このデータセットは、実験要件に従い、80%をトレーニング用、残りの20%をテスト用とするランダムな分割を行った。

課題１： 学習データについて、Speciesを目的変数,他の
4変数を説明変数としてCNNモデルを選びました

まずは、畳み込み層を設置しました。
・Conv1D(64, 2)： この畳み込み層が64個の畳み込みカーネルを持ち、それぞれのカーネルのサイズが2であることを示す。
・input_shape=(4, 1)： 入力データが4特徴で、それぞれ1信号であることを示す。
・activation='relu'： ReLU (Rectified Linear Unit) 活性化関数を使用する層であることを示す。
この畳み込み層は、入力データから特徴を抽出するモデルの部分です。64個の畳み込みカーネルは、いわば64種類の畳み込みカーネルを表し、64種類の特徴抽出方法を表しています。各畳み込みカーネルのサイズは2であり、これは特定のケースに適応することもでき、カーネルによって抽出できる特徴の長さを表している。各特徴は1信号であり、入力データが1次元であることを意味します。

畳み込み層は活性化関数ReLUと併用することで、非線形な特徴をよりよく抽出し、データへの適合性を高めることができます。

```
# build CNN model
model = Sequential()
model.add(Conv1D(64, 2, input_shape=(4, 1), activation='relu'))  # convolution
model.add(MaxPool1D(pool_size=2))  # pooling
model.add(Flatten())  # flatten
model.add(Dense(128, activation='relu'))  # fc
model.add(Dropout(0.3))  # dropout
model.add(Dense(num_classes, activation='softmax'))

# model compile
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
model.summary()

# model training
batch_size = 1
epochs = 1000
model = model.fit(x_train, y_train,
                  batch_size=batch_size,
                  epochs=epochs,
                  verbose=2,
                  validation_data=(x_test, y_test))

# draw loss
plt.plot(model.history['loss'])
plt.plot(model.history['val_loss'])
plt.title('model train vs validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```

次には、畳み込み層について：
・MaxPool1D(pool_size=2)： このプーリングレイヤーのプーリングエリアサイズが2であることを示す。
プーリング層は、入力データの空間サイズを縮小する畳み込み層の一部である。 これにより、モデルの計算量を減らし、モデルの学習速度と汎化能力を向上させることができます。

・pool_size=2 は、プーリング領域の大きさが 2 であることを意味します。これにより、最大値で 2 個のデータポイントを保持することができ、データの次元を減らしてモデルの計算量を減らし、かつ最も重要な特徴を保持してモデルの性能を向上させることができます。

このパラメータはケースバイケースで調整できるが、一般にpool_sizeが小さいほどデータの詳細を保持するのに有利であるが、より多くの計算が必要となります。 ここでは、データベースのサイズが小さいため、より詳細な情報を保持することが望まれます。

```
# build CNN model
model = Sequential()
model.add(Conv1D(64, 2, input_shape=(4, 1), activation='relu'))  # convolution
model.add(MaxPool1D(pool_size=2))  # pooling
model.add(Flatten())  # flatten
model.add(Dense(128, activation='relu'))  # fc
model.add(Dropout(0.3))  # dropout
model.add(Dense(num_classes, activation='softmax'))

# model compile
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
model.summary()

# model training
batch_size = 1
epochs = 1000
model = model.fit(x_train, y_train,
                  batch_size=batch_size,
                  epochs=epochs,
                  verbose=2,
                  validation_data=(x_test, y_test))
```

次には、展開した後で、完全連結層（Dense層）を設けました：
・Dense(128)：この完全連結層が128個のニューロンを持つことを示す
・activation='relu'：ReLU（Rectified Linear Unit）活性化関数を使用する層であることを示します。
この完全連結層は、畳み込み層の出力を最終的な分類結果にマッピングするモデルの部分である。128個のニューロンは、いわば128通りの分類結果を次元的に表現している。ReLUは非線形活性化関数で、モデルの非線形能力を向上させ、データの複雑さをよりよくとらえることができる。
128個のニューロンは、特定のケースに適応することが可能な選択である。この例では、虹彩データセットには3つのカテゴリがあるので、最終的な完全連結層は3つのカテゴリそれぞれに対応する3つのニューロンを持つ必要がある。 しかし、一般的な手法として、出力層のニューロン数をカテゴリ数より少し多めに設定することで、学習過程でモデルが重みを調整する余地が生まれ、最終的にデータにうまくフィットするようになる。
また、入力層に多くのニューロンを持つネットワークでは、それに応じて各層のニューロン数が増え、ネットワークが非常に大きくなり、学習や予測に非常に時間がかかることを考慮する必要があります。

```
# build CNN model
model = Sequential()
model.add(Conv1D(64, 2, input_shape=(4, 1), activation='relu'))  # convolution
model.add(MaxPool1D(pool_size=2))  # pooling
model.add(Flatten())  # flatten
model.add(Dense(128, activation='relu'))  # fc
model.add(Dropout(0.3))  # dropout
model.add(Dense(num_classes, activation='softmax'))

# model compile
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
model.summary()

# model training
batch_size = 1
epochs = 1000
model = model.fit(x_train, y_train,
                  batch_size=batch_size,
                  epochs=epochs,
                  verbose=2,
                  validation_data=(x_test, y_test))
```

最後には、ドロップアウト層を設置して、もう一回完全連結層を使います：
ドロップアウトの閾値を0.3に設定することで、以下のようなメリットがあります：

1. オーバーフィッティングの防止：学習過程でニューロンの重みの30%をランダムに0に設定することで、モデルの学習データへの依存度を下げ、新しいデータへの汎化能力を向上させることができます。
2. モデルの汎化性向上：学習中、ネットワークは継続的に学習し、特徴を再結合することで、新しいデータに対してより良い予測を行うことができるようになります。
3. ニューロンの過剰共有の防止：過剰共有は、モデルがある特徴に過度に依存し、他の特徴を無視する原因となる。ドロップアウトはこれを防止する。

- Dense(num_classes)：完全連結層が num_classes 個のニューロンを持つことを示す。
- activation='softmax'：レイヤーがソフトマックス活性化関数を使用することを意味します。

この完全連結層は、モデルの最終層であり、モデルの予測結果を出力する。num_classesニューロンは、分類結果の可能性があるnum_classes、つまりIrisデータセットのクラス数を意味します。

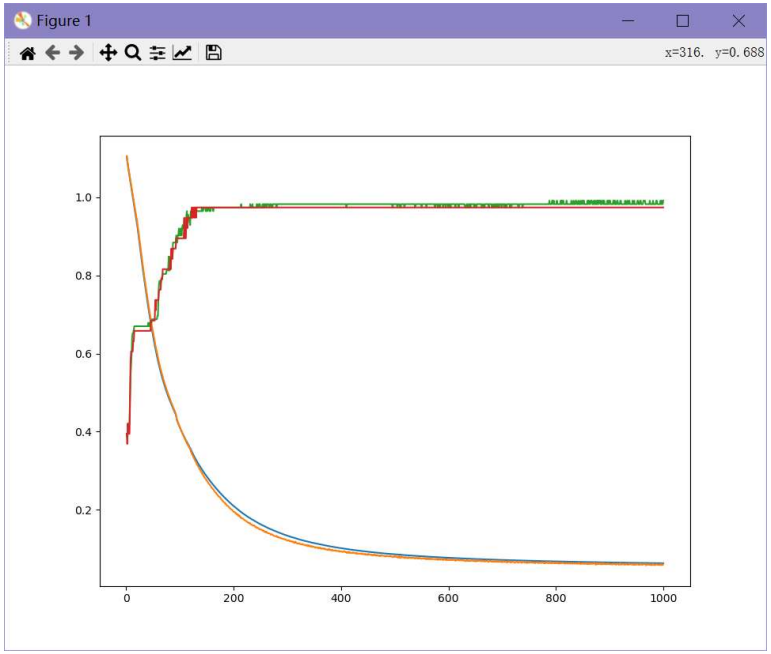
```
# build CNN model
model = Sequential()
model.add(Conv1D(64, 2, input_shape=(4, 1), activation='relu'))  # convolution
model.add(MaxPool1D(pool_size=2))  # pooling
model.add(Flatten())  # flatten
model.add(Dense(128, activation='relu'))  # fc
model.add(Dropout(0.3))  # dropout
model.add(Dense(num_classes, activation='softmax'))

# model compile
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])
model.summary()

# model training
batch_size = 1
epochs = 1000
model = model.fit(x_train, y_train,
                  batch_size=batch_size,
                  epochs=epochs,
                  verbose=2,
                  validation_data=(x_test, y_test))
```

課題２：１000回トレーニングされた後で、（平均）判別精度は以下になります：

```
epoch:  983 train_loss:  0.064 train_accuracy:  0.982 test_loss:  0.058 test_accuracy:  0.974
epoch:  984 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.059 test_accuracy:  0.974
epoch:  985 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.059 test_accuracy:  0.974
epoch:  986 train_loss:  0.063 train_accuracy:  0.991 test_loss:  0.06 test_accuracy:  0.974
epoch:  987 train_loss:  0.063 train_accuracy:  0.991 test_loss:  0.061 test_accuracy:  0.974
epoch:  988 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.059 test_accuracy:  0.974
epoch:  989 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.059 test_accuracy:  0.974
epoch:  990 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.059 test_accuracy:  0.974
epoch:  991 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.059 test_accuracy:  0.974
epoch:  992 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.059 test_accuracy:  0.974
epoch:  993 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.058 test_accuracy:  0.974
epoch:  994 train_loss:  0.063 train_accuracy:  0.991 test_loss:  0.061 test_accuracy:  0.974
epoch:  995 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.059 test_accuracy:  0.974
epoch:  996 train_loss:  0.063 train_accuracy:  0.991 test_loss:  0.061 test_accuracy:  0.974
epoch:  997 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.058 test_accuracy:  0.974
epoch:  998 train_loss:  0.063 train_accuracy:  0.982 test_loss:  0.059 test_accuracy:  0.974
epoch:  999 train_loss:  0.063 train_accuracy:  0.991 test_loss:  0.061 test_accuracy:  0.974
```

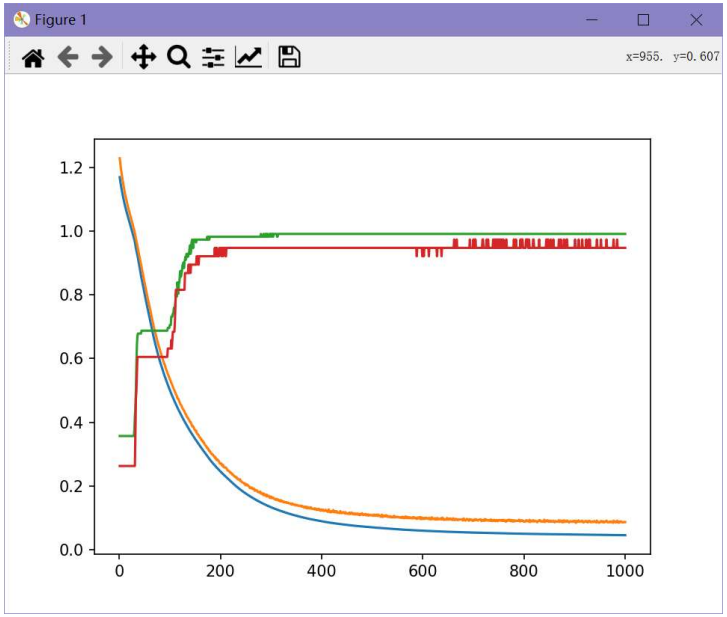


更なる分析結果(他の手法との比較,他のデータへの適用)

今回のCNNモデルと前回ナイーブベイズ分類モデルに比べて、判別制度は向上しました。
しかし、CNNモデルはより多くのメモリを消費し、設計もより複雑になります。

もし、データセットはwineデータセットを利用する場合、より正確な結果も出てきました。

```
epoch:  983 train_loss:  0.047 train_accuracy:  0.991 test_loss:  0.092 test_accuracy:  0.974
epoch:  984 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.09 test_accuracy:  0.947
epoch:  985 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.086 test_accuracy:  0.947
epoch:  986 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.085 test_accuracy:  0.947
epoch:  987 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.088 test_accuracy:  0.947
epoch:  988 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.086 test_accuracy:  0.947
epoch:  989 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.088 test_accuracy:  0.947
epoch:  990 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.089 test_accuracy:  0.947
epoch:  991 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.089 test_accuracy:  0.947
epoch:  992 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.087 test_accuracy:  0.947
epoch:  993 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.087 test_accuracy:  0.947
epoch:  994 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.086 test_accuracy:  0.947
epoch:  995 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.087 test_accuracy:  0.947
epoch:  996 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.087 test_accuracy:  0.947
epoch:  997 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.087 test_accuracy:  0.947
epoch:  998 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.088 test_accuracy:  0.947
epoch:  999 train_loss:  0.046 train_accuracy:  0.991 test_loss:  0.087 test_accuracy:  0.947
```



感想：

(1) ベイジアンネットワークは生成モデル、ニューラルネットワークは識別モデル
(2) ベイジアンネットワークの各点は、現実の意味を持つ確率変数を表し、人間が設計する必要があり、点と点の間のエッジは因果関係を表す、ニューラルネットワークの点は確率変数ではなく、その本当の意味を説明することは困難である。 ニューラルネットワークのポイントはランダム変数ではないので、実際の意味を解釈するのは難しく、いくらでも持つことができ、ポイントとポイントの間のエッジは機能的な関係を表すだけです。
(3) ベイジアンネットワークが解くべき問題は、一般に結果の原因を強制すること、すなわち下流のノードの値を知って上流のノードを反転させることであり、ニューラルネットワークは上流のノードに特徴を入力し、下流のノードを計算することである。