]

data["Total\_Amt\_Chng\_Q4\_Q1"] = (month - month.mean()) / np.sqrt(month)

month = data["Total\_Trans\_Amt"]

data["Total\_Trans\_Amt"] = (month - month.mean()) / np.sqrt(month)

month = data["Total\_Trans\_Ct"]

data["Total\_Trans\_Ct"] = (month - month.mean()) / np.sqrt(month)

month = data["Total\_Ct\_Chng\_Q4\_Q1"]

data["Total\_Ct\_Chng\_Q4\_Q1"] = (month - month.mean()) / np.sqrt(month)

month = data["Credit\_Limit"]

data["Credit\_Limit"] = (month - month.mean()) / np.sqrt(month)

data.replace([np.inf, -np.inf], np.nan, inplace=True)

data.dropna()

data = data.iloc[:, :-2]

X = data

def conditions(x):

    if x <= 0:

        return 1

    else:

        return 0

func = np.vectorize(conditions)

X['Customer\_Age'] = func(X['Customer\_Age'])

Y = X["Customer\_Age"]

X = X.drop('Customer\_Age', 1)

X.fillna(X.mean(), inplace=True)

Y.fillna(Y.mean(), inplace=True)

x\_train, x\_validation, y\_train, y\_validation = train\_test\_split(X, Y, test\_size=.33, random\_state=1)

T = DecisionTreeClassifier(random\_state=42, max\_depth = 4)

T = T.fit(x\_train, y\_train)

from sklearn.tree import plot\_tree

from matplotlib import pyplot as plt

fig = plt.figure(figsize=(25,25))

\_ = plot\_tree(T, filled=True, class\_names=True)

print("accuracy:", np.average(cross\_val\_score(T, x\_validation, y\_validation, scoring='accuracy')))

print("f1:", np.average(cross\_val\_score(T, x\_validation, y\_validation, scoring='f1')))

print("precision:", np.average(cross\_val\_score(T, x\_validation, y\_validation, scoring='precision')))

print("recall:", np.average(cross\_val\_score(T, x\_validation, y\_validation, scoring='recall')))

import warnings

warnings.filterwarnings('ignore')

tree = DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

            max\_features=None, max\_leaf\_nodes=None,

            min\_impurity\_decrease=0, min\_samples\_leaf=2, min\_samples\_split=3,

            min\_weight\_fraction\_leaf=0, presort=False, random\_state=42,

            splitter='best')

params = {

    'max\_depth': list(range(2, 30)),

    'min\_samples\_split': [2, 3, 4, 5, 6, 7, 8, 9, 10]

}

gridsearch = GridSearchCV(cv=3, error\_score='raise-deprecating',

       estimator=tree, iid='warn', n\_jobs=-1,

       param\_grid=params, pre\_dispatch='2\*n\_jobs',

       refit=True, return\_train\_score='warn',

       verbose=1)

gridsearch.fit(x\_train, y\_train)

T = gridsearch.best\_estimator\_

T

print("accuracy:", np.average(cross\_val\_score(T, x\_validation, y\_validation, scoring='accuracy')))

print("f1:", np.average(cross\_val\_score(T, x\_validation, y\_validation, scoring='f1')))

print("precision:", np.average(cross\_val\_score(T, x\_validation, y\_validation, scoring='precision')))

print("recall:", np.average(cross\_val\_score(T, x\_validation, y\_validation, scoring='recall')))

fig = plt.figure(figsize=(25,25))

\_ = plot\_tree(T, filled=True, class\_names=True)

print("X[7]:", X.columns[7])

print("X[2]:", X.columns[2])

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

param\_grid = { 'n\_estimators': [5, 10, 50, 100, 200, 300, 400],'max\_features': ['auto'],'max\_depth' : list(range(2, 20)), 'criterion' :['gini']}

RFC = GridSearchCV(estimator=RandomForestClassifier(), param\_grid=param\_grid, cv= 5, refit = True)

RFC.fit(x\_train, y\_train)

print("accuracy:"+str(np.average(cross\_val\_score(RFC.best\_estimator\_, x\_validation, y\_validation, scoring='accuracy'))))

print("f1:"+str(np.average(cross\_val\_score(RFC.best\_estimator\_, x\_validation, y\_validation, scoring='f1'))))

print("precision:"+str(np.average(cross\_val\_score(RFC.best\_estimator\_, x\_validation, y\_validation, scoring='precision'))))

print("recall:"+str(np.average(cross\_val\_score(RFC.best\_estimator\_, x\_validation, y\_validation, scoring='recall'))))

len(RFC.best\_estimator\_.estimators\_)

RFC.best\_estimator\_.estimators\_