



گزارش تمرین سوم

آرین محمدخانی- 810603136



هوش مصنوعی

# **مقدمه**

در این گزارش، مراحل انجام‌شده برای حل تمرین سوم که هدف آن دسته‌بندی نمونه‌های داده‌شده بر اساس خراب بودن یا نبودن ماشین‌های فرزکاری (Milling) است، توضیح داده شده است. در تمامی بخش‌ها، کدهای پیاده‌سازی به همراه توضیح پارامترهای مربوطه ارائه شده و خلاصه‌ای از نتایج حاصل نیز بیان شده است. برای نمایش بهتر نتایج نهایی، از قالب گزارش در Jupyter Notebook استفاده شده است. همچنین این فایل داخل سایت github به آدرس زیر آپلود شده است.

# **کتابخانه‌ها)**

در ابتدای این تمرین کتابخانه‌های مورد نیاز از جمله pandas، NumPy، matplotlib، sklearn، imblearn و seaborn برای انجام بخش های مختلف Import شده‌اند.

کد 1

from pandas import \*

from numpy import \*

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import  LabelEncoder, StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix

from imblearn.over\_sampling import SMOTE

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.multiclass import OneVsRestClassifier

# **الف -1)**

در کد ارائه‌شده ابتدا فایل داده‌ای با نام milling\_machine.csvرا با استفاده از تابع بارگذاری شده و آن در یک DataFrame با نام data\_frame ذخیره می‌شود. سپس با فراخوانی متد info()، خلاصه‌ای از ساختار داده‌ها شامل تعداد سطرها و ستون‌ها، نوع داده‌ی هر ستون، تعداد مقادیر غیر‌خالی و میزان حافظه‌ی اشغال‌شده نمایش داده می‌شود.

کد 2

data\_frame = read\_csv('milling\_machine.csv')

data\_frame.info()

data\_frame.describe()

|  |  |
| --- | --- |
|  |  |

خروجی 1

# **الف-2)**

این کد با هدف شناسایی داده‌های گمشده در مجموعه داده اجرا می‌شود. ابتدا با استفاده از متدisnull().sum() تعداد مقادیر گمشده در هر ستون محاسبه می‌شود و در متغیری به نام missing\_v ذخیره می‌گردد. سپس نسبت مقادیر گمشده به کل ردیف‌ها برای هر ستون باisnull().mean() محاسبه شده و در متغیر missing\_v\_ratio نگهداری می‌شود. در ادامه، یک DataFrame جدید به نام missing\_data ایجاد می‌شود که شامل دو ستون است: یکی برای تعداد مقادیر گمشده (Missing value Count) و دیگری برای نسبت آن‌ها (Missing value Ratio).

کد 3

missing\_v = data\_frame.isnull().sum()

missing\_v\_ratio = data\_frame.isnull().mean()

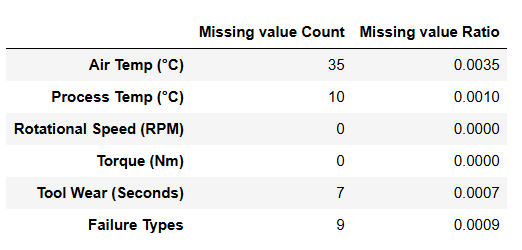
missing\_data = DataFrame({

'Missing value Count': missing\_v,

'Missing value Ratio': missing\_v\_ratio

})

print(missing\_data)



خروجی 2

# **الف-3)**

در ابتدا، یک کپی از مجموعه داده اصلی به نام [[1]](#footnote-1)new\_dataساخته می‌شود تا تغییرات روی نسخه اصلی اعمال نشود. سپس با استفاده از LabelEncoder، تمام ستون‌هایی که نوع داده‌ی آن‌ها object (یعنی متنی یا دسته‌ای) است، به مقادیر عددی تبدیل می‌شوند. در ادامه، با استفاده از تابع corr()، ماتریس همبستگی بین همه‌ی ویژگی‌های عددی محاسبه می‌شود. سپس مقدار همبستگی هر ویژگی با ستون Failure Types استخراج شده و بر اساس میزان همبستگی به صورت نزولی مرتب می‌شود. این لیست می‌تواند در شناسایی ویژگی‌های تاثیرگذار بر نوع خرابی ابزار مفید باشد. در نهایت، با استفاده از seaborn.heatmap، کل ماتریس همبستگی به صورت گرافیکی نمایش داده می‌شود تا روابط بین ویژگی‌ها به شکل بصری قابل تحلیل باشد.

کد 4

new\_data = data\_frame.copy()

label\_encoder = LabelEncoder()

for column in new\_data.select\_dtypes(include='object').columns:

new\_data[column] = label\_encoder.fit\_transform(new\_data[column])

correlation\_matrix = new\_data.corr(numeric\_only=True)

correlation\_with\_failure = correlation\_matrix['Failure Types']

sorted\_corr = correlation\_with\_failure.sort\_values(ascending=False)

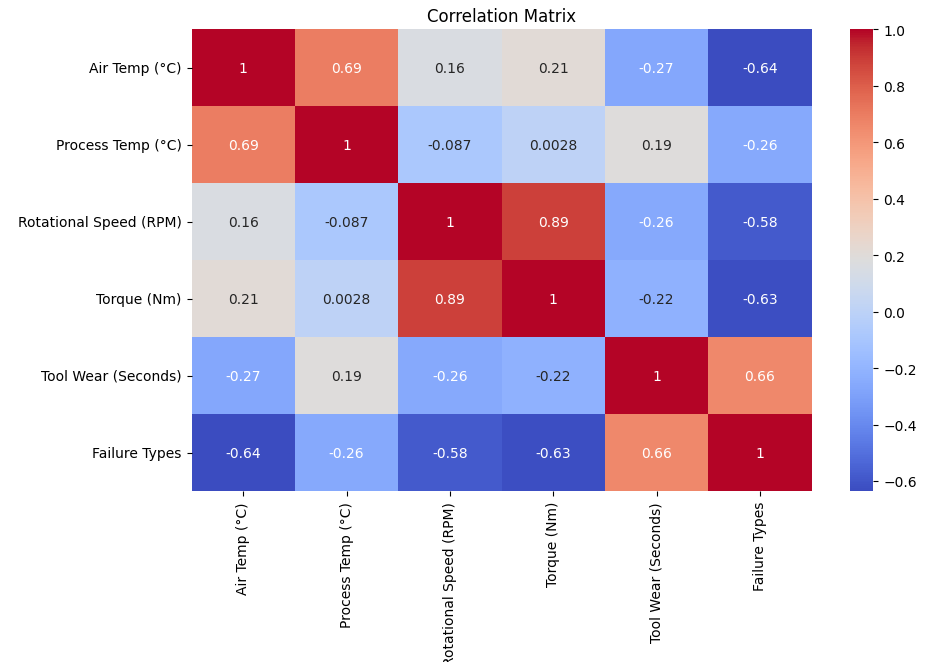
print(sorted\_corr)

plt.figure(figsize=(10, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()



خروجی 3

# **الف-4)**

در این بخش از کد، تمرکز بر شناسایی و تحلیل ویژگی‌هایی است که بیشترین همبستگی را با متغیر هدف Failure Types دارند. ابتدا با استفاده از .drop('Failure Types')، ستون مربوط به خود متغیر هدف از لیست همبستگی‌ها حذف می‌شود. سپس با استفاده از .abs().nlargest(3)، سه ویژگی با بیشترین مقدار مطلق همبستگی انتخاب می‌شوند، چرا که همبستگی منفی نیز می‌تواند نشان‌دهنده‌ی تأثیر قوی باشد. در ادامه، برای هر یک از این سه ویژگی برتر، یک نمودار هیستوگرام (histogram) ترسیم می‌شود که توزیع مقادیر آن ویژگی را در داده‌ها نشان می‌دهد.

در این کد، برای هر یک از سه ویژگی برتر، یک هیستوگرام با استفاده از تابع sns.histplot ترسیم می‌شود. پارامتر bins=10 داده‌ها را به ۱۰ بازه‌ی مساوی تقسیم می‌کند تا توزیع مقادیر بهتر قابل مشاهده باشد. همچنین با تنظیم kde=False، از نمایش منحنی چگالی احتمال جلوگیری شده و تنها ستون‌های هیستوگرام نمایش داده می‌شود.

کد 5

top\_features = sorted\_corr.drop('Failure Types').abs().nlargest(3).index

for feature in top\_features:

    plt.figure(figsize=(8, 4))

    sns.histplot(data=data\_frame, x=feature, bins=10, kde=False)

    plt.title(f'Distribution of {feature} (Binned)')

    plt.xlabel(feature)

    plt.ylabel('Count')

    plt.tight\_layout()

    plt.show()

|  |  |
| --- | --- |
|  |  |
|  | |

خروجی 4

# **ب-1)**

در این قسمت ، تعداد مقادیر گم‌شده در هر ستون شمارش می‌شود. در صورتی که تعداد مقادیر گم‌شده بیشتر از صفر باشد، برای ستون‌های عددی (float64)، میانگین مقادیر هر گروه از "Failure Types" برای پر کردن مقادیر گم‌شده استفاده می‌شود. برای ستون‌های متنی(object)، مقادیر گم‌شده با رشته 'Unknown' جایگزین می‌شوند تا از حذف سطرها جلوگیری شود. تغییرات در یک DataFrame جدید به نام data\_filled ذخیره می‌شود. در این کد، از توابع isnull(), sum(), groupby(), transform(), و fillna() استفاده شده است تا مقادیر گم‌شده مدیریت شوند.

ابتدا یک شیء از LabelEncoder ایجاد می‌شود. سپس برای هر ستون در DataFrame که نوع داده آن object است، از متدfit\_transform برای تبدیل مقادیر متنی به مقادیر عددی استفاده می‌شود. این تبدیل به این صورت انجام می‌شود که هر مقدار متنی یک عدد منحصر به فرد دریافت می‌کند که نشان‌دهنده آن مقدار است. در نهایت، DataFrame اصلی (data\_frame) با مقادیر عددی به‌روز می‌شود. در این کد از متدهای select\_dtypes() برای انتخاب ستون‌های متنی وfit\_transform() برای تبدیل استفاده شده است.

کد 6

for column in data\_frame.columns:

missing\_value = data\_frame[column].isnull().sum()

if missing\_value > 0:

if data\_frame[column].dtype == 'float64' :

if column != 'Failure Types':

data\_frame[column]=data\_frame.groupby('FailureTypes')[column].transform(lambdax:x.fillna(x.mean()))

elif data\_frame[column].dtype == 'object' :

data\_frame.dropna(subset=[column], inplace=True)

# **ب-2)**

در پیش‌پردازش داده‌ها برای یادگیری ماشین، دو روش رایج برای مقیاس‌بندی ویژگی‌های عددی وجود دارد: استانداردسازی (Standardization) و نرمال‌سازی (Normalization).

**استانداردسازی:**  
استانداردسازی به معنای تبدیل مقادیر ویژگی‌ها به توزیعی با میانگین صفر و انحراف معیار یک است. این کار باعث می‌شود که ویژگی‌ها مقیاسی یکسان داشته باشند و از پراکندگی بیش از حد جلوگیری شود. فرمول استانداردسازی به صورت زیر است:

|  |  |
| --- | --- |
| معادله 1 |  |

در این فرمول:

* μمیانگین مقادیر ویژگی
* σانحراف معیار ویژگی

استانداردسازی زمانی مفید است که داده‌ها دارای پراکندگی زیاد باشند یا الگوریتم‌های مورد استفاده به مقیاس داده‌ها حساس باشند مانند SVM ، KNN و Logistic Regression.

**نرمال‌سازی:**  
نرمال‌سازی به معنای مقیاس‌بندی ویژگی‌ها به یک بازه مشخص، معمولاً بین ۰ تا ۱ است. این کار معمولاً زمانی انجام می‌شود که مقیاس مطلق داده‌ها مهم باشد یا داده‌ها دارای مقادیر بسیار متنوع و خارج از محدوده باشند. روش رایج نرمال‌سازی به صورت زیر است:

|  |  |
| --- | --- |
| معادله 2 |  |

نرمال‌سازی بیشتر در مواردی مانند پردازش تصویر یا هنگام استفاده از شبکه‌های عصبی کاربرد دارد که نیاز به ورودی در یک محدوده مشخص است.

در این تمرین که با هدف دسته‌بندی داده‌ها با استفاده از الگوریتم‌هایی مانند SVM،KNN و Logistic Regression انجام شده است. برای اینکار ابتدا، یک لیست از ستون‌های عددی (numeric\_cols) تعریف می‌شود که شامل ویژگی‌هایی مانند دما، سرعت چرخشی، گشتاور و زمان فرسایش ابزار است. سپس، یک شیء از کلاس StandardScaler ایجاد می‌شود که برای مقیاس‌بندی داده‌ها به کار می‌رود. با استفاده از متد fit\_transform، داده‌های هر یک از ستون‌های عددی به گونه‌ای مقیاس‌بندی می‌شوند که میانگین آن‌ها صفر و انحراف معیار آن‌ها یک شود. پس از انجام مقیاس‌بندی، مقادیر جدید به همان ستون‌ها در DataFrame اعمال می‌شود.

کد 7

numeric\_cols = ['Air Temp (°C)', 'Process Temp (°C)', 'Rotational Speed (RPM)', 'Torque (Nm)', 'Tool Wear (Seconds)']

scaler = StandardScaler()

data\_frame[numeric\_cols] = scaler.fit\_transform(data\_frame[numeric\_cols])

# ج-1-2)

در این قسمت، یک ستون جدید به نام Failure\_Binary به DataFrame اضافه می‌شود که بر اساس مقادیر موجود در ستون Failure Types ایجاد می‌شود. در این عملیات، برای هر مقدار در ستون Failure Types، از متد apply و یک تابع lambda استفاده می‌شود. اگر مقدار ستون Failure Types صفر نباشد، مقدار ‘Failure’ به ستون جدید اختصاص داده می‌شود و در غیر این صورت، مقدار’No Failure’ به آن نسبت داده می‌شود. سپس، با استفاده از متد value\_counts()، تعداد وقوع هر یک از مقادیر ‘Failure’ و ‘No Failure’ در ستون Failure\_Binary محاسبه می‌شود. این کد به این صورت عمل می‌کند که وضعیت شکست را به‌صورت دودویی در DataFrame نشان می‌دهد و تعداد هر دسته را بررسی می‌کند.

کد 8

data\_frame['Failure\_Binary'] = data\_frame['Failure Types'].apply(lambda x: 'Failure' if x != 0 else 'No Failure')

data\_frame['Failure\_Binary'].value\_counts()

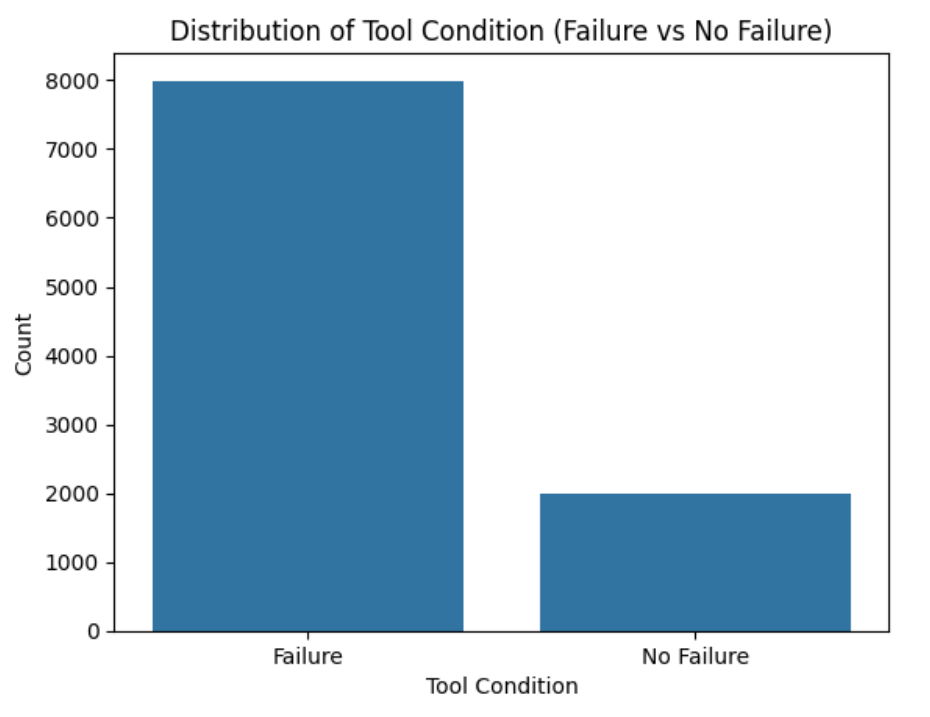
sns.countplot(data=data\_frame, x='Failure\_Binary')

plt.title('Distribution of Tool Condition (Failure vs No Failure)')

plt.xlabel('Tool Condition')

plt.ylabel('Count')

plt.show()



خروجی 5

# ج-3)

عدم توازن داده‌ها (Imbalanced Data) یکی از چالش‌های مهم در مسائل یادگیری ماشین به‌ویژه در دسته‌بندی (Classification) است. این مشکل زمانی به‌وجود می‌آید که تعداد نمونه‌های متعلق به یک یا چند کلاس به‌مراتب کمتر از کلاس‌های دیگر باشد؛ برای مثال، در این تمرین، داده‌های مربوط به حالت "خراب" بسیار بیشتر از داده‌های "سالم" می‌باشند. در چنین شرایطی، مدل یاد می‌گیرد که بیشتر نمونه‌ها را به کلاس غالب نسبت دهد، چون این کار از نظر آماری دقت ظاهری بالایی به مدل می‌دهد. اما در واقع، توان مدل در شناسایی کلاس‌های اقلیت که ممکن است از نظر کاربردی بسیار حیاتی باشند کاهش می‌یابد. این موضوع باعث می‌شود مقادیری مثل Recallو F1-Scoreبرای کلاس اقلیت کاهش یابد و در نهایت عملکرد کلی مدل در تصمیم‌گیری واقعی ضعیف شود. بنابراین، تشخیص و رفع عدم توازن داده‌ها، به‌ویژه در کاربردهای حساس، از اهمیت بالایی برخوردار است.

# ج-4-6)

در این قسمت، ابتدا داده‌ها به دو بخش ویژگی‌ها (X) و برچسب‌ها (y) تقسیم می‌شوند. ستون‌های 'Failure Types' و 'Failure\_Binary' از DataFrame حذف می‌شوند و ستون 'Failure\_Binary' به عنوان متغیر هدف (target) در نظر گرفته می‌شود. سپس، از LabelEncoder برای تبدیل مقادیر دسته‌بندی شده در y به مقادیر عددی استفاده می‌شود. پس از آن، از تکنیک SMOTE (Synthetic Minority Over-sampling Technique) برای برطرف کردن عدم توازن داده‌ها استفاده می‌شود.SMOTE برای ایجاد نمونه‌های مصنوعی از کلاس اقلیت در داده‌ها به کار می‌رود. پس از اعمالSMOTE، داده‌های بازنمونه‌گیری شده (x\_resampled و y\_resampled) ایجاد می‌شود. در مرحله بعد، تعداد نمونه‌ها در هر کلاس در داده‌های بازنمونه‌گیری شده با استفاده از unique و counts شمارش می‌شود و سپس با استفاده از inverse\_transform، برچسب‌های عددی به مقادیر اصلی (متنی) تبدیل می‌شوند و تعداد هر کلاس چاپ می‌شود. در نهایت، داده‌های بازنمونه‌گیری شده به دو مجموعه آموزشی و آزمایشی تقسیم می‌شوند (X\_train، X\_test، y\_trainو y\_test) با استفاده از train\_test\_split و 20% از داده‌ها به عنوان مجموعه آزمایشی انتخاب می‌شود.

کد 9

X = data\_frame.drop(columns=['Failure Types', 'Failure\_Binary'])

y = data\_frame['Failure\_Binary']

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y)

smote = SMOTE(random\_state=42)

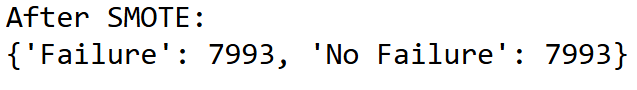
X\_resampled, y\_resampled = smote.fit\_resample(X, y\_encoded)

print("After SMOTE:")

uniques, counts = unique(y\_resampled, return\_counts=True)

print(dict(zip(label\_encoder.inverse\_transform(uniques), counts)))

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



خروجی 6

# ج-7-8)

در این قسمت، چهار مدل یادگیری ماشین برای دسته‌بندی دودویی وضعیت شکست ابزار آموزش داده می‌شوند. مدل‌ها شاملLogistic Regression،K-Nearest Neighbors با ۵ همسایه، Support Vector Machine (SVM) با هسته خطی و SVM با هسته RBF هستند که در یک دیکشنری با نام models تعریف شده‌اند.

در یک حلقه for، هر مدل روی داده‌های آموزشی (X\_train و y\_train) آموزش داده می‌شود. سپس پیش‌بینی‌ها روی مجموعه آزمایشی (X\_test) انجام می‌گیرد و عملکرد مدل از طریق معیارهایی مانند دقت (Accuracy)، ماتریس آشفتگی و گزارش دسته‌بندی ارزیابی می‌شود. گزارش دسته‌بندی به‌صورت دیکشنری استخراج شده و به طور خاص معیارهای precision،recall و F1-score برای کلاس شکست ذخیره می‌شود. نتایج هر مدل در قالب یک دیکشنری به لیست results افزوده می‌گردد.

کد 10

models = {

"Logistic Regression": LogisticRegression(max\_iter=1000),

"KNN": KNeighborsClassifier(n\_neighbors=5),

"SVM Linear": SVC(kernel='linear'),

"SVM RBF": SVC(kernel='rbf')

}

results = []

for name, model in models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

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

report = classification\_report(y\_test, y\_pred, output\_dict=True)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"\n{name}:\n")

print("Confusion Matrix:\n", conf\_matrix)

print("Accuracy:", acc)

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

results.append({

"Model": name,

"Accuracy": acc,

"Precision (Failure)": report['1']['precision'],

"Recall (Failure)": report['1']['recall'],

"F1-Score (Failure)": report['1']['f1-score']

})

|  |  |
| --- | --- |
|  |  |
|  |  |

خروجی 7

در این قسمت، از تکنیک Grid Search همراه با اعتبارسنجی متقاطع برای تنظیم بهینه‌ پارامترهای سه مدل مختلف استفاده شده است. برای Logistic Regression، شبکه‌ای از مقادیر پارامتر C (که معیاری برای تنظیم شدت منظم‌سازی است) تعریف شده و فقط از جریمهl2 استفاده می‌شود. مدل با استفاده از GridSearchCV با پنج برابر اعتبارسنجی متقاطع (cv=5) آموزش داده می‌شود و بهترین ترکیب پارامترها با توجه به معیار دقت (accuracy) چاپ می‌شود. برای KNN، پارامتر قابل تنظیم n\_neighbors در بازه 1 تا 20 بررسی می‌شود و بهترین مقدار K با استفاده از همان روش Grid Search مشخص می‌گردد. در بخش SVM، سه مقدار برای پارامتر C، سه مقدار برایgamma ، و تنها هسته 'rbf'مورد بررسی قرار گرفته‌اند. بهترین ترکیب از این پارامترها نیز پس از Grid Search گزارش می‌شود. و در انتها به کمک شاخص‌های معرفی شده در درس عملکرد مدل‎ها باهم مقایسه می‌شوند.

# Logistic Regression

param\_grid\_lr = {

    'C': [0.01, 0.1, 1, 10],

    'penalty': ['l2'],  }

grid\_lr = GridSearchCV(LogisticRegression(max\_iter=1000), param\_grid\_lr, cv=5, scoring='accuracy')

grid\_lr.fit(X\_train, y\_train)

print("Best Params (LR):", grid\_lr.best\_params\_)

#KNN

param\_grid\_knn = {'n\_neighbors': range(1, 21)}

grid\_knn = GridSearchCV(KNeighborsClassifier(), param\_grid\_knn, cv=5)

grid\_knn.fit(X\_train, y\_train)

print("Best K for KNN:", grid\_knn.best\_params\_)

#SVM linear

param\_grid\_svm\_linear = {

    'C': [0.01, 0.1, 1, 10, 100],

    'kernel': ['linear']}

grid\_svm\_linear = GridSearchCV(SVC(), param\_grid\_svm\_linear, cv=5, scoring='accuracy')

grid\_svm\_linear.fit(X\_train, y\_train)

print("Best parameters for SVM (Linear):", grid\_svm\_linear.best\_params\_)

#SVM RBF

param\_grid\_svm = {

    'C': [0.1, 1, 10],

    'gamma': [1, 0.1, 0.01],

    'kernel': ['rbf']}

grid\_svm = GridSearchCV(SVC(), param\_grid\_svm, cv=5)

grid\_svm.fit(X\_train, y\_train)

print("Best parameters for SVM (RBF):", grid\_svm.best\_params\_)

comparison\_table = DataFrame(results)

print(comparison\_table.sort\_values(by="Accuracy", ascending=False))

کد 11

|  |  |
| --- | --- |
|  | Best Params (LR): {'C': 0.01, 'penalty': 'l2'} |
| Best K for KNN: {'n\_neighbors': 1} |
| Best parameters for SVM (Linear): {'C': 0.01, 'kernel': 'linear' |
| Best parameters for SVM (RBF): {'C': 0.1, 'gamma': 1, 'kernel': 'rbf'} |

خروجی 8

# د-1-2)

در این قسمت، مسئله دسته‌بندی چندکلاسه برای ستون Failure Types پیاده‌سازی شده است. ابتدا ویژگی‌ها (X\_multi) با حذف ستون‌های Failure Types و Failure\_Binary از DataFrame اصلی استخراج می‌شوند و خود ستون Failure Types به عنوان متغیر هدف (y\_multi) در نظر گرفته می‌شود.

برای مقابله با عدم توازن بین کلاس‌ها، از روش SMOTE با مقدار random\_state=42 استفاده شده است تا نمونه‌های مصنوعی از کلاس‌های کم‌تعداد ایجاد شود. سپس داده‌های متوازن‌شده به دو مجموعه آموزشی و آزمایشی تقسیم می‌شوند.

چهار مدل یادگیری ماشین برای دسته‌بندی چندکلاسه مورد استفاده قرار گرفته‌اند:

* K-Nearest Neighbors (KNN)
* Decision Tree
* Random Forest
* SVM (One-vs-Rest and One-vs-one)

در حلقه‌ای که بر روی مدل‌ها اجرا می‌شود، هر مدل روی داده‌های آموزشی آموزش می‌بیند و بر روی داده‌های آزمون پیش‌بینی انجام می‌دهد. سپس معیارهای ارزیابی شامل دقت، ماتریس و گزارش دسته‌بندیمحاسبه و چاپ می‌شود. در نهایت، مقادیر حاصل از هر مدل شامل دقت و میانگین ماکرو precision، recall و F1-score در قالب یک دیکشنری در لیست results\_multiclass ذخیره می‌شوند تا امکان مقایسه عملکرد مدل‌های مختلف در مسئله دسته‌بندی چندکلاسه فراهم شود.

کد 12

X\_multi = data\_frame.drop(columns=['Failure Types', 'Failure\_Binary'])

y\_multi = data\_frame['Failure Types']

smote\_multi = SMOTE(random\_state=42)

X\_res\_multi, y\_res\_multi = smote\_multi.fit\_resample(X\_multi, y\_multi)

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

models\_multiclass = {

    "KNN": KNeighborsClassifier(n\_neighbors=5),

    "Decision Tree": DecisionTreeClassifier(random\_state=42),

    "Random Forest": RandomForestClassifier(random\_state=42),

    "SVM (One-vs-Rest)": OneVsRestClassifier(SVC(kernel='rbf')),}

results\_multiclass = []

for name, model in models\_multiclass.items():

    model.fit(X\_train\_m, y\_train\_m)

    y\_pred\_m = model.predict(X\_test\_m)

    acc = accuracy\_score(y\_test\_m, y\_pred\_m)

    conf\_matrix = confusion\_matrix(y\_test\_m, y\_pred\_m)

    report = classification\_report(y\_test\_m, y\_pred\_m, output\_dict=True)

    print(f"\n{name}:\n")

    print("Confusion Matrix:\n", conf\_matrix)

    print("Accuracy:", acc)

    print("Classification Report:\n", classification\_report(y\_test\_m, y\_pred\_m))

    results\_multiclass.append({

        "Model": name,

        "Accuracy": acc,

        "Macro Precision": report['macro avg']['precision'],

        "Macro Recall": report['macro avg']['recall'],

        "Macro F1": report['macro avg']['f1-score']

    })

|  |  |
| --- | --- |
|  |  |

خروجی 9، سایر خروجی ها بدلیل طولانی بودن در گزارش اعلام نتایج آورده شده است.

# د-3-4)

در این بخش از کد، دو هدف دنبال شده است: مقایسه عملکرد مدل‌های چندکلاسه و تنظیم بهینه پارامترهای آن‌ها با استفاده از Grid Search.

ابتدا، داده‌های ذخیره‌شده در لیست results\_multiclass به یک DataFrame تبدیل می‌شود و با استفاده از تابع sort\_values بر اساس معیار دقت (Accuracy) مرتب شده و نمایش داده می‌شود تا مدل با بالاترین عملکرد مشخص گردد.

سپس برای هر یک از مدل‌های KNN، درخت تصمیم، جنگل تصادفی و SVM با ساختار One-vs-Rest، تنظیم پارامتر با استفاده از GridSearchCV و اعتبارسنجی متقاطع پنج‌گانه انجام می‌شود:

* برای KNN، بهترین مقدار n\_neighbors در بازه 1 تا 20 جستجو می‌شود.
* برای درخت تصمیم، پارامترهای max\_depth و min\_samples\_split بررسی می‌شوند تا ساختار بهینه درخت پیدا شود.
* برای Random Forest، تعداد درخت‌ها (n\_estimators) و عمق آن‌ها (max\_depth) تنظیم می‌شود.
* برای SVM باOne-vs-Rest، پارامترهایC و gamma برای تخمین‌گر پایه (estimator) بررسی می‌شوند.

در نهایت، در مرحله بعد ، همان جدول نتایج اولیه دوباره چاپ و بر اساس دقت مرتب می‌شود. این مقایسه می‌تواند به بررسی تأثیر بهینه‌سازی پارامترها بر دقت مدل‌ها کمک کند.

کد 13

# KNN

param\_grid\_knn = {'n\_neighbors': range(1, 21)}

grid\_knn\_multi = GridSearchCV(KNeighborsClassifier(), param\_grid\_knn, cv=5)

grid\_knn\_multi.fit(X\_train\_m, y\_train\_m)

print("Best K for KNN (Multiclass):", grid\_knn\_multi.best\_params\_)

# Decision Tree

param\_grid\_dt = {'max\_depth': [3, 5, 10, None], 'min\_samples\_split': [2, 5, 10]}

grid\_dt = GridSearchCV(DecisionTreeClassifier(random\_state=42), param\_grid\_dt, cv=5)

grid\_dt.fit(X\_train\_m, y\_train\_m)

print("Best params for Decision Tree:", grid\_dt.best\_params\_)

# Random Forest

param\_grid\_rf = {'n\_estimators': [50, 100, 150], 'max\_depth': [None, 10, 20]}

grid\_rf = GridSearchCV(RandomForestClassifier(random\_state=42), param\_grid\_rf, cv=5)

grid\_rf.fit(X\_train\_m, y\_train\_m)

print("Best params for Random Forest:", grid\_rf.best\_params\_)

# SVM

param\_grid\_svm\_multi = {

    'estimator\_\_C': [0.1, 1, 10],

    'estimator\_\_gamma': [1, 0.1, 0.01]

}

grid\_svm\_multi = GridSearchCV(OneVsRestClassifier(SVC(kernel='rbf')), param\_grid\_svm\_multi, cv=5)

grid\_svm\_multi.fit(X\_train\_m, y\_train\_m)

print("Best params for SVM (One-vs-Rest):", grid\_svm\_multi.best\_params\_)

#d4

comparison\_multiclass = DataFrame(results\_multiclass)

print(comparison\_multiclass.sort\_values(by="Accuracy", ascending=False))

|  |
| --- |
| Best K for KNN (Multiclass): {'n\_neighbors': 4} |
| Best params for Decision Tree: {'max\_depth': 5, 'min\_samples\_split': 2} |
| Best params for Random Forest: {'max\_depth': None, 'n\_estimators': 50} |
| Best params for SVM (One-vs-Rest): {'estimator\_\_C': 10, 'estimator\_\_gamma': 0.1} |
| Best params for SVM (One-vs-One): {'estimator\_\_C': 0.1, 'estimator\_\_gamma': 1} |
|  |

خروجی 10

1. به این دلیل در این قسمت یک کپی ساخته شده است که در ادامه ابتدا missing valueهای دیتا فریم اصلی قبل از عدد گذاری حذف شده و پس از آن لیبل گذاری می شوند. [↑](#footnote-ref-1)