Task

Practice the ensemble methods with scikit-learn APIs

- Voting(hard, soft)
- Bagging
- Stacking
- Adaboost
- Gradient Boosting
- Compare ensemble methods with weak learns on the wine dataset

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.datasets import make_classification

from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import StackingClassifier
from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import make_classification

from sklearn.datasets import make_hastie_10_2
from sklearn.ensemble import GradientBoostingClassifier

import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import (
    BaggingClassifier,
    RandomForestClassifier,
    StackingClassifier,
    AdaBoostClassifier,
    GradientBoostingClassifier,
)
from sklearn.metrics import accuracy_score, roc_curve, auc, roc_auc_score
```

Voting Classifier

```
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(n_estimators=50, random_state=1)
clf3 = SVC(probability=True)
X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
y = np.array([1, 2, 1, 2, 2, 1])

# Hard voting
eclf1 = VotingClassifier(estimators=[
        ('lr', clf1), ('rf', clf2), ('svc', clf3)], voting='hard') # hard voting
eclf1 = eclf1.fit(X, y)
print("Hard voting: ", eclf1.predict(X))
print("Hard voting accuracy: ", eclf1.score(X, y))

# Soft voting
eclf2 = VotingClassifier(estimators=[
```

```
       ('lr', clf1), ('rf', clf2), ('svc', clf3)],
       voting='soft', weights=[1,1,1]) # soft voting
eclf2 = eclf2.fit(X, y)
print("Soft voting: ", eclf2.predict(X))
print("Soft voting accuracy: ", eclf2.score(X, y))
```

```
Hard voting:  [1 1 1 2 2 1]
Hard voting accuracy:  0.8333333333333334
Soft voting:  [1 2 1 2 2 1]
Soft voting accuracy:  1.0
```

Bagging Classifier

```
# Generate example data
X, y = make_classification(n_samples=100, n_features=4,
                           n_informative=2, n_redundant=0,
                           random_state=0, shuffle=False)

# Bagging with SVC estimator
clf_svc = BaggingClassifier(estimator=SVC(),
                        n_estimators=20, random_state=42).fit(X, y)
print("SVC Bagging Prediction:", clf_svc.predict([[0, 0, 0, 0]]))
print("SVC Bagging accuracy: ", clf_svc.score(X, y))

# Bagging with DecisionTreeClassifier estimator
clf_dt = BaggingClassifier(estimator=DecisionTreeClassifier(),
                       n_estimators=20, random_state=42).fit(X, y)
print("Decision Tree Bagging Prediction:", clf_dt.predict([[0, 0, 0, 0]]))
print("DT Bagging accuracy: ", clf_dt.score(X, y))
```

```
SVC Bagging Prediction: [1]
SVC Bagging accuracy:  0.92
Decision Tree Bagging Prediction: [1]
DT Bagging accuracy:  0.99
```

Stacking Classifier

```
X, y = load_iris(return_X_y=True)

# Set base estimators
estimators = [
    ('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
    ('svr', make_pipeline(StandardScaler(),
                          LinearSVC(random_state=42)))
]

# Set meta-model estimator
clf = StackingClassifier(
    estimators=estimators, final_estimator=LogisticRegression()
)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, stratify=y, random_state=42
)

clf.fit(X_train, y_train).score(X_test, y_test)
```

```
0.9473684210526315
```

Adaboost

```
X, y = make_classification(n_samples=1000, n_features=4,
                           n_informative=2, n_redundant=0,
                           random_state=0, shuffle=False)
clf = AdaBoostClassifier(n_estimators=100, random_state=0)
clf.fit(X, y)
print("Predicted label: ", clf.predict([[0, 0, 0, 0]]))
print("Adaboost accuracy: ", clf.score(X, y))
```

```
Predicted label:  [1]
Adaboost accuracy:  0.96
```

Gradient Boosting

```
X, y = make_hastie_10_2(random_state=0)
X_train, X_test = X[:2000], X[2000:]
y_train, y_test = y[:2000], y[2000:]

clf = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,
    max_depth=1, random_state=0).fit(X_train, y_train)
print("Predicted label: ", clf.predict(X_test[:10]))
print("Gradient Boosting accuracy: ", clf.score(X_test, y_test))
```

```
Predicted label:  [ 1. -1. -1.  1. -1. -1.  1. -1.  1.  1.]
Gradient Boosting accuracy:  0.913
```

Compare ensemble methods with weak learners on wine dataset

```
# Load the Wine dataset
wine = load_wine()
X, y = wine.data, wine.target

# Binary classification for ROC/AUC (considering only two classes)
X_binary = X[y != 2]
y_binary = y[y != 2]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
X_train_binary, X_test_binary, y_train_binary, y_test_binary = train_test_split(X_binary, y_binary, test_size=0.3, random_state=42)
```

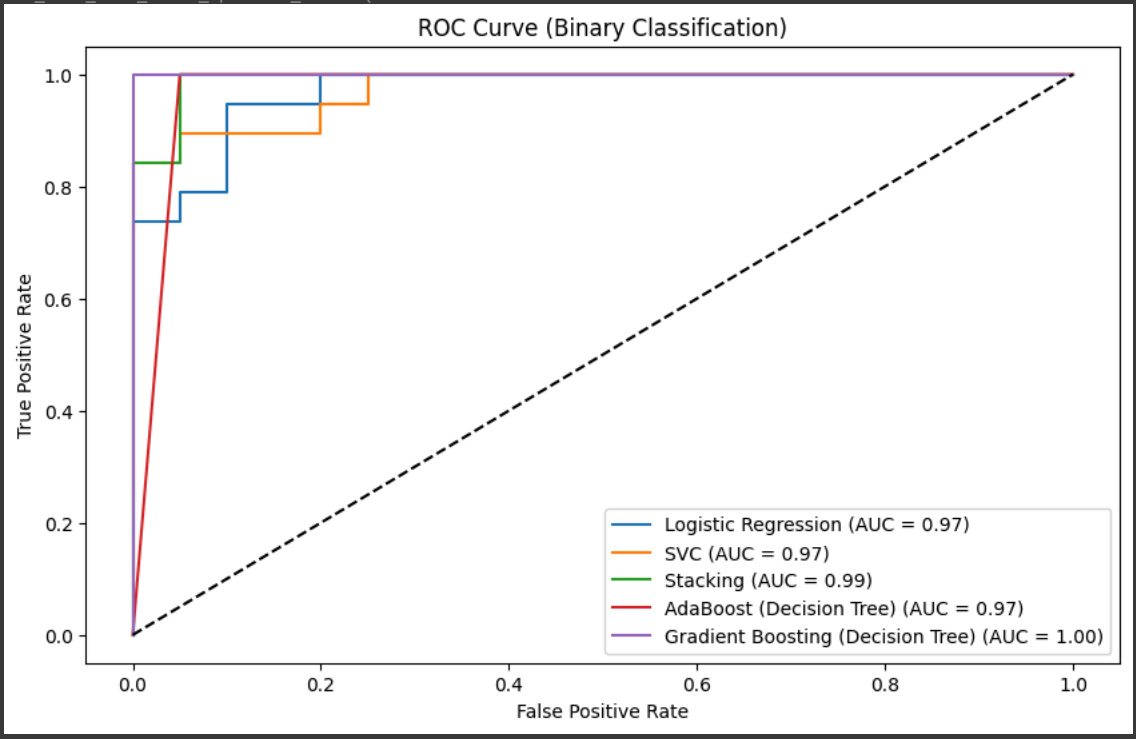
```
# Weak Learners
weak_learners = {
    'Logistic Regression': LogisticRegression(max_iter=100),
    'SVC': SVC(probability=True),
    'Decision Tree': DecisionTreeClassifier(),
}

# Ensemble Methods
ensemble_methods = {
    'Bagging (Decision Tree)': BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=50),
    'Random Forest': RandomForestClassifier(n_estimators=50),
    'Stacking': StackingClassifier(
        estimators=[('svc', SVC(probability=True)), ('dt', DecisionTreeClassifier())],
        final_estimator=LogisticRegression(max_iter=1000)
    ),
    'AdaBoost (Decision Tree)': AdaBoostClassifier(estimator=DecisionTreeClassifier(), n_estimators=50),
    'Gradient Boosting (Decision Tree)': GradientBoostingClassifier(n_estimators=50),
}

# Train and evaluate
all_models = {**weak_learners, **ensemble_methods} #Combine two dict.
accuracies = {}
roc_auc_scores = {}

for name, model in all_models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracies[name] = accuracy_score(y_test, y_pred)

    if name in ['Logistic Regression', 'SVC', 'Stacking', 'AdaBoost (Decision Tree)', 'Gradient Boosting (Decision Tree)']:
        model.fit(X_train_binary, y_train_binary)
        try:
            y_pred_proba = model.predict_proba(X_test_binary)[:, 1]
            roc_auc_scores[name] = roc_auc_score(y_test_binary, y_pred_proba)
        except AttributeError:
            roc_auc_scores[name] = np.nan
    else:
        roc_auc_scores[name] = np.nan

# Print accuracies
print("Model Accuracies:")
```

```
for name, accuracy in accuracies.items():
    print(f"{name}: {accuracy:.4f}")
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Model Accuracies:
Logistic Regression: 0.9815
SVC: 0.7593
Decision Tree: 0.9630
Bagging (Decision Tree): 0.9630
Random Forest: 1.0000
Stacking: 0.9630
AdaBoost (Decision Tree): 0.9444
Gradient Boosting (Decision Tree): 0.9074
```

```
# ROC Curve plot
plt.figure(figsize=(10, 6))
for name in ['Logistic Regression', 'SVC', 'Stacking', 'AdaBoost (Decision Tree)', 'Gradient Boosting (Decision Tree)']:
    if name in ['Logistic Regression', 'SVC', 'Stacking', 'AdaBoost (Decision Tree)', 'Gradient Boosting (Decision Tree)']:
        model = all_models[name]
        model.fit(X_train_binary, y_train_binary)
        try:
            y_pred_proba = model.predict_proba(X_test_binary)[:, 1]
            fpr, tpr, thresholds = roc_curve(y_test_binary, y_pred_proba)
            roc_auc = auc(fpr, tpr)
            plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')
        except AttributeError:
            pass

plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve (Binary Classification)')
plt.legend(loc='lower right')
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```



Your work

Practice the sklearn ensemble methods: voting, stacking, bagging, adaboost and gradient boosting. Example is accessible via the link below.

https://scikit-learn.org/stable/api/sklearn.ensemble.html

Submit your notebook in PDF format to BrightSpace by 3/30/2025 11:59 pm.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer

# Load the dataset
url = "https://github.com/mwaskom/seaborn-data/raw/master/penguins.csv"
df = pd.read_csv(url)

# Handle missing values
imputer = SimpleImputer(strategy='most_frequent')
df = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)

# Encode categorical variables
label_encoder = LabelEncoder()
df['species'] = label_encoder.fit_transform(df['species'])
df['island'] = label_encoder.fit_transform(df['island'])
df['sex'] = label_encoder.fit_transform(df['sex'])

# Split the data into features and target
X = df.drop('species', axis=1)
y = df['species']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.svm import SVC

# Define base classifiers
clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(n_estimators=50, random_state=1)
clf3 = SVC(probability=True, random_state=1)

# Hard Voting
eclf1 = VotingClassifier(estimators=[
    ('lr', clf1), ('rf', clf2), ('svc', clf3)], voting='hard')
eclf1.fit(X_train, y_train)
print("Hard Voting Predictions: ", eclf1.predict(X_test))
print("Hard Voting Accuracy: ", eclf1.score(X_test, y_test))

# Soft Voting
eclf2 = VotingClassifier(estimators=[
    ('lr', clf1), ('rf', clf2), ('svc', clf3)], voting='soft')
eclf2.fit(X_train, y_train)
print("Soft Voting Predictions: ", eclf2.predict(X_test))
print("Soft Voting Accuracy: ", eclf2.score(X_test, y_test))
```

```
Hard Voting Predictions:  [1 1 2 1 2 2 2 2 0 2 2 0 0 0 2 0 0 2 2 0 0 0 2 0 1 1 0 1 0 1 0 0 1 0 0 0 2
 1 0 1 2 2 2 0 0 0 2 0 0 1 0 2 2 0 0 1 0 1 1 2 1 0 2 0 2 0 0 0 0]
Hard Voting Accuracy:  0.9855072463768116
Soft Voting Predictions:  [1 1 2 1 2 2 2 2 0 2 2 0 0 0 2 0 0 2 2 0 0 0 2 0 1 1 0 1 0 1 0 0 1 0 0 0 2
 1 0 1 2 2 2 0 0 0 2 0 0 1 0 2 2 0 0 1 0 1 1 2 1 0 2 0 2 0 0 0 0]
Soft Voting Accuracy:  0.9855072463768116
```

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

# Bagging with SVC
clf_svc = BaggingClassifier(estimator=SVC(), n_estimators=20, random_state=42)
clf_svc.fit(X_train, y_train)
print("SVC Bagging Predictions: ", clf_svc.predict(X_test))
print("SVC Bagging Accuracy: ", clf_svc.score(X_test, y_test))

# Bagging with Decision Tree
clf_dt = BaggingClassifier(estimator=DecisionTreeClassifier(), n_estimators=20, random_state=42)
clf_dt.fit(X_train, y_train)
print("Decision Tree Bagging Predictions: ", clf_dt.predict(X_test))
print("Decision Tree Bagging Accuracy: ", clf_dt.score(X_test, y_test))
```

```
SVC Bagging Predictions:  [1 1 2 1 2 2 2 2 0 2 2 0 0 0 2 0 0 2 2 0 0 0 2 0 1 1 0 1 0 1 0 0 1 0 0 0 2
 1 0 1 2 2 2 0 0 0 2 0 0 1 0 2 2 0 0 1 0 1 1 2 1 0 2 0 2 0 0 0 0]
SVC Bagging Accuracy:  0.9855072463768116
Decision Tree Bagging Predictions:  [1 1 2 1 1 2 2 2 0 2 2 0 0 0 2 0 0 2 2 0 0 0 2 0 1 1 0 1 0 1 0 0 1 0 0 0 2
 1 0 1 2 2 2 0 0 0 2 0 0 1 0 2 2 0 0 1 0 1 1 2 1 0 2 0 2 0 1 0 0]
Decision Tree Bagging Accuracy:  0.9565217391304348
```

```
from sklearn.ensemble import StackingClassifier
from sklearn.pipeline import make_pipeline
from sklearn.svm import LinearSVC

# Set base estimators
estimators = [
    ('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
    ('svr', make_pipeline(StandardScaler(), LinearSVC(random_state=42)))
]

# Set meta-model estimator
clf_stack = StackingClassifier(
    estimators=estimators, final_estimator=LogisticRegression()
)

# Train and evaluate
clf_stack.fit(X_train, y_train)
print("Stacking Predictions: ", clf_stack.predict(X_test))
print("Stacking Accuracy: ", clf_stack.score(X_test, y_test))
```

```
Stacking Predictions:  [1 1 2 1 2 2 2 2 0 2 2 0 0 0 2 0 0 2 2 0 0 0 2 0 1 1 0 1 0 1 0 0 1 0 0 0 2
 1 0 1 2 2 2 0 0 0 2 0 0 1 0 2 2 0 0 1 0 1 1 2 1 0 2 0 2 0 0 0 0]
Stacking Accuracy:  0.9855072463768116
```

```
from sklearn.ensemble import AdaBoostClassifier

# AdaBoost
clf_ada = AdaBoostClassifier(n_estimators=100, random_state=0)
clf_ada.fit(X_train, y_train)
print("AdaBoost Predictions: ", clf_ada.predict(X_test))
print("AdaBoost Accuracy: ", clf_ada.score(X_test, y_test))
```

```
AdaBoost Predictions:  [1 1 2 1 2 2 2 2 0 2 2 0 0 0 2 0 0 2 2 0 0 0 2 0 1 1 0 1 0 1 0 0 1 0 0 0 2
 1 0 1 2 2 2 0 0 0 2 0 0 1 0 2 2 0 0 1 0 1 1 2 1 0 2 0 2 0 1 0 0]
AdaBoost Accuracy:  0.9710144927536232
```

```
from sklearn.ensemble import GradientBoostingClassifier

# Gradient Boosting
clf_gb = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=0)
clf_gb.fit(X_train, y_train)
print("Gradient Boosting Predictions: ", clf_gb.predict(X_test))
print("Gradient Boosting Accuracy: ", clf_gb.score(X_test, y_test))
```

```
Gradient Boosting Predictions:  [1 1 2 1 2 2 2 2 0 2 2 0 0 0 2 0 0 2 2 0 0 0 2 0 1 1 0 1 0 1 0 0 1 0 0 0 2
 1 0 1 2 2 2 0 0 0 2 0 0 1 0 2 2 0 0 1 0 1 1 2 1 0 2 0 2 0 0 0 0]
Gradient Boosting Accuracy:  0.9855072463768116
```