ハロートレーニング
急がば学べ

技能向上訓練
データサイエンス
プログラミングコース

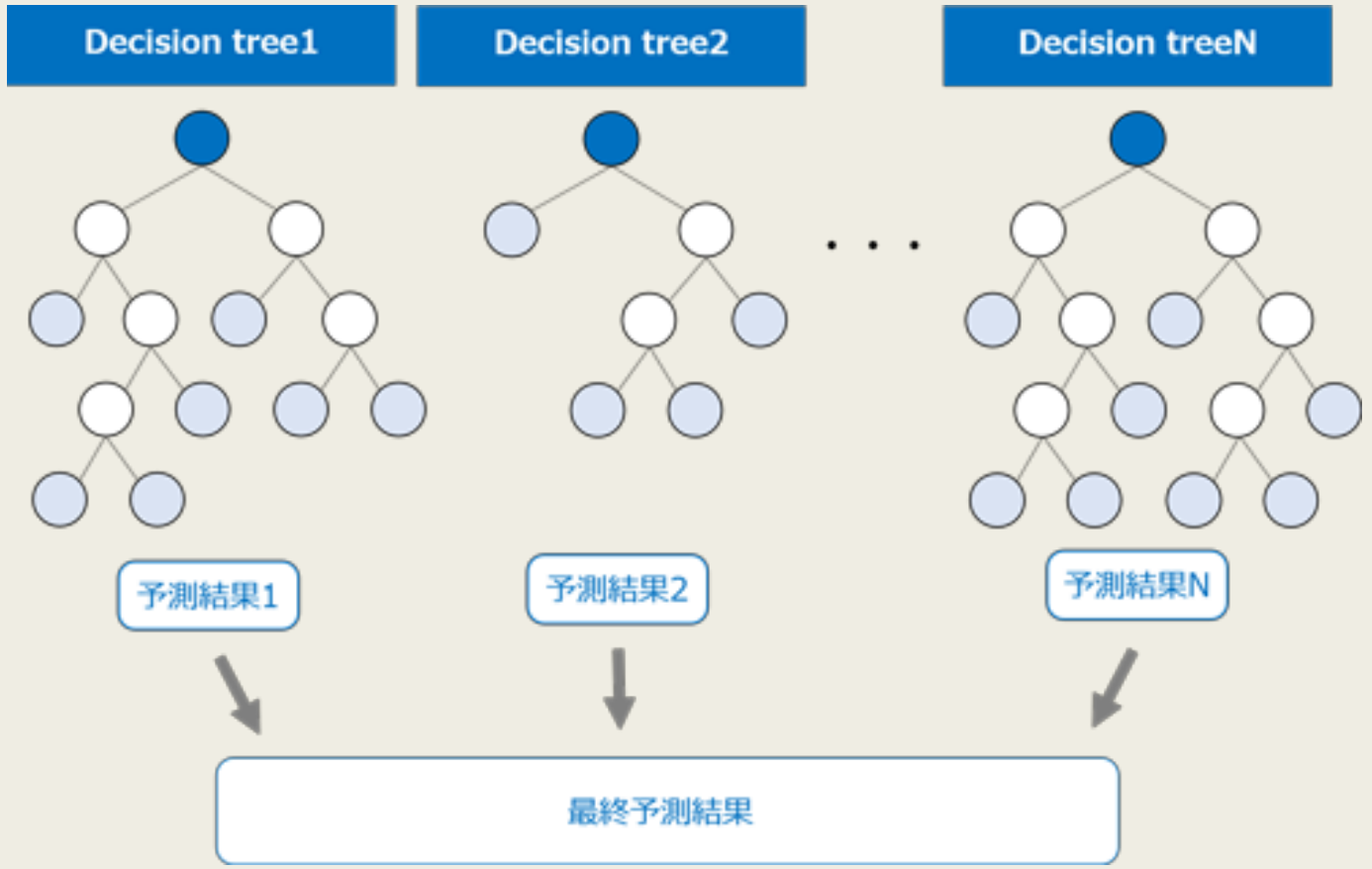
Google Colaboratory

- https://colab.research.google.com/notebooks/intro.ipynb
- クラウド上で実行される、Jupyter Notebook
- Googleアカウントでログインが必要

ランダム
フォレスト

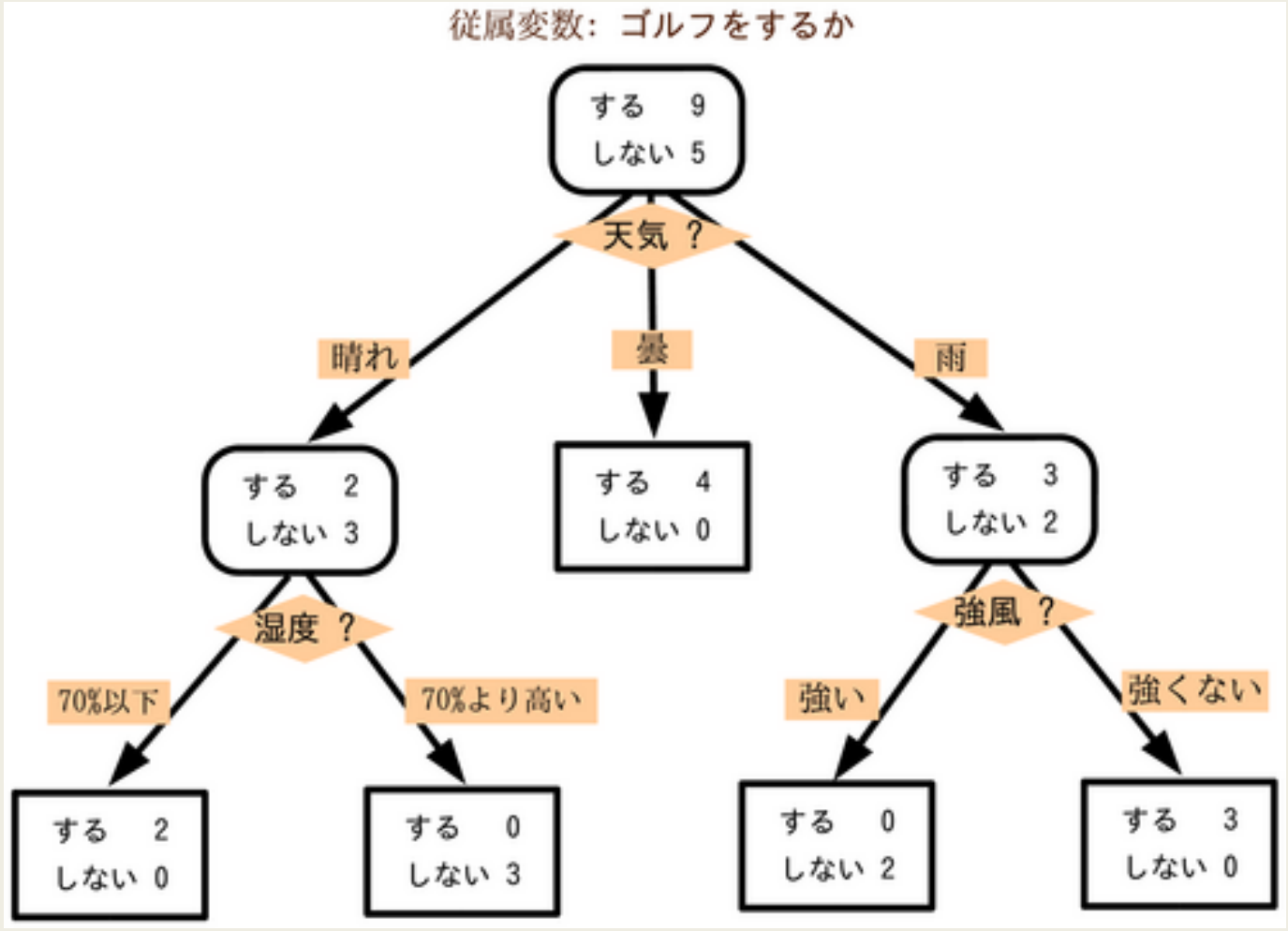
- 複数の決定木を作成し、それぞれに予測をしてもらう。
- 各予測結果をもとに、最終予測結果を決定する。

GoogleDriveより、
「random_forest.ipynb」を
選択してください。



決定木
（ DecisionTree）

- 入力値から、複数の選択を行い、出力内容を決定する。



ノートブックを開く

- GoogleDrive上の「random_forest.ipynb」を開いてください。

ライブラリのインポート

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import *
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
```

- sklearnライブラリの必要なパッケージをインポートする
- また、データフレームを扱うためのpandasもインポートする

GoogleDriveのマウント

データのロード

```
# データのロード
iris_df = pd.read_csv(filepath_or_buffer="/content/drive/My Drive/ensyu3/iris.csv")
display(iris_df.head())
```

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

sepal length (cm)	がく片の長さ
sepal width (cm)	がく片の幅
petal length (cm)	花弁の長さ
petal width (cm)	花弁の幅
Species	種類

- 今回は、アヤメの分類を目的として作られたデータセットを使用。
- アヤメには「Iris-setosa」「 Iris-versicolor」「 Iris-virginica」の３種類がある。

データの分割

```
#目的変数の設定
obj_var = 'Species'

#説明変数と目的変数に分ける
iris_df_X = iris_df.drop(obj_var,axis=1) #説明変数（目的変数以外）
iris_df_y = iris_df[obj_var]             #目的変数

#データを分割
train_X, test_X, train_y, test_y = \
  train_test_split(iris_df_X, iris_df_y, test_size=0.3, stratify = iris_df_y,random_state=0)
```

- 読み込んだデータセットを、目的変数と説明変数に分ける
- 分けたデータを、更に訓練データとテスト用データに分ける

```
#説明変数と目的変数に分ける
iris_df_X = iris_df.drop(obj_var,axis=1) #説明変数（目的変数以外）
iris_df_y = iris_df[obj_var]             #目的変数
```

iris_df

SepalLength	SepalWidth	PetalLength	PetalWidth	Species
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
・・・	・・・	・・・	・・・	・・・
6.5	3	5.2	2	Iris-virginica
6.2	3.4	5.4	2.3	Iris-virginica
5.9	3	5.1	1.8	Iris-virginica

目的変数を削除

iris_df_X

SepalLength	SepalWidth	PetalLength	PetalWidth
5.1	3.5	1.4	0.2
4.9	3	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5	3.6	1.4	0.2
5.4	3.9	1.7	0.4
・・・	・・・	・・・	・・・
6.5	3	5.2	2
6.2	3.4	5.4	2.3
5.9	3	5.1	1.8

目的変数のみ抽出

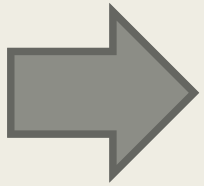
iris_df_y

Species
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
Iris-setosa
・・・
Iris-virginica
Iris-virginica
Iris-virginica

```
#データを分割
train_X, test_X, train_y, test_y = \
  train_test_split(iris_df_X, iris_df_y, test_size=0.3, stratify = iris_df_y,random_state=0)
```

iris_df_X

SepalLength	SepalWidth	PetalLength	PetalWidth
5.1	3.5	1.4	0.2
4.9	3	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5	3.6	1.4	0.2
5.4	3.9	1.7	0.4
・・・	・・・	・・・	・・・
6.5	3	5.2	2
6.2	3.4	5.4	2.3
5.9	3	5.1	1.8



SepalLength	SepalWidth	PetalLength	PetalWidth
5.1	3.5	1.4	0.2
4.6	3.1	1.5	0.2
6.2	3.4	5.4	2.3
5.4	3.9	1.7	0.4
4.9	3	1.4	0.2
6.5	3	5.2	2
・・・	・・・	・・・	・・・

train_X

SepalLength	SepalWidth	PetalLength	PetalWidth
4.7	3.2	1.3	0.2
5	3.6	1.4	0.2
5.9	3	5.1	1.8
・・・	・・・	・・・	・・・

test_X

- 上記の例では、iris_df_Xを分割。iris_df_Yも同様に分割を行っている。
- test_size = 0.3 → 元のデータの３割をテスト用データ、残りを訓練データにする。
- staratify = iris_df_y → 目的変数が偏らないように分配する。

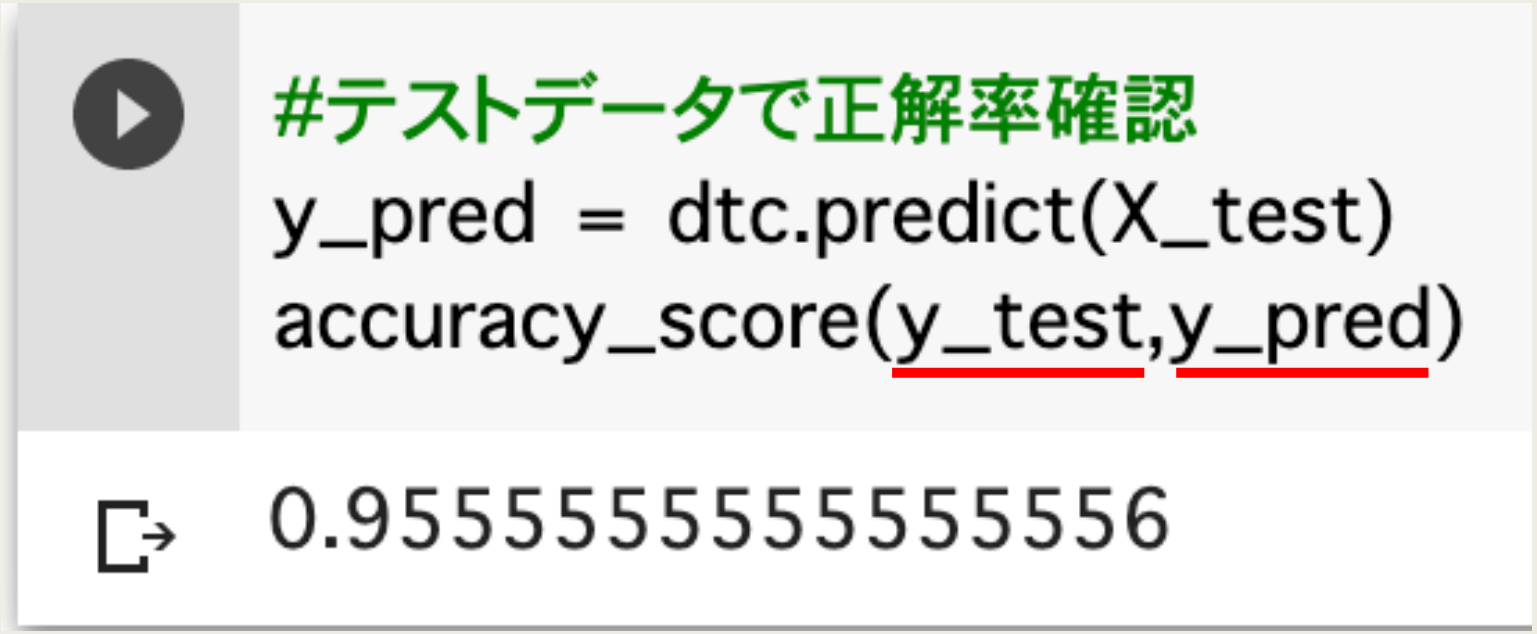
決定木の生成

```
#決定木の生成
dtc = DecisionTreeClassifier()
dtc.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, 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='deprecated',
                       random_state=None, splitter='best')
```

- 訓練データをもとに、決定木を作成する。
- 既にあるライブラリを使用するので、決定木作成のアルゴリズムを組む必要はない。

正解率の表示



- 作成した決定木を使用し、テスト用データでの結果を得る。
- その後、正解データと比較し、正答率を表示する。

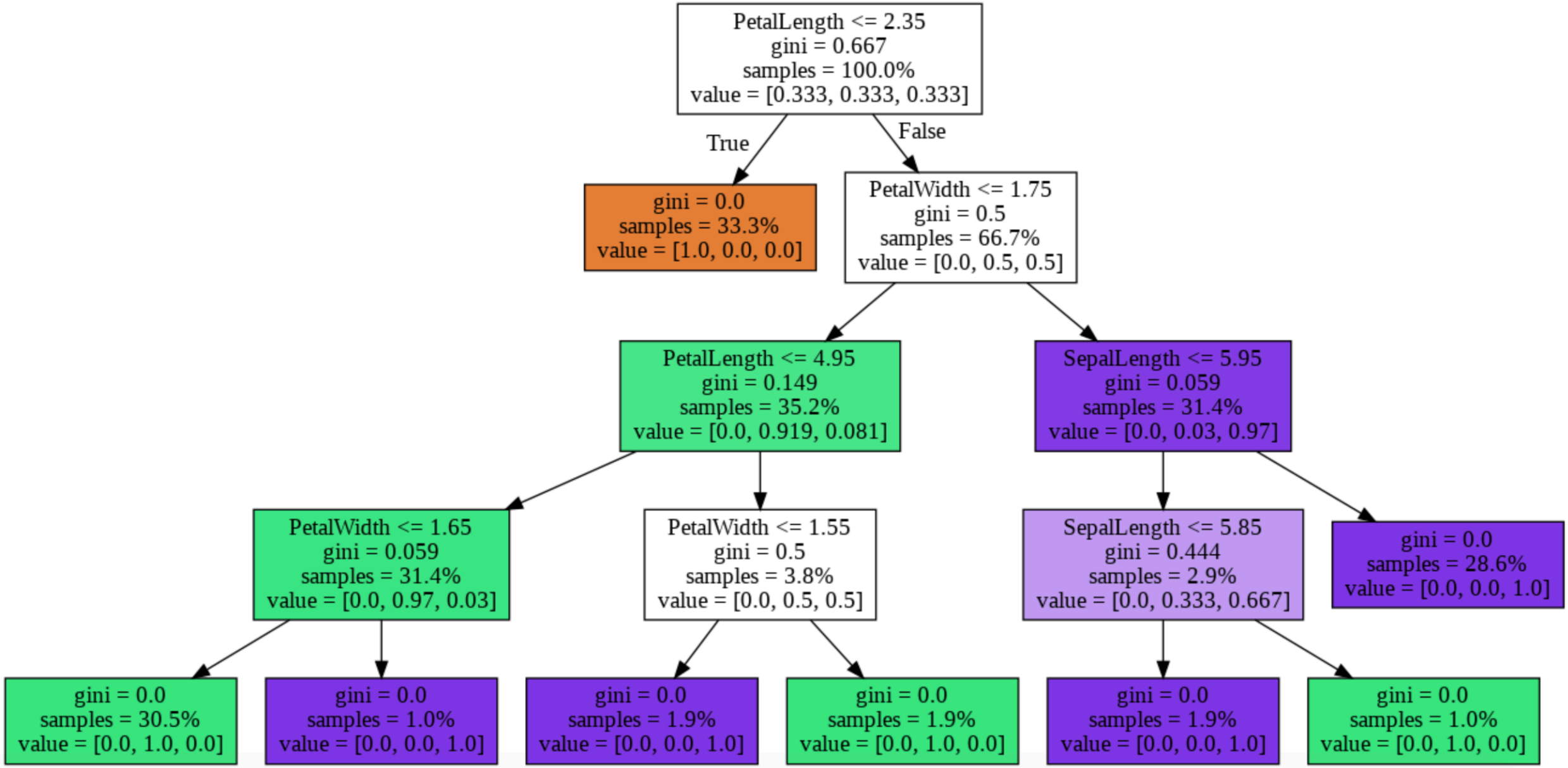
結果の確認

```
#確認
result = pd.DataFrame(test_y)
result["pred"] = pred_y

display(result)
```

- 正解の値と、予測値を並べて表示してみる。

決定木の表示

```
from sklearn import tree
import pydotplus
from IPython.display import Image
from graphviz import Digraph

dot_data = tree.export_graphviz(
    dtc,
    out_file=None,
    feature_names=iris_df_X.columns.values,
    filled=True,
    proportion=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

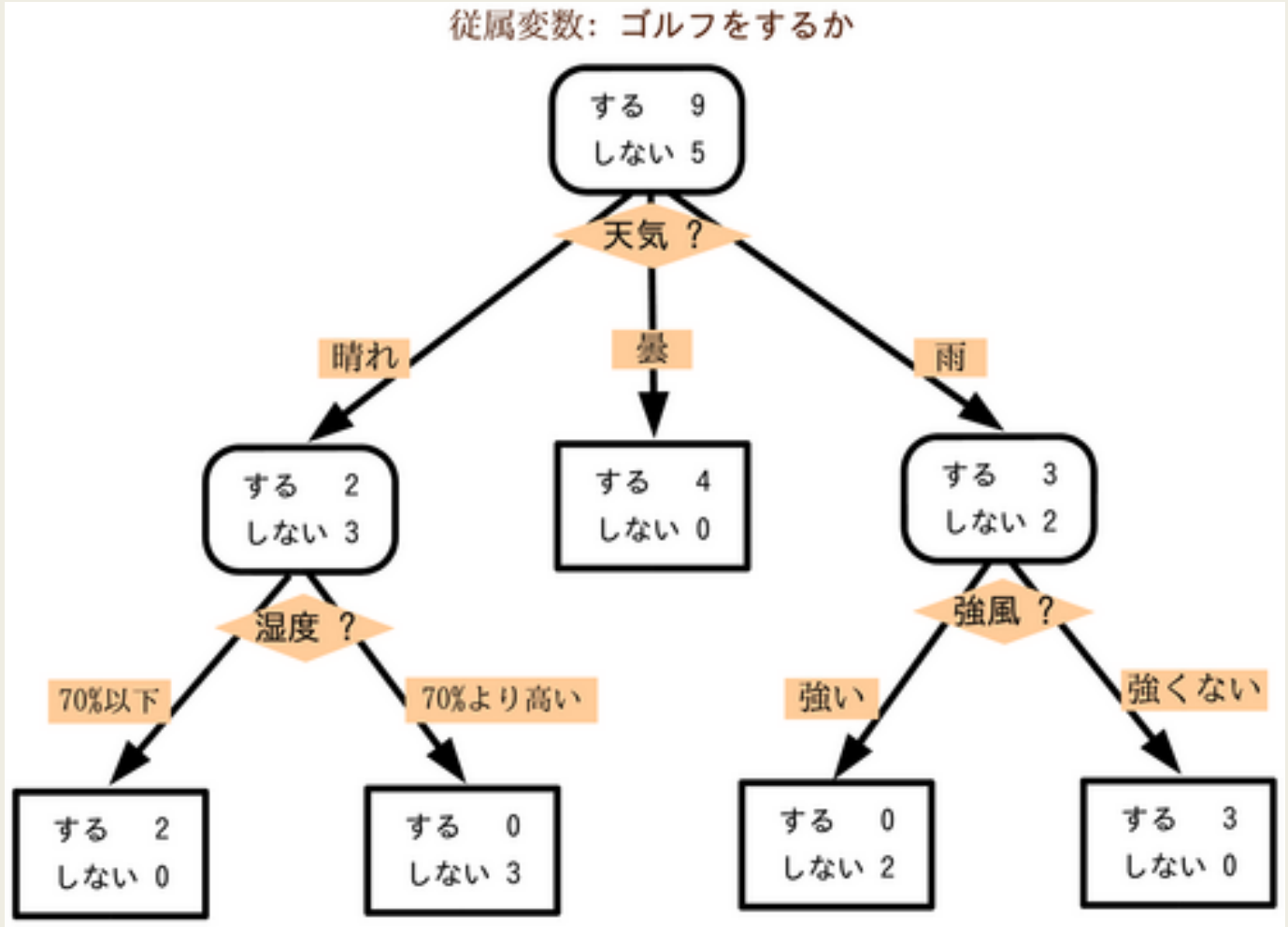


補足：gini = ジニ不純度　　1 - (各クラスの出現割合の２乗の合計)

この値が０なら、単一のクラスしかない。これが小さくなるように、各ノードが生成される。

ランダムフォレストのアルゴリズム

- 訓練データから、いくつかのデータを抜き、決定木を作成する。
- 同じ決定木が出来ないよう、抜くデータは変える。
- 完成した複数の決定木の結果をもとに最終結果を割り出す。



ランダムフォレストを実行し、正答率表示

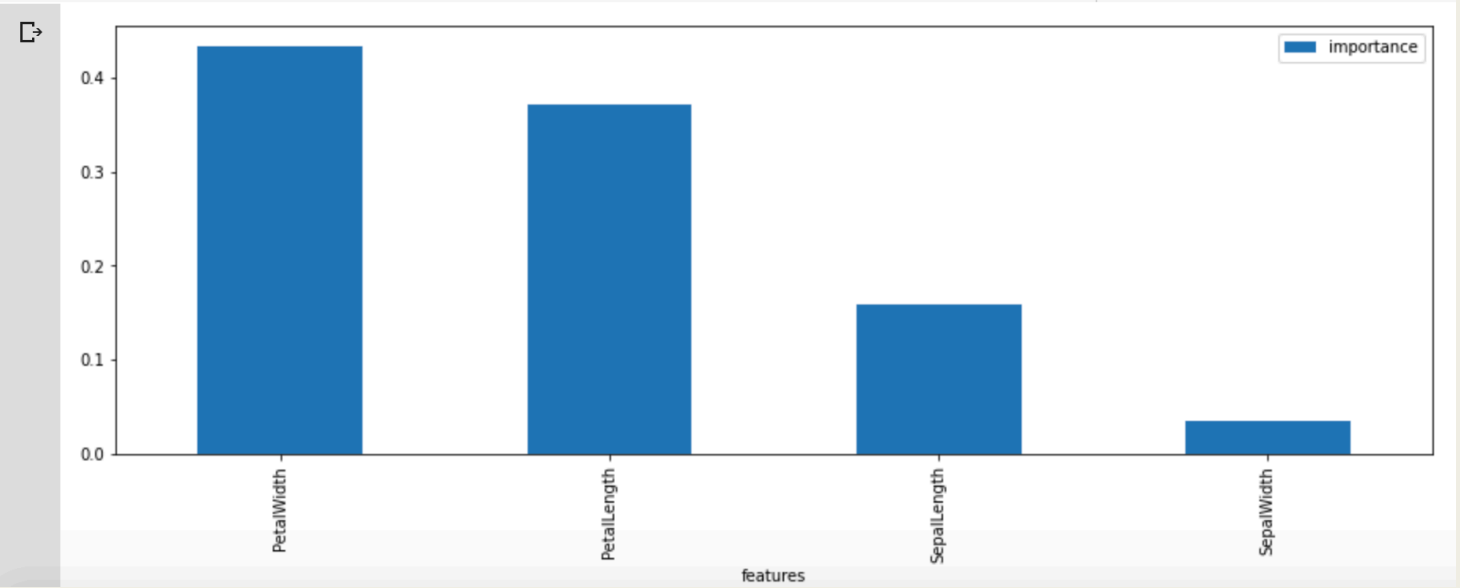
```
# ランダムフォレストモデルを生成
# n_estimators 決定木をいくつ生成するか（デフォルトは10）
clf = RandomForestClassifier(random_state=0, n_estimators=10)
clf = clf.fit(train_X, train_y)

# 結果検証
rdf_y_pred = clf.predict(test_X)
accuracy_score(y_test,rdf_y_pred)
```

```
0.9777777777777777
```

- 正答率が少し増えている。
- 決定木の数を変えて、いろいろ試してみよう。

変数重要度の表示

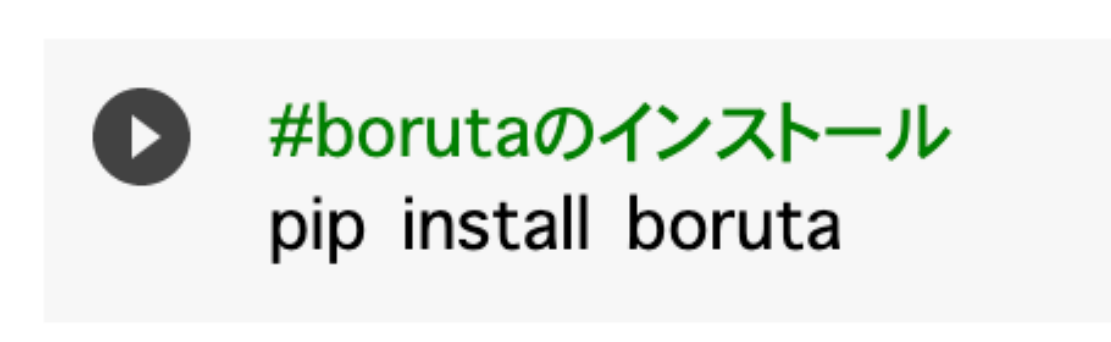
```
import matplotlib.pyplot as plt

#特徴量の重要度を表示
clf = clf.fit(train_X, train_y)

default_imp=pd.DataFrame(data={'importance':clf.feature_importances_,'features':train_X.columns})
default_imp=default_imp.set_index('features')
default_imp=default_imp.sort_values('importance',ascending=False)
default_imp=default_imp[:30]
default_imp.plot(kind='bar',figsize=(15,5))
plt.show()
```



- 推測に影響のある変数ほど、高い数値が出る。

Borutaを使用して、特徴量の抽出

- Borutaとは、必要と思われる特徴量を選択してくれるライブラリです。
- 初期段階でこのライブラリは使用できないので、pipコマンドでインストールをします。



- pipは、Pythonのパッケージ管理ツールで、ライブラリのインストールにも使用できます。

ダミーデータの追加

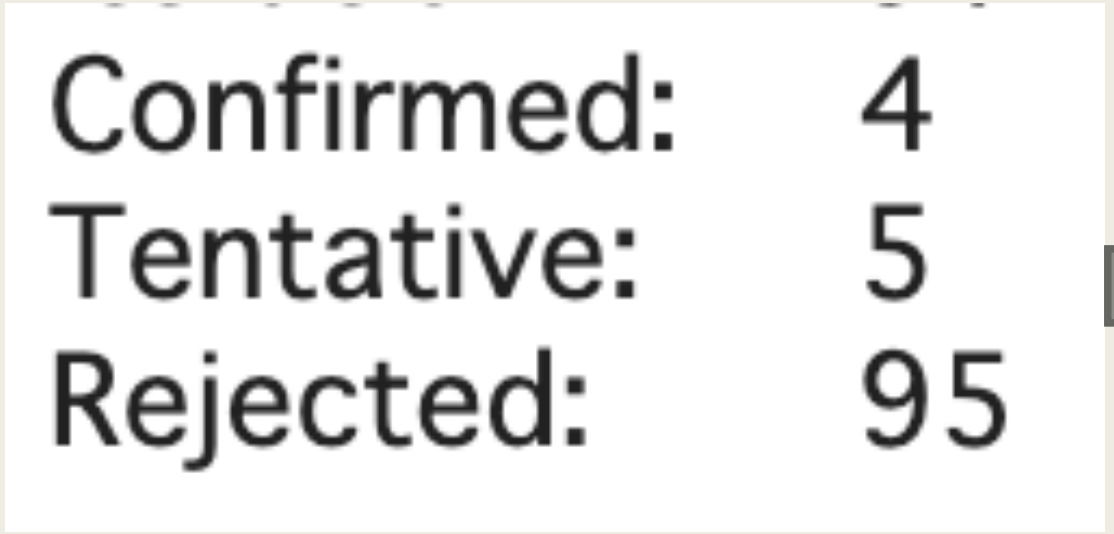
- Borutaの性能を測る前に、アヤメのデータセットにあらかじめダミーデータを追加します。
- Borutaを実行した結果、ダミーデータを除外したら成功です。

```
import numpy as np
#100列分のダミーデータを追加
df_lie = pd.DataFrame(np.random.randint(0,10,size=(150, 100)), columns=['dummy']*100)
iris_df_L = pd.concat([iris_df, df_lie], axis=1)
display(iris_df_L.head())
```

ランダムフォレストアルゴリズムを利用して Borutaの実行

```
#borutaで、特徴量の抽出
from boruta import BorutaPy

rf = RandomForestClassifier(random_state=0, n_estimators=10)

iris_dfL_X = iris_df_L.drop(obj_var,axis=1) #説明変数(目的変数以外)
iris_dfL_y = iris_df_L[obj_var]                #目的変数

(trainL_X, testL_X, trainL_y, testL_y) = train_test_split(
    iris_dfL_X, iris_dfL_y, test_size=0.3, random_state=0)

feat_selector = BorutaPy(rf,
                         n_estimators='auto',  # 特徴量の数に比例して、木の本数を増やす
                         verbose=2, # 0: no output,1: displays iteration number,2: which features have been selected already
                         alpha=0.05, # 有意水準
                         max_iter=10, # 試行回数
                         random_state=1
                        )

# 実行
feat_selector.fit(trainL_X.values, trainL_y.values)
```

結果



- 特徴量数：104
- 重要な特徴量 : 4
- どちらとも言えない:5
- 必要ない : 95

何が選択されたかを確認

```
print(feat_selector.support_)
```

```
[ True  True  True  True False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False False False False False False]
```

- 先頭の４つが選択されている。（True）
- ダミーデータは後ろに追加しているので、正しく選択されいる事がわかる。

選択されたデータのみを抜き出す

```
train_X_selected = trainL_X.iloc[:,feat_selector.support_]
test_X_selected = testL_X.iloc[:,feat_selector.support_]
display(train_X_selected.head())
display(test_X_selected.head())
```

- feat_selector.support_が「True」のデータを抜き出し、新たなデータフレームを作成する。
- 今回は、元々のデータに戻るだけなので、意味はない。

機械故障率

- 機械の故障率を求めるデータセットを使用して、故障予測を行ってみる。
- データダウンロード先
  - https://github.com/IBM/iot-predictive-analytics/blob/master/data/iot_sensor_dataset.csv

	footfall	atemp	selfLR	ClinLR	DoleLR	PID	outpressure	inpressure	temp	fail
1	0	7	7	1	6	6	36	3	1	1
2	190	1	3	3	5	1	20	4	1	0
3	31	7	2	2	6	1	24	6	1	0
4	83	4	3	4	5	1	28	6	1	0
5	640	7	5	6	4	0	68	6	1	0
6	110	3	3	4	6	1	21	4	1	0
7	100	7	5	6	4	1	77	4	1	0
8	31	1	5	4	5	4	21	4	1	0