Floor-Cleaning-Machine-ML

August 8, 2024

1 Floor Cleaning Machine ML Model with Random Forest

This code is fully owned by Sasindu Malhara (2022/E/126,University of jaffna) group R1.

1.0.1 Import libraries

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import json
from sklearn.metrics import accuracy_score
```

1.0.2 Import data

[2]:
```
data=pd.read_csv("D:\\Sem 3 - E22\\E22\\DP\\Our\\2.
↪0\\Git\\Floor-cleaning-machine\\Code\\ML\\Floor cleaning machine ML dataset .
↪csv")
```

1.0.3 Data mapping

[3]:
```
data['next movement'] = data['next movement'].map({' L': 0, 'R': 1, 'B': 2, 'F':
↪ 3})
```

[4]:
```
data.fillna(0, inplace=True)
```

[5]:
```
X = data.drop('next movement', axis=1)
y = data['next movement']
```

[6]:
```
X
```

[6]:

	Usf	Usl	Usr
0	1	0	0
1	1	1	0
2	1	0	1
3	1	1	1
4	0	0	0
..	…	…	…
59	1	1	1

```
60    0    0    0
61    0    1    1
62    0    1    0
63    0    0    1

[64 rows x 3 columns]
```

[7]:
```
y
```

[7]:
```
0     1.0
1     1.0
2     0.0
3     2.0
4     3.0
     …
59    2.0
60    3.0
61    3.0
62    3.0
63    3.0
Name: next movement, Length: 64, dtype: float64
```

[8]:
```
print('X.shape=', X.shape, 'y.shape=', y.shape)
```

```
X.shape= (64, 3) y.shape= (64,)
```

1.0.4 Splitting data

[9]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

1.0.5 Finding best parameters

[11]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_dist = {
    'n_estimators': randint(100, 1000),
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': randint(4, 20),
    'min_samples_split': randint(2, 10),
    'min_samples_leaf': randint(1, 4),
    'bootstrap': [True, False]
}

rf = RandomForestClassifier()
random_search = RandomizedSearchCV(estimator = rf, param_distributions =␣
↪param_dist,
```

```
                                    n_iter = 100, cv = 3, n_jobs = -1, verbose =␣
 ↪2)
random_search.fit(X_train, y_train)
best_params = random_search.best_params_
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
c:\Users\sasin\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be
removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'`
or remove this parameter as it is also the default value for
RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
```

[12]:
```
best_params
```

[12]:
```
{'bootstrap': True,
 'max_depth': 15,
 'max_features': 'auto',
 'min_samples_leaf': 2,
 'min_samples_split': 4,
 'n_estimators': 134}
```

1.0.6 Model

[20]:
```
model = RandomForestClassifier(
    bootstrap=True,
    max_depth=3,
    max_features='auto',
    min_samples_leaf=2,
    min_samples_split=4,
    n_estimators=10
)
model.fit(X_train, y_train)
```

```
c:\Users\sasin\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:424:
FutureWarning: `max_features='auto'` has been deprecated in 1.1 and will be
removed in 1.3. To keep the past behaviour, explicitly set `max_features='sqrt'`
or remove this parameter as it is also the default value for
RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
```

[20]:
```
RandomForestClassifier(max_depth=3, max_features='auto', min_samples_leaf=2,
                       min_samples_split=4, n_estimators=10)
```

[21]:
```
y_pred = model.predict(X_test)
```

1.0.7 Evaluating model

[22]:
```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 1.00
```

1.0.8 Predicting

[17]:
```
# Sample input
sample_input = [[0, 0, 1, ]]

# Predict
y_pred = model.predict(sample_input)

print("Predicted class:", y_pred)
```

```
Predicted class: [3.]
```

```
c:\Users\sasin\anaconda3\Lib\site-packages\sklearn\base.py:439: UserWarning: X
does not have valid feature names, but RandomForestClassifier was fitted with
feature names
  warnings.warn(
```

[18]:
```
next_movement_mapping = {0: 'L', 1: 'R', 2: 'B', 3: 'F'}
predicted_movement = next_movement_mapping[y_pred[0]]
print(f'Predicted next movement: {predicted_movement}')
```

```
Predicted next movement: F
```

1.0.9 Converting the model to c++ for inject Arduino

[23]:
```
from micromlgen import port
from sklearn.ensemble import RandomForestClassifier
import json


# Convert the model to C code
c_code = port(model, platform='arduino')
print(c_code)
```

```
#pragma once
#include <cstdarg>
namespace Eloquent {
    namespace ML {
        namespace Port {
            class RandomForest {
                public:
                    /**
                    * Predict class for features vector
```

```
*/
int predict(float *x) {
    uint8_t votes[4] = { 0 };
    // tree #1
    if (x[0] <= 0.5) {
        votes[3] += 1;
    }

    else {
        if (x[1] <= 0.5) {
            if (x[2] <= 0.5) {
                votes[1] += 1;
            }

            else {
                votes[0] += 1;
            }
        }

        else {
            if (x[2] <= 0.5) {
                votes[1] += 1;
            }

            else {
                votes[2] += 1;
            }
        }
    }

    // tree #2
    if (x[2] <= 0.5) {
        if (x[0] <= 0.5) {
            votes[3] += 1;
        }

        else {
            votes[1] += 1;
        }
    }

    else {
        if (x[0] <= 0.5) {
            votes[3] += 1;
        }

        else {
            if (x[1] <= 0.5) {
```

```
            votes[0] += 1;
        }

        else {
            votes[2] += 1;
        }
    }
}

// tree #3
if (x[0] <= 0.5) {
    votes[3] += 1;
}

else {
    if (x[1] <= 0.5) {
        if (x[2] <= 0.5) {
            votes[1] += 1;
        }

        else {
            votes[0] += 1;
        }
    }

    else {
        if (x[2] <= 0.5) {
            votes[1] += 1;
        }

        else {
            votes[2] += 1;
        }
    }
}

// tree #4
if (x[2] <= 0.5) {
    if (x[1] <= 0.5) {
        if (x[0] <= 0.5) {
            votes[3] += 1;
        }

        else {
            votes[1] += 1;
        }
    }
```

```
    else {
        if (x[0] <= 0.5) {
            votes[3] += 1;
        }

        else {
            votes[1] += 1;
        }
    }
}

else {
    if (x[0] <= 0.5) {
        votes[3] += 1;
    }

    else {
        if (x[1] <= 0.5) {
            votes[0] += 1;
        }

        else {
            votes[2] += 1;
        }
    }
}

// tree #5
if (x[0] <= 0.5) {
    votes[3] += 1;
}

else {
    if (x[2] <= 0.5) {
        votes[1] += 1;
    }

    else {
        if (x[1] <= 0.5) {
            votes[0] += 1;
        }

        else {
            votes[2] += 1;
        }
    }
}
```

```
// tree #6
if (x[1] <= 0.5) {
    if (x[0] <= 0.5) {
        votes[3] += 1;
    }

    else {
        if (x[2] <= 0.5) {
            votes[1] += 1;
        }

        else {
            votes[0] += 1;
        }
    }
}

else {
    if (x[0] <= 0.5) {
        votes[3] += 1;
    }

    else {
        if (x[2] <= 0.5) {
            votes[1] += 1;
        }

        else {
            votes[2] += 1;
        }
    }
}

// tree #7
if (x[2] <= 0.5) {
    if (x[0] <= 0.5) {
        votes[3] += 1;
    }

    else {
        votes[1] += 1;
    }
}

else {
    if (x[0] <= 0.5) {
        votes[3] += 1;
    }
```

```
    else {
        if (x[1] <= 0.5) {
            votes[0] += 1;
        }

        else {
            votes[2] += 1;
        }
    }
}

// tree #8
if (x[2] <= 0.5) {
    if (x[1] <= 0.5) {
        if (x[0] <= 0.5) {
            votes[3] += 1;
        }

        else {
            votes[1] += 1;
        }
    }

    else {
        if (x[0] <= 0.5) {
            votes[3] += 1;
        }

        else {
            votes[1] += 1;
        }
    }
}

else {
    if (x[1] <= 0.5) {
        if (x[0] <= 0.5) {
            votes[3] += 1;
        }

        else {
            votes[0] += 1;
        }
    }

    else {
        if (x[0] <= 0.5) {
```

```
            votes[3] += 1;
        }

        else {
            votes[2] += 1;
        }
    }
}

// tree #9
if (x[1] <= 0.5) {
    if (x[0] <= 0.5) {
        votes[3] += 1;
    }

    else {
        if (x[2] <= 0.5) {
            votes[1] += 1;
        }

        else {
            votes[0] += 1;
        }
    }
}

else {
    if (x[0] <= 0.5) {
        votes[3] += 1;
    }

    else {
        if (x[2] <= 0.5) {
            votes[1] += 1;
        }

        else {
            votes[2] += 1;
        }
    }
}

// tree #10
if (x[1] <= 0.5) {
    if (x[2] <= 0.5) {
        if (x[0] <= 0.5) {
            votes[3] += 1;
        }
```

```
                        else {
                            votes[1] += 1;
                        }
                    }

                    else {
                        if (x[0] <= 0.5) {
                            votes[3] += 1;
                        }

                        else {
                            votes[0] += 1;
                        }
                    }
                }

                else {
                    if (x[0] <= 0.5) {
                        votes[3] += 1;
                    }

                    else {
                        votes[2] += 1;
                    }
                }

                // return argmax of votes
                uint8_t classIdx = 0;
                float maxVotes = votes[0];

                for (uint8_t i = 1; i < 4; i++) {
                    if (votes[i] > maxVotes) {
                        classIdx = i;
                        maxVotes = votes[i];
                    }
                }

                return classIdx;
            }

        protected:
        };
    }
}
}
```

[ ]: