エンジニア　2018.09.20

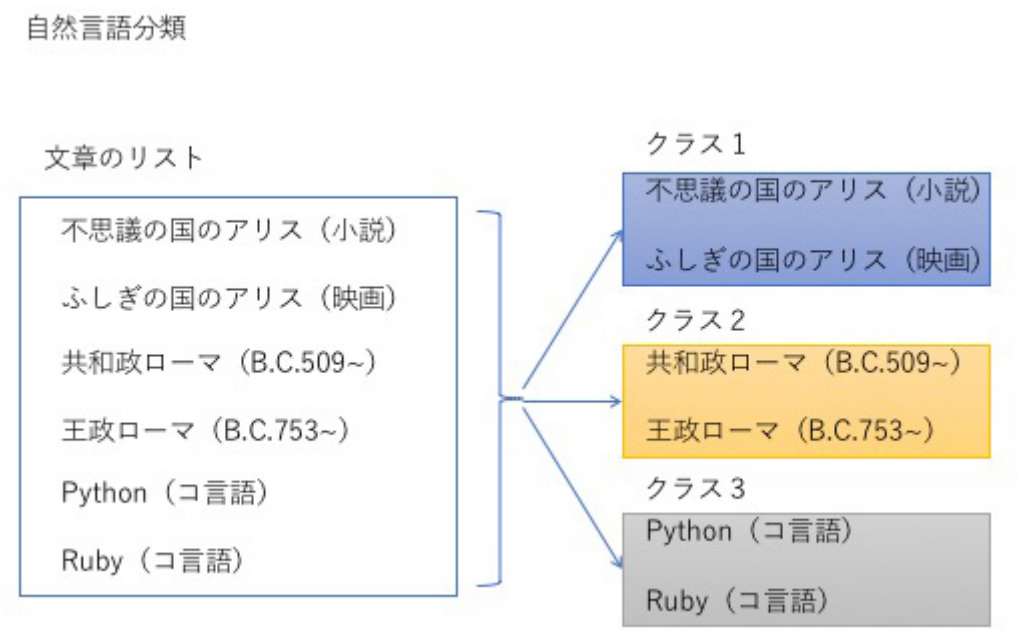
アンサンブル学習による自然言語分類　-後編-

AI戦略室の坂本です。
前回はアンサンブル学習と呼ばれる学習手法について、基本的なロジックをざっくり紹介しました。
今回は実際のプログラムコードを元にして、TF-IDFベクトルとアンサンブル学習による自然言語分類の手法を紹介します。
なお、ここで紹介しているコードは全て、
https://github.com/cocon-ai-group/ensemble-sample
にて公開しています。

目次

データセットの入手

まずはサンプルとして使用する、文章データの入手からです。
ここでは、日本語Wikipediaの文章から、「共和政ローマ」「王政ローマ」「不思議の国のアリス」「ふしぎの国のアリス」「Python」「Ruby」の6つの記事をダウンロードして、その中にある文章を3つのクラスに分類します。



つまり、「共和政ローマ」「不思議の国のアリス」「Python」の記事に含まれている文章から学習して、「王政ローマ」「ふしぎの国のアリス」「Ruby」の記事を正しく、「世界史に関する記事」「不思議の国のアリスに関する記事」「コンピューター言語に関する記事」に分類できるモデルを作成する、という事がお題となります。

そのためのプログラムは、上記GitHub内のget_word_vector.pyにあります。
データのダウンロードと形態素解析に関するコードは解説を省略します。
上記のプログラムで使用されるのは、ダウンロードした文章からTF-IDFベクトルを作成して返す関数で、以下のように定義されています。

```
1
2  def get_vectors():
3      if not (os.path.exists('1.txt') and os.path.exists('2.txt') and os.path.exists('
4              os.path.exists('4.txt') and os.path.exists('5.txt') and os.path.exists('
5          get_files()
6      # 学習データを読み込む
7      clz1txt = open('1.txt').readlines()
8      clz2txt = open('3.txt').readlines()
9      clz3txt = open('5.txt').readlines()
10
11     # 全部繋げる
12     alltxt = []
13     alltxt.extend([s for s in clz1txt])
14     alltxt.extend([s for s in clz2txt])
15     alltxt.extend([s for s in clz3txt])
16
17     # クラスを作成
18     clazz_train = [0] * len(clz1txt) + [1] * len(clz2txt) + [2] * len(clz3txt)
19
20     # TF-IDFベクトル化
```

```
    vectorizer = TfidfVectorizer(use_idf=True, token_pattern='(?u)\\b\\w+\\b')
    vecs_train = vectorizer.fit_transform(alltxt)

    # テスト用データを読み込む
    tst1txt = open('2.txt').readlines()
    tst2txt = open('4.txt').readlines()
    tst3txt = open('6.txt').readlines()
    alltxt = []
    alltxt.extend([s for s in tst1txt])
    alltxt.extend([s for s in tst2txt])
    alltxt.extend([s for s in tst3txt])

    # クラスを作成
    clazz_test = [0] * len(tst1txt) + [1] * len(tst2txt) + [2] * len(tst3txt)

    # TF-IDFベクトル化
    vecs_test = vectorizer.transform(alltxt)

    # 単語を保存
    with open('voc.txt', 'w') as f:
        f.write('\n'.join(vectorizer.vocabulary_.keys()))

    return ((vecs_train,clazz_train), (vecs_test,clazz_test))

```

この関数では、「1.txt」～「6.txt」まで6つのファイルに、形態素解析して分かち書き済みの文章が保存されていることを前提にしています。
そして、ファイルの内容を読み込んだ後、TfidfVectorizerでTF-IDFベクトルを作成し、学習用データのセットと、テスト用データのセットを返します。
また、TfidfVectorizerからベクトルの要素に対応する語彙を取得して、「voc.txt」として保存します。

ランダムフォレスト法による分類

文章がTF-IDFベクトルになれば、機械学習のアルゴリズムを使用してモデルを作成するのは簡単で、ほぼライブラリのAPIを叩くだけです。
まずはScikit-learnにあるランダムフォレスト法を使用するコードから。

```

def get_rf(train, rs=None):
    vecs, clazz = train
    # モデルを学習
    clf = RandomForestClassifier(n_estimators=10, random_state=rs)
    clf.fit(vecs, clazz)
    return clf

```

上記の関数は、ランダムフォレスト法で学習したモデルを返します。
作成したモデルは、以下のようにして使用することが出来ます。

```

train, test = get_vectors()
clf = get_rf(train, rs=1)

# クラス分類を行う
vecs, clazz = test
clz = clf.predict(vecs)

```

分析結果をスコアとして表示するには、Scikit-learnのclassification_report関数を使用します。

```
report = classification_report(clazz, clz, target_names=['class1','class2','class3'])
print(report)
```

以上の内容をまとめた、ランダムフォレスト法による分類を行うコードは、。上記GitHub内のrandomforest_classifier.pyにあります。
このプログラムを実行すると、以下のように分類のスコアが表示されます。

```
$ python3 randomforest_classifier.py
             precision    recall  f1-score   support

      class1       1.00      0.50      0.67        32
      class2       0.86      0.55      0.67        11
      class3       0.62      1.00      0.77        33

avg / total       0.82      0.72      0.71        76
```

なお、この結果は乱数種の指定や元データによってだいぶ変化することに注意してください。
同じコードで実行しても、Wikipediaの文章が変更されているかもしれないため、同一の結果を保証するものではありません。

LightGBMによる分類

次にLightGBMを使用したコードです。
LightGBMではScikit-learnと同じ形式のAPIも用意されていて、そちらの形式を紹介している記事が多いですが、ここではオリジナルのAPIを使用してLightGBMを使用します。

```
import lightgbm as lgb

def get_lgb(train, rs=None):
    vecs, clazz = train
    # モデルを学習
    X_train, X_test, Y_train, Y_test = train_test_split(vecs, clazz, test_size=0.1,
    lgbm_params =  {
        'task': 'train',
        'boosting_type': 'gbdt',
        'objective': 'multiclass',
        'metric': 'multi_logloss',
        'num_class': 3,
        'max_depth': 15,
        'num_leaves': 48,
        'feature_fraction': 1.0,
        'bagging_fraction': 1.0,
        'learning_rate': 0.05,
        'verbose': 0
    }
    lgtrain = lgb.Dataset(X_train, Y_train)
    lgvalid = lgb.Dataset(X_test, Y_test)
    lgb_clf = lgb.train(
        lgbm_params,
        lgtrain,
        num_boost_round=500,
        valid_sets=[lgtrain, lgvalid],
```

```
28          valid_names=['train','valid'],
29          early_stopping_rounds=5,
30          verbose_eval=5
31      )
32
33      return lgb_clf
34
```

APIの使用方法としては難しいところは無く、学習パラメーターをディクショナリで指定してやり、「lgb.Dataset」クラスで学習用データセットを指定してやるくらいです。
ランダムフォレスト法と異なるのは、学習の際に学習用データセットだけでは無く評価用のデータセットも一緒に指定することで、ここで指定した学習用データセットが決定木内のパラメーター学習に使用される一方、評価用のデータセットを使用して過学習を防ぐように動作します。
つまり、LightGBMのアルゴリズムは、評価用のデータセットによるスコアがそれ以上向上しなくなるか最大の学習回数まで、学習用データセットを使用して決定木の分岐を作成してゆきます。
「lgb.train」関数で指定している「num_boost_round」が最大の学習回数で、
「early_stopping_rounds」は、その回数だけ評価用のデータセットによるスコアが連続して悪くなった場合に学習を終了するというパラメーターです。

predictの結果は、全てのクラスに対するスコアとなるので、最大のスコアからなる配列を作成して、最終的な結果とします。

```
1
2  clz = np.argmax(clf.predict(vecs), axis=1)
3
```

以上の内容をまとめた、LightGBMによる分類を行うコードは、。上記GitHub内のlightgbm_classifier.pyにあります。
このプログラムを実行すると、以下のように分類のスコアが表示されます。

```
1
2  $ python3 lightgbm_classifier.py
3               precision    recall  f1-score   support
4
5       class1       0.58      0.66      0.62        32
6       class2       0.44      0.73      0.55        11
7       class3       0.77      0.52      0.62        33
8
9  avg / total       0.65      0.61      0.61        76
10
```

印象として、LightGBMはパラメーターの最適化が必要で、小さいデータセットに対して漫然と使用しても、ランダムフォレスト法と同等以下の結果しか出ない場合があるようです。
一方、大規模なデータに対してチューニングしたパラメーターを指定してやると、非常に良い結果をもたらすので、使用の際にはそれなりの配慮が必要になりそうです。

XGBoostによる分類

次はXGBoostを使用したコードです。
XGBoostによるコードもLightGBMとほぼ同じで、難しいところはありません。

```
1
```

```
import xgboost as xgb

def get_xgb(train, test, rs=None):
    vecs, clazz = train
    # モデルを学習
    X_train, X_test, Y_train, Y_test = train_test_split(vecs, clazz, test_size=0.1,
    xgtrain = xgb.DMatrix(X_train, Y_train)
    xgvalid = xgb.DMatrix(X_test, Y_test)
    xgb_params = {
        'objective': 'multi:softmax',
        'num_class': 3,
        'eta': 0.01,
        'max_depth': 15,
        'max_leaves': 48,
        'silent': True,
        'random_state': rs
    }

    xgb_clf = xgb.train(
        xgb_params,
        xgtrain,
        30,
        [(xgtrain,'train'), (xgvalid,'valid')],
        maximize=False,
        verbose_eval=10,
        early_stopping_rounds=10
    )

    return xgb_clf, xgb.DMatrix(test[0])

```

LightGBMと異なるのは、学習済みのモデルに対してpredictを呼び出すところで、LightGBMではnumpyのndarrayをそのままpredict出来ましたが、ここでは「xgb.DMatrix」クラスのインスタンスである必要があります。
そこで、「get_xgb」関数の最後では、学習済みモデルと一緒に「xgb.DMatrix」クラスのインスタンスにしたテスト用データを返すようにしています。実行時には以下のように使用するようにしました。

```

train, test = get_vectors()
clf, vecs_test = get_cb(train, test, rs=1)

# クラス分類を行う
vecs, clazz = test
clz = np.argmax(clf.predict(vecs_test), axis=1)

```

以上の内容をまとめた、LightGBMによる分類を行うコードは、。上記GitHub内のxgboost_classifier.pyにあります。
このプログラムを実行すると、以下のように分類のスコアが表示されます。

```

$ python3 xgboost_classifier.py
              precision    recall  f1-score   support

      class1       0.85      0.72      0.78        32
      class2       0.80      0.73      0.76        11
      class3       0.72      0.85      0.78        33

avg / total       0.79      0.78      0.78        76

```

CatBoostによる分類

そして問題のCatBoostによる分類です。
実は、ここで使用しているサンプルデータでは、データサイズが小さいため、前回紹介したような次元削減の手法を使用しなくても、そのままCatBoostアルゴリズムを使用することが出来ます。
しかしそれでは、上記のLightGBM・XGBoostと同じことをするだけで面白くないので、ここではあえて次元削減の手法を使用して、小さくて密な行列によるCatBoostの使用方法を紹介することにします。

```
1
2  import catboost as cb
3
4  def get_cb(train, test, use_dense=True, lgb=None, rs=None):
5      # LightGBMで学習
6      if not lgb:
7          lgb = get_lgb(train, rs)
8      # 重要度でソート
9      fi = lgb.feature_importance(importance_type='split')
10     inds = np.argsort(fi)[::-1]
11     # 上位15個の単語を表示
12     with open('voc.txt', 'r') as f:
13         vocs = f.readlines()
14     for i in range(15):
15         print(vocs[fi[inds[i]]].strip())
16
```

まずは入力データから重要度の高いデータを取得するところです。
重要度はLightGBMの学習済みモデルから取得するようにしており、ここでは関数の引数に指定が無ければ、新しく学習を走らせて学習済みモデルを取得するようにしました。
そして重要度の値でソートして、上位15個を表示します。ここで使用しているデータは、文章データから作成したTF-IDFベクトルなので、語彙として保存しておいたファイルを読み込めば、重要な単語を取得することが出来ます。
上のコードが実行されると、以下のように単語のリストが表示されます。

```
1
2  関わら
3  小説
4  反発
5  侵入
6  パトリキ
7  置か
8  pleasance
9  達し
10 国法
11 混乱
12 プリンセス
13 置き換え
14 ロビンスン
15 抱い
16 執筆
17
```

次に、重要度で上位500個の単語のベクトルと、次元削減のアルゴリズム5種によって作成した、それぞれ100次元ずつのベクトル5個を合わせて、合計1000次元のベクトルを作成します。
この、1000次元のベクトルデータが、学習データとなる密なベクトルとなります。

```
# 重要度で上位500個の単語ベクトルを作成
imp_train = train[0][:,fi[inds[0:500]]].toarray()
imp_test = test[0][:,fi[inds[0:500]]].toarray()

# 5個の異なるアルゴリズムで100次元に次元削減したデータ5個
pca = PCA(n_components=100, random_state=rs)
pca_train = pca.fit_transform(train[0].toarray())
pca_test = pca.transform(test[0].toarray())
tsvd = TruncatedSVD(n_components=100, random_state=rs)
tsvd_train = tsvd.fit_transform(train[0])
tsvd_test = tsvd.transform(test[0])
ica = FastICA(n_components=100, random_state=rs)
ica_train = ica.fit_transform(train[0].toarray())
ica_test = ica.transform(test[0].toarray())
grp = GaussianRandomProjection(n_components=100, eps=0.1, random_state=rs)
grp_train = grp.fit_transform(train[0])
grp_test = grp.transform(test[0])
srp = SparseRandomProjection(n_components=100, dense_output=True, random_state=rs)
srp_train = srp.fit_transform(train[0])
srp_test = srp.transform(test[0])

# 合計1000次元のデータにする
vecs_train = np.hstack([imp_train, pca_train, tsvd_train, ica_train, grp_train, srp_
vecs_test = np.hstack([imp_test, pca_test, tsvd_test, ica_test, grp_test, srp_test])

```

最後の機械学習部分はXGBoostの時とほぼ同じで、異なっているのはパラメーターの指定と、データの指定に「cb.Pool」クラスを使用しているくらいです。

```
# モデルを学習
clazz_train = train[1]
X_train, X_test, Y_train, Y_test = train_test_split(vecs_train, clazz_train, test_si
cb_clf = cb.train(cb.Pool(X_train, label=Y_train),
    eval_set=cb.Pool(X_test, label=Y_test),
    params={'loss_function':'MultiClass',
            'classes_count':3,
            'eval_metric':'F1',
            'iterations':10,
            'learning_rate':0.1,
            'classes_count':3,
            'depth':4,
            'random_seed':rs})

return cb_clf, vecs_test

```

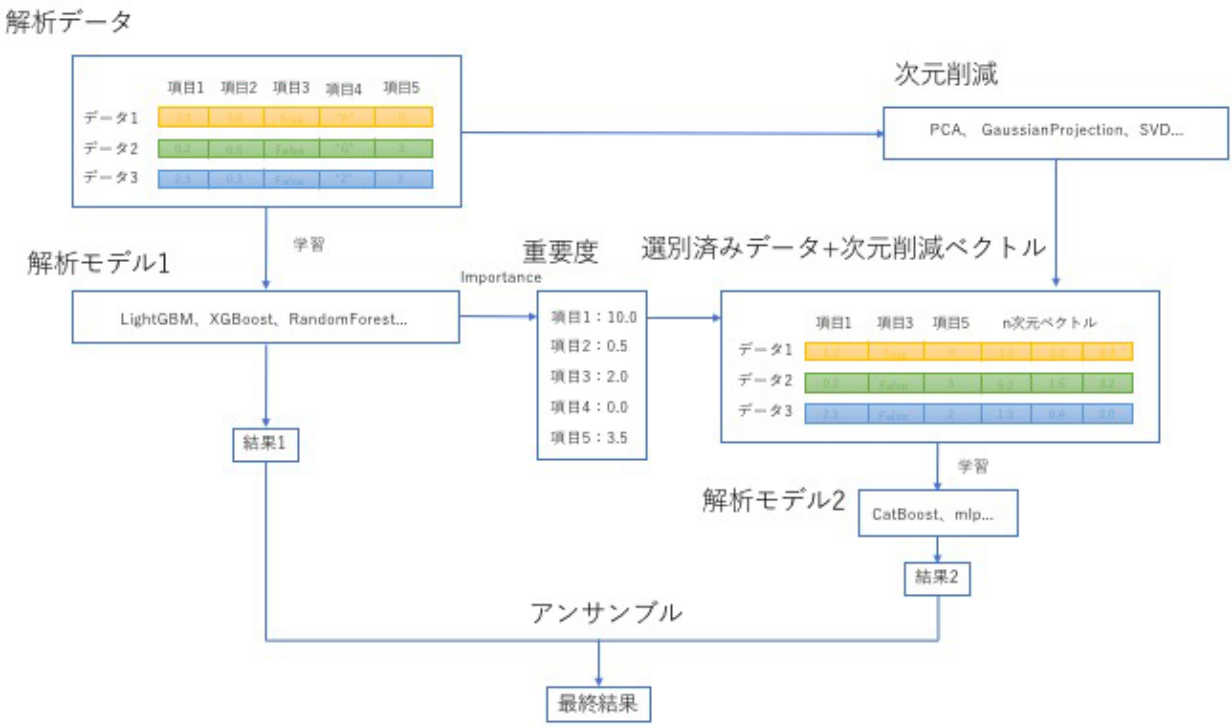
以上の内容をまとめた、CatBoostによる分類を行うコードは、。上記GitHub内の
catboost_classifier.py
関わらにあります。
このプログラムを実行すると、以下のように分類のスコアが表示されます。

```

$ python3 xgboost_classifier.py
              precision    recall  f1-score   support

      class1       0.90      0.56      0.69        32
      class2       0.39      0.82      0.53        11
      class3       0.91      0.91      0.91        33

avg / total       0.83      0.75      0.76        76

```

前述の通り、ここで使用しているサンプルデータは、データサイズが小さいため特に次元削減の手法を使用せずともCatBoostアルゴリズムを適用可能です。
ここでは例として次元削減を使用しましたが、使用しない場合の方がCatBoost単体でのスコアは高くなります。

アンサンブルによる分類

そしていよいよアンサンブル学習です。
アンサンブル学習では、これまで紹介した全てのアルゴリズムを使用して学習を行い、その結果から多数決を取って最終的な結果とします。
前回の記事にも載せたアンサンブル学習のロジック図をもう一度掲載するので、ソースコードと見比べてみてください。



まずは、アンサンブル学習を行う部分のコードです。

```
1
2  from get_word_vector import get_vectors
3  from randomforest_classifier import get_rf
4  from lightgbm_classifier import get_lgb
5  from xgboost_classifier import get_xgb
6  from catboost_classifier import get_cb
7
8  # 多数決を行う関数
9  def get_one(bin):
10     return np.argmax(np.bincount(bin))
11
12 # アンサンブル学習
13 def ensemble(train, test, rs=1):
14     rf_clf = get_rf(train, rs=rs)
15     lgb_clf = get_lgb(train, rs=rs)
16     xgb_clf, xgb_test = get_xgb(train, test, rs=rs)
17     cb_clf, vecs_test = get_cb(train, test, use_dense=True, lgb=lgb_clf, rs=rs)
18
```

```
    # クラス分類を行う
    vecs, clazz = test
    clz1 = rf_clf.predict(vecs)
    clz2 = np.argmax(lgb_clf.predict(vecs), axis=1)
    clz3 = xgb_clf.predict(xgb_test)
    clz4 = np.argmax(cb_clf.predict(vecs_test), axis=1)
    clz = [get_one([clz1[i],clz2[i],clz3[i]]) for i in range(len(clz1))]
    return clz

```

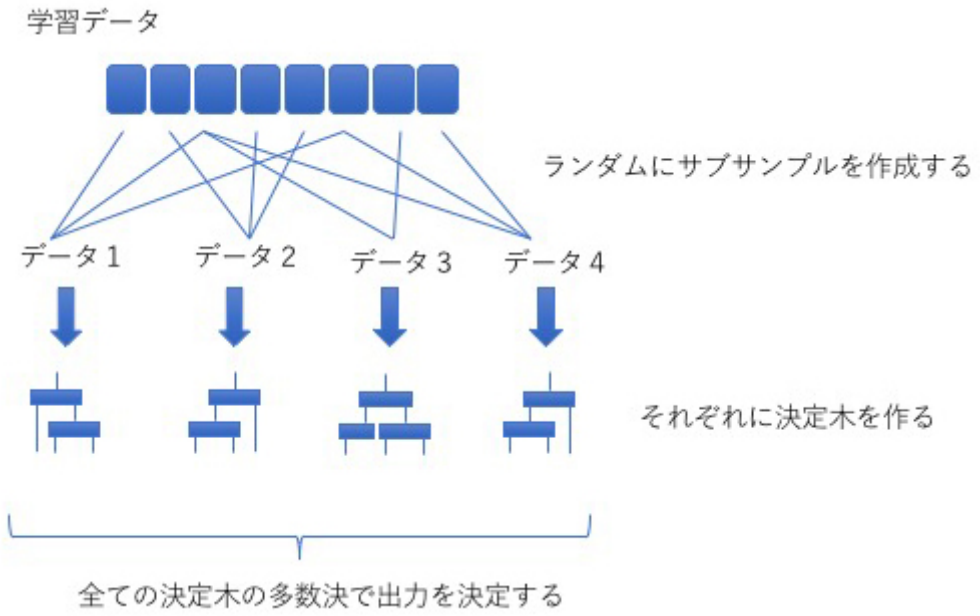
多数決を取る関数として「get_one」を定義し、4つのアルゴリズムで学習した結果を、最終的に一つの結果のリストにしています。
そして、そのアンサンブル学習を呼び出す部分です。

```

if __name__ == '__main__':
    train, test = get_vectors()
    clazz = test[1]
    # アンサンブル学習1回
    print('ensemble 1:')
    clz = ensemble(train, test, rs=1)
    report = classification_report(clazz, clz, target_names=['class1','class2','clas
    print(report)

```

上のコードでは、4つのアルゴリズムからなるアンサンブル学習を一回だけ実行し、その結果のスコアを表示しています。
上のコードが実行されると、以下のようにスコアの値が表示されます。

```

             precision    recall  f1-score   support

     class1       0.81      0.66      0.72        32
     class2       0.78      0.64      0.70        11
     class3       0.71      0.88      0.78        33

avg / total       0.76      0.75      0.75        76

```

これだけだとあまり良いスコアにはなっておらず、アンサンブル学習のメリットが感じられないかもしれません。
そこで今度は、乱数種を変えながら複数回アンサンブル学習を行い、それらの結果からさらに多数決を取るようにします。
これは、今回使用したアルゴリズムでは、学習用データセットと評価用のデータセットを分けて使用するものが多く、データセットの分割によって結果が大きく変わる場合があるためです。
ランダムフォレスト法ではサブサンプルの作成と結果の結合が主な原理でしたが、それと同じ事で、これはいわばアンサンブル学習のランダムなフォレストとも言えます。



また、乱数種を変えて複数回実行する方法の他に、FKold等で分割した複数回の学習を行う事もあります。

```
# 乱数列を変更しながらアンサンブル学習を繰り返す
print('random ensemble:')
clazzes = []
random_seeds = [1,3,7,9,13,17]
for rs in random_seeds:
    clz = ensemble(train, test, rs=rs)
    clazzes.append(clz)
clz = [get_one([clazzes[j][i] for j in range(len(clazzes))]) for i in range(len(claz
report = classification_report(clazz, clz, target_names=['class1','class2','class3']
print(report)
```

上記のコードが実行されると、以下のように結果のスコアが表示されます。

```
              precision    recall  f1-score   support

      class1       0.84      0.84      0.84        32
      class2       0.70      0.64      0.67        11
      class3       0.82      0.85      0.84        33

 avg / total       0.81      0.82      0.81        76
```

今度は、F1スコアを見ると、アンサンブル学習で使用したランダムフォレスト法・LightGBM・XGBoost・CatBoostそれぞれ単体でのどのスコアよりも、良いスコアが出てきました。
単体で最も良いモデルが最上で、悪いモデルに引きずられてスコアが下がるかと思いきや、面白いことにアンサンブルを取ることで全体としてより優れたモデルが作成されている訳です。

まとめ

以上、前後二回にわたってアンサンブル学習の手法を紹介してきました。
アンサンブル学習はディープラーニングと同じく機械学習の手法の一つですが、回帰分析やクラス分

類においては多くの場合ディープラーニングよりも良い結果を出すことが出来て、しかも学習時間などの面で優位性もあります。
アンサンブル学習では、それぞれの学習アルゴリズムについてパラメーターのチューニングが必要になるなど、適切なモデルの作成に手間がかかるのが難点ではありますが、少なくとも回帰分析やクラス分類などの問題に対しては、現状で最高に近い結果をもたらしうる解析手法と言えるのではないでしょうか。

イベント 広報 2019.05.22

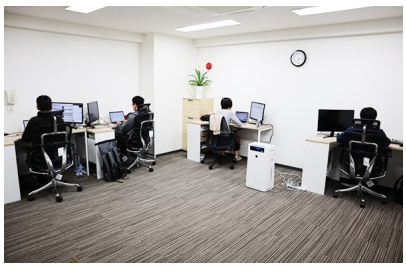
PHDays 9 現地レポート

広報の馬場です。 現在、ロシア・モスクワで開催されている情報セキュリティカンファレンス「Positive Hac…

イベント エンジニア 2019.05.17

ペパコンナイトを開催しました

2019年5月13日（月）にペパボ研究所とココン技術研究室との共同研究成果の発表会であるペパコンナイトを開…



インタビュー エンジニア 働き方 2019.05.16

イエラエ福岡オフィスで働くエンジニアインタビュー

広報の馬場です。 イエラエセキュリティでは、事業拡大に伴い2019年2月より第4の拠点として福岡にオフィス…