【鈴木教授による解説シリーズ14】集団化する人工知能

だれでも試せる集合知AIプログラミング

2020年２月26日

お伝えしたいポイント

- 集合知AIのプログラミング方法をご紹介します
- Pythonに搭載されているサンプルデータ（アヤメの品種データ）で動作確認します
- AIの意見一致率（コンセンサスレシオ）によって判別性能を強化します

投資信託においては、受益者にとって分かりやすい運用指針が望まれます。そのため本シリーズを通してブラックボックスになりやすいAI技術を可能な限り直感的に理解できるように解説してきました。さらに今回は、だれでも簡単なプログラミングによって集合知AIを試せるように、サンプルプログラムをご紹介します。AIの意見が一致するほど、判別能力が向上することを体験できます。

本稿ではAIを構築するツールとして、主流なPythonを用います。興味はあるけどいまだ使ったことがない方もいらっしゃると思います。ぜひ最初の演習として本稿をご活用いただければ幸いです。データもプログラムも全て無料でご利用いただけます。

本稿で用いるPythonは、Anaconda（https://www.anaconda.com/distribution/）によって環境構築しました。 Pythonのバージョンは現在最新の3.7を用います。

集合知AIの詳細については、下記のバックナンバーをご参照ください。

- **001「AI運用に挑む」**
  https://www.daiwa-am.co.jp/specialreport/quants/2017/quantst_20171207.html
- **012「集合知AIモデルのシミュレーション(前編)」**
  https://www.daiwa-am.co.jp/specialreport/quants/20190805_01.html
- **013「集合知AIモデルのシミュレーション(後編)」**
  https://www.daiwa-am.co.jp/specialreport/quants/20190906_01.pdf

サンプルデータの確認

Pythonを起動したのち、本稿で用いるライブラリーを読み込みます。なお「#」に続く文字は、プログラムとして認識されません。メモ書きとお考えください。

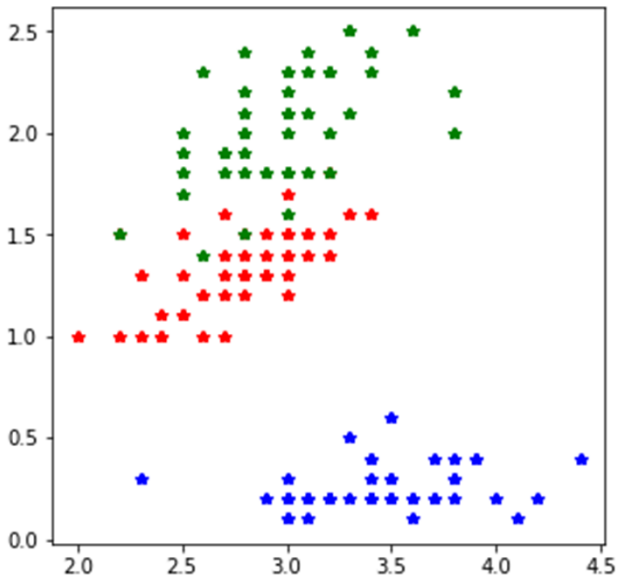
```
# ライブラリーの読込
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics, datasets
from collections import Counter
from copy import copy
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Pythonのサンプルデータ（アヤメの品種データ）を読み込みます。アヤメは3品種に分類でき、その手掛かりとして「がく片の長さ、がく片の幅、花びらの長さ、花びらの幅」を参照します。品種番号 (0, 1, 2) を「目的変数」、参照する手がかりを「説明変数」とします。

```
# サンプルデータの読込
iris = datasets.load_iris()
X = iris.data      # 説明変数
Y = iris.target    # 目的変数
```

まずはデータの様子を見てみましょう。np.where()関数によって()内の条件を満たす要素番号を特定し、プロット点の色を変えています。青色が品種0、赤色が品種1、緑色が品種2に相当します。説明変数は4種類ですが、可視化の都合により「がく片の幅」と「花びらの幅」を用いています。

```
# サンプルデータの可視化
a = np.where( Y==0 )[0]
b = np.where( Y==1 )[0]
c = np.where( Y==2 )[0]

plt.figure( figsize=(5,5) )
plt.plot( X[a, 1], X[a, 3],  'b*' )
plt.plot( X[b, 1], X[b, 3],  'r*' )
plt.plot( X[c, 1], X[c, 3],  'g*' )
```

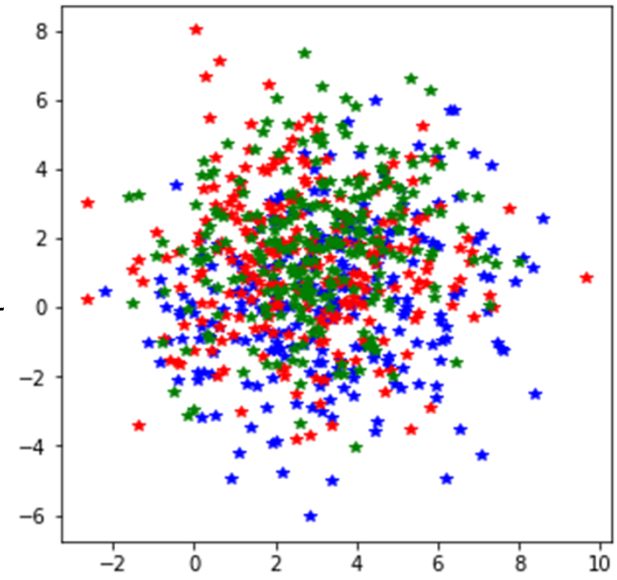


（出所）大和証券投資信託委託

実験用データの作成

前ページの図のように、アヤメの品種は明確な特徴を有しているため、高い精度で品種分類が可能です。しかし金融市場においては、価格変動の特徴は明確ではありません。そこで金融市場を模擬するために、サンプルデータにノイズを加えて特徴を破壊します。なお本稿では、平均0かつ標準偏差2の正規乱数を付加しました。またサンプルデータは150個しかないため、データ量を5倍にしてからノイズを加えました。比較のため可視化すると、品種分類が困難になったことを確認できます。

```
# データ量を5倍に増加
X = np.r_[ X, X, X, X, X ]
Y = np.r_[ Y, Y, Y, Y, Y ]

# ノイズを加えて特徴を破壊
X = X + 2*np.random.randn( np.size(X,0), np.size(X,1) )

# 実験用データの可視化
a = np.where( Y==0 )[0]
b = np.where( Y==1 )[0]
c = np.where( Y==2 )[0]

plt.figure( figsize=(5,5) )
plt.plot( X[a, 1], X[a, 3],  'b*' )
plt.plot( X[b, 1], X[b, 3],  'r*' )
plt.plot( X[c, 1], X[c, 3],  'g*' )
```



（出所）大和証券投資信託委託

予測や判別などの「教師あり機械学習」においては、学習用データとテスト用データに分割して機械学習の性能を評価します。アヤメのサンプルデータの場合、品種ごとにまとまって記録されているため、学習用データ (*_train) とテスト用データ (*_test) を交互に抽出します。

```
# 学習用データ（偶数行: 0, 2, 4, 6, …）
X_train = X[ np.arange( 0, len(X), 2 ) ]
Y_train = Y[ np.arange( 0, len(Y), 2 ) ]

# テスト用データ（奇数行: 1, 3, 5, 7, …）
X_test = X[ np.arange( 1, len(X), 2 ) ]
Y_test = Y[ np.arange( 1, len(Y), 2 ) ]
```

集合知AIの実装例 (1): バギングを用いる場合

さて、ここからが集合知AIプログラミングになります。本稿では2種類の実装例を紹介しますが、まずは「バギング」を用います。集団学習では各AIの独立性が重要なので、学習用データをランダムに復元抽出することでAIの独立性を高めます。詳しくはバックナンバー第12回をご覧ください。

下記プログラム中の「A」と「model」は自由に変更可能です。「A」はAI集団を形成するエージェント数です。数が多いほど集団学習の効果を期待できますが、学習に伴う計算時間が増加します。「model」は各AIの学習に用いるアルゴリズムを指定します。本稿では一例として「k近傍法」を指定します。なおexec()関数は、()内の文字列を実行する関数です。集合知AIプログラミングでは後に全てのAIの意見を参照するため、model_に続く番号を変えることで全ての学習結果を保存しておきます。その際に、copy()関数によってk近傍法の学習結果をコピーします。

```
# [Step1] 学習 (バギング)

A = 500   # エージェント数
model = KNeighborsClassifier( n_neighbors=5 )   # k近傍法

N = len(Y_train)
for i in range(A):

    index = np.random.choice( range(N), N )   # 学習データのランダム復元抽出
    X_train_2 = X_train[ index ]
    Y_train_2 = Y_train[ index ]

    model.fit( X_train_2, Y_train_2 )
    exec( "model_" + str(i) + "= copy(model)" )
```

次に、テスト用データに変えて学習結果を評価します。テスト用の説明変数 (X_test) を与えて、テスト用の目的変数 (Y_test) を判別します。これを全てのAIエージェントについて行い、それぞれの判別結果を「Y_test_predicted」に保存します。なおeval()関数は、 exec()関数と同様に()内の文字列を実行する関数ですが、戻り値を受け取ることができます。どちらも集合知AIプログラミングにおいて重要な関数です。

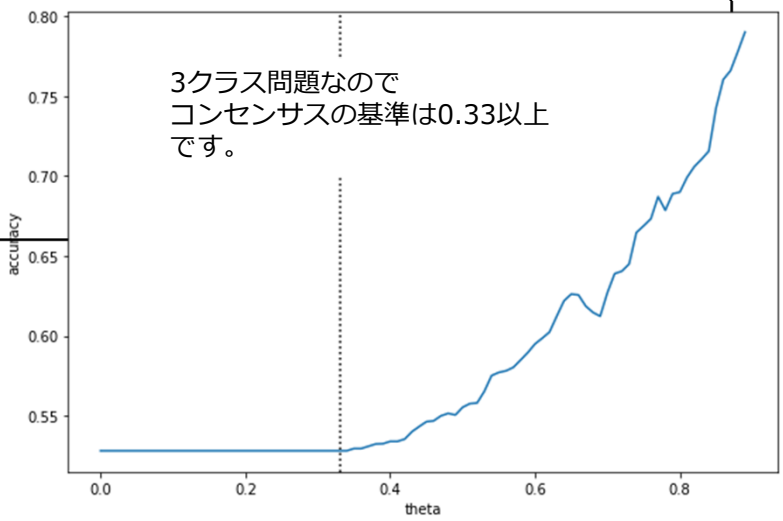
```
# [Step2] テスト

Y_test_predicted = []

for i in range(A):
    tmp = eval( "model_" + str(i) + ".predict( X_test )" )
    Y_test_predicted.append( tmp )
```

テスト用の目的変数 (Y_test) について全AIエージェント (A=500体) の判別結果を集計します。例えば np.array(Y_test_predicted)[:,0] には、Y_test[0]に対する全AIエージェントの判別結果が格納されています。これをopinionsに代入し、most_common()関数によって最頻値を計算しています。最頻値はAI集団としての最終判別結果 (集合知) であり、最頻値を出力した個体数を全エージェント数で割ることで「意見一致率 (コンセンサスレシオ)」を計算します。

```
# [Step3] 意見一致率（コンセンサスレシオ）の算出

Ans = np.zeros( len(Y_test) )   # 「集合知」を記録する配列
Con = np.zeros( len(Y_test) )   # 「意見一致率」を記録する配列

for i in range( len(Y_test) ):
    opinions = np.array( Y_test_predicted )[:,i]
    tmp = Counter(opinions).most_common()    # 最頻値の計算

    Ans[i] = tmp[0][0]                       # [0][0]: 最頻値の抽出 (集合知)
    Con[i] = tmp[0][1] / len(opinions)       # [0][1]: 最頻値の頻度の抽出 (意見一致率)
```

最後に、集合知AIの効果を確認します。判別対象を「意見一致率 (コンセンサスレシオ) が高い時のみ」に限定し、判別精度を評価します。思惑通り、コンセンサスレシオが高い時に限定するほど (横軸)、判別精度を向上できます (縦軸)。AI運用においては、このような性質を利用して、コンセンサスレシオが高い銘柄やタイミングに限定することで運用パフォーマンスの向上を目指します。

```
# [Step4] 判別対象を「コンセンサスレシオ > 閾値theta」に限定

theta = np.arange( 0.0, 0.90, 0.01 )
accuracy = np.zeros( len(theta) )

for x in range( len(theta) ):
    i = np.where( Con > theta[x] )[0]      # コンセンサスレシオ >theta に限定
    accuracy[x] = metrics.accuracy_score( Y_test[i], Ans[i] )        # 判別精度


# 結果の表示
plt.figure( figsize=(9,6) )
plt.plot( theta, accuracy )
plt.xlabel( "theta" )
plt.ylabel( "accuracy" )
plt.axvline( 0.33, ls = ":", color = "k")
```



（出所）大和証券投資信託委託

集合知AIの実装例 (2): ランダムフォレストを用いる場合

説明変数の種類が多いほど、 k近傍法のように2点間距離によって類似度を評価する機械学習は不向きです。そこで説明変数を取捨選択する「決定木」が良く用いられます。先のバギングにおいてmodelを変更するのは非常に簡単です。例えば以下の通りです。その他の変更は不要です。

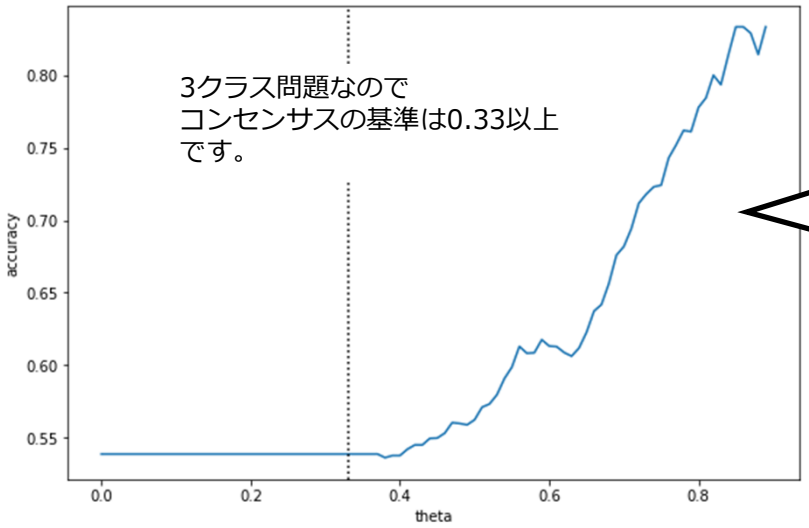
```
   model = KNeighborsClassifier( n_neighbors=5 )      # k近傍法
⇒ model = DecisionTreeClassifier( max_depth=10 )     # 決定木
```

ただ、バギングによって学習データに関する独立性を強化できるものの、さらに説明変数の取捨選択についてもランダム性を取り入れることで更に独立性を強化できます。そのような機械学習法の一例として「ランダムフォレスト」があります。ランダムフォレストも集団学習の一種ですが、Pythonのライブラリーでは集団全体の学習結果のみを保存するため、個別AIの様子が分かりません。そのため、コンセンサスレシオを計算できません。そこでexec()関数を用いることで、個別の学習結果を保存できるようにします。

下記プログラムでは、3体の決定木でランダムフォレストを構成し、これを1つのAIエージェントとしています。 集団を構成するAIエージェント数はA=500のように指定しますので、この3体に特別な意味はありません。なお乱数の種 (random_state) も変えることで、各ランダムフォレストの独立性を高めています。

```
# [Step1] 学習 (ランダムフォレスト)

A = 500   # エージェント数

for i in range(A):

    # ランダムフォレスト (¥は改行のコマンド)
    exec("model_" + str(i) + "= RandomForestClassifier( n_estimators=3, ¥
    max_depth=10, random_state=" + str(i) + ")" )

    exec("model_" + str(i) + ".fit( X_train, Y_train )" )
```

その他のプログラムは変更不要です。[Step2] ～ [Step4]を実行すると、以下のような結果が得られます。乱数を使用しているため結果は多少異なりますが、k近傍法と同様の結果が得られます。つまり、コンセンサスレシオが高い時に限定するほど (横軸)、判別精度を向上できます (縦軸)。なおサンプルデータの説明変数は高々4種類であるため、k近傍法に比べてランダムフォレストの優位性は大きくありません。しかし、もしAI運用において多種類の説明変数を用いるならば、ランダムフォレストに基づいた実装例(2)が効果的だと思います。もしくはディープラーニングのように説明変数を圧縮する方法も考えられます。この場合は実装例(1)の「model」をディープラーニングに変更します。



【集合知AIの狙い】

コンセンサスレシオが高いほど
判別精度が高い

⇩

コンセンサスレシオが高い時のみ
投資判断を行う

（出所）大和証券投資信託委託

鈴木智也（すずきともや）

茨城大学大学院理工学研究科機械システム工学領域教授。理学博士。2017年より大和証券投資信託委託クウォンツ運用部特任主席研究員、2018年よりCollabWiz代表取締役を兼務。国際テクニカルアナリスト連盟検定テクニカルアナリスト（MFTA）および同連盟理事。2005年東京理科大学大学院理学研究科博士課程修了後、東京電機大学工学部助手、同志社大学理工学部専任講師、茨城大学工学部准教授を経て、2016年より現職。

学習用データとテスト用データの作成

```
# ライブラリーの読込
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics, datasets
from collections import Counter
from copy import copy
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

# サンプルデータの読込
iris = datasets.load_iris()
X = iris.data      # 説明変数
Y = iris.target    # 目的変数

# サンプルデータの可視化
a = np.where( Y==0 )[0]
b = np.where( Y==1 )[0]
c = np.where( Y==2 )[0]
plt.figure( figsize=(5,5) )
plt.plot( X[a, 1], X[a, 3],  'b*' )
plt.plot( X[b, 1], X[b, 3],  'r*' )
plt.plot( X[c, 1], X[c, 3],  'g*' )

# データ量を5倍に増加
X = np.r_[ X, X, X, X, X ]
Y = np.r_[ Y, Y, Y, Y, Y ]

# ノイズを加えて特徴を破壊
X = X + 2*np.random.randn( np.size(X,0), np.size(X,1) )

# 実験用データの可視化
a = np.where( Y==0 )[0]
b = np.where( Y==1 )[0]
c = np.where( Y==2 )[0]
plt.figure( figsize=(5,5) )
plt.plot( X[a, 1], X[a, 3],  'b*' )
plt.plot( X[b, 1], X[b, 3],  'r*' )
plt.plot( X[c, 1], X[c, 3],  'g*' )

# 学習用データ（偶数行: 0, 2, 4, 6, …）
X_train = X[ np.arange( 0, len(X), 2 ) ]
Y_train = Y[ np.arange( 0, len(Y), 2 ) ]

# テスト用データ（奇数行: 1, 3, 5, 7, …）
X_test = X[ np.arange( 1, len(X), 2 ) ]
Y_test = Y[ np.arange( 1, len(Y), 2 ) ]
```

集合知AIプログラミング・実装例 (1)

```
# [Step1] 学習 (バギング)
A = 500    # エージェント数
model = KNeighborsClassifier( n_neighbors=5 )   # k近傍法

N = len(Y_train)
for i in range(A):
    index = np.random.choice( range(N), N )   # 学習データのランダム復元抽出
    X_train_2 = X_train[ index ]
    Y_train_2 = Y_train[ index ]
    model.fit( X_train_2, Y_train_2 )
    exec( "model_" + str(i) + "= copy(model)" )


# [Step2] テスト
Y_test_predicted = []
for i in range(A):
    tmp = eval( "model_" + str(i) + ".predict( X_test )" )
    Y_test_predicted.append( tmp )


# [Step3] 意見一致率（コンセンサスレシオ）の算出
Ans = np.zeros( len(Y_test) )   # 「集合知」を記録する配列
Con = np.zeros( len(Y_test) )   # 「意見一致率」を記録する配列
for i in range( len(Y_test) ):
    opinions = np.array( Y_test_predicted )[:,i]
    tmp = Counter(opinions).most_common()    # 最頻値の計算
    Ans[i] = tmp[0][0]                       # [0][0]: 最頻値の抽出 (集合知)
    Con[i] = tmp[0][1] / len(opinions)    # [0][1]: 最頻値の頻度の抽出 (意見一致率)


# [Step4] 判別対象を「コンセンサス>閾値theta」に限定
theta = np.arange( 0.0, 0.90, 0.01 )
accuracy = np.zeros( len(theta) )
for x in range( len(theta) ):
    i = np.where( Con > theta[x] )[0]          # コンセンサス>theta に限定
    accuracy[x] = metrics.accuracy_score( Y_test[i], Ans[i] )     # 判別精度


# 結果の表示
plt.figure( figsize=(10,6) )
plt.plot( theta, accuracy )
plt.xlabel( "theta" )
plt.ylabel( "accuracy" )
plt.axvline( 0.33, ls = ":", color = "k")
```

集合知AIプログラミング・実装例 (2)

```
# [Step1] 学習 (ランダムフォレスト)
A = 500    # エージェント数

for i in range(A):
    # ランダムフォレスト (¥は改行のコマンド)
    exec("model_" + str(i) + "= RandomForestClassifier( n_estimators=3, ¥
    max_depth=10, random_state=" + str(i) + ")" )
    exec("model_" + str(i) + ".fit( X_train, Y_train )" )


# [Step2] テスト
Y_test_predicted = []
for i in range(A):
    tmp = eval( "model_" + str(i) + ".predict( X_test )" )
    Y_test_predicted.append( tmp )


# [Step3] 意見一致率（コンセンサスレシオ）の算出
Ans = np.zeros( len(Y_test) )   # 「集合知」を記録する配列
Con = np.zeros( len(Y_test) )   # 「意見一致率」を記録する配列
for i in range( len(Y_test) ):
    opinions = np.array( Y_test_predicted )[:,i]
    tmp = Counter(opinions).most_common()    # 最頻値の計算
    Ans[i] = tmp[0][0]                       # [0][0]: 最頻値の抽出 (集合知)
    Con[i] = tmp[0][1] / len(opinions)    # [0][1]: 最頻値の頻度の抽出 (意見一致率)


# [Step4] 判別対象を「コンセンサス>閾値theta」に限定
theta = np.arange( 0.0, 0.90, 0.01 )
accuracy = np.zeros( len(theta) )
for x in range( len(theta) ):
    i = np.where( Con > theta[x] )[0]          # コンセンサス>theta に限定
    accuracy[x] = metrics.accuracy_score( Y_test[i], Ans[i] )     # 判別精度


# 結果の表示
plt.figure( figsize=(10,6) )
plt.plot( theta, accuracy )
plt.xlabel( "theta" )
plt.ylabel( "accuracy" )
plt.axvline( 0.33, ls = ":", color = "k")
```

バックナンバー

鈴木教授による解説シリーズ

- **001「AI運用に挑む」**
  https://www.daiwa-am.co.jp/specialreport/quants/2017/quantst_20171207.html
- **002「集団化する人工知能」**
  https://www.daiwa-am.co.jp/specialreport/quants/2018/quantst_20180125.html
- **003「２年目のジンクスを集合知AIで緩和」**
  https://www.daiwa-am.co.jp/specialreport/quants/2018/quantst_20180301.html
- **004「 時系列データの見えない法則をつかむ」**
  https://www.daiwa-am.co.jp/specialreport/quants/2018/quantst_20180409.html
- **005「愚かな人間心理・カモにするAI」**
  https://www.daiwa-am.co.jp/specialreport/quants/2018/quantst_20180501.html
- **006「ナイトメア★アノマリーを狙え」**
  https://www.daiwa-am.co.jp/specialreport/quants/2018/quantst_20180601.html
- **007「ブルーオーシャンAI戦略」**
  https://www.daiwa-am.co.jp/specialreport/quants/2018/quantst_20180703.html
- **008「深層学習による株価予測 (前編)」**
  https://www.daiwa-am.co.jp/specialreport/quants/2018/quantst_20180802.html
- **009「深層学習による株価予測 (後編)」**
  https://www.daiwa-am.co.jp/specialreport/quants/2018/quantst_20181001.html
- **010「ニュースを読んで投資判断する集合知AI」**
  https://www.daiwa-am.co.jp/specialreport/quants/2018/1204.html
- **011「投資理論とコンピュータの歴史」**
  https://www.daiwa-am.co.jp/specialreport/quants/20190701_01.html
- **012「集合知AIモデルのシミュレーション(前編)」**
  https://www.daiwa-am.co.jp/specialreport/quants/20190805_01.html
- **013「集合知AIモデルのシミュレーション(後編)」**
  https://www.daiwa-am.co.jp/specialreport/quants/20190906_01.pdf

佐々木先生による解説シリーズ

- **001「テキストデータからの特徴抽出　ニュースからの単語による特徴表現」**
  https://www.daiwa-am.co.jp/specialreport/quants/20190204.html
- **002「テキストデータからの特徴抽出　単語の頻度から分かるニュース記事の特徴①」**
  https://www.daiwa-am.co.jp/specialreport/quants/20190426.html
- **003「テキストデータからの特徴抽出　単語の頻度から分かるニュース記事の特徴②」**
  https://www.daiwa-am.co.jp/specialreport/quants/20190531_01.html