Performance Evaluation, Ensemble Methods

Machine Learning

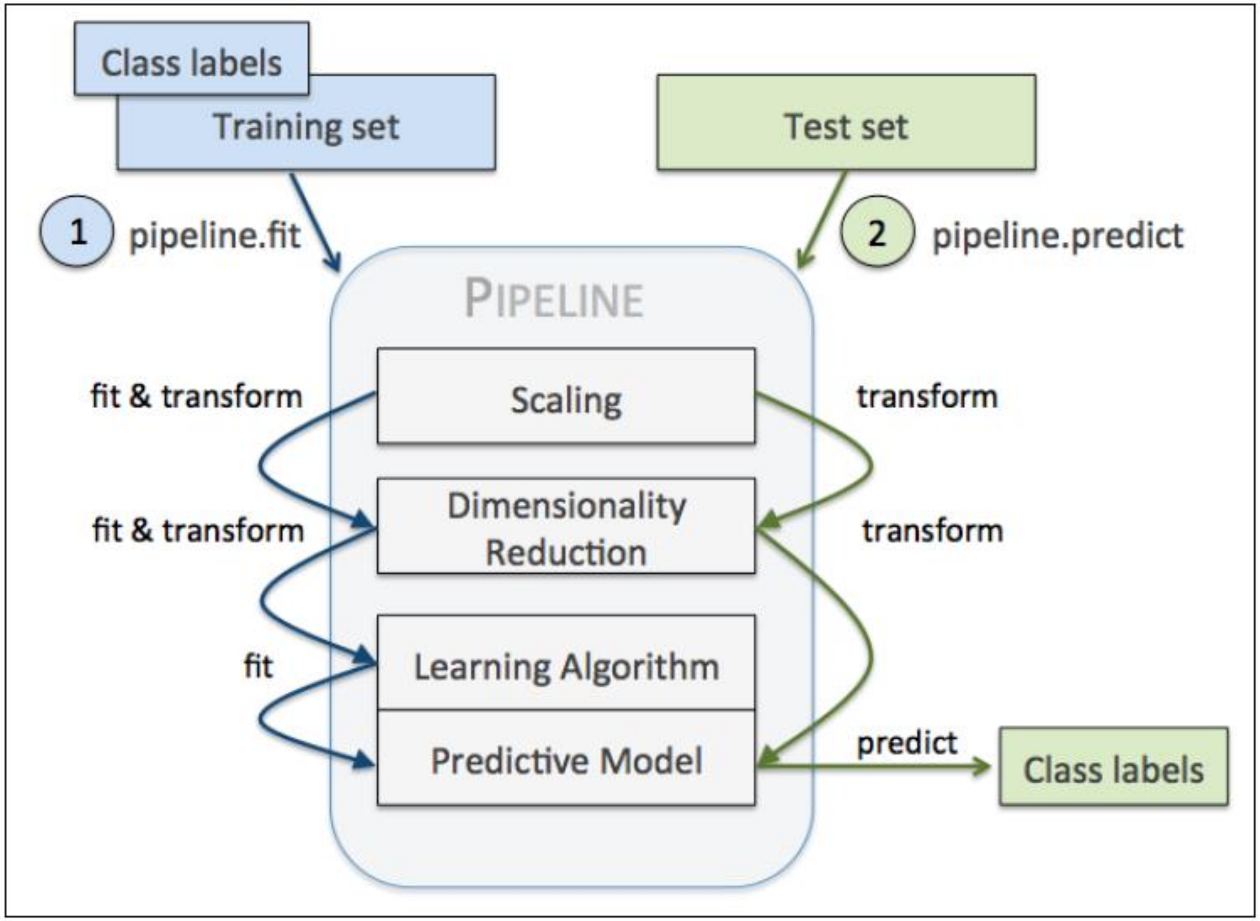
Contents

- Model Evaluation & Hyperparameter Tuning
  - Pipelining
  - K-fold Cross validation
  - Learning Curve
  - Validation Curve
  - Grid Search
  - Performance Evaluation Metrics

- Ensemble Methods
  - Bagging
  - Boosting

Model Evaluation & Hyperparameter Tuning

- Pipelining : Streamlining workflows with pipelines



Model Evaluation & Hyperparameter Tuning

- Loading the Breast Cancer Wisconsin dataset

```
import pandas as pd
df = pd.read_csv('https://archive.ics.uci.edu/ml/'
                 'machine-learning-databases'
                 '/breast-cancer-wisconsin/wdbc.data', header=None)
df.head()
df.shape
```

	0	1	2	3	4	
0	842302	M	17.99	10.38	122.80	10(
1	842517	M	20.57	17.77	132.90	13
2	84300903	M	19.69	21.25	130.00	12(
3	84348301	M	11.42	20.38	77.58	38
4	84358402	M	20.29	14.34	135.10	12

```
(569, 32)
```

```
from sklearn.preprocessing import LabelEncoder
# Get X, y. Encoding class label with scikit-learn
X = df.loc[:, 2:].values # remove 1st column(ID) & 2nd colum
y = df.loc[:, 1].values

le = LabelEncoder()
y = le.fit_transform(y)
X.shape
```

```
(569, 30)
```

```
from sklearn.model_selection import train_test_split
# train / test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.20,
                                                    stratify=y,
                                                    random_state=1)
X_train.shape
```

```
(455, 30)
```

Model Evaluation & Hyperparameter Tuning

- Combining transformers and estimators in a pipeline

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline, Pipeline

# make pipeline
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(random_state=1))
'''
# Instead you can use Pipeline class
pipe_lr = Pipeline(steps=[
     ('scale', StandardScaler()),
     ('PCA', PCA(n_components=2)),
     ('LR', LogisticRegression(random_state=1))
])
'''

# process using pipeline
pipe_lr.fit(X_train, y_train)

print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
```

```
Test Accuracy: 0.956
```

Model Evaluation & Hyperparameter Tuning

- Using k-fold cross validation to assess model performance

Model Evaluation & Hyperparameter Tuning

- Using k-fold cross validation to assess model performance

```
import numpy as np
from sklearn.model_selection import StratifiedKFold

# make indices for stratified k-fold cross validation
kfold = StratifiedKFold(n_splits=5,
                        shuffle=False, # random_state=1 sklean <= 0.22
                        ).split(X, y)

# train and compute test score for each set
scores = []
for k, (train, test) in enumerate(kfold):
    pipe_lr.fit(X[train], y[train])
    score = pipe_lr.score(X[test], y[test])
    scores.append(score)
    print('Fold: %2d, Class dist.: %s, Acc: %.3f' % (k+1,
          np.bincount(y[train]), score))

print('\nCV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

```
Fold:  1, Class dist.: [286 169], Acc: 0.939
                    ⋮
Fold:  5, Class dist.: [286 170], Acc: 0.982

CV accuracy: 0.951 +/- 0.016
```

Model Evaluation & Hyperparameter Tuning

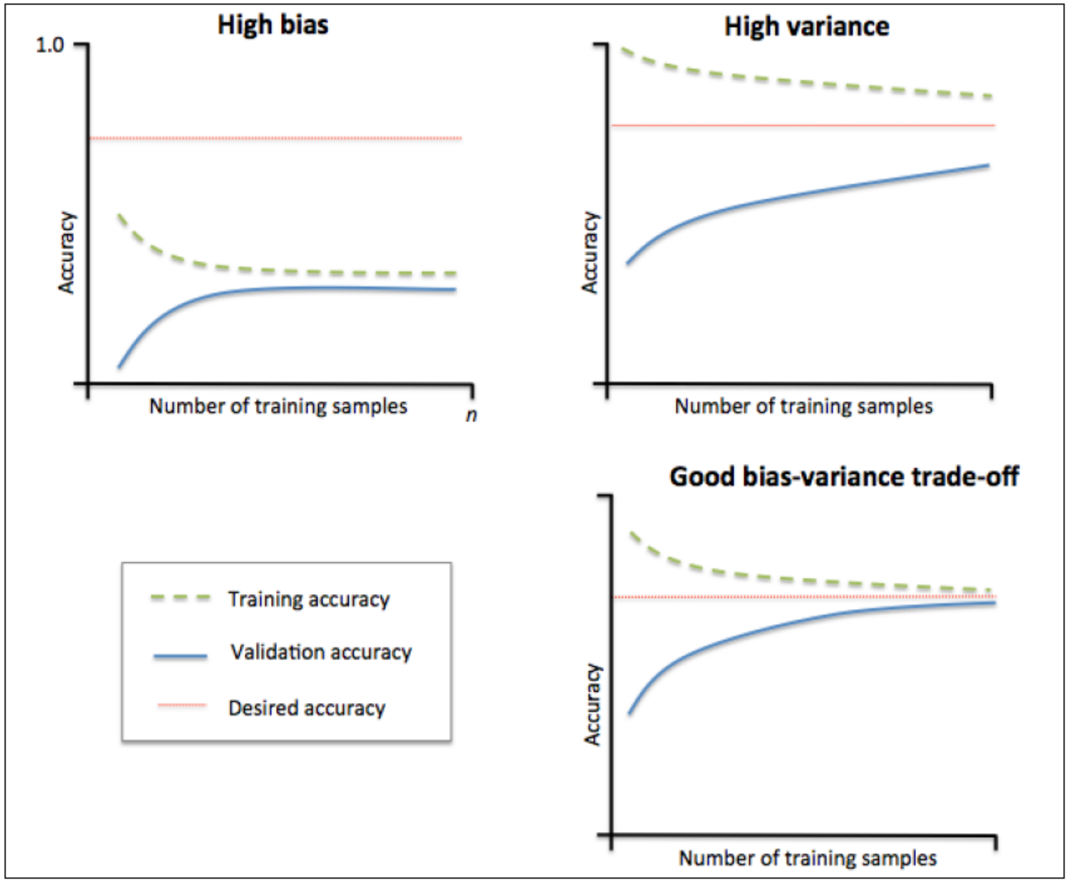
- Using k-fold cross validation to assess model performance

```
from sklearn.model_selection import cross_val_score

# cross validation using cross_val_score
scores = cross_val_score(estimator=pipe_lr,
                         X=X,
                         y=y,
                         cv=5,
                         n_jobs=1)
print('Scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```

```
Scores: [0.93859649 0.94736842 0.93859649 0.94736842 0.98230088]
CV accuracy: 0.951 +/- 0.016
```

Model Evaluation & Hyperparameter Tuning

- Checking performance with learning curves



Model Evaluation & Hyperparameter Tuning

- Checking performance with learning curves

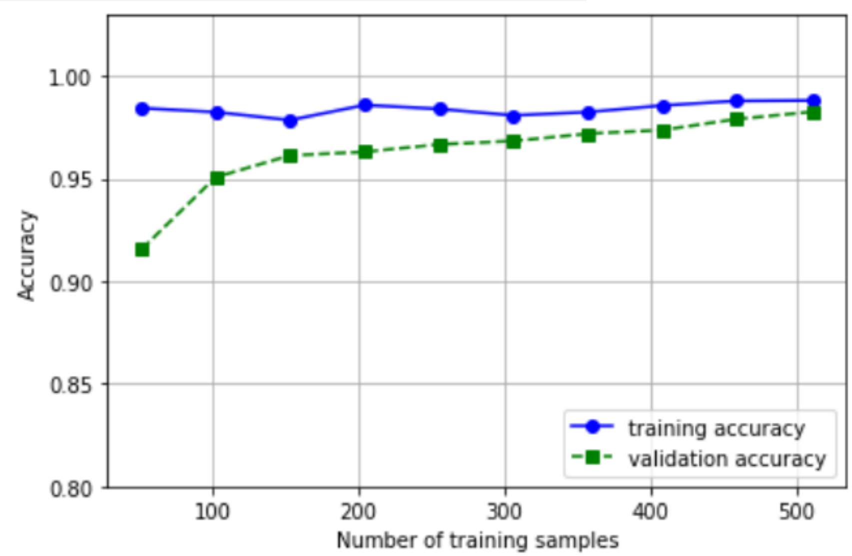
```
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve

pipe_lr = make_pipeline(StandardScaler(),
                        LogisticRegression(penalty='l2',
                                           random_state=1,
                                           max_iter=500))

# accuracies for different training set size
train_sizes, train_scores, test_scores =\
                        learning_curve(estimator=pipe_lr,
                        X=X,
                        y=y,
                        train_sizes=np.linspace(0.1, 1.0, 10),
                        cv=10,
                        n_jobs=1)
print(train_sizes)
print(train_scores.shape)
```

```
[ 51 102 153 204 256 307 358 409 460 512]
(10, 10)
```

Model Evaluation & Hyperparameter Tuning

- Checking performance with learning curves

```
train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)

plt.plot(train_sizes, train_mean,
           color='blue', marker='o',
           label='training accuracy')

plt.plot(train_sizes, test_mean,
           color='green', linestyle='--', marker='s',
           label='validation accuracy')

plt.grid()
plt.xlabel('Number of training samples')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.ylim([0.8, 1.03])
plt.tight_layout()

plt.show()
```



Model Evaluation & Hyperparameter Tuning

- Checking overfitting and underfitting with validation curves

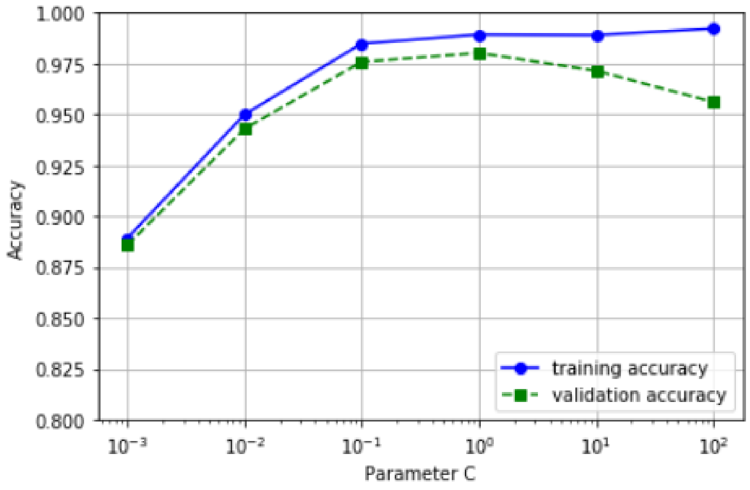
```
from sklearn.model_selection import validation_curve

# accuracies for different regularization parameters
param_range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
train_scores, test_scores = validation_curve(estimator=pipe_lr,
                                             X=X_train,
                                             y=y_train,
                                             param_name='logisticregression__C',
                                             param_range=param_range,
                                             cv=10)
print(param_range)
print(train_scores.shape)
```

```
[0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
(6, 10)
```

Model Evaluation & Hyperparameter Tuning

- Checking overfitting and underfitting with validation curves

```
train_mean = np.mean(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)

plt.plot(param_range, train_mean,
          color='blue', marker='o',
          label='training accuracy')

plt.plot(param_range, test_mean,
          color='green', linestyle='--', marker='s',
          label='validation accuracy')
plt.grid()
plt.xscale('log')
plt.legend(loc='lower right')
plt.xlabel('Parameter C')
plt.ylabel('Accuracy')
plt.ylim([0.8, 1.0])
plt.tight_layout()

plt.show()
```



Model Evaluation & Hyperparameter Tuning

- Tuning hyperparameters via grid search

```
from sklearn.model_selection import GridSearchCV

stdsc = StandardScaler()
X = stdsc.fit_transform(X)

# training with various parameter combinations
param_grid = [{'C': [0.01, 0.1, 1.0, 10.0, 100.0],
               'penalty': ['l1', 'l2']}]

gs = GridSearchCV(estimator=LogisticRegression(),
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=5,
                  n_jobs=-1)
gs = gs.fit(X, y)

means = gs.cv_results_['mean_test_score']
stds = gs.cv_results_['std_test_score']
params = gs.cv_results_['params']
for mean, std, params in zip(means, stds, params):
    print("%0.3f (+/-%0.3f) for %s" % (mean, std * 2, params))
print()
print("Best score:", gs.best_score_)
print("Parameters:", gs.best_params_)
```

```
nan (+/-nan) for {'C': 0.01, 'penalty': 'l1'}
0.949 (+/-0.026) for {'C': 0.01, 'penalty': 'l2'}
nan (+/-nan) for {'C': 0.1, 'penalty': 'l1'}
0.975 (+/-0.013) for {'C': 0.1, 'penalty': 'l2'}
nan (+/-nan) for {'C': 1.0, 'penalty': 'l1'}
0.981 (+/-0.013) for {'C': 1.0, 'penalty': 'l2'}
nan (+/-nan) for {'C': 10.0, 'penalty': 'l1'}
0.970 (+/-0.028) for {'C': 10.0, 'penalty': 'l2'}
nan (+/-nan) for {'C': 100.0, 'penalty': 'l1'}
0.963 (+/-0.037) for {'C': 100.0, 'penalty': 'l2'}

Best score: 0.9806862288464524
Parameters: {'C': 1.0, 'penalty': 'l2'}
```

Model Evaluation & Hyperparameter Tuning

- Tuning hyperparameters via grid search

```
# the best model
clf = gs.best_estimator_
clf.fit(X_train, y_train)
print('Test accuracy: %.3f' % clf.score(X_test, y_test))
```

```
Test accuracy: 0.947
```

Model Evaluation & Hyperparameter Tuning

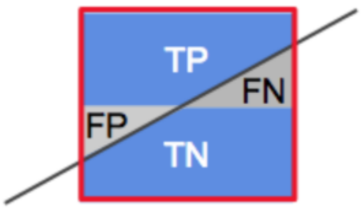
- Confusion Matrix
  - The confusion matrix is simply a square matrix that reports the counts of the *true positive*, *true negative*, *false positive*, and *false negative* predictions of a classifier

		Predicted class	
		P	N
Actual Class	P	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Model Evaluation & Hyperparameter Tuning

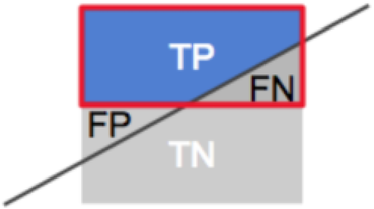
- ACC, TPR, FPR
  - Accuracy(ACC)

$$ACC = \frac{TP + TN}{FP + FN + TP + TN}$$



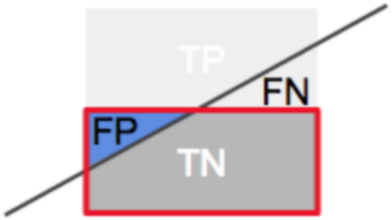
  - True Positive Rate(TPR)

$$TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$



  - False Positive Rate(FPR)

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN}$$

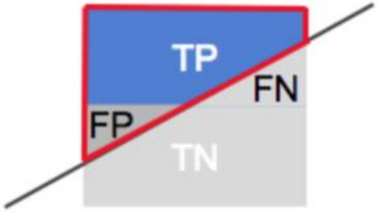


https://bcho.tistory.com/1206

Performance Evaluation Metrics

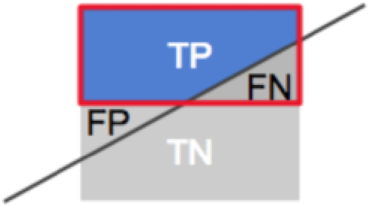
- Precision, Recall, F1
  - Precision(PRE)

$$PRE = \frac{TP}{TP + FP}$$



  - Recall(REC)

$$REC = TPR = \frac{TP}{P} = \frac{TP}{FN + TP}$$



  - $F1 - score$

$$F1 = 2\frac{PRE \times REC}{PRE + REC}$$

https://bcho.tistory.com/1206

Model Evaluation & Hyperparameter Tuning

- Confusion Matrix

```
from sklearn.metrics import confusion_matrix

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
confmat = confusion_matrix(y_true=y_test, y_pred=y_pred)
print(confmat)
```

```
[[68  4]
 [ 2 40]]
```

- Precision, Recall, and F1

```
from sklearn.metrics import precision_score, recall_score, f1_score

print('Precision: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
print('Recall: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))
```

```
Precision: 0.909
Recall: 0.952
F1: 0.930
```

Model Evaluation & Hyperparameter Tuning

- ROC (Receiver Operating Characteristic) Curve

```
from sklearn.metrics import roc_curve, auc
from scipy import interp

pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(penalty='l2',
                                           random_state=1,
                                           C=0.1))
X_train2 = X_train[:, [4, 5]]
X_test2 = X_test[:, [4, 5]]

# prediction probability of class 1
pipe_lr.fit(X_train2, y_train)
probas = pipe_lr.predict_proba(X_test2)
probas[:,1]
```

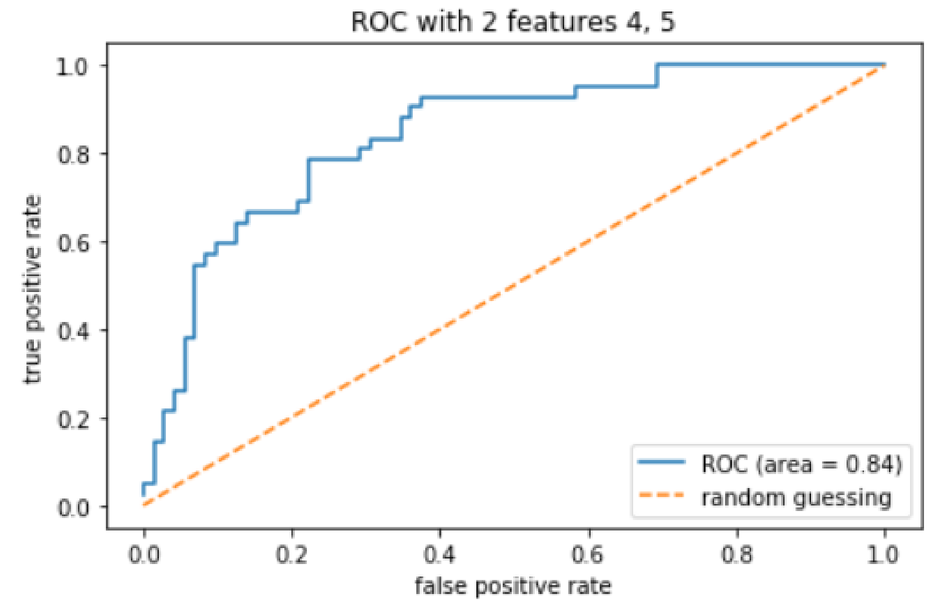
```
array([ 0.54725907,  0.16785314,  0.53630409,  0.64486632,  0.43203903,
        0.45691422,  0.86281954,  0.35614241,  0.32248996,  0.09339228,
        0.14273763,  0.13536069,  0.95496384,  0.26308029,  0.68040328,
        0.51767461,  0.43428621,  0.08286686,  0.37322008,  0.3374638 ,
        0.27978014,  0.14773322,  0.13018966,  0.23524479,  0.85951027,
```

⋮

Model Evaluation & Hyperparameter Tuning

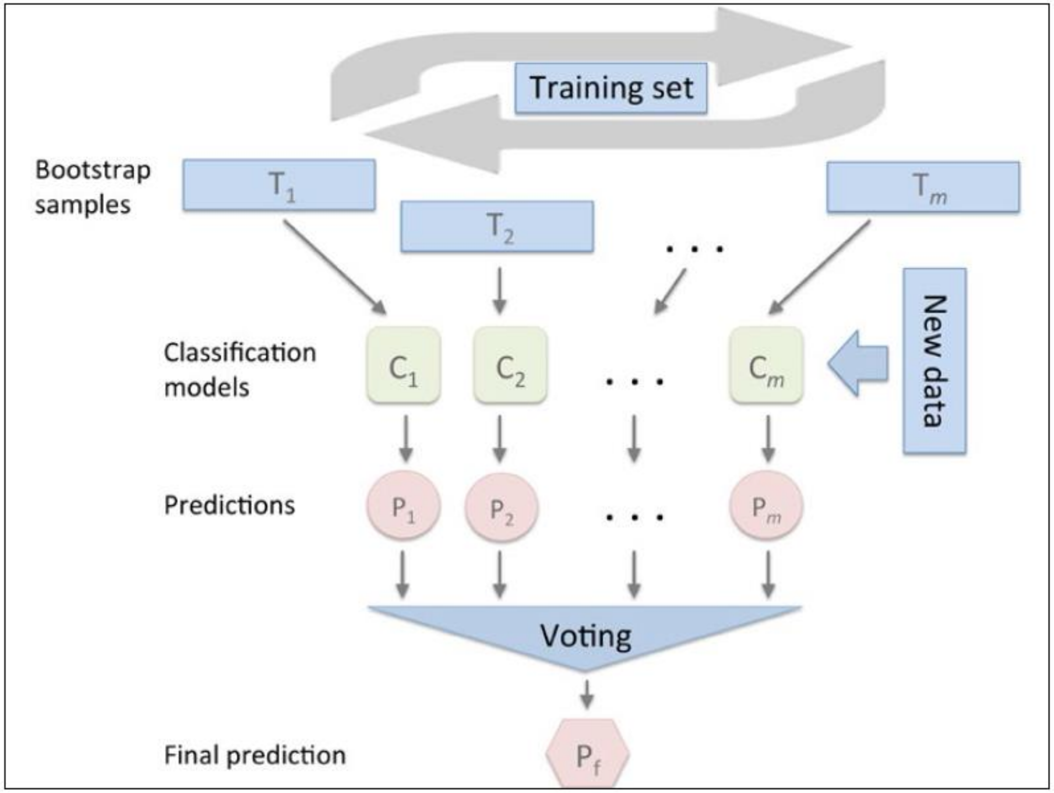
- ROC (Receiver Operating Characteristic) Curve

```
# FPR, TPR, AUC
fpr, tpr, thresholds = roc_curve(y_test, probas[:,1], pos_label=1)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr,
          label='ROC (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1],
          linestyle='--',
          label='random guessing')

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.title('ROC with 2 features 4, 5')
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.legend(loc="lower right")

plt.tight_layout()
plt.show()
```



Ensemble Methods : (1) Bagging

- Building an ensemble of classifiers from bootstrap samples
  - Bagging is an ensemble learning technique that is closely related to the Majority Vote Classifier, as illustrated in the following diagram



Ensemble Methods : (1) Bagging

- Building an ensemble of classifiers from bootstrap samples
  - Instead of using the same training set to fit the individual classifiers in the ensemble, we draw bootstrap samples (random samples with replacement) from the initial training set.

Sample indices	Bagging round 1	Bagging round 2	...
1	2	7	...
2	2	3	...
3	1	2	...
4	3	1	...
5	7	1	...
6	2	7	...
7	4	7	...

C_1 C_2 C_m

Pros

- Improve the accuracy of unstable models
- Decrease the degree of overfitting

Ensemble Methods : (1) Bagging

- Loading Wine Dataset

```
import pandas as pd

df_wine = pd.read_csv('https://archive.ics.uci.edu/ml/'
                      'machine-learning-databases/wine/wine.data',
                      header=None)

df_wine.columns = ['Class label', 'Alcohol', 'Malic acid', 'Ash',
                   'Alcalinity of ash', 'Magnesium', 'Total phenols',
                   'Flavanoids', 'Nonflavanoid phenols', 'Proanthocyanins',
                   'Color intensity', 'Hue', 'OD280/OD315 of diluted wines',
                   'Proline']
# drop 1 class
df_wine = df_wine[df_wine['Class label'] != 1]

y = df_wine['Class label'].values
X = df_wine[['Alcohol', 'OD280/OD315 of diluted wines']].values

X.shape
```

```
(119, 2)
```

Ensemble Methods : (1) Bagging

- Loading Wine Dataset

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

le = LabelEncoder()
y = le.fit_transform(y)
X_train, X_test, y_train, y_test =train_test_split(X,
                                                   y,
                                                   test_size=0.2,
                                                   random_state=1,
                                                   stratify=y)

X_train.shape
```

```
(95, 2)
```

Ensemble Methods : (1) Bagging

- Bagging

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

# decision tree classifier
tree = DecisionTreeClassifier(criterion='entropy',
                              max_depth=4,
                              random_state=1)
tree.fit(X_train, y_train)

# bag of trees classifier
bag = BaggingClassifier(base_estimator=tree,
                        n_estimators=100,
                        max_samples=0.5,
                        max_features=1.0,
                        bootstrap=True,
                        bootstrap_features=False,
                        n_jobs=1,
                        random_state=1)
bag.fit(X_train, y_train)

print('Tree training/test accuracy: %.2f / %.2f'
      % (tree.score(X_train, y_train), tree.score(X_test, y_test)))
print('Bag training/test accuracy: %.2f / %.2f'
      % (bag.score(X_train, y_train), bag.score(X_test, y_test)))
```

```
Tree training/test accuracy: 0.92 / 0.79
Bag training/test accuracy: 0.92 / 0.88
```

Ensemble Methods : (1) Bagging

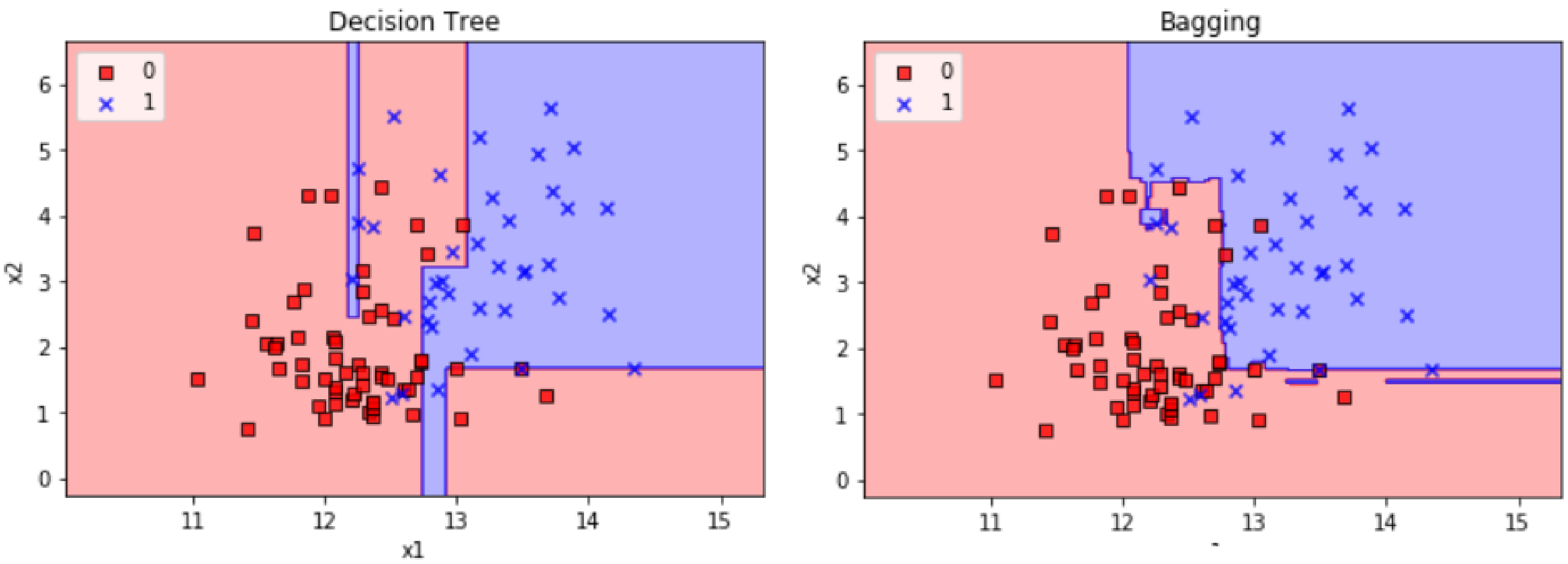
- Bagging

```
# decision boundary of the tree
plot_decision_regions(X_train, y_train, classifier=tree)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title("Decision Tree")
plt.legend(loc='upper left')
plt.show()
```

```
# decision boundary of the bag of trees classifier
plot_decision_regions(X_train, y_train, classifier=bag)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title("Bagging")
plt.legend(loc='upper left')
plt.show()
```

Ensemble Methods : (1) Bagging

- Bagging



dongguk UNIVERSITY

Ensemble Methods : (2) Boosting

- Leveraging weak learners via boosting
  - In contrast to bagging, the boosting algorithm learns simple classifiers k_j sequentially, and the final classifier is the weighted combination of k_j

$$C_m(x_i) = \alpha_1 k_1(x_i) + \alpha_2 k_2(x_i) + \ldots + \alpha_m k_m(x_i)$$

  - The key concept behind boosting is to focus on training samples that are hard to classify, that is, to let the weak learners subsequently learn from misclassified training samples to improve the performance of the ensemble
  - Example
    1. Draw random subset of training samples d_1 d without replacement from the training set D to train a weak learner C_1
    2. Draw second random training subset $d_2$without replacement from the training set and add 50 percent of the samples that were previously misclassified to train a weak learner C_2
    3. Find the training samples d_3 in the training set D on which C_1 and C_2 disagree to train a third weak learner C_3
    4. Combine the weak learners C_1 , C_2, and C_3 via majority voting

Ensemble Methods : (2) Boosting

- Boosting

```
from sklearn.ensemble import AdaBoostClassifier

# decision tree classifier
tree = DecisionTreeClassifier(criterion='entropy',
                                 max_depth=1,
                                 random_state=1)
tree.fit(X_train, y_train)

# boosting classifier
ada = AdaBoostClassifier(base_estimator=tree,
                                 n_estimators=100,
                                 learning_rate=0.1,
                                 random_state=1)
ada.fit(X_train, y_train)

print('Tree training/test accuracy: %.2f / %.2f'
     % (tree.score(X_train, y_train), tree.score(X_test, y_test)))
print('AdaBoost training/test accuracy: %.2f / %.2f'
     % (ada.score(X_train, y_train), ada.score(X_test, y_test)))
```

```
Tree training/test accuracy: 0.85 / 0.75
AdaBoost training/test accuracy: 0.93 / 0.88
```

Ensemble Methods : (2) Boosting

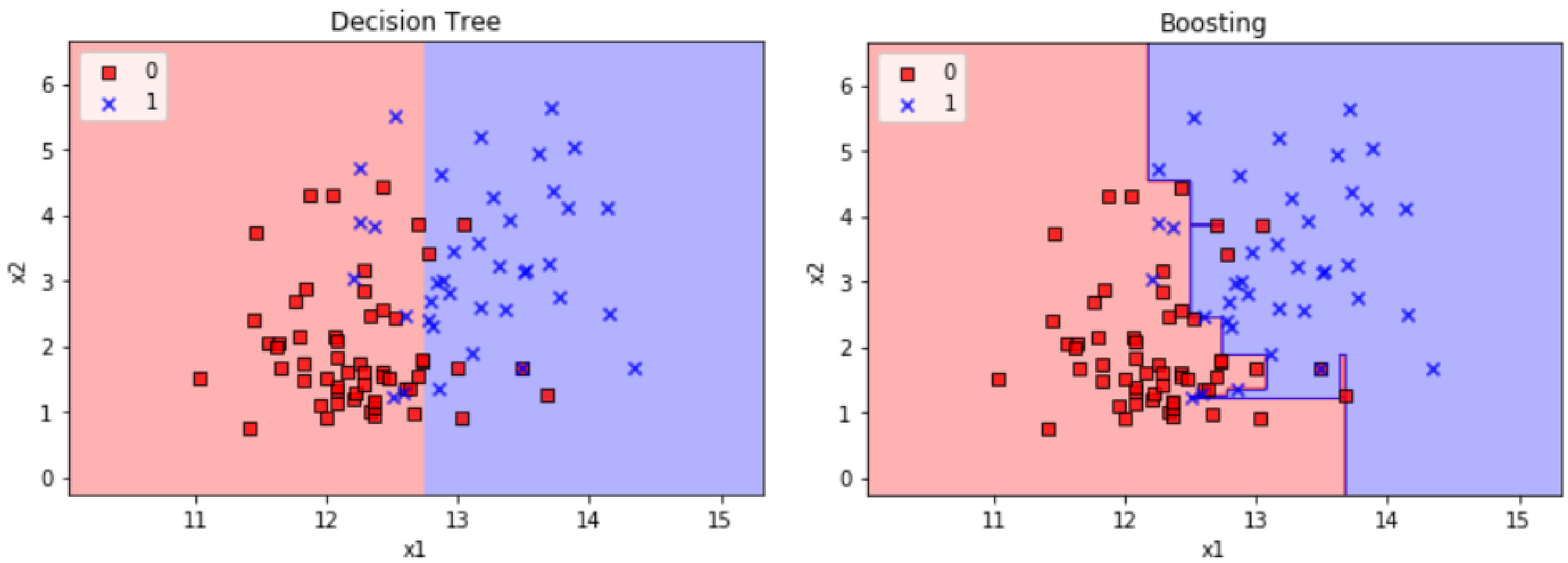
- Boosting

```
# decision boundary of the tree
plot_decision_regions(X_train, y_train, classifier=tree)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title("Decision Tree")
plt.legend(loc='upper left')
plt.show()
```

```
# decision boundary of the boosting classifier
plot_decision_regions(X_train, y_train, classifier=bag)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title("Boosting")
plt.legend(loc='upper left')
plt.show()
```

Ensemble Methods : (2) Boosting

- Boosting



Submit

- To make sure if you have completed this practice,
  Submit your practice file(Week09_givencode.ipynb) to e-class.
- **Deadline : tomorrow 11:59pm**
- Modify your ipynb file name as “Week09_StudentNum_Name.ipynb”
  Ex) **Week09_2020123456_홍길동.ipynb**
- You can upload this file without taking the quiz, but homework will be provided like a quiz every three weeks, so it is recommended to take the quiz as well.

Quiz 1 : Performance Evaluation

- Fine-tune your Logistic Regression model you build last week quiz
  - On the F1-Score,
    - Do K-fold Cross Validation
    - Plot Learning curve
    - Plot Validation curve
    - Find optimal hyperparameter using Grid Search

https://www.kaggle.com/c/titanic/data

Quiz 2 : Ensemble Methods

- Build a Bagging Classifier using decision tree classifiers

- Build a AdaBoosting Classifier using decision tree classifiers

https://www.kaggle.com/c/titanic/data