# Improve Model Performance Using Boosting and Stacking

# Concept Session

## Demo - 7.1: Boosting with Adaboost

We will use the Ensemble methods: Boosting with Adaboost and Stacking for Classification. To compare the results, we will also evaluate a simple Decision Tree and Bagging with Random Forest.

In [ ]:
```
# general imports
import pandas as pd
import numpy as np
from matplotlib import pyplot
from numpy import mean
from numpy import std

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
# evaluation imports
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
```

### 1. Load the Data

**We use the UCI breast cancer dataset to classify tumors as being malignant or benign.**

**We use the scikit-learn API to import the dataset into our program.**

In [ ]:
```
# data imports
from sklearn.datasets import load_breast_cancer
from sklearn.preprocessing import LabelEncoder
```

In [ ]:
```
# load data
breast_cancer = load_breast_cancer()

X = pd.DataFrame(breast_cancer.data, columns=breast_cancer.feature_names)
y = pd.Categorical.from_codes(breast_cancer.target, breast_cancer.target_names)
```

**Since the label is categorical, it must be encoded as numbers. As the result, malignant is set to 1 and benign to 0.**

In [ ]:
```
# encode data
encoder = LabelEncoder()
binary_encoded_y = pd.Series(encoder.fit_transform(y))
```

### Learn Ensembles

We will evaluate all models using repeated stratified k-fold cross-validation, with three repeats and 10 folds.

We will report the mean and standard deviation of the F1-Score of the model across all repeats and folds.

In [ ]:
```
# define lists to gather results for plotting later
results, names = list(), list()
```

### 2. Baseline: Decision Tree Classifier (For comparison)

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [ ]:
```
# define the model
model = DecisionTreeClassifier()

# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, binary_encoded_y, scoring='f1', cv=cv, n_jobs=-1, error_score='rai
se')

results.append(n_scores)
names.append('cart')

# report performance
print('F1-Score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

### 3. Bagging with Random Forest (For comparison)

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
# define the model
model = RandomForestClassifier()

# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, binary_encoded_y, scoring='f1', cv=cv, n_jobs=-1, error_score='rai
se')

results.append(n_scores)
names.append('rf')

# report performance
print('F1-Score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

### 4. Boosting with Adaboost

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [ ]:
```
# define the model
model = AdaBoostClassifier()

# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, binary_encoded_y, scoring='f1', cv=cv, n_jobs=-1, error_score='rai
se')

results.append(n_scores)
names.append('ada')

# report performance
print('F1-Score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

## Demo - 7.2: Stacking

1. For Stacking, first, we choose the base model algorithms. Each algorithm will be evaluated using default model hyperparameters.

   Logistic Regression. k-Nearest Neighbors. Decision Tree. Support Vector Machine. Naive Bayes.

1. Next, we combine these five models into a single ensemble model using stacking.

We can use a logistic regression model to learn how to best combine the predictions from each of the separate five models.

The get_stacking() function below defines the StackingClassifier model by first defining the five base models, then defining the logistic regression meta-model to combine the predictions from the base models using 5-fold cross-validation.

In [ ]:
```
# required Python libraries
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import StackingClassifier
```

In [ ]:
```
# get a stacking ensemble of models
def get_stacking():
        # define the base models
        level0 = list()
        level0.append(('lr', LogisticRegression()))
        level0.append(('knn', KNeighborsClassifier()))
        level0.append(('cart', DecisionTreeClassifier()))
        level0.append(('svm', SVC()))
        level0.append(('bayes', GaussianNB()))
        # define meta learner model
        level1 = LogisticRegression()
        # define the stacking ensemble
        model = StackingClassifier(estimators=level0, final_estimator=level1, cv=5)
        return model
```

In [ ]:
```
# define the model
model = get_stacking()

# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, binary_encoded_y, scoring='f1', cv=cv, n_jobs=-1, error_score='rai
se')

results.append(n_scores)
names.append('stacking')

# report performance
print('F1-Score: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

### 3. Plot for Final Comparison

In [ ]:
```
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

### Conclusion

In [ ]:
```
```