بسم الله الرحمن الرحیم

موضوع :مینی پروژه – پاسخ سوال دوم

درس مربوط : یادگیری ماشین لرنینگ

اساتید راهنما: آقایان دکتر علیاری و احمدی

دانشجو: ابوذر بختیاری برزیده

شماره دانشجویی: 4021634202

فروردین 1403

پاسخ سوال دوم :

بخش اول

CWRU Bearing به دیتاستی اشاره دارد که برای تشخیص خرابی در بلبرینگ‌ها استفاده می‌شود. این دیتاست شامل داده‌هایی است که از بلبرینگ‌های مختلفی در محیط‌های مختلف جمع‌آوری شده است. هدف اصلی این دیتاست، تشخیص و پیش‌بینی مراحل خرابی بلبرینگ‌ها است. کاربردهای اصلی این دیتاست عبارتند از: 1. تشخیص خرابی 2. پیشگیری از خرابی 3. بهبود عملکرد

به طور خلاصه، دیتاست CWRU Bearing ابزاری است که به مهندسین و تحقیقاتی‌ها کمک می‌کند تا خرابی‌های بلبرینگ را تشخیص داده و برای بهبود عملکرد و ایمنی ماشین‌آلات از آن استفاده کنند.

بعضی از اهداف اصلی دیتاست CWRU Bearing شامل موارد زیر می‌شود:

1. تشخیص و تمیزکاری از عیوب در بلبرینگ‌ها: این دیتاست برای تشخیص و تمیزکاری از عیوب مختلف در بلبرینگ‌ها مانند خوردگی، ترکیدگی، ارتعاشات نامطلوب و ... استفاده می‌شود.

2. آزمون الگوریتم‌های یادگیری ماشین: این دیتاست برای آزمون و ارزیابی عملکرد الگوریتم‌های یادگیری ماشین برای تشخیص عیب در بلبرینگ‌ها مورد استفاده قرار می‌گیرد.

3. ارزیابی روش‌های تشخیص عیب: این دیتاست به عنوان یک بنچمارک استاندارد برای ارزیابی و مقایسه عملکرد روش‌های مختلف تشخیص عیب در بلبرینگ‌ها مورد استفاده قرار می‌گیرد.

ویژگی‌های این دیتاست عموماً شامل سیگنال‌های ارتعاشی از بلبرینگ‌ها است که با استفاده از سنسورهای ارتعاشی جمع‌آوری شده‌اند. هر نمونه از دیتاست ممکن است ویژگی‌های مختلفی داشته باشد از جمله: فرکانس، شدت، مدت زمان، و ... که مرتبط با وضعیت عیب‌دار یا سالم بودن بلبرینگ‌ها هستند.

حالت‌های مختلف این دیتاست شامل بلبرینگ‌های سالم و بلبرینگ‌هایی که به آن‌ها عیبی تحمیل شده است می‌شود. این عیوب ممکن است شامل خوردگی، ترکیدگی، ناهمواری‌های سطحی، افزایش فرکانس ارتعاشات و ... باشند.

بخش دوم قسمت آ : برای ایجاد دیتاست با تعداد نمونه‌های یکسان از هر کلاس، می‌توانید به شکل زیر عمل کنید:

1. انتخاب N نمونه از هر کلاس با طول تعیین شده.

2. تشکیل دادن دو زیرمجموعه از این نمونه‌ها، یک زیرمجموعه برای هر کلاس.

3. ادغام این دو زیرمجموعه به یک دیتاست نهایی.

در ادامه کدی آمده است که این کار را انجام می‌دهد:

```python
import numpy as np
import pandas as pd
# تعداد نمونه‌های هر کلاس
N = 100
# انتخاب N نمونه از هر کلاس با طول تعیین شده
class_1_samples = np.random.randn(N, 2)  # نمونه‌های کلاس اول با ویژگی‌های تصادفی
class_2_samples = np.random.randn(N, 2) + 2  # نمونه‌های کلاس دوم با ویژگی‌های تصادفی و از میانه جدا
# تشکیل دادن دو زیرمجموعه برای هر کلاس
class_1_labels = np.zeros(N)  # برچسب 0 برای کلاس اول
class_2_labels = np.ones(N)  # برچسب 1 برای کلاس دوم
# ادغام داده‌های دو کلاس به یک دیتافریم
data = np.vstack([class_1_samples, class_2_samples])
labels = np.hstack([class_1_labels, class_2_labels])
```

```
# ایجاد دیتافریم نهایی
df = pd.DataFrame(data, columns=['Feature 1', 'Feature 2'])
df['Class'] = labels
# نمایش نمونه‌های اولیه از دیتافریم نهایی
print(df.head())
```

این کد ابتدا N نمونه از هر کلاس را با ویژگی‌های تصادفی ایجاد می‌کند. سپس برچسب‌های متناظر با هر کلاس را ایجاد می‌کند. در نهایت، داده‌ها و برچسب‌ها را به یک دیتافریم Pandas ادغام می‌کند و آن را نمایش می‌دهد.

بخش دوم قسمت ب : استخراج ویژگی‌ها یک مرحله بسیار مهم در فرایند یادگیری ماشین است. این ویژگی‌ها معمولاً مشخصه‌هایی از داده‌ها هستند که اطلاعات مفیدی را ارائه می‌دهند و به ماشین امکان می‌دهند الگوریتم‌ها و مدل‌های یادگیری ماشین بهتری را ایجاد کنند. اهمیت استخراج ویژگی‌ها به دلایل زیر است:

۱. کاهش ابعاد داده: استخراج ویژگی‌ها می‌تواند به کاهش ابعاد داده‌ها کمک کند، به خصوص در صورتی که داده‌ها دارای تعداد زیادی ویژگی باشند. این کاهش ابعاد می‌تواند به بهبود سرعت آموزش مدل و کاهش پیچیدگی مسئله کمک کند.

۲. زیبایی مدل: ویژگی‌های مناسب و ارتباطی با مسئله می‌توانند به دقت و قدرت مدل کمک کنند. با انتخاب ویژگی‌های مناسب، می‌توان مدل را به شکلی زیبا و کارا آموزش داد.

۳. افزایش قابلیت تفسیر: ویژگی‌های استخراج شده معمولاً قابلیت تفسیر بالایی دارند که به تحلیل و تفسیر نتایج کمک می‌کند و درک بهتری از داده‌ها فراهم می‌کند.

۴. تقویت عملکرد مدل: ویژگی‌های مناسب می‌توانند عملکرد مدل را بهبود بخشند، به خصوص در صورتی که داده‌ها دارای نویز یا اطلاعات ناخواسته باشند.

۵. اهمیت تحلیل داده: استخراج ویژگی‌ها می‌تواند به محققان و متخصصان کمک کند تا به اطلاعات مفیدی درباره داده‌ها دست یابند و الگوهای جدیدی را شناسایی کنند.

حالا با استفاده از روش‌های استخراج ویژگی مطرح شده در جدول ۱، ۸ ویژگی را از دیتاست خود استخراج کرده و یک دیتاست جدید ایجاد می‌کنیم. برای این کار، به عنوان مثال، می‌توانید از روش‌های مختلفی مانند Standard Deviation، Peak، Skewness و ... استفاده کنید. انتخاب روش‌های مناسب بستگی به نوع داده و مسئله مورد نظر دارد.

بخش دوم قسمت ج : مخلوط کردن داده‌ها یک مرحله مهم در فرآیند پیش‌پردازش داده است که قبل از آموزش مدل‌های یادگیری ماشین انجام می‌شود. این فرآیند اهمیت زیادی دارد زیرا می‌تواند بهبود عملکرد مدل‌ها، جلوگیری از برازش بیش از حد ((overfittingو افزایش قابلیت تعمیم ((generalizationکمک کند.

۱. مخلوط کردن داده‌ها: در این مرحله، داده‌های موجود از منابع مختلف (مثلاً دیتاست‌های مختلف یا بخش‌های مختلف از یک دیتاست) با هم ترکیب می‌شوند تا یک دیتاست جدید و یکنواخت ایجاد شود. این کار می‌تواند به افزایش تنوع داده‌ها، کاهش تاثیر نویز و تعمیم بهتر مدل‌ها کمک کند.

۲. تقسیم داده: پس از مخلوط کردن داده‌ها، داده‌های حاصل را به دو بخش آموزش و آزمون تقسیم می‌کنیم. بخش آموزش برای آموزش مدل‌ها استفاده می‌شود و بخش آزمون برای ارزیابی عملکرد مدل‌ها استفاده می‌شود. نسبت تقسیم داده می‌تواند بستگی به حجم داده و مسئله مورد نظر داشته باشد.

با توجه به اهمیت فرآیند مخلوط کردن داده و تقسیم آن، انجام این مراحل با دقت و درستی می‌تواند به بهبود عملکرد مدل‌های یادگیری ماشین کمک کند و پیش‌پردازش مناسبی برای داده‌ها فراهم کند.

بخش دوم قسمت د : نرمال‌سازی داده‌ها یک فرآیند مهم در پیش‌پردازش داده است که هدف آن استانداردسازی واحدهای داده‌ها است. این فرآیند می‌تواند بهبودی در عملکرد الگوریتم‌های یادگیری ماشین داشته باشد و مواردی مانند افزایش سرعت آموزش مدل، جلوگیری از برازش بیش از حد ((overfittingو افزایش دقت مدل را ایجاد کند. در زیر دو روش نرمال‌سازی رایج را توضیح می‌دهم:

۱. Min-Max Scaling : در این روش، ویژگی‌های داده را به یک بازه خاص تبدیل می‌کنیم، معمولاً به بازه [0, 1] یا [-1, 1]. فرمول استفاده شده برای نرمال‌سازی به صورت زیر است:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \]]\
```
```

اهمیت این روش این است که از تغییرات بیش از حد در داده‌ها جلوگیری می‌کند و تمام ویژگی‌ها به یک مقیاس مشابه تبدیل می‌شوند.

۲. Standardization (Z-score Normalization) : در این روش، میانگین داده‌ها صفر و انحراف معیار آن‌ها یک می‌شود. فرمول استفاده شده برای نرمال‌سازی به صورت زیر است:

\[X_{\text{scaled}} = \frac{X - \mu}{\sigma} \]]\

این روش از تغییرات در ویژگی‌ها برای تفسیر اطلاعات استفاده می‌کند و منجر به تمرکز بیشتر داده‌ها در اطراف میانگین می‌شود.

با توجه به اینکه Standardization (Z-score Normalization) بیشتر در الگوریتم‌هایی که به فرض توزیع نرمال داده‌ها هستند موثر است، برای فرآیند نرمال‌سازی از این روش استفاده می‌کنیم. به عنوان مثال، در الگوریتم‌هایی مانند ماشین‌های بردار پشتیبانی (SVM) و عمیق ترین شبکه‌های عصبی (Deep Neural Networks) که بر اساس توزیع نرمال داده‌ها عمل می‌کنند، استانداردسازی معمولاً بهبودی در عملکرد مدل دارد.

3- در اینجا یک کد نمونه برای ساخت یک مدل طبقه‌بندی ساده بدون استفاده از کتابخانه‌های آماده پایتون را ارائه می‌دهم. در این مثال، از الگوریتم ماشین بردار پشتیبانی (SVM) به عنوان مدل طبقه‌بند استفاده می‌کنیم، تابع اتلاف (Loss function) را به عنوان تابع هزینه خطای طبقه‌بندی استفاده می‌کنیم و به عنوان الگوریتم یادگیری از روش گرادیان کاهشی (Gradient Descent) استفاده می‌کنیم.

python```

import numpy as np

تابع اتلاف (Loss function)

def hinge_loss(y_true, y_pred):

return np.maximum(0, 1 - y_true * y_pred)

```
# الگوریتم یادگیری (Gradient Descent)
def gradient_descent(X_train, y_train, learning_rate=0.01, epochs=100):
    n_samples, n_features = X_train.shape
    weights = np.zeros(n_features)
    bias = 0
    for epoch in range(epochs):
        for i in range(n_samples):
            condition = y_train[i] * (np.dot(X_train[i], weights) + bias) >= 1
            if condition:
                weights -= learning_rate * (2 * 1/epochs * weights)
            else:
                weights -= learning_rate * (2 * 1/epochs * weights - np.dot(X_train[i], y_train[i]))
                bias -= learning_rate * y_train[i]
    return weights, bias

# ارزیابی مدل
def evaluate_model(X_test, y_test, weights, bias):
    predictions = np.dot(X_test, weights) + bias
    accuracy = np.mean(np.sign(predictions) == y_test)
    return accuracy

# داده‌ها
X_train = np.array([[1, 2], [2, 3], [3, 4], [4, 5]])
y_train = np.array([-1, -1, 1, 1])
X_test = np.array([[5, 6], [6, 7]])
y_test = np.array([1, 1])
```

آموزش مدل

weights, bias = gradient_descent(X_train, y_train)

ارزیابی مدل

accuracy = evaluate_model(X_test, y_test, weights, bias)

print("Accuracy:", accuracy)

```

در این کد، تابع اتلاف (Loss function) از نوع تابع هزینه خطای طبقه‌بندی برای مدل SVM استفاده می‌شود. سپس با استفاده از الگوریتم گرادیان کاهشی (Gradient Descent)، مدل طبقه‌بند آموزش داده می‌شود. در نهایت، با استفاده از داده‌های تست، عملکرد مدل با دقت محاسبه می‌شود.

برای تحلیل نمودار تابع اتلاف، می‌توانیم مشاهده کنیم که با پیشرفت آموزش مدل، تابع اتلاف کاهش می‌یابد و در نهایت به مقدار مینیمم می‌رسد. اما بر اساس نمودار تابع اتلاف قبل از مرحله ارزیابی، نمی‌توانیم با قطعیت در مورد عملکرد مدل نظر دهیم. این تنها نشان‌دهنده آموزش مدل است و نه عملکرد آن بر روی داده‌های تست. برای ارزیابی دقیق مدل، نیاز به استفاده از داده‌های تست و محاسبه معیارهای ارزیابی مانند دقت (accuracy) و ماتری

4- بله، در scikit-learn می‌توان از ویژگی `loss_curve_` در طبقه‌بند خطی (`LinearSVC`یا `SGDClassifier`) برای نمایش نمودار تابع اتلاف (Loss function) استفاده کرد. دستورات زیر یک مدل `LinearSVC` را آموزش می‌دهند، نمودار تابع اتلاف را رسم می‌کنند و سپس مدل را بر روی داده‌های تست ارزیابی می‌کنند:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
```
```

```
# ساخت داده‌ها
X, y = make_classification(n_samples=1000, n_features=2, n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# نرمال‌سازی داده‌ها
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# آموزش مدل
model = LinearSVC(loss='hinge')
model.fit(X_train_scaled, y_train)

# رسم نمودار تابع اتلاف
plt.plot(model.loss_curve_)
plt.title('Loss Curve')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()

# ارزیابی مدل
accuracy = model.score(X_test_scaled, y_test)
print("Accuracy:", accuracy)
```

در این کد، داده‌ها با استفاده از `make_classification` تولید شده و سپس به دو بخش آموزش و تست تقسیم می‌شوند. سپس داده‌ها نرمال‌سازی می‌شوند و مدل `LinearSVC` با استفاده از داده‌های آموزش آموزش داده می‌شود. سپس نمودار تابع اتلاف (Loss Curve) با استفاده از ویژگی `loss_curve_` رسم شده و در نهایت مدل با داده‌های تست ارزیابی می‌شود.

5- Orange یک نرم‌افزار داده‌کاوی و تحلیل داده است که ابزارها و ویژگی‌های متنوعی برای انجام وظایف مختلف داده‌کاوی فراهم می‌کند. این ابزار قابلیت‌های گوناگونی از جمله بارگذاری، پیش‌پردازش، تجزیه و تحلیل داده، مدل‌سازی و ارزیابی مدل‌ها را فراهم می‌کند.

برخی از ویژگی‌ها و قابلیت‌های Orange عبارتند از:

1. بارگذاری و مدیریت داده: این ابزار قابلیت بارگذاری و مدیریت داده‌ها از فایل‌های مختلف را دارد، شامل فرمت‌های CSV، Excel، SQLite و ... می‌شود.

2. پیش‌پردازش داده: Orange ابزارهایی برای پیش‌پردازش داده‌ها ارائه می‌کند که شامل تبدیل و تغییر فرمت داده، پاکسازی داده، انتخاب ویژگی‌ها و ...

3. تجزیه و تحلیل داده: این ابزار قابلیت تجزیه و تحلیل داده‌ها با استفاده از روش‌های مختلف داده‌کاوی و ماشین‌های یادگیری را فراهم می‌کند.

4. مدل‌سازی و ارزیابی: با استفاده از Orange می‌توان مدل‌های یادگیری ماشینی را ایجاد کرده و ارزیابی کرد، همچنین می‌توان از ابزارهای بصری برای نمایش و تفسیر نتایج استفاده کرد.

برای نمونه، یک مثال ساده از استفاده از Orange می‌تواند شامل بارگذاری یک مجموعه داده، پیش‌پردازش آن (مانند پاکسازی داده‌های نامرتب، تبدیل ویژگی‌ها و ...)، ساخت و ارزیابی یک مدل یادگیری ماشینی (مانند رگرسیون خطی یا SVM) و نمایش نتایج ارزیابی به صورت گرافیکی با استفاده از ابزارهای بصری موجود در Orange باشد.

برای اطلاعات بیشتر در مورد استفاده از ویژگی‌ها و قابلیت‌های Orange و نمونه‌های بیشتر، می‌توانید به مستندات و منابع آموزشی آن مراجعه کنید.