


```
[37]: if run_lda==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f} seconds'.format(tm_feature_test_improved))

    [train_time, lda_model] = train_lda(feature_train, label_train,solver='eigen', shrinkage=1, n_comp
onents=1)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(lda_model,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,lda_model)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series(['Feature Extraction Train Time',tm_feature_train_improved,
                    'Feature Extraction Test Time',tm_feature_test_improved,
                    'Train Time',train_time,
                    'Prediction Time',prediction_time,
                    'Accuracy',accuracy,
                    'AUC',auc,
                    'Balanced Accuracy',balanced_accuracy,name='LDA'])
    model_results_df = model_results_df.append(row)

Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049 seconds

Training time: 5.042372 seconds
Prediction time: 0.019037 seconds

Accuracy: 0.821667
Balanced Accuracy: 0.636339
AUC: 0.786203
```

Logistic Model

```
In [38]: #grid='C0':[0.001,0.01,0.1,0.1,0.25,0.5,1,10]

#cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
# gscv = GridSearchCV(LogisticRegression(dual=False, fit_intercept=True,
#                                     intercept_scaling=1, max_iter=100000,
#                                     multi_class='multinomial', penalty='l2',
#                                     solver='lbfgs', tol=0.0001,grid,cv=cv,return_train_score=True)
# gscv.fit(feature_train,label_train)
# gscv.best_params_
# output: {'C': 0.01}

In [39]: if run_logistic==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f} seconds'.format(tm_feature_test_improved))

    weights = (0:1.0, 1:1.0)
    [train_time,lr_model] = train_logistic(feature_train,label_train,
                                           C=0.001, dual=False, fit_intercept=True,
                                           intercept_scaling=1, max_iter=100000,
                                           multi_class='multinomial', penalty='l2',
                                           solver='lbfgs', tol=0.0001,class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(lr_model,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,lr_model)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series(['Feature Extraction Train Time',tm_feature_train_improved,
                    'Feature Extraction Test Time',tm_feature_test_improved,
                    'Train Time',train_time,
                    'Prediction Time',prediction_time,
                    'Accuracy',accuracy,
                    'AUC',auc,
                    'Balanced Accuracy',balanced_accuracy,name='Logistic Regression'])
    model_results_df = model_results_df.append(row)

Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049 seconds

Training time: 97.020723 seconds
Prediction time: 0.022679 seconds

Accuracy: 0.821667
Balanced Accuracy: 0.699697
AUC: 0.822310
```

Weighted Logistic Model

```
In [40]: #grid='C0':[0.001,0.01,0.01,0.1,0.25,0.5,1,10]

#cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
# gscv = GridSearchCV(LogisticRegression(dual=False, fit_intercept=True,
#                                     intercept_scaling=1, max_iter=100000,
#                                     multi_class='multinomial', penalty='l2',
#                                     solver='lbfgs', tol=0.0001,class_weight=weights),grid,cv=cv,return_train_score=Tru
e)
# gscv.fit(feature_train,label_train)
# gscv.best_params_
# output: {'C': 0.001}

In [41]: if run_weighted_logistic==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f} seconds'.format(tm_feature_test_improved))

    weights = (0:80.0, 1:20.0)
    [train_time,lr_w] = train_logistic(feature_train,label_train,
                                       C=0.001, dual=False, fit_intercept=True,
                                       intercept_scaling=1, max_iter=10000000000,
                                       multi_class='multinomial', penalty='l2',
                                       solver='lbfgs', tol=0.0001,class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(lr_w,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,lr_w)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series(['Feature Extraction Train Time',tm_feature_train_improved,
                    'Feature Extraction Test Time',tm_feature_test_improved,
                    'Train Time',train_time,
                    'Prediction Time',prediction_time,
                    'Accuracy',accuracy,
                    'AUC',auc,
                    'Balanced Accuracy',balanced_accuracy,name='Weighted Logistic'])
    model_results_df = model_results_df.append(row)

Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049 seconds

Training time: 123.215835 seconds
Prediction time: 0.022588 seconds

Accuracy: 0.830000
Balanced Accuracy: 0.679063
AUC: 0.822843
```

\\Users\rohan\opt\anaconda3\lib\python3.7\site-packages\sklearn\linear_model_logistic.py:764: Converge
nceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of f AND q EVALUATIONS EXCEEDS LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
extra_warning_msg='LOGISTIC_SOLVER_CONVERGENCE_MSG')

SVM

```
In [42]: # #grid search with cv 3 to find the best performed parameters
# param= {'C': [0.0001,0.0001,0.001,0.01,1,10],
#         'kernel':['linear', 'rbf', 'poly'],
#         'degree':[2,3,4]}

# gscv = GridSearchCV(SVC(random_state = 2020), param, cv=3, return_train_score=True)
# gscv.fit(feature_train,label_train)
# gscv.best_params_
# output: {'C': 10, 'degree': 4, 'kernel': 'poly'}
```

```
In [43]: from IPython.fs.full_train_svm import train_svm

#Improved svm using parameters from grid search
if run_svm==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f} seconds'.format(tm_feature_test_improved))

    weights = (0:1.0, 1:1.0)
    [train_time,svm_model] = train_svm(feature_train,label_train,
                                       C=10, kernel='poly', degree=4,
                                       probability=True,class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(svm_model,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,svm_model)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series(['Feature Extraction Train Time',tm_feature_train_improved,
                    'Feature Extraction Test Time',tm_feature_test_improved,
                    'Train Time',train_time,
                    'Prediction Time',prediction_time,
                    'Accuracy',accuracy,
                    'AUC',auc,
                    'Balanced Accuracy',balanced_accuracy,name='SVM'])
    model_results_df = model_results_df.append(row)

Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049 seconds

Training time: 79.830844 seconds
Prediction time: 1.624750 seconds

Accuracy: 0.855000
Balanced Accuracy: 0.683400
AUC: 0.828070
```

SVM With PCA

```
In [44]: # #grid search with cv 3 to find the best performed parameters
# param= {'C': [0.001,0.01,1,10,15,20],
#         'kernel':['linear', 'rbf', 'poly'],
#         'degree':[2,3,4]}

# gscv = GridSearchCV(SVC(random_state = 2020), param, cv=3, return_train_score=True)
# gscv.fit(feature_train_PCA,label_train_PCA)
# gscv.best_params_
# output: {'C': 10, 'degree': 2, 'kernel': 'rbf'}
```

```
In [45]: #Improved svm with PCA

if run_svm_pca==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_PCA))
    print('Feature extraction time for test: {:4f} seconds'.format(tm_feature_test_PCA))

    weights = (0:1.0, 1:1.0)
    [train_time,svm_PCA] = train_svm(feature_train_PCA,label_train_PCA,
                                       C=10,degree=2,kernel='rbf',probability=True,
                                       class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(svm_PCA,feature_test_PCA)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test_PCA,label_test_PCA,test_preds,svm
_PCA)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series(['Feature Extraction Train Time',tm_feature_train_PCA,
                    'Feature Extraction Test Time',tm_feature_test_PCA,
                    'Train Time',train_time,
                    'Prediction Time',prediction_time,
                    'Accuracy',accuracy,
                    'AUC',auc,
                    'Balanced Accuracy',balanced_accuracy,name='SVM with PCA'])
    model_results_df = model_results_df.append(row)

Feature extraction time for train: 6.715786 seconds
Feature extraction time for test: 0.114706 seconds

Training time: 0.746498 seconds
Prediction time: 0.018930 seconds

Accuracy: 0.786667
Balanced Accuracy: 0.585341
AUC: 0.745118
```

Weighted SVM

```
In [46]: # weights = (0:80.0, 1:20.0)
# param= {'C': [1,10,15,20],
#         'kernel':['linear', 'rbf', 'poly'],
#         'degree':[2,3,4]}

# cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
# gscv = GridSearchCV(LogisticRegression(penalty='l1', class_weight=weights), params, cv=3, scoring='ro
c_auc',verbose=True)
# gscv.fit(feature_train,label_train)
# gscv.best_params_
# output: output: {'C': 10, 'degree': 4, 'kernel': 'poly'}
```

```
In [47]: if run_weighted_svm==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f} seconds'.format(tm_feature_test_improved))

    weights = (0:80.0, 1:20.0)
    [train_time,svm_w] = train_svm(feature_train,label_train,
                                    C=10, kernel='poly', degree=4,
                                    probability=True,class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(svm_w,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,svm_w)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series(['Feature Extraction Train Time',tm_feature_train_improved,
                    'Feature Extraction Test Time',tm_feature_test_improved,
                    'Train Time',train_time,
                    'Prediction Time',prediction_time,
                    'Accuracy',accuracy,
                    'AUC',auc,
                    'Balanced Accuracy',balanced_accuracy,name='Weighted SVM'])
    model_results_df = model_results_df.append(row)

Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049 seconds

Training time: 384.685321 seconds
Prediction time: 1.667472 seconds

Accuracy: 0.836667
Balanced Accuracy: 0.717851
AUC: 0.837193
```

Naive Bayes

```
In [48]: if run_naivebayes == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f} seconds'.format(tm_feature_test_improved))

    [train_time,gnb] = train_naive_bayes(feature_train,label_train)
    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(gnb,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,gnb)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series(['Feature Extraction Train Time',tm_feature_train_improved,
                    'Feature Extraction Test Time',tm_feature_test_improved,
                    'Train Time',train_time,
                    'Prediction Time',prediction_time,
                    'Accuracy',accuracy,
                    'AUC',auc,
                    'Balanced Accuracy',balanced_accuracy,name='Naive Bayes'])
    model_results_df = model_results_df.append(row)

Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049 seconds

Training time: 0.138798 seconds
Prediction time: 0.037708 seconds

Accuracy: 0.663333
Balanced Accuracy: 0.628073
AUC: 0.660369
```

Lasso

```
In [49]: if run_lasso==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f} seconds'.format(tm_feature_test_improved))

    weights = (0:1.0, 1:1.0)
    [train_time,lasso_model] = train_lasso(feature_train,label_train,
                                           penalty='l1',solver='liblinear',
                                           class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(lasso_model,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,lasso_model
)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series(['Feature Extraction Train Time',tm_feature_train_improved,
                    'Feature Extraction Test Time',tm_feature_test_improved,
                    'Train Time',train_time,
                    'Prediction Time',prediction_time,
                    'Accuracy',accuracy,
                    'AUC',auc,
                    'Balanced Accuracy',balanced_accuracy,name='Lasso'])
    model_results_df = model_results_df.append(row)

Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049 seconds

Training time: 197.398233 seconds
Prediction time: 0.017074 seconds

Accuracy: 0.825000
Balanced Accuracy: 0.693171
AUC: 0.826389
```

Weighted Lasso

```
In [50]: # weights = (0:80.0, 1:20.0)
# param= {'solver':['liblinear', 'saga'],
#         'C':[1,10,15,20],
#         'degree':[2,3,4]}

# cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
# gscv = GridSearchCV(LogisticRegression(penalty='l1', class_weight=weights), params, cv=3, scoring='ro
c_auc',verbose=True)
# gscv.fit(feature_train,label_train)
# gscv.best_params_

In [51]: if run_weighted_lasso==True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_improved))
    print('Feature extraction time for test: {:4f} seconds'.format(tm_feature_test_improved))

    weights = (0:80.0, 1:20.0)
    [train_time,lasso_w] = train_lasso(feature_train,label_train,
                                       penalty='l1',solver='liblinear',
                                       class_weight=weights)

    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(lasso_w,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,lasso_w)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series(['Feature Extraction Train Time',tm_feature_train_improved,
                    'Feature Extraction Test Time',tm_feature_test_improved,
                    'Train Time',train_time,
                    'Prediction Time',prediction_time,
                    'Accuracy',accuracy,
                    'AUC',auc,
                    'Balanced Accuracy',balanced_accuracy,name='Weighted Lasso'])
    model_results_df = model_results_df.append(row)

Feature extraction time for train: 0.181775 seconds
Feature extraction time for test: 0.036049 seconds

Training time: 56.160900 seconds
Prediction time: 0.017086 seconds

Accuracy: 0.831667
Balanced Accuracy: 0.622522
AUC: 0.819164
```

SMOTE Bagging

```
In [52]: #params = {'n_estimators':[25,50,75,100]}

# cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)
# gscv = GridSearchCV(LogisticClassifier(), params,cv=cv,scoring='roc_auc').fit(feature_train_sm,label_tra
in_sm)
# gscv.best_params_
# output: {'n_estimators': 100}

In [53]: if run_bagging_smote == True:

    print('Feature extraction time for train: {:4f} seconds'.format(tm_feature_train_SMOTE))
    print('Feature extraction time for test: {:4f} seconds'.format(tm_feature_test_improved))

    [train_time,smote_bagging] = train_bagging(feature_train_sm,label_train_sm,n_estimators=100)
    print('\nTraining time: {:4f} seconds'.format(train_time))

    [prediction_time,test_preds] = test_model(smote_bagging,feature_test)
    print('Prediction time: {:4f} seconds'.format(prediction_time))

    [accuracy, balanced_accuracy, auc] = compute_metrics(feature_test,label_test,test_preds,smote_baggi
ng)
    print('\nAccuracy: {:4f}'.format(accuracy))
    print('Balanced Accuracy: {:4f}'.format(balanced_accuracy))
    print('AUC: {:4f}'.format(auc))

    row = pd.Series(['Feature Extraction Train Time',tm_feature_train_SMOTE,
                    'Feature Extraction Test Time',tm_feature_test_improved,
                    'Train Time',train_time,
                    'Prediction Time',prediction_time,
                    'Accuracy',accuracy,
                    'AUC',auc,
                    'Balanced Accuracy',balanced_accuracy,name='SMOTE Bagging'])
    model_results_df = model_results_df.append(row)

Feature extraction time for train: 2.594956 seconds
Feature extraction time for test: 0.036049 seconds

Training time: 639.069607 seconds
Prediction time: 0.614470 seconds

Accuracy: 0.806667
Balanced Accuracy: 0.632585
AUC: 0.794535
```

Model Results Table

In this part, we display the model results table for all of the models that were set to run.

```
In [54]: model_results_df = model_results_df.applymap(lambda x: round(x,3))
model_results_df['Train Time'] = [str(x)+' s' for x in list(model_results_df['Train Time'])]
model_results_df['Prediction Time'] = [str(x)+' s' for x in list(model_results_df['Prediction Time'])]
model_results_df['Feature Extraction Train Time'] = [str(x)+' s' for x in list(model_results_df['Fea
ture Extraction Train Time'])]
model_results_df['Feature Extraction Test Time'] = [str(x)+' s' for x in list(model_results_df['Fea
ture Extraction Test Time'])]

In [55]: model_results_df
```

	Feature Extraction Train Time	Feature Extraction Test Time	Train Time	Prediction Time	Accuracy	Balanced Accuracy	AUC
Baseline	5.326 s	1.203 s	186.644 s	0.076 s	0.808	0.588	0.785
Advanced (SMOTEBoost)	2.595 s	0.036 s	206.306 s	0.068 s	0.810	0.707	0.827
Baseline with Improved SMOTE	0.182 s	0.036 s	183.700 s	0.024 s	0.817	0.610	0.798
Baseline with PCA	6.716 s	0.115 s	1.192 s	0.002 s	0.790	0.536	0.693
KNN	0.182 s	0.036 s	0.585 s	5.694 s	0.792	0.517	0.675
SMOTE KNN	2.595 s	0.036 s	0.732 s	7.721 s	0.607	0.638	0.684
XGBoost	0.182 s	0.036 s	85.553 s	0.068 s	0.813	0.689	0.809
Random Forest	0.182 s	0.036 s	6.694 s	0.035 s	0.810	0.563	0.786
LDA	0.182 s	0.036 s	5.942 s	0.019 s	0.822	0.636	0.786
Logistic Regression	0.182 s	0.036 s	97.021 s	0.023 s	0.822	0.700	0.822
Weighted Logistic	0.182 s	0.036 s	122.26 s	0.023 s	0.830	0.679	0.823
SVM	0.182 s	0.036 s	79.831 s	1.625 s	0.857	0.683	0.828
SVM with PCA	6.716 s	0.115 s	0.746 s	0.019 s	0.787	0.585	0.745
Weighted SVM	0.182 s	0.036 s	384.686 s	1.667 s	0.837	0.718	0.837
Naive Bayes	0.182 s	0.036 s	0.139 s	0.038 s	0.663	0.628	0.660
Lasso	0.182 s	0.036 s	197.398 s	0.017 s	0.822	0.693	0.826
Weighted Lasso	0.182 s	0.036 s	56.16 s	0.017 s	0.835	0.683	0.819
SMOTE Bagging	2.595 s	0.036 s	639.07 s	0.614 s	0.807	0.633	0.795

```
In [56]: #store model_saved_model_results == True:
if overwrite_saved_model_results == True:
    pickle.dump(model_results_df, open("../output/model_results.p", "wb" ) )
```

Optional Step: 10-fold Cross Validation

This is how we did the 10-fold cross validation with AUC scoring to choose from the best candidate models. By default the option to run this cell is set to False.

```
In [57]: if run_10_cv == True:

    # weighted_lasso
    print("Cross Validation Score: ", np.mean(cross_val_score(lasso_w, feature_train, label_train, cv=1
0, scoring='roc_auc')))
    # output:0.8010359440647241

    # weighted_svm
    print("Cross Validation Score: ", np.mean(cross_val_score(svm_w, feature_train, label_train, cv=10,
scoring='roc_auc')))
    # output:0.817648633580459

    # weighted_logistic
    print("Cross Validation Score: ", np.mean(cross_val_score(lr_w, feature_train, label_train, cv=10,
scoring='roc_auc')))
    # output:0.8300000000000001

    # XGBoost with SMOTE
    print("Cross Validation Score: ", np.mean(cross_val_score(advanced, feature_train, label_train, cv=
10, scoring='roc_auc')))
    # output:0.838799826113715
```

References

- <https://machinelearningmastery.com/sMOTE-oversampling-for-imbalanced-classification/>
- <https://arxiv.org/pdf/1106.1813.pdf>
- <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3694838/>