Predicting Wine Quality Using Machine Learning

Meta-Learning for Automated Machine Learning Pipeline Selection

Melissa Butler and Emma Franz
University of Wyoming
April 30, 2023

1 Introduction

The Franz Butler Vineyard (TM) would like to predict how much they can charge for a bottle of wine from their vineyard. The local market has three main price points: Boxed, Good, and Fancy. Each of these categories contains two sub-categories that can affect pricing, albeit not as significantly. They would like to compare various categorical Machine Learning models with optimized hyperparameters, to decide the most accurate model for predicting the quality of a wine, based on 11 testable features.

2 Dataset Description

Our dataset consists of quantitative descriptions of various wines. There are 1599 wines sampled. We have 11 features for each observation, each with a numeric rating of a physiochemical property of the wine such as alcohol content and acidity. The output variable we seek to predict corresponds to the quality rating of each wine, from sensory data. There are no missing values. A closer inspection of the quality outputs shows a total of 6 integer categories (3-8) as no wine on the list scored a 1, 2, 9, or 10. So, our input values are 11 numerical valued features and our output is a categorical rating ranging from 3-8.

	fixed acidity	volatile acidity	citric acid	...	sulphates	alcohol	quality
0	7.4	0.70	0.00	...	0.56	9.4	5
1	7.8	0.88	0.00	...	0.68	9.8	5
2	7.8	0.76	0.04	...	0.65	9.8	5
3	11.2	0.28	0.56	...	0.58	9.8	6
4	7.4	0.70	0.00	...	0.56	9.4	5

3 Experimental Setup

We first run a base model using a Random Forest Classifier, with the default parameters, on a 5-fold cross validation. Then a 3-fold cross validation and *scikit learn*'s RandomSearchCV is used to run a random search over the entire parameter field. Iterations of 50, 100, and 1000 were run. The parameter space used includes data preprocessing, feature selection, four classifiers and their corresponding hyperparameter spaces. A total of 1320 candidates are available for the search. The top three models are then reported. Lastly, a grid search over all possible candidates was run.

3.1 Data Preprocessing

We apply several preprocessing techniques to the data. Since, we are using the distance based classifier K-Nearest Neighbors, three data scaling techniques were apply: Min-Max Scalar, Absolute-Max Scalar, and the Robust Scalar. Min-Max and Absolute-Max were chosen, since all of our values are positive with varying ranges across features. We are not guaranteed an absence of outliers, so the Robust Scalar was also added, with a three quantile ranges of $(25, 75), (15, 75), (25, 85)$. The first range is the default values and the second two are simply expanded in either direction, for comparison. Our data is not assumed to have normal distribution of the data, so a Standard Scalar was not used. Our other classifiers are decision based and do not require data scaling.

3.2 Feature Selection

To improve computation time and reduce cost of testing future wine, feature selection is implemented. We try a Variance Threshold feature selection to see any correlation between features. Current literature recommends a wide range of thresholds, so we use a range from low to high $0.1, 0.5, 0.9$ and also include no threshold to include all features. We also try univariate selection to determine the most influential features. Our current lab has pricing options of testing $3, 9, 12$ features, so we use a Select K Best with these values. Since our target is categorical and our features are continuous, we use the default ANOVA F-Test Classification.

3.3 Hyperparameters

Using the *scikit learn* documentation for each classifier, we chose the recommend hyperparameters to tune. We use the default values as well as any suggested values. If no values were suggested in the documentation, we create a range to theoretically increase accuracy. The following classifiers and hyperparameter ranges were used.

Classifier	Hyperparameter	Default	Range
Random Forest	n estimators	100	100, 500
	max features	1	1, sqrt(# of features)
K-Nearest Neighbors	n neighbors	5	1,2,3,4,5,6,7,8,9,10
	weights	uniform	uniform, distance
Support Vector Machine	kernel	rbf	rbf, linear
	C	1	0.1, 1, 10, 100
Ada Boost Classifier	n estimators	50	10, 50, 100
	learning rate	1	0.01, 1, 5, 10

4 Results

4.1 Default Random Forest Classifier Model

As a base comparison model, a standard Random Forest Classifier was ran with a cross validation of 5 folds.

```
Base Random Forest Model Accuracy
[0.553125   0.546875   0.621875   0.60625    0.56426332]
Average Random Forest Model Accuracy
0.5784776645768025
```

4.2 Full Parameter Space Search

Even with 1000 iterations, the accuracy did not improve significantly over a standard Random Forest Classifier. The full grid search, resulted in a slightly higher accuracy and shows the optimal model. Going forward, we see two options. Continue to improve the model by expanding the search area by hyperparameter ranges and including more classifiers, while implementing Bayesian Optimization to search the space for greater efficiency. Or create a fancier bottle to convince connoisseurs our wine is Fancy.

4.2.1 50 Iterations

```
Fitting 3 folds for each of 50 candidates, totalling 150 fits
RandomizedSearchCV took 40.57 seconds for 50 candidates.

Model with rank: 1
Mean validation score: 0.573 (std: 0.036)
Parameters: {'scaler__selected_model': ('minmax', {'copy': True}),
            'features__selected_model': ('kbest', {'k': 11}),
```

```
            'classifier__selected_model': ('svm', {'C': 10, 'kernel': 'rbf'})}

Model with rank: 2
Mean validation score: 0.560 (std: 0.025)
Parameters: {'scaler__selected_model': ('max', {'copy': True}),
            'features__selected_model': ('kbest', {'k': 9}),
            'classifier__selected_model': ('rf', {'max_features': 'sqrt', 'n_estimators': 500})}

Model with rank: 3
Mean validation score: 0.559 (std: 0.029)
Parameters: {'scaler__selected_model': ('minmax', {'copy': True}),
            'features__selected_model': ('var', {'threshold': 0}),
            'classifier__selected_model': ('rf', {'max_features': 1.0, 'n_estimators': 500})}
```

4.2.2 100 Iterations

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
RandomizedSearchCV took 58.61 seconds for 100 candidates.

Model with rank: 1
Mean validation score: 0.577 (std: 0.026)
Parameters: {'scaler__selected_model': ('robust', {'quantile_range': (25.0, 85.0)}),
            'features__selected_model': ('kbest', {'k': 9}),
            'classifier__selected_model': ('svm', {'C': 10, 'kernel': 'rbf'})}

Model with rank: 2
Mean validation score: 0.576 (std: 0.006)
Parameters: {'scaler__selected_model': ('max', {'copy': True}),
            'features__selected_model': ('kbest', {'k': 9}),
            'classifier__selected_model': ('svm', {'C': 1, 'kernel': 'linear'})}

Model with rank: 3
Mean validation score: 0.571 (std: 0.022)
Parameters: {'scaler__selected_model': ('robust', {'quantile_range': (15.0, 75.0)}),
            'features__selected_model': ('kbest', {'k': 9}),
            'classifier__selected_model': ('svm', {'C': 10, 'kernel': 'rbf'})}
```

4.2.3 1000 Iterations

```
Fitting 3 folds for each of 1000 candidates, totalling 3000 fits
RandomizedSearchCV took 669.97 seconds for 1000 candidates.

Model with rank: 1
Mean validation score: 0.582 (std: 0.023)
Parameters: {'scaler__selected_model': ('max', {'copy': True}),
            'features__selected_model': ('kbest', {'k': 11}),
            'classifier__selected_model': ('svm', {'C': 10, 'kernel': 'rbf'})}

Model with rank: 1
Mean validation score: 0.582 (std: 0.023)
Parameters: {'scaler__selected_model': ('max', {'copy': True}),
            'features__selected_model': ('var', {'threshold': 0}),
            'classifier__selected_model': ('svm', {'C': 10, 'kernel': 'rbf'})}

Model with rank: 3
```

```
Mean validation score: 0.582 (std: 0.010)
Parameters: {'scaler__selected_model': ('max', {'copy': True}),
            'features__selected_model': ('kbest', {'k': 9}),
            'classifier__selected_model': ('svm', {'C': 10, 'kernel': 'rbf'})}
```

4.3 Grid Search

```
Fitting 3 folds for each of 1320 candidates, totalling 3960 fits
GridSearchCV took 825.14 seconds for 1320 candidates.

Model with rank: 1
Mean validation score: 0.582 (std: 0.023)
Parameters: {'classifier__selected_model': ('svm', {'C': 10, 'kernel': 'rbf'}),
            'features__selected_model': ('var', {'threshold': 0}),
            'scaler__selected_model': ('max', {'copy': True})}

Model with rank: 1
Mean validation score: 0.582 (std: 0.023)
Parameters: {'classifier__selected_model': ('svm', {'C': 10, 'kernel': 'rbf'}),
            'features__selected_model': ('kbest', {'k': 11}),
            'scaler__selected_model': ('max', {'copy': True})}

Model with rank: 3
Mean validation score: 0.582 (std: 0.010)
Parameters: {'classifier__selected_model': ('svm', {'C': 10, 'kernel': 'rbf'}),
            'features__selected_model': ('kbest', {'k': 9}),
            'scaler__selected_model': ('max', {'copy': True})}
```

5 Resources Used

https://scikit-learn.org/stable/auto_examples/model_selection/plot_randomized_search.html#
sphx-glr-auto-examples-model-selection-plot-randomized-search-py
https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35
https://www.kaggle.com/code/tanmayunhale/feature-selection-variance-threshold
https://towardsdatascience.com/mistakes-in-applying-univariate-feature-selection-methods-34c43ce8b93d
https://github.com/bmurauer/pipelinehelper https://scikit-learn.org/stable/index.html

6 Appendix

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import pandas as pd
import numpy as np

from time import time
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV, cross_val_score,
   GridSearchCV
from sklearn.preprocessing import MaxAbsScaler, MinMaxScaler, RobustScaler
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from pipelinehelper import PipelineHelper
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import SelectKBest
from sklearn.svm import SVC

import warnings
warnings.filterwarnings("ignore")

df = pd.read_csv('winequality-red.csv', delimiter = ";")
df.shape
X = df.values[:,0:-1]
y = df.values[:,-1]
print(df.head())

# Base Random Forest Model
rfc = RandomForestClassifier()
scores = cross_val_score(rfc, X, y, cv=5)
print("Base Random Forest Model Accuracy")
print(scores)
print("Average Random Forest Model Accuracy")
print(scores.mean())


# Report best scores
def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results["rank_test_score"] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print(
                "Mean validation score: {0:.3f} (std: {1:.3f})".format(
                    results["mean_test_score"][candidate],
                    results["std_test_score"][candidate],
                )
            )
            print("Parameters: {0}".format(results["params"][candidate]))
            print("")

#Pipeline
```

```
pipe = Pipeline([
    # Data Scaling
    ('scaler', PipelineHelper([
        ('minmax', MinMaxScaler()),
        ('max', MaxAbsScaler()),
        ('robust', RobustScaler()),
    ])),
    # Feature Selection
    ('features', PipelineHelper([
        ('var', VarianceThreshold()),
        ('kbest', SelectKBest()),
    ])),
    # Classifiers
    ('classifier', PipelineHelper([
        ('rf', RandomForestClassifier()),
        ('knn', KNeighborsClassifier()),
        ('svm', SVC()),
        ('ada', AdaBoostClassifier())
    ])),
])

#Parameters
params = {
    # Data Scaling
    'scaler__selected_model': pipe.named_steps['scaler'].generate({
        'minmax__copy': [True],
        'max__copy': [True],
        'robust__quantile_range': [(25.0,75.0), (15.0,75.0), (25.0,85.0)],
    }),
    # Feature Selection
    'features__selected_model': pipe.named_steps['features'].generate({
        'var__threshold': [0, 0.5, 0.9],
        'kbest__k': [3, 9, 11],
    }),
    # Hyperparmeters
    'classifier__selected_model': pipe.named_steps['classifier'].generate
       ({
        'rf__n_estimators': [100, 500],
        'rf__max_features': [1.0, 'sqrt'],
        'knn__n_neighbors': [1,2,3,4,5,6,7,8,9,10],
        'knn__weights': ['uniform', 'distance'],
        'svm__kernel' : ['linear', 'rbf'],
        'svm__C' : [0.1, 1, 10, 100],
        'ada__n_estimators' : [10, 50, 100],
        'ada__learning_rate' : [0.01, 1, 5, 10],
    })
}

n_iter_search = 1320
random_search = GridSearchCV(
    pipe,
    params,
    cv = 3,
    scoring='accuracy',
```

```
    #n_iter=n_iter_search,
    verbose=1,)

start = time()
random_search.fit(X, y)
end = time()
print(f"RandomizedSearchCV took {(end-start):.2f} seconds for {
   n_iter_search} candidates.")
report(random_search.cv_results_)
#print(grid.best_params_)
#print(grid.best_score_)
```