

Small data technique

Practice Session 15

Changho Suh

January 26, 2024

Outline

Implementation of **random forests**

Tasks: Iris plants classification

California housing price prediction

Handwritten digit classification

Random forests for Iris plants classification

```
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
```

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

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
```

↑
use all available CPU's

```
rnd_clf.fit(X_train, y_train)
```

```
y_pred = rnd_clf.predict(X_test)
```

```
print(rnd_clf.score(X_test, y_test))
```

0.9666666666666667

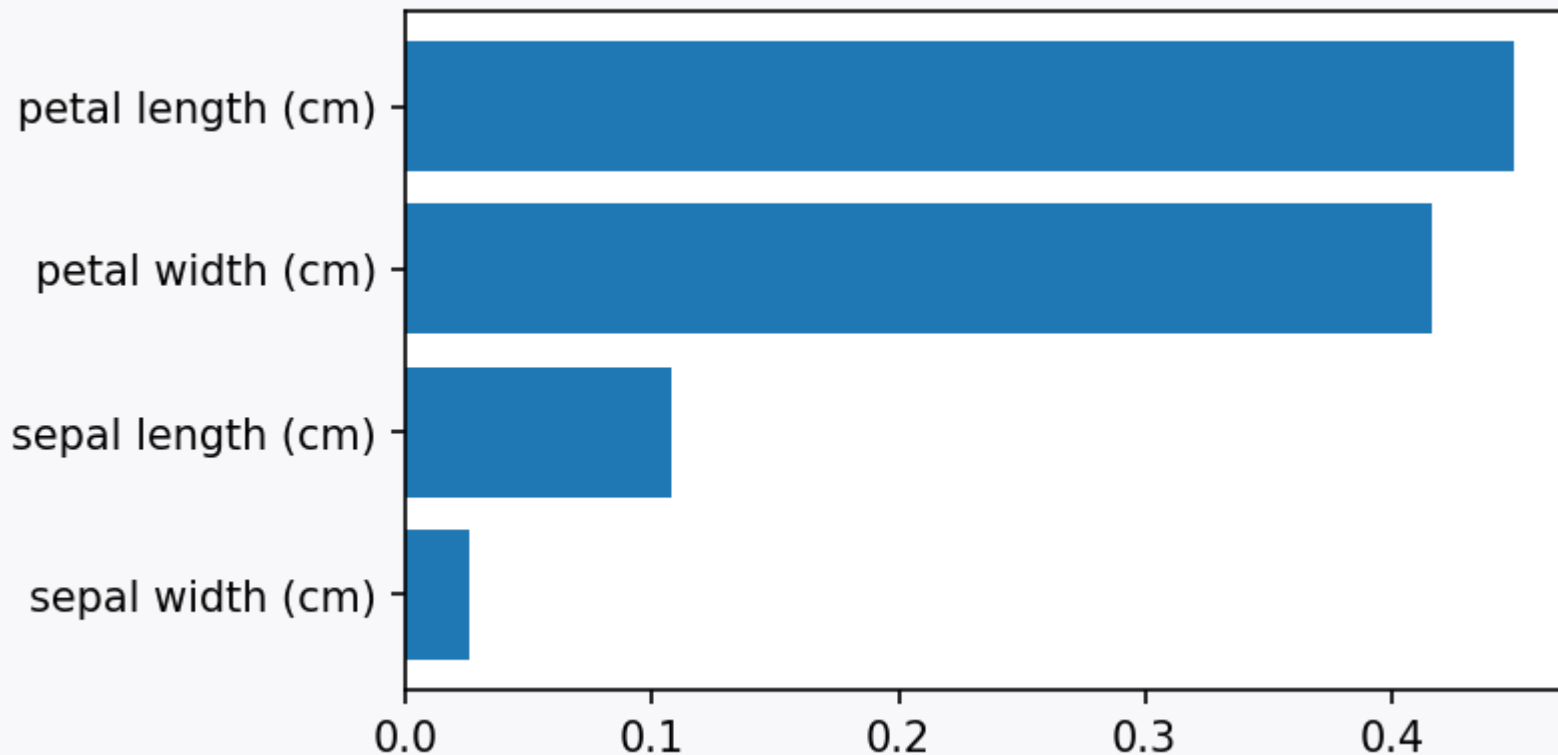
Feature importance

```
for name, score in zip(iris.feature_names, rnd_clf.feature_importances_):  
    print(name, score)
```

```
sepal length (cm) 0.08955824422368722  
sepal width (cm) 0.02154267813628801  
petal length (cm) 0.4454233433559405  
petal width (cm) 0.44347573428408404
```

Feature importance visualization

```
sorted_idx = rnd_clf.feature_importances_.argsort()  
plt.figure(figsize=(5,3), dpi=150)  
plt.barh(np.asarray(iris.feature_names)[sorted_idx],  
         rnd_clf.feature_importances_[sorted_idx])
```



Random forests for Cali House Price Pred.

```
from sklearn.datasets import fetch_california_housing
cali_prices = fetch_california_housing()
X_reg = cali_prices.data
y_reg = cali_prices.target

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

X_train, X_test, y_train, y_test = train_test_split(X_reg, y_reg, test_size = 0.01)
for_reg = RandomForestRegressor()
for_reg.fit(X_train, y_train)

y_pred_train = for_reg.predict(X_train)
y_pred_test = for_reg.predict(X_test)
print(mean_squared_error(y_pred_train, y_train))
print(mean_squared_error(y_pred_test, y_test))
```

```
0.03454954220428868
0.3383040535545396
```

Random forests for MNIST classification

```
from keras.datasets import mnist
from sklearn.model_selection import train_test_split

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train, X_test = X_train / 255., X_test / 255.

X_train = X_train.reshape(-1, 28*28)
X_test = X_test.reshape(-1, 28*28)

from sklearn.ensemble import RandomForestClassifier
rnd_clf = RandomForestClassifier(n_estimators=500,
                                max_leaf_nodes=16,
                                n_jobs=-1)

rnd_clf.fit(X_train, y_train)
y_pred = rnd_clf.predict(X_test)
print(rnd_clf.score(X_test, y_test))
```

0.8298

Feature importance visualization

```
import matplotlib.pyplot as plt
```

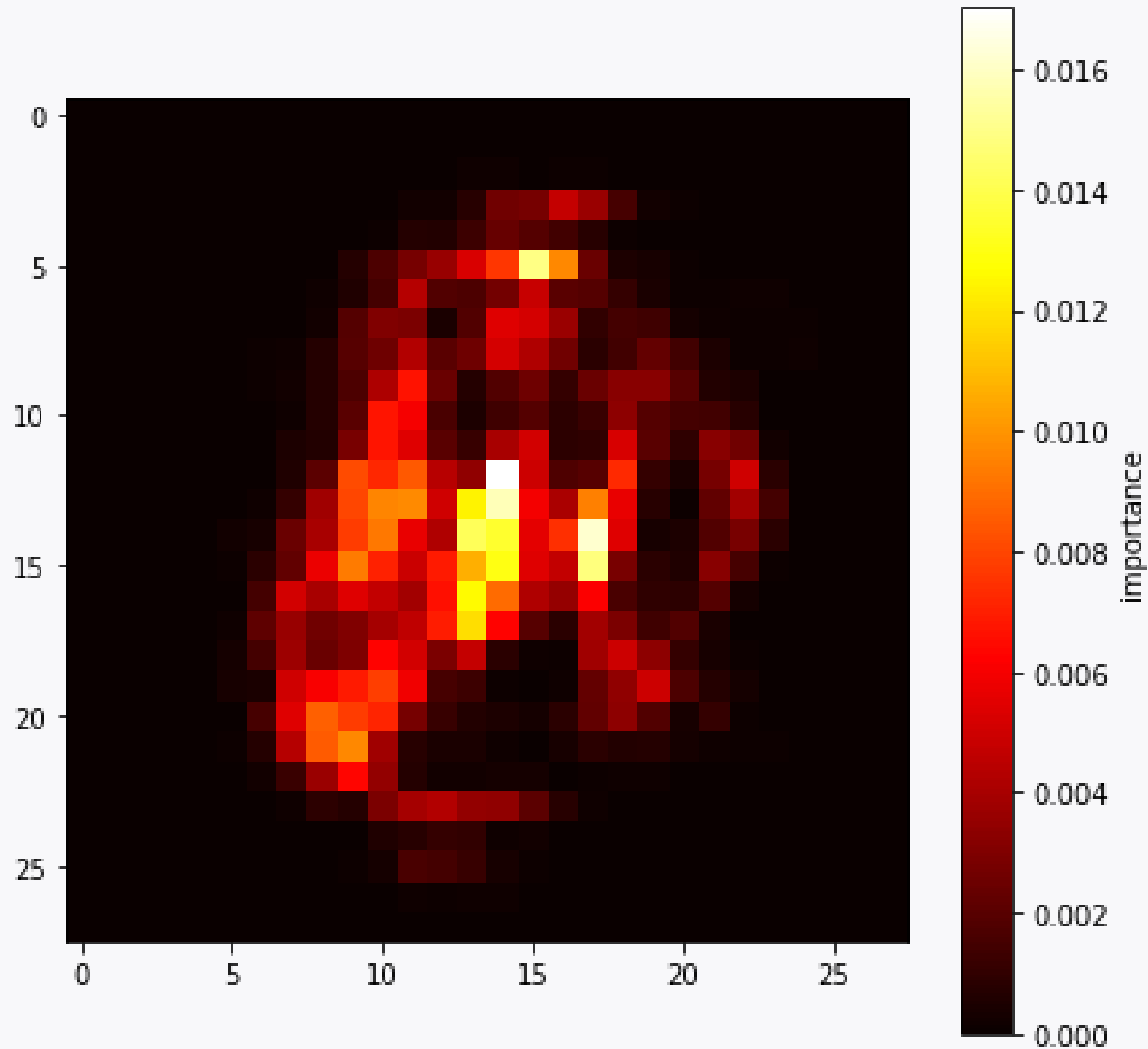
```
feature_importances = rnd_clf.feature_importances_  
feature_importances = feature_importances.reshape(28, 28)
```

```
plt.imshow(feature_importances, cmap='hot')
```

hot colormap

```
plt.colorbar(label='importance')  
plt.show()
```


Feature importance visualization



Hyperparameter search: GridSearchCV

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

from sklearn.model_selection import GridSearchCV

param_grid = {'n_estimators': [3, 10, 100, 500],
              'max_features': [0.25, 0.5, 0.75, 1]}

forest_clf = RandomForestClassifier(max_depth=2)
grid_search = GridSearchCV(forest_clf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

Hyperparameter search: GridSearchCV

```
print(grid_search.best_params_)
```

```
{'max_features': 0.75, 'n_estimators': 10}
```

```
print(grid_search.best_estimator_)
```

```
RandomForestClassifier(max_depth=2, max_features=0.75, n_estimators=10)
```

```
print(grid_search.best_score_)    0.9666666666666666
```

```
feature_importances = grid_search.best_estimator_.feature_importances_
```

```
print(feature_importances)
```

```
[0.          0.          0.31446851 0.68553149]
```

Hyperparameter search: GridSearchCV

```
from sklearn.metrics import accuracy_score
```

```
y_pred = grid_search.best_estimator_.predict(X_test)
```

```
print(accuracy_score(y_pred, y_test))
```

```
0.9333333333333333
```

```
print(grid_search.best_estimator_.score(X_test,y_test))
```

```
0.9333333333333333
```

RandomizedSearchCV

```
from sklearn.model_selection import RandomizedSearchCV

param_distributions = {'n_estimators': range(500),
                       'max_features': range(1, 5)}

forest_clf = RandomForestClassifier(max_depth=2)

randomized_search = RandomizedSearchCV(forest_clf,
                                       param_distributions,
                                       cv=4, n_iter = 50,
                                       scoring='accuracy')

randomized_search.fit(X_train, y_train)
```

RandomizedSearchCV

```
print(randomized_search.best_params_)
```

```
{'n_estimators': 419, 'max_features': 2}
```

```
print(randomized_search.best_estimator_)
```

```
RandomForestClassifier(max_depth=2, max_features=2, n_estimators=419)
```

```
print(randomized_search.best_score_)
```

```
0.9333333333333333
```

```
feature_importances_ = randomized_search.best_estimator_.feature_importances_  
print(feature_importances_)
```

```
[0.10610445 0.018726 0.41529867 0.45987088]
```

RandomizedSearchCV

```
y_pred = randomized_search.best_estimator_.predict(X_test)
```

```
print(accuracy_score(y_pred, y_test))
```

```
0.9666666666666667
```

```
print(randomized_search.best_estimator_.score(X_test,y_test))
```

```
0.9666666666666667
```