Random Forests and Grid Search

Random Forests

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

https://en.wikipedia.org/wiki/Random_forest

- Trees are notorious for overfitting
- IID vs ID (Identically distributed but not Independent)
- Random Forest tries to reduce this correlation by building trees decorrelated from each other using depth, width, leaf node, etc. bagging
- More nodes means less bias but more variance
- Ensemble trees can have some with high bias and some with low bias
- Ensemble provides different perspectives of y while neural net uses dropout where weights remain the same, just not used

Cross Validation

Cross-validation, sometimes called rotation estimation or out-of-sample testing, is any of various similar model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set. Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations.

https://en.wikipedia.org/wiki/Cross-validation_(statistics)

Hyperparameter

In machine learning, a hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are derived via training. Hyperparameters can be classified as model hyperparameters, that cannot be inferred while fitting the machine to the training set because they refer to the model selection task, or algorithm hyperparameters, that in principle have no influence on the performance of the model but affect the speed and quality of the learning process.

https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)

Grid Search

The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a hold-out validation set.

https://en.wikipedia.org/wiki/Hyperparameter_optimization#Grid_search

```
# use the penguins dataset; species = y; explore the
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OneHotEncoder

df = sns.load_dataset('penguins')
print(df.info())
df.dropna(how='any', inplace=True)
X_train, X_test, y_train, y_test = train_test_split(d

X_train['sex'] = X_train['sex'].map({'Female':1,'Male
X_test['sex'] = X_test['sex'].map({'Female':1,'Male':

ohe = OneHotEncoder(categories='auto', drop='first',

ohe_train = ohe.fit_transform(X_train[['island']])
ohe_train = pd.DataFrame(ohe_train, columns=ohe.get_f
ohe_train.index = X_train.index
X_train = X_train.join(ohe_train)
X_train.drop(['island'], axis=1, inplace=True)
```

```
ohe_test = ohe.transform(X_test[['island']])
ohe_test = pd.DataFrame(ohe_test, columns=ohe.get_fea
ohe_test.index = X_test.index
X_test = X_test.join(ohe_test)
X_test.drop(['island'], axis=1, inplace=True)

y_train.value_counts()
y_train = y_train.map({'Adelie':0,'Gentoo':1, 'Chinst
y_test = y_test.map({'Adelie':0,'Gentoo':1, 'Chinstra
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            344 non-null    object 
 1   island             344 non-null    object 
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object 
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
None
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

hyperparameters = {
            'n_estimators': [50, 200],
            'criterion': ['entropy', 'gini'],
            'max_depth': [3, 4],
            'max_leaf_nodes': [7, 9],
            'bootstrap': [True, False]
            }
```

```
    grid_search = GridSearchCV(estimator = RandomForestCl
                               param_grid = hyperparamete
                               scoring = 'accuracy',
                               cv = 10)

    grid_search = grid_search.fit(X_train, y_train)

    best_accuracy = grid_search.best_score_
    best_parameters = grid_search.best_params_

    print('best accuracy', best_accuracy)
    print('best parameters', best_parameters)
```

```
best accuracy 0.992
best parameters {'bootstrap': True, 'criterion': 'entr
```

Our Final Model with Best (Hyper)Parameters

best_params from gridsearch: {'bootstrap': True, 'criterion': 'entropy', 'max_depth': 3, 'max_leaf_nodes': 7, 'n_estimators': 50}

```
model = RandomForestClassifier(
            n_estimators=50,
            criterion='entropy',
            max_depth=3,
            max_leaf_nodes=7,
            bootstrap=True
            )
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

model = RandomForestClassifier(n_estimators = 50,
                               criterion = 'entropy',
                               max_depth = 3,
                               max_leaf_nodes = 7,
                               bootstrap = True,
                               random_state = 42)

model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(accuracy_score(y_test, predictions))
```

```
0.9880952380952381
```

```
# build your final RandomForestClassifier model here
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

model = RandomForestClassifier(random_state = 42).set
model.fit(X_train, y_train)
predictions = model.predict(X_test)
print(accuracy_score(y_test, predictions))
print(model.get_params)
```

```
0.9880952380952381
<bound method BaseEstimator.get_params of RandomForest
                       n_estimators=200, random_state=
```