人工智慧與數位學習 期中報告

◆ 作業大綱

09/13	請畫出各種 degree 的多項式回歸和訓練樣本數 N 的關係
09/20	依照投影片資料，以 GD 訓練出線性回歸
09/27	對登記的資料集 進行決策樹與隨機森林的分析
10/04	隨機森林 Random Forest 或 Adaboost，對參數進行分析
10/11	KNN, CART, LR 比較 k 值
10/25	SVR, SVC

◆ 各種 degree 的多項式回歸和訓練樣本數 N 的關係

(1) 分別定義兩種訓練樣本數的 x 和 y

訓練樣本 50 筆	訓練樣本 100 筆
x1 = np.linspace(0,2*np.pi,50) y1 = np.sin(x1)+np.random.randn(len(x1))/5.0	x2 = np.linspace(0,2*np.pi,100) y2 = np.sin(x2)+np.random.randn(len(x2))/5.0

(2) 分別計算迴歸係數和截距

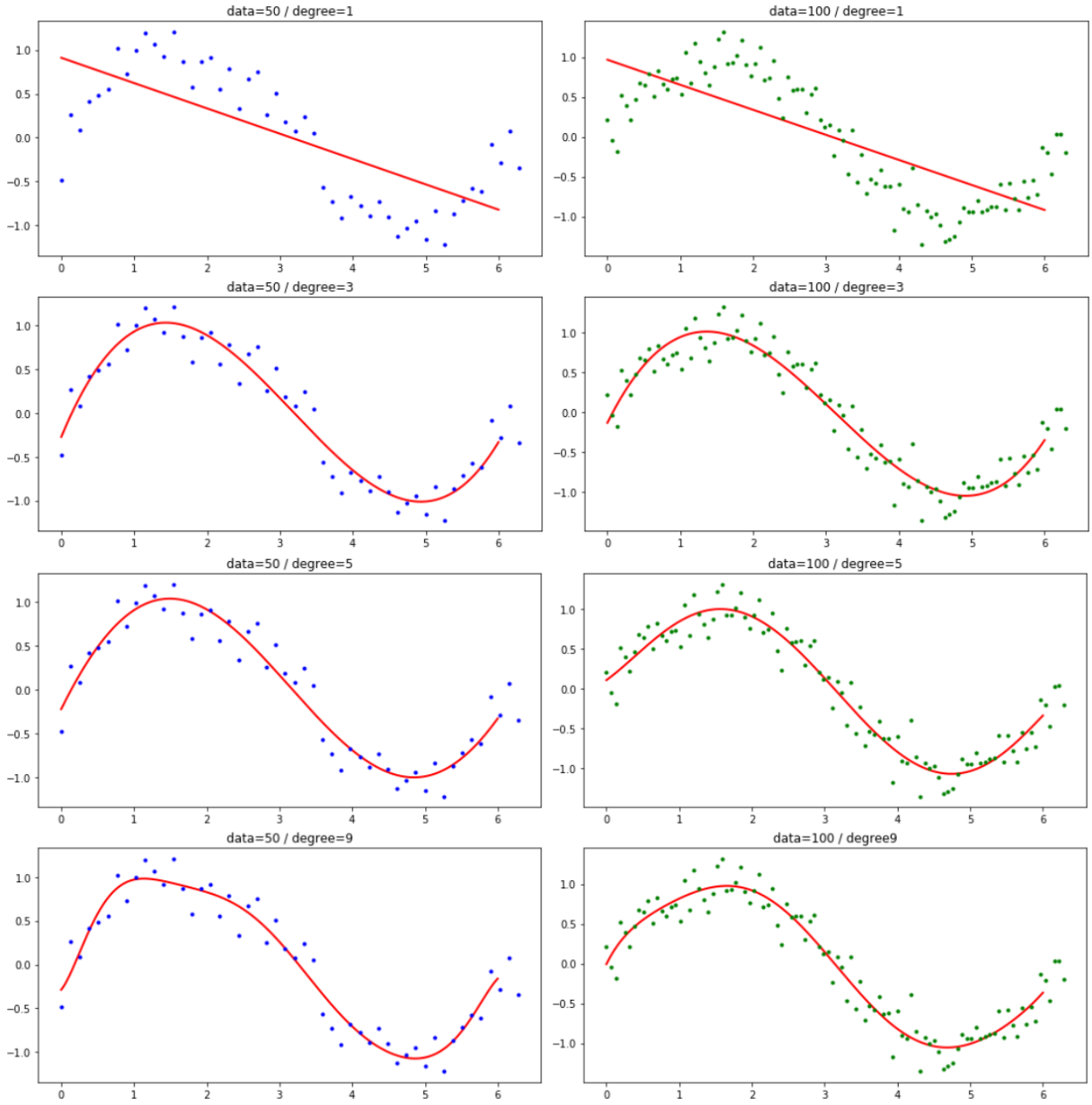
訓練樣本 50 筆	訓練樣本 100 筆
slr = LinearRegression() x1 = x1.reshape(-1,1) slr.fit(x1,y1) print("樣本數 50 的迴歸係數：",slr.coef_) print("樣本數 50 的截距：",slr.intercept_)	slr = LinearRegression() x2 = x2.reshape(-1,1) slr.fit(x2,y2) print("樣本數 100 的迴歸係數：",slr2.coef_) print("樣本數 100 的截距：",slr2.intercept_)
樣本數 50 的迴歸係數： [-0.28863508] 樣本數 50 的截距： 0.9112656660524244	樣本數 100 的迴歸係數： [-0.31414697] 樣本數 100 的截距： 0.9657530018885288

(3) 使用 subplots 來視覺化呈現

以訓練樣本 50 筆，degree = 1 的作為例子，其餘的以此類推。

```
5  #訓練樣本50筆，degree = 1
6  poly_features_1 = PolynomialFeatures(degree=1,include_bias=False)
7  X_poly_1 = poly_features_1.fit_transform(x1)
8  lin_reg_1 = LinearRegression()
9  lin_reg_1.fit(X_poly_1,y1)
10 #print(lin_reg_1.intercept_,lin_reg_1.coef_)
11 X_plot = np.linspace(0,6,1000).reshape(-1,1)
12 X_plot_poly = poly_features_1.fit_transform(X_plot)
13 y_plot = np.dot(X_plot_poly,lin_reg_1.coef_.T)+lin_reg_1.intercept_
14 plt.subplot(4,2,1)
15 plt.plot(X_plot,y_plot, 'r-', lw=2)
16 plt.plot(x1,y1,'b.')
17 plt.title("data=50 / degree=1")
```

(4) 畫出函數結果



(5) 結果分析

由上圖發現，不管訓練樣本數為 50 或 100，degree = 3 時，訓練出來的多項式回歸線有最佳的擬合；degree＝9 時，函數圖會穿過一部分的樣本點，幾乎所有的訓練樣本都落在了擬合的曲線上，訓練誤差趨近於 0，可以說是近乎完美的模型了。但是，這樣的曲線與最一開始數據源差異非常大。訓練誤差非常小，但是新數據點的測試誤差非常大的情況，就叫形成過擬合(overfitting)，表示模型過於複雜，使得模型的泛化能力下降。

(6) 解決方法

- 降低模型複雜度(縮小 degree)
- 降維(減少特徵的數量)
- 增加訓練樣本
- 添加正則化項

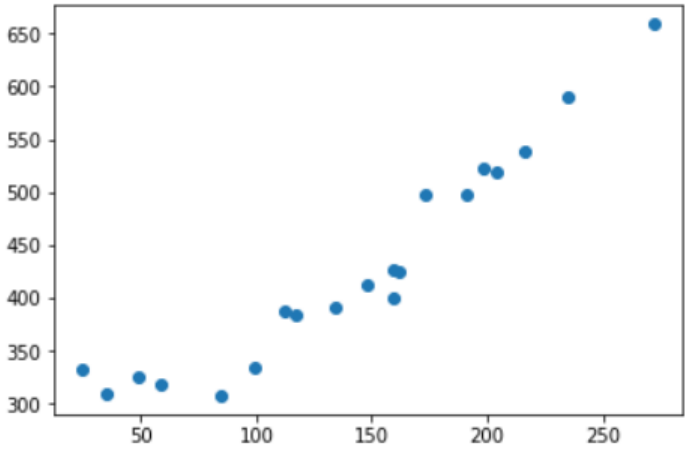
◆ 梯度下降法(Gradient Descent，GD)、線性回歸(Linear Regression，LR)

一種不斷去更新參數找解的方法，希望用此方法找到函數的局部最小值，因為梯度的方向是走向局部最大的方向，所以在梯度下降法中是往梯度的反方向走。

(1) 載入資料集

	0	1
0	25	332
1	35	310
2	49	325
3	59	319
4	85	308
5	99	334
6	112	387
7	117	385
8	134	392
9	148	413
10	159	400
11	159	427
12	162	425
13	173	498
14	191	498
15	198	522
16	204	519
17	216	539
18	235	591
19	272	659

將左邊的資料用散佈圖表示



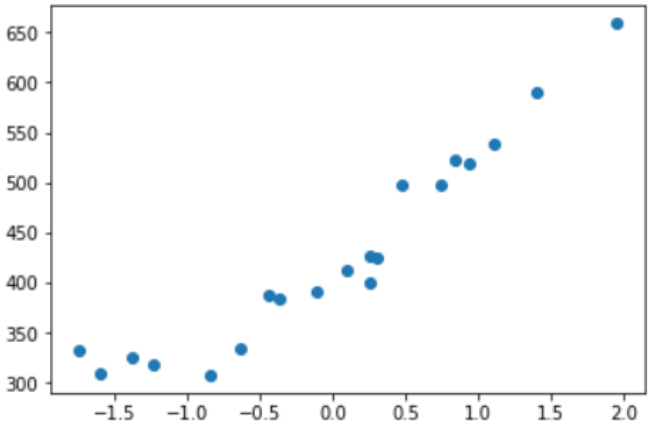
(2) 標準化(standardize)

```
#標準化
mu = train_x.mean()
sigma = train_x.std()
def standardize(x):
    return (x-mu)/sigma

train_z = standardize(train_x)
print(train_z)

plt.scatter(train_z,train_y)
plt.show()
```

把 train_x 標準化後的定義為 train_z 並用散佈圖表示



(3) 定義參數

隨機定義 theta0 和 theta1

定義函數

學習速率設為 1e-3(0.0001)

```
theta0 = np.random.rand()
theta1 = np.random.rand()

def f(x):
    return theta0 + theta1 * x

def E(x,y):
    return 0.5 * np.sum((y - f(x)) ** 2)

ETA = 1e-3
diff = 1
count = 0
```

(4) 計算迭代的次數

定義 error

定義 diff

```
error = E(train_z,train_y)
while diff > 1e-2:

    top_theta0 = theta0 - ETA * np.sum((f(train_z) - train_y))
    top_theta1 = theta1 - ETA * np.sum((f(train_z) - train_y) * train_z)

    theta0 = top_theta0
    theta1 = top_theta1

    current_error = E(train_z, train_y)
    diff = error - current_error
    error = current_error

    count += 1
    log = '迭代第{}次: theta0 = {:.3f}, theta1 = {:.3f}, 總分 = {:.4f}'
    print(log.format(count, theta0, theta1, diff))

print('迭代次數: %d' % count)
```

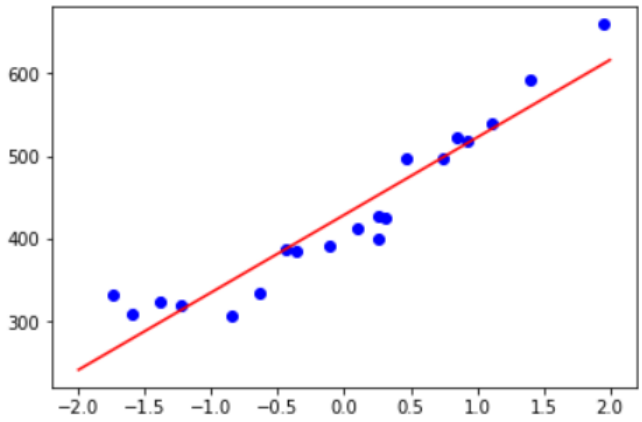
```
迭代第1次: theta0 = 8.904, theta1 = 1.919, 總分 = 76276.4077
迭代第2次: theta0 = 17.309, theta1 = 3.751, 總分 = 73255.8620
迭代第3次: theta0 = 25.546, theta1 = 5.545, 總分 = 70354.9299
迭代第4次: theta0 = 33.618, theta1 = 7.304, 總分 = 67568.8746
迭代第5次: theta0 = 41.529, theta1 = 9.027, 總分 = 64893.1472
迭代第6次: theta0 = 49.281, theta1 = 10.716, 總分 = 62323.3786
迭代第7次: theta0 = 56.879, theta1 = 12.372, 總分 = 59855.3728
迭代第8次: theta0 = 64.324, theta1 = 13.994, 總分 = 57485.1000
迭代第9次: theta0 = 71.621, theta1 = 15.583, 總分 = 55208.6901
迭代第10次: theta0 = 78.771, theta1 = 17.141, 總分 = 53022.4259
                              ⋮
迭代第386次: theta0 = 428.974, theta1 = 93.440, 總分 = 0.0134
迭代第387次: theta0 = 428.978, theta1 = 93.441, 總分 = 0.0129
迭代第388次: theta0 = 428.981, theta1 = 93.442, 總分 = 0.0123
迭代第389次: theta0 = 428.984, theta1 = 93.443, 總分 = 0.0119
迭代第390次: theta0 = 428.988, theta1 = 93.443, 總分 = 0.0114
迭代第391次: theta0 = 428.991, theta1 = 93.444, 總分 = 0.0109
迭代第392次: theta0 = 428.994, theta1 = 93.445, 總分 = 0.0105
迭代第393次: theta0 = 428.997, theta1 = 93.445, 總分 = 0.0101
迭代第394次: theta0 = 429.000, theta1 = 93.446, 總分 = 0.0097
迭代次數: 394
```

(5) 結果分析

利用每次迭代的 theta0 和 theta1 來訓練迴歸，下圖為訓練後的迴歸線。

```
X = range(-2,3)
Y = [(theta1*i+theta0) for i in X]

plt.scatter(train_z,train_y,color='blue')
plt.plot(X,Y,color='red')
plt.show()
```



◆ 決策樹(Decision Tree)

在學習的時候就需要考慮特徵節點的選取順序，常用的度量方式包括信息熵(Information Gain)和基尼不純性(Gini Impurity)，性能分析會使用準確率(Accuracy)、召回率(Recall)、精確率(Percision)以及 F1 指標。

(1) 選擇資料集

以 electricity.csv 這個 dataset 作為例子

```
#匯入資料
data = pd.read_csv('data/electricity.csv')
```

(2) 資料預處理

2.1 資料集欄位資料進行分類

把原資料集 electricity_consumption 依據使用的電量進行分類，分成 7 類，把分類的結果在資料集中新增一個欄位 electricity_consumption_category，如下圖所示。

```
# 針對用電量設定類別
def get_consumption_category(wt):
    if wt < 200:
        return 1
    elif 200 < wt < 400:
        return 2
    elif 400 < wt < 600:
        return 3
    elif 600 < wt < 800:
        return 4
    elif 800 < wt < 1000:
        return 5
    elif 1000 < wt < 1200:
        return 6
    else:
        return 7

data["electricity_consumption_category"] = data["electricity_consumption"].map(get_consumption_category)
data
```

	ID	datetime	temperature	var1	pressure	windspeed	var2	electricity_consumption	electricity_consumption_category
0	0	2013-07-01 00:00:00	-11.4	-17.1	1003.0	571.910	A	216.0	2
1	1	2013-07-01 01:00:00	-12.1	-19.3	996.0	575.040	A	210.0	2
2	2	2013-07-01 02:00:00	-12.9	-20.0	1000.0	578.435	A	225.0	2
3	3	2013-07-01 03:00:00	-11.4	-17.1	995.0	582.580	A	216.0	2
4	4	2013-07-01 04:00:00	-11.4	-19.3	1005.0	586.600	A	222.0	2
5	5	2013-07-01 05:00:00	-10.7	-19.3	1013.0	2.790	A	216.0	2

2.2 只列出需要使用的欄位

```
#只列出指定的欄位
df = data[['temperature','pressure', 'windspeed', 'electricity_consumption_category']]
print(type(df))

#檢視前 5 筆資料
df.head()
```

	temperature	pressure	windspeed	electricity_consumption_category
0	-11.4	1003.0	571.910	2
1	-12.1	996.0	575.040	2
2	-12.9	1000.0	578.435	2
3	-11.4	995.0	582.580	2
4	-11.4	1005.0	586.600	2

2.3 定義 X 和 y (X 移除 electricity_consumption_category)

```
X=df.drop('electricity_consumption_category', axis=1)
y=df['electricity_consumption_category']
```

2.4 分割訓練集和測試集

將資料的 70%拿出來 train，剩下的 30%用來檢測 train 的好壞

```
from sklearn.cross_validation import train_test_split
```

```
#分割訓練和測試集
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.3, random_state=101)
```

(3) 執行決策樹

3.1 載入決策樹函式 DecisionTreeClassifier

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

```
DT=DecisionTreeClassifier()
```

```
DT.fit(X_train, y_train)
```

3.2 使用 y_train 來作預測

```
y_pred_DT=DT.predict(X_test)
```

3.3 列出隨機森林相關分析數據

分別有召回率(Recall)、精確率(Percision)以及 F1 指標

```
#列出隨機森林相關分析數據
from sklearn.metrics import classification_report, accuracy_score
print("Decision Tree Classifier")
print(classification_report(y_test, y_pred_DT))
```

```
Decision Tree Classifier
             precision    recall  f1-score   support

          1       0.19      0.20      0.20       809
          2       0.77      0.75      0.76      6008
          3       0.18      0.19      0.18       929
          4       0.04      0.04      0.04       181
          5       0.00      0.00      0.00        13
          6       0.00      0.00      0.00         2
          7       0.00      0.00      0.00         7

avg / total       0.62      0.61      0.62      7949
```

計算決策樹的準確度(Accuracy)

```
#決策樹準確度
print(accuracy_score(y_test, y_pred_DT))
```

```
0.6145427097748144
```

◆ 隨機森林(Random Forest，RF)

利用相同的訓練數據來建立多個分類模型，以少數服從多數的原則做出最終的分類決策，而隨機森林分類器就是這一類型的代表。在相同的訓練數據上同時建立多棵決策樹，一棵標準的決策樹根據每維特徵來對預測結果的影響程度進行排序，進而決定不同特徵從上至下構建分裂節點的順序，但如果都這樣，那麼每一棵樹都類似，那麼就丟失了多樣性，而隨機森林分類器在構建過程中放棄了這一固定的排序算法，轉而隨機選取特徵。

(1) 選擇資料集

與前一節相同

(2) 資料預處理

與前一節相同

(3) 執行隨機森林

3.1 設定隨機種子

```
# 設定隨機種子
np.random.seed(0)
```

3.2 載入隨機森林函式 RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
```

```
RF=RandomForestClassifier(n_estimators=100, oob_score=True, random_state=42)
```

```
RF.fit(X_train, y_train)
```

3.3 使用 X_test 來作預測

```
y_pred_rf=np.around(RF.predict(X_test))
```

```
y_pred_rf
```

```
array([2, 2, 3, ..., 4, 2, 2], dtype=int64)
```

3.4 列出隨機森林相關分析數據

分別有召回率(Recall)、精確率(Percision)以及 F1 指標

```
#列出隨機森林相關分析數據
from sklearn.metrics import classification_report, accuracy_score
print("Random Forest Classifier")
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Classifier
             precision    recall  f1-score   support

          1       0.25      0.11      0.15       809
          2       0.77      0.91      0.83      6008
          3       0.21      0.10      0.13       929
          4       0.02      0.01      0.01       181
          5       0.00      0.00      0.00        13
          6       0.00      0.00      0.00         2
          7       0.00      0.00      0.00         7

avg / total       0.63      0.71      0.66      7949
```

計算決策樹的準確度(Accuracy)

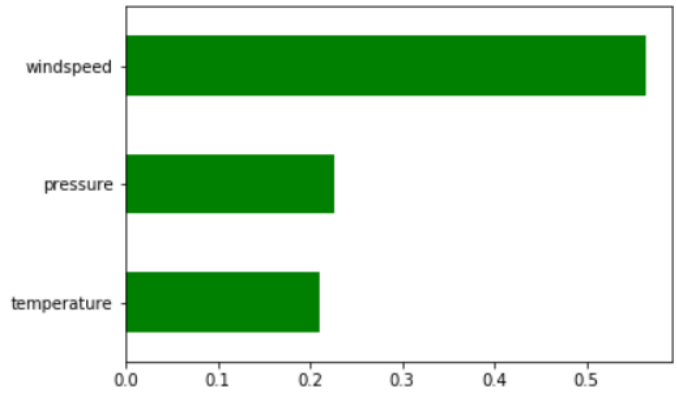
```
#隨機森林準確度
print(accuracy_score(y_test, y_pred_rf))
```

```
0.7075103786639829
```

(4) 分析特徵的重要程度

```
#特徵重要程度
feature_importance=pd.Series(RF.feature_importances_, index=X.columns)
feature_importance.sort_values().plot(kind='barh', color='g')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x21eb581b828>
```



資料集中 3 個特徵有 windspeed、pressure、temperature，由上圖可以發現，windspeed 對預測結果有較大的影響力。

- **比較隨機森林(Random Forest)和決策樹(Decision Tree)**

```
comparison=pd.DataFrame(np.array(['Random Forest Classifier', 'Decision Tree Classifier']))
```

```
comparison.columns=['Method']
```

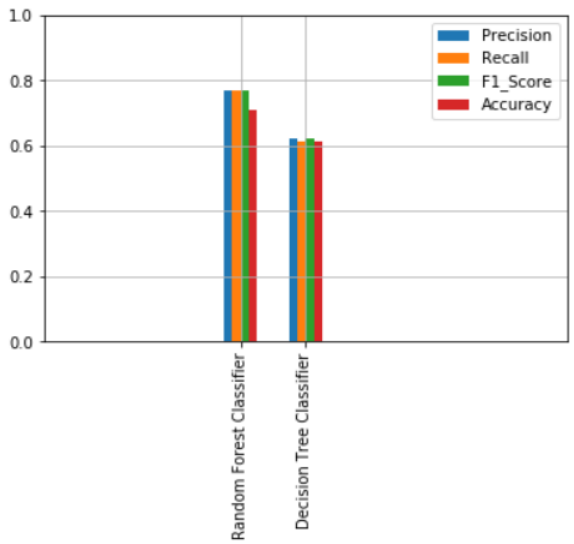
```
comparison['Precision']=[0.77, 0.62]
comparison['Recall']=[0.77, 0.61]
comparison['F1_Score']=[0.77, 0.62]
comparison['Accuracy']=[0.71, 0.61]
```

```
comparison
```

	Method	Precision	Recall	F1_Score	Accuracy
0	Random Forest Classifier	0.77	0.77	0.77	0.71
1	Decision Tree Classifier	0.62	0.61	0.62	0.61

```
comparison.plot(kind='bar')
plt.ylim(0,1)
plt.xlim(-3, 5)
plt.grid(True)
plt.xticks([0, 1], ['Random Forest Classifier', 'Decision Tree Classifier'])
```

```
([<matplotlib.axis.XTick at 0x21eb696e828>,
  <matplotlib.axis.XTick at 0x21eb696e128>],
 <a list of 2 Text xticklabel objects>)
```



以 electricity.csv 這個資料集來說，
可以發現 Random Forest 的性能比較好。

◆ AdaBoost

Adaboost 是一種反覆運算演算法，其核心思想是針對同一個訓練集訓練不同的分類器（弱分類器），然後把這些弱分類器集合起來，構成一個更強的最終分類器（強分類器）。Adaboost 演算法本身是通過改變資料分佈來實現的，它根據每次訓練集之中每個樣本的分類是否正確，以及上次的總體分類的準確率，來確定每個樣本的權值。將修改過權值的新資料集送給下層分類器進行訓練，最後將每次得到的分類器最後融合起來，作為最後的決策分類器。

(1) 選擇資料集

以 electricity.csv 這個 dataset 作為例子

```
#匯入資料
data = pd.read_csv('data/electricity.csv')
```

(2) 資料預處理

與前一節相同

(3) 建立訓練與測試資料

將資料的 70%拿出來 train，剩下的 30%用來檢測 train 的好壞

```
#分割訓練和測試集
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X, y, test_size=0.3, random_state=101)
```

(4) 建立 boosting 模型

4.1 載入隨機森林函式 AdaBoostClassifier

```
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier(n_estimators=100)
clf.fit(X_train, y_train)
```

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=100, random_state=None)
```

4.2 預測 y_pred

```
y_pred= clf.predict(X_test)
print(y_pred)
```

```
[1 2 7 ... 7 2 1]
```

4.3 計算準確率

```
from sklearn import cross_validation, ensemble, preprocessing, metrics
accuracy = metrics.accuracy_score(y_test, y_pred)
print(accuracy)
```

```
0.21436658699207448
```

- **K-近鄰演算法(K-Nearest Neighbor，KNN)**

對於一個待分類的樣本，選取待分類樣本在特徵空間中距離最近的 K 個已標記樣本作為參考，來幫助做出分類決策，因此可以得知 K 值不同，得到的分類器也可能不同，過程中沒有參數訓練過程，所以如果數據集非常的龐大，那麼會增加計算的複雜度。

- **CART(Classification and Regression Trees)**

CART 是一種產生二元樹的技術，以吉尼係數(Gini index)做為選擇屬性的依據。CART 與 ID3、C4.5、C5.0 演算法的最大相異之處是，其在每一個節點上都是採用二分法，也就是一次只能夠有兩個子節點，ID3、C4.5、C5.0 則在每一個節點上可以產生不同數量的分枝。

- **線性回歸(Logistic Regression，LR)**

在線性分類器的時候，為了把實數域的結果映射到(0,1)區間，引入了邏輯斯函數，而在線性回歸問題中，預測目標直接就是實數域上的數值，因此優化目標更為簡單，即最小化預測結果與真實值之間的差值。

- **比較 KNN、CART、LR**

(1) 選擇資料集

以 electricity.csv 這個 dataset 作為例子

```
#匯入資料
data = pd.read_csv('data/electricity.csv')
```

(2) 資料預處理

針對用電量設定類別

```
# 針對用電量設定類別
def get_consumption_category(wt):
    if wt < 200:
        return 1
    elif 200 < wt < 400:
        return 2
    elif 400 < wt < 600:
        return 3
    elif 600 < wt < 800:
        return 4
    elif 800 < wt < 1000:
        return 5
    elif 1000 < wt < 1200:
        return 6
    else:
        return 7

data["electricity_consumption_category"] = data["electricity_consumption"].map(get_consumption_category)
```

挑選 4 個欄位作為主要依據

```
df = data[['temperature','pressure', 'windspeed', 'electricity_consumption_category']]
array = df.values
X = array[:,0:3]
Y = array[:,3]
```

(3) 設置隨機種子

```
# 設置交叉驗證的隨機種子
seed = 7
```

(4) 各類模型比較

定義一個 models 陣列來存放比較的結果

```
# prepare models
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
```

(5) 各類模型比較結果

使用的評估依據是 accuracy 準確率，並搭配 10 次交叉驗證，印出各類別的平均準確率和括號中的標準差。

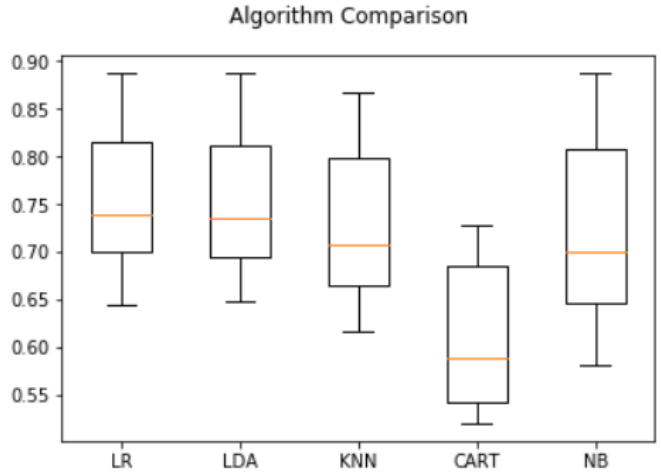
```
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.758570 (0.075302)
LDA: 0.754646 (0.075393)
KNN: 0.727661 (0.081987)
CART: 0.611379 (0.076124)
NB: 0.724644 (0.099898)
```

(6) 視覺化表現

利用箱體圖(boxplot)表現，從下到上五條線分別表示準確度的最小值、下四分位數、中位數、上四分位數和最大值。

```
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



可以發現以 LR 的表現最好。

(7) KNN 比較不同的 K 值

7.1 匯入模組，使用 4 個欄位並分別定義 X 和 Y

```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt

df = data[['temperature','pressure', 'windspeed', 'electricity_consumption_category']]
array = df.values
X = array[:,0:3]
Y = array[:,3]
```

7.2 設定 k 的範圍

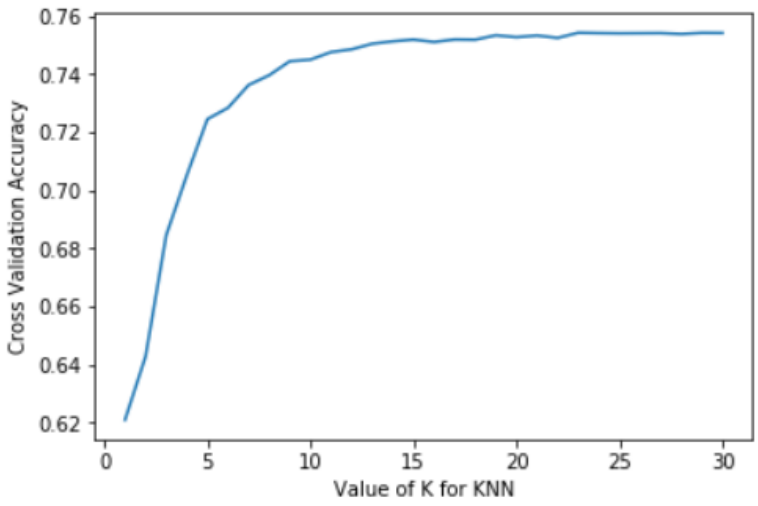
這邊把 range 設在 1~31 這區間，cv=10 代表 10 次交叉驗證法，scoring = 'accuracy'代表使用準確率來評估每一個 K 值。

```
results = []
k_range = range(1, 31)
scoring = 'accuracy'

for k in k_range:
    knn = KNeighborsClassifier(k)
    cv_results = model_selection.cross_val_score(knn, X, Y, cv=10, scoring=scoring)
    results.append(cv_results.mean())
```

7.3 視覺化

```
plt.plot(k_range,results)
plt.xlabel('Value of K for KNN')
plt.ylabel('Cross Validation Accuracy')
plt.show()
```



由上圖發現，隨著 K 值的增加，準確率也會提升，但到了一定的 K 值，準確率成長的趨勢比較緩慢。

◆ 支援向量迴歸(Support Vector Regression，SVR)

它是使用 SVM 來擬合曲線，做迴歸分析。

(1) 匯入模組、載入數據集

以 Position_Salaries.csv 數據集為例

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
# 載入資料集
dataset = pd.read_csv('data/Position_Salaries.csv')
x = dataset.iloc[:,1].values
y = dataset.iloc[:,-1].values
```

(2) 資料預處理

利用 reshape 更改數组的形状，利用 StandardScaler 來計算在一個訓練集上的平均值和標準差，使用這模組的好處是可以保存訓練集中的參數直接使用其物件轉換測試集資料。

```
# reshape x 和 y
x = x.reshape(-1,1)
y = y.reshape(-1,1)
```

```
# feature scaling
from sklearn.preprocessing import StandardScaler
standardscaler_x = StandardScaler()
x = standardscaler_x.fit_transform(x)
standardscaler_y = StandardScaler()
y = standardscaler_y.fit_transform(y)
```

(3) 數據處理(poly kernel、縮放測試數據)

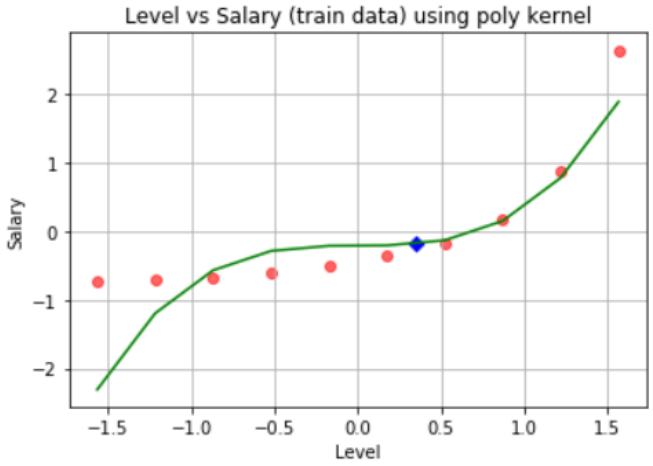
```
# Fittign SVR模型使用POLY內核進行數據處理
from sklearn.svm import SVR
regressor = SVR(kernel='poly')
regressor = regressor.fit(x,y)
```

```
# 縮放測試數據以進行預測
test = np.zeros(1) # we are testing just one value
test[0]= 6.5
test = test.reshape(1,1) # reshape 成二維陣列
test = standardscaler_x.transform(test)
```

```
# 做預測
y_pred = regressor.predict(test)
y_predict = standardscaler_y.inverse_transform(y_pred)
```

(4) poly kernel 視覺化

```
#視覺化fit的模型和數據集
plt.scatter(x,y, color ='red', alpha=0.6)
plt.scatter(test,y_pred,color = 'blue', marker='D')
plt.plot(x,regressor.predict(x),color='green')
plt.title('Level vs Salary (train data) using poly kernel')
plt.xlabel('Level')
plt.ylabel('Salary')
plt.grid()
plt.show()
```



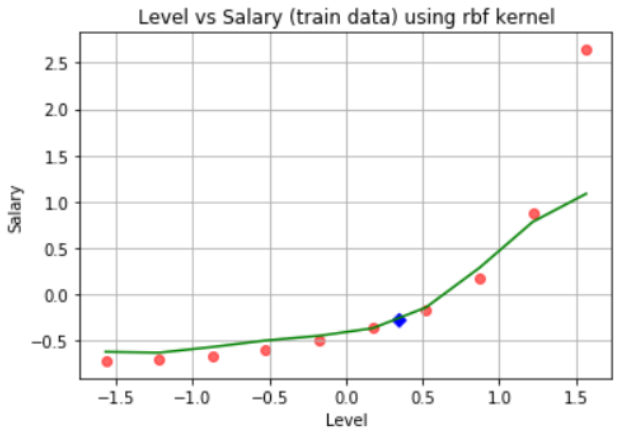
(5) 數據處理(RBF kernel)

```
# 用RBF核與SVR模型fit數據
regressor2 = SVR(kernel='rbf')
regressor2 = regressor2.fit(x,y)

# 做預測
y_pred = regressor2.predict(test)
y_predict = standardscaler_y.inverse_transform(y_pred)
```

(6) RBF kernel 視覺化

```
#視覺化fit的模型和數據集
plt.scatter(x,y, color ='red', alpha=0.6)
plt.scatter(test,y_pred,color = 'blue', marker='D')
plt.plot(x,regressor2.predict(x),color='green')
plt.title('Level vs Salary (train data) using rbf kernel')
plt.xlabel('Level')
plt.ylabel('Salary')
plt.grid()
plt.show()
```



◆ 支援向量分類 (Support Vector Classification，SVC)

它是使用 SVM 來擬合曲線，做分類分析。

(1) 匯入模組、載入數據集

以 fruit_data_with_colors.txt 數據集為例

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
fruits = pd.read_table('fruit_data_with_colors.txt')
fruits.head()
```

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

(2) 數據集資料統計(資料維度、欄位資料查詢)

```
print(fruits.shape)
```

```
(59, 7)
```

```
print(fruits['fruit_name'].unique())
```

```
['apple' 'mandarin' 'orange' 'lemon']
```

```
print(fruits.groupby('fruit_name').size())
```

```
fruit_name
apple       19
lemon       16
mandarin     5
orange      19
dtype: int64
```

(3) 定義新欄位

新增的欄位命名為 feature_names

```
feature_names = ['mass', 'width', 'height', 'color_score']
X = fruits[feature_names]
y = fruits['fruit_label']
```

(4) 分割訓練集和數據集

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

(5) 計算 SVC 的準確率

```
from sklearn.svm import SVC
svm = SVC()
svm.fit(X_train, y_train)
print('Accuracy of SVM classifier on training set: {:.2f}'.format(svm.score(X_train, y_train)))
print('Accuracy of SVM classifier on test set: {:.2f}'.format(svm.score(X_test, y_test)))
```

```
Accuracy of SVM classifier on training set: 0.61
Accuracy of SVM classifier on test set: 0.33
```