implementation_supervised

November 30, 2025

1 Supervised Learning: Implementation Notebook

This notebook builds a straightforward classification pipeline. Steps:

- Load features and the target
- Quick EDA of numeric columns
- Train/test split
- Preprocessing + two baseline models (LogReg, Random Forest)
- Metrics and a confusion matrix plot

Results and metrics are saved to `data/processed/`.

[1]:
```
# Clean imports for supervised workflow
# pandas/plotting for data handling and readable visuals
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# scikit-learn tooling: splitting, models, preprocessing, pipelines, and metrics
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# local helper to load the cleaned features/target used in class
from src.data_io import load_supervised_xy

# Load features (X) and the raw target (y). We'll bin y into 3 classes later.
X, y = load_supervised_xy()
# Quick sanity checks: shapes should align (same row count) and X has many␣
 ↪columns.
print("X shape:", X.shape)
print("y shape:", y.shape)
# Peek at the first rows so we get a feel for the columns.
X.head()
```

```
X shape: (2000, 23)
y shape: (2000,)
```

[1]:

	age	gender	daily_screen_time_hours	phone_usage_hours	laptop_usage_hours	tablet_usage_hours	tv_usage_hours	social_media_hours	work_related_hours	entertainment_hours	…	mood_rating	stress_level	physical_activity_hours_per_week	location_type	uses_wellness_apps	eats_healthy	caffeine_intake_mg_per_day	weekly_anxiety_score	weekly_depression_score	mindfulness_minutes_per_day
0	51	0	4.8	3.4	1.3	1.6	1.6	4.1	2.0	1.0	…	6	10	0.7	Urban	1	1	125.2	13	15	4.0
1	64	1	3.9	3.5	1.8	0.9	2.0	2.7	3.1	1.0	…	5	6	4.3	Suburban	0	1	150.4	19	18	6.5
2	41	2	10.5	2.1	2.6	0.7	2.2	3.0	2.8	4.1	…	5	5	3.1	Suburban	0	0	187.9	7	3	6.9
3	27	2	8.8	0.0	0.0	0.7	2.5	3.3	1.6	1.3	…	10	5	0.0	Rural	0	1	73.6	7	2	4.8
4	55	1	5.9	1.7	1.1	1.5	1.6	1.1	3.6	0.8	…	8	7	3.0	Urban	1	1	217.5	8	10	0.0

[5 rows x 23 columns]

[ ]:
```
# Load configuration if available
from pathlib import Path
import yaml
# We allow optional overrides (e.g., test_size, output paths) via config.yaml.
config_path = Path('config.yaml')
config = {}
if config_path.exists():
    # Safe-load YAML; if the file is empty, fall back to an empty dict.
    with open(config_path, 'r', encoding='utf-8') as f:
        config = yaml.safe_load(f) or {}
# Show the effective config so it's visible in the notebook output.
config
```

[ ]:
```
{'paths': {'supervised': 'data/raw/clean_supervised.csv',
  'unsupervised': 'data/raw/unsupervised_no_target.csv'},
 'supervised': {'target': 'mental_health_score',
  'test_size': 0.2,
  'random_state': 42},
 'unsupervised': {'use_numeric_only': True, 'random_state': 42, 'k': 3}}
```

1.1 Exploratory Data Analysis

[3]:
```
# Basic info and target identification
# Display the dataframe info to confirm dtypes and missing values.
X.info()

# Force 3-class target binning for a clear classification task.
# The raw mental_health_score spans many unique values; we group it into␣
 ↪tertiles
# to create balanced, interpretable classes for a simple classifier demo.
import numpy as np

# Always derive a simple 3-class label from the raw score.
# Bins: (-inf, 33] -> 'low', (33, 66] -> 'medium', (66, inf) -> 'high'.
y_raw = y.astype(float)
y_final = pd.cut(
    y_raw,
    bins=[-np.inf, 33, 66, np.inf],
    labels=['low', 'medium', 'high']
)
print("Forced 3-class target (L/M/H)")
print(y_final.value_counts())

# Keep a simple handle to the target name and the feature list for later␣
 ↪reference.
target_col = 'mental_health_score_binned'
features = X.columns.tolist()
```

```
(target_col, len(features))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 23 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   age                               2000 non-null   int64
 1   gender                            2000 non-null   int64
 2   daily_screen_time_hours           2000 non-null   float64
 3   phone_usage_hours                 2000 non-null   float64
 4   laptop_usage_hours                2000 non-null   float64
 5   tablet_usage_hours                2000 non-null   float64
 6   tv_usage_hours                    2000 non-null   float64
 7   social_media_hours                2000 non-null   float64
 8   work_related_hours                2000 non-null   float64
 9   entertainment_hours               2000 non-null   float64
 10  gaming_hours                      2000 non-null   float64
 11  sleep_duration_hours              2000 non-null   float64
 12  sleep_quality                     2000 non-null   int64
 13  mood_rating                       2000 non-null   int64
 14  stress_level                      2000 non-null   int64
 15  physical_activity_hours_per_week  2000 non-null   float64
 16  location_type                     2000 non-null   object
 17  uses_wellness_apps                2000 non-null   int64
 18  eats_healthy                      2000 non-null   int64
 19  caffeine_intake_mg_per_day        2000 non-null   float64
 20  weekly_anxiety_score              2000 non-null   int64
 21  weekly_depression_score           2000 non-null   int64
 22  mindfulness_minutes_per_day       2000 non-null   float64
dtypes: float64(13), int64(9), object(1)
memory usage: 359.5+ KB
Forced 3-class target (L/M/H)
mental_health_score
medium    1095
low        463
high       442
Name: count, dtype: int64
```

[3]:
```
('mental_health_score_binned', 23)
```

[4]:
```
# Quick distributions and correlations
# Split columns into numeric and categorical for later preprocessing.
numeric_cols = X.select_dtypes(include='number').columns.tolist()
categorical_cols = [c for c in X.columns if c not in numeric_cols]
# Basic numeric summary to spot scales and outliers.
display(X[numeric_cols].describe())
```

```
# A correlation heatmap is most meaningful with 2+ numeric columns.
if len(numeric_cols) > 1:
    plt.figure(figsize=(8,6)); sns.heatmap(X[numeric_cols].corr(),␣
 ↪cmap='coolwarm', annot=False); plt.title('Numeric Correlations'); plt.show()
```

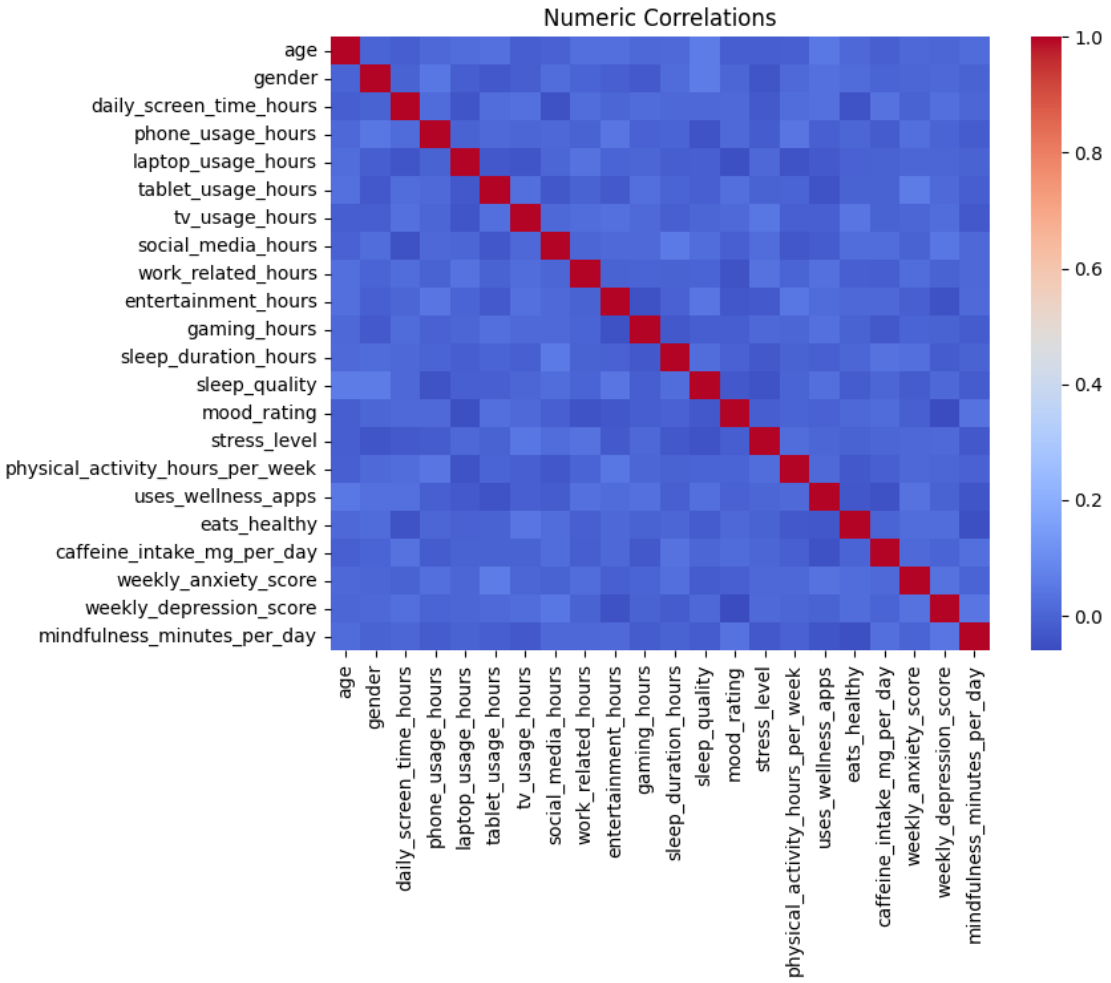
```
                age     gender  daily_screen_time_hours  phone_usage_hours  \
count  2000.000000  2000.0000              2000.000000        2000.000000
mean     38.805500     0.6240                 6.025600           3.023700
std      14.929203     0.6464                 1.974123           1.449399
min      13.000000     0.0000                 0.000000           0.000000
25%      26.000000     0.0000                 4.700000           2.000000
50%      39.000000     1.0000                 6.000000           3.000000
75%      51.000000     1.0000                 7.325000           4.000000
max      64.000000     2.0000                13.300000           8.400000

       laptop_usage_hours  tablet_usage_hours  tv_usage_hours  \
count         2000.000000         2000.000000     2000.000000
mean             1.999950            0.995650        1.503700
std              0.997949            0.492714        0.959003
min              0.000000            0.000000        0.000000
25%              1.300000            0.600000        0.800000
50%              2.000000            1.000000        1.500000
75%              2.700000            1.300000        2.200000
max              5.600000            2.500000        4.700000

       social_media_hours  work_related_hours  entertainment_hours  …  \
count         2000.000000         2000.000000          2000.00000  …
mean             2.039200            2.010250             2.46735  …
std              1.133435            1.116111             1.23686  …
min              0.000000            0.000000             0.00000  …
25%              1.200000            1.200000             1.60000  …
50%              2.000000            2.000000             2.40000  …
75%              2.800000            2.800000             3.30000  …
max              5.800000            5.900000             6.80000  …

       sleep_quality  mood_rating  stress_level  \
count    2000.000000  2000.000000   2000.000000
mean        5.567000     5.591000      5.541500
std         2.826217     2.899814      2.885731
min         1.000000     1.000000      1.000000
25%         3.000000     3.000000      3.000000
50%         6.000000     6.000000      6.000000
75%         8.000000     8.000000      8.000000
max        10.000000    10.000000     10.000000

       physical_activity_hours_per_week  uses_wellness_apps  eats_healthy  \
count                       2000.000000         2000.000000   2000.000000
mean                           3.087150            0.387500      0.507500
```

std	1.885258	0.487301	0.500069
min	0.000000	0.000000	0.000000
25%	1.600000	0.000000	0.000000
50%	3.000000	0.000000	1.000000
75%	4.400000	1.000000	1.000000
max	9.700000	1.000000	1.000000

	caffeine_intake_mg_per_day	weekly_anxiety_score	\
count	2000.00000	2000.000000	
mean	148.07970	9.887500	
std	48.86066	6.027853	
min	0.80000	0.000000	
25%	113.90000	5.000000	
50%	147.45000	10.000000	
75%	180.70000	15.000000	
max	364.90000	20.000000	

	weekly_depression_score	mindfulness_minutes_per_day
count	2000.00000	2000.000000
mean	10.04900	10.753750
std	6.05334	7.340269
min	0.00000	0.000000
25%	5.00000	4.900000
50%	10.00000	10.400000
75%	15.00000	15.800000
max	20.00000	36.400000

[8 rows x 22 columns]



1.2 Modeling Pipeline

[ ]:
```
# Train/validation split
# Use the final target (forced L/M/H) for classification.
# Stratify preserves the class proportions in both train and test splits.
X_train, X_test, y_train, y_test = train_test_split(
    X, y_final,
    test_size=config.get('supervised', {}).get('test_size', 0.2),  # default␣
 ↪20% test
    random_state=42,  # reproducibility
    stratify=y_final
)
# Return sizes as a quick check.
len(X_train), len(X_test)
```

[ ]:
```
(1600, 400)
```

[ ]:
```
# Preprocessing and model candidates
# Recompute basic column lists (harmless repeat in notebooks).
numeric_cols = X.select_dtypes(include='number').columns.tolist()
categorical_cols = [c for c in X.columns if c not in numeric_cols]

# ColumnTransformer applies:
# - StandardScaler to numeric columns (zero mean, unit variance).
# - OneHotEncoder to categoricals (ignore unseen categories at test time).
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_cols)
])

# Two lightweight baseline models wrapped in Pipelines so preprocessing happens␣
 ↪inside CV/inference:
log_reg = Pipeline([('prep', preprocessor), ('clf',␣
 ↪LogisticRegression(max_iter=500))])
rf_clf = Pipeline([('prep', preprocessor), ('clf',␣
 ↪RandomForestClassifier(random_state=42))])

# We'll compare them side-by-side below.
models = { 'log': log_reg, 'rf': rf_clf }
models
```

[ ]:
```
{'log': Pipeline(steps=[('prep',
                 ColumnTransformer(transformers=[('num', StandardScaler(),
                                                  ['age', 'gender',
                                                   'daily_screen_time_hours',
                                                   'phone_usage_hours',
                                                   'laptop_usage_hours',
                                                   'tablet_usage_hours',
                                                   'tv_usage_hours',
                                                   'social_media_hours',
                                                   'work_related_hours',
                                                   'entertainment_hours',
                                                   'gaming_hours',
                                                   'sleep_duration_hours',
                                                   'sleep_quality',
                                                   'mood_rating',
                                                   'stress_level',
    'physical_activity_hours_per_week',
                                                   'uses_wellness_apps',
                                                   'eats_healthy',
    'caffeine_intake_mg_per_day',
                                                   'weekly_anxiety_score',
                                                   'weekly_depression_score',
    'mindfulness_minutes_per_day']),
```

```
                                                  ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                   ['location_type'])])),
                 ('clf', LogisticRegression(max_iter=500))]),
 'rf': Pipeline(steps=[('prep',
                        ColumnTransformer(transformers=[('num', StandardScaler(),
                                                         ['age', 'gender',
                                                          'daily_screen_time_hours',
                                                          'phone_usage_hours',
                                                          'laptop_usage_hours',
                                                          'tablet_usage_hours',
                                                          'tv_usage_hours',
                                                          'social_media_hours',
                                                          'work_related_hours',
                                                          'entertainment_hours',
                                                          'gaming_hours',
                                                          'sleep_duration_hours',
                                                          'sleep_quality',
                                                          'mood_rating',
                                                          'stress_level',
'physical_activity_hours_per_week',
                                                          'uses_wellness_apps',
                                                          'eats_healthy',
'caffeine_intake_mg_per_day',
                                                          'weekly_anxiety_score',
                                                          'weekly_depression_score',
'mindfulness_minutes_per_day']),
                                                  ('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                   ['location_type'])])),
                 ('clf', RandomForestClassifier(random_state=42))])}
```

[ ]:
```
# Fit and evaluate both models
# Train each candidate, predict on the held-out test set, and collect metrics.
results = {}
for key, model in models.items():
    # Train the pipeline (fits preprocessing + estimator).
    model.fit(X_train, y_train)
    # Predict class labels on the test set.
    y_pred = model.predict(X_test)
    # Basic metrics for a quick comparison; report dict helps later saving/
 ↪analysis.
    acc = accuracy_score(y_test, y_pred)
    results[key] = {
        'accuracy': acc,
        'report': classification_report(y_test, y_pred, output_dict=True,␣
 ↪zero_division=0)
```

```
    }
# Show the structured results for transparency.
results
```

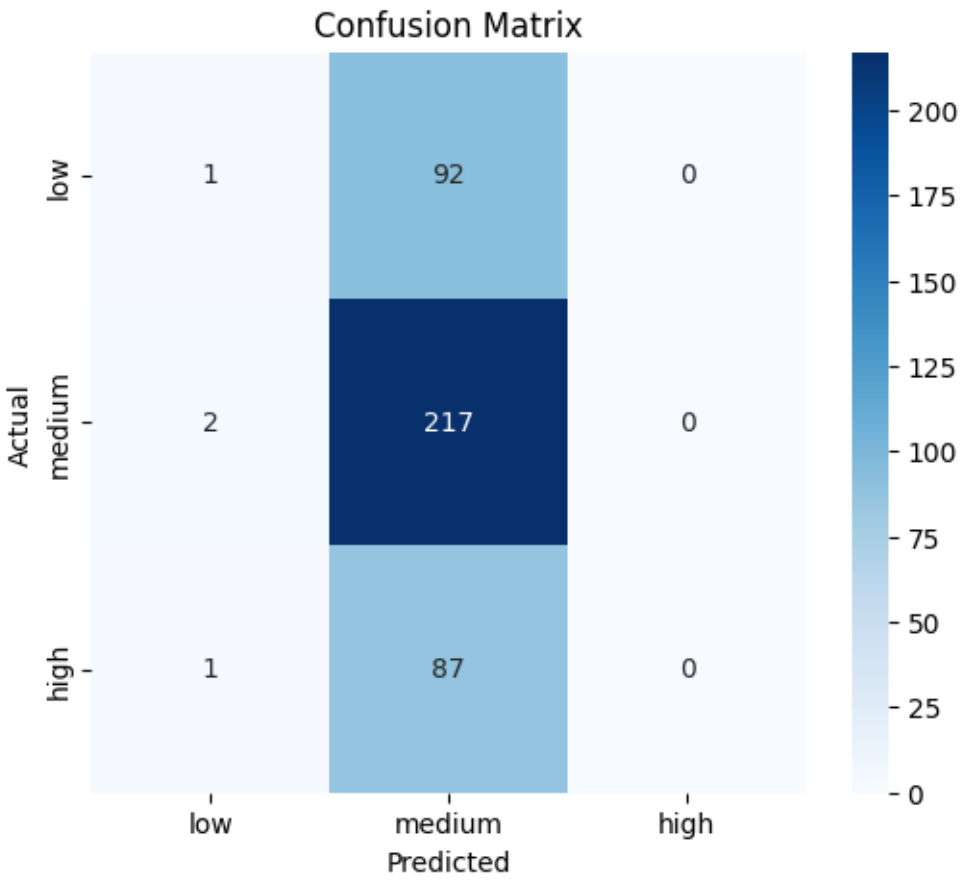
[ ]:

```
{'log': {'accuracy': 0.545,
  'report': {'high': {'precision': 0.0,
    'recall': 0.0,
    'f1-score': 0.0,
    'support': 88.0},
   'low': {'precision': 0.25,
    'recall': 0.010752688172043012,
    'f1-score': 0.020618556701030927,
    'support': 93.0},
   'medium': {'precision': 0.547979797979798,
    'recall': 0.9908675799086758,
    'f1-score': 0.7056910569105691,
    'support': 219.0},
   'accuracy': 0.545,
   'macro avg': {'precision': 0.265993265993266,
    'recall': 0.33387342269357295,
    'f1-score': 0.24210320453720002,
    'support': 400.0},
   'weighted avg': {'precision': 0.3581439393939394,
    'recall': 0.545,
    'f1-score': 0.39115966809152625,
    'support': 400.0}}},
 'rf': {'accuracy': 0.535,
  'report': {'high': {'precision': 0.3333333333333333,
    'recall': 0.011363636363636364,
    'f1-score': 0.02197802197802198,
    'support': 88.0},
   'low': {'precision': 0.125,
    'recall': 0.010752688172043012,
    'f1-score': 0.019801980198019802,
    'support': 93.0},
   'medium': {'precision': 0.5449871465295629,
    'recall': 0.9680365296803652,
    'f1-score': 0.6973684210526315,
    'support': 219.0},
   'accuracy': 0.535,
   'macro avg': {'precision': 0.33444015995429877,
    'recall': 0.3300509514053482,
    'f1-score': 0.24638280774289112,
    'support': 400.0},
   'weighted avg': {'precision': 0.400776296058269,
    'recall': 0.535,
    'f1-score': 0.3912483357575202,
```

```
    'support': 400.0}}}}
```

[ ]:
```
# Choose best and show metrics
if not results:
    raise RuntimeError("No models produced results; check data/preprocessing.")

# Pick the model with the highest test accuracy (simple tie-break: first max).
best_key = max(results, key=lambda k: results[k]['accuracy'])
best_model = models[best_key]
print('Best model:', best_key, 'Accuracy:', results[best_key]['accuracy'])
# Human-readable classification report (precision/recall/F1 per class).
print(classification_report(y_test, best_model.predict(X_test),␣
 ↪zero_division=0))

# Confusion matrix plot
# Explicit label order keeps rows/columns consistent with our L/M/H binning.
labels = ['low', 'medium', 'high']
conf_mat = confusion_matrix(y_test, best_model.predict(X_test), labels=labels)
plt.figure(figsize=(6,5))
# Add axis tick labels so it's easy to read which cell is which.
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', xticklabels=labels,␣
 ↪yticklabels=labels)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Best model: log Accuracy: 0.545
              precision    recall  f1-score   support

        high       0.00      0.00      0.00        88
         low       0.25      0.01      0.02        93
      medium       0.55      0.99      0.71       219

    accuracy                           0.55       400
   macro avg       0.27      0.33      0.24       400
weighted avg       0.36      0.55      0.39       400
```



1.2.1 Notes on Target and Metrics

- The original `mental_health_score` spans many unique values (0–100 range), which is not ideal for basic classification.
- We explicitly grouped it into three categories: low / medium / high using cutoffs at 33 and 66.
- This produces a compact 3×3 confusion matrix and a readable report (often with the medium class dominating).
- If accuracy is modest, that's expected with simple baselines; the goal is a clear, defensible pipeline and explanation.

1.3 Save Artifacts

[ ]:
```
# Save processed and metrics according to config
from pathlib import Path
import json
# Resolve processed output directory (defaults to data/processed).
processed_dir = Path(config.get('paths', {}).get('processed', 'data/processed'))
processed_dir.mkdir(parents=True, exist_ok=True)
```

```
# Write the metrics dictionary so it can be reused outside the notebook.
metrics_path = processed_dir / 'supervised_metrics.json'
with open(metrics_path, 'w', encoding='utf-8') as f:
    json.dump(results, f, indent=2)
# Return the path for a quick visual confirmation in the output cell.
metrics_path
```

[ ]:
```
WindowsPath('data/processed/supervised_metrics.json')
```

1.3.1 Summary (for presentation)

- Problem: predict mental health category from available features.
- Features used: numeric and one-hot encoded categorical columns from the dataset.
- Models tried: Logistic Regression, Random Forest.
- Winner: the model with higher accuracy on the test set (reported above).
- Next steps: tune hyperparameters, engineer features, or treat the original score as a regression problem.