분류 알고리즘 - 2

Kyungsik Han

Decision Tree

CODING

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
dt.fit(X_train_std, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

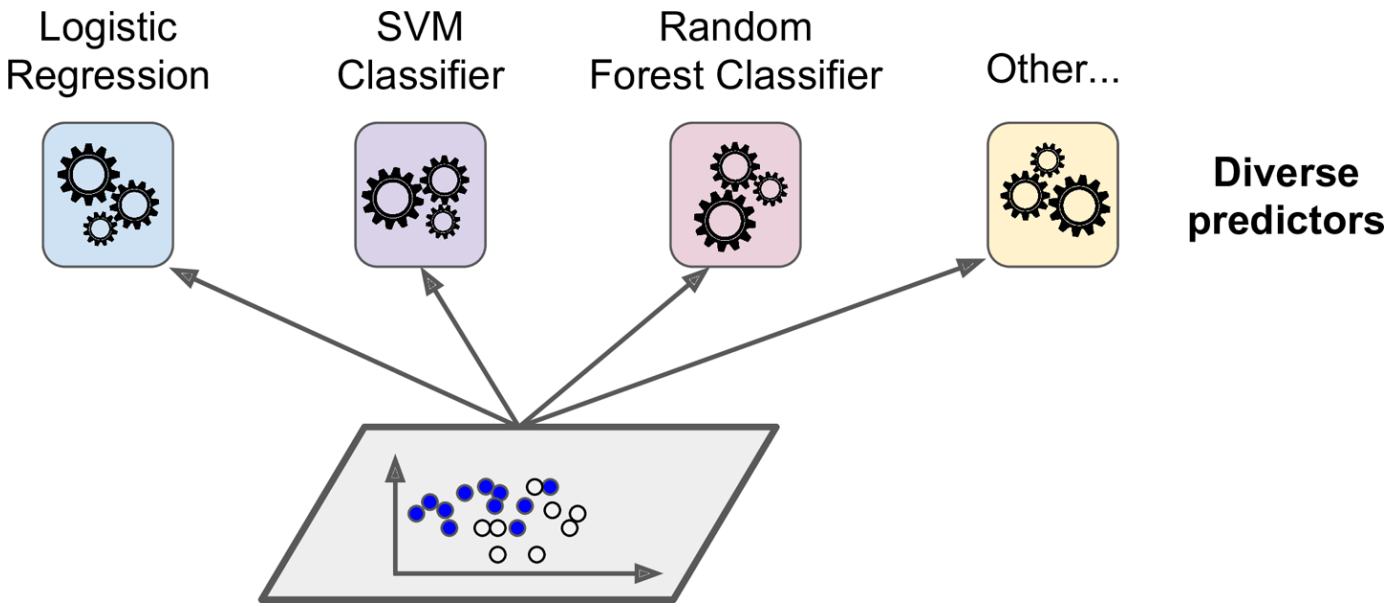
```
# X 테스트 데이터의 예측값 구하기
y_pred = dt.predict(X_test_std)

# 정답값과 예측값의 비교
accuracy_score(y_test, y_pred)
```

```
0.96666666666666667
```

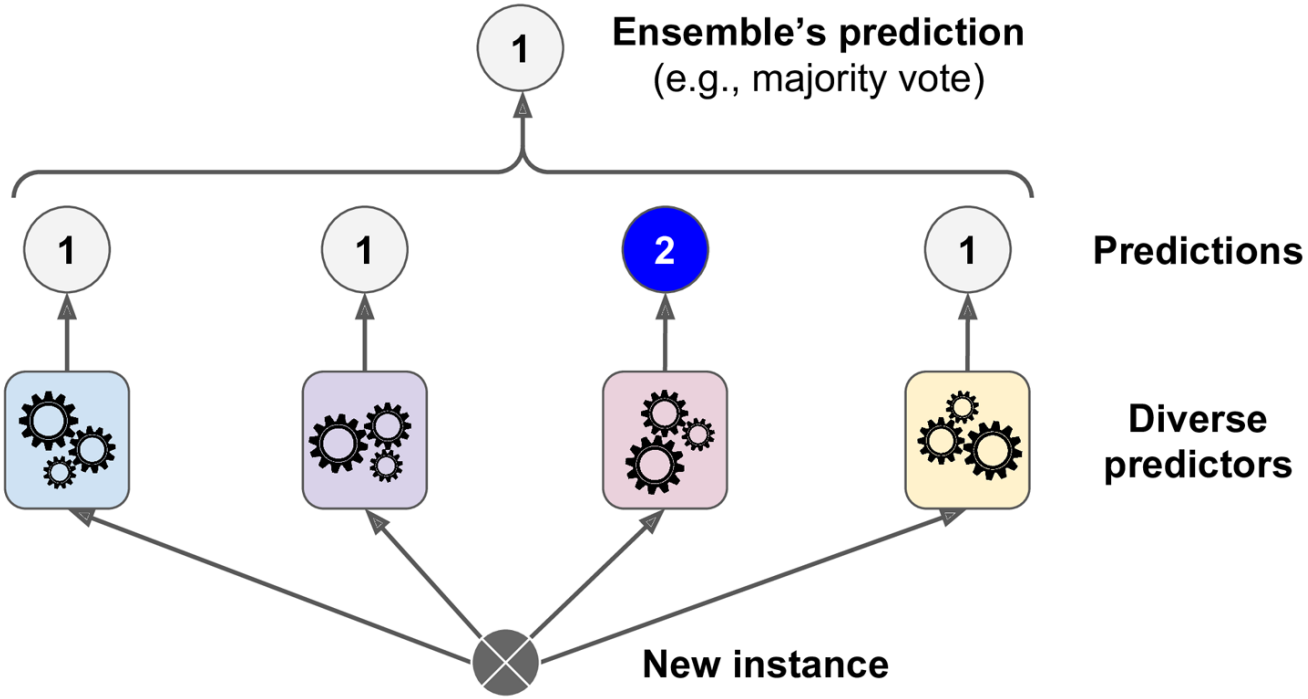
Random Forest

Random Forest



다양한 분류기가 있다고 가정

Random Forest (Ensemble)



직접 투표 분류기:
각 분류기의 예측을 모아서 가장 많이 선택된 클래스를 예측하는 것

(성능) 다수결 투표 분류기 > 개별 분류기
이는 큰 수의 법칙(law of large number) 때문.

Random Forest (Ensemble)

CODING

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

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

log_clf = LogisticRegression(solver='liblinear', random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=10, random_state=42)
svm_clf = SVC(gamma='auto', random_state=42)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```

```
from sklearn.metrics import accuracy_score

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.864
RandomForestClassifier 0.872
SVC 0.888
VotingClassifier 0.896
```

Ensemble 방식이 높은 성능을 보여줌

Random Forest (Ensemble)

CODING ✖

```
log_clf = LogisticRegression(solver='liblinear', random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=10, random_state=42)
svm_clf = SVC(gamma='auto', probability=True, random_state=42)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='soft')
voting_clf.fit(X_train, y_train)
```

```
VotingClassifier(estimators=[('lr', LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=42, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)), ('rf', Rando...bf',
  max_iter=-1, probability=True, random_state=42, shrinking=True,
  tol=0.001, verbose=False))],
       flatten_transform=None, n_jobs=None, voting='soft', weights=None)
```

```
from sklearn.metrics import accuracy_score

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

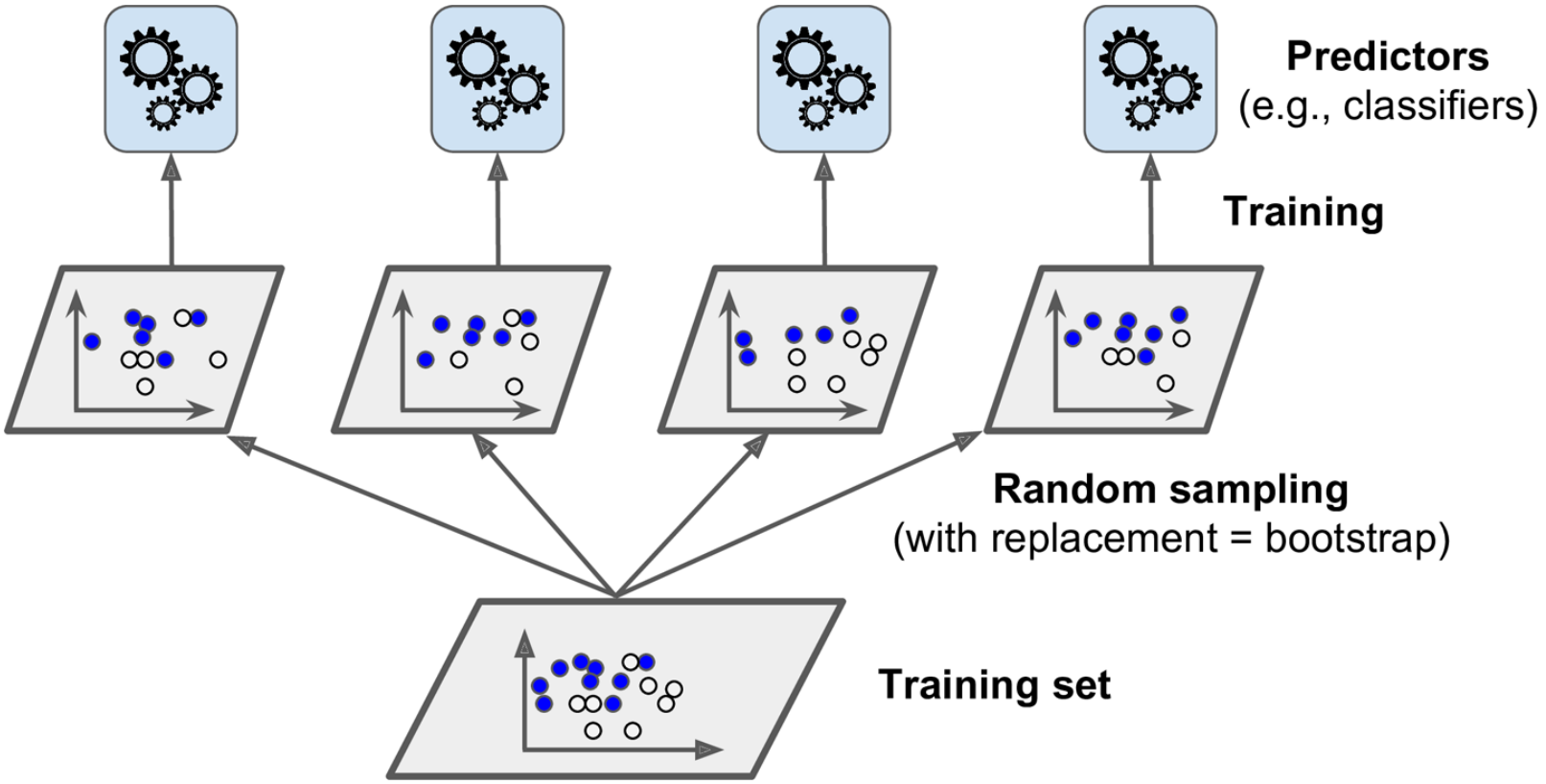
```
LogisticRegression 0.864
RandomForestClassifier 0.872
SVC 0.888
VotingClassifier 0.912
```

Soft Ensemble 방식이 Hard 보다 높은 성능을 보여줌

Begging or Pasting(Bootstrapping)

- 각기 다른 훈련 알고리즘을 사용할 수 있지만 (이전 예제처럼),
- 같은 알고리즘을 사용하고 훈련 세트의 서브셋을 무작위로 구성하여 각기 다르게 학습 가능
  - 배깅(bagging): 훈련 세트에서 중복을 허용하여 샘플링
  - 페이스팅(pasting) or 부트스트래핑(bootstrapping): 중복을 허용하지 않고 샘플링

Begging or Pasting(Bootstrapping)



Begging or Pasting(Bootstrapping)

CODING

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

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

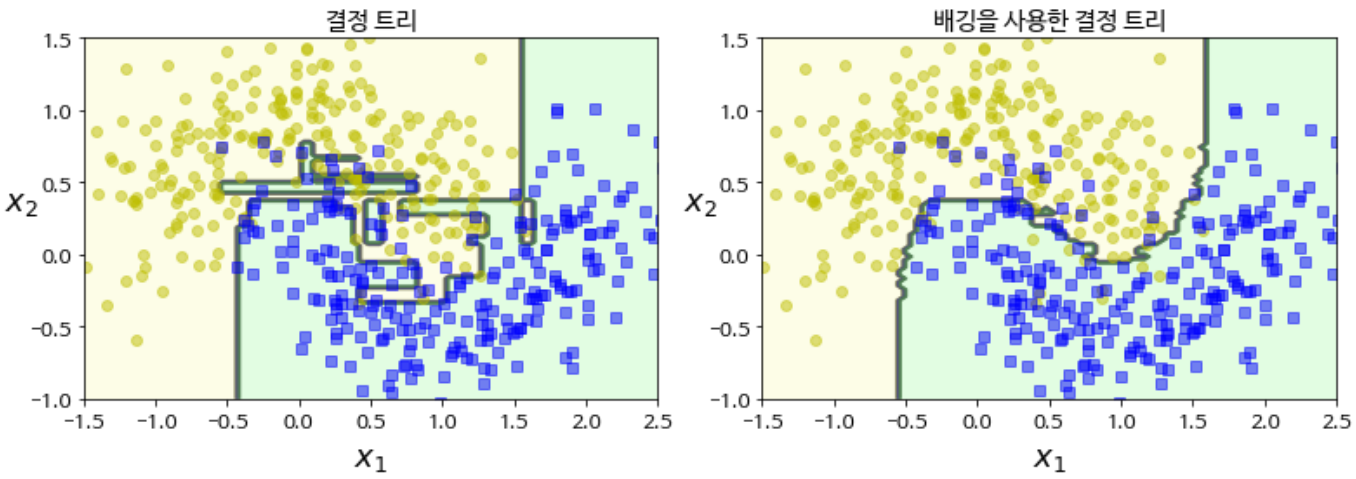
"bootstrap = True or False" 에 따라서
bootstrapping 인지 begging 인지 결정

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.904
```

```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
y_pred_tree = tree_clf.predict(X_test)
print(accuracy_score(y_test, y_pred_tree))
```

```
0.856
```



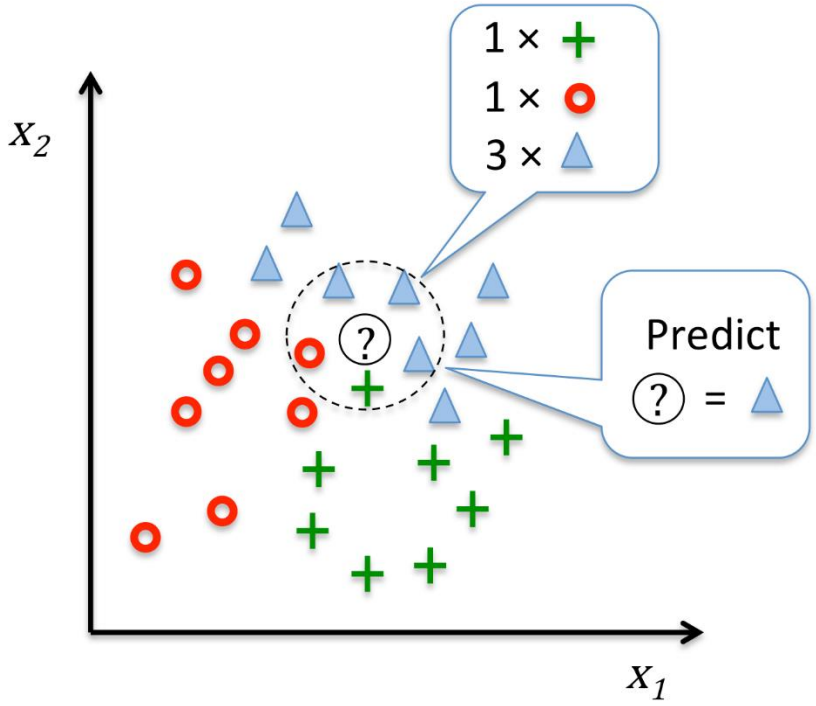
kNN (k-Nearest Neighbor)

kNN

- kNN은 지금까지 논의했던 지도학습 알고리즘과 근본적으로 다름
- kNN은 Lazy Learner 기반 알고리즘
  - 훈련 데이터로부터 식별 함수를 학습하는 대신, 데이터를 기억하기 때문

kNN

- k 에 해당하는 숫자와 거리 메트릭을 선택함
- 분류하고자 하는 샘플에 대한 k개의 근접한 이웃을 찾음
- 다수결 투표 방식으로 분류 레이블을 할당함



kNN

- k를 알맞게 선택하는 것이 아주 중요
- 데이터 features 에 대한 적당한 거리 metric을 선택해야 함
- 많이 쓰는 거리 함수
  - Minkowski distance
  - Euclidean distance
  - Manhattan distance

kNN

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski')
knn.fit(X_train_std, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

```
# X 테스트 데이터의 예측값 구하기
y_pred = knn.predict(X_test_std)

# 정답값과 예측값의 비교
accuracy_score(y_test, y_pred)
```

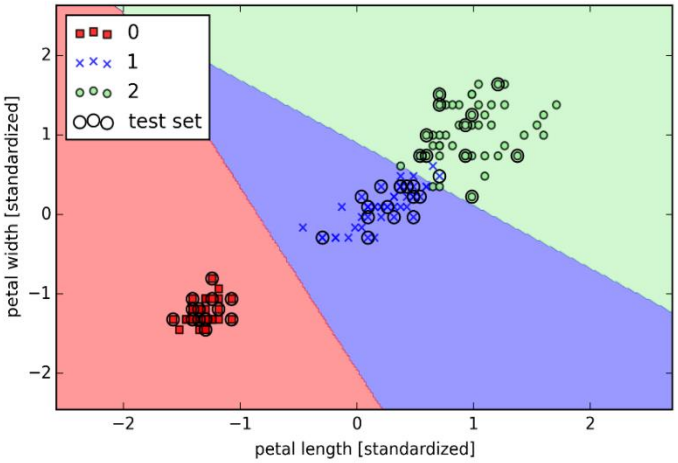
```
1.0
```

kNN 평가

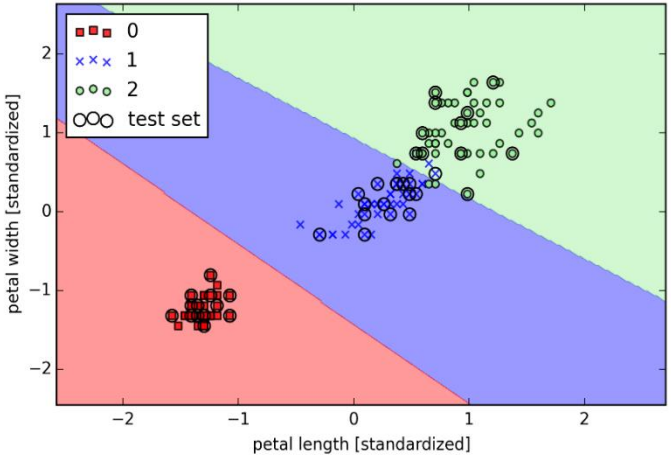
CODING

- 알고리즘이 간단하여 구현하기 쉬움
- 수치 기반 데이터 분류 작업에서 성능이 좋음
- 학습 데이터의 양이 많으면 분류 속도가 느려짐
- 차원(벡터)의 크기가 크면 계산량이 많아짐

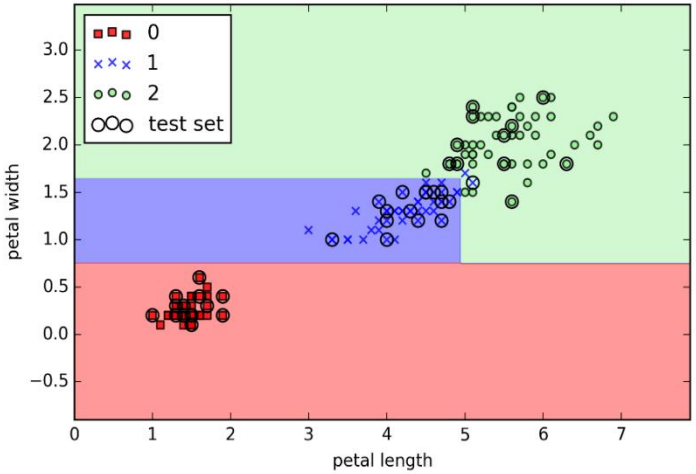
Differences Among Classifiers



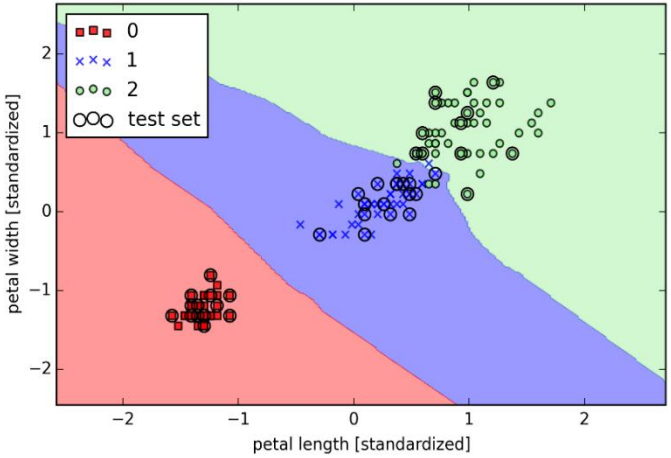
Logistic Regression



Support Vector Machine

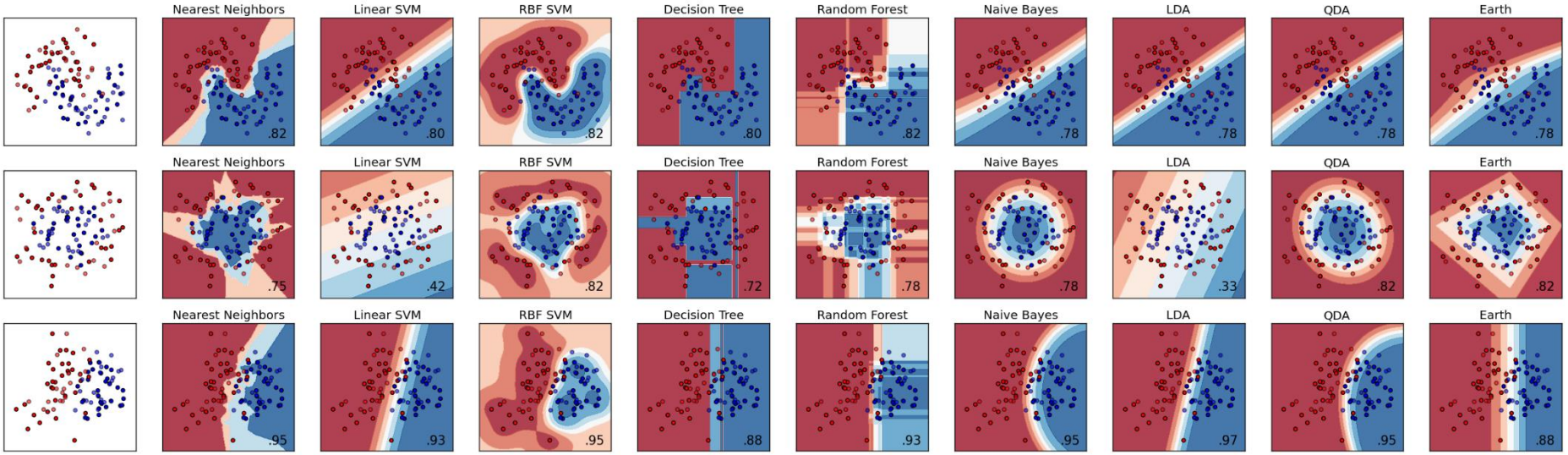


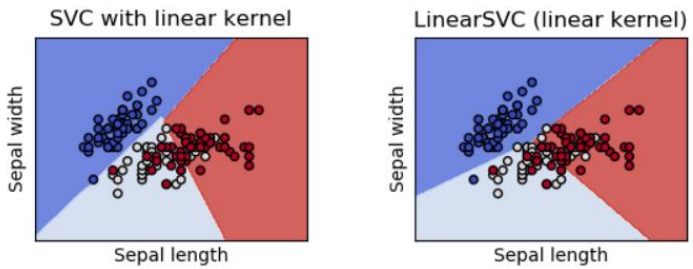
Decision Tree

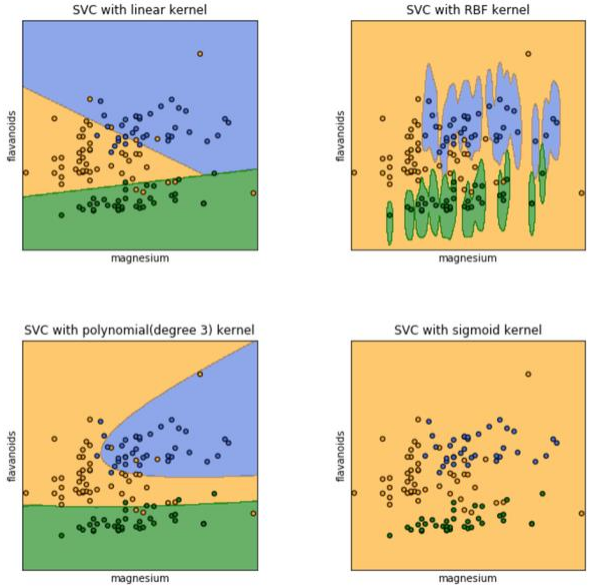


kNN

Look at How Each Algorithm Works







다음 영상에서 배울 내용

CODING

- 지도학습 모델 평가
  - 데이터셋
  - 평가지표 및 측정(이진분류)
  - 과적합(overfitting)
  - 오차행렬
  - 분류 평가지표
    - Accuracy, Precision, Recall, F1-score, ROC, AUC
  - 회귀 평가지표
    - R^2

수고하셨습니다