問1（ワインデータ）

(1) 隠れ素子数の数 M や正則化項のパラメータ λ は，CV 法を用いて汎化誤差が最小になるように選択した．具体的には，M は 1~10，λ は 0.1, 0.01, 0.001, 0.0001, 0.00001 でそれぞれを組み合わせて調査した．選択した結果は以下のようになった．プログラムはソースコード 1 であり，この時の CV 法は 10 分割で行い，それによる誤り率は，0.064 であった．また，正確度は 0.98 であった．

$$M = 6, \quad \lambda = 0.00001$$

図 1 のように識別境界のプロットを試みたが，上手くプロットすることができなかった．

(2) 線形カーネルと RBF カーネルを取り上げ，for 文でいくつかのパラメータ c や σ で調査した結果，汎化誤差が最小になったのは，線形カーネルを用いた時の $c = 1$ であった．この時の正確度は，1.0 であった．（RBF カーネルよりも線形カーネルの方が正確度が高かった．）

また，識別境界のプロットを以下の図 1 に示す．このプロットは，最初に LDA(Linear Discriminant Analysis) を使って 13 次元のワインデータを 2 次元に射影した．

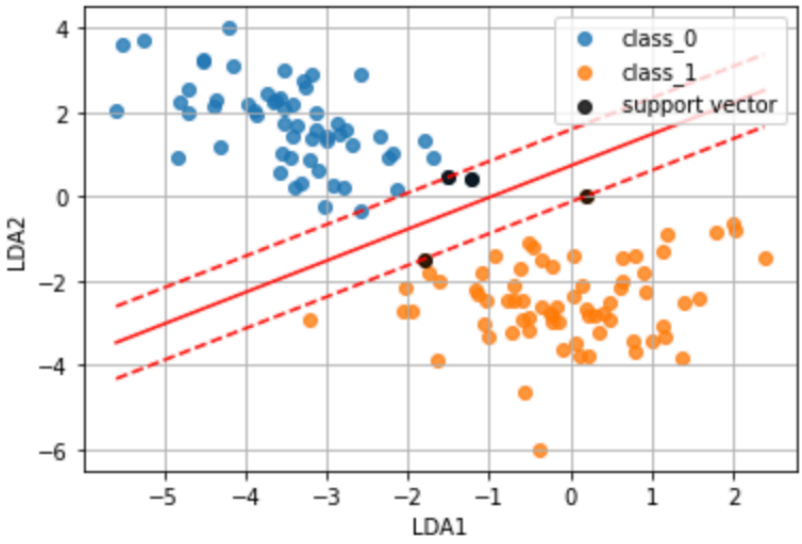

図 1: SVM の識別境界のプロット（線形カーネル，$c = 1$）

(3) (1)，(2) の結果，ワインデータの識別には，サポートベクトルマシンの学習規則を用いる方が望ましい．

正確度を見ると，サポートベクトルマシンの方が大きいため望ましいと考える．これは，データの次元が大きい場合でも識別精度が良いという SVM の特徴が現れているからだと考える．しかし，誤差逆伝搬法の学習規則でも正確度が高いと言えるため，どちらの学習規則も精度は良いと考えられる．

問2（カニデータ）

(1) 体長を記録した 5 つの特徴を用いて主成分分析を行い，2 次元に射影したデータのプロットは以下の図 2 である．また以下，PC1，PC2 を第 1 主成分，第 2 主成分として表している．

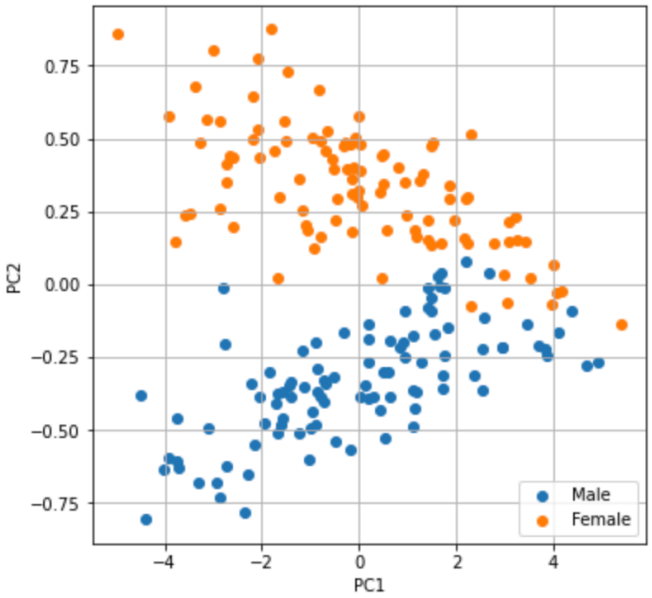

図 2: 5 つの特徴の主成分分析による 2 次元に射影したデータのプロット

また，寄与率と累積寄与率は以下の表 1 の結果となった．また，累積寄与率のプロットを以下の図 3 に示す．

表 1: カニデータの寄与率と累積寄与率

	PC1	PC2	PC3	PC4	PC5
寄与率	0.958	0.0303	0.00933	0.00223	0.000342
累積寄与率	0.958	0.988	0.998	1.00	1.00

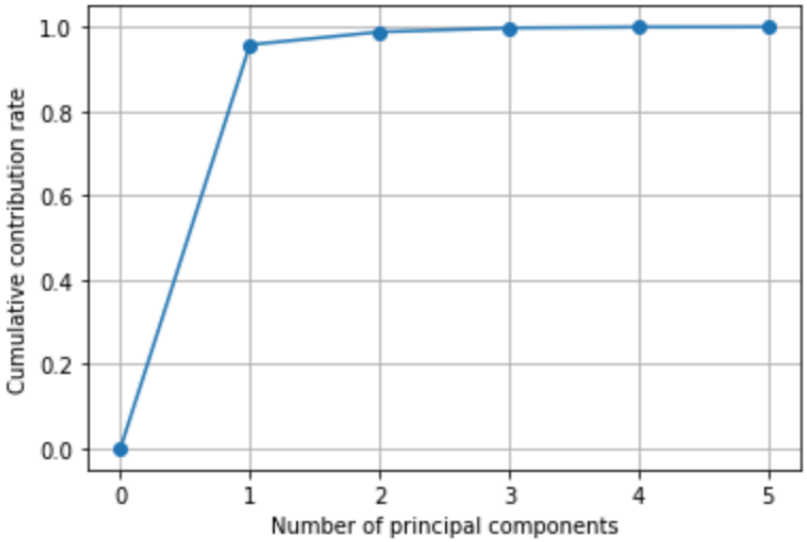

図 3: カニデータの累積寄与率のプロット

従って，第 1 主成分の寄与率は，0.958，第 2 主成分の寄与率は，0.0303である．累積寄与率は，表 1 の通りである．

(2) 体長を記録した 5 つの特徴を用いて階層的クラスタリングを行った結果は，以下の図 4 のデンドログラム通りである．

手法については，クラスタ間の距離は，Ward 法，対象間の距離は，ユークリッド距離を採用した．

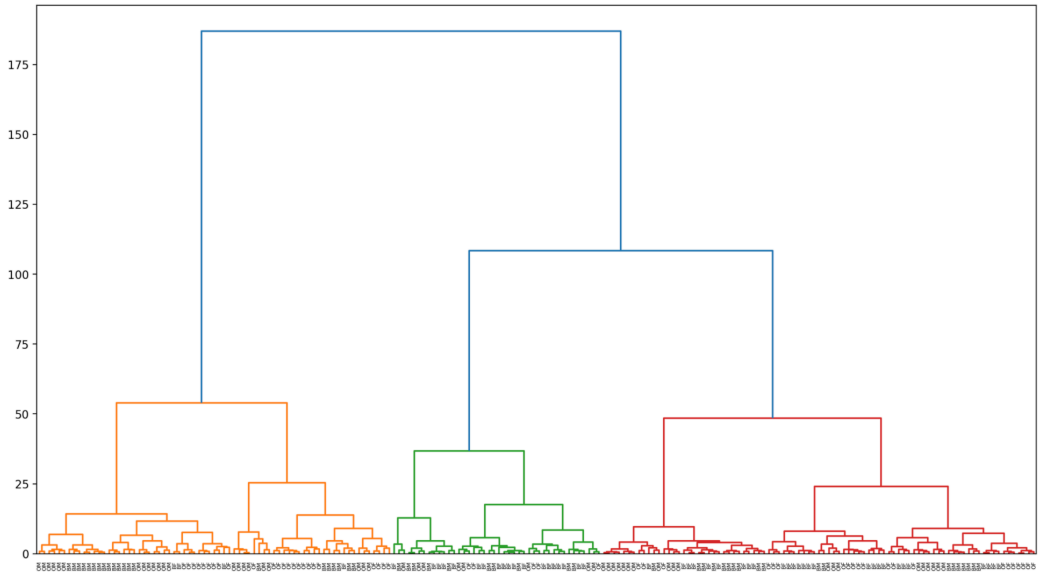

図 4: カニデータのクラスタリング結果（Ward 法）

また，手法を対象間の距離は，ユークリッド距離のままで，クラスタ間の距離を最短距離法（Single）にした場合の結果を以下の図 5 に示す．

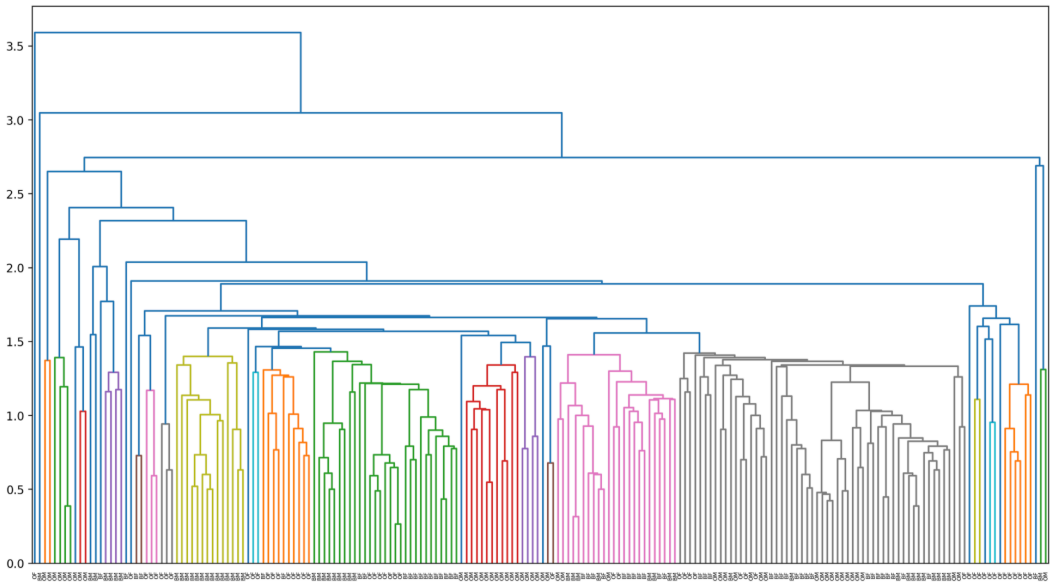

図 5: カニデータのクラスタリング結果（最短距離法）

教師なし分類とは，教師が無い（予め与えられたクラスが無い）学習によって分類するということであり，この場合，入力データ間の距離や類似度，統計的な性質に基づいて，ク

ラスを自動的に生成するということである．そして，今回行った「階層的クラスタリング」が教師なし分類である．

図 4 の階層的クラスタリングは，比較的グループ分けができていると考える．それに対し図 5 では，鎖のように連なっているようになっており，非常に分かりづらいグループ分けとなっている．よって，階層的クラスタリングは，対象間とクラスタ間の距離の定義が非常に重要であることが言える．また，クラスタリングは距離にのみ基づいているため，解釈するためには，別の情報が必要になってくると考えられる．

(3) **問 1** と同様に，誤差逆伝搬法とサポートベクトルマシンを用いた識別を行う．

誤差逆伝搬法では，CV 法を用いて汎化誤差が最小になるように M と λ を選択した．その結果，以下のようになった．

$$M = 7, \quad \lambda = 0.1$$

この時の正確度は，0.983 であった．

SVM でも**問 1** と同様に（ソースコード 1 と同様のパラメータの調べ方），調べた結果，線形カーネルと RBF カーネル共に正確度が 0.983 になった．この時のパラメータは，以下の表 2 の通りである．

表 2: SVM のパラメータ

	線形カーネル	RBF カーネル
パラメータ	$c = 10$	$c = 1000,\ \sigma = 0.8$

SVM の識別境界のプロットを以下の図 6 に示す．

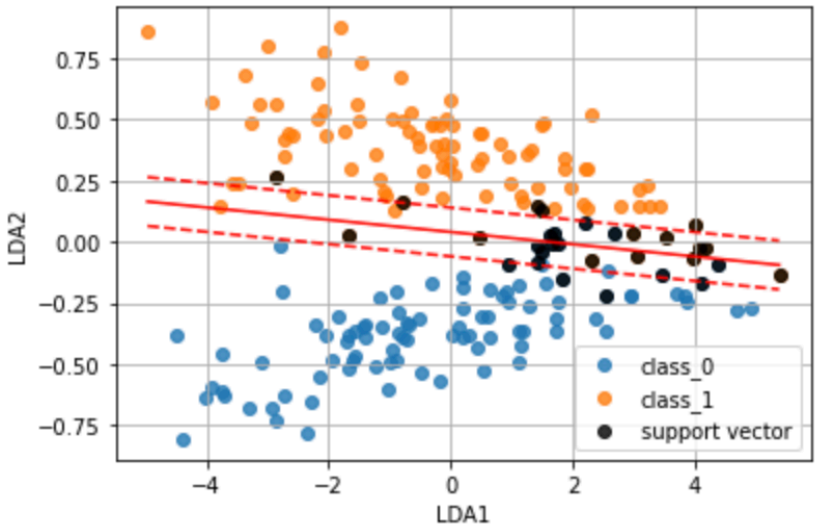

図 6: SVM の識別境界のプロット（線形カーネル，$c = 10$）

以上より，誤差逆伝搬法と SVM の両者の正確度は同じ結果となり，どちらも精度の良い識別規則であることが言える．

問3（タイタニックデータ）

(1) 今回用いる変数は，”Pclass”，”Age”，”Sex”，”Fare”，”SibSp”，”Parch”，”Embarked” の 7 つの変数である．（プログラムで容易に扱えると考えた 7 つである．）

そして決定木，バギング，ブースティング (アダブースト)，ランダムフォレストにより識別器を構成し，学習データを交差検証法によって誤り率を求め，以下の表 3 にまとめた．交差検証法は全て 10 分割に設定している．

表 3: 学習データの誤り率

	決定木	バギング	アダブースト	ランダムフォレスト
誤り率	0.184	0.173	0.186	0.165

また，木の深さや集団学習に使う弱学習器の個数を変化させながらの正確度のプロットは，以下の図 7，図 8，図 9，図 10 に示す．

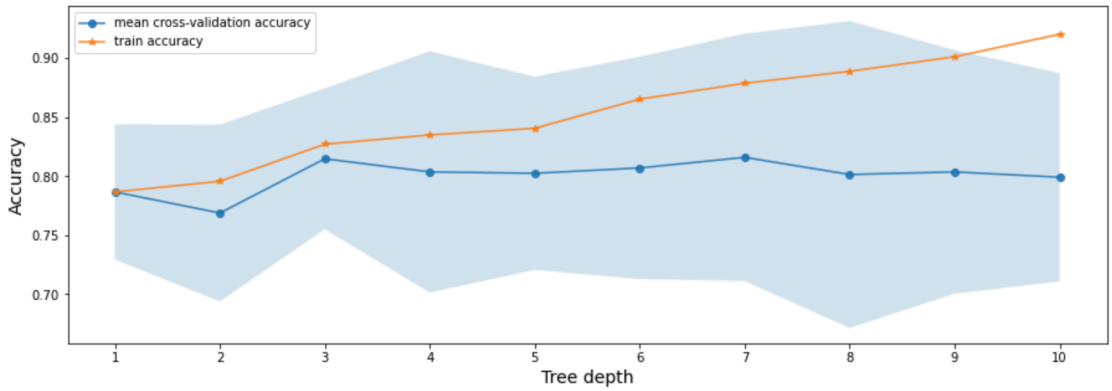

図 7: 決定木の正確度のプロット

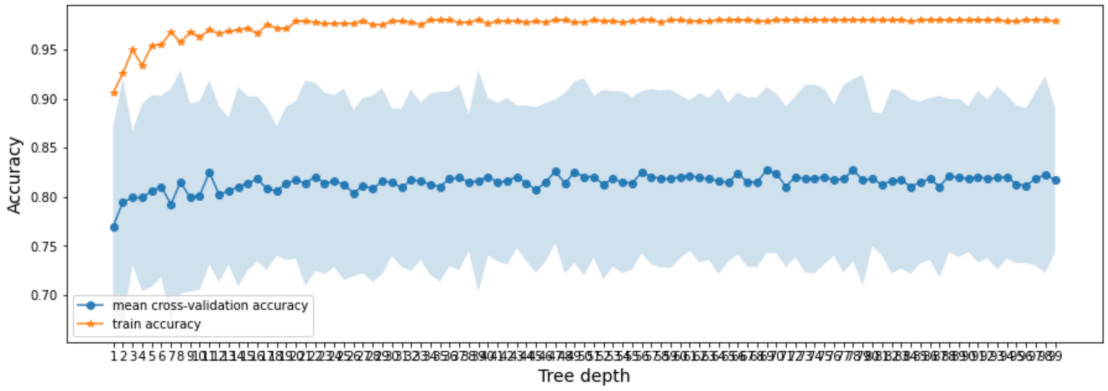

図 8: バギングの正確度のプロット

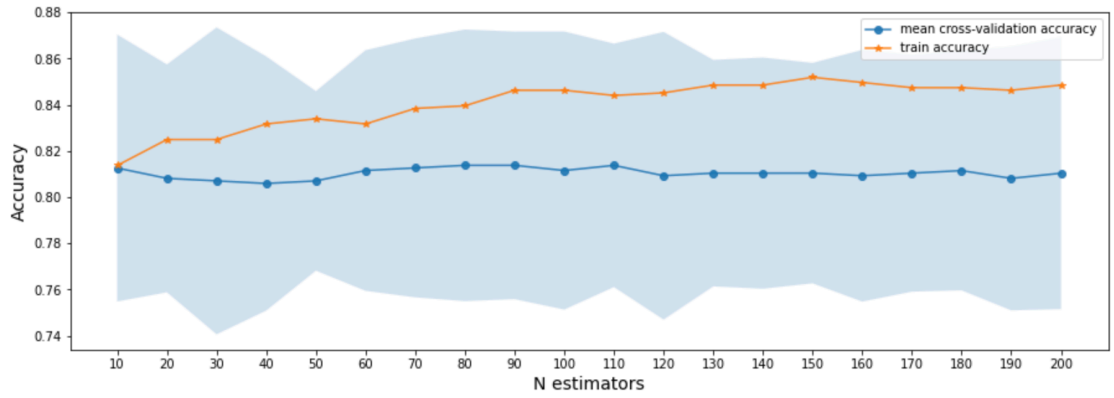

図 9: アダブーストの正確度のプロット

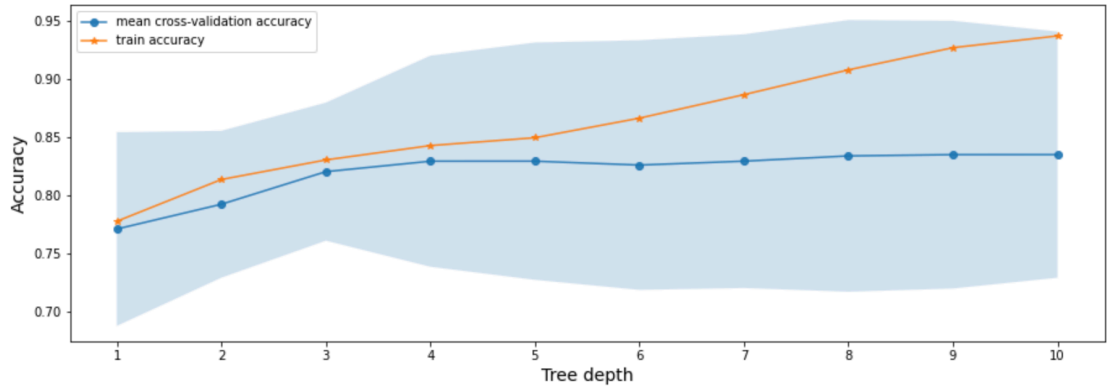

図 10: ランダムフォレストの正確度のプロット

上記の結果より，ランダムフォレストが 4 つの中で最も精度の良い識別器であることが分かった．

次にテストデータの誤り率であるが，test データには Survived 生存フラグが無いため，今回は，train データの Survived 生存フラグと test データを用いて，交差検証法による誤り率を求めることを試みた．誤り率の結果は以下の表 4 に示す．

表 4: テストデータの誤り率

	決定木	バギング	アダブースト	ランダムフォレスト
誤り率	0.390	0.411	0.402	0.390

正しい識別では無い可能性があるが，こちらでもランダムフォレストの精度が比較的良いということが分かる．また，ランダムフォレストは森のサイズによる過学習が発生しないことが分かっているため，さらにサイズを大きくさせれば，精度が良くなると考えられる．

(2) ランダムフォレスト（木の深さの最大値は 4 とした．）を用いて，重要変数のプロットを行った結果を以下の図 11 に示す．（train データを用いた．）

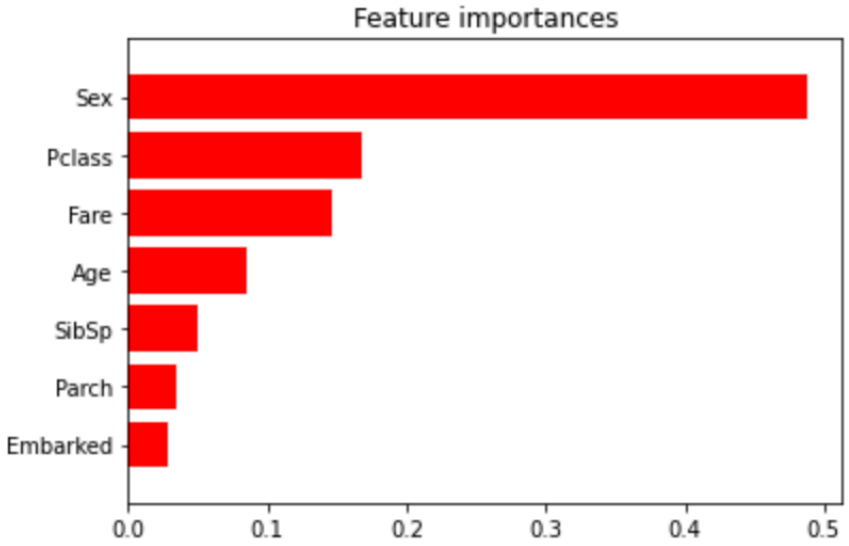

図 11: ランダムフォレストによる重要変数のプロット

図11より，ランダムフォレストでは識別に最も寄与する重要変数は，”Sex”で，次に”Pclass”であることが分かった．

同様に，ブースティング（アダブーストで n_estimators=200）を用いて，重要変数のプロットを行った結果を以下の図 12 に示す．（train データを用いた．）

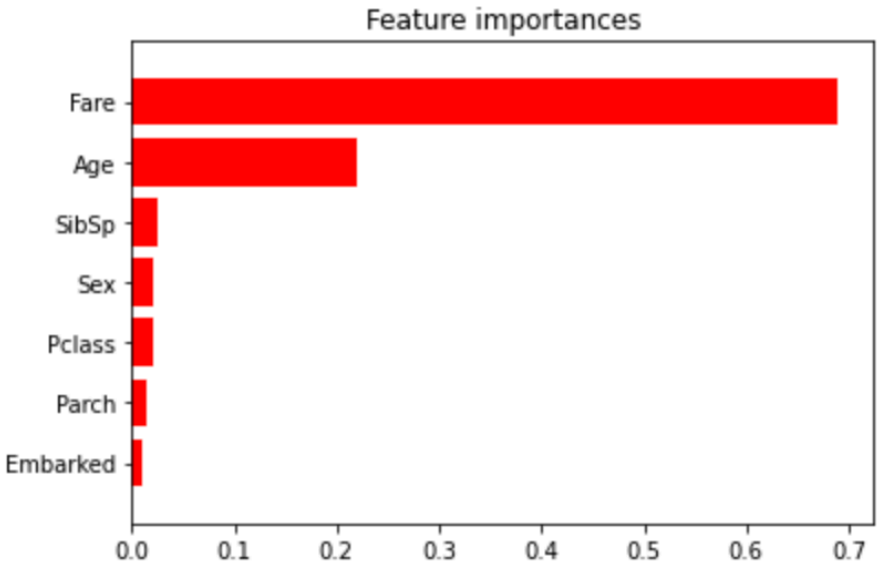

図 12: アダブーストによる重要変数のプロット

図 12 より，アダブーストでは識別に最も寄与する重要変数は，”Fare”で，次に”Age”であることが分かった．

ランダムフォレストとアダブーストで重要変数の結果は異なるが，両者とも考慮すると，最も寄与している変数は，図 11，図 12 より，”Fare”であると考える．（ランダムフォレストでは，3 番目に，アダブーストでは，1 番目に寄与しているという結果であるため．）

参考文献

[1] 主成分分析を Python で理解する
URL：https://qiita.com/maskot1977/items/082557fcda78c4cdb41f
最終閲覧日：2020 年 9 月 1 日

[2] 【Kaggle 初心者入門編】タイタニック号で生き残るのは誰？
URL：https://www.codexa.net/kaggle-titanic-beginner/
最終閲覧日：2020 年 9 月 1 日

[3] クラスタリング手法のクラスタリング
URL：https://qiita.com/suecharo/items/20bad5f0bb2079257568
最終閲覧日：2020 年 9 月 1 日

付録

ソースコード 1: 問 1 のパラメータなどのチューニング

```
from sklearn.datasets import load_wine
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as lda
wine = load_wine()

import pandas as pd
pd.DataFrame(wine.data, columns=wine.feature_names)

X = wine.data
y = wine.target
target_names = wine.target_names

X_tr, X_te, Y_tr, Y_te = train_test_split(X, y, test_size=0.3, random_state
    =2020)

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from sklearn.neural_network import MLPClassifier
import numpy as np


# 誤差逆伝搬法の学習規則 M λ
cv_error = 100
alpha = [0.1, 0.01, 0.001, 0.0001, 0.00001]
cv = 10
scoring='accuracy'

cv_scores_list = []
cv_scores_std = []
cv_scores_mean = []
accuracy_scores = []

for i in range(1, 11):
    print("i = ", i)
```

```
    for j in alpha:
        clf = MLPClassifier(hidden_layer_sizes=(i, ), alpha=j,
                                        activation ='logistic',
                                        solver="adam", random_state=2020,
                                        max_iter=10000)
        clf_fit = clf.fit(X_tr, Y_tr)
        Y_pred = clf.predict(X_te)
        cv_scores = cross_val_score(clf, X_tr, Y_tr, cv=cv, scoring=scoring)
        cv_scores_list.append(cv_scores)
        cv_scores_mean.append(cv_scores.mean())
        cv_scores_std.append(cv_scores.std())
        accuracy_scores.append(accuracy_score(y_true=Y_te, y_pred=Y_pred))
        if cv_error > 1-np.mean(cv_scores):
            cv_error = 1-np.mean(cv_scores)
            cv_score_mean = cv_scores.mean()
            cv_score_std = cv_scores.std()
            error_min = 1-accuracy_score(y_true=Y_te, y_pred=Y_pred)
            best_accuracy = accuracy_score(y_true=Y_te, y_pred=Y_pred)
            alpha_min = j
            hidden_min = i

print("accuracy = ", max(accuracy_scores))
print("正確度 = ", best_accuracy)
print("cv error = ", cv_error)
print("汎化誤差 = ", error_min)
print("hidden_layer_sizes = ", hidden_min)
print("alpha = ", alpha_min)


# サポートベクトルマシン
from sklearn.svm import SVC
import numpy as np


C = [0.01, 0.1, 1, 10, 100, 1000, 10000]

# 線形カーネル
error_min = 100
for c in C:
    model = SVC(C=c, kernel='linear')
    model.fit(X_tr, Y_tr)
    Y_pred = model.predict(X_te)
    if error_min > 1-accuracy_score(Y_te, Y_pred):
        error_min = 1-accuracy_score(Y_te, Y_pred)
        best_accuracy = accuracy_score(y_true=Y_te, y_pred=Y_pred)
        best_C = c
        best_y_pred = Y_pred

print("*** Best ***\n (線形カーネル)\nC:%s" %(best_C))
print('正確度: ', best_accuracy)
print('汎化誤差: ', error_min)


```

```
# RBF カーネル
error_min = 100
sigma= np.array([0.2,0.4,0.8,2,4])
sigma = 1/((sigma**2)*2)

for c in C:
    for sig in sigma:
        model= SVC(C=c, kernel='rbf', gamma=sig)
        model.fit(X_tr, Y_tr)
        Y_pred = model.predict(X_te)
        if error_min > 1-accuracy_score(Y_te, Y_pred):
            error_min = 1-accuracy_score(Y_te, Y_pred)
            best_accuracy = accuracy_score(y_true=Y_te, y_pred=Y_pred)
            best_sigma = sig
            best_C = c
            best_y_pred = Y_pred

print("\n(RBF カーネル)\n C:%s, gamma:%s" %(best_C, best_sigma))
print('正確度: ', best_accuracy)
print('汎化誤差: ', error_min)
```

ソースコード 2: 問 1，問 2 の SVM の識別境界

```
# 2次元に縮小
lda = lda(n_components=2)
Z = lda.fit(X, y).transform(X)
Z1 = Z[y == 0, ]
Z2 = Z[y == 1, ]
Z = np.concatenate([Z1, Z2], axis=0)

from sklearn.svm import SVC
clf= SVC(C=1, kernel='linear')
clf.fit(Z, Y)

from matplotlib.colors import ListedColormap
colors = ['blue', 'red', 'green']

for color, i, target_name in zip(colors, [0, 1], target_names):
    plt.scatter(Z[Y == i, 0], Z[Y == i, 1], alpha=.8,
                label=target_name)
plt.xlabel('LDA1')
plt.ylabel('LDA2')
plt.scatter(Z[clf.support_, 0], Z[clf.support_, 1], alpha=.8, color="black",
                label='support vector')
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('LDA of Wine dataset')
plt.grid()
plt.show()

xmin, xmax = [np.min(Z[:,0]), np.max(Z[:,0])]
ymin, ymax = [np.min(Z[:,1]), np.max(Z[:,1])]
LD = np.linspace(xmin, xmax, 100)
LD1 = (-clf.coef_[0,0]*LD - clf.intercept_)/clf.coef_[0,1]
LD2 = (-clf.coef_[0,0]*LD +1- clf.intercept_)/clf.coef_[0,1]
```

```
LD3 = (-clf.coef_[0,0]*LD -1 - clf.intercept_)/clf.coef_[0,1]

for color, i, target_name in zip(colors, [0, 1], target_names):
    plt.scatter(Z[Y == i, 0], Z[Y == i, 1], alpha=.8,
                label=target_name)
plt.xlabel('LDA1')
plt.ylabel('LDA2')
plt.scatter(Z[clf.support_, 0], Z[clf.support_, 1], alpha=.8, color="black",
                label='support vector')
plt.legend(loc=1, shadow=False, scatterpoints=1)
plt.plot(LD, LD1, color = "red")
plt.plot(LD, LD2, color = "red", linestyle = "dashed")
plt.plot(LD, LD3, color = "red", linestyle = "dashed")
plt.grid()
plt.show()
```

ソースコード 3: 問 2 の主成分分析

```
crab = pd.read_csv("crabs.csv", index_col=0)

# 標準化
crabs = crab.iloc[:, 3:].apply(lambda x: (x-x.mean())/x.std(), axis=0)
crabs.head()

#主成分分析の実行
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(crabs)

# データを主成分空間に写像
feature = pca.transform(crabs)
print(feature.shape)

Y = crab["sex"].map({'M': 0, 'F': 1})

# 寄与率
pd.DataFrame(pca.explained_variance_ratio_, index=["PC{}".format(x + 1) for
    x in range(len(crabs.columns))])

# 累積寄与率を図示する
import matplotlib.ticker as ticker
plt.gca().get_xaxis().set_major_locator(ticker.MaxNLocator(integer=True))
plt.plot([0] + list( np.cumsum(pca.explained_variance_ratio_)), "-o")
plt.xlabel("Number of principal components")
plt.ylabel("Cumulative contribution rate")
plt.grid()
plt.show()

# 2次元に射影したデータのプロット
import matplotlib.pyplot as plt
plt.figure(figsize=(6,6))
colors = ['blue', 'red']
classes = ['Male', 'Female']
for color, i, target_name in zip(colors, [0, 1], Y):
```

```
    scatter = plt.scatter(feature[Y == i, 0],feature[Y == i, 1], label=
        target_name)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.grid()
plt.legend(classes, loc=4)
plt.show()

# デンドログラム
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster

sps = crab.sp.values.tolist()
sexes = crab.sex.values.tolist()
crab_labels = []

for sp, sex in zip(sps, sexes):
    crab_labels.append(sp + sex)

# 階層型クラスタリングの実施
linkage_result = linkage(crab.iloc[:, 3:], method = 'single', metric='
    euclidean')
threshold = 0.4 * np.max(linkage_result[:, 2])

# 階層型クラスタリングの可視化
plt.figure(num=None, figsize=(16, 9), dpi=200, facecolor='w', edgecolor='k
    ')
dendrogram(linkage_result, color_threshold=threshold, labels=crab_labels)
plt.show()
```

ソースコード 4: 問 3 の前処理

```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

# 欠損確認
def kesson_table(df):
    null_val = df.isnull().sum()
    percent = 100 * df.isnull().sum()/len(df)
    kesson_table = pd.concat([null_val, percent], axis=1)
    kesson_table_ren_columns = kesson_table.rename(
    columns = {0 : '欠損数', 1 : '%'})
    return kesson_table_ren_columns

train["Age"] = train["Age"].fillna(train["Age"].median())
train["Embarked"] = train["Embarked"].fillna("S")

train["Sex"][train["Sex"] == "male"] = 0
train["Sex"][train["Sex"] == "female"] = 1
train["Embarked"][train["Embarked"] == "S" ] = 0
train["Embarked"][train["Embarked"] == "C" ] = 1
train["Embarked"][train["Embarked"] == "Q"] = 2

test["Age"] = test["Age"].fillna(test["Age"].median())
test["Sex"][test["Sex"] == "male"] = 0
```

```
test["Sex"][test["Sex"] == "female"] = 1
test["Embarked"][test["Embarked"] == "S"] = 0
test["Embarked"][test["Embarked"] == "C"] = 1
test["Embarked"][test["Embarked"] == "Q"] = 2
test.Fare[152] = test.Fare.median()


features = train[["Pclass","Age","Sex","Fare", "SibSp", "Parch", "Embarked"]].values
features_te = test[["Pclass","Age","Sex","Fare", "SibSp", "Parch", "Embarked"]].values
Y_tr = train['Survived'].values
Y_te = Y_tr[0:418]
```

ソースコード 5: 問 3 の交差検証法による誤り率と正確度のプロット（決定木）

```
from sklearn.model_selection import cross_val_score

def run_cross_validation_on_trees(X, y, tree_depths, cv=10, scoring='accuracy'):
    cv_scores_list = []
    cv_scores_std = []
    cv_scores_mean = []
    accuracy_scores = []
    cv_error = []
    for depth in tree_depths:
        #tree_model = DecisionTreeClassifier(max_depth=depth)
        tree_model = tree.DecisionTreeClassifier(max_depth=depth)
        cv_scores = cross_val_score(tree_model, X, y, cv=cv, scoring=scoring)
        cv_scores_list.append(cv_scores)
        cv_scores_mean.append(cv_scores.mean())
        cv_scores_std.append(cv_scores.std())
        accuracy_scores.append(tree_model.fit(X, y).score(X, y))
        cv_error.append(1-np.mean(cv_scores))
    cv_scores_mean = np.array(cv_scores_mean)
    cv_scores_std = np.array(cv_scores_std)
    accuracy_scores = np.array(accuracy_scores)
    return cv_scores_mean, cv_scores_std, accuracy_scores, cv_error

def plot_cross_validation_on_trees(depths, cv_scores_mean, cv_scores_std, accuracy_scores, title):
    fig, ax = plt.subplots(1,1, figsize=(15,5))
    ax.plot(depths, cv_scores_mean, '-o', label='mean cross-validation accuracy', alpha=0.9)
    ax.fill_between(depths, cv_scores_mean-2*cv_scores_std, cv_scores_mean+2*cv_scores_std, alpha=0.2)
    ax.plot(depths, accuracy_scores, '-*', label='train accuracy', alpha=0.9)
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('Tree depth', fontsize=14)
    ax.set_ylabel('Accuracy', fontsize=14)
    ax.set_xticks(depths)
    ax.legend()

```

```
sm_tree_depths = range(1,11)
sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores, cv_error =
    run_cross_validation_on_trees(features, Y_tr, sm_tree_depths)
print(min(cv_error))

sm_cv_scores_mean, sm_cv_scores_std, sm_accuracy_scores, cv_error =
    run_cross_validation_on_trees(features_te, Y_te, sm_tree_depths)
print(min(cv_error))

import matplotlib.pyplot as plt
plot_cross_validation_on_trees(sm_tree_depths, sm_cv_scores_mean,
    sm_cv_scores_std, sm_accuracy_scores, ' ')
```

ソースコード 6: 問 3 の重要変数

```
# ランダムフォレスト
clf= RandomForestClassifier(max_depth=4, random_state=0)
clf.fit(features, Y_tr)
VIP = clf.feature_importances_
print('Feature Importances:')
names = ["Pclass","Age","Sex","Fare", "SibSp", "Parch", "Embarked"]
for i, feat in enumerate(names):
    print('\t{0:20s} : {1:>.6f}'.format(feat, VIP[i]))

import matplotlib.pyplot as plt
indices = np.argsort(VIP)
plt.figure()
plt.title("Feature importances")
plt.barh(range(features.shape[1]), VIP[indices],color="r", align="center")
names = ["Embarked","Parch","SibSp","Age", "Fare", "Pclass", "Sex"]
plt.yticks(range(features.shape[1]), names)
plt.ylim([-1, features.shape[1]])
plt.show()

# アダブースト
clf = AdaBoostClassifier(n_estimators=200, random_state=2020)
clf.fit(features, Y_tr)
VIP = clf.feature_importances_
print('Feature Importances:')
names = ["Pclass","Age","Sex","Fare", "SibSp", "Parch", "Embarked"]
for i, feat in enumerate(names):
    print('\t{0:20s} : {1:>.6f}'.format(feat, VIP[i]))

import matplotlib.pyplot as plt
indices = np.argsort(VIP)
plt.figure()
plt.title("Feature importances")
plt.barh(range(features.shape[1]), VIP[indices],color="r", align="center")
names = ["Embarked","Parch","Pclass","Sex", "SibSp", "Age", "Fare"]
plt.yticks(range(features.shape[1]), names)
plt.ylim([-1, features.shape[1]])
plt.show()
```