1. Методами машинного обучения (не статистическими тестами) показать, что разбиение на трейн и тест репрезентативно.

Репрезентативность разбиения - тренировочный и тестовый наборы данных должны быть представительными для общей популяции данных, на которых будет применяться модель машинного обучения.

Если разбиение является репрезентативным, то результаты обучения модели на тренировочном наборе данных будут хорошо обобщаться на новые, ранее не виданные данные.

Для обеспечения репрезентативности разбиения необходимо учитывать:

- *Случайность: Разбиение на тренировочный и тестовый наборы данных должно быть случайным. Это гарантирует, что данные в обоих наборах будут представительными для общей популяции.*
- *Баланс классов: Если в данных присутствует дисбаланс классов, то необходимо убедиться, что разбиение сохраняет пропорцию классов в тренировочном и тестовом наборах данных. В противном случае, модель может быть смещена в сторону класса с большим представлением.*
- *Уникальность данных: Если данные содержат повторяющиеся примеры или зависимости между ними, необходимо убедиться, что эти зависимости сохраняются в обоих наборах данных. Иначе модель может быть переобучена на повторяющихся примерах и показывать плохую обобщающую способность.*
- *Временные зависимости: Если данные имеют временную зависимость, например, в задачах прогнозирования временных рядов, важно учитывать это при разбиении на тренировочный и тестовый наборы данных. Тестовый набор должен содержать данные, которые идут после тренировочного набора по времени, чтобы модель могла быть проверена на способность прогнозировать будущие значения.*

Для того чтобы показать, что разбиение на тренировочный и тестовый наборы данных является репрезентативным, можно использовать:

Кросс-валидацию

Метод позволяет оценить производительность модели на разных подмножествах данных. Например, можно использовать k-кратную кросс-валидацию, где данные разбиваются на k подмножеств, и модель обучается и тестируется k раз. Если результаты на всех подмножествах данных схожи, то это может быть признаком ***репрезентативности разбиения****.*

Анализ распределения

Позволяет провести анализ распределения признаков в тренировочном и тестовом наборах данных. Если распределения признаков схожи, то это может указывать на ***репрезентативность разбиения****. Для этого можно использовать гистограммы, ящики с усами и другие графические методы.*

Сравнение статистик

Сравнивает различные статистические метрики между тренировочным и тестовым наборами данных. Например, можно сравнить средние значения, медианы, дисперсии и другие метрики для каждого признака. Если эти метрики схожи, то это может указывать ***на репрезентативность разбиения****.*

Визуализация данных

Использование визуализации может помочь в оценке ***репрезентативности разбиения****. Можно построить диаграммы рассеяния, графики плотности распределения и другие типы графиков для визуального сравнения данных в тренировочном и тестовом наборах.*

В целом, репрезентативность разбиения является важным аспектом успешного применения моделей машинного обучения. Правильное разбиение данных помогает избежать проблем переобучения и недообучения, а также позволяет получить более точные и обобщающие результаты модели.

Важно отметить, что репрезентативность разбиения зависит от конкретной задачи и данных. Поэтому необходимо применять несколько методов и анализировать результаты с учетом контекста задачи и предметной области.

2. Есть кластеризованный датасет на 4 кластера (1, 2, 3, 4). Бизнес аналитики посчитали, что самым прибыльным является кластер 2. Каждый клиент представлен в виде 10-тимертного вектора, где первые 6 значений транзакции, а оставшиеся: возраст, пол, социальный статус (женат (замужем)/неженат (не замужем)), количество детей. Нужно поставить задачу оптимизации для каждого клиента не из кластера 2 так, чтобы увидеть как должен начать вести себя клиент, чтобы перейти в кластер 2.

Постановка задачи оптимизации, которая поможет решить задачу:

Определение целевой функции

Целевая функция должна отражать желаемый результат - переход клиента из текущего кластера в кластер 2. Можно использовать различные метрики для определения "близости" кластеров, например, евклидово расстояние между векторами клиентов. Цель состоит в минимизации этой функции.

Определение переменных и ограничений

Переменные представляют собой параметры, которые клиент может изменить, чтобы достичь желаемого результата. В данном случае, возможные переменные могут быть связаны с транзакциями, возрастом, полом, социальным статусом и количеством детей. Ограничения могут быть наложены на значения этих переменных, чтобы они соответствовали реалистичным значениям.

Постановка задачи оптимизации для каждого клиента, не находящегося в кластере 2, чтобы перейти в этот кластер, можно использовать следующий подход:

1. Определение переменных и ограничений:
  - Переменные:
Для каждого клиента не из кластера 2, переменные связаны: с транзакциями, возрастом, полом, социальным статусом и количеством детей. Обозначим их x1, x2, ..., x10.
  - Ограничения:
Введем ограничения на значения переменных, : возраст может быть ограничен определенным диапазоном, количество детей может быть неотрицательным и т.д.

2. Определение целевой функции:
  - Целевая функция:
Целевая функция результата - переход клиента из текущего кластера в кластер 2. Используем различные метрики для определения "близости" кластеров, например, евклидово расстояние между векторами клиентов. Цель состоит в минимизации этой функции.

3. Создание модели оптимизации с использованием библиотеки SciPy:
  - Установите библиотеку SciPy,: `pip install scipy`
  - Импортируем модули: `from scipy.optimize import minimize`
  - Определим функцию целевой функции и ограничений:

```
def objective(x):
    # код для расчета целевой функции
    return objective_value

def constraints(x):
    # код для определения ограничений
    return constraints_values
```

  - Определим начальное значение переменных и вызовите функцию оптимизации:

```
x0 = [initial_values]  # Начальные значения переменных
bounds = [(lower_bound, upper_bound)] * len(x0)  # Ограничения на переменные
solution = minimize(objective, x0, bounds=bounds, constraints=constraints)
```

4. Валидация и интерпретация результатов:

- После получения оптимальных значений переменных, проведем валидацию результатов и интерпретируйте их с бизнес-точки зрения. Анализируем, какие конкретные изменения в параметрах клиента могут помочь ему перейти в кластер 2.

3. Что лучше 2 модели случайного леса по 500 деревьев или одна на 1000, при условии, что ВСЕ параметры кроме количества деревьев одинаковы?

Если сравниваем две модели случайного леса с разным количеством деревьев (500 и 1000) и одинаковыми остальными параметрами, учитываем факторы:

1. Вычислительная сложность:

Обучение модели случайного леса затратный процесс. Если есть достаточно вычислительных ресурсов, то обучение модели с 1000 деревьями может быть выполнено без проблем. Однако, если ограниченные вычислительные ресурсы, обучение модели с 500 деревьями может быть предпочтительнее.

2. Размер данных:

Большие объем данных, обучение модели с 1000 деревьями лучше, так как более сложные модели могут лучше захватывать сложные закономерности в данных. Однако, если данных не очень большие, модель с 500 деревьями может быть достаточно.

3. Проблема переобучения:

Увеличение количества деревьев в случайном лесу может привести к большему склонности к переобучению модели, особенно если небольшой объем данных. Поэтому, при увеличении количества деревьев до 1000, важно следить за переобучением и применять соответствующие методы регуляризации, ограничение глубины деревьев или случайная выборка признаков.

4. Оценка модели:

Для принятия решения о том, какая модель лучше необходимо провести оценку моделей на отложенной выборке или с помощью кросс-валидации. По метрикам точность, полнота, F1-мера или кривая ROC, чтобы определить, какая модель лучше справляется с задачей.

5. Вывод:

Выбор между двумя моделями случайного леса с разным количеством деревьев зависит от доступности вычислительных ресурсов, размера данных и желаемых метрик качества модели.

4. В наличии датасет с данными по дефолту клиентов. Как, имея в инструментарии только алгоритм kmeans получить вероятность дефолта нового клиента.

Алгоритм k-means - алгоритмом кластеризации, который разбивает набор данных на кластеры на основе их сходства. Он не предоставляет вероятность дефолта, так как вероятность - это мера уверенности или вероятности того, что определенное событие произойдет.

Использовать алгоритм k-means для создания модели, которая будет предсказывать вероятность дефолта нового клиента. Для этого потребуется набор данных: информацию о клиентах и информацию о том, является ли клиент дефолтным или нет.

Для реализации необходимо:

1. Подготовить набор данных: включить все необходимые признаки, такие как возраст, доход, кредитная история и т.д., а также целевую переменную, указывающую, является ли клиент дефолтным или нет.

2. Выполнить кластеризацию с использованием алгоритма k-means: что позволит разбить данные на группы схожих клиентов. Количество кластеров выбираем экспериментально.

3. Создаем модель предсказания вероятности дефолта: используем обученную модель машинного обучения, такую как логистическая регрессия или случайный лес, чтобы предсказать вероятность дефолта для нового клиента **на основе его признаков и кластера, к которому он относится**.

4. Обучаем модель на обучающем наборе данных: используем исходный набор данных, чтобы обучить модель на основе выбранных признаков и целевой переменной.

5. Оцениваем производительность модели: используя метрики оценки модели, такие как точность, полнота и F1-мера, чтобы оценить производительность модели на тестовом наборе данных.

6. Применяя модель для предсказания вероятности дефолта нового клиента: используя обученую модель для предсказания вероятности дефолта нового клиента на основе его признаков.

Пример решения задачи на Python с использованием библиотеки scikit-learn и pandas:

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# Загрузка данных
data = pd.read_csv('data.csv')
# Подготовка данных
```

```
X = data.drop(['default'], axis=1)  # Признаки
y = data['default']  # Целевая переменная
# Кластеризация с использованием алгоритма k-means
kmeans = KMeans(n_clusters=3)  # Задайте количество кластеров
kmeans.fit(X)
# Предсказание кластера для нового клиента
new_client = [[25, 50000, 1]]  # Пример данных для нового клиента
cluster = kmeans.predict(new_client)
# Формирование нового набора данных с добавлением кластера
X_clustered = X.copy()
X_clustered['cluster'] = kmeans.labels_
# Создание модели предсказания вероятности дефолта
model = LogisticRegression()  # Можно выбрать другую модель
model.fit(X_clustered, y)
# Обучение модели на обучающем наборе данных
predictions = model.predict(X_clustered)
# Оценка производительности модели
accuracy = accuracy_score(y, predictions)
precision = precision_score(y, predictions)
recall = recall_score(y, predictions)
f1 = f1_score(y, predictions)
# Предсказание вероятности дефолта нового клиента
new_client_clustered = new_client.copy()
new_client_clustered.append(cluster[0])
probability = model.predict_proba([new_client_clustered])[0][1]
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Probability of default for new client:", probability)
```

В примере мы использовали алгоритм k-means для кластеризации данных и добавили информацию о кластере в набор данных.

Затем мы создали модель логистической регрессии, используя признаки и кластер, и обучили ее на обучающем наборе данных.

Затем мы оценили производительность модели и предсказали вероятность дефолта для нового клиента на основе его признаков и кластера.

5. Есть выборка клиентов с заявкой на кредитный продукт. Датасет состоит из персональных данных: возраст, пол и т.д. Необходимо предсказывать доход

клиента, который представляет собой непрерывные данные, но сделать это нужно используя только модель классификации.

Для предсказания непрерывных данных: доход клиента, используются модели регрессии, а не модели классификации. Модели классификации предназначены для предсказания категориальных переменных или классов.

Однако, если есть ограничение использовать только модель классификации, тогда нужно применять методы, которые преобразовывают задачу регрессии в задачу классификации.

1. Подход с использованием порогового значения:

- Разделить доход на интервалы или категории: низкий, средний и высокий доход.
- Обучить модель классификации на основе признаков и целевой переменной, где каждый интервал или категория дохода будет представлять отдельный класс.
- После обучения модели, предсказываем класс дохода для новых клиентов на основе их признаков.

2. Подход с использованием ранжирования:

- Отсортировать клиентов по доходу в порядке возрастания.
- Разделить отсортированный список на несколько равных интервалов или квантилей.
- Присвоить каждому клиенту класс на основе интервала или квантили, в которую он попадает.
- Обучить модель классификации на основе признаков и классов, полученных из ранжирования.
- После обучения модели, предсказываем класс дохода для новых клиентов на основе их признаков.

Оба подхода позволяют использовать модель классификации для предсказания дохода клиента, но они не будут точно предсказывают конкретное значение дохода, а только классифицируют клиентов по интервалам или категориям дохода.

пример на python:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, mean_squared_error
# Загрузка данных
data = pd.read_csv('data.csv')
# Подготовка данных
X = data.drop(['income'], axis=1)  # Признаки
```

```
y = data['income']  # Целевая переменная
# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Создание модели классификации
model = LogisticRegression()  # Можно выбрать другую модель
# Обучение модели
model.fit(X_train, y_train)
# Предсказание дохода для тестовых данных
y_pred = model.predict(X_test)
# Оценка производительности модели
accuracy = accuracy_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print("Accuracy:", accuracy)
print("Mean Squared Error:", mse)
```

В этом примере использована модель логистической регрессии как модель классификации.

Загрузили данные, подготовили признаки и целевую переменную, а затем разделили данные на обучающую и тестовую выборки с помощью функции `train_test_split`.

Затем создана модель логистической регрессии и обучена она на обучающей выборке. После обучения предсказываем доход для тестовых данных и оцениваем производительность модели с помощью метрик accuracy и mean squared error.

В данном случае предсказали классы дохода, а не конкретные значения дохода. Если необходимо предсказывать конкретные значения дохода, следует использовать модели регрессии, такие как линейная регрессия или случайный лес.