Machine Learning Pipelines and Grid Search Fetal Health Classification

Prashant Kumar Singh

20/01/2025

Abstract

This report summarizes the implementation of **machine learning pipelines**, ensemble methods, and hyperparameter tuning for fetal health classification. The project focuses on preprocessing, resampling imbalanced datasets, applying ensemble methods (Random Forest, AdaBoost, XGBoost, Voting Classifier), and integrating **Grid Search** and **Randomized Search** into reusable pipelines for model selection.

1 Introduction

The goal of this lab is to understand the complete machine learning workflow, from preprocessing to model selection. We use the **Fetal Health Classification** dataset from Kaggle, containing 2126 records from Cardiotocogram (CTG) exams. CTGs monitor fetal heart rate (FHR), movements, and uterine contractions to prevent maternal and child mortality.

The dataset contains three classes:

- Normal: 1
- Suspect: 2
- Pathological: 3

2 Data Exploration

We first load the dataset and perform basic exploration:

```
import pandas as pd
data = pd.read_csv("fetal_health.csv")
data.head()
```

Check for duplicates and drop them:

```
data_dropped = data.drop_duplicates()
```

Split features and labels:

```
X = data_dropped.iloc[:, :-1]
y = data_dropped["fetal_health"].astype(int)
```

2.1 Label Distribution

```
y.value_counts() / len(y)
```

The dataset is imbalanced, with class 1 being dominant. Resampling will be applied later.

3 Preprocessing

3.1 Train-Test Split and Scaling

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

3.2 Handling Imbalanced Dataset

We use **RandomOverSampler**, **SMOTE**, and **ADASYN**:

```
from imblearn.over_sampling import RandomOverSampler, SMOTE, ADASYN

ros = RandomOverSampler(random_state=12345)
X_resampled, y_resampled = ros.fit_resample(X_train_scaled, y_train)
```

4 Ensemble Methods

4.1 Voting Classifier

```
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

voting = VotingClassifier([
    ('lr', LogisticRegression()),
    ('dt', DecisionTreeClassifier()),
    ('svm', SVC())
], voting='hard')

voting.fit(X_train_scaled, y_train)
voting.score(X_test_scaled, y_test)
```

4.2 Bagging, Random Forest, AdaBoost, XGBoost

```
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier,
    AdaBoostClassifier
import xgboost as xgb

bag_clf = BaggingClassifier(DecisionTreeClassifier(), n_estimators=500,
    oob_score=True)
rf = RandomForestClassifier()
ab = AdaBoostClassifier()
xgbm = xgb.XGBClassifier()
```

5 Hyperparameter Tuning

5.1 Randomized Search and Grid Search

```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

rf_random = RandomizedSearchCV(estimator=rf, param_distributions=
   random_grid, n_iter=100, cv=3)
rf_random.fit(X_train_scaled, y_train)
rf_random.best_estimator_
```

6 Machine Learning Pipelines

6.1 Custom Transformer

```
from sklearn.base import BaseEstimator, TransformerMixin
class SquaredFeatureTransformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        return np.hstack((X, X**2))
```

6.2 Pipeline with Grid Search

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler

pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('squared', SquaredFeatureTransformer()),
    ('selector', VarianceThreshold(0.1)),
    ('classifier', RandomForestClassifier())
])

parameters = {
    'scaler': [StandardScaler(), MinMaxScaler()],
    'selector__threshold': [0, 0.001, 0.01],
    'classifier': [AdaBoostClassifier(), RandomForestClassifier()]
```

```
}

grid_search = GridSearchCV(pipe, parameters, cv=5)
```

7 Nested k-Fold Cross Validation

We performed nested cross-validation using the breast cancer dataset to select the best model while avoiding bias.

```
from sklearn.datasets import load_breast_cancer
X, y = load_breast_cancer(return_X_y=True)
# Outer CV
outer_cv = StratifiedKFold(n_splits=2, shuffle=True, random_state=42)
# Inner CV with GridSearchCV
# ...
final_accuracy = {'RandomForest': 0.9455, 'AdaBoost': 0.9455}
```

8 Conclusion

We successfully built a complete ML workflow including preprocessing, handling imbalanced datasets, ensemble methods, hyperparameter tuning, and pipeline integration. The best performing model for fetal health classification was found using XGBoost and Random Forest with pipeline and grid search optimization.