機械学習応用(性質を用いた分析)

1　決定境界

ただ分類するだけなら関数を用いればできますが、どのように分類しているかを可視化できれば分類結果に正当性を示すことができ、学習アルゴリズムによっては分布の特徴を得ることができます。

例えば線形 SVM を用いて scikit-learn のライブラリにあるデータセットを用いてデータの決定境界を描画します。描画した結果は図 6-1 のようになります。

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.svm import LinearSVC
from sklearn.datasets.samples_generator import make_blobs
from sklearn import preprocessing
Xdata, ydata = make_blobs(n_samples=50, centers=2,random_state=0, cluster_std=0.60)
Xdata = preprocessing.minmax_scale(Xdata)
model = LinearSVC()
model.fit(Xdata,ydata)
fig, ax = plt.subplots(figsize=(8, 6))
X, Y = np.meshgrid(np.linspace(*ax.get_xlim(), 1000), np.linspace(*ax.get_ylim(), 1000))
XY = np.column_stack([X.ravel(), Y.ravel()])
Z = model.predict(XY).reshape(X.shape)
plt.contourf(X, Y, Z, alpha=0.1, cmap='brg')
plt.scatter(Xdata[:, 0], Xdata[:, 1], c=ydata, s=50, cmap='brg')
plt.xlim(min(Xdata[:,0]),max(Xdata[:,0]))
plt.ylim(min(Xdata[:,1]),max(Xdata[:,1]))
plt.show()
```

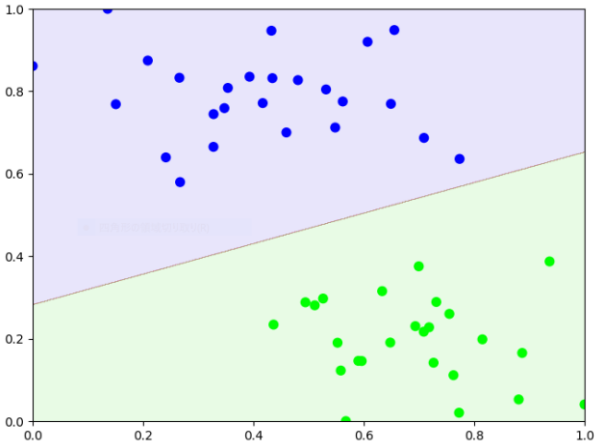


図 6-1　SVM を用いた決定境界の描画

しかし、このようなやり方で決定境界を描画できないアルゴリズムが一例だけあります。それがナイーブベイズです。ナイーブベイズを用いた決定境界の描画ではランダムに点を打ち込み決定境界を描画します。描画した結果は図 6-2 になります。点をある低度増やせば主観的輪郭線で境界が分かるようになると思います。

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.naive_bayes import MultinomialNB
from sklearn.datasets.samples_generator import make_blobs
from sklearn import preprocessing
Xdata, ydata = make_blobs(n_samples=50, centers=2,random_state=0, cluster_std=0.60)
Xdata = preprocessing.minmax_scale(Xdata)
model = MultinomialNB()
model.fit(Xdata,ydata)
fig, ax = plt.subplots(figsize=(8, 6))
rng = np.random.RandomState(0)
Xnew = [min(Xdata[:,0]), min(Xdata[:,1])] + [max(Xdata[:,0])*2, max(Xdata[:,1])*2] * rng.rand(5000, 2)
ynew = model.predict(Xnew)
lim = plt.axis()
plt.scatter(Xnew[:, 0], Xnew[:, 1], c=ynew, s=20, cmap='brg', alpha=0.1)
plt.axis(lim)
plt.scatter(Xdata[:, 0], Xdata[:, 1], c=ydata, s=50, cmap='brg')
plt.xlim(min(Xdata[:,0]),max(Xdata[:,0]))
plt.ylim(min(Xdata[:,1]),max(Xdata[:,1]))
plt.show()
```

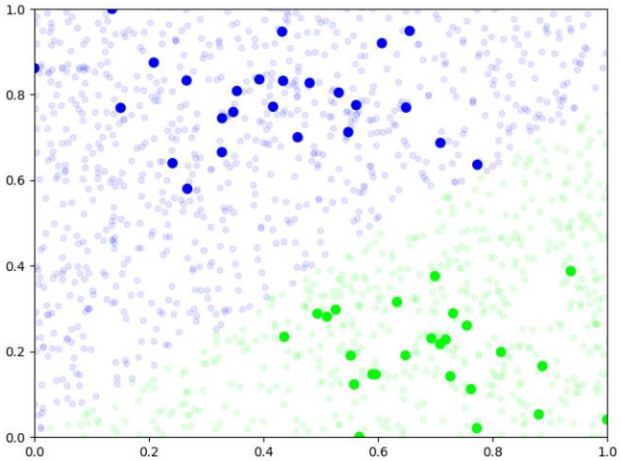


図 6-2　ナイーブベイズを用いた決定境界の描画

このようにデータの分類を可視化することができますが、これらはあくまで説明変数が 2 次元であったらの場合です。実際のデータセットは多くの場合 3 次元以上なので同様に使用する場合は主成分分析などを行いデータを 2 次元にまで圧縮した上で決定境界を描画しましょう。

2　因子分析

2.1　重回帰分析を用いた因子分析

これは連続的な目的変数(y)を伴う説明変数(x)の重要度を測定するのに使います。重回帰分析は以下の式に表すことができ重み(w)の絶対値が大きいものほど重要な変数を表し絶対値が低いほど重要度の低い変数となります。

$$y = w_0x_0 + w_1x_1 + \cdots + w_nx_n + b = \sum_{i=0}^{n} w_ix_i + b$$

例えばコンビニエンスストアのデータに対し重回帰分析を用いて目的変数を売り上げとして説明変数として「品揃え」「立地」「接客」とします。結果として出力された決定係数および切片と重みは図 6-3 のようになります。使用したデータはこちら(https://markezine.jp/article/detail/16294)になり、ファイル名を「uriage2.csv」としています。出力結果に表れる決定係数は 1 に近いほど有効であるという意味になります。

```
from sklearn.linear_model import LinearRegression
import csv
f=open('uriage2.csv','r',encoding='shift-jis')
reader=csv.reader(f)
y=[]
x=[]
KARI=[]
i=0
label=[]
pur="売上高"#目的変数を選択
non="店舗名"#データ分析に使わない項目
for row in reader:
    if i==0:
        for j in range(len(row)):
            label.append(row[j])
        i=1
    else:
        for j in range(len(row)):
```

```
                    if label[j]==pur:
                        y.append(float(row[j]))
                    elif label[j]==non:
                        a=j
                    else:
                        KARI.append(float(row[j]))
                x.append(KARI)
                KARI=[]
del label[a]
factor=LinearRegression()
factor.fit(x,y)
print("決定係数:"+str(factor.score(x,y))+"¥n")
print("回帰係数")
print("切片:"+str(factor.intercept_))
a=0
for j in range(len(factor.coef_)):
    if label[j]==pur:
        a=1
    if a==1:
        print(label[j+1]+":"+str(factor.coef_[j]))
    else:
        print(label[j]+":"+str(factor.coef_[j]))
```

```
決定係数:0.7990932392049901

回帰係数
切片:66.38119946937763
接客:133.58036586763913
品揃え:80.08500608687571
立地:-43.56442534979066
```

図 6-3　重回帰分析を行った結果出力された決定係数および回帰係数

2.2　決定木を用いた因子分析

重回帰分析では連続的で具体的な目的変数の予測を行うための因子分析でしたが、決定木は記号論を用いた分類であるため因子分析ではラベリングされた離散的な目的変数への分類経過を表す樹形図を表示することができます。手法としてはプログラムを作成して dot ファイルを作成し、コマンドプロンプトから樹形図を作成します。ただし、Graphviz というソフトウェアが無いとできません。

例えば、scikit-learn のライブラリに入っているワインデータの分類に対して樹形図を表示すると図 6-4 のようになります。

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_wine
from sklearn import tree
data=load_wine()
X=data.data
y=data.target
model = RandomForestClassifier(n_estimators=1)
model.fit(X,y)
for i,val in enumerate(model.estimators_):
    tree.export_graphviz(model.estimators_[i], out_file='tree/tree_%d.dot'%i)
```

コマンドライン（Graphviz2.38¥bin から）

```
>dot -T pdf dot ファイルの絶対参照  -o  出力先を絶対参照
```

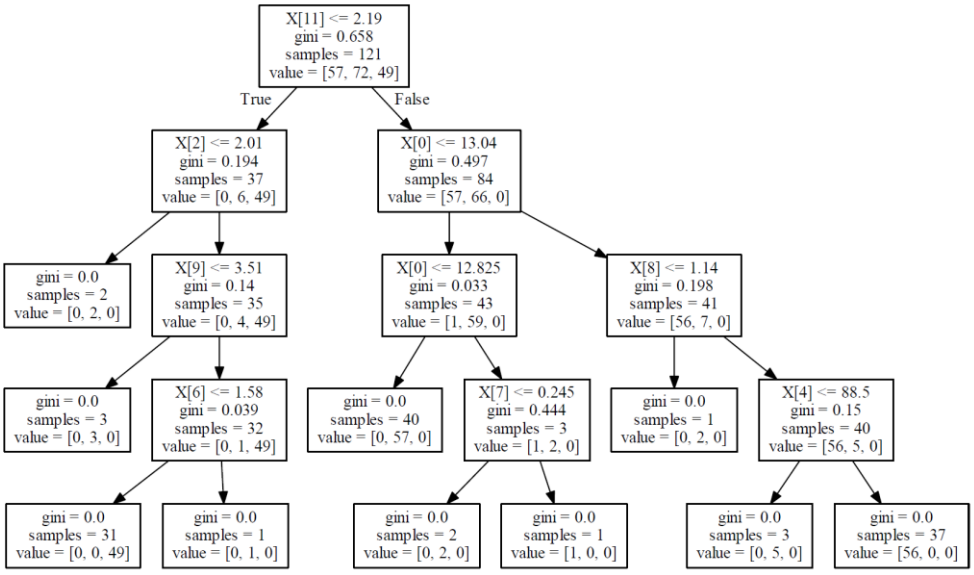


図 6-4　決定木を可視化した樹形図

2.3　主成分分析を用いた因子分析

主成分分析は教師無し学習の一つであるため目的変数の有無を問わず説明変数の中でどの程度特徴量をもっているかを表すことができます。その中で、どの因子がどの程度主成分に強く寄与しているかを表すかに固有ベクトルを用います。そしてこの固有ベクトルは主成分の累積寄与率を算出した時に 0.8 を超えるまでの主成分を用います。

例えば、scikit-learn のライブラリに入っているワインデータを用いて固有ベクトルを算出すると表 6-1 のように因子分析を行うことができます。この時、累積寄与率は図 6-5 のようになるため第 5 主成分まで分析します。

```
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
from sklearn import preprocessing
import csv
data=load_wine()
X=data.data
y=data.target
features = preprocessing.minmax_scale(data.data)
pca=PCA(n_components=len(features[0]))
pca=pca.fit(features)
KIYO=[]
K=[]
kiyoisu=pca.explained_variance_ratio_
for i in range(len(kiyoisu)):
    K.append("PCA"+str(i+1))
    K.append(str(float(kiyoisu[i])))
    KIYO.append(K)
    K=[]
koyuuvector=pca.components_
VEC=[]
K=[]
for i in range(len(koyuuvector)):
    K.append("PCA"+str(i+1))
    for j in range(len(koyuuvector[i])):
        K.append(str(float(koyuuvector[i][j])))
    VEC.append(K)
    K=[]
```

表 6-1　算出した主成分ごとの固有ベクトル

	因子1	因子2	因子3	因子4	因子5	因子6	因子7	因子8	因子9	因子10	因子11	因子12	因子13
PCA1	-0.13337	0.248516	-0.00074	0.177839	-0.08866	-0.39507	-0.41459	0.333109	-0.2529	0.092329	-0.25114	-0.47349	-0.28686
PCA2	-0.55088	-0.22739	-0.16309	0.079776	-0.18817	-0.07414	-0.00101	-0.00996	-0.03142	-0.51971	0.237206	0.215562	-0.44389
PCA3	-0.08385	0.492039	0.403009	0.477242	0.006551	0.253065	0.196105	0.285982	0.228342	-0.0331	-0.10659	0.297776	-0.15197
PCA4	-0.04033	0.486032	-0.24198	-0.08169	0.01588	-0.05264	-0.02703	-0.7091	0.076574	-0.02632	-0.3525	0.082062	-0.22979
PCA5	-0.27801	-0.41289	0.286394	0.418305	0.470899	-0.01803	0.002573	-0.36004	0.148666	0.200582	-0.07716	-0.2634	-0.07656

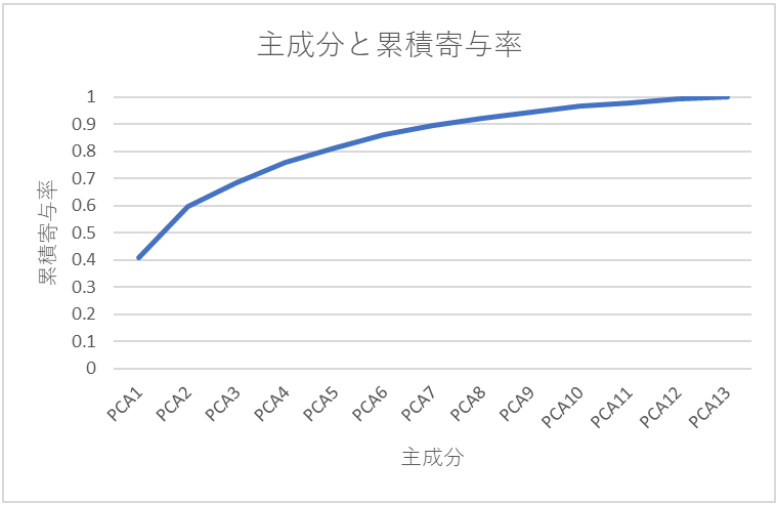


図 6-5　算出した累積寄与率

表 6-1 から因子分析を行う。この時、各固有ベクトルの絶対値が 1 に近いほど重要な数値となる。表 6-1 から重要な因子を 3 つピックアップすると各主成分ごとの重要な因子は表 6-2 のようになる。

表 6-2　第 5 主成分までで重要な因子を 3 つピックアップした結果

	因子1	因子2	因子3	因子4	因子5	因子6	因子7	因子8	因子9	因子10	因子11	因子12	因子13
PCA1	-0.13337	0.248516	-0.00074	0.177839	-0.08866	-0.39507	-0.41459	0.333109	-0.2529	0.092329	-0.25114	-0.47349	-0.28686
PCA2	-0.55088	-0.22739	-0.16309	0.079776	-0.18817	-0.07414	-0.00101	-0.00996	-0.03142	-0.51971	0.237206	0.215562	-0.44389
PCA3	-0.08385	0.492039	0.403009	0.477242	0.006551	0.253065	0.196105	0.285982	0.228342	-0.0331	-0.10659	0.297776	-0.15197
PCA4	-0.04033	0.486032	-0.24198	-0.08169	0.01588	-0.05264	-0.02703	-0.7091	0.076574	-0.02632	-0.3525	0.082062	-0.22979
PCA5	-0.27801	-0.41289	0.286394	0.418305	0.470899	-0.01803	0.002573	-0.36004	0.148666	0.200582	-0.07716	-0.2634	-0.07656

3　Softmax 値

前章で二層以上のニューラルネットワークの仕組みに触れましたが、その中に出力層の計算として Softmax 関数があります。この Softmax の値について掘り下げると、どのラベルに分類されるかについてを百分率でそれぞれのラベルの確率を表しています。言い換えてみれば学習アルゴリズムが「どれくらい自信をもってその結果にしたのか」についての指標になります。

例えば、scikit-learn のライブラリにある手書き文字のデータセットを分類した時のそれぞれの Softmax の値を格納し最初の 5 つの分類の Softmax 値は表 6-3 のようになります。なお、最も大きい Softmax 値が分類結果として出力されるラベルになります。

```
from sklearn.datasets import load_digits
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
import csv
data=load_digits()
x=data.images.reshape(len(data.images),-1)
y=data.target
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
model=model=MLPClassifier(hidden_layer_sizes=(500,500,))
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
fb = open('softmax.csv', 'w',encoding='utf-8')
writer = csv.writer(fb, lineterminator='¥n')
writer.writerows(model.predict_proba(X_test))
fb.close()
```

表 6-3　各目的変数に対する Softmax 値

	0	1	2	3	4	5	6	7	8	9
0	8.60E-10	0.000329	3.65E-09	1.12E-09	1.39E-05	6.42E-13	1.93E-11	0.999504	1.22E-08	0.000153
1	2.18E-09	3.55E-11	6.36E-15	2.26E-14	1	9.74E-12	2.33E-07	1.35E-10	7.68E-11	5.83E-15
2	1.92E-09	1.00E-07	1.40E-09	1.59E-07	4.06E-10	7.85E-07	4.64E-08	2.27E-10	0.999999	1.69E-07
3	2.09E-07	1.07E-06	1.44E-07	3.60E-06	2.03E-05	1.78E-06	0.99995	1.04E-07	2.32E-05	1.58E-08
4	7.76E-08	2.02E-11	1.02E-10	4.68E-09	3.79E-12	1.39E-07	1.35E-13	5.49E-09	2.83E-08	1
5	0.999994	8.51E-11	1.75E-08	2.00E-09	2.61E-09	7.04E-08	5.79E-06	5.91E-10	4.50E-08	1.39E-07

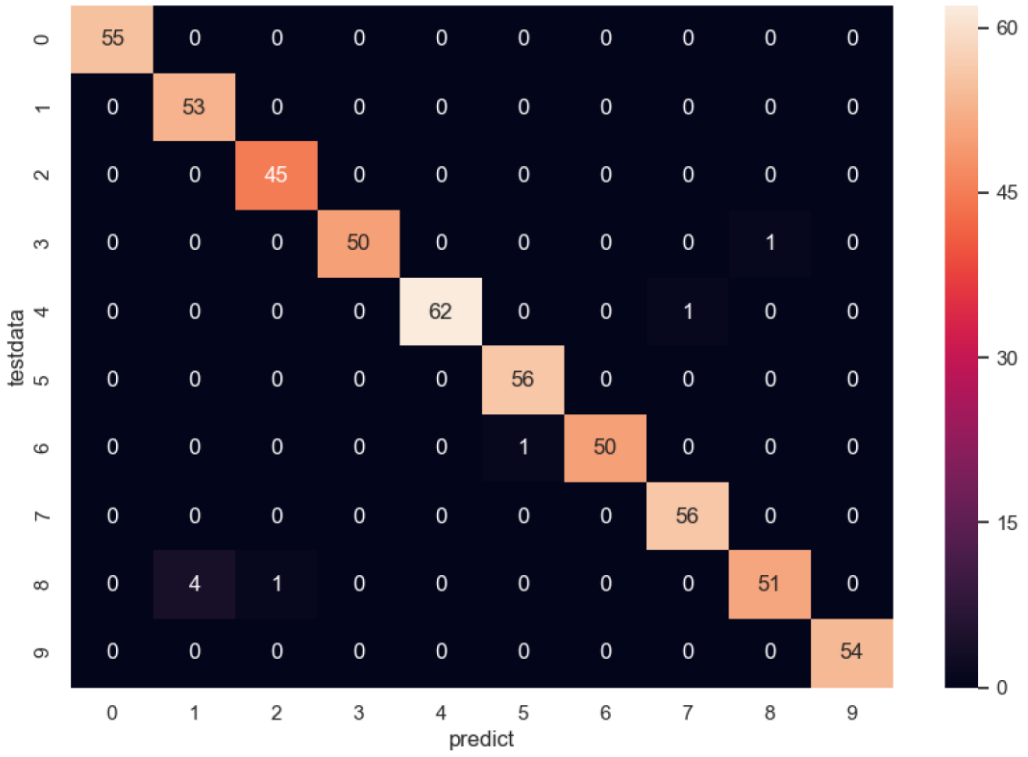
4　分類結果の可視化

どんな機械学習においても分類結果が必ずしも正しいとは限りません。そのため、テストデータと予測の齟齬を見れば時としてどのデータとどのデータが判別しにくいかというのを知ることができます。ここではテストデータに対して予測がどうなっているかについてマトリックス表を用いて表します。

例えば、scikit-learn のライブラリにある手書き文字のデータセットを分類した時におけるテストデータと予測のマトリックス表は表 6-4 のようになります。

```
from sklearn.datasets import load_digits
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns; sns.set()
data=load_digits()
x=data.images.reshape(len(data.images),-1)
y=data.target
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
model=model=MLPClassifier(hidden_layer_sizes=(500,500,))
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
labels = sorted(list(set(y_test)))
cmx_data = confusion_matrix(y_test, y_pred, labels=labels)
df_cmx = pd.DataFrame(cmx_data, index=labels, columns=labels)
plt.figure(figsize = (10,7))
sns.heatmap(df_cmx, annot=True)
plt.xlabel("predict")
plt.ylabel("testdata")
plt.show()
```

表 6-4　テストデータと予測のマトリックス表



5　教師有り学習における精度の指標

ここから先は 4 項のソースコードに付け加える形で記述していきます。

表 6-4 における正解率・再現率・適合率・F 値は図 6-6 にように出力されます。一番上の「0.9888888888888889」は正解率で「precision」は適合率を表し、「recall」は再現率で「f1-score」は F 値を表します。

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

```
0.9888888888888889
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        59
           1       1.00      1.00      1.00        55
           2       1.00      1.00      1.00        54
           3       1.00      0.98      0.99        54
           4       0.98      1.00      0.99        47
           5       0.95      1.00      0.98        61
           6       1.00      0.98      0.99        54
           7       0.98      1.00      0.99        54
           8       0.98      0.98      0.98        46
           9       1.00      0.96      0.98        56

   micro avg       0.99      0.99      0.99       540
   macro avg       0.99      0.99      0.99       540
weighted avg       0.99      0.99      0.99       540
```

図 6-6　表 6-4 における各指標

5.1　正解率(Accuracy)

これは読んで字のごとく全テストデータに対して予測がどれだけ合っていたかの指標です。

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

5.2　再現率(Recall)

再現率は結果として本来出力されるデータのうち予測で実際に出てきた割合の指標です。

```
from sklearn.metrics import recall_score
recall_score(y_test,y_pred,average='macro')
```

5.3 適合率(Precision)

適合率はシステムが出力した結果の中で実際に正しかったものの割合になります。

```
from sklearn.metrics import precision_score
precision_score(y_test,y_pred,average='macro')
```

5.4 F 値(f1)

F 値は適合率と再現率の両方の傾向を反映させたもので、計算方法としては再現率と適合率の調和平均を用います。

$$F1score = \frac{2 \times precision \times recall}{precision + recall}$$

```
from sklearn.metrics import f1_score
f1_score(y_test,y_pred,average='macro')
```

6 教師無し学習から教師有り学習へ

データそのものにラベリングがされていない未知のデータがあった時にグルーピングとして k-means 法などを用いたクラスタリングを行います。そうすれば k-means 法の出力結果として各データがラベリングされるため未知のデータが新たに入ってきても再度クラスタリングを行う必要が無く教師有り学習でどのグループに属するかを分類することができます。

例えば scikit-learn のライブラリにあるデータセットに対して k-means 法でラベリングを行い SVM によって分類し、データの決定境界を描画します。分類前のデータと分類して描画した結果は図 6-7 のようになります。

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets.samples_generator import make_blobs
from sklearn import preprocessing
from sklearn.cluster import KMeans
from sklearn.svm import LinearSVC
Xdata, y_true = make_blobs(n_samples=300, centers=4,cluster_std=0.60, random_state=0)
Xdata = preprocessing.minmax_scale(Xdata)
plt.scatter(Xdata[:, 0], Xdata[:, 1], s=50)
plt.show()
kmeans = KMeans(n_clusters=4)
kmeans.fit(Xdata)
y_kmeans = kmeans.predict(Xdata)
model=LinearSVC()
model.fit(Xdata,y_kmeans)
fig, ax = plt.subplots(figsize=(8, 6))
X, Y = np.meshgrid(np.linspace(*ax.get_xlim(), 1000), np.linspace(*ax.get_ylim(), 1000))
XY = np.column_stack([X.ravel(), Y.ravel()])
Z = model.predict(XY).reshape(X.shape)
plt.contourf(X, Y, Z, alpha=0.1, cmap='brg')
plt.scatter(Xdata[:, 0], Xdata[:, 1], c=y_kmeans, s=50, cmap='brg')
plt.xlim(min(Xdata[:,0]),max(Xdata[:,0]))
plt.ylim(min(Xdata[:,1]),max(Xdata[:,1]))
plt.show()
```

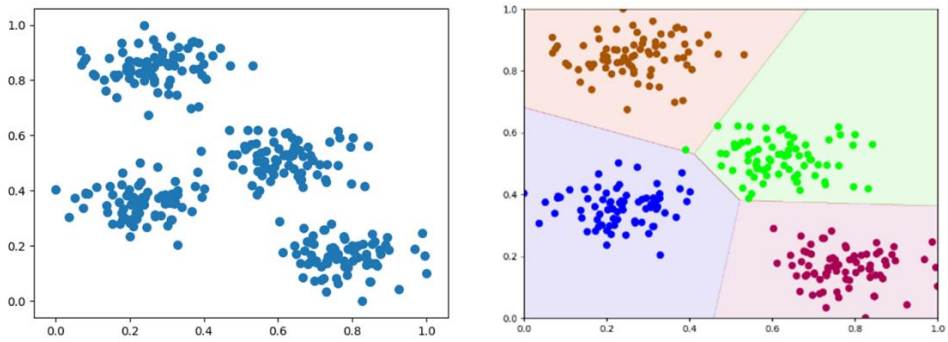


図 6-7　クラスタリングされたデータを SVM で学習させた結果

7　特徴抽出を用いたノイズ除去

2.3 項で主成分分析を用いた因子分析に触れました。ここでは特徴量を抽出することでどの因子が特徴を持っているかについて分析しています。そこで、主成分分析を用いれば使用するデータセットから特徴量を抽出してノイズを除去できます。実際にあらゆる統計データにおいて少なからずノイズ(外れ値など)が入っていることの方が多いです。

例えば、scikit-learn のライブラリにある手書き文字のデータセットに対して図 6-8 のように人工的にノイズを付与します。そこで主成分分析を用いてノイズを除去すれば図 6-9 のようになります。ただし、何次元にまで圧縮するかは調整していく必要があるためここでは分散の 50%を主成分として使います。

```
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt
import numpy as np
def plot_mnist(X):
    fig, axes = plt.subplots(4, 10, figsize=(10, 4),subplot_kw={'xticks':[],
'yticks':[]},gridspec_kw=dict(hspace=0.1, wspace=0.1))
    for i, ax in enumerate(axes.flat):
        ax.imshow(X[i].reshape(8, 8),cmap='binary', interpolation='nearest',clim=(0, 16))
    plt.show()
data=load_digits()
X=data.data
plot_mnist(X)
np.random.seed(42)
noise=np.random.normal(X,4)
plot_mnist(noise)
pca=PCA(0.5)
pca.fit(noise)
com=pca.transform(noise)
filterX=pca.inverse_transform(com)
plot_mnist(filterX)
```

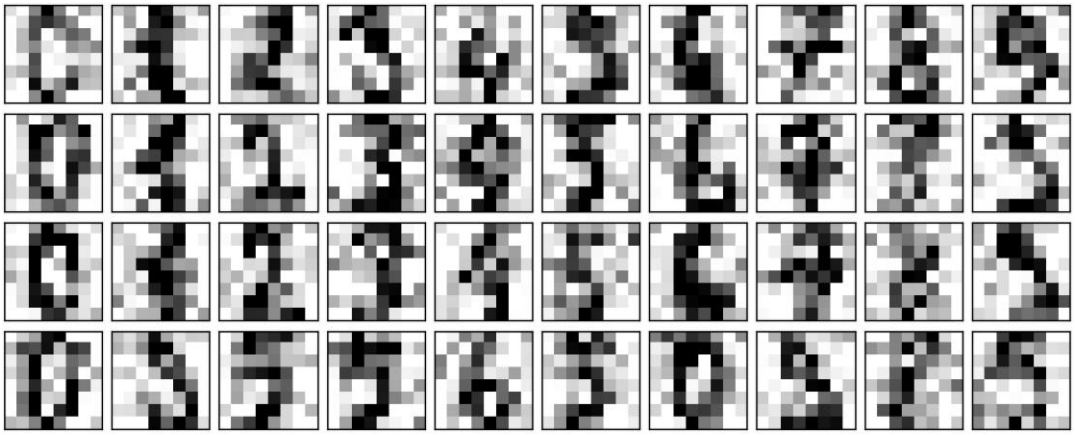


図 6-8　人工的にノイズを付与した手書き文字

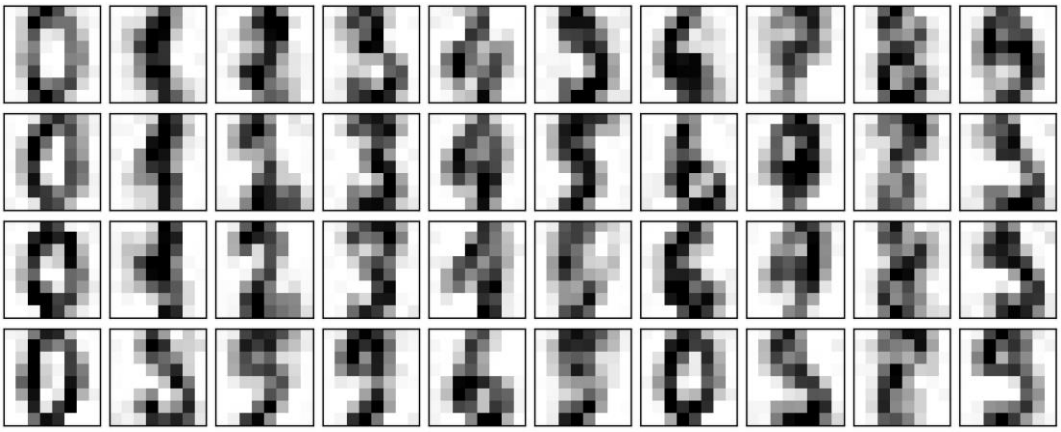


図 6-9　主成分分析を用いてノイズを除去した手書き文字

8　数量化理論

今まで使用したデータは説明変数が必ず具体的な数値でしたが、統計的データにはYes/No等のカテゴリを用いた値もあります。そもそもデータにおける数値は表6-4ような4種類があります。

表6-4　統計学における4つの尺度

尺度	値の意味	統計量	例
名義尺度	同じ値かどうかのみ意味がある	度数 最頻値	性別 好きな色 職業
順序尺度	値の大小に意味がある	上に加え 中央値 四分位	成績段階 レビュー
間隔尺度	値の大小と差の大きさに意味がある。 0は相対的な意味にしかならない	上に加え 平均 標準偏差	摂氏での気温 偏差値
比例尺度	値の大小・差・比に意味がある 0が絶対的な意味を持っている	上に加え 変動係数 幾何平均	身長 体重 年齢

ここで、数量化理論では「名義尺度」と「順序尺度」を説明変数に使います。具体的な実装方法はカテゴリの値に対してダミー変数を与えます。

8.1　数量化理論Ⅰ類

数量化理論1類では説明変数がカテゴリで目的変数が「間隔尺度」または「比例尺度」になります。実装方法としてはダミー変数を用いて重回帰分析を使います。

例として、Wikipediaの決定木のページにある表(以下「ゴルフデータ」)を用いて「天気」と「風」から「温度」を推定する時の因子分析を行います。

```
import pandas as pd
from sklearn import linear_model
# CSV ファイルを読み込んでデータフレームに格納
df = pd.read_csv("m1.csv",encoding = "shift-jis")
```

```
# 説明変数をダミー変数に変換
x = pd.get_dummies(df[['天気', '風']])

# 目的変数:満足度
y = df['気温'].values

# 予測モデルを作成(重回帰)
reg = linear_model.LinearRegression()
reg.fit(x, y)

# 回帰係数と切片の抽出
a = reg.coef_
b = reg.intercept_

#出力

out = []
tmp = []
for i in range(len(x.columns)):
    tmp.append(x.columns[i])
    tmp.append(a[i])
    out.append(tmp)
    tmp = []
tmp.append("切片")
tmp.append(b)
out.append(tmp)
tmp = []
tmp.append("決定係数")
tmp.append(reg.score(x,y))
out.append(tmp)
print(out)
```

出力

```
[['天気_晴', 1.4078431372549018], ['天気_曇', 0.7843137254901965], ['天気_雨', -2.1921568627450965], ['風_弱', 1.1323529411764708], ['風_強', -1.1323529411764706], ['切片', 22.965686274509803], ['決定係数', 0.31781519883830567]]
```

8.2　数量化理論Ⅱ類

数量化理論Ⅰ類では目的変数が連続的な値で回帰問題でした。それに対して数量化理論Ⅱ類では目的変数が離散的でカテゴリとなっている分類問題となります。そのため、決定木やランダムフォレストを用いることで各因子の重要度を求めることができます。ただし、アルゴリズムの特性上毎回重要度が変わります。

例として、ゴルフデータにおける「天気」と「風」から「ゴルフ」(をするかしないか)についての分類において各因子の重要度を算出します。

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

# CSV ァイルを読み込んでデータフレームに格納
df = pd.read_csv("m1-2.csv",encoding = "shift-jis")

# 説明変数をダミー変数に変換
x = pd.get_dummies(df[['天気', '風']])

# 目的変数:満足度
y = df['ゴルフ'].values

# 予測モデルを作成(ランダムフォレスト)
clf = RandomForestClassifier(n_estimators=10)
clf.fit(x, y)
importances = clf.feature_importances_

#出力

out = []
tmp = []
for i in range(len(x.columns)):
    tmp.append(x.columns[i])
    tmp.append(importances[i])
    out.append(tmp)
    tmp = []
print(out)
```

出力

```
[['天気_晴', 0.20534091391777584], ['天気_曇', 0.19905239110367315], ['天気_雨',
0.18485532210287361], ['風_弱', 0.0724758654806732], ['風_強', 0.3382755073950042]]
```

8.3　数量化理論Ⅲ類

Ⅰ類とⅡ類では目的変数(教師データ)がありましたが、Ⅲ類では教師データがありません。その場合は主成分分析を用いて因子分析を行います。手法としては各主成分の寄与率を各因子の主成分ごとに掛けて総和を求めます。

例として、ゴルフデータの「天気」と「風」から因子の重要度を算出します。

```
import pandas as pd
from sklearn.decomposition import PCA

# CSV ァイルを読み込んでデータフレームに格納
df = pd.read_csv("m1.csv",encoding = "shift-jis")

#  説明変数をダミー変数に変換
x = pd.get_dummies(df[['天気', '風']])

#主成分分析で予測
pca = PCA(n_components = len(x.columns))
pca.fit(x)
kiyoritsu = pca.explained_variance_ratio_
koyuuvector = pca.components_
out = []
tmp = []

#出力

for i in range(len(x.columns)):
    tmp.append(x.columns[i])
    sump = 0
    for j in range(len(koyuuvector)):
        sump = sump + abs(koyuuvector[j][i]) * kiyoritsu[j]
    tmp.append(sump)
    out.append(tmp)
    tmp = []
print(out)
```

出力

```
[['天気_晴', 0.3553939540532539], ['天気_曇', 0.2727571586989127], ['天気_雨',
0.35539395405325414], ['風_弱', 0.3328276549341541], ['風_強', 0.33282765493415406]]
```

8.4　数量化理論Ⅳ類

Ⅰ類からⅢ類までは具体的な数値として出力していました。しかしⅣ類では低次元に圧縮して画像を分布の画像を出力します。

例として、ゴルフデータの「天気」「風」「ゴルフ」にダミー変数を与えて二次元に圧縮して散布図を出力します。ここで、データにはそれぞれのラベルが無いため行番号をデータとして出力します。その結果は図 6-10 のようになります。

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
# CSV ァイルを読み込んでデータフレームに格納
df = pd.read_csv("m1.csv",encoding = "shift-jis")

# 説明変数をダミー変数に変換
x = pd.get_dummies(df[['天気', '風','ゴルフ']])

#主成分分析で圧縮
pca = PCA(n_components=2)
pca.fit(x)
tx = pca.transform(x)

#図示
for i in range(len(tx)):
    plt.text(tx[i][0], tx[i][1], str(i))
plt.scatter(tx[:,0], tx[:,1])

plt.xlim(min(tx[:,0]) - 0.5, max(tx[:,0]) + 0.5)
plt.ylim(min(tx[:,1]) - 0.5, max(tx[:,1]) + 0.5)
plt.show()
```

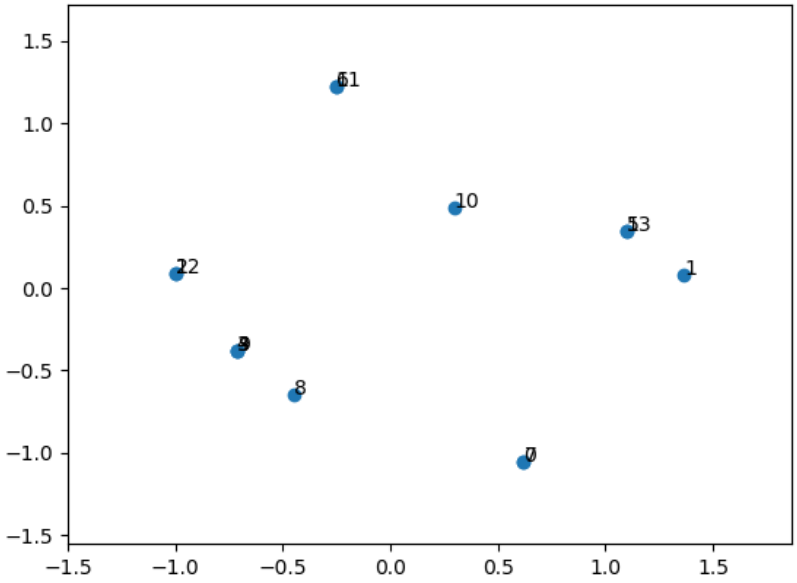


図 6-10　各因子を圧縮した散布図