NavSanyaAnand_classification.ipynb

Nav Sanya Anand

Table of contents

Description

DATA SOURCE

The board game company Cards Against Humanity (CAH) is known for putting a lot of effort into silly publicity stunts. In 2017, they launched Cards Against Humanity Saves America, a series of politically-related activities. One such activity was a monthly poll, asking some silly questions and some serious questions about the world.

To conduct their polls in a scientifically rigorous manner, they partnered with Survey Sampling International — a professional research firm — to contact a nationally representative sample of the American public.

The analyses of the data conducted by CAH can be found here: https://thepulseofthenation.com/#poll-12. You are welcome to reference their analyses to help inform your own. Note that some questions from the poll have been omitted for purposes of this exam, and that others have been cleaned or edited so that not all possible responses are included.

Please also note that, while the data itself is collected by a rigorous neutral party, the discussion on the CAH website is highly opinionated. I share it for you information, not as an endorsement of their messages.

GOAL

Your mission today is to use the questions asked in the CAH poll to predict the political affiliation (Democrat, Republican, or Independent) of the individual.

METRIC

The evaluation metric for this competition is the overall accuracy of your predictions.

SUBMISSION FORMAT

Submission files should contain two columns: id_num and political_affiliation. The ID Numbers should be the same as those in the test.csv file, and the political_affiliation_predicted values should be your predicted affiliation.

You can generate a properly formatted prediction file as follows, assuming test_data is your test dataset and final_model_fit is your fitted final chosen model:

final_predictions = pd.DataFrame({"id_num": test_data['id_num'], "political_affiliation_predicted": final_model_fit.predict(test_data)})

```python
import pandas as pd
import numpy as np

from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

Data

```python
train_path = "Data/gsb-544-fall-2025-classification/CAH-201803-train.csv"
test_path = "Data/gsb-544-fall-2025-classification/CAH-201803-test.csv"

train = pd.read_csv(train_path)
test = pd.read_csv(test_path)
```

```python
display(train.head())
```

	id_num	Q1	Q2	political_affiliation	Q4	Q5	Q6	Q7	Q8	Q9
0	1	Male	53	Independent	Liberal	College degree	Black	No	No	No
1	5	Female	66	Independent	Conservative	Some college	White	Yes	No	Yes
2	7	Female	58	Democrat	Liberal	College degree	White	No	No	No
3	8	Male	55	Independent	Moderate	High school or less	White	Yes	Yes	Yes
4	9	Male	64	Republican	Conservative	High school or less	White	Yes	Yes	Yes

```python
display(test.head())
```

	id_num	Q1	Q2	Q4	Q5	Q6	Q7	Q8	Q9	Q10
0	2	Female	78	Conservative	College degree	White	Yes	Yes	No	Yes, very religious
1	3	Male	59	Moderate	High school or less	Black	Yes	Yes	Yes	Yes, very religious
2	4	Male	59	Moderate	High school or less	White	Yes	No	Yes	Yes, very religious
3	6	Male	52	Moderate	Graduate degree	White	Yes	Yes	Yes	Yes, somewhat reli
4	11	Female	33	Moderate	High school or less	White	No	No	Yes	Yes, somewhat reli

id_num	Q1	Q2	Q4	Q5	Q6	Q7	Q8	Q9	Q10

```
display(train.describe())
```

	id_num	Q2	Q15	Q16	Q17
count	169.000000	169.000000	169.000000	169.000000	169.000000
mean	166.786982	47.508876	4.284024	2.970414	3.497041
std	97.770210	16.057233	1.277942	1.712788	1.484540
min	1.000000	18.000000	1.000000	1.000000	1.000000
25%	82.000000	35.000000	4.000000	1.000000	2.000000
50%	162.000000	50.000000	5.000000	2.000000	4.000000
75%	248.000000	60.000000	5.000000	5.000000	5.000000
max	335.000000	79.000000	5.000000	5.000000	5.000000

```
display(test.describe())
```

	id_num	Q2	Q15	Q16	Q17
count	166.000000	166.000000	166.000000	166.000000	166.000000
mean	169.234940	48.240964	4.409639	2.993976	3.373494
std	96.184918	15.775002	1.056422	1.707371	1.649322
min	2.000000	18.000000	1.000000	1.000000	1.000000
25%	91.000000	37.000000	4.000000	1.000000	2.000000
50%	171.500000	49.000000	5.000000	2.000000	4.000000
75%	253.750000	60.000000	5.000000	5.000000	5.000000
max	334.000000	92.000000	5.000000	5.000000	5.000000

```
display(train.nunique())
```

```
id_num                   169
Q1                         2
Q2                        56
political_affiliation      3
Q4                         3
Q5                         4
Q6                         4
Q7                         2
Q8                         2
```

```
Q9                           2
Q10                          3
Q11                          2
Q12                          2
Q13                          2
Q14                          3
Q15                          4
Q16                          4
Q17                          4
Q18                          2
dtype: int64
```

```
display(test.nunique())
```

```
id_num    166
Q1          2
Q2         58
Q4          3
Q5          4
Q6          4
Q7          2
Q8          2
Q9          2
Q10         3
Q11         2
Q12         2
Q13         2
Q14         3
Q15         4
Q16         4
Q17         4
Q18         2
dtype: int64
```

```
# Null values per column
display(train.isnull().sum())
```

```
id_num                   0
Q1                       0
Q2                       0
political_affiliation    0
```

```
Q4                         0
Q5                         0
Q6                         0
Q7                         0
Q8                         0
Q9                         0
Q10                        0
Q11                        0
Q12                        0
Q13                        0
Q14                        0
Q15                        0
Q16                        0
Q17                        0
Q18                        0
dtype: int64
```

```
# Null values per column
display(test.isnull().sum())
```

```
id_num    0
Q1        0
Q2        0
Q4        0
Q5        0
Q6        0
Q7        0
Q8        0
Q9        0
Q10       0
Q11       0
Q12       0
Q13       0
Q14       0
Q15       0
Q16       0
Q17       0
Q18       0
dtype: int64
```

Numeric vs Categorical columns

```python
X = train.drop(columns=["id_num", "political_affiliation"])
y = train["political_affiliation"]
```

```python
cat_cols = X.select_dtypes(include=["object", "category"]).columns.tolist()
num_cols = X.select_dtypes(include=["int64", "float64"]).columns.tolist()
```

```python
print("Categorical columns:", cat_cols)
```

```
Categorical columns: ['Q1', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9', 'Q10', 'Q11', 'Q12', 'Q13',
```

```python
print("Numeric columns:", num_cols)
```

```
Numeric columns: ['Q2', 'Q15', 'Q16', 'Q17']
```

Preprocessing

```python
preprocess = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore"), cat_cols),
        ("num", StandardScaler(), num_cols),
    ],
    remainder='passthrough'
)
```

Modeling

```python
models = {}
```

- Multinomial logistic regression (L2)

- Multinomial logistic regression (L1, penalized)

- kNN with several k values

```
logreg_pipe = Pipeline(
    steps=[
        ("preprocess", preprocess),
        ("clf", LogisticRegression(
            multi_class="multinomial",
            solver="saga",        # supports L1 & L2
            max_iter=100000
        )),
    ]
)

logreg_param_grid = {
    "clf__penalty": ["l1", "l2"],
    "clf__C": [0.005, 0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.5,
               1.0, 2.0, 3.0, 5.0, 10.0],
    "clf__class_weight": [None, "balanced"],
}
```

```
logreg_grid = GridSearchCV(
    estimator=logreg_pipe,
    param_grid=logreg_param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
```

```
logreg_grid.fit(X, y)

print("\nBest Logistic Regression params:", logreg_grid.best_params_)
print("Best Logistic Regression CV accuracy:", logreg_grid.best_score_)
```

```
Best Logistic Regression params: {'clf__C': 0.2, 'clf__class_weight': None, 'clf__penalty':
Best Logistic Regression CV accuracy: 0.6509803921568628
```

```
C:\Users\navsa\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\linear_model
  warnings.warn(
```

```python
knn_pipe = Pipeline(
    steps=[
        ("preprocess", preprocess),
        ("clf", KNeighborsClassifier()),
    ]
)

knn_param_grid = {
    "clf__n_neighbors": [7, 9, 11, 13, 15, 17, 19],
    "clf__weights": ["uniform", "distance"],
}
```

```python
knn_grid = GridSearchCV(
    estimator=knn_pipe,
    param_grid=knn_param_grid,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)
```

```python
knn_grid.fit(X, y)

print("\nBest kNN params:", knn_grid.best_params_)
print("Best kNN CV accuracy:", knn_grid.best_score_)
```

```
Best kNN params: {'clf__n_neighbors': 13, 'clf__weights': 'distance'}
Best kNN CV accuracy: 0.5449197860962567
```

```python
if logreg_grid.best_score_ >= knn_grid.best_score_:
    best_model_name = "LogisticRegression"
    best_model = logreg_grid.best_estimator_
    best_cv_score = logreg_grid.best_score_
else:
    best_model_name = "kNN"
    best_model = knn_grid.best_estimator_
    best_cv_score = knn_grid.best_score_
```

```
print(f"\nChosen best model: {best_model_name}")
print(f"Best CV accuracy: {best_cv_score:.4f}")
```

```
Chosen best model: LogisticRegression
Best CV accuracy: 0.6510
```

Submission

```
X_test = test.drop(columns=["id_num"])
test_preds = best_model.predict(X_test)

submission = pd.DataFrame({
    "id_num": test["id_num"],
    "political_affiliation_predicted": test_preds
})

submission_filename = f"CAH_submission3_{best_model_name}.csv"
submission.to_csv(submission_filename, index=False)

print(f"\nSubmission file written to: {submission_filename}")
print(submission.head())
```

```
Submission file written to: CAH_submission3_LogisticRegression.csv
   id_num political_affiliation_predicted
0       2                      Republican
1       3                        Democrat
2       4                        Democrat
3       6                      Republican
4      11                     Independent
```

New Attempt

```python
import numpy as np
import pandas as pd

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import (
    StandardScaler,
    OneHotEncoder,
    PolynomialFeatures
)
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold, train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```python
numeric_features = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_features = X.select_dtypes(exclude=[np.number]).columns.tolist()

print("Numeric columns:", numeric_features)
print("Categorical columns:", categorical_features)
```

```
Numeric columns: ['Q2', 'Q15', 'Q16', 'Q17']
Categorical columns: ['Q1', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9', 'Q10', 'Q11', 'Q12', 'Q13',
```

```python
# Numeric pipeline: impute -> polynomial features -> scale
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("poly", PolynomialFeatures(include_bias=False)),  # degree set via GridSearch
    ("scaler", StandardScaler())
])

# Categorical pipeline: impute -> one-hot encode
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])
```

```python
preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
```

```python
        ("cat", categorical_transformer, categorical_features),
    ]
)
```

```python
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

```python
# Logistic Regression pipeline
logreg_pipe = Pipeline(steps=[
    ("preprocess", preprocessor),
    ("clf", LogisticRegression(max_iter=5000))
])

# kNN pipeline
knn_pipe = Pipeline(steps=[
    ("preprocess", preprocessor),
    ("clf", KNeighborsClassifier())
])
```

```python
logreg_param_grid = {
    "preprocess__num__poly__degree": [1, 2],       # 1 = linear, 2 = quadratic
    "clf__C": [0.01, 0.1, 1, 10, 100],
    "clf__penalty": ["l2"],
    "clf__multi_class": ["ovr", "multinomial"],
    "clf__solver": ["lbfgs"],                      # works with both multi_class options
}
```

```python
knn_param_grid = {
    "preprocess__num__poly__degree": [1],          # keep it simple for kNN
    "clf__n_neighbors": [3, 5, 7, 9, 11],
    "clf__weights": ["uniform", "distance"],
    "clf__p": [1, 2],                              # 1 = Manhattan, 2 = Euclidean
}
```

```python
models = {
    "LogisticRegression": (logreg_pipe, logreg_param_grid),
    "kNN": (knn_pipe, knn_param_grid),
}
```

```python
best_model_name = None
best_model = None
best_cv_score = -np.inf
best_search = None

for name, (pipe, param_grid) in models.items():
    print("=" * 60)
    print(f"Fitting model: {name}")
    grid = GridSearchCV(
        estimator=pipe,
        param_grid=param_grid,
        cv=cv,
        scoring="accuracy",
        n_jobs=-1,
        verbose=0
    )
    grid.fit(X, y)

    print(f"{name} best CV accuracy: {grid.best_score_}")
    print(f"{name} best params: {grid.best_params_}\n")

    if grid.best_score_ > best_cv_score:
        best_cv_score = grid.best_score_
        best_model_name = name
        best_model = grid.best_estimator_
        best_search = grid

print("=" * 60)
print(f"\nChosen best model: {best_model_name}")
print(f"Best CV accuracy: {best_cv_score}")
```

```
============================================================
Fitting model: LogisticRegression

C:\Users\navsa\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\linear_model
  warnings.warn(

LogisticRegression best CV accuracy: 0.6169117647058824
LogisticRegression best params: {'clf__C': 0.1, 'clf__multi_class': 'multinomial', 'clf__pena

============================================================
Fitting model: kNN
```

```
C:\Users\navsa\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\model_selec
 0.50330882 0.50955882        nan 0.52720588 0.55110294 0.53970588
        nan 0.51544118 0.50919118 0.54448529        nan 0.55110294
 0.49742647 0.50330882]
  warnings.warn(

kNN best CV accuracy: 0.5518382352941177
kNN best params: {'clf__n_neighbors': 5, 'clf__p': 1, 'clf__weights': 'distance', 'preproces

============================================================

Chosen best model: LogisticRegression
Best CV accuracy: 0.6169117647058824
```

New Attepmt

```python
import numpy as np
import pandas as pd

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder, PolynomialFeatures
from sklearn.impute import SimpleImputer

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

from sklearn.model_selection import GridSearchCV, RepeatedStratifiedKFold
```

```python
X = train.drop(columns=["id_num", "political_affiliation"])
y = train["political_affiliation"]

numeric_features = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_features = X.select_dtypes(exclude=[np.number]).columns.tolist()

print("Numeric:", numeric_features)
print("Categorical:", categorical_features)
```

```
Numeric: ['Q2', 'Q15', 'Q16', 'Q17']
Categorical: ['Q1', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9', 'Q10', 'Q11', 'Q12', 'Q13', 'Q14', '
```

```
numeric_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("poly", PolynomialFeatures(include_bias=False)),
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)
```

```
cv = RepeatedStratifiedKFold(
    n_splits=10,
    n_repeats=5,
    random_state=42
)
```

```
logreg_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", LogisticRegression(max_iter=10000))
])

svm_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", LinearSVC())
])
```

```
logreg_grid = {
    "preprocess__num__poly__degree": [1, 2],
    "clf__penalty": ["l1", "l2"],
```

```
    "clf__C": [0.001, 0.01, 0.1, 1, 10],
    "clf__solver": ["saga"],
    "clf__class_weight": [None, "balanced"]
}

svm_grid = {
    "preprocess__num__poly__degree": [1, 2],
    "clf__C": [0.001, 0.01, 0.1, 1, 10],
    "clf__class_weight": [None, "balanced"]
}
```

```
models = {
    "LogisticRegression": (logreg_pipe, logreg_grid),
    "LinearSVM": (svm_pipe, svm_grid)
}
```

```
best_model_name = None
best_model = None
best_cv_score = -np.inf

for name, (pipe, params) in models.items():
    print("=" * 70)
    print(f"Fitting: {name}")

    grid = GridSearchCV(
        estimator=pipe,
        param_grid=params,
        cv=cv,
        scoring="accuracy",
        n_jobs=-1,
        verbose=0
    )

    grid.fit(X, y)

    print(f"{name} BEST CV ACCURACY = {grid.best_score_}")
    print("Best params:", grid.best_params_, "\n")

    if grid.best_score_ > best_cv_score:
        best_cv_score = grid.best_score_
        best_model_name = name
        best_model = grid.best_estimator_
```

```
```

```
================================================================================
Fitting: LogisticRegression
LogisticRegression BEST CV ACCURACY = 0.6152941176470589
Best params: {'clf__C': 0.1, 'clf__class_weight': None, 'clf__penalty': 'l2', 'clf__solver':

================================================================================
Fitting: LinearSVM
LinearSVM BEST CV ACCURACY = 0.615514705882353
Best params: {'clf__C': 0.1, 'clf__class_weight': None, 'preprocess__num__poly__degree': 1}
```

```
print(f"  BEST MODEL: {best_model_name}")
print(f"  BEST CV ACCURACY: {best_cv_score}")
```

```
  BEST MODEL: LinearSVM
  BEST CV ACCURACY: 0.615514705882353
```

New Attempt

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import (
    StandardScaler,
    OneHotEncoder,
    PolynomialFeatures
)
from sklearn.impute import SimpleImputer
```

```
numeric_features = X.select_dtypes(include=[np.number]).columns.tolist()
categorical_features = X.select_dtypes(exclude=[np.number]).columns.tolist()

print("Numeric columns:", numeric_features)
print("Categorical columns:", categorical_features)
```

```
Numeric columns: ['Q2', 'Q15', 'Q16', 'Q17']
Categorical columns: ['Q1', 'Q4', 'Q5', 'Q6', 'Q7', 'Q8', 'Q9', 'Q10', 'Q11', 'Q12', 'Q13',
```

```python
try:
    ohe = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
except TypeError:
    ohe = OneHotEncoder(handle_unknown="ignore", sparse=False)

# Numeric pipeline: impute -> polynomial features -> scale
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("poly",    PolynomialFeatures(include_bias=False)),  # degree controlled in grids
    ("scaler",  StandardScaler())
])

# Categorical pipeline: impute -> one-hot
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot",  ohe)
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC, SVC
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import (
    accuracy_score,
    f1_score,
    balanced_accuracy_score,
    classification_report,
    confusion_matrix,
)
```

```python
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

```
logreg_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", LogisticRegression(max_iter=5000, multi_class="multinomial"))
])

lda_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", LinearDiscriminantAnalysis())
])

knn_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", KNeighborsClassifier())
])

linear_svm_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", LinearSVC())
])

rbf_svm_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", SVC(kernel="rbf"))
])

tree_pipe = Pipeline([
    ("preprocess", preprocessor),
    ("clf", DecisionTreeClassifier(random_state=42))
])
```

```
scoring = {
    "accuracy": "accuracy",
    "f1_macro": "f1_macro",
    "balanced_accuracy": "balanced_accuracy",
}

logreg_grid = {
    "preprocess__num__poly__degree": [1, 2],  # 1 = linear; 2 = quadratic features on numeri
    "clf__C": [0.01, 0.1, 1, 10],
    "clf__penalty": ["l2"],
    "clf__solver": ["lbfgs"],  # works with multinomial+l2
    "clf__class_weight": [None, "balanced"],
```

```
}

lda_grid = {
    # LDA already captures some structure; keep numerics simple
    "preprocess__num__poly__degree": [1],
}

knn_grid = {
    "preprocess__num__poly__degree": [1],   # scaling only, no poly
    "clf__n_neighbors": [3, 5, 7, 9, 11],
    "clf__weights": ["uniform", "distance"],
    "clf__p": [1, 2],                        # 1 = Manhattan, 2 = Euclidean
}

linear_svm_grid = {
    "preprocess__num__poly__degree": [1, 2],
    "clf__C": [0.01, 0.1, 1, 10],
    "clf__class_weight": [None, "balanced"],
}

rbf_svm_grid = {
    "preprocess__num__poly__degree": [1],   # kernel already adds nonlinearity; keep simple
    "clf__C": [0.1, 1, 10],
    "clf__gamma": ["scale", "auto"],
    "clf__class_weight": [None, "balanced"],
}

tree_grid = {
    "preprocess__num__poly__degree": [1],   # trees don't need scaling or poly, but this is
    "clf__max_depth": [None, 3, 5, 7],
    "clf__min_samples_leaf": [1, 2, 4],
    "clf__min_samples_split": [2, 5, 10],
}
```

```
models = {
    "LogisticRegression": (logreg_pipe, logreg_grid),
    "LDA":                (lda_pipe, lda_grid),
    "kNN":                (knn_pipe, knn_grid),
    "LinearSVM":          (linear_svm_pipe, linear_svm_grid),
    "RBFSVM":             (rbf_svm_pipe, rbf_svm_grid),
    "DecisionTree":       (tree_pipe, tree_grid),
```

```
}
```

```
best_model_name = None
best_model = None
best_cv_acc = -np.inf

summary_rows = []

for name, (pipe, param_grid) in models.items():
    print("=" * 80)
    print(f"Fitting model: {name}")

    grid = GridSearchCV(
        estimator=pipe,
        param_grid=param_grid,
        cv=cv,
        scoring=scoring,
        refit="accuracy",      # use accuracy to pick best parameters
        n_jobs=-1,
        verbose=0
    )

    grid.fit(X, y)

    # Best by accuracy:
    best_idx = grid.best_index_
    results = grid.cv_results_

    best_acc  = results["mean_test_accuracy"][best_idx]
    best_f1   = results["mean_test_f1_macro"][best_idx]
    best_bacc = results["mean_test_balanced_accuracy"][best_idx]

    print(f"{name} best params: {grid.best_params_}")
    print(f"{name} CV accuracy (mean):          {best_acc:.4f}")
    print(f"{name} CV macro F1 (mean):           {best_f1:.4f}")
    print(f"{name} CV balanced accuracy (mean):  {best_bacc:.4f}")

    summary_rows.append({
        "model": name,
        "cv_accuracy": best_acc,
        "cv_f1_macro": best_f1,
        "cv_balanced_accuracy": best_bacc
```

```python
    })

    if best_acc > best_cv_acc:
        best_cv_acc = best_acc
        best_model_name = name
        best_model = grid.best_estimator_

print("=" * 10)
print("Summary of best models by CV metrics:")
summary_df = pd.DataFrame(summary_rows).sort_values(
    by="cv_accuracy", ascending=False
)
print(summary_df.to_string(index=False))

print("\nChosen BEST model (by accuracy):", best_model_name)
print("Best CV accuracy:", best_cv_acc)
```

```
================================================================================
Fitting model: LogisticRegression

C:\Users\navsa\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\linear_mode
  warnings.warn(

LogisticRegression best params: {'clf__C': 0.1, 'clf__class_weight': None, 'clf__penalty': '
LogisticRegression CV accuracy (mean):          0.6169
LogisticRegression CV macro F1 (mean):           0.6079
LogisticRegression CV balanced accuracy (mean):  0.6189
================================================================================
Fitting model: LDA
LDA best params: {'preprocess__num__poly__degree': 1}
LDA CV accuracy (mean):          0.5989
LDA CV macro F1 (mean):           0.5768
LDA CV balanced accuracy (mean):  0.6022
================================================================================
Fitting model: kNN

C:\Users\navsa\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\model_selec
 0.50330882 0.50955882        nan 0.52720588 0.55110294 0.53970588
        nan 0.51544118 0.50919118 0.54448529        nan 0.55110294
 0.49742647 0.50330882]
  warnings.warn(
```

```
C:\Users\navsa\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\model_selec
 0.4930094  0.50317157        nan 0.52297017 0.54101584 0.53279472
        nan 0.50483037 0.49631977 0.53617023        nan 0.54754533
 0.48757363 0.49285733]
  warnings.warn(
C:\Users\navsa\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\model_selec
 0.50111111 0.51              nan 0.53111111 0.55222222 0.54333333
        nan 0.51888889 0.50777778 0.54666667        nan 0.55333333
 0.49666667 0.50444444]
  warnings.warn(

kNN best params: {'clf__n_neighbors': 5, 'clf__p': 1, 'clf__weights': 'distance', 'preproces
kNN CV accuracy (mean):          0.5518
kNN CV macro F1 (mean):           0.5438
kNN CV balanced accuracy (mean):  0.5511
================================================================================
Fitting model: LinearSVM
LinearSVM best params: {'clf__C': 0.01, 'clf__class_weight': None, 'preprocess__num__poly__de
LinearSVM CV accuracy (mean):          0.6169
LinearSVM CV macro F1 (mean):           0.6062
LinearSVM CV balanced accuracy (mean):  0.6178
================================================================================
Fitting model: RBFSVM
RBFSVM best params: {'clf__C': 1, 'clf__class_weight': None, 'clf__gamma': 'auto', 'preproces
RBFSVM CV accuracy (mean):          0.6287
RBFSVM CV macro F1 (mean):           0.6198
RBFSVM CV balanced accuracy (mean):  0.6267
================================================================================
Fitting model: DecisionTree
DecisionTree best params: {'clf__max_depth': 5, 'clf__min_samples_leaf': 4, 'clf__min_sample
DecisionTree CV accuracy (mean):          0.5511
DecisionTree CV macro F1 (mean):           0.5373
DecisionTree CV balanced accuracy (mean):  0.5511
==========
Summary of best models by CV metrics:
             model  cv_accuracy  cv_f1_macro  cv_balanced_accuracy
            RBFSVM     0.628676     0.619811              0.626667
LogisticRegression     0.616912     0.607912              0.618889
         LinearSVM     0.616912     0.606159              0.617778
               LDA     0.598897     0.576770              0.602222
               kNN     0.551838     0.543753              0.551111
      DecisionTree     0.551103     0.537262              0.551111
```

```
Chosen BEST model (by accuracy): RBFSVM
Best CV accuracy: 0.6286764705882353
```