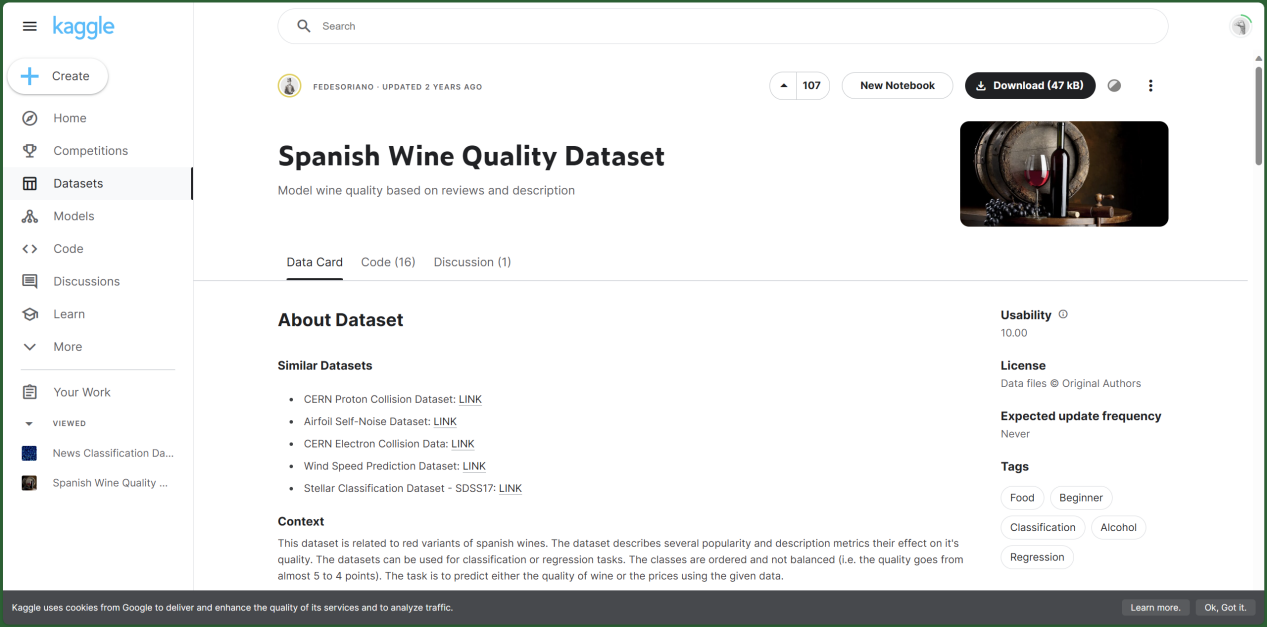
**分类实验报告**

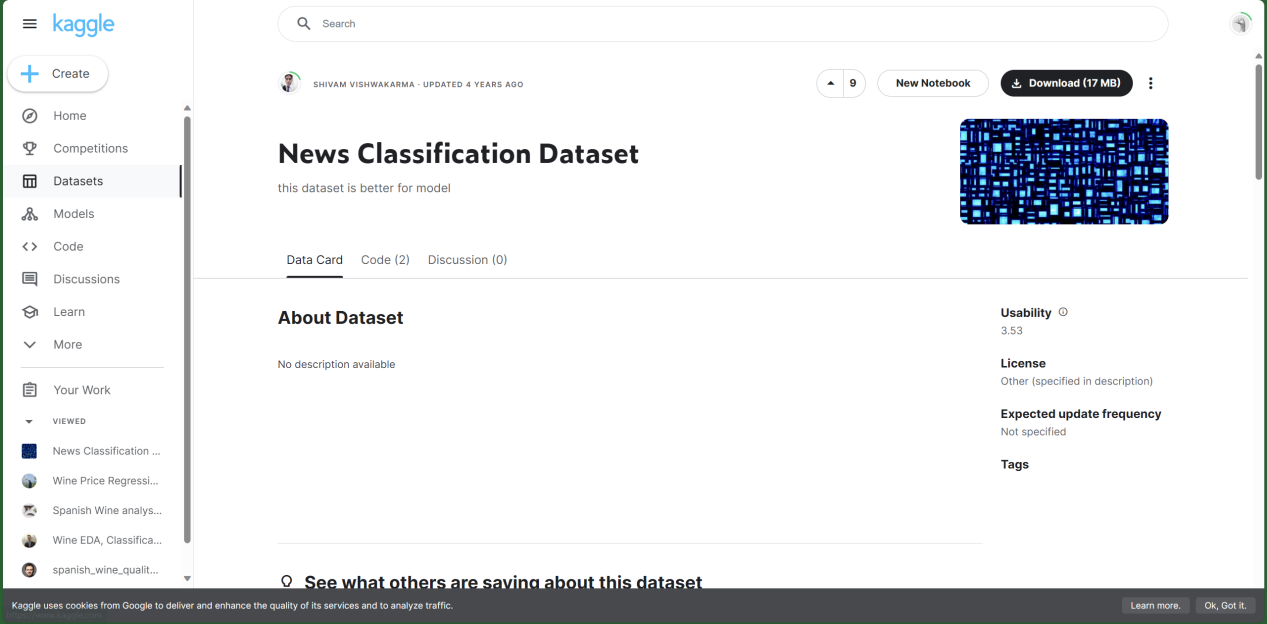
**学号: 366666626 姓名: 张大炮 班级: 3班**

1. **数据**

**西班牙葡萄酒质量数据集（Spanish Wine Quality Dataset）**



新闻分类数据集（News Classification Dataset）



1. 算法

针对**西班牙葡萄酒质量数据集**，首先进行EDA，观察label数据分布

import matplotlib.pyplot as plt

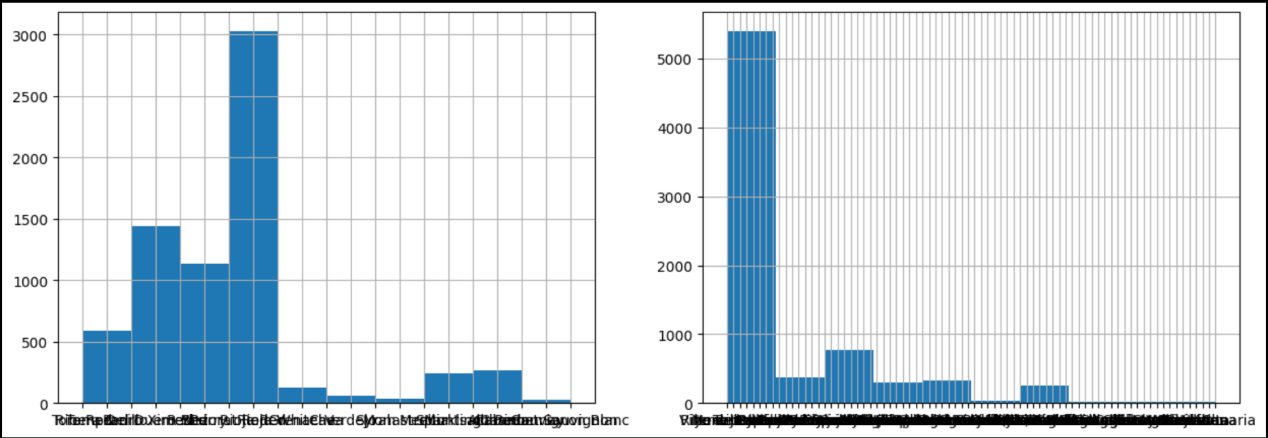
plt.figure(figsize=(15,5))

plt.subplot(121)

df['type'].hist()

plt.subplot(122)

df['region'].hist()



可见对于type和region列都有不同程度的不平衡，其中region label的不平衡成都更严重。但是首先需要进行数据清洗和labelencoder。

df["year"] = df["year"].replace("N.V.",np.NaN)

df = df.dropna()

df['year'] = df['year'].astype(np.int64)

print(df['year'].unique())

上述代码，清洗掉错误的year

from sklearn.preprocessing import LabelEncoder

for col in df.columns:

    if df[col].dtype == "object":

        print(str(col))

        label = LabelEncoder()

        label = label.fit(df[col])

        df[col] = label.transform(df[col].astype(str))

X = df.drop("type",axis=1)

y = df[["type"]]

y = y.values.ravel()

df.head()

上述代码，使用sklearn库内的labelencoder进行标签转化，以便非树状分类器的使用。

from sklearn.preprocessing import StandardScaler

from imblearn.over\_sampling import RandomOverSampler

print(X.\_\_len\_\_())

ros = RandomOverSampler(sampling\_strategy='auto')

X, y = ros.fit\_resample(X, y)

# Initialize the StandardScaler

scaler = StandardScaler()

# Fit and transform the scaler on the DataFrame

X\_standardized = scaler.fit\_transform(X)

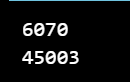
# Convert the result back to a DataFrame (optional)

X\_std = pd.DataFrame(X\_standardized, columns = X.columns)

print(X\_std.\_\_len\_\_())

X\_std.head()

上述代码做了两件事，第一，StandardScaler标准归一化，有助于分类器拟合数据，第二，使用imblearn库中的随机重采样，对数据进行处理，此处策略没有特殊选择，选择了默认的auto。



重采样后的数据数目发生变化，是由于少数label的数据被重复采样造成的。

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_std, y, test\_size=0.6, random\_state=42)

X\_train.head()

上述代码，对数据集进行分割，训练集：测试集为4：6，别看这个比例如此夸张，由于我们做了标准归一化和labelencoder，这能很好地提高模型精度，即使是如此变态的比例也可以得到很好的acc。

from sklearn.svm import SVC

from xgboost import XGBClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

# Initialize and train regression models

models = [

    ("SVC", SVC()),

    ("XGBClassifier", XGBClassifier()),

    ("KNeighborsClassifier", KNeighborsClassifier()),

    ("RandomForestClassifier", RandomForestClassifier()),

]

# Initialize a list to store results

results = []

# Fit and evaluate each model

for name, model in models:

    print(name)

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    ACC = accuracy\_score(y\_test, y\_pred)

    results.append([name, ACC])

# Create a DataFrame from the results list

results\_df = pd.DataFrame(results, columns=["Model", "ACC"])

# Find the model with the lowest MSE and highest R-squared

best\_r2\_model = results\_df.loc[results\_df["ACC"].idxmax()]

print("\nModel Evaluation Results:")

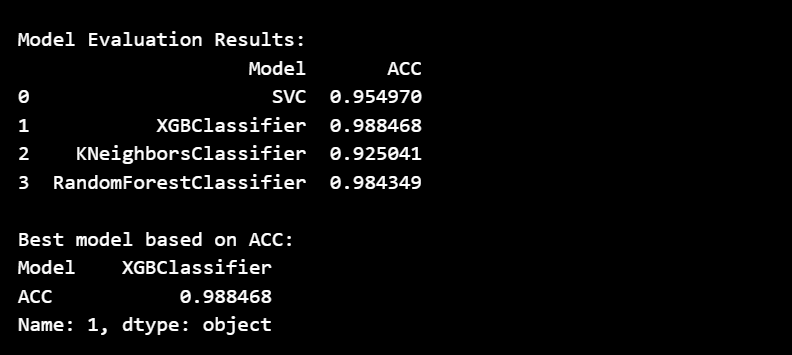
print(results\_df)

print("\nBest model based on ACC:")

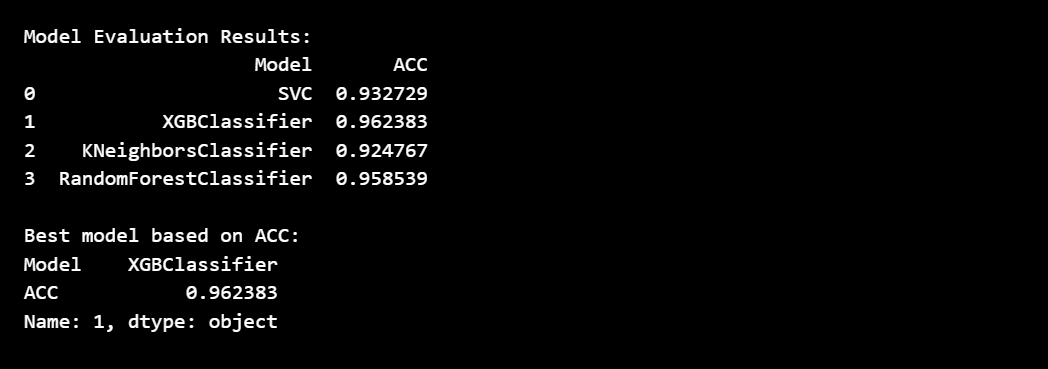
print(best\_r2\_model)

采用sklearn库中的分类器进行实验，由于kaggle官网的bassline是回归器，此处改为分类器，采用四个分类算法进行试验，1是svc支持向量机分类，默认高斯核，2是XGBT分类算法，增强梯度提升树，竞赛常客（竞赛bassline），3是K近邻分类器，采用KNN算法，没有过多调整，4是随机森林分类器，没有调整。

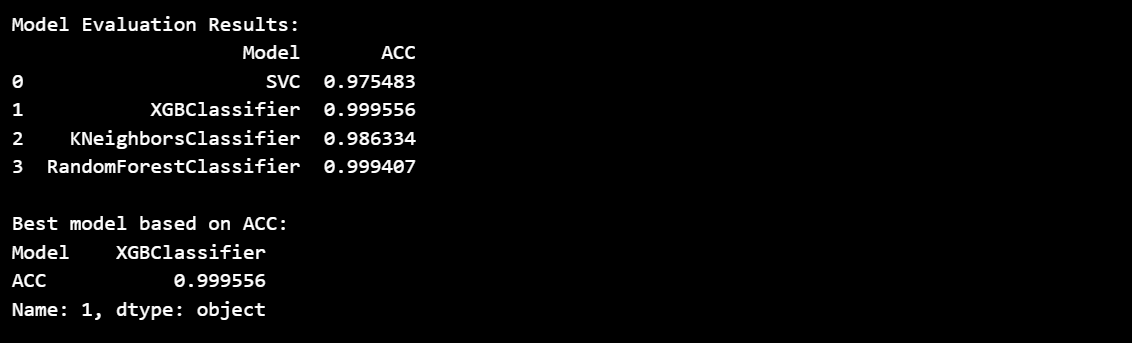
此处首先使用type属性做分类，结果已经非常好了，但是由于是长尾实验的缘故，有些数据即使是分类失败了也看不太出来，因为被盖过去了。



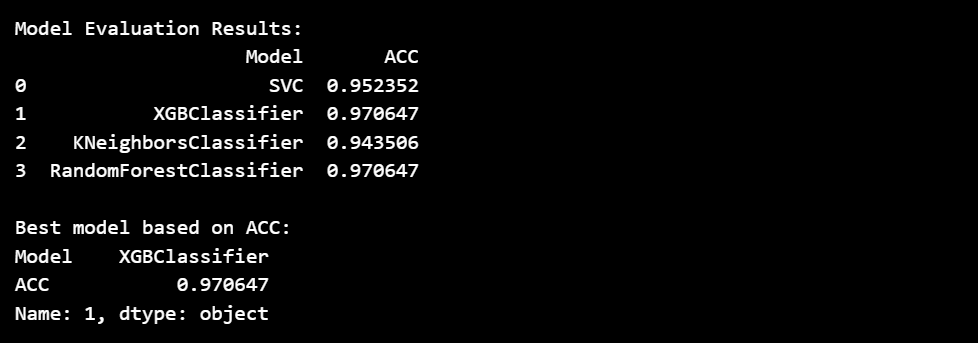
然后使用region属性做分类，数据如下



分类精度很高，但是看一下重采样后的效果，首先使用type属性，将少数类重采样至多数类数量。



几乎全部接近99%，xgbt甚至达到了99.9%，对比上次而言非常离谱。



针对region，由于原数据集的数据分布过于极端，使用过采样会导致数据量暴增，此处选择minority策略，降低多数类采样，并提高少数类采样。Acc都得到了升高，相较于上次普遍提高了1-2个点。

**新闻分类数据集**

其数据量较大，又是文本信息，首先要做的是进行文本特征提取

import pandas as pd

import numpy as np

from sklearn.preprocessing import OneHotEncoder, LabelEncoder

df = pd.read\_csv('News.csv', header='infer')[:2000]

df['text'] = df['headline'] + df['short\_description'] + df['authors']

df['category'] = LabelEncoder().fit\_transform(df['category'])

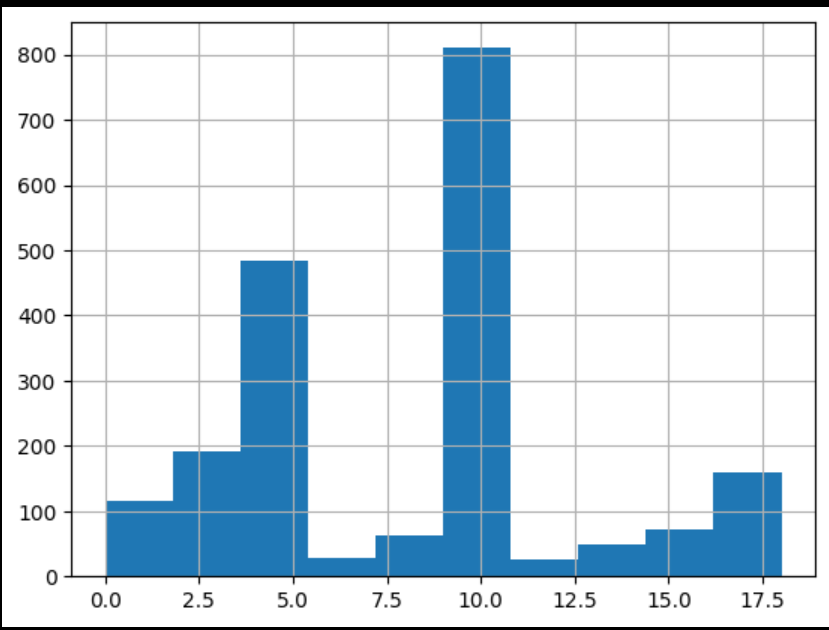
print(df.\_\_len\_\_())

df = df.dropna(how='any')

print(df.\_\_len\_\_())

df.head(1)

进行EDA操作，观察到标签信息是不平衡的。



此处为了实验的进行较为快速，选用前2000个数据进行试验。

from sklearn.feature\_extraction.text import CountVectorizer

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(df['text']).toarray()

y = df['category']

使用sklearn库的CountVectorizer统计词频信息，转化为普通张量。

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

from sklearn.svm import LinearSVC

from xgboost import XGBClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear\_model import SGDClassifier

from sklearn.metrics import accuracy\_score

# Initialize and train regression models

models = [

    ("LinearSVC", LinearSVC()),

    ("DecisionTreeClassifier", DecisionTreeClassifier()),

    ("SGDClassifier", SGDClassifier()),

    ("XGBClassifier", XGBClassifier()),

]

# Initialize a list to store results

results = []

# Fit and evaluate each model

for name, model in models:

    print(name)

    model.fit(X\_train, y\_train)

    y\_pred = model.predict(X\_test)

    ACC = accuracy\_score(y\_test, y\_pred)

    results.append([name, ACC])

# Create a DataFrame from the results list

results\_df = pd.DataFrame(results, columns=["Model", "ACC"])

# Find the model with the lowest MSE and highest R-squared

best\_r2\_model = results\_df.loc[results\_df["ACC"].idxmax()]

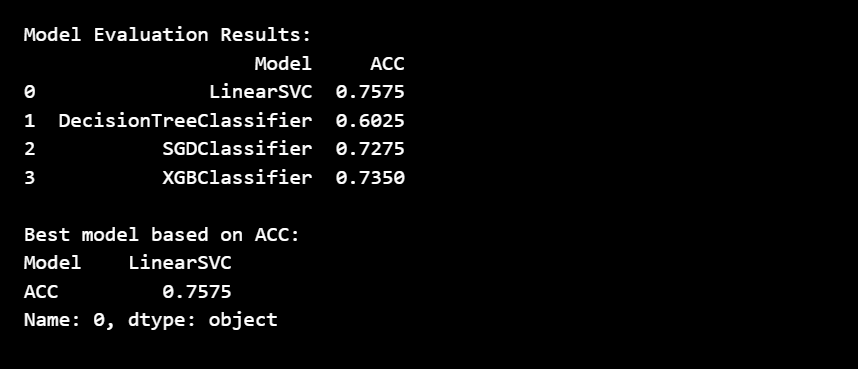
print("\nModel Evaluation Results:")

print(results\_df)

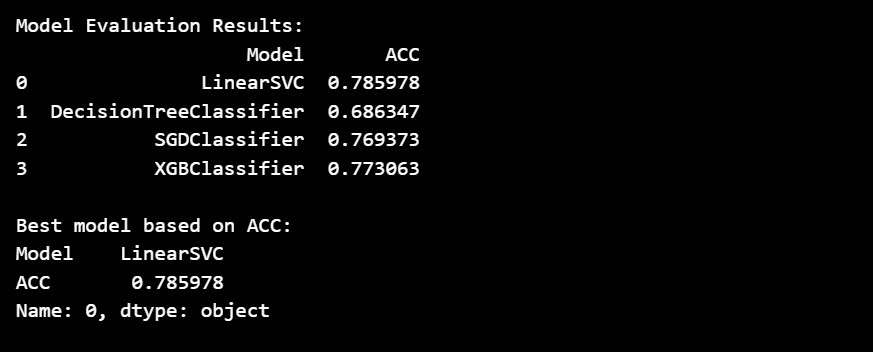
print("\nBest model based on ACC:")

print(best\_r2\_model)

此处选用四种分类模型，LinearSVC线性支持向量机，DecisionTreeClassifier决策树，SGDClassifier随机梯度下降分类器和XGBClassifier加强梯度提升分类。输出信息如下。



现在加入重采样操作。



可观察到所有分类器的精确度都上升了大约5个点。

1. 结论

重采样操作是为了解决数据标签不平衡的问题的方案，在我们的NLP课程中，我们组分到的一个论文，就提到了Unbalanced数据的特性，当时还没有确切的印象，此处设身处地地体会到了重采样的必要性，另外，对于文本信息提取也有了很好的实践，一开始我打算使用llm中的encoder进行文本信息提取，后来由于速度较慢，无法满足要求，最终放弃，采用传统的词频统计进行试验，效果很好。