과제 #2: moons classification

- Run classifiers to classify **noised moons** dataset
  - Logistic Regression
  - Stochastic Gradient Descent
  - Decision Tree
  - Random Forest
  - Support Vector Machine (linear / nonlinear)
  - Total Ensemble
    - Voting
    - Bagging
    - Boosting (AdaBoost, Gradient Boosting)
- Make a **stacking classifier** better than others

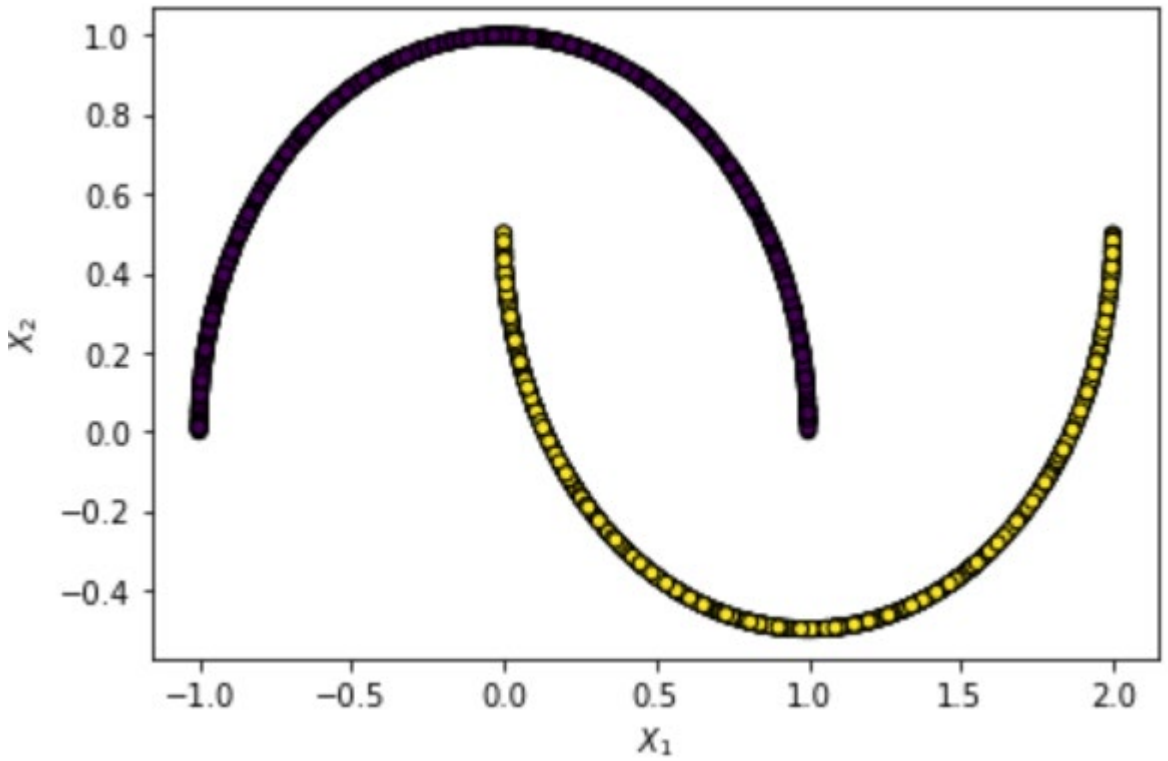
과제 #2 (설명)

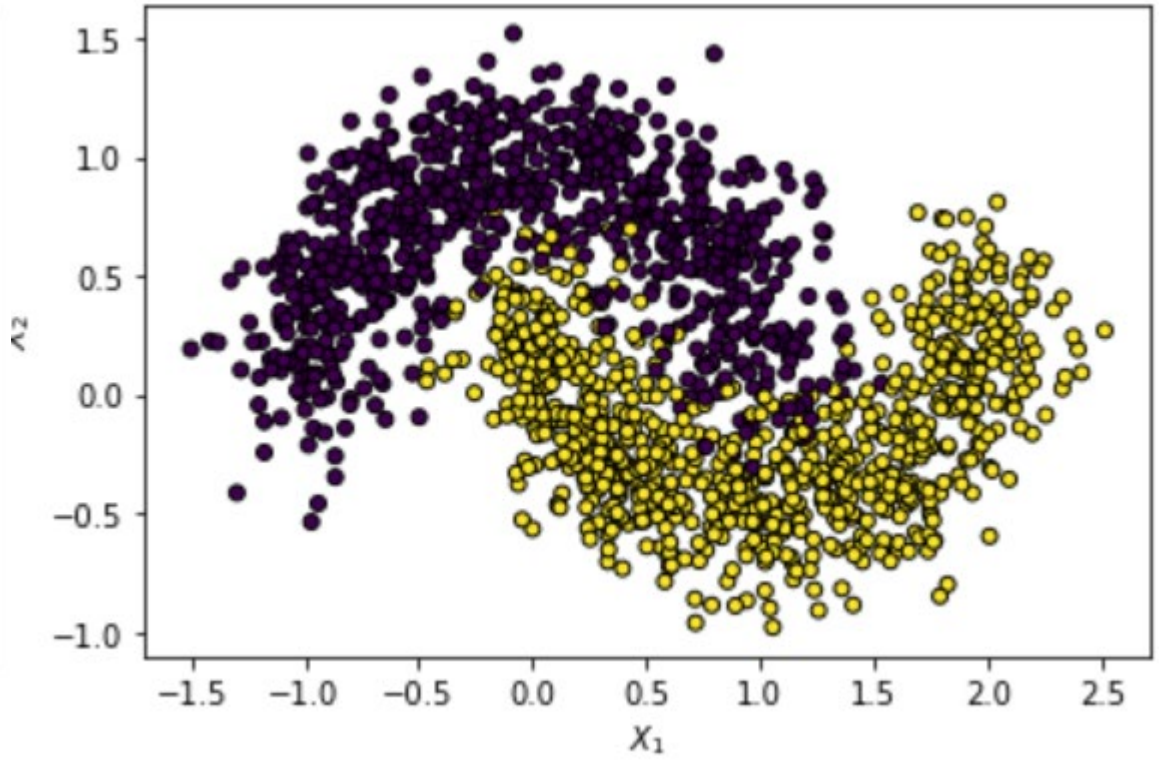
```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

X, y = make_moons(n_samples=3000, noise=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)
```

- What is '**make_moons**'?
  - **Left**: w/o noise,

Right: w/ noise





과제 #2 (설명)

- Code
  - get_models()

```python
def get_models(r_state=42, n_est=50, lr=0.1):
    models = dict()
    models['log'] =  LogisticRegression(solver="lbfgs", random_state=42)
    models['sgd']= SGDClassifier(loss="hinge", learning_rate="constant",
                                 eta0=0.001, max_iter=10000, tol=1e-3,
                                 random_state=r_state)
    models['dt'] = DecisionTreeClassifier(random_state=r_state)
    models['rf'] = RandomForestClassifier(
                                          random_state=r_state)
    models['lsvm'] = SVC(kernel="linear", random_state=r_state)
    models['polsvm'] = Pipeline([
        ("poly_features", PolynomialFeatures(degree=10)),
        ("scaler", StandardScaler()),
        ("svm_clf", LinearSVC(C=10, loss="hinge", random_state=42))
    ])
    models['vote'] = VotingClassifier(
        estimators=[('log', models['log']),
                    ('sgd', models['sgd']),
                    ('rf', models['rf']),
                    ('polsvm', models['polsvm'])],
        voting='hard')
    models['bag'] = BaggingClassifier(
        DecisionTreeClassifier(random_state=r_state), n_estimators=n_est,
        bootstrap=True, random_state=r_state)
    models['adab'] = AdaBoostClassifier(
        DecisionTreeClassifier(random_state=r_state), n_estimators=n_est,
        algorithm="SAMME.R", learning_rate=lr, random_state=r_state)
    models['grab'] = GradientBoostingClassifier(
        random_state=r_state)

    return models
```

과제 #2 (설명)

- Code
  - evaluate_model()

In [21]:

```
from sklearn.metrics import accuracy_score
import time

def evaluate_model(model, X_train, X_test, y_train, y_test):
    start = time.time()

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    elaptime = time.time() - start
    acc = accuracy_score(y_test, y_pred)
    return acc, elaptime

models = get_models()

results, names, times = list(), list(), list()
for name, model in models.items():
    acc, elaptime = evaluate_model(model, X_train, X_test,
                                   y_train, y_test)
    results.append(acc)
    times.append(elaptime)
    names.append(name)
    print('%s₩t %.4f (time: %.3f)' % (name, acc, elaptime))
```

과제 #2 (설명)

- Result

```
for name, model in models.items():
    acc, elaptime = evaluate_model(model, X_train, X_test,
                                   y_train, y_test)
    results.append(acc)
    times.append(elaptime)
    names.append(name)
    print('%s₩t %.4f (time: %.3f)' % (name, acc, elaptime))
```

```
log     0.8620 (time: 0.006)
sgd     0.8607 (time: 0.002)
dt      0.9620 (time: 0.002)
rf      0.9700 (time: 0.194)
lsvm    0.8593 (time: 0.020)
polsvm  0.9687 (time: 0.023)
```

```
C:₩Users₩seasik_corner₩Anaconda3₩envs₩mlbasic₩lib₩site-packages₩
verge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
C:₩Users₩seasik_corner₩Anaconda3₩envs₩mlbasic₩lib₩site-packages₩
verge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
vote    0.9193 (time: 0.230)
bag     0.9660 (time: 0.126)
adab    0.9627 (time: 0.004)
grab    0.9673 (time: 0.132)
```

과제 #2 (설명)

- Stacking

```
stack    0.9667 (time: 1.900)
```

```python
from sklearn.ensemble import StackingClassifier

def get_stacking(models, nfold=5):
    layer0 = list()
    layer0.append(('sgd', models['sgd']))
    layer0.append(('rf', models['rf']))
    layer0.append(('polsvm', models['polsvm']))
    layer0.append(('adab', models['adab']))
    layer0.append(('grab', models['grab']))
    layer1 = LogisticRegression()

    model = StackingClassifier(estimators=layer0,
                               final_estimator=layer1,
                               cv=nfold)
    return model


modelstack = dict()
modelstack['stack'] = get_stacking(models)

results, names, times = list(), list(), list()
for name, model in modelstack.items():
    acc, elaptime = evaluate_model(model, X_train, X_test,
                                   y_train, y_test)
    results.append(acc)
    times.append(elaptime)
    names.append(name)
    print('%s₩t %.4f (time: %.3f)' % (name, acc, elaptime))
```

과제 #2: moons classification

- 1) Make a **stacking classifier** better than others
  - Capture the code of **get_stacking**()
  - Compare its result with others
- 2) Report which classifier has the highest complexity
  - Measure all times of classifiers and find which is the worst case
  - Describe the reason or guess on the worst case
- **Make one code** that can show the entire modification and results mentioned above.
  - Thus, in your submission via LMS, there must be two files:
  - 1) Report (in MS or PDF format only)
  - 2) Python code or colab (for colab only: not as a link, but a complete notebook file)

과제 #2 (주의)

- NOTE:
  - Do not change other parameters
    - Except that of **LinearSVC**

```
15 def get_models(r_state=42, n_est=50, lr=0.1):
16     models = dict()
17     models['log'] =  LogisticRegression(solver="lbfgs", random_state=42)
18     models['sgd']= SGDClassifier(loss="hinge", learning_rate="constant",
19                                  eta0=0.001, max_iter=10000, tol=1e-3,
20                                  random_state=r_state)
21     models['dt'] = DecisionTreeClassifier(random_state=r_state)
22     models['rf'] = RandomForestClassifier(
23                                          random_state=r_state)
24     models['lsvm'] = SVC(kernel="linear", random_state=r_state)
25     models['polsvm'] = Pipeline([
26         ("poly_features", PolynomialFeatures(degree=10)),
27         ("scaler", StandardScaler()),
28         ("svm_clf", LinearSVC(C=10, loss="hinge", random_state=42))
29     ])
30     models['vote'] = VotingClassifier(
31         estimators=[('log', models['log']),
32                     ('sgd', models['sgd']),
33                     ('rf', models['rf']),
34                     ('polsvm', models['polsvm'])],
35         voting='hard')
36     models['bag'] = BaggingClassifier(
37         DecisionTreeClassifier(random_state=r_state), n_estimators=n_est,
38         bootstrap=True, random_state=r_state)
39     models['adab'] = AdaBoostClassifier(
40         DecisionTreeClassifier(random_state=r_state), n_estimators=n_est,
41         algorithm="SAMME.R", learning_rate=lr, random_state=r_state)
42     models['grab'] = GradientBoostingClassifier(
43         random_state=r_state)
44
45     return models
```

과제 #2 (주의)

- NOTE:
  - Do not change other parameters
    - Except that of **LinearSVC**
    - Except 'VotingClassifier'
      - Though not recommended

```
15 def get_models(r_state=42, n_est=50, lr=0.1):
16     models = dict()
17     models['log'] =  LogisticRegression(solver="lbfgs", random_state=42)
18     models['sgd']= SGDClassifier(loss="hinge", learning_rate="constant",
19                                  eta0=0.001, max_iter=10000, tol=1e-3,
20                                  random_state=r_state)
21     models['dt'] = DecisionTreeClassifier(random_state=r_state)
22     models['rf'] = RandomForestClassifier(
23                                          random_state=r_state)
24     models['lsvm'] = SVC(kernel="linear", random_state=r_state)
25     models['polsvm'] = Pipeline([
26         ("poly_features", PolynomialFeatures(degree=10)),
27         ("scaler", StandardScaler()),
28         ("svm_clf", LinearSVC(C=10, loss="hinge", random_state=42))
29     ])
30     models['vote'] = VotingClassifier(
31         estimators=[('log', models['log']),
32                     ('sgd', models['sgd']),
33                     ('rf', models['rf']),
34                     ('polsvm', models['polsvm'])],
35         voting='hard')
36     models['bag'] = BaggingClassifier(
37         DecisionTreeClassifier(random_state=r_state), n_estimators=n_est,
38         bootstrap=True, random_state=r_state)
39     models['adab'] = AdaBoostClassifier(
40         DecisionTreeClassifier(random_state=r_state), n_estimators=n_est,
41         algorithm="SAMME.R", learning_rate=lr, random_state=r_state)
42     models['grab'] = GradientBoostingClassifier(
43         random_state=r_state)
44
45     return models
```

과제 #2 (주의)

- NOTE:
  - Do not change other parameters
    - But add or remove **models**

```
from sklearn.ensemble import StackingClassifier

def get_stacking(models, nfold=5):
    layer0 = list()
    layer0.append(('sgd', models['sgd']))
    layer0.append(('rf', models['rf']))
    layer0.append(('polsvm', models['polsvm']))
    layer0.append(('adab', models['adab']))
    layer0.append(('grab', models['grab']))
    layer1 = LogisticRegression()

    model = StackingClassifier(estimators=layer0,
                               final_estimator=layer1,
                               cv=nfold)
    return model


modelstack = dict()
modelstack['stack'] = get_stacking(models)

results, names, times = list(), list(), list()
for name, model in modelstack.items():
    acc, elaptime = evaluate_model(model, X_train, X_test,
                                   y_train, y_test)
    results.append(acc)
    times.append(elaptime)
    names.append(name)
    print('%s₩t %.4f (time: %.3f)' % (name, acc, elaptime))
```

과제 #2 (주의)

- 주의사항과 요구사항을 충족할 경우에만 본 과제 만점처리
  - 주의사항을 지키지 않을 경우 등 충족불가한 요소가 있을 경우, 감점
- 기한 : 3주
- 제출방법 : LMS 內 '과제' 메뉴에서 첨부파일로서 제출
- 기타
  - 예제 소스코드/노트북파일은 LMS內 '과제' 메뉴에서 다운받을 것
  - "IML-Assign2_classifiers.py" 등