Decisonensemble

February 20, 2025

[25]:
```
# load libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets

#load data
iris = datasets.load_iris()
features = iris.data
target = iris.target

# Tree decision classifier
decisiontree = DecisionTreeClassifier(random_state=0)

# train the model

model = decisiontree.fit(features, target)

# create a new observation
observation = [[5, 4, 3, 2]]

# Prediction
model.predict(observation)
```

[25]:
```
array([1])
```

[27]:
```
# Print probabilities for the three classes
model.predict_proba(observation)
```

[27]:
```
array([[0., 1., 0.]])
```

[29]:
```
# load libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn import datasets#load data

#load data
iris = datasets.load_iris()
features = iris.data
target = iris.target
```

```
# Impurity measurment entropy
decisiontree_entropy = DecisionTreeClassifier(criterion='entropy',␣
 ↪random_state=0)

# train the model
model_entropy = decisiontree_entropy.fit(features, target)

# create a new observation
observation = [[5, 4, 3, 2]]

# Prediction
model.predict(observation)
```

[29]:
```
array([1])
```

[31]:
```
# Decision tree as a regressor
#load libraries
from sklearn.tree import DecisionTreeRegressor
from sklearn import datasets

#load a dataset with 2 features
diabetes = datasets.load_diabetes()
features = diabetes.data
target = diabetes.target

# Create a regressor
decisiontree = DecisionTreeRegressor(random_state=0)

# Learn model
model = decisiontree.fit(features, target)

# create observation
observation = [features[0]]

# Prediction
model.predict(observation)
```

[31]:
```
array([151.])
```

[43]:
```
# Decision tree as a regressor with mean absolute error
#load libraries
from sklearn.tree import DecisionTreeRegressor
from sklearn import datasets

#load a dataset with 2 features
diabetes = datasets.load_diabetes()
```

```
features = diabetes.data
target = diabetes.target

# Create a tree regressor using Mean absolute error
decisiontree_mae = DecisionTreeRegressor(criterion='absolute_error',␣
 ↪random_state=0)

# Learn model
model_mae = decisiontree_mae.fit(features, target)

# create observation
observation = [features[0]]

# Prediction
model.predict(observation)
```

[43]:
```
array([151.])
```

[47]:
```
# Random forest
# Load libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets

# Load data
iris = datasets.load_iris()
features = iris.data
target = iris.target

# Create forest fires
randomforest = RandomForestClassifier(random_state=0, n_jobs=-1)
# important paprameter rendom forest are
# max_features : fix max number of features taking into consideration in each␣
 ↪node , default number is the total number of features
# bootstrap :refer to a sampling technique used to create multiple training␣
 ↪datasets from the original dataset.
#This process is called Bootstrap Aggregation (or Bagging), and it helps improve␣
 ↪model accuracy and reduce overfitting.
# n_estimators : Fix the number of decision tree , default value is 10


#train model
model = randomforest.fit(features, target)
```

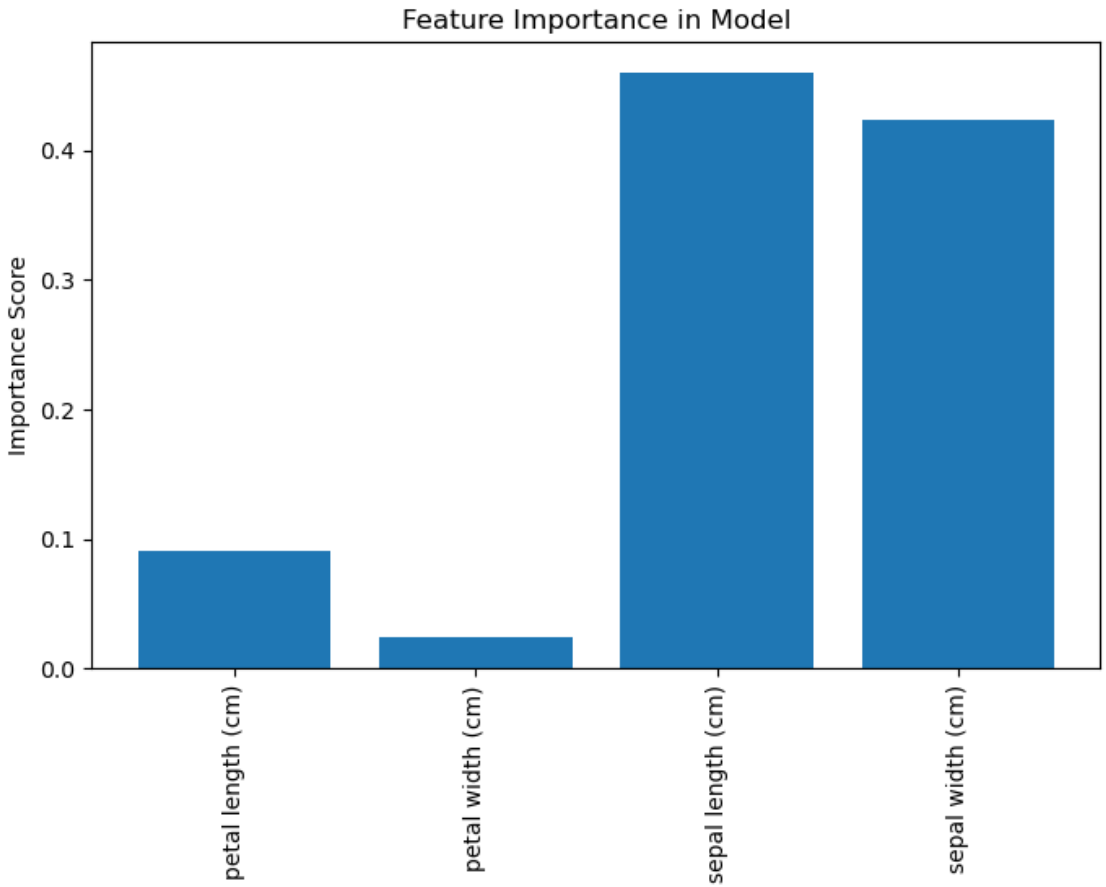
[49]:
```
# Evaluate random forest with out of bag error
#Load libraries

from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn import datasets

# Load data
iris = datasets.load_iris()
feautures = iris.data
target = iris.target

# Create forest fires
randomforest = RandomForestClassifier(random_state=0, n_estimators=1000,␣
 ↪oob_score=True, n_jobs=-1)
#train model
model = randomforest.fit(features, target)

#Print error out of the bag
randomforest.oob_score_
```

[49]: 0.9533333333333334

[50]:
```
#Load libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets

# Load data
iris = datasets.load_iris()
feautures = iris.data
target = iris.target

# Create forest fires
randomforest = RandomForestClassifier(random_state=0, n_jobs=-1)
#train model
model = randomforest.fit(features, target)

# important caracteristic
importances = model.feature_importances_

# order the importance
indices = np.argsort(importances)[::-1]
#rerganize the features name
names = [iris.feature_names[i] for i in indices]


# Create bar plot
plt.figure(figsize=(8, 5))
plt.bar(range(features.shape[1]), importances, tick_label=names)  # Corrected␣
 ↪tick labels
```

```
plt.xticks(rotation=90)  # Rotate feature names for readability
plt.ylabel("Importance Score")
plt.title("Feature Importance in Model")
plt.show()
```



[53]:
```
# importance of features
model.feature_importances_
```

[53]:
```
array([0.09090795, 0.02453104, 0.46044474, 0.42411627])
```

[59]:
```
#Random forest regressor
#Load libraries

from sklearn.ensemble import RandomForestRegressor
from sklearn import datasets

#load a dataset with 2 features
diabetes = datasets.load_diabetes()
features = diabetes.data
```

```
target = diabetes.target

#Regressor
randomforest = RandomForestRegressor(random_state=0, n_jobs=-1)

#train a model
model = randomforest.fit(features, target)
```

[67]:
```
# improve performances based on boosting
#Load libraries
from sklearn.ensemble import AdaBoostClassifier
from sklearn import datasets

#Load data
iris = datasets.load_iris()
features = iris.data
target = iris.target

# Create classifier
adaboost = AdaBoostClassifier(random_state=0)

#train
model = adaboost.fit(features, target)
```

```
C:\Users\Thinkpad\anaconda3\Lib\site-
packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```

[ ]:
```
# XGboost
#Load data
import xgboost as xgb
from sklearn import datasets, preprocessing
from sklearn.metrics import classification_report
from numpy import argmax

#Load data
iris = datasets.load_iris()
features = iris.data
target = iris.target

#create data
xgb_train = xgb.DMatrix(features, label=target)

#define parameters
param = {
    'objective':'multi:softprob',
```

```
    'num_class' : 3
    }
#train
gbm = xgb.train(param, xgb_train)
# Prediction
predictions = argmax(gbm.predict(gbm.predict(xgb_train)), axis=1)

print(classification_report(target, predictions))
```

[ ]: