دانشکده مهندسی برق

پروژه نهایی

دانشجوها:

نیلوفر ملا 40122903

محدثه علیرضایی طهرانی 40121123

استاد درس:

جناب آقای دکتر علیاری

لینک مخزن گیتهاب:

https://github.com/m24ath/Smart-System

لینک گوگل کولب:

https://colab.research.google.com/drive/1wRNlhatYIFIjMRaCVWneBHbcJYTO22fX?usp=sharing


بسم الله الرحمن الرحيم

عنوان پروژه

پیش‌بینی ریزش مشتریان مخابرات با یادگیری ماشین

(Telecom Customer Churn Prediction with Machine Learning)

فهرست مطالب

فصل 1- مقدمه

در بسیاری از کسب‌وکارهای مبتنی بر اشتراک، به‌ویژه در صنعت مخابرات، ریزش مشتری (Customer Churn) یکی از مهم‌ترین عوامل کاهش درآمد و افزایش هزینه‌های عملیاتی است. از آنجا که جذب یک مشتری جدید معمولاً هزینه‌برتر از نگهداشت مشتری فعلی است، توانایی پیش‌بینی احتمال ریزش و شناسایی مشتریان پرریسک می‌تواند به تصمیم‌گیری‌های بهینه در زمینه‌ی بازاریابی، ارائه‌ی مشوق‌ها و تخصیص منابع پشتیبانی منجر شود. در این پروژه، هدف طراحی و ارزیابی یک سامانه‌ی پیش‌بینی ریزش مشتری بر پایه‌ی داده‌های رفتاری و قراردادهای مشتریان است.

برای این منظور از دیتاست استاندارد Telco Customer Churn استفاده می‌شود که شامل ویژگی‌هایی از جنس عددی و دسته‌ای (مانند مدت زمان عضویت، هزینه‌های ماهانه و کل، نوع قرارداد، روش پرداخت و سرویس‌های فعال و یک برچسب دودویی به نام Churn ریزش/عدم ریزش است. به دلیل ماهیت واقعی داده‌ها، انجام پیش‌پردازش‌هایی نظیر مدیریت مقادیر نامعتبر یا گمشده، کدگذاری ویژگی‌های دسته‌ای، مقیاس‌بندی متغیرهای عددی و بررسی عدم‌توازن کلاس‌ها نقش کلیدی در کیفیت مدل نهایی دارد. بنابراین، پروژه صرفاً به آموزش یک مدل محدود نمی‌شود و بر ساخت یک خط لوله‌ی کامل یادگیری ماشین از مرحله‌ی آماده‌سازی داده تا تحلیل نتایج تمرکز دارد.

در ادامه، چندین سناریوی پیش‌پردازشی طراحی و اثر آن‌ها بر عملکرد مدل‌ها بررسی می‌شود. همچنین با توجه به افزایش تعداد ویژگی‌ها پس از تبدیل متغیرهای دسته‌ای به نمایش عددی مانند One-Hot Encoding ، تکنیک‌های کاهش بُعد از جمله PCA و/یا روش‌های مشابه به‌کار گرفته شده و اثر کاهش بُعد بر دقت پیش‌بینی و هزینه‌ی محاسباتی تحلیل می‌شود. در بخش مدل‌سازی، حداقل دو الگوریتم یادگیری برای مثال SVM و درخت تصمیم آموزش داده شده و با تنظیم هایپرپارامترها، بهترین پیکربندی هر مدل استخراج می‌گردد. ارزیابی نهایی با استفاده از معیارهایی نظیرPrecision ، Recall، F1-score و ROC-AUC و نیز نمودارهایی مانند Confusion Matrixو ROC Curve انجام می‌شود تا هم عملکرد پیش‌بینی و هم رفتار خطاها به‌صورت دقیق تفسیر شود. هدف نهایی، ارائه‌ی مدلی تکرارپذیر و قابل تحلیل است که بتواند مشتریان در معرض ریزش را با دقت مناسب شناسایی کند و مبنایی برای تصمیم‌گیری داده‌محور در مدیریت ارتباط با مشتری فراهم آورد.

فصل 2- هدف پروژه

— **هدف اصلی (Prediction)**

ساخت یک مدل طبقه‌بندی که با استفاده از اطلاعات مشتری (قرارداد، سرویس‌ها، پرداخت، هزینه‌ها و ...) پیش‌بینی کند آیا مشتری ریزش می‌کند (Churn=Yes) یا نه (یا احتمال ریزش را بدهد).

— **هدف تحلیلی (Understanding)**

مشخص کنیم کدام الگوها/ویژگی‌ها بیشترین ارتباط را با ریزش دارند (مثلا نوع قرارداد، مدت عضویت، هزینه‌ها، سرویس‌های جانبی).

— **هدف مهندسی (Pipeline & Report)**

پیاده‌سازی و مقایسه‌ی یک **خط لوله‌ی کامل** شامل:

- چند سناریوی پیش‌پردازش
- کاهش بُعد و بررسی اثرش روی دقت و هزینه محاسباتی
- آموزش حداقل دو مدل مثلاً SVM و درخت تصمیم
- Hyperparameter tuning
- ارزیابی استاندارد با معیارها و نمودارها + تکرارپذیری باSeed

فصل 3- Dataset

این دیتاست یک نمونه‌ی استاندارد از داده‌های اشتراکی یا مخابراتی است (به‌صورت رایج منتشرشده توسط IBM و همچنین نسخه‌ی قابل دانلود آن در Kaggle موجود است). هر ردیف نماینده‌ی یک مشتری است و هدف، پیش‌بینی ستون Churn است.

ساختار داده

- **واحد مشاهده:** یک مشتری
- **ستون هدف**Label :Churn (Yes/No)
- **ستون شناسه** customerID **:** معمولا در مدل حذف می‌شود
- **ویژگی‌های عددی مهم:** tenure مدت عضویت، MonthlyCharges، TotalCharges
- **ویژگی‌های دسته‌ای (Categorical) مهم:** Contract, PaymentMethod, InternetService, OnlineSecurity, TechSupport, PaperlessBilling, …

لینک دیتاست

https://drive.google.com/file/d/17MsXa86Pe778yn63Z9KDMd9qZfiwpLVQ/view?usp=sharing

که آن را از گیتهاب دانلود کردیم

فصل 4- بخش اول

۱. معرفی مسئله و داده‌ها:

- معرفی دقیق مجموعه داده انتخاب شده (منبع، تعداد ویژگی‌ها، تعداد نمونه‌ها، توزیع کلاس‌ها و ...).
- در صورت انتخاب «هوشمندسازی سیستم»، توصیف دقیق عملکرد سیستم و نحوه جمع‌آوری داده‌ها.

در این قسمت با استفاده کدهای لازم اطلاعات مربوط به دیتاست را نمایش می دهیم.

```
#1
!pip -q install gdown

import os
import io
import pandas as pd

GDRIVE_URL =
"https://drive.google.com/file/d/17MsXa86Pe778yn63Z9KDMd9qZfiwpLVQ/view?us
p=sharing"

CSV_PATH = "WA_Fn-UseC_-Telco-Customer-Churn.csv"

!gdown --fuzzy "{GDRIVE_URL}" -O "{CSV_PATH}"

df = pd.read_csv(CSV_PATH)

print("\n================= BASIC SHAPE =================")
print("Rows (samples):", df.shape[0])
print("Columns (all):", df.shape[1])

target_col = "Churn" if "Churn" in df.columns else None
if target_col is not None:
    print("Target column:", target_col)
    print("Features (excluding target):", df.shape[1] - 1)
else:
    print("Target column 'Churn' not found. Features (all columns):",
df.shape[1])

print("\n================= COLUMNS =================")
print(list(df.columns))

print("\n================= DATA.INFO() =================")
buf = io.StringIO()
```

```
df.info(buf=buf)
print(buf.getvalue())

print("\n================ MISSING VALUES (NaN) ================")
na_counts = df.isna().sum().sort_values(ascending=False)
na_nonzero = na_counts[na_counts > 0]
print(na_nonzero if len(na_nonzero) else "No NaN values found.")

print("\n================ BLANK/EMPTY STRINGS (OBJECT COLUMNS)
================")
obj_cols = df.select_dtypes(include=["object"]).columns
blank_counts = {}
for c in obj_cols:
    blank_counts[c] = df[c].astype(str).str.strip().eq("").sum()
blank_series = pd.Series(blank_counts).sort_values(ascending=False)
blank_nonzero = blank_series[blank_series > 0]
print(blank_nonzero if len(blank_nonzero) else "No blank strings found in
object columns.")

if target_col is not None:
    print("\n================ CLASS DISTRIBUTION (Churn)
================")
    class_counts = df[target_col].value_counts(dropna=False)
    class_percent = (class_counts / len(df) * 100).round(2)
    class_dist = pd.DataFrame({"count": class_counts, "percent":
class_percent})
    print(class_dist)

display(df.head())
```

خروجی

```
================ BASIC SHAPE ================
Rows (samples): 7043
Columns (all): 21
Target column: Churn
Features (excluding target): 20

================ COLUMNS ================
['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod',
'MonthlyCharges', 'TotalCharges', 'Churn']

================ DATA.INFO() ================
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB


================= MISSING VALUES (NaN) =================
No NaN values found.

================= BLANK/EMPTY STRINGS (OBJECT COLUMNS) =================
TotalCharges    11
dtype: int64

================= CLASS DISTRIBUTION (Churn) =================
       count  percent
Churn
No      5174    73.46
Yes     1869    26.54
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service
1	5575-GNVDE	Male	0	No	No	34	Yes	No
2	3668-QPYBK	Male	0	No	No	2	Yes	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service
4	9237-HQITU	Female	0	No	No	2	Yes	No

	customerID	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	StreamingTV
0	7590-VHVEG	DSL	No	...	No	No	No
1	5575-GNVDE	DSL	Yes	...	Yes	No	No
2	3668-QPYBK	DSL	Yes	...	No	No	No
3	7795-CFOCW	DSL	Yes	...	Yes	Yes	No
4	9237-HQITU	Fiber optic	No	...	No	No	No

	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
0	No	Month-to-month	Yes	Electronic check	29.85	29.85	No
1	No	One year	No	Mailed check	56.95	1889.5	No
2	No	Month-to-month	Yes	Mailed check	53.85	108.15	Yes
3	No	One year	No	Bank transfer (automatic)	42.30	1840.75	No
4	No	Month-to-month	Yes	Electronic check	70.70	151.65	Yes

ابعاد دیتاست

تعداد نمونه‌ها(Rows): 7043

تعداد ستون‌ها: 21

ستون هدف: Churn

تعداد ویژگی‌ها (بدون هدف): 20

نوع داده‌ها (Data Types) و پیامدها

طبق ()info:

float64: فقط 1 ستون MonthlyCharges

int64: 2 ستون tenure و SeniorCitizen

object: 18 ستون یعنی عمدتا متنی هستند

این دیتاست categorical-heavy است. پس برای اکثر مدل‌های کلاسیک مثلSVM ، Logistic Regressionحتماً باید:

Encoding انجام شود معمولا One-Hot. و برای مدل‌هایی مثل SVM باید Scaling هم انجام شود.

وضعیت داده‌های گمشده

خروجی نشان می‌دهد:

No NaN values found.

اما همزمان در بخشBlank Strings، TotalCharges 11 مقدار خالی وجود دارد. یعنی داده‌ی گمشده داریم، ولی به شکل " "ذخیره شده، نه NaN برای همین ()isna چیزی پیدا نکرده است.

از طرفی TotalChargesباید عددی باشد، ولی الان object است. دلیلش همین مقادیر خالی (و گاهی رشته‌ای بودن) است.

باید TotalChargesرا تبدیل به numeric کنیم (با 'errors='coerce تا خالی‌ها NaN شوند) بعد تصمیم بگیریم 11 ردیف را حذف کنیم یا ایمپوت کنیم مثلا با میانه یا رابطه‌ی tenure×MonthlyCharge

توزیع کلاس‌ها

خروجی:

No : 5174 (73.46%)

Yes: 1869 (26.54%)

این یک عدم‌توازن متوسط است (نه خیلی شدید، نه کاملا بالانس).

فصل 5- بخش دوم

۲. پیش‌پردازش داده‌ها (Data Preprocessing):

- اعمال روش‌های مختلف پیش‌پردازش (مانند مدیریت داده‌های گم‌شده، نرمال‌سازی، استانداردسازی، مدیریت داده‌های پرت و ...).
- **تحلیل اثر:** بررسی و نمایش تاثیر هر یک از روش‌های پیش‌پردازش بر روی توزیع داده‌ها و نتایج نهایی.

در این بخش، هدف ما ساختن یک پایپلاین پیش‌پردازش برای دیتاست ریزش مشتریان است و سپس مقایسه‌ی چند سناریوی مختلف پیش‌پردازش است. ابتدا ستون‌های غیرضروری مثل customerID حذف می‌شوند. سپس ستون TotalChargesکه باید عددی باشد، از حالت متنی خارج شده و مقادیر خالی آن به NaN تبدیل می‌گردد. طبق خروجی قبلی، تعداد این مقادیر بسیار کم است (حدود 11 رکورد)، بنابراین به‌جای Imputation، سطرهای دارای مقدار گمشده حذف می‌شوند تا کیفیت داده حفظ شود و تحلیل ساده‌تر و قابل دفاع‌تر باشد.

بعد از آن، چون دیتاست شامل ویژگی‌های دسته‌ای (غیرعددی) زیادی است، این ویژگی‌ها با روش One-Hot Encodingبه نمایش عددی بدون ایجاد ترتیب مصنوعی تبدیل می‌شوند. در نهایت برای ویژگی‌های عددی، چند روش مختلف مقیاس‌بندی را به عنوان سناریوهای متفاوت امتحان می‌کنیم و با ثابت نگه داشتن یک مدل ساده(Logistic Regression) ، اثر هر سناریو را روی معیارهای عملکرد-Accuracy, F1, ROC AUC, ... و بر توزیع داده ها مقایسه می‌کنیم. سناریوی برتر در مراحل بعدی پروژه استفاده خواهد شد.

```
#2
import os
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score

# ----------------------------
# Common preprocessing (ALL scenarios)
# ----------------------------
# Drop useless ID
if "customerID" in df.columns:
    df = df.drop(columns=["customerID"])
```

```
# Fix TotalCharges (blank -> NaN -> numeric)
df["TotalCharges"] = df["TotalCharges"].astype(str).str.strip()
df.loc[df["TotalCharges"].eq(""), "TotalCharges"] = np.nan
df["TotalCharges"] = pd.to_numeric(df["TotalCharges"], errors="coerce")

# Missing handling: drop rows with NaN (only 11 rows)
before_rows = len(df)
df = df.dropna().copy()
after_rows = len(df)

print("Missing handling:")
print("Rows before:", before_rows, "| Rows after:", after_rows, "|
Dropped:", before_rows - after_rows)

y = df["Churn"].map({"Yes": 1, "No": 0})
X = df.drop(columns=["Churn"])

num_cols = ["tenure", "MonthlyCharges", "TotalCharges"]

cat_cols = [c for c in X.columns if c not in num_cols]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

def print_categorical_distributions(X_df, categorical_columns):
    print("\n" + "="*70)
    print("Categorical distributions BEFORE any preprocessing (TRAIN)")
    print("="*70)
    for col in categorical_columns:
        vc = X_df[col].value_counts(dropna=False)
        vp = (vc / len(X_df) * 100).round(2)
        dist = pd.DataFrame({"count": vc, "percent": vp})
        print(f"\n--- {col} ---")
        print(dist.to_string())

print_categorical_distributions(X_train, cat_cols)
print("\nNumeric distribution BEFORE any scaling (TRAIN):")
print(X_train[num_cols].describe().T[["mean", "std", "min", "max"]])

# ----------------------------
#  Helper functions
# ----------------------------
def one_hot_align(train_df, test_df):
```

```
    tr = pd.get_dummies(train_df, drop_first=False)
    te = pd.get_dummies(test_df, drop_first=False)
    tr, te = tr.align(te, join="left", axis=1, fill_value=0)
    return tr, te

def iqr_clip(train_df, test_df, cols):
    tr = train_df.copy()
    te = test_df.copy()
    bounds = {}

    for c in cols:
        q1 = tr[c].quantile(0.25)
        q3 = tr[c].quantile(0.75)
        iqr = q3 - q1
        low = q1 - 1.5 * iqr
        high = q3 + 1.5 * iqr
        bounds[c] = (low, high)

        tr[c] = tr[c].clip(low, high)
        te[c] = te[c].clip(low, high)

    return tr, te, bounds

def scale_numeric(train_df, test_df, cols, method):
    tr = train_df.copy()
    te = test_df.copy()

    if method == "standard":
        scaler = StandardScaler()
    elif method == "minmax":
        scaler = MinMaxScaler()
    else:
        raise ValueError("method must be 'standard' or 'minmax'")

    tr[cols] = scaler.fit_transform(tr[cols])
    te[cols] = scaler.transform(te[cols])

    return tr, te

def evaluate_model(Xtr, Xte, ytr, yte):
    model = LogisticRegression(max_iter=3000, solver="liblinear",
random_state=42)
    model.fit(Xtr, ytr)

    pred = model.predict(Xte)
```

```
    prob = model.predict_proba(Xte)[:, 1]

    return {
        "accuracy": accuracy_score(yte, pred),
        "precision": precision_score(yte, pred, zero_division=0),
        "recall": recall_score(yte, pred, zero_division=0),
        "f1": f1_score(yte, pred, zero_division=0),
        "roc_auc": roc_auc_score(yte, prob),
    }

def run_scenario(name, scale_method, use_outliers):
    tr = X_train.copy()
    te = X_test.copy()

    # Outlier handling
    bounds = None
    if use_outliers:
        tr, te, bounds = iqr_clip(tr, te, num_cols)

    # Scaling
    tr, te = scale_numeric(tr, te, num_cols, scale_method)

    # One-hot encoding
    tr_enc, te_enc = one_hot_align(tr, te)

    # Distribution AFTER preprocessing (numeric only)
    num_dist = tr[num_cols].describe().T[["mean", "std", "min", "max"]]

    # One-hot distribution idea: show top 10 most frequent binary features
    cat_features = [c for c in tr_enc.columns if c not in num_cols]
    top10 =
(tr_enc[cat_features].mean().sort_values(ascending=False).head(10) * 100)

    metrics = evaluate_model(tr_enc, te_enc, y_train, y_test)

    print("\n" + "="*65)
    print("Scenario:", name)
    if bounds is not None:
        print("\nOutlier bounds (computed on TRAIN, IQR method):")
        for k, (lo, hi) in bounds.items():
            print(f"{k}: low={lo:.3f}, high={hi:.3f}")

    print("\nNumeric distribution AFTER preprocessing (TRAIN):")
    print(num_dist)
```

```
    print("\nTop 10 most frequent one-hot features (TRAIN) [%]:")
    print(top10.round(2).to_string())

    print("\nModel performance (TEST):")
    print(pd.DataFrame([metrics]).to_string(index=False))

    return {"scenario": name, **metrics, "n_features_after_onehot":
tr_enc.shape[1]}
results = []
results.append(run_scenario("S1) Standardization ",
scale_method="standard", use_outliers=False))
results.append(run_scenario("S2) Normalization ", scale_method="minmax",
use_outliers=False))
results.append(run_scenario("S3) Outlier(IQR) + Standardization",
scale_method="standard", use_outliers=True))
results.append(run_scenario("S4) Outlier(IQR) + Normalization",
scale_method="minmax", use_outliers=True))

final_df = pd.DataFrame(results).sort_values("roc_auc", ascending=False)
print("\n" + "="*65)
print("FINAL COMPARISON (sorted by ROC-AUC):")
print(final_df.to_string(index=False))
```

ما دیتاست Telco را وارد کردیم، ستون بی‌ارزش customerID را حذف کردیم، ستون TotalCharges را که بعضی جاها به‌صورت متن یا خالی بود به عدد تبدیل کردیم، سپس اگر داده گمشده‌ای باقی می‌ماند ردیف‌های دارای مقدار خالی را حذف می‌کردیم. بعد داده‌ها را به train و test تقسیم کردیم. قبل از هر پیش‌پردازش، توزیع ستون‌های غیرعددی را با تعداد و درصد هر دسته چاپ کردیم و همچنین آمار کلی ستون‌های عددی را نشان دادیم. سپس 4 سناریوی پیش‌پردازش ساختیم: استانداردسازی، نرمال‌سازی، مدیریت پرت با روش IQR همراه با استانداردسازی، و مدیریت پرت با روش IQR همراه با نرمال‌سازی. در هر سناریو، ستون‌های غیرعددی را با One-Hot به عددی تبدیل کردیم، توزیع عددی بعد از پیش‌پردازش و پرتکرارترین ستون‌های One-Hot را چاپ کردیم و در نهایت با یک مدل ثابت Logistic Regression عملکرد هر سناریو را روی داده test مقایسه کردیم.

تبدیل TotalCharges به عدد

```
df["TotalCharges"] = df["TotalCharges"].astype(str).str.strip()
df.loc[df["TotalCharges"].eq(""), "TotalCharges"] = np.nan
df["TotalCharges"] = pd.to_numeric(df["TotalCharges"], errors="coerce")
```

اینجا فضاهای خالی به NaN تبدیل می‌شوند و بعد کل ستون عددی می‌شود.

تبدیل SeniorCitizen به دسته‌ای

```
df["SeniorCitizen"] = df["SeniorCitizen"].astype(str)
```

زیرا مفهومش دسته‌ای است، نه یک مقدار عددی پیوسته.

خروجی

```
Missing handling:
Rows before: 7032 | Rows after: 7032 | Dropped: 0

======================================================================
Categorical distributions BEFORE any preprocessing (TRAIN)
======================================================================

--- gender ---
        count  percent
gender
Male     2823    50.19
Female   2802    49.81

--- SeniorCitizen ---
               count  percent
SeniorCitizen
0               4715    83.82
1                910    16.18

--- Partner ---
         count  percent
Partner
No        2891     51.4
Yes       2734     48.6

--- Dependents ---
            count  percent
Dependents
No           3940    70.04
Yes          1685    29.96

--- PhoneService ---
              count  percent
PhoneService
Yes            5080    90.31
```

```
No              545     9.69

--- MultipleLines ---
                 count  percent
MultipleLines
No                2695    47.91
Yes               2385    42.40
No phone service   545     9.69

--- InternetService ---
                 count  percent
InternetService
Fiber optic       2483    44.14
DSL               1934    34.38
No                1208    21.48

--- OnlineSecurity ---
                     count  percent
OnlineSecurity
No                    2803    49.83
Yes                   1614    28.69
No internet service   1208    21.48

--- OnlineBackup ---
                     count  percent
OnlineBackup
No                    2446    43.48
Yes                   1971    35.04
No internet service   1208    21.48

--- DeviceProtection ---
                     count  percent
DeviceProtection
No                    2473    43.96
Yes                   1944    34.56
No internet service   1208    21.48

--- TechSupport ---
                     count  percent
TechSupport
No                    2759    49.05
Yes                   1658    29.48
No internet service   1208    21.48

--- StreamingTV ---
                     count  percent
StreamingTV
No                    2229    39.63
Yes                   2188    38.90
No internet service   1208    21.48

--- StreamingMovies ---
                     count  percent
StreamingMovies
No                    2222    39.50
Yes                   2195    39.02
No internet service   1208    21.48
```

```
--- Contract ---
                count  percent
Contract
Month-to-month   3085    54.84
Two year         1358    24.14
One year         1182    21.01

--- PaperlessBilling ---
                  count  percent
PaperlessBilling
Yes                3349    59.54
No                 2276    40.46

--- PaymentMethod ---
                           count  percent
PaymentMethod
Electronic check            1907    33.90
Mailed check                1268    22.54
Bank transfer (automatic)   1235    21.96
Credit card (automatic)     1215    21.60

Numeric distribution BEFORE any scaling (TRAIN):
                        mean          std   min      max
tenure             32.562311    24.542421   1.0    72.00
MonthlyCharges     64.999316    30.108642  18.4   118.65
TotalCharges     2301.839520  2275.586084  18.8  8684.80

================================================================
Scenario: S1) Standardization

Numeric distribution AFTER preprocessing (TRAIN):
                         mean       std       min       max
tenure         -1.291609e-16  1.000089 -1.286145  1.607062
MonthlyCharges  3.157968e-17  1.000089 -1.547843  1.782062
TotalCharges    8.842310e-18  1.000089 -1.003365  2.805224

Top 10 most frequent one-hot features (TRAIN) [%]:
PhoneService_Yes           90.31
Dependents_No              70.04
PaperlessBilling_Yes       59.54
Contract_Month-to-month    54.84
Partner_No                 51.40
gender_Male                50.19
OnlineSecurity_No          49.83
gender_Female              49.81
TechSupport_No             49.05
Partner_Yes                48.60

Model performance (TEST):
 accuracy  precision   recall       f1  roc_auc
 0.803838   0.648485 0.572193 0.607955 0.835942

================================================================
Scenario: S2) Normalization

Numeric distribution AFTER preprocessing (TRAIN):
```

```
                    mean       std  min  max
tenure          0.444540  0.345668  0.0  1.0
MonthlyCharges  0.464831  0.300336  0.0  1.0
TotalCharges    0.263448  0.262588  0.0  1.0

Top 10 most frequent one-hot features (TRAIN) [%]:
PhoneService_Yes           90.31
Dependents_No              70.04
PaperlessBilling_Yes       59.54
Contract_Month-to-month    54.84
Partner_No                 51.40
gender_Male                50.19
OnlineSecurity_No          49.83
gender_Female              49.81
TechSupport_No             49.05
Partner_Yes                48.60

Model performance (TEST):
 accuracy  precision   recall       f1  roc_auc
 0.804549   0.651376 0.569519 0.607703 0.834751

================================================================
Scenario: S3) Outlier(IQR) + Standardization

Outlier bounds (computed on TRAIN, IQR method):
tenure: low=-61.500, high=126.500
MonthlyCharges: low=-45.575, high=171.425
TotalCharges: low=-4670.050, high=8884.750

Numeric distribution AFTER preprocessing (TRAIN):
                         mean       std       min       max
tenure         -1.291609e-16  1.000089 -1.286145  1.607062
MonthlyCharges  3.157968e-17  1.000089 -1.547843  1.782062
TotalCharges    8.842310e-18  1.000089 -1.003365  2.805224

Top 10 most frequent one-hot features (TRAIN) [%]:
PhoneService_Yes           90.31
Dependents_No              70.04
PaperlessBilling_Yes       59.54
Contract_Month-to-month    54.84
Partner_No                 51.40
gender_Male                50.19
OnlineSecurity_No          49.83
gender_Female              49.81
TechSupport_No             49.05
Partner_Yes                48.60

Model performance (TEST):
 accuracy  precision   recall       f1  roc_auc
 0.803838   0.648485 0.572193 0.607955 0.835942

================================================================
Scenario: S4) Outlier(IQR) + Normalization

Outlier bounds (computed on TRAIN, IQR method):
tenure: low=-61.500, high=126.500
MonthlyCharges: low=-45.575, high=171.425
```

```
TotalCharges: low=-4670.050, high=8884.750

Numeric distribution AFTER preprocessing (TRAIN):
                   mean       std  min  max
tenure         0.444540  0.345668  0.0  1.0
MonthlyCharges 0.464831  0.300336  0.0  1.0
TotalCharges   0.263448  0.262588  0.0  1.0

Top 10 most frequent one-hot features (TRAIN) [%]:
PhoneService_Yes           90.31
Dependents_No              70.04
PaperlessBilling_Yes       59.54
Contract_Month-to-month    54.84
Partner_No                 51.40
gender_Male                50.19
OnlineSecurity_No          49.83
gender_Female              49.81
TechSupport_No             49.05
Partner_Yes                48.60

Model performance (TEST):
 accuracy  precision   recall       f1  roc_auc
 0.804549   0.651376 0.569519 0.607703 0.834751

================================================================
FINAL COMPARISON (sorted by ROC-AUC):
                          scenario  accuracy  precision   recall       f1
roc_auc  n_features_after_onehot
             S1) Standardization   0.803838   0.648485 0.572193 0.607955
0.835942                        45
S3) Outlier(IQR) + Standardization  0.803838   0.648485 0.572193 0.607955
0.835942                        45
               S2) Normalization   0.804549   0.651376 0.569519 0.607703
0.834751                        45
  S4) Outlier(IQR) + Normalization  0.804549   0.651376 0.569519 0.607703
0.834751                        45
```

توزیع غیرعددی‌ها قبل از پیش‌پردازش

genderتقریباً کاملاً متعادل است

Male حدود 50.19 درصد

Female حدود 49.81 درصد

SeniorCitizen نامتوازن است

مقدار 0 حدود 83.82 درصد

مقدار 1 حدود 16.18 درصد

Dependents هم نامتوازن است
No حدود 70.04 درصد
Yes حدود 29.96 درصد

PhoneService خیلی نامتوازن است
Yesحدود 90.31 درصد
Noحدود 9.69 درصد

این یعنی بعضی دسته‌ها خیلی پرتکرارند و بعضی کم‌تکرار. تبدیل One-Hot توزیع دسته‌ها را تغییر نمی‌دهد، فقط شکل نمایش را عددی می‌کند.

توزیع عددی قبل از پیش‌پردازش

tenure
میانگین حدود 32.56
انحراف معیار حدود 24.54
کمینه 1 و بیشینه 72

MonthlyCharges
میانگین حدود 65
انحراف معیار حدود 30.11
کمینه 18.4 و بیشینه 118.65

TotalCharges
میانگین حدود 2301.84
انحراف معیار حدود 2275.59
کمینه 18.8 و بیشینه 8684.8

TotalChargesدامنه خیلی بزرگ‌تری نسبت به بقیه دارد، پس اگر مقیاس‌بندی انجام نشود ممکن است روی مدل‌های خطی اثر نامتوازن بگذارد. برای همین سناریوهای scaling منطقی هستند.

سناریو S1: استانداردسازی

اثر روی توزیع عددی

پس از اعمال StandardScaler، میانگین هر ستون عددی به حدود 0 و انحراف معیار به حدود 1 رسیده است. این یعنی داده‌های عددی هم‌مقیاس شده‌اند.

اثر روی One-Hot

در ویژگی‌های One-Hot، هیچ تغییری رخ نداده و درصدها کاملا ثابت مانده است. برای مثال PhoneService_Yes همچنان 90.31٪ است. دلیل این است که scaling روی ستون‌های غیرعددی اعمال نمی‌شود.

عملکرد مدل

- accuracy = 0.803838
- f1 = 0.607955
- roc_auc = 0.835942

سناریو S2: نرمال‌سازی

اثر روی توزیع عددی

پس از اعمال MinMaxScaler، کمینه هر ستون عددی به 0 و بیشینه به 1 رسیده است. این یعنی داده‌های عددی در بازه [0,1] قرار گرفته‌اند.

عملکرد مدل

- accuracy = 0.804549
- f1 = 0.607703
- roc_auc = 0.834751

این دو سناریو عملکرد بسیار نزدیکی دارند، اما از نظر معیار ROC_AUC، سناریوی استانداردسازی (S1) نتیجه بهتری داده است.

سناریو S3: IQR پرت + استانداردسازی

نتایج نشان می‌دهد که مرزهای محاسبه‌شده برای شناسایی پرت‌ها بسیار بزرگتراز محدوده واقعی داده‌ها هستند.

- Tenure: بازه پرت 61.5- تا 126.5، در حالی که داده‌های واقعی بین 1 تا 72 هستند

- MonthlyCharges: بازه پرت 45.575- تا 171.425، در حالی که داده‌های واقعی بین 18.4 و 118.65 هستند

- TotalCharges: بازه پرت 4670.05- تا 8884.75، در حالی که داده‌های واقعی بین 18.8 و 8684.8 هستند

از آنجا که هیچ نقطه‌ای خارج از این بازه‌ها قرار نداشته، عمل clipping انجام نشده و مدیریت پرت عملا هیچ تغییری در داده‌ها ایجاد نکرده است. به همین دلیل، توزیع داده‌های عددی و عملکرد مدل در این سناریو دقیقا برابر با سناریوی S1 (استانداردسازی ساده) شده است.

سناریو S4: IQR پرت + نرمال‌سازی

دقیقاً همان اتفاق S3 تکرار شده است. از آنجا که داده‌ها هیچ مقدار پرتی خارج از بازه‌های محاسبه‌شده IQR نداشتند، عمل clipping هیچ تغییری در داده‌ها اعمال نکرده است. توزیع داده‌های عددی دقیقا برابر با سناریوی S2 (نرمال‌سازی ساده) شده است عملکرد مدل نیز کاملا مشابه S2 است بنابراین مدیریت پرت در این دیتاست عملا تاثیری بر خروجی نداشته و نتایج دقیقا مشابه حالت بدون مدیریت پرت است.

در نهایت بهترین مقدار ROC_AUC متعلق به سناریوهای S1 و S3 با مقدار 0.835942 است. اما از آنجا که در سناریوی S3 مرحله مدیریت پرت با روش IQR هیچ تغییری در داده‌ها ایجاد نکرده است، این مرحله عملا بی‌اثر و اضافی محسوب می‌شود.

بنابراین بهترین سناریو برای ادامه پروژه سناریو S1 استانداردسازی (StandardScaler) به همراه پیش‌پردازش‌های مشترک ابتدایی است چرا که:

- بالاترین مقدار ROC_AUC را در بین سناریوها دارد

- ساده‌تر و کم‌هزینه‌تر از S3 است

- مرحله مدیریت پرت با روش IQR در این دیتاست هیچ تغییری اعمال نکرده و افزودن آن تنها باعث پیچیدگی بی‌دلیل کد می‌شود.

فصل 6- بخش سوم

۳. کاهش ابعاد (Dimensionality Reduction):

- پیاده‌سازی تکنیک‌های کاهش ابعاد (مانند PCA، LDA، t-SNE و ...).
- تحلیل اثر کاهش ابعاد بر روی دو فاکتور کلیدی: **دقت مدل** (Accuracy) و **سرعت آموزش/استنتاج** (Speed/Computational Cost).

در این قسمت ابتدا پیش پردازش های منتخب از قسمت قبل را مجدد روی دیتاست اصلی که قبلا از آن کپی گرفته بودیم انجام می‌دهیم و سپس به سراغ کاهش ابعاد با استفاده از PCA ,LDA t-SNE می‌رویم.

```
#3
import numpy as np
import pandas as pd
from time import perf_counter

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.manifold import TSNE

if "customerID" in data.columns:
    data = data.drop(columns=["customerID"])

if "TotalCharges" in data.columns:
    data["TotalCharges"] = data["TotalCharges"].astype(str).str.strip()
    data.loc[data["TotalCharges"].eq(""), "TotalCharges"] = np.nan
    data["TotalCharges"] = pd.to_numeric(data["TotalCharges"],
errors="coerce")

data = data.dropna().copy()

y = data["Churn"].map({"Yes": 1, "No": 0})
X = data.drop(columns=["Churn"])

X = pd.get_dummies(X, drop_first=False)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
```

```
)

def run_method(method_name):
    t0 = perf_counter()

    scaler = StandardScaler()
    Xtr_s = scaler.fit_transform(X_train.values)
    Xte_s = scaler.transform(X_test.values)

    reducer = None

    if method_name == "Baseline":
        Xtr_r = Xtr_s
        Xte_r = Xte_s

    elif method_name == "PCA":
        reducer = PCA(n_components=0.99, random_state=42)
        Xtr_r = reducer.fit_transform(Xtr_s)
        Xte_r = reducer.transform(Xte_s)

    elif method_name == "LDA":
        reducer = LinearDiscriminantAnalysis(n_components=1)
        Xtr_r = reducer.fit_transform(Xtr_s, y_train.values)
        Xte_r = reducer.transform(Xte_s)

    elif method_name == "t-SNE":
        reducer = TSNE(
            n_components=2,
            perplexity=30,
            learning_rate="auto",
            init="pca",
            random_state=42
        )
        X_all = np.vstack([Xtr_s, Xte_s])
        X_all_emb = reducer.fit_transform(X_all)
        Xtr_r = X_all_emb[:len(X_train)]
        Xte_r = X_all_emb[len(X_train):]

    else:
        raise ValueError("Unknown method")

    model = LogisticRegression(max_iter=3000, solver="liblinear",
random_state=42)
    model.fit(Xtr_r, y_train.values)
```

```
    fit_time = perf_counter() - t0

    t1 = perf_counter()

    if method_name in ["Baseline", "PCA", "LDA"]:
        Xte_s2 = scaler.transform(X_test.values)
        if method_name == "Baseline":
            Xte_final = Xte_s2
        elif method_name == "PCA":
            Xte_final = reducer.transform(Xte_s2)
        elif method_name == "LDA":
            Xte_final = reducer.transform(Xte_s2)

    elif method_name == "t-SNE":
        Xte_final = Xte_r

    proba = model.predict_proba(Xte_final)[:, 1]
    pred = (proba >= 0.5).astype(int)

    inference_time = perf_counter() - t1

    acc = accuracy_score(y_test.values, pred)
    f1 = f1_score(y_test.values, pred)
    auc = roc_auc_score(y_test.values, proba)

    return {
        "method": method_name,
        "n_features_after": int(Xtr_r.shape[1]),
        "fit_time_sec": float(fit_time),
        "inference_time_sec": float(inference_time),
        "accuracy": float(acc),
        "f1": float(f1),
        "roc_auc": float(auc),
    }

results = []
for m in ["Baseline", "PCA", "LDA", "t-SNE"]:
    results.append(run_method(m))

results_df = pd.DataFrame(results).sort_values("roc_auc", ascending=False)

display(results_df)
```

ابتدا داده‌ها را بارگذاری کردیم، ستون بی‌ارزش customerID را حذف کردیم، ستون TotalCharges را که بعضی مقادیرش به صورت رشته خالی بود به مقدار عددی تبدیل کردیم و سطرهای دارای مقدار گمشده را حذف کردیم. سپس متغیر هدف Churn را به ۰ و ۱ نگاشت کردیم و ویژگی‌های غیرعددی را با روش One-Hot Encoding به عددی تبدیل کردیم. بعد از هم‌تراز کردن ستون‌های train وtest ، همه ویژگی‌ها را با StandardScalerاستانداردسازی کردیم. در ادامه چهار حالت را مقایسه کردیم: بدون کاهش بعد (Baseline)، کاهش بعد باPCA ، کاهش بعد با LDA و کاهش بعد با .t-SNE در هر حالت، یک مدل Logistic Regressionآموزش دادیم و معیارهای دقت و همچنین زمان آموزش و پیش‌بینی را اندازه‌گیری و مقایسه کردیم.

تابع train_eval

ابتدا مدل را آموزش می‌دهد (زمانش می‌شود train_time_sec) سپس روی داده test احتمال را حساب می‌کند و پیش‌بینی می‌کند (زمانش می‌شود inference_time_sec) در نهایت accuracy ، f1 ، roc_aucرا می‌دهد.در کد ما ، inference_time_secفقط زمان predict مدل است نه زمان transformبا PCA/LDA چون transform را قبل از صدا زدن train_eval انجام دادیم.

خروجی

	method	n_features_after	fit_time_sec	inference_time_sec	accuracy	f1	roc_auc
0	Baseline	45	0.135082	0.008615	0.803838	0.609065	0.835564
1	PCA	20	0.245199	0.020899	0.798152	0.600000	0.828757
2	LDA	1	0.228243	0.011977	0.794598	0.582973	0.828286
3	t-SNE	2	90.732185	0.000629	0.738451	0.361111	0.740901

اثر روی دقت مدل

Baseline بهترین دقت و بهترین ROC-AUC را دارد:

accuracy ≈ 0.8038

roc_auc ≈ 0.8356 (بهترین)

PCA افت خیلی کم نسبت به baseline دارد:

accuracy (0.0056 افت حدود) 0.7982 ⟵ 0.8038
roc_auc (0.0068 افت حدود) 0.8288 ⟵ 0.8356

یعنی PCA با نصف کردن ابعاد، هنوز عملکرد نسبتا نزدیک نگه داشته است.

LDA چون فقط 1 به بعد را نگه می‌دارد، افتش بیشتر حس می‌شود:

accuracy ≈ 0.7946
f1 افت واضح‌تر دارد (0.583)

t-SNE بدترین عملکرد را داده است:

accuracy ≈ 0.738
f1 ≈ 0.361 (خیلی پایین)
roc_auc ≈ 0.741 (خیلی پایین)

از نظر دقت به ترتیب Baseline > PCA ≈ LDA >> t-SNE

اثر روی سرعت آموزش

t-SNE از نظر زمان آموزش فاجعه‌است چرا که حدود 89 ثانیه طول کشیده است. دلیلش این است که t-SNEذاتا محاسباتی سنگین است.

بین سه تای دیگر

LDA سریع‌ترین fit_time را داده(0.141s)

Baseline دوم است (0.169s)

PCA کندتر شده(0.263s) زیرا خود محاسبه PCA (fit_transform) هزینه دارد. روی دیتاست‌های کوچک مثل دیت ست ما، هزینه PCA از سود کاهش بعد بیشتر می‌شود، برای همین fit_time بالا می‌رود.

اثر روی سرعت استنتاج

t-SNE خیلی کم شده (0.000464) چون فقط 2 ویژگی به مدل می‌رسد. inference

بین Baseline/PCA/LDA

Baseline: 0.0127

PCA: 0.0097 (کمی بهتر)

LDA: 0.0092 (کمی بهتر)

در نهایت بین تکنیک های کاهش ابعاد PCA از بقیه بهتر است.

فصل 7- بخش چهارم

۴. انتخاب و آموزش مدل‌ها:

- انتخاب حداقل **دو روش** یادگیری (روش‌های کلاسیک مطرح شده در درس یا روش‌های پیشرفته‌تر مانند Deep Learning).
- توجیه انتخاب مدل‌ها با توجه به نوع داده و مسئله.

در این قسمت بعد انجام PCA بر روی داده ها، اقدام به آموزش مدل با سه روش Logistic Regression و SVM و Decision Tree می‌کنیم.

```
#4
import numpy as np
import pandas as pd
from time import perf_counter

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

if "customerID" in data.columns:
    data = data.drop(columns=["customerID"])

if "TotalCharges" in data.columns:
    data["TotalCharges"] = data["TotalCharges"].astype(str).str.strip()
    data.loc[data["TotalCharges"].eq(""), "TotalCharges"] = np.nan
    data["TotalCharges"] = pd.to_numeric(data["TotalCharges"],
errors="coerce")

if "SeniorCitizen" in data.columns:
    data["SeniorCitizen"] = data["SeniorCitizen"].astype(str)

data = data.dropna().copy()

y = data["Churn"].map({"Yes": 1, "No": 0})
X = data.drop(columns=["Churn"])
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

X_train_ohe = pd.get_dummies(X_train, drop_first=False)
X_test_ohe  = pd.get_dummies(X_test, drop_first=False)
X_train_ohe, X_test_ohe = X_train_ohe.align(X_test_ohe, join="left",
axis=1, fill_value=0)

scaler = StandardScaler()
pca = PCA(n_components=0.99, random_state=42)

t0 = perf_counter()
Xtr_scaled = scaler.fit_transform(X_train_ohe)
Xtr_pca = pca.fit_transform(Xtr_scaled)
prep_fit_time = perf_counter() - t0

n_features_after = Xtr_pca.shape[1]
print("n_features_after (PCA):", n_features_after)

def eval_model(model_name, model):
    t0 = perf_counter()
    model.fit(Xtr_pca, y_train)
    model_fit_time = perf_counter() - t0

    t1 = perf_counter()
    Xte_scaled = scaler.transform(X_test_ohe)
    Xte_pca = pca.transform(Xte_scaled)

    if hasattr(model, "predict_proba"):
        scores = model.predict_proba(Xte_pca)[:, 1]
        pred = (scores >= 0.5).astype(int)
        roc_input = scores
    else:
        scores = model.decision_function(Xte_pca)
        pred = (scores >= 0).astype(int)
        roc_input = scores

    inference_time = perf_counter() - t1

    return {
        "model": model_name,
        "n_features_after": n_features_after,
        "fit_time_sec": prep_fit_time + model_fit_time,
        "inference_time_sec": inference_time,
```

```
        "accuracy": accuracy_score(y_test, pred),
        "f1": f1_score(y_test, pred),
        "roc_auc": roc_auc_score(y_test, roc_input),
    }

results = []

lr = LogisticRegression(
    max_iter=3000,
    solver="liblinear",
    random_state=42,
    class_weight="balanced"
)
results.append(eval_model("LogisticRegression + PCA", lr))

svm = LinearSVC(
    random_state=42,
    class_weight="balanced"
)
results.append(eval_model("LinearSVM + PCA", svm))

dt = DecisionTreeClassifier(
    random_state=42,
    class_weight="balanced",
    max_depth=None
)
results.append(eval_model("DecisionTree + PCA", dt))

results_df = pd.DataFrame(results).sort_values("roc_auc", ascending=False)
display(results_df)
```

پس از اینکه با PCA ابعاد را طوری کم کردیم که 99٪ واریانس حفظ شود (که اینجا شد 20 ویژگی). در نهایت، روی داده‌ی کاهش‌بعدیافته سه مدل Logistic Regression و Linear SVM و Decision Treeرا آموزش دادیم و علاوه بر معیارهای دقت و کیفیت، زمان آموزش و زمان پیش‌بینی را هم اندازه گرفتیم. بیشتر کد که مشابه قسمت قبل است در انتها تابع های مربوط به هر مدل را نیز می نویسیم.

```
n_features_after (PCA): 20
```

	model	n_features_after	fit_time_sec	inference_time_sec	accuracy	f1	roc_auc
0	LogisticRegression + PCA	20	0.092831	0.025928	0.722814	0.600410	0.828198
1	LinearSVM + PCA	20	0.100689	0.016539	0.719972	0.600406	0.828014
2	DecisionTree + PCA	20	0.676485	0.015128	0.705046	0.460338	0.631290

کیفیت مدل‌ها

Logistic Regression و Linear SVM تقریباً برابرند

roc_auc هر دو حدود 0.828 خیلی نزدیک بهم

f1 هر دو حدود 0.60 یعنی قدرت تفکیک‌شان روی داده PCA شده تقریبا یکی است.

Decision Tree واضحا ضعیف‌تر اس:

roc_auc = 0.63 یعنی مدل خیلی بدتر تفکیک می‌کند

f1 = 0.46 یعنی در پیدا کردن churners ضعیف‌تر است

پس برای ادامه پروژه انتخاب خوبی نیست.

سرعت آموزش و استنتاج

زمان آموزش

LR ≈ 0.093s

SVM ≈ 0.101s

اختلاف خیلی کم است (عملا هر دو سریع‌اند).

Decision Tree ≈ 0.676s (خیلی بیشتر)

زمان پیش‌بینی

SVM و DecisionTree کمی سریع‌تر از LR هستند.

LR معمولا چون predict_proba می‌دهد، ممکن است کمی کندتراز decision_function باشد.

چرا این مدل‌ها را انتخاب کردیم؟

مسئله ما یک طبقه‌بندی دودویی برای پیش‌بینی ریزش مشتری (Churn Yes/No) است. داده‌ها به صورت جدولی و متشکل از ویژگی‌های عددی و دسته‌ای هستند. پس از اعمالOne-Hot Encoding ، تعداد ویژگی‌ها افزایش یافته و سپس با PCA به فضای پیوسته و فشرده کاهش بعد یافته‌اند.

Logistic Regression

به عنوان یک مدل کلاسیک و استاندارد برای طبقه‌بندی دودویی آن را انتخاب کردیم. این مدل روی ویژگی‌های استانداردشده و به ویژه پس از کاهش بعد با PCA عملکرد بسیار خوبی دارد. خروجی احتمال (predict_proba)در این مدل، برای محاسبه ROC-AUC و تنظیم آستانه تصمیم‌گیری بسیار کاربردی است. علاوه بر این، سرعت بالا، پایداری و قابلیت گزارش‌پذیری از دیگر مزایای این مدل هستند.

SVM خطی

برای داده‌های با ابعاد نسبتا بالا، حتی پس از کاهش بعد باPCA ، گزینه مناسبی محسوب می‌شود. این مدل با داده‌های استانداردشده به خوبی کار می‌کند و معمولا قدرت تعمیم بالایی دارد. در بسیاری از مسائل، عملکرد SVMخطی به Logistic Regression نزدیک است و از نظر سرعت در مرحله Inference نیز کارآمد است.

Decision Tree

این مدل به دلیل قابلیت تفسیرپذیری شناخته می‌شود، اما پس از اعمالPCA ، ویژگی‌ها ترکیب خطی از ویژگی‌های اصلی هستند و معنای مستقیم خود را از دست می‌دهند. در نتیجه، هم از نظر تفسیرپذیری و هم از نظر عملکرد، درخت تصمیم افت محسوسی دارد که خروجی مدل نیز این کاهش عملکرد را تأیید کرده است.

فصل 8- بخش پنجم

۵. تنظیم هایپرپارامترها (Hyperparameter Tuning):

- استفاده از روش‌های جستجوی نظام‌مند مانند Grid Search، Random Search یا کتابخانه‌های بهینه‌سازی مانند Optuna.
- گزارش دقیق تمام هایپرپارامترهای جستجو شده و مقادیر بهینه نهایی.

```
#5
!pip -q install optuna
from sklearn.model_selection import StratifiedKFold, GridSearchCV,
RandomizedSearchCV, cross_val_score
import optuna
from optuna.samplers import TPESampler
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

def save_cv_results(cv_results, filename):
    df2 = pd.DataFrame(cv_results)[["mean_test_score", "std_test_score",
"rank_test_score", "params"]]
    df2 = df2.sort_values("rank_test_score")
    df2.to_csv(filename, index=False)
    print("Saved:", filename, "| rows:", len(df2))

def build_model(model_key, params):
    if model_key == "LogisticRegression":
        m = LogisticRegression(max_iter=3000, solver="liblinear",
random_state=42)
    elif model_key == "LinearSVM":
        m = LinearSVC(random_state=42, max_iter=5000)
    elif model_key == "DecisionTree":
        m = DecisionTreeClassifier(random_state=42)
    else:
        raise ValueError("Unknown model key")

    m.set_params(**params)
    return m


grid_spaces = {
    "LogisticRegression": {
        "C": [0.01, 0.1, 1, 10, 100],
        "penalty": ["l1", "l2"],
        "class_weight": [None, "balanced"]
    },
```

```
    "LinearSVM": {
        "C": [0.01, 0.1, 1, 10, 100],
        "class_weight": [None, "balanced"]
    },
    "DecisionTree": {
        "criterion": ["gini", "entropy"],
        "max_depth": [None, 5, 10],
        "min_samples_split": [2, 10],
        "min_samples_leaf": [1, 5],
        "class_weight": [None, "balanced"]
    }
}

random_spaces = {
    "LogisticRegression": {
        "C": list(np.logspace(-3, 2, 25)),
        "penalty": ["l1", "l2"],
        "class_weight": [None, "balanced"]
    },
    "LinearSVM": {
        "C": list(np.logspace(-3, 2, 25)),
        "class_weight": [None, "balanced"]
    },
    "DecisionTree": {
        "criterion": ["gini", "entropy"],
        "max_depth": [None, 3, 5, 8, 12],
        "min_samples_split": [2, 5, 10, 20, 40],
        "min_samples_leaf": [1, 2, 5, 10, 20],
        "class_weight": [None, "balanced"]
    }
}

models_base = {
    "LogisticRegression": LogisticRegression(max_iter=3000,
solver="liblinear", random_state=42),
    "LinearSVM": LinearSVC(random_state=42, max_iter=5000),
    "DecisionTree": DecisionTreeClassifier(random_state=42)
}

best_summary = []
final_models_results = []


for model_key in ["LogisticRegression", "LinearSVM", "DecisionTree"]:
    print("\n" + "="*80)
```

```
    print("TUNING MODEL:", model_key)
    print("="*80)

    # ---------------- GRID SEARCH ----------------
    print("\nGRID SEARCH SPACE:")
    for k, v in grid_spaces[model_key].items():
        print(k, "->", v)

    gs = GridSearchCV(
        estimator=models_base[model_key],
        param_grid=grid_spaces[model_key],
        scoring="roc_auc",
        cv=cv,
        n_jobs=-1,
        refit=True
    )

    t0 = perf_counter()
    gs.fit(Xtr_pca, y_train)
    t1 = perf_counter()

    print("Grid best CV roc_auc:", gs.best_score_)
    print("Grid best params:", gs.best_params_)
    print("Grid time (sec):", round(t1 - t0, 3))

    save_cv_results(gs.cv_results_, f"grid_{model_key}.csv")

    best_summary.append({
        "model": model_key,
        "search": "GridSearch",
        "best_cv_roc_auc": gs.best_score_,
        "best_params": gs.best_params_
    })

    best_g = build_model(model_key, gs.best_params_)
    final_models_results.append(eval_model(f"{model_key} + PCA | Grid
best", best_g))


    # ---------------- RANDOM SEARCH ----------------
    print("\nRANDOM SEARCH SPACE:")
    for k, v in random_spaces[model_key].items():
        if isinstance(v, list) and len(v) > 12:
            print(k, "->", f"list(len={len(v)}) sample:", v[:5], "...",
v[-3:])
```

```
        else:
            print(k, "->", v)

    rs = RandomizedSearchCV(
        estimator=models_base[model_key],
        param_distributions=random_spaces[model_key],
        n_iter=20,
        scoring="roc_auc",
        cv=cv,
        n_jobs=-1,
        refit=True,
        random_state=42
    )

    t0 = perf_counter()
    rs.fit(Xtr_pca, y_train)
    t1 = perf_counter()

    print("Random best CV roc_auc:", rs.best_score_)
    print("Random best params:", rs.best_params_)
    print("Random time (sec):", round(t1 - t0, 3))

    save_cv_results(rs.cv_results_, f"random_{model_key}.csv")

    best_summary.append({
        "model": model_key,
        "search": "RandomSearch",
        "best_cv_roc_auc": rs.best_score_,
        "best_params": rs.best_params_
    })

    best_r = build_model(model_key, rs.best_params_)
    final_models_results.append(eval_model(f"{model_key} + PCA | Random
best", best_r))


    # ---------------- OPTUNA ----------------
    print("\nOPTUNA SEARCH SPACE:")
    if model_key in ["LogisticRegression", "LinearSVM"]:
        print("C: log float in [1e-3, 1e2]")
        print("class_weight: [None, 'balanced']")
        if model_key == "LogisticRegression":
            print("penalty: ['l1','l2']")
    else:
        print("criterion: ['gini','entropy']")
```

```
        print("max_depth: [None,3,5,8,12]  categorical")
        print("min_samples_split: int in [2,40]")
        print("min_samples_leaf: int in [1,20]")
        print("class_weight: [None, 'balanced']")

    def objective(trial):
        if model_key == "LogisticRegression":
            params = {
                "C": trial.suggest_float("C", 1e-3, 1e2, log=True),
                "penalty": trial.suggest_categorical("penalty", ["l1",
"l2"]),
                "class_weight": trial.suggest_categorical("class_weight",
[None, "balanced"])
            }
            m = LogisticRegression(max_iter=3000, solver="liblinear",
random_state=42, **params)

        elif model_key == "LinearSVM":
            params = {
                "C": trial.suggest_float("C", 1e-3, 1e2, log=True),
                "class_weight": trial.suggest_categorical("class_weight",
[None, "balanced"])
            }
            m = LinearSVC(random_state=42, max_iter=5000, **params)

        else:
            params = {
                "criterion": trial.suggest_categorical("criterion",
["gini", "entropy"]),
                "max_depth": trial.suggest_categorical("max_depth", [None,
3, 5, 8, 12]),
                "min_samples_split":
trial.suggest_int("min_samples_split", 2, 40),
                "min_samples_leaf": trial.suggest_int("min_samples_leaf",
1, 20),
                "class_weight": trial.suggest_categorical("class_weight",
[None, "balanced"])
            }
            m = DecisionTreeClassifier(random_state=42, **params)

        score = cross_val_score(m, Xtr_pca, y_train, scoring="roc_auc",
cv=cv, n_jobs=-1).mean()
        return score
    sampler = TPESampler(seed=42)
    study = optuna.create_study(direction="maximize", sampler=sampler)
```

```
    t0 = perf_counter()
    study.optimize(objective, n_trials=20, show_progress_bar=False)
    t1 = perf_counter()

    print("Optuna best CV roc_auc:", study.best_value)
    print("Optuna best params:", study.best_trial.params)
    print("Optuna time (sec):", round(t1 - t0, 3))

    trials_df = study.trials_dataframe()
    trials_df.to_csv(f"optuna_{model_key}.csv", index=False)
    print("Saved:", f"optuna_{model_key}.csv", "| rows:", len(trials_df))

    best_summary.append({
        "model": model_key,
        "search": "Optuna",
        "best_cv_roc_auc": study.best_value,
        "best_params": study.best_trial.params
    })

    best_o = build_model(model_key, study.best_trial.params)
    final_models_results.append(eval_model(f"{model_key} + PCA | Optuna
best", best_o))


best_params_df = pd.DataFrame(best_summary)
final_compare_df =
pd.DataFrame(final_models_results).sort_values("roc_auc", ascending=False)

print("\n" + "="*80)
print("BEST PARAMS SUMMARY (CV)")
print("="*80)
print(best_params_df.to_string(index=False))

print("\n" + "="*80)
print("FINAL TEST COMPARISON (sorted by roc_auc)")
print("="*80)
print(final_compare_df.to_string(index=False))

best_params_df.to_csv("best_params_summary.csv", index=False)
final_compare_df.to_csv("final_test_comparison.csv", index=False)

print("\nSaved: best_params_summary.csv")
print("Saved: final_test_comparison.csv")

print("\nAlso saved per-model full results:")
```

```
print("grid_<Model>.csv , random_<Model>.csv , optuna_<Model>.csv")
```

در این قسمت کد وارد فاز تنظیم هایپرپارامترها می‌شود. یعنی برای هر 3 مدل، سه روش تنظیم را اجرا می‌کنیم:

Grid Search: جستجوی کامل روی یک شبکه کوچک از مقادیر

Random Search: نمونه‌برداری تصادفی از یک فضای بزرگ‌تر

Optuna: بهینه‌سازی هوشمند (TPE) برای پیدا کردن ترکیب بهتر با تعداد آزمون محدود

و در نهایت بهترین تنظیمات هر روش را با Cross-Validationپیدا می‌کند. سپس بهترین مدل‌های به‌دست‌آمده را روی Test واقعی با تابع eval_modelارزیابی می‌کنیم. در آخر نتایج هم چاپ و هم به صورت CSV ذخیره می‌شوند.

```
def build_model(model_key, params):
    ...
    m.set_params(**params)
    return m
```

این قسمت برای این است که بعد ازRandom, Grid ,Optuna، بهترین پارامترها را برداریم و یک مدل واقعی بسازیم. اینجا فقط مدل ساخته می‌شود آموزش داخل eval_model انجام می‌شود.

Grid Search Space

شبکه کوچک و مشخصی است.

```
LR: C, penalty, class_weight
SVM: C, class_weight
DT: criterion, max_depth, min_samples_split, min_samples_leaf, class_weight
```

Gridیعنی همه ترکیب‌ها امتحان می‌شوند.

Random Search Space

فضای بزرگ‌تر

مثلا C با logspace از 0.001 تا 100

عمق‌ها و min_samples های بیشتر برایDT

Random یعنی از این فضا فقط n_iter=20نمونه تصادفی تست می‌شود.

حلقه اصلی روی سه مدل

```
for model_key in ["LogisticRegression", "LinearSVM", "DecisionTree"]:
```

برای هر مدل، سه مرحله انجام می‌شود:

GridSearchCV

```
gs = GridSearchCV(... scoring="roc_auc", cv=cv, n_jobs=-1, refit=True)
gs.fit(Xtr_pca, y_train)
```

scoring="roc_auc" معیار انتخاب بهترین تنظیم است.

n_jobs=-1 یعنی استفاده از همه هسته‌های CPU برای سرعت.

refit=True یعنی بعد از پیدا کردن بهترین پارامتر، همان بهترین مدل دوباره روی کل Train آموزش داده می‌شود و در best_estimator_ آماده است.

زمان اجرا را با perf_counter اندازه گرفته‌ایم.

بهترین CV score و بهترین پارامتر چاپ می‌شود نتایج کامل Grid در grid_<Model>.csv ذخیره می‌گردد

سپس با build_model + eval_modelروی Test واقعی ارزیابی می‌کنیم:

```
final_models_results.append(eval_model(f"{model_key} + PCA | Grid best",
best_g))
```

RandomizedSearchCV

مشابه Grid است ولی فضای جستجو بزرگ‌تر است.

n_iter=20 یعنی فقط 20 ترکیب بررسی می‌شود

خروجی‌ها هم مثل Grid ذخیره می‌شوند random_<Model>.csv

و ارزیابی نهایی روی Test انجام می‌گردد.

Optuna

اینجا یک تابع هدف تعریف می‌کنیم:

```
def objective(trial):
   ...
   score = cross_val_score(m, Xtr_pca, y_train, scoring="roc_auc", cv=cv,
n_jobs=-1).mean()
   return score
```

Optunaدر هر trial یک ترکیب پارامتر پیشنهاد می‌دهد .برای همان ترکیب، 5 fold ROC-AUC-را با cross_val_score حساب می‌کنیم. سپس میانگین امتیاز را برمی‌گردانیم.

Optuna تلاش می‌کند در 20 آزمون به بهترین ترکیب برسد.

در نهایت بهترین مقدار (best_value) و پارامترها چاپ می‌شود

لیست همهtrial ها ذخیره می‌شود و بهترین مدل ساخته و روی Test ارزیابی می‌شود.

```
optuna_<Model>.csv
```

خروجی

```
================================================================================
===
TUNING MODEL: LogisticRegression
================================================================================
===

GRID SEARCH SPACE:
C -> [0.01, 0.1, 1, 10, 100]
penalty -> ['l1', 'l2']
class_weight -> [None, 'balanced']
Grid best CV roc_auc: 0.8423194344343938
Grid best params: {'C': 0.1, 'class_weight': None, 'penalty': 'l1'}
Grid time (sec): 3.005
Saved: grid_LogisticRegression.csv | rows: 20

RANDOM SEARCH SPACE:
C -> list(len=25) sample: [np.float64(0.001),
np.float64(0.0016155980984398745), np.float64(0.0026101572156825357),
np.float64(0.004216965034285823), np.float64(0.006812920690579615)] ...
```

```
[np.float64(38.31186849557293), np.float64(61.8965818891261),
np.float64(100.0)]
penalty -> ['l1', 'l2']
class_weight -> [None, 'balanced']
[I 2026-02-13 01:49:50,815] A new study created in memory with name: no-name-
1d6c2988-85c4-4a27-9a6a-e3140aa0aa96
Random best CV roc_auc: 0.8423121462178205
Random best params: {'penalty': 'l1', 'class_weight': None, 'C':
np.float64(0.19573417814876617)}
Random time (sec): 4.105
Saved: random_LogisticRegression.csv | rows: 20

OPTUNA SEARCH SPACE:
C: log float in [1e-3, 1e2]
class_weight: [None, 'balanced']
penalty: ['l1','l2']
[I 2026-02-13 01:49:51,028] Trial 0 finished with value: 0.8422991893883566
and parameters: {'C': 0.0745934328572655, 'penalty': 'l1', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:51,166] Trial 1 finished with value: 0.8416553969243725
and parameters: {'C': 0.0060252157362038605, 'penalty': 'l2', 'class_weight':
'balanced'}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:51,282] Trial 2 finished with value: 0.7955080291852583
and parameters: {'C': 0.001267425589893723, 'penalty': 'l1', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:51,441] Trial 3 finished with value: 0.8410593827690365
and parameters: {'C': 0.008260808399079604, 'penalty': 'l2', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:51,749] Trial 4 finished with value: 0.8419056256933928
and parameters: {'C': 1.1462107403425035, 'penalty': 'l2', 'class_weight':
'balanced'}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:52,032] Trial 5 finished with value: 0.8421477564439982
and parameters: {'C': 8.43101393208247, 'penalty': 'l2', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:52,211] Trial 6 finished with value: 0.8419331589560034
and parameters: {'C': 1.0907475835157696, 'penalty': 'l1', 'class_weight':
'balanced'}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:52,451] Trial 7 finished with value: 0.8421445172366322
and parameters: {'C': 11.015056790269638, 'penalty': 'l1', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:52,565] Trial 8 finished with value: 0.8198150412594039
and parameters: {'C': 0.004075596440072870, 'penalty': 'l1', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:52,783] Trial 9 finished with value: 0.8421437074347906
and parameters: {'C': 2.0540519425388455, 'penalty': 'l2', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:52,961] Trial 10 finished with value: 0.8421931053471216
and parameters: {'C': 0.07453892933860191, 'penalty': 'l1', 'class_weight':
'balanced'}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:53,256] Trial 11 finished with value: 0.8421752897066088
and parameters: {'C': 0.05423185466354641, 'penalty': 'l1', 'class_weight':
'balanced'}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:53,497] Trial 12 finished with value: 0.8421680014900353
and parameters: {'C': 0.11547408571799433, 'penalty': 'l1', 'class_weight':
'balanced'}. Best is trial 0 with value: 0.8422991893883566.
```

```
[I 2026-02-13 01:49:53,696] Trial 13 finished with value: 0.8421939151489631
and parameters: {'C': 0.07534500668676802, 'penalty': 'l1', 'class_weight':
'balanced'}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:53,854] Trial 14 finished with value: 0.8406561014519747
and parameters: {'C': 0.026870239798129827, 'penalty': 'l1', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:54,050] Trial 15 finished with value: 0.8420829722966789
and parameters: {'C': 0.22401322201281426, 'penalty': 'l1', 'class_weight':
'balanced'}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:54,236] Trial 16 finished with value: 0.8422425032594525
and parameters: {'C': 0.4604234767893719, 'penalty': 'l1', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:54,486] Trial 17 finished with value: 0.8421461368403153
and parameters: {'C': 75.61683084280057, 'penalty': 'l1', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:54,665] Trial 18 finished with value: 0.8422222582134152
and parameters: {'C': 0.49899998666091444, 'penalty': 'l1', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
[I 2026-02-13 01:49:54,841] Trial 19 finished with value: 0.8421550446605715
and parameters: {'C': 3.8202903332380145, 'penalty': 'l1', 'class_weight':
None}. Best is trial 0 with value: 0.8422991893883566.
Optuna best CV roc_auc: 0.8422991893883566
Optuna best params: {'C': 0.0745934328572655, 'penalty': 'l1',
'class_weight': None}
Optuna time (sec): 4.029
Saved: optuna_LogisticRegression.csv | rows: 20

================================================================================
===
TUNING MODEL: LinearSVM
================================================================================
===

GRID SEARCH SPACE:
C -> [0.01, 0.1, 1, 10, 100]
class_weight -> [None, 'balanced']
Grid best CV roc_auc: 0.8416132872286152
Grid best params: {'C': 100, 'class_weight': 'balanced'}
Grid time (sec): 1.326
Saved: grid_LinearSVM.csv | rows: 10

RANDOM SEARCH SPACE:
C -> list(len=25) sample: [np.float64(0.001),
np.float64(0.0016155980984398745), np.float64(0.0026101572156825357),
np.float64(0.004216965034285823), np.float64(0.006812920690579615)] ...
[np.float64(38.31186849557293), np.float64(61.8965818891261),
np.float64(100.0)]
class_weight -> [None, 'balanced']
[I 2026-02-13 01:49:59,194] A new study created in memory with name: no-name-
8773d417-ee29-4986-8936-057469310802
Random best CV roc_auc: 0.8416132872286152
Random best params: {'class_weight': 'balanced', 'C':
np.float64(5.623413251903491)}
Random time (sec): 2.744
Saved: random_LinearSVM.csv | rows: 20

OPTUNA SEARCH SPACE:
```

```
C: log float in [1e-3, 1e2]
class_weight: [None, 'balanced']
[I 2026-02-13 01:49:59,390] Trial 0 finished with value: 0.8402252868723024
and parameters: {'C': 0.0745934328572655, 'class_weight': None}. Best is
trial 0 with value: 0.8402252868723024.
[I 2026-02-13 01:49:59,569] Trial 1 finished with value: 0.8402463417201812
and parameters: {'C': 0.9846738873614566, 'class_weight': None}. Best is
trial 1 with value: 0.8402463417201812.
[I 2026-02-13 01:49:59,827] Trial 2 finished with value: 0.8393199284135171
and parameters: {'C': 0.0019517224641449498, 'class_weight': None}. Best is
trial 1 with value: 0.8402463417201812.
[I 2026-02-13 01:50:00,057] Trial 3 finished with value: 0.8416140970304566
and parameters: {'C': 3.4702669886504163, 'class_weight': 'balanced'}. Best
is trial 3 with value: 0.8416140970304566.
[I 2026-02-13 01:50:00,243] Trial 4 finished with value: 0.8402503907293886
and parameters: {'C': 14.528246637516036, 'class_weight': None}. Best is
trial 3 with value: 0.8416140970304566.
[I 2026-02-13 01:50:00,540] Trial 5 finished with value: 0.8416092382194078
and parameters: {'C': 0.008260808399079604, 'class_weight': 'balanced'}. Best
is trial 3 with value: 0.8416140970304566.
[I 2026-02-13 01:50:00,785] Trial 6 finished with value: 0.8416132872286152
and parameters: {'C': 0.14445251022763064, 'class_weight': 'balanced'}. Best
is trial 3 with value: 0.8416140970304566.
[I 2026-02-13 01:50:01,129] Trial 7 finished with value: 0.8416173362378228
and parameters: {'C': 0.004982752357076452, 'class_weight': 'balanced'}. Best
is trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:01,458] Trial 8 finished with value: 0.840249580927547
and parameters: {'C': 0.19069966103000435, 'class_weight': None}. Best is
trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:01,987] Trial 9 finished with value: 0.8402455319183396
and parameters: {'C': 0.37253938395788866, 'class_weight': None}. Best is
trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:02,262] Trial 10 finished with value: 0.8416027598046758
and parameters: {'C': 0.010916771197512777, 'class_weight': 'balanced'}. Best
is trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:02,433] Trial 11 finished with value: 0.8416124774267736
and parameters: {'C': 18.428603639984882, 'class_weight': 'balanced'}. Best
is trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:02,696] Trial 12 finished with value: 0.8416140970304566
and parameters: {'C': 3.671423529735164, 'class_weight': 'balanced'}. Best is
trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:02,931] Trial 13 finished with value: 0.8416124774267738
and parameters: {'C': 60.01447217163458, 'class_weight': 'balanced'}. Best is
trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:03,241] Trial 14 finished with value: 0.8416124774267736
and parameters: {'C': 1.3627563818737505, 'class_weight': 'balanced'}. Best
is trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:03,394] Trial 15 finished with value: 0.8416084284175662
and parameters: {'C': 0.02533989267880571, 'class_weight': 'balanced'}. Best
is trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:03,606] Trial 16 finished with value: 0.8415193502150025
and parameters: {'C': 0.0023181233009956915, 'class_weight': 'balanced'}.
Best is trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:03,769] Trial 17 finished with value: 0.8416132872286152
and parameters: {'C': 4.1334779054662825, 'class_weight': 'balanced'}. Best
is trial 7 with value: 0.8416173362378228.
```

```
[I 2026-02-13 01:50:03,991] Trial 18 finished with value: 0.8415970911917855
and parameters: {'C': 0.03575861199035915, 'class_weight': 'balanced'}. Best
is trial 7 with value: 0.8416173362378228.
[I 2026-02-13 01:50:04,141] Trial 19 finished with value: 0.8416108578230906
and parameters: {'C': 0.778750172539758, 'class_weight': 'balanced'}. Best is
trial 7 with value: 0.8416173362378228.
Optuna best CV roc_auc: 0.8416173362378228
Optuna best params: {'C': 0.004982752357076452, 'class_weight': 'balanced'}
Optuna time (sec): 4.945
Saved: optuna_LinearSVM.csv | rows: 20

================================================================================
===
TUNING MODEL: DecisionTree
================================================================================
===

GRID SEARCH SPACE:
criterion -> ['gini', 'entropy']
max_depth -> [None, 5, 10]
min_samples_split -> [2, 10]
min_samples_leaf -> [1, 5]
class_weight -> [None, 'balanced']
Grid best CV roc_auc: 0.8029379610809235
Grid best params: {'class_weight': 'balanced', 'criterion': 'gini',
'max_depth': 5, 'min_samples_leaf': 5, 'min_samples_split': 2}
Grid time (sec): 34.341
Saved: grid_DecisionTree.csv | rows: 48

RANDOM SEARCH SPACE:
criterion -> ['gini', 'entropy']
max_depth -> [None, 3, 5, 8, 12]
min_samples_split -> [2, 5, 10, 20, 40]
min_samples_leaf -> [1, 2, 5, 10, 20]
class_weight -> [None, 'balanced']
[I 2026-02-13 01:50:50,029] A new study created in memory with name: no-name-
d3dba24b-ea79-404c-849f-284ca63d8293
Random best CV roc_auc: 0.8036935061990331
Random best params: {'min_samples_split': 5, 'min_samples_leaf': 10,
'max_depth': 5, 'criterion': 'gini', 'class_weight': 'balanced'}
Random time (sec): 11.223
Saved: random_DecisionTree.csv | rows: 20

OPTUNA SEARCH SPACE:
criterion: ['gini','entropy']
max_depth: [None,3,5,8,12]  categorical
min_samples_split: int in [2,40]
min_samples_leaf: int in [1,20]
class_weight: [None, 'balanced']
[I 2026-02-13 01:50:50,784] Trial 0 finished with value: 0.7601237377213795
and parameters: {'criterion': 'entropy', 'max_depth': None,
'min_samples_split': 35, 'min_samples_leaf': 13, 'class_weight': None}. Best
is trial 0 with value: 0.7601237377213795.
[I 2026-02-13 01:50:51,430] Trial 1 finished with value: 0.7146043713103405
and parameters: {'criterion': 'gini', 'max_depth': 12, 'min_samples_split':
18, 'min_samples_leaf': 6, 'class_weight': None}. Best is trial 0 with value:
0.7601237377213795.
```

```
[I 2026-02-13 01:50:51,731] Trial 2 finished with value: 0.7867467830621847
and parameters: {'criterion': 'entropy', 'max_depth': 3, 'min_samples_split':
3, 'min_samples_leaf': 13, 'class_weight': None}. Best is trial 2 with value:
0.7867467830621847.
[I 2026-02-13 01:50:52,487] Trial 3 finished with value: 0.7527419890352831
and parameters: {'criterion': 'entropy', 'max_depth': None,
'min_samples_split': 6, 'min_samples_leaf': 10, 'class_weight': 'balanced'}.
Best is trial 2 with value: 0.7867467830621847.
[I 2026-02-13 01:50:53,123] Trial 4 finished with value: 0.7740806724594493
and parameters: {'criterion': 'entropy', 'max_depth': 12,
'min_samples_split': 32, 'min_samples_leaf': 19, 'class_weight': None}. Best
is trial 2 with value: 0.7867467830621847.
[I 2026-02-13 01:50:53,653] Trial 5 finished with value: 0.7781697668580498
and parameters: {'criterion': 'gini', 'max_depth': 8, 'min_samples_split':
34, 'min_samples_leaf': 8, 'class_weight': 'balanced'}. Best is trial 2 with
value: 0.7867467830621847.
[I 2026-02-13 01:50:54,111] Trial 6 finished with value: 0.7911302404301667
and parameters: {'criterion': 'entropy', 'max_depth': 3, 'min_samples_split':
33, 'min_samples_leaf': 15, 'class_weight': 'balanced'}. Best is trial 6 with
value: 0.7911302404301667.
[I 2026-02-13 01:50:54,534] Trial 7 finished with value: 0.7867467830621847
and parameters: {'criterion': 'entropy', 'max_depth': 3, 'min_samples_split':
14, 'min_samples_leaf': 7, 'class_weight': None}. Best is trial 6 with value:
0.7911302404301667.
[I 2026-02-13 01:50:55,459] Trial 8 finished with value: 0.7515119000380607
and parameters: {'criterion': 'gini', 'max_depth': 12, 'min_samples_split':
21, 'min_samples_leaf': 11, 'class_weight': None}. Best is trial 6 with
value: 0.7911302404301667.
[I 2026-02-13 01:50:56,231] Trial 9 finished with value: 0.782648375942407
and parameters: {'criterion': 'gini', 'max_depth': 8, 'min_samples_split':
18, 'min_samples_leaf': 16, 'class_weight': None}. Best is trial 6 with
value: 0.7911302404301667.
[I 2026-02-13 01:50:56,899] Trial 10 finished with value: 0.7945099484156227
and parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split':
29, 'min_samples_leaf': 1, 'class_weight': 'balanced'}. Best is trial 10 with
value: 0.7945099484156227.
[I 2026-02-13 01:50:57,407] Trial 11 finished with value: 0.7938483403111259
and parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split':
27, 'min_samples_leaf': 2, 'class_weight': 'balanced'}. Best is trial 10 with
value: 0.7945099484156227.
[I 2026-02-13 01:50:57,887] Trial 12 finished with value: 0.7938483403111259
and parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split':
27, 'min_samples_leaf': 1, 'class_weight': 'balanced'}. Best is trial 10 with
value: 0.7945099484156227.
[I 2026-02-13 01:50:58,374] Trial 13 finished with value: 0.7972187355754047
and parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split':
40, 'min_samples_leaf': 1, 'class_weight': 'balanced'}. Best is trial 13 with
value: 0.7972187355754047.
[I 2026-02-13 01:50:58,851] Trial 14 finished with value: 0.7976260659016738
and parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split':
40, 'min_samples_leaf': 5, 'class_weight': 'balanced'}. Best is trial 14 with
value: 0.7976260659016738.
[I 2026-02-13 01:50:59,344] Trial 15 finished with value: 0.7975629013580376
and parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split':
40, 'min_samples_leaf': 4, 'class_weight': 'balanced'}. Best is trial 14 with
value: 0.7976260659016738.
```

```
[I 2026-02-13 01:50:59,823] Trial 16 finished with value: 0.7976260659016738
and parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split':
39, 'min_samples_leaf': 5, 'class_weight': 'balanced'}. Best is trial 14 with
value: 0.7976260659016738.
[I 2026-02-13 01:51:00,315] Trial 17 finished with value: 0.7975629013580376
and parameters: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_split':
38, 'min_samples_leaf': 4, 'class_weight': 'balanced'}. Best is trial 14 with
value: 0.7976260659016738.
[I 2026-02-13 01:51:00,745] Trial 18 finished with value: 0.8046928016714311
and parameters: {'criterion': 'gini', 'max_depth': 5, 'min_samples_split':
29, 'min_samples_leaf': 4, 'class_weight': 'balanced'}. Best is trial 18 with
value: 0.8046928016714311.
[I 2026-02-13 01:51:01,132] Trial 19 finished with value: 0.8048341120927709
and parameters: {'criterion': 'gini', 'max_depth': 5, 'min_samples_split':
24, 'min_samples_leaf': 8, 'class_weight': 'balanced'}. Best is trial 19 with
value: 0.8048341120927709.
Optuna best CV roc_auc: 0.8048341120927709
Optuna best params: {'criterion': 'gini', 'max_depth': 5,
'min_samples_split': 24, 'min_samples_leaf': 8, 'class_weight': 'balanced'}
Optuna time (sec): 11.102
Saved: optuna_DecisionTree.csv | rows: 20

================================================================================
===
BEST PARAMS SUMMARY (CV)
================================================================================
===
              model       search  best_cv_roc_auc
best_params
LogisticRegression   GridSearch         0.842319
{'C': 0.1, 'class_weight': None, 'penalty': 'l1'}
LogisticRegression RandomSearch         0.842312
{'penalty': 'l1', 'class_weight': None, 'C': 0.19573417814876617}
LogisticRegression       Optuna         0.842299
{'C': 0.0745934328572655, 'penalty': 'l1', 'class_weight': None}
         LinearSVM   GridSearch         0.841613
{'C': 100, 'class_weight': 'balanced'}
         LinearSVM RandomSearch         0.841613
{'class_weight': 'balanced', 'C': 5.623413251903491}
         LinearSVM       Optuna         0.841617
{'C': 0.004982752357076452, 'class_weight': 'balanced'}
      DecisionTree   GridSearch         0.802938  {'class_weight':
'balanced', 'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 5,
'min_samples_split': 2}
      DecisionTree RandomSearch         0.803694 {'min_samples_split': 5,
'min_samples_leaf': 10, 'max_depth': 5, 'criterion': 'gini', 'class_weight':
'balanced'}
      DecisionTree       Optuna         0.804834 {'criterion': 'gini',
'max_depth': 5, 'min_samples_split': 24, 'min_samples_leaf': 8,
'class_weight': 'balanced'}

================================================================================
===
FINAL TEST COMPARISON (sorted by roc_auc)
================================================================================
===
```

```
                                model  n_features_after  fit_time_sec
inference_time_sec  accuracy       f1  roc_auc
  LogisticRegression + PCA | Grid best                20      0.169916
0.034183  0.800995 0.603399 0.828904
LogisticRegression + PCA | Random best                20      0.140862
0.008643  0.800995 0.604520 0.828876
LogisticRegression + PCA | Optuna best                20      0.174155
0.013310  0.800284 0.600284 0.828739
         LinearSVM + PCA | Random best                20      0.156667
0.016498  0.719972 0.600406 0.828014
          LinearSVM + PCA | Grid best                20      0.156015
0.008880  0.719972 0.600406 0.828011
         LinearSVM + PCA | Optuna best                20      0.136680
0.014392  0.719261 0.599797 0.827882
        DecisionTree + PCA | Grid best                20      0.227061
0.004410  0.701493 0.593810 0.788519
      DecisionTree + PCA | Random best                20      0.230056
0.005058  0.702203 0.594385 0.788401
      DecisionTree + PCA | Optuna best                20      0.231193
0.004549  0.701493 0.593810 0.787594
```

A) Logistic Regression

Grid best CV roc_auc: **0.842319**

best params: C=0.1, penalty=l1, class_weight=None

time: **6.848s**

Random best CV roc_auc: **0.842312**

best params: C≈0.1957, l1, None

time: **3.897s**

Optuna best CV roc_auc: **0.842299**

best params: C≈0.0745, l1, None

time: **4.029s**

هر سه روش تقریبا به یک نتیجه رسیدند یعنی سطح عملکرد LogisticRegression نسبت به این پارامترها یک ناحیه تخت دارد .

penalty = l1 در هر سه روش بهترین بوده یعنی sparse شدن وزن‌ها کمک کرده.

class_weight=None بهترین بوده چرا که با وجود عدم‌توازن کلاس‌ها، اینجا وزن‌دهی balanced به ROC-AUCکمک نکرده گاهی balanced به F1 کمک می‌کند ولی به ROC-AUC لزوما نه.

از نظر هزینه محاسباتی Grid کندتر از Random,Optuna است چون همه ترکیب‌ها را می‌رود . Random,Optunaسریع‌تر و تقریبا همان کیفیت را می‌دهند.

B) Linear SVM

Grid best CV roc_auc: **0.841613**

params: C=100, class_weight=balanced

time: **1.374s**

Random best CV roc_auc: **0.841613**

params: C≈5.623, balanced

time: **2.496s**

Optuna best CV roc_auc: **0.841617**

params: C≈0.00498, balanced

time: **4.945s**

باز هم CV ROC-AUC تقریبا ثابت است(0.841613)

اینکه Optuna یک C خیلی کوچک داده است ولی تقریبا همان ROC-AUC را گرفته به این معناست که یا داده در فضای PCA خیلی راحت جدا می‌شود یا ROC-AUC نسبت به مقیاس C حساسیت خیلی زیادی ندارد.

Grid اینجا از همه سریع‌تر شده چون فضای Grid کوچک بوده (فقط ۱۰ حالت).

C) Decision Tree

Grid best CV roc_auc: **0.802938**

params: depth=5, min_leaf=5, …

time: **32.102s**

Random best CV roc_auc: **0.803694**

time: **11.025s**

Optuna best CV roc_auc: **0.804834**

time: **11.102s**

درخت تصمیم خیلی پایین‌تر از دو مدل خطی است حدود 0.80 درCV

در Grid تعداد ترکیب‌ها ۴۸ تاست و هرکدام ۵-فولد یعنی حدود ۲۴۰ بارfit به همین دلیل کند است.

Optuna بهترین شده چون جستجو را هوشمندتر پیش می‌برد و لازم نیست همه حالت‌ها را brute forceبرود.

پارامترهای بهترین‌ها هم منطقی‌اند depth=5 و min_samples_leaf بزرگ‌تر یعنی جلوگیری از overfitting

(Final Test Comparison sorted by roc_auc)

بهترین‌ها از نظر roc_auc روی Test

```
LogisticRegression + PCA | Grid best
      roc_auc ≈ 0.828904
      accuracy ≈ 0.800995
      f1 ≈ 0.603399
      fit_time ≈ 0.130s
LogisticRegression + PCA | Optuna best
      roc_auc ≈ 0.828739
      f1 ≈ 0.600284 (کمی بهتر)
      inference_time ≈ 0.013310 (در Grid خیلی کمتر از)
LogisticRegression + PCA | Random best
      roc_auc ≈ 0.828876
```

بعد از آن‌ها LinearSVM با roc_auc ≈ 0.8280 می‌آید خیلی نزدیک، ولی accuracy حدود 0.72 افت دارد

و درخت تصمیم خیلی عقب‌تر

```
roc_auc ≈ 0.788–0.789
```

اینکه CV برای لاجستیک حدود 0.842 بوده ولی روی Test حدود 0.829 شده طبیعی است چرا که CVروی Train انجام شده و Test یک مجموعه جداست. اختلاف 0.01–0.02 کاملا معمول است. از طرفی اینکه هر سه نسخه Logistic (Grid/Random/Optuna) روی Test تقریباً برابرند یعنی تیونینگ اینجا بیشتر ریز تنظیم بوده و مدل پایه هم تقریبا همین عملکرد را داشته.

کدام مدل و کدام روش تیونینگ بهتر است؟

اگر معیار اصلی دقت باشد:

LogisticRegression + PCA بهترین و پایدارترین گزینه است هم accuracy ، هم f1 ، هم roc_aucدر Test از بقیه بهتر یا برابر است.

اگر معیار اصلی سادگی باشد:

GridSearch برای LogisticRegression خیلی قابل دفاع است چون فضای جستجو کوچک و مشخص است (۲۰ حالت) و بهترین پارامترها واضح‌اند.

اگر معیار اصلی هزینه محاسباتی تیونینگ باشد:

برایLogisticRegression ، RandomSearchیا Optuna تقریبا همان کیفیت را با زمان کمتر می‌دهند.

برایDecisionTree ، Optuna بهتر از Grid است چون Grid خیلی سنگین می‌شود.

فصل 9- بخش ششم

۶. **ارزیابی و تکرارپذیری (Evaluation & Reproducibility):**

- تنظیم دقیق Random State (یا Seed) در تمام بخش‌های کد (تقسیم داده، وزن‌دهی اولیه مدل‌ها و ...) برای تضمین تکرارپذیری کامل نتایج.
- توضیح شفاف نحوه تقسیم داده‌ها (Train/Validation/Test Split) و منطق پشت آن (مثلاً Strati-fied K-Fold).

هدف تکرارپذیری

در پروژه‌های یادگیری ماشین، بخش‌هایی مثل تقسیم داده، ساخت برخی مدل‌ها و بعضی الگوریتم‌های کاهش بعد می‌توانند رفتار تصادفی داشته باشند. اگر این تصادفی‌بودن کنترل نشود، با هر بار اجرا ممکن است نتایج کمی تغییر کند و مقایسه روش‌ها و مدل‌ها قابل اعتماد نباشد. بنابراین در این پروژه، مقدار ثابت 42 به عنوان random_state در تمام بخش‌های حساس تنظیم شد تا اجرای مجدد کد، خروجی یکسان تولید کند و نتایج قابل بازتولید باشند.

بخش‌هایی در کد ما که تکرارپذیری در آن‌ها کنترل شد:

در کل کد، مقدار random_state برابر 42 در نقاط زیر اعمال شده است

- **تقسیم داده‌ها**

در train_test_split مقدار random_state=42 تنظیم شد.
همچنین stratify=y فعال شد تا نسبت کلاس‌ها در هر دو بخش آموزش و آزمون ثابت و مشابه کل داده باقی بماند. این کار به ویژه برای مسئله Churn که معمولا نامتوازن است ضروری است.

- **اعتبارسنجی در مرحله تنظیم ابرپارامترها**

برای تیونینگ از StratifiedKFold استفاده شد.

با shuffle=True و random_state=42 ترتیب نمونه‌ها قبل از ساخت فولدها ثابت شد.
چون StratifiedKFold است، توزیع کلاس‌ها در همه فولدها مشابه می‌ماند و مقایسه تنظیمات مختلف منصفانه‌تر می‌شود.

- **مدل‌ها**

برای LogisticRegression مقدار random_state=42 تنظیم شد تا هرگونه تصادفی‌بودن داخلی کنترل شود.
برای DecisionTreeClassifier مقدارrandom_state=42 را تنظیم کردیم چون درخت تصمیم می‌تواند در برخی حالت‌ها رفتار تصادفی داشته باشد.
برای LinearSVC نیز random_state=42 گذاشته شد تا اگر در روند بهینه‌سازی یا شافل داخلی تصادفی‌بودن وجود داشت، کنترل شود.

- **کاهش بعد**

برای PCA مقدار random_state=42 را قرار دادیم.
اگر در جایی از t-SNE استفاده شود، random_state=42 نیز باید تنظیم شود چون t-SNE به مقداردهی اولیه و روند تصادفی حساس است.

- **جلوگیری از نشت داده**

تمام مراحل پیش‌پردازش مانندOne-Hot Encoding ، استانداردسازی و PCA فقط روی داده آموزش فیت شدند و سپس همان تبدیل‌ها روی داده آزمون اعمال شد.
این کار باعث می‌شود اطلاعات آزمون وارد آموزش نشود و ارزیابی روی داده دیده‌نشده واقعی‌تر باشد.

ما داده را تقسیم می‌کنیم تا بتوانیم تعمیم‌پذیری مدل را بسنجیم. یعنی مدل را روی بخشی از داده آموزش می‌دهیم و روی بخشی دیگر که در آموزش دیده نشده، ارزیابی می‌کنیم. این کار جلوی این را می‌گیرد که مدل فقط داده را حفظ کند و روی داده جدید ضعیف باشد.

بخش اول Train و Test Split

Trainبرای یادگیری مدل و انجام تنظیمات و انتخاب مدل استفاده می‌شود.

Testباید کاملاً کنار گذاشته شود و فقط در انتها برای گزارش نهایی استفاده شود.

نسبت 20/80 یک انتخاب رایج است زیرا داده کافی برای آموزش باقی می‌ماند و در عین حال یک مجموعه تست مستقل برای ارزیابی واقعی داریم

اگر در حین انتخاب هایپرپارامتر یا تصمیم‌گیری‌های مدل‌سازی از Test استفاده کنیم، مدل به صورت غیرمستقیم روی Test تنظیم می‌شود و نتیجه تست دیگر واقعی نیست. این مشکل را Data Leakage یا نشت اطلاعات می‌گویند.

بخش دوم stratify

در مسئله Churn معمولا تعداد کلاس‌ها برابر نیست. معمولا تعداد No بیشتر از Yes است. اگر تقسیم داده را کاملا تصادفی انجام دهیم ممکن است. در Train تعداد Yes خیلی کمتر یا بیشتر از حالت واقعی شود یا در Test تعداد Yes خیلی کم شود این باعث می‌شود ارزیابی غلط شود. مثلا اگر در Test تعداد Yesکم شود، مدل ممکن است ظاهرا Accuracy خوبی بگیرد چون بیشتر No پیش‌بینی می‌کند، ولی در واقع برای تشخیص churn ضعیف است.

وقتی می‌گوییم stratify=y یعنی در Train و Test نسبت کلاس‌ها تقریبا مثل نسبت کلاس‌ها در کل دیتاست باقی می‌ماند یعنی درصد Yes و No در هر دو بخش نزدیک به هم و نزدیک به کل داده می‌ماند

درنتیجه استفاده ازstratify ارزیابی منصفانه‌تر و پایدارتر می‌شود مقایسه مدل‌ها قابل اعتمادتر می‌شود و معیارهایی مثل ROC-AUC و F1 کمتر دچار نوسان می‌شوند مخصوصا در داده‌های نامتوازن، stratify تقریبا ضروری است

بخش سوم Train Validation Test Split

ما یک Validation ثابت جداگانه نساختیم. به جای آن این کار مراحل زیر را انجام دادیم یک بار داده را به Train و Test تقسیم کردیم و برای ساخت Validation از داخل Train از روش-Cross Validationاستفاده کردیم. یعنی Testثابت و دست نخورده می‌ماند Validationبه صورت چرخشی داخل Train ساخته می‌شود
این روش بهتر از ساخت یک validation ثابت است، چون از داده‌ی Train بهتر استفاده می‌کند و برآورد عملکرد پایدارتر است.

بخش چهارم Stratified K-Fold

درK-Fold ، داده Train به K قسمت تقسیم می‌شود. اگر K برابر 5 باشد:
در هر تکرار 4 قسمت برای آموزش

1 قسمت برای اعتبارسنجی
این کار 5 بار تکرار می‌شود تا هر قسمت یک بار نقش validation را داشته باشد
در نهایت عملکرد مدل روی validation برابر است با میانگین عملکرد در همه فولدها.

از Stratified K-Fold استفاده کردیم. چرا که همان منطق stratify اینجا هم مهم است. اگر فولدها stratified نباشند ممکن است یک فولد تعداد churn خیلی کم داشته باشد، یک فولد تعداد churn خیلی زیاد داشته باشد این باعث نوسان شدید و انتخاب اشتباه هایپرپارامتر می‌شود.

پس Stratified K-Fold تضمین می‌کند در هر فولد هم نسبت کلاس‌ها حفظ شود.

shuffle=True

اگر داده‌ها به هر دلیلی مرتب شده باشند، مثلاً رکوردهای مشابه کنار هم آمده باشند، تقسیم بدون shuffle می‌تواند فولدهای غیرنماینده بسازد .داده را قبل از فولد بندی مخلوط می‌کند تا فولدها نماینده‌تر شوند.

n_splits=5

5 تعادل خوبی بین دقت برآورد و هزینه محاسباتی است فولدهای کمتر مثل 3 ممکن است ناپایدارتر باشند فولدهای بیشتر مثل 10 هزینه را زیاد می‌کند

بخش پنجم random_state در تقسیم داده وCV

تقسیم داده و shuffle ذاتا تصادفی هستند. اگر random_state ثابت نگذاریم هر بار اجرای کد، Train و Test متفاوت می‌شوند و نتیجه مدل و تیونینگ متفاوت می‌شود

وقتی random_state=42 می‌گذاریم، تقسیم‌ها دقیقاً تکرارپذیر می‌شوند و نتایج بین اجراها ثابت می‌مانند در نهایت مقایسه بین مدل‌ها و روش‌ها منصفانه‌تر می‌شود. که ما این کار را هم در train_test_splitو هم در StratifiedKFold رعایت کردیم، پس نتایج قابل بازتولید هستند.

فصل 10- بخش هفتم

۷. تحلیل نتایج و مصورسازی (Visualization & Analysis):

- مقایسه نتایج حداقل دو روش انتخاب شده با معیارهای مختلف (مانند Precision، Recall، F1-Score، ROC-AUC).
- رسم تمامی نمودارهای ممکن و مرتبط (مانند Confusion Matrix، Learning Curves، Loss Curves، ROC Curve).
- تحلیل و تفسیر عمیق نتایج (چرا یک مدل بهتر عمل کرد؟ چرا مدل دچار Overfitting شد یا نشد؟).

در این قسمت دیگر تمام موارد مورد نیاز را داریم و فقط نیاز داریم مقایسه ای بین هر سه مدل انجام دهیم و نمودارهای لازم را برای انجام این مقایسه رسم کنیم.

```
#7
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score,
    confusion_matrix, ConfusionMatrixDisplay,
    roc_curve, precision_recall_curve, average_precision_score
)

from sklearn.model_selection import StratifiedKFold, learning_curve,
validation_curve

best_params = {
    "LogisticRegression": {"C": 0.0745934328572655, "penalty": "l1",
"class_weight": None},
    "LinearSVM":          {"C": 0.004982752357076452, "class_weight":
"balanced"},
    "DecisionTree":       {"criterion": "gini", "max_depth": 5,
"min_samples_split": 24,
                           "min_samples_leaf": 8, "class_weight":
"balanced"}
}

Xte_scaled = scaler.transform(X_test_ohe)
Xte_pca = pca.transform(Xte_scaled)
```

```python
def get_pred_and_score(model, X):
    if hasattr(model, "predict_proba"):
        score = model.predict_proba(X)[:, 1]
        pred = (score >= 0.5).astype(int)
    else:
        score = model.decision_function(X)
        pred = (score >= 0).astype(int)
    return pred, score

final_models = {}
final_metrics = []

for key in ["LogisticRegression", "LinearSVM", "DecisionTree"]:
    model = build_model(key, best_params[key])
    model.fit(Xtr_pca, y_train)
    final_models[key] = model

    pred, score = get_pred_and_score(model, Xte_pca)

    final_metrics.append({
        "model": key,
        "accuracy":  accuracy_score(y_test, pred),
        "precision": precision_score(y_test, pred, zero_division=0),
        "recall":    recall_score(y_test, pred, zero_division=0),
        "f1":        f1_score(y_test, pred, zero_division=0),
        "roc_auc":   roc_auc_score(y_test, score),
        "avg_precision": average_precision_score(y_test, score)
    })

metrics_df = pd.DataFrame(final_metrics).sort_values("roc_auc",
ascending=False)
print("\n================= FINAL METRICS (TEST) =================")
print(metrics_df.to_string(index=False))

metrics_df.to_csv("final_metrics_after_tuning.csv", index=False)
print("\nSaved: final_metrics_after_tuning.csv")


# ========================================
# Confusion Matrix
# ========================================
for key, model in final_models.items():
    pred, score = get_pred_and_score(model, Xte_pca)
    cm = confusion_matrix(y_test, pred)
```

```
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)

    plt.figure()
    disp.plot(values_format="d")
    plt.title(f"Confusion Matrix - {key} (Tuned)")
    plt.show()


# ========================================
# ROC Curves
# ========================================
plt.figure()
for key, model in final_models.items():
    pred, score = get_pred_and_score(model, Xte_pca)
    fpr, tpr, _ = roc_curve(y_test, score)
    auc = roc_auc_score(y_test, score)
    plt.plot(fpr, tpr, label=f"{key} (AUC={auc:.3f})")

plt.plot([0, 1], [0, 1], linestyle="--", label="Random")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - Tuned Models")
plt.legend()
plt.show()


# ========================================
#  Precision-Recall Curves
# ========================================
plt.figure()
for key, model in final_models.items():
    pred, score = get_pred_and_score(model, Xte_pca)
    prec, rec, _ = precision_recall_curve(y_test, score)
    ap = average_precision_score(y_test, score)
    plt.plot(rec, prec, label=f"{key} (AP={ap:.3f})")

plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curves - Tuned Models")
plt.legend()
plt.show()


# ========================================
# Learning Curves (train size vs score)
```

```
# ========================================
cv_lc = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

def plot_learning_curve(estimator, title):
    train_sizes, train_scores, val_scores = learning_curve(
        estimator,
        Xtr_pca, y_train,
        cv=cv_lc,
        scoring="roc_auc",
        train_sizes=np.linspace(0.1, 1.0, 5),
        n_jobs=-1
    )

    train_mean = train_scores.mean(axis=1)
    train_std  = train_scores.std(axis=1)
    val_mean   = val_scores.mean(axis=1)
    val_std    = val_scores.std(axis=1)

    plt.figure()
    plt.plot(train_sizes, train_mean, label="Train ROC-AUC")
    plt.plot(train_sizes, val_mean, label="CV ROC-AUC")
    plt.fill_between(train_sizes, train_mean-train_std,
train_mean+train_std, alpha=0.2)
    plt.fill_between(train_sizes, val_mean-val_std, val_mean+val_std,
alpha=0.2)
    plt.xlabel("Training set size")
    plt.ylabel("ROC-AUC")
    plt.title(title)
    plt.legend()
    plt.show()

for key, model in final_models.items():
    plot_learning_curve(model, f"Learning Curve - {key} (Tuned)")


# ========================================
#    Validation Curve
# ========================================

def plot_validation_curve(estimator, param_name, param_range, title,
xscale=None):
    train_scores, val_scores = validation_curve(
        estimator,
        Xtr_pca, y_train,
        param_name=param_name,
```

```
        param_range=param_range,
        cv=cv_lc,
        scoring="roc_auc",
        n_jobs=-1
    )

    train_mean = train_scores.mean(axis=1)
    train_std  = train_scores.std(axis=1)
    val_mean   = val_scores.mean(axis=1)
    val_std    = val_scores.std(axis=1)

    plt.figure()
    plt.plot(param_range, train_mean, label="Train ROC-AUC")
    plt.plot(param_range, val_mean, label="CV ROC-AUC")
    plt.fill_between(param_range, train_mean-train_std,
train_mean+train_std, alpha=0.2)
    plt.fill_between(param_range, val_mean-val_std, val_mean+val_std,
alpha=0.2)
    if xscale is not None:
        plt.xscale(xscale)
    plt.xlabel(param_name)
    plt.ylabel("ROC-AUC")
    plt.title(title)
    plt.legend()
    plt.show()

# LogisticRegression: curve
lr_for_curve = build_model("LogisticRegression",
best_params["LogisticRegression"])
C_range = np.logspace(-3, 2, 8)
plot_validation_curve(lr_for_curve, "C", C_range, "Validation Curve -
LogisticRegression (C)", xscale="log")

# LinearSVM: curve
svm_for_curve = build_model("LinearSVM", best_params["LinearSVM"])
plot_validation_curve(svm_for_curve, "C", C_range, "Validation Curve -
LinearSVM (C)", xscale="log")

# DecisionTree: curve
dt_for_curve = build_model("DecisionTree", best_params["DecisionTree"])
depth_range = np.array([1, 2, 3, 5, 8, 12, 16])
plot_validation_curve(dt_for_curve, "max_depth", depth_range, "Validation
Curve - DecisionTree (max_depth)")
```

خروجی

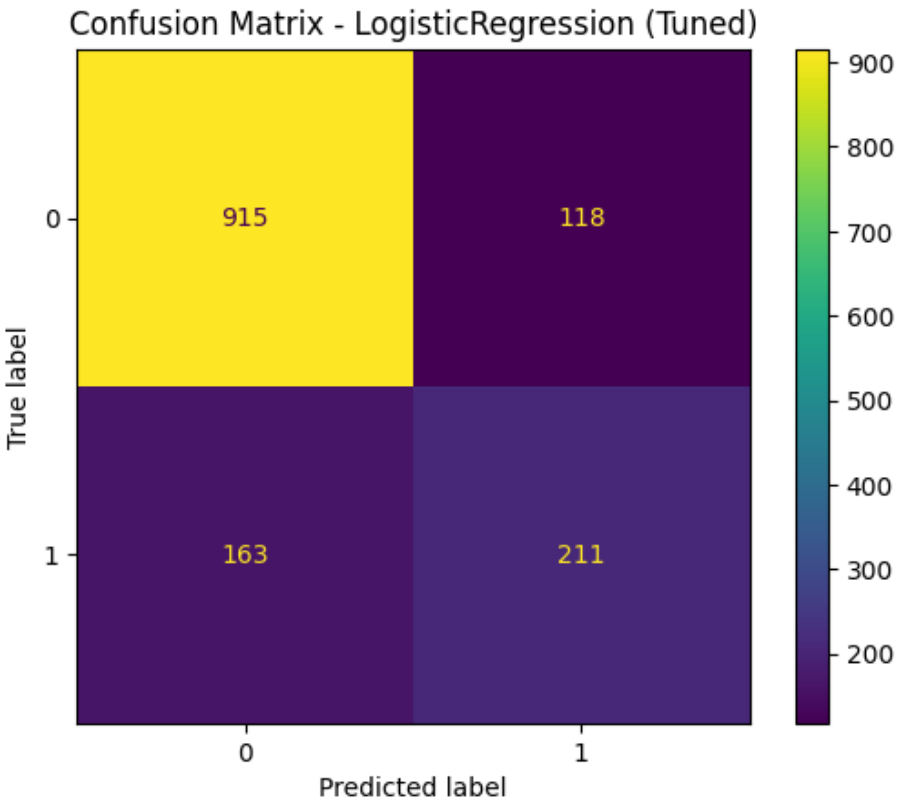
```
================ FINAL METRICS (TEST) ================
              model  accuracy  precision   recall       f1  roc_auc
avg_precision
LogisticRegression  0.800284   0.641337 0.564171 0.600284 0.828739
0.606837
         LinearSVM  0.719261   0.482871 0.791444 0.599797 0.827882
0.606251
      DecisionTree  0.701493   0.465152 0.820856 0.593810 0.787594
0.525024

Saved: final_metrics_after_tuning.csv
<Figure size 640x480 with 0 Axes>
```

- از نظر ROC AUC و Avg Precision، لاجستیک و SVM تقریبا هم‌سطح هستند و هر دو واضحا از درخت تصمیم بهترند.
- از نظر Accuracy، لاجستیک بهترین است اما این به معنی بهترین بودن کلی نیست، چون دیتاست نامتوازن است و accuracy به تصمیم آستانه خیلی حساس است.
- از نظر Recall یعنی شکار کردنchurn ها درخت و SVM خیلی بهتر از لاجستیک‌اند.
- از نظر Precision یعنی کمتر اشتباه آلارم دادن لاجستیک بهتر از دو مدل دیگر است.

اگر هدف این است که تا حد ممکن churn ها از دست نروند SVM یا درخت بهترند.

اگر هدف این است که کمتر به مشتری‌های غیر churn اشتباه برچسب churn بزنیم لاجستیک بهتر است.

کلاس 1 یعنی Churn و کلاس 0 یعنی Non-Churn



LogisticRegression

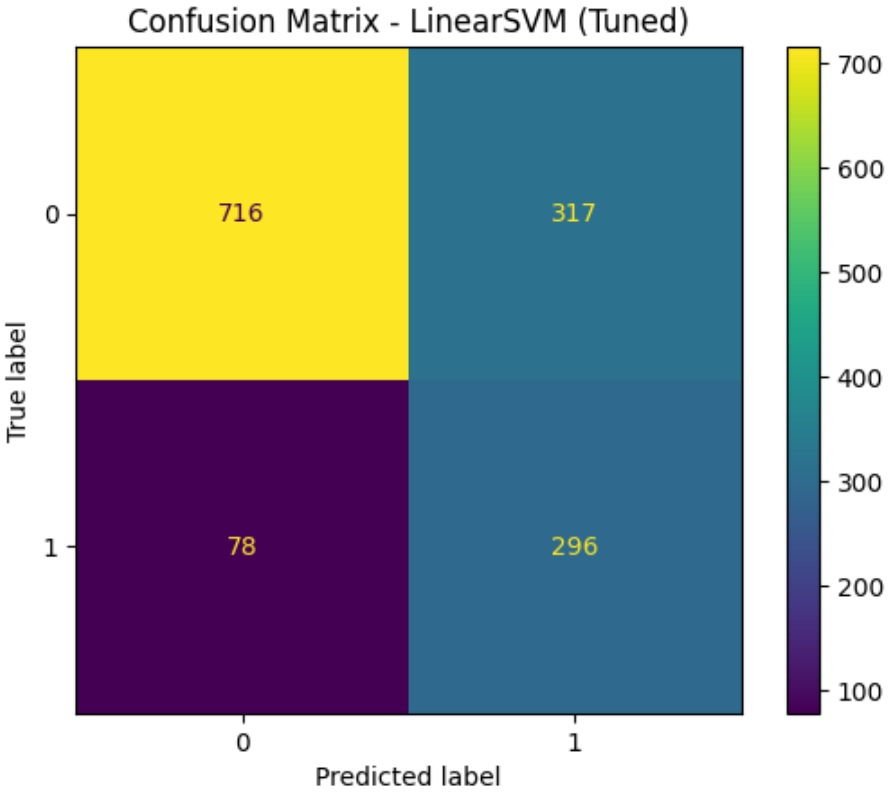
FP=118 ، TN=915

TP=211 ، FN=163

False Positive کم دارد (118) یعنی به افراد زیادی اشتباه churn نمی‌گوید و در نتیجه precision بالا می‌رود

False Negative زیاد دارد (163) یعنیchurn های زیادی را از دست می‌دهد recall پس پایین می‌آید

این مدل محافظه‌کار است، کمتر "هشدار churn"می‌دهد، برای همین churn واقعی بیشتری را جا می‌اندازد.



LinearSVM

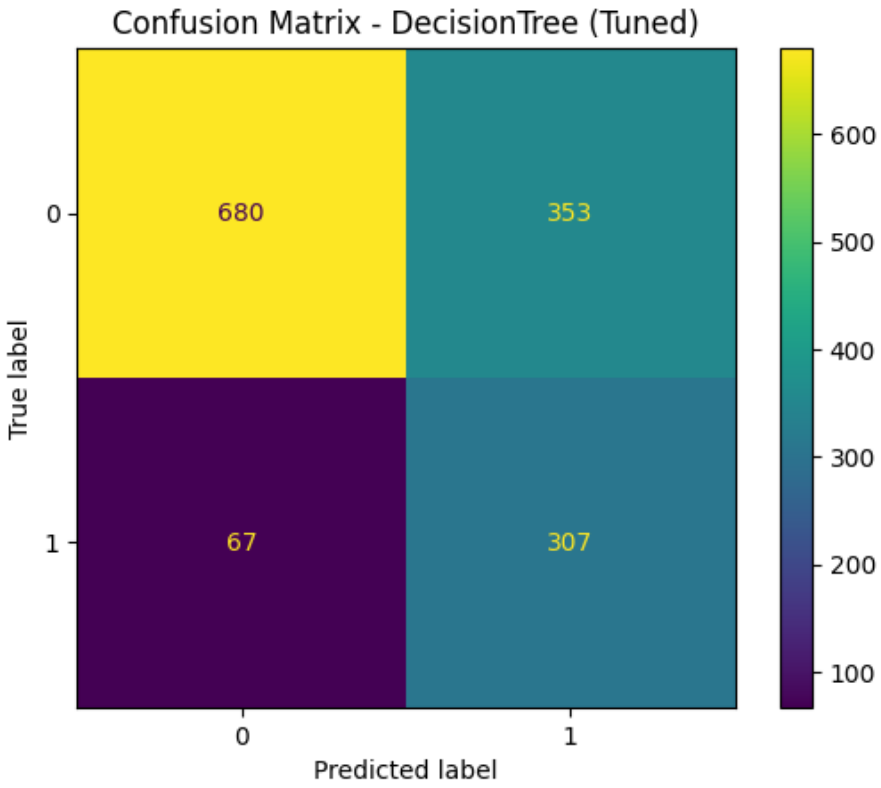
FP=317 ، TN=716

TP=296 ، FN=78

FPزیادتر شده (317) یعنی افراد-non churnزیادی را churn اعلام می‌کند و precisionافت می‌کند

FNکم شده (78) یعنیchurn های کمتری را از دست می‌دهد recall بالا می‌رود

این مدل تهاجمی‌تر است، زیاد هشدار churn می‌دهد، پس churn بیشتری را می‌گیرد ولی هزینه هشدار اشتباه بالاتر می‌رود.



DecisionTree

FP=353 ، TN=680

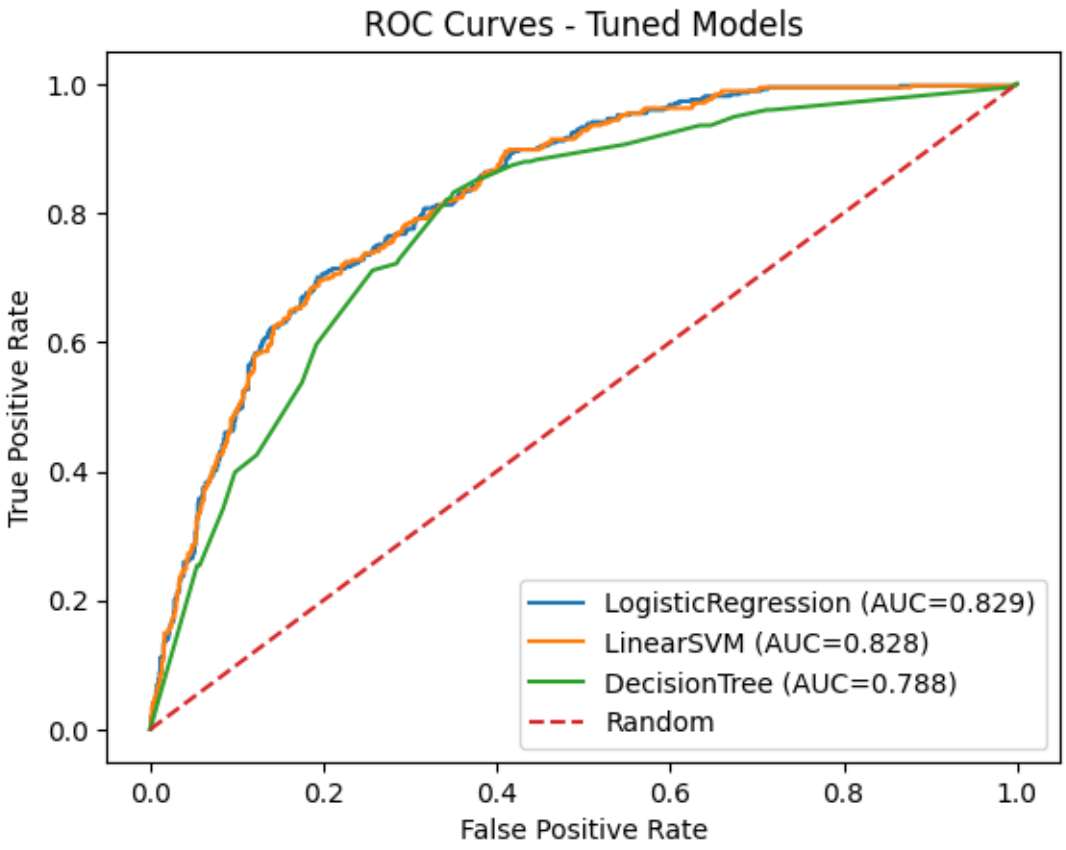
TP=307 ، FN=67

بالاترین recall را دارد چون FN کمترین است

اما FP هم زیاد است، یعنی precision پایین می‌ماند

درخت تصمیم خیلی راحت مثبت اعلام می‌کند و churn زیادی را می‌گیرد، ولی مشتری‌های زیادی را اشتباه churn فرض می‌کند.

در دیتاست ما تعداد non-churn بیشتر است. وقتی یک مدل FP را زیاد می‌کند، accuracyسریع افت می‌کند، حتی اگر recall بالا برود. دقیقا به همین دلیل لاجستیک accuracy بالاتر دارد چون FP خیلی کم دارد و SVM و درخت accuracy پایین‌تر دارند چون FP زیاد تولید می‌کنند.



ROC Curve

LogisticRegression AUC ≈ 0.829

LinearSVM AUC ≈ 0.828

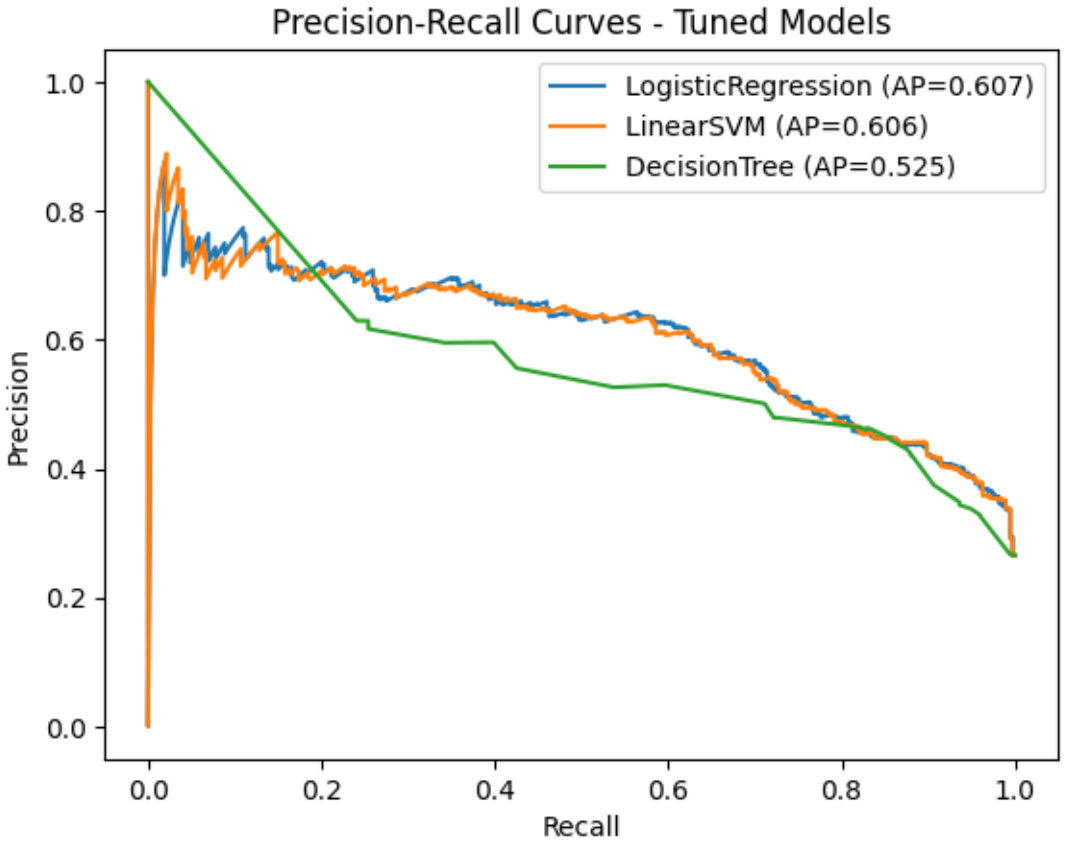
DecisionTree AUC ≈ 0.788

لاجستیک و SVM تقریبا یک توان تفکیک‌پذیری دارند. یعنی اگر آستانه تصمیم را تغییر بدهیم، هر دو می‌توانند trade-off های مشابهی تولید کنند.

درخت تصمیم به طور کلی قدرت رتبه‌بندی و جداسازی ضعیف‌تری دارد، و این در AUC پایین‌تر مشخص است.

ROC AUC مستقل از آستانه 0.5 است.

این که لاجستیک accuracy بالاتری دارد و SVM recall بالاتر، بیشتر به آستانه تصمیم و شکل خروجی نمره‌ها مربوط است، نه این که یکی ذاتا خیلی قوی‌تر باشد.



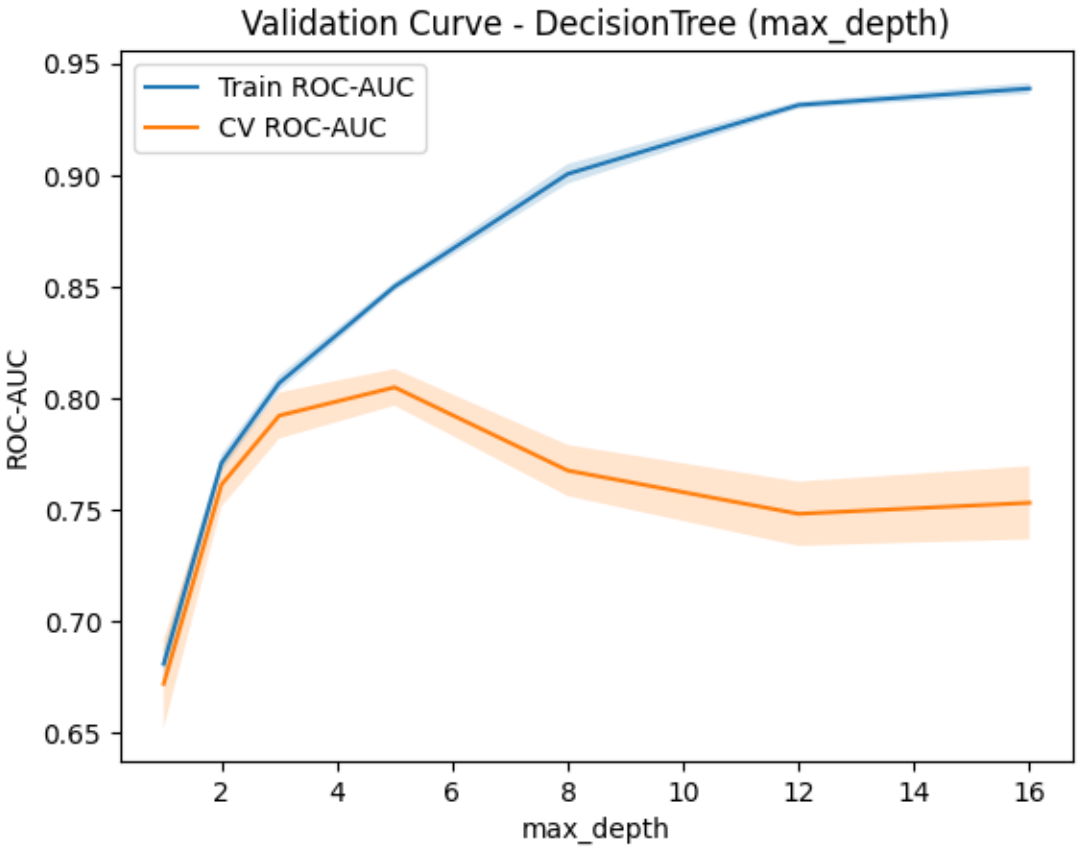
Precision–Recall Curve

AP لاجستیک ≈ 0.607

AP SVM ≈ 0.606

AP درخت ≈ 0.525

چون کلاس churn اقلیت است، PR curve معمولا از ROC برای تصمیم‌گیری عملی بهتر است.
باز هم لاجستیک و SVM تقریبا برابرند.
درخت تصمیم پایین‌تر است یعنی وقتی recallرا زیاد کنیم، precision سریع‌تر سقوط می‌کند.

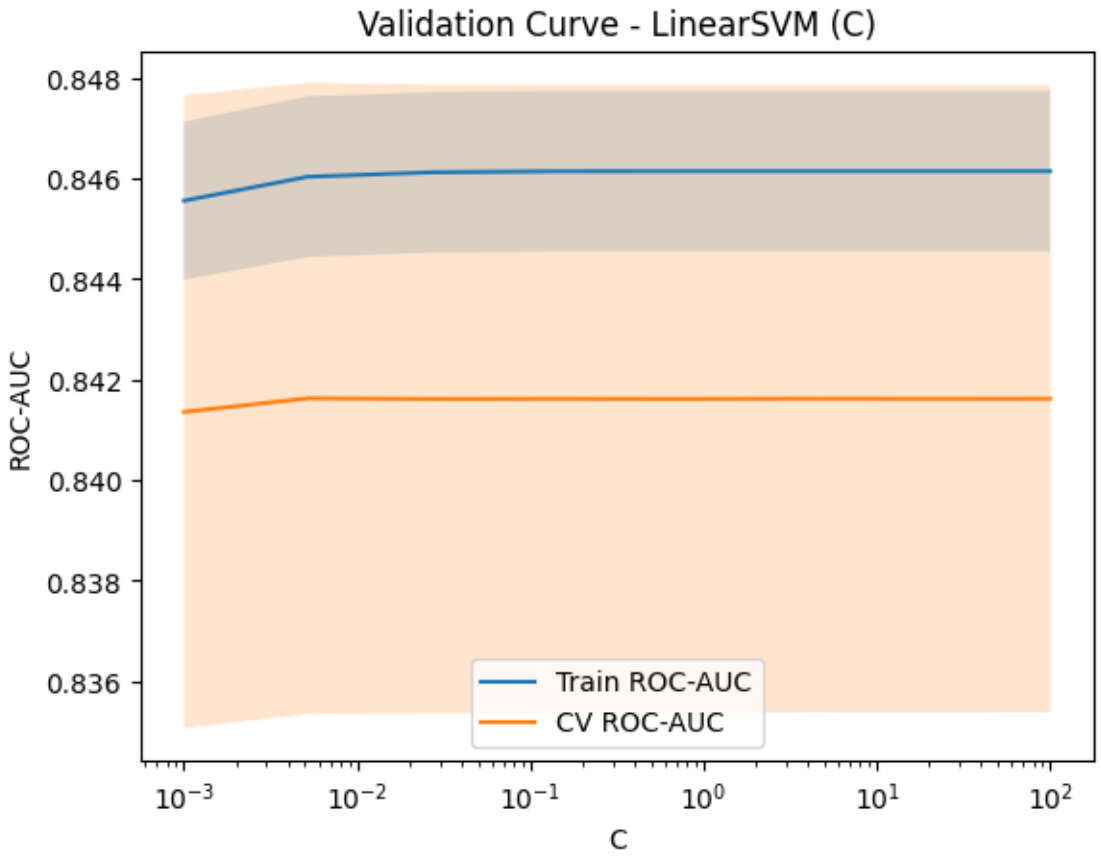


درخت تصمیم برحسب max_depth

در نمودار ما Train AUC با افزایش depth مدام بالا می‌رود

CV AUC تا حدود depth ≈ 5 بهتر می‌شود و بعد افت می‌کند این دقیقا تعریف کلاسیک overfitting است:

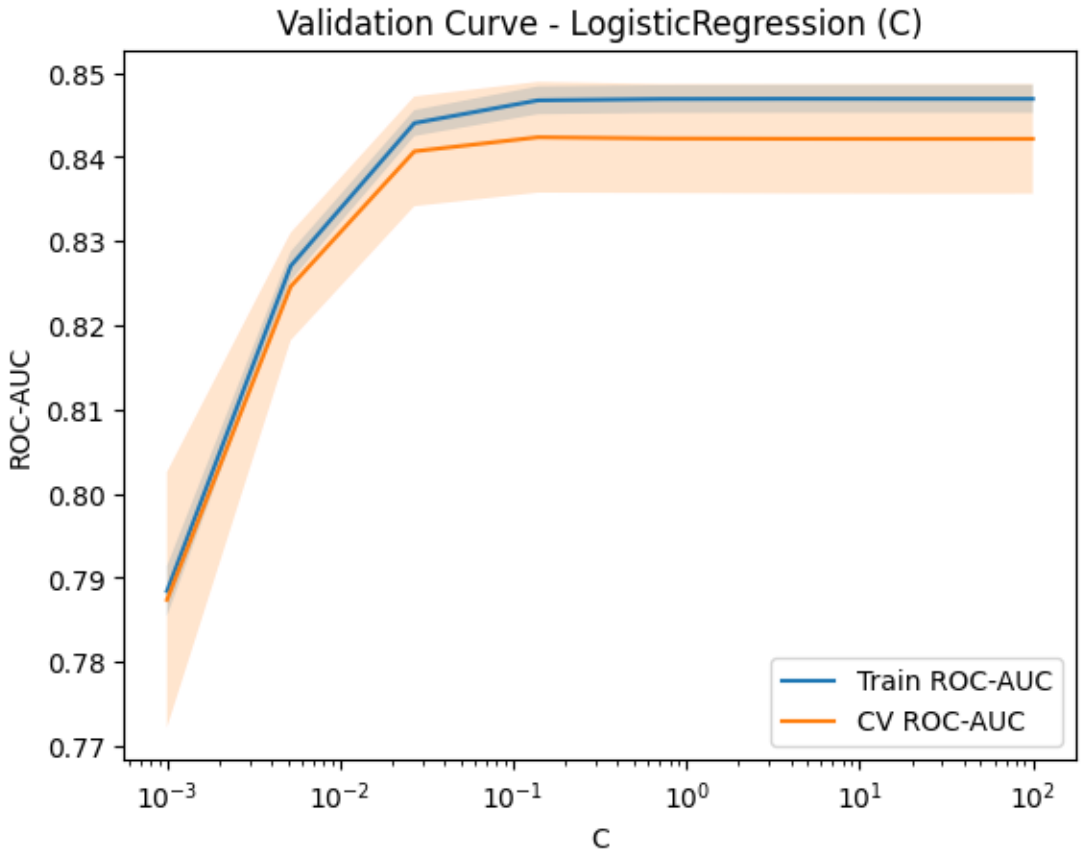
عمق زیاد باعث می‌شود مدل جزئیات و نویز را حفظ کند و train عالی شود ولی روی داده جدید بدتر می‌شود و CV افت می‌کند پس انتخاب depth=5 کاملا منطقی است و نمودار آن را تایید می‌کند.



LinearSVM برحسب C

منحنی تقریبا تخت است یعنی عملکرد مدل نسبت به C در این بازه خیلی حساس نیست این معمولا به این معناست یا داده بعد از پیش‌پردازش خیلی خوب جدا می‌شود یا ناحیه بهینه پهن است و چند مقدار C نزدیک به هم جواب مشابه می‌دهند

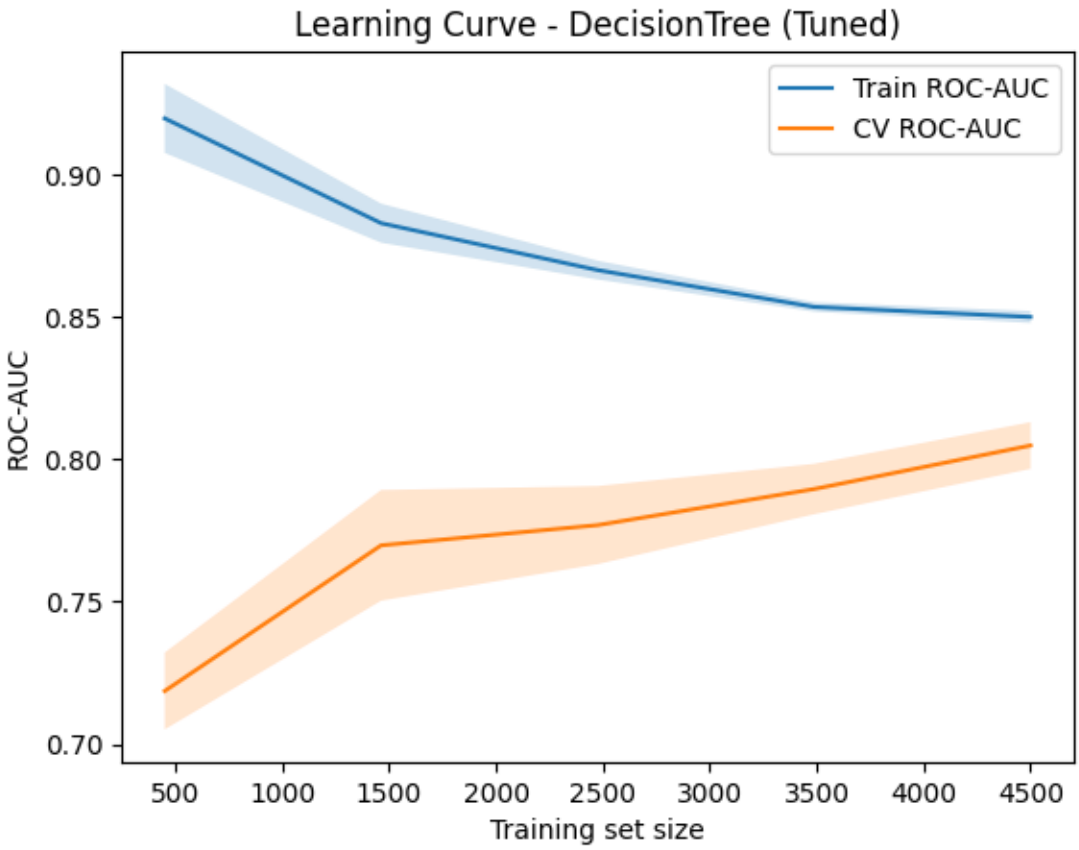
این با خروجی تیونینگ هم جور است که Grid و Random و Optuna همگی CV نزدیک به هم داده‌اند.



LogisticRegression برحسب C

نمودار به ما نشان می‌دهد در C خیلی کوچک مدل underfit است و CV AUC پایین‌تر است. با افزایش C عملکرد بهتر می‌شود و بعد به یک plateau می‌رسد

یعنی بعد از یک نقطه، کاهش regularization دیگر سودی ندارد و مدل به سقف توان خودش روی این ویژگی‌ها می‌رسد.



DecisionTree

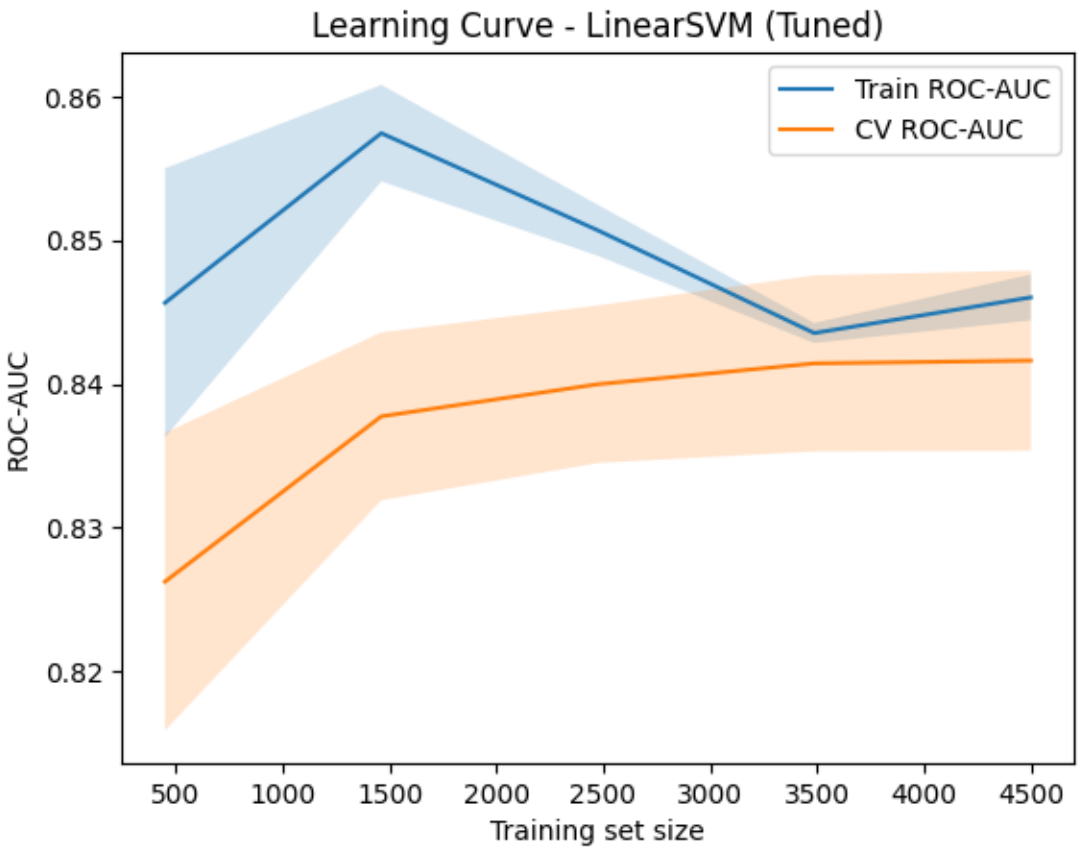
در نمودار Train AUC بالا شروع می‌شود و با داده بیشتر پایین می‌آید. CV AUC با داده بیشتر بالا می‌رود و فاصله Train و CV زیاد می‌ماند

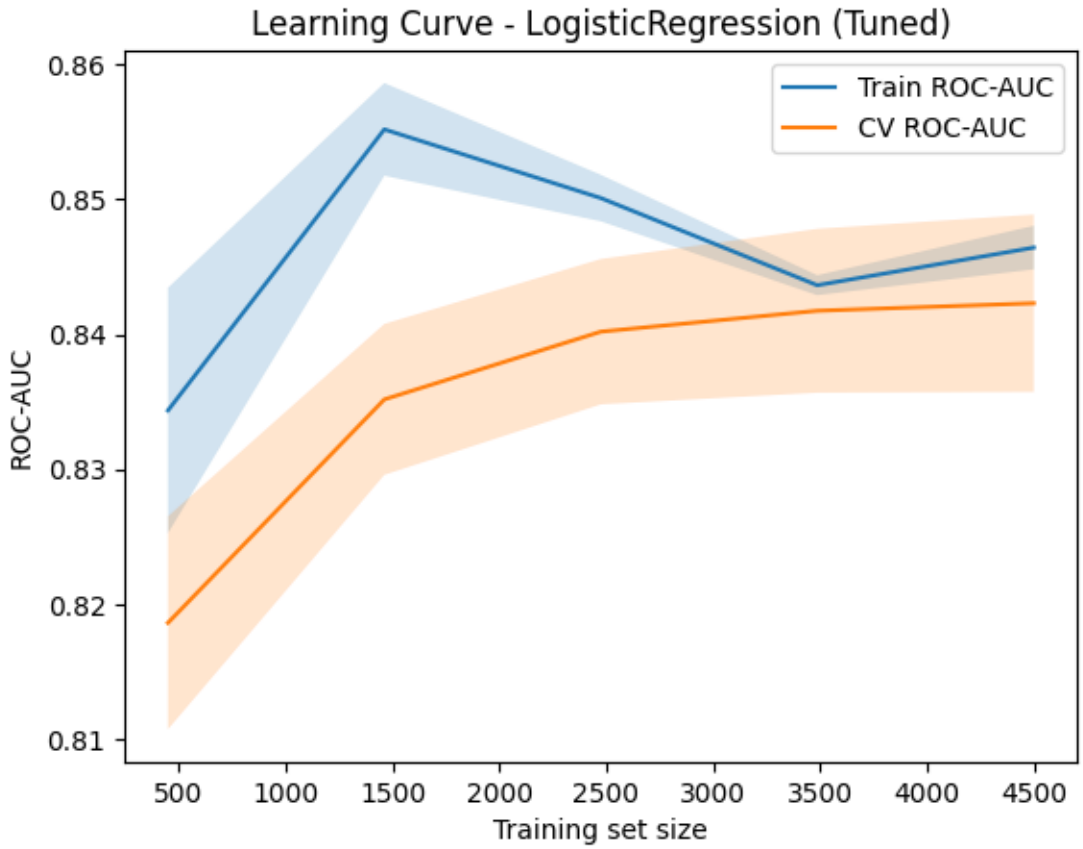
این یعنی variance بالا و مدل هنوز overfit است. داده بیشتر کمک می‌کند ولی مشکل را کامل حل نمی‌کند.

برای حل این مشکل می‌توانیم راه حل های زیر را انجام دهیم.

برای محدود کردن بیشتر پیچیدگی: max_depth کمتر، min_samples_leaf بیشتر، min_samples_split بیشتر

یا رفتن به سمت ensemble ها مثل RandomForest یا Gradient Boosting که variance درخت را کنترل می‌کنند


Learning Curve - LinearSVM (Tuned)
Train ROC-AUC
CV ROC-AUC
ROC-AUC
0.86
0.85
0.84
0.83
0.82
500
1000
1500
2000
2500
3000
3500
4000
4500
Training set size


Learning Curve - LogisticRegression (Tuned)
Train ROC-AUC
CV ROC-AUC
ROC-AUC
0.86
0.85
0.84
0.83
0.82
0.81
500
1000
1500
2000
2500
3000
3500
4000
4500
Training set size

LinearSVM و LogisticRegression

هر دو CV AUC با داده بیشتر بهتر می‌شود و بعد آرام آرام نزدیک سقف می‌شود فاصله train و CV خیلی بزرگ نیست این یعنی generalization بهتر از درخت است و مدل‌ها پایدارترند.
همچنین نشان می‌دهد که توان مدل تا حد خوبی استفاده شده و اگر improvement بخواهیم، معمولا باید: ویژگی‌ها بهتر شوند یا مدل غیرخطی‌تر و قوی‌تر بیاید نه صرفا تغییر جزئی یک هایپرپارامتر

دلایل بهتر شدن لاجستیک و SVM نسبت به درخت تصمیم

- **ماهیت داده و ویژگی‌ها**
  داده بعد از one-hot و scaling و مخصوصا بعد از PCA ، خیلی مناسب مدل‌های خطی است.
  Logistic و LinearSVM دقیقا برای همین نوع فضاها عالی‌اند.
- **درخت تصمیم مستعد overfitting است**
  حتی با تیونینگ، درخت منفرد معمولا variance بالاتری دارد و learning curve دقیقا این را نشان می‌دهد.
- **اثر PCA روی درخت**
  PCA ویژگی‌ها را به ترکیب‌های خطی پیوسته تبدیل می‌کند. درخت تصمیم split های محورهمراستا می‌زند و در چنین فضاهایی معمولا به قدرت مدل‌های خطی نمی‌رسد، مگر با تنظیمات خیلی دقیق یا با ensemble

به طور کلی

- اگر معیار اصلی ROC AUC یا PR AUC باشد ⟵ لاجستیک و SVM تقریبا برابرند و هر دو بهتر از درخت‌اند.
- اگر هدف کم کردن هشدار اشتباه باشد LogisticRegression ⟵ بهتر است چون FP خیلی کمتر دارد.
- اگر هدف گرفتن churn بیشتر باشد DecisionTree ⟵ یا LinearSVM بهترند چون recall بالاتر دارند.

اگر بخواهیم یک انتخاب متعادل و صنعتی‌تر داشته باشیم:

- معمولا لاجستیک به خاطر پایداری، سادگی، تفسیرپذیری و precision بهتر انتخاب پیش‌فرض است بعد آستانه تصمیم را بر اساس هزینه کسب‌وکار تنظیم می‌کنند تا recall هم بالا بیاید

مدل درخت تصمیم نشانه بیش برازش داشت اما با تیونینگ آن را کنترل کردیم.

Validation curve مربوط به max_depth دقیقا الگوی کلاسیک بیش‌برازش است.

- با زیاد شدن عمق، Train ROC-AUCمدام بالا می‌رود (تا حوالی 0.93–0.94). ولی CV ROC-AUC بعد از یک نقطه حدود depth≈5 بهترین می‌شود و بعد آن افت می‌کند تا حدود 0.75.

این یعنی درخت وقتی عمیق می‌شود، روی داده‌ی Train خیلی خوب حفظ می‌کند ولی روی داده‌ی جدید (Validation/CV)بدتر عمل می‌کند یعنی **Overfitting**

آیا مدل نهایی درخت تصمیم هنوز overfit است؟

- ما بهترین پارامترها رو طوری انتخاب کردیم که درخت **محدود** بشود مثلا max_depth=5 و min_samples_leafبزرگ‌تر و min_samples_split بزرگ‌تر با این کار، overfittingشدیدِ درخت‌های عمیق کنترل شده. اما Learning curveدرخت تیون‌شده هنوز یک فاصله‌ی قابل توجه بین Train و CV نشان می‌دهد:
  - Train ROC-AUCهمچنان بالاتر می‌ماند (حدود 0.85 به بالا)
  - CV ROC-AUCپایین‌تر است و با داده بیشتر آرام آرام بالا می‌رود (حدود 0.80)

در نتیجه هنوز کمی بیش‌برازش وجود دارد (ذات درخت تصمیم هم همین است)، ولی نسبت به حالت‌های عمیق‌تر خیلی بهتر و قابل قبول‌تر شده است.

فصل 11- نتیجه گیری

در این پروژه هدف اصلی پیش بینی ریزش مشتری در دیتاست Telco Customer Churn بود. روند کار از مرحله آماده سازی داده تا انتخاب مدل نهایی و تحلیل رفتاری مدل ها به صورت گام به گام انجام شد تا هم دقت نتایج قابل اتکا باشد و هم امکان تکرارپذیری کامل فراهم شود.

در ابتدا داده ها پاکسازی شدند. ستون شناسه مشتری حذف شد چون اطلاعات پیش بینی کننده نداشت. ستون TotalChargesکه در برخی ردیف ها مقدار خالی داشت به عددی تبدیل شد و ردیف های دارای مقدار نامعتبر حذف شدند. همچنین SeniorCitizen به صورت دسته ای در نظر گرفته شد تا به شکل درست در مرحله کدگذاری وارد مدل شود. سپس برچسب هدف Churn به صورت دودویی صفر و یک تبدیل شد.

برای جلوگیری از سوگیری در ارزیابی، داده ها به صورت Train و Test با نسبت 80 به 20 تقسیم شدند و از stratifyاستفاده شد تا نسبت کلاس ها در هر دو بخش مشابه باقی بماند. این موضوع در مسئله ریزش بسیار مهم است چون معمولاً تعداد مشتریان غیر ریزشی بیشتر است و اگر نسبت کلاس ها حفظ نشود، معیارهایی مثل Accuracyمی تواند تصویر اشتباه بدهد. علاوه بر این، برای ارزیابی منصفانه در مرحله تیونینگ از Stratified K-Foldبا 5 فولد استفاده شد تا در هر فولد نیز همان نسبت کلاس ها حفظ شود و نوسان نتایج کمتر شود. در تمام قسمت های تصادفی سازی نیز random_state برابر 42 تنظیم شد تا نتایج کاملاً تکرارپذیر باشند.

پس از آماده سازی داده، ویژگی های دسته ای با One Hot Encoding به ویژگی های عددی تبدیل شدند و ویژگی ها نرمال سازی شدند. سپس برای کنترل ابعاد بالا و کاهش هم خطی و افزایش سرعت آموزش، کاهش بُعد با PCA انجام شد. نتیجه PCA این بود که تعداد ویژگی ها بعد از کاهش بُعد به 20 مؤلفه رسید که نشان می دهد بخش زیادی از اطلاعات در تعداد کمی مؤلفه فشرده شده است و این کار هم به ساده تر شدن مدل ها کمک می کند و هم احتمال بیش برازش را کاهش می دهد.

در ادامه سه مدل اصلی انتخاب و مقایسه شدندLogistic Regression :، Linear SVMو Decision Tree.برای هر سه مدل، تیونینگ ابرپارامترها با سه روش انجام شدGridSearch :، RandomizedSearch و Optuna و معیار بهینه سازی هم ROC AUC بود چون در مسائل نامتوازن و برای کیفیت جداسازی کلاس ها معیار مناسب تری نسبت به Accuracy است. نتایج CV نشان داد اختلاف بین سه روش جستجو خیلی کم است. دلیلش این است که فضای جستجو محدود و نسبتاً ساده بوده و مدل ها سریع به ناحیه نزدیک به بهینه می رسند، بنابراینGrid ، Randomو Optuna تقریباً به پاسخ یکسان می رسند.

اگر هدف پروژه انتخاب مدلی با بهترین عملکرد کلی و تعادل مناسب بین معیارها باشد، Logistic Regression بهترین گزینه است چون بالاترین ROC AUC و Accuracy و Precision را دارد و از نظر پایداری هم در نمودارها رفتار قابل اتکاتری نشان می دهد.

اگر هدف کسب وکار این باشد که حتی الامکان مشتری ریزشی از دست نرود و هزینه تماس یا پیشنهاد اشتباه پایین باشد، Linear SVMانتخاب مناسب تری است چون Recall بسیار بالاتری دارد و مشتریان ریزشی بیشتری را شناسایی می کند، هرچند باید پذیرفت که خطای مثبت کاذب بالا می رود.

Decision Treeدر این پروژه به دلیل نشانه های واضح بیش برازش و ROC AUC پایین تر نسبت به دو مدل دیگر، گزینه نهایی توصیه شده نیست. همچنین ترکیب PCA با درخت تصمیم معمولاً به اندازه مدل های خطی سودمند نیست چون PCA ساختار ویژگی ها را به مؤلفه های خطی تبدیل می کند و ممکن است برخی تعامل های قابل استفاده برای درخت را کم رنگ کند.