Arrhythmia classification (paper)

January 2, 2022

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
# Transformation

import numpy as np
from sklearn import datasets
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import power_transform
from sklearn.pipeline import Pipeline
# Feature Selection

# Models
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
```

[ ]:
```
!pip install -U pymrmr
```

[ ]:
```
#load data
data = pd.read_csv('arrhythmia_data_set.csv')
data.head()
#data = data.apply(pd.to_numeric) # convert all columns of DataFrame
```

[ ]:
```
#Divide into feature (X) and Labels (Y)
X = data.drop('Arrhythmia Class (0=yes,1=No)',axis=1) # X = all 'data' except␣
 ↪the 'Arrhythmia Class (0=yes,1=No)' column
y = data['Arrhythmia Class (0=yes,1=No)'] # y = 'Arrhythmia Class (0=yes,1=No)'␣
 ↪column from 'data'
```

[ ]:
```
#Calculate correlation coefficients
from scipy.stats.stats import pearsonr
features = list(X)
correlation = []
significance = []
```

```python
for feature in features:
    correl = pearsonr(X[feature].values, y.values)
    correlation.append(correl[0])
    significance.append(correl[1])
df = pd.DataFrame()
df['feature'] = features
df['correlation'] = correlation
df['abs_correlation'] = np.abs(correlation)
df['significance'] = significance
df['significant'] = df['significance'] < 0.05 # Label those P<0.01
df.sort_values(by='abs_correlation', ascending=False, inplace=True)
df
```

[ ]:

```python
df1=df[df['significant']==True]
df1['feature']
ordered_features = list(df1['feature'])

X1=data[['QRS duration',
 'Amplitude AVR channel QRSTA (area)',
 'Amplitude DI channel T wave',
 'Amplitude DI channel QRSTA (area)',
 'Amplitude AVR channel T wave',
 'Amplitude V6 channel T wave',
 'Amplitude V6 channel ',
 'Channel V1 Number of intrinsic deflections',
 'Amplitude DII channel QRSTA (area)',
 'Amplitude AVLchannel T wave',
 'Amplitude V3 channel QRSTA (area)',
 'sex',
 'Amplitude DII channel T wave',
 'Amplitude V5 channel T wave',
 'T int',
 'Amplitude V3 channel QRSA (sum of area) ',
 'Amplitude V1 channel QRSTA (area)',
 'Amplitude DI channel S wave',
 'Amplitude DII channel S wave',
 'Amplitude V4 channel QRSTA (area)',
 'Channel V4 Average width S',
 'Channel AVL  Number of intrinsic deflections',
 'Amplitude V5 channel QRSTA (area)',
 'Amplitude V4 channel QRSA (sum of area) ',
 'Channel AVR Average width R',
 'Amplitude V4 channel  S wave',
 'Channel V6  Number of intrinsic deflections',
 'Amplitude DII channel QRSA (sum of area) ',
 'Amplitude V6 channel JJ wave',
 'Amplitude V2 channel QRSTA (area)',
 'Amplitude AVR channel JJ wave',
 'Amplitude AVR channel Q wave',
 'Amplitude V3 channel S wave',
 'Channel AVF Number of intrinsic deflections',
 'Channel V3 Average width R',
 'Amplitude DI channel JJ wave',
 'Amplitude AVFchannel  QRSTA (area)',
 'Channel V5 Number of intrinsic deflections',
 'Amplitude AVR channel QRSA (sum of area) ',
 'Amplitude AVLchannel S wave',
 'Amplitude AVLchannel QRSTA (area)',
 'Amplitude V1 channel T wave',
 'Amplitude V5 channel S wave',
 'Amplitude V1 channel  R wave',
 'Amplitude AVFchannel  QRSA (sum of area) ',
 'Channel V4 Average width R',
 'Amplitude V4 channel T wave',
 'Channel V2 Number of intrinsic deflections',
 'Amplitude V6 channel S wave',
 'Channel AVL Average width R',
 'Channel V5 Average width S',
 'Channel V2 Average width S',
 'Amplitude V5 channel JJ wave',
 'Amplitude AVLchannel R wave',
 'Amplitude V2 channel QRSA (sum of area) ',
 'Channel DII  Number of intrinsic deflections',
 'Channel DI Number of intrinsic deflections',
 'Amplitude DII channel R wave',
 'Channel AVR  Number of intrinsic deflections',
 'Amplitude V3 channel T wave',
 'Channel DIII Number of intrinsic deflections',
 ' Amplitude V2 channel T wave',
 'Amplitude V5 channel QRSA (sum of area) ',
 'Amplitude AVLchannel JJ wave',
 'Amplitude DII channel JJ wave',
 'P int',
 'Channel V3 Average width S',
 'Amplitude AVFchannel  T wave',
 'Amplitude V1 channel JJ wave',
 'Amplitude DII channel P wave',
 'Heart rate']]
X1
```

[ ]:

```
from mrmr import mrmr_classif
```

[ ]:

```
selected_features = mrmr_classif(X, y, K = 60)
print(selected_features)
```

[ ]:

```
import pymrmr
selected_features2 = pymrmr.mRMR(data, 'MIQ',40) # n is number of features we␣
 ↪want to select
selected_features2
```

[ ]:

```
P=list(selected_features2)
X3=data[P]
X3.head()
```

[ ]:

```
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression(max_iter=10000)
# feature extraction
model = LogisticRegression(solver='lbfgs')
rfe = RFE(model, 20)
fit = rfe.fit(X, y)
print("Num Features: %d" % fit.n_features_)
```

[ ]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import RFECV
rfc = RandomForestClassifier(random_state=100)
rfecv = RFECV(estimator=rfc, step=1, cv=StratifiedKFold(10), scoring='accuracy')
rfecv.fit(X3, y)
```

[ ]:

```
print('Optimal number of features: {}'.format(rfecv.n_features_))
rfecv.support_
rfecv.ranking_
```

[ ]:

```
plt.figure(figsize=(9, 6))
plt.title('Recursive Feature Elimination with Cross-Validation', fontsize=18,␣
 ↪fontweight='bold', pad=20)
plt.xlabel('Number of features selected', fontsize=20, labelpad=20)
plt.ylabel('% Correct Classification', fontsize=20, labelpad=20)
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_,␣
 ↪color='#303F9F', linewidth=3)

plt.show()
```

[ ]:

```
print(np.where(rfecv.support_ == False)[0])

X3.drop(X3.columns[np.where(rfecv.support_ == False)[0]], axis=1, inplace=True)
```

[ ]:

```
dset = pd.DataFrame()
dset['attr'] = X3.columns
dset['importance'] = rfecv.estimator_.feature_importances_
```

```
dset = dset.sort_values(by='importance', ascending=False)


plt.figure(figsize=(5, 8))
plt.barh(y=dset['attr'], width=dset['importance'], color='#1976D2')
plt.title('RFECV - Feature Importances', fontsize=10, fontweight='bold', pad=10)
plt.xlabel('Importance', fontsize=10, labelpad=20)
plt.show()
print(list(dset['attr']))
X5=X3[list(dset['attr'])]
X5.head()
```

[ ]:

```
X5=X3[['Heart rate', 'QRS duration', 'Amplitude DII channel QRSTA (area)',␣
 ↪'Amplitude V3 channel QRSTA (area)', 'Amplitude DI channel QRSTA (area)',␣
 ↪'Vector angle T', 'Amplitude V3 channel QRSA (sum of area) ', 'Amplitude DI␣
 ↪channel QRSA (sum of area) ', 'Amplitude V6 channel ', 'T int', 'Amplitude␣
 ↪V2 channel QRSA (sum of area) ', 'Amplitude V4 channel  R wave', 'Amplitude␣
 ↪V1 channel QRSTA (area)', 'Amplitude V4 channel QRSTA (area)', 'QT int',␣
 ↪'Channel AVF Number of intrinsic deflections', 'Amplitude V4 channel QRSA␣
 ↪(sum of area) ', 'Amplitude V2 channel QRSTA (area)', 'Vector angle QRS',␣
 ↪'Channel AVL  Number of intrinsic deflections', 'Amplitude AVLchannel QRSTA␣
 ↪(area)', 'Amplitude AVR channel QRSA (sum of area) ', 'Vector angle QRST',␣
 ↪'Amplitude V1 channel  S wave', 'PR int', 'Height', 'P int', 'Amplitude V1␣
 ↪channel  R wave', 'Amplitude V5 channel QRSTA (area)', 'Weight', 'Channel␣
 ↪DIII Average width R', 'Amplitude AVLchannel QRSA (sum of area) ']]
X5.head()
```

[ ]:

```
import pandas as pd
import numpy as np
from sklearn.pipeline import make_pipeline
from skrebate import ReliefF
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
data = pd.read_csv('arrhythmia_data_set.csv')
X = data.drop('Arrhythmia Class (0=yes,1=No)',axis=1) # X = all 'data' except␣
 ↪the 'Arrhythmia Class (0=yes,1=No)' column
y = data['Arrhythmia Class (0=yes,1=No)'] # y = 'Arrhythmia Class (0=yes,1=No)'␣
 ↪column from 'data'
clf = make_pipeline(ReliefF(n_features_to_select=60,␣
 ↪n_neighbors=100),RandomForestClassifier(n_estimators=100))

print(clf)
```

[ ]:

```
print(data)
```

[ ]:

```
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score,␣
 ↪recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis,␣
 ↪QuadraticDiscriminantAnalysis
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import mean_squared_error as mse
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import LinearSVC
from sklearn.ensemble import VotingClassifier
```

[ ]:

```
#logistic regression
X_train, X_test, y_train, y_test = train_test_split(X5,y,test_size=0.34)
LR = LogisticRegression(max_iter=10000)

scoring = ['accuracy', 'precision_macro', 'recall_macro' , 'f1_weighted',␣
 ↪'roc_auc']
scores = cross_validate(LR, X_train, y_train, scoring=scoring, cv=20)

sorted(scores.keys())
LR_fit_time = scores['fit_time'].mean()
LR_score_time = scores['score_time'].mean()
LR_accuracy = scores['test_accuracy'].mean()
LR_precision = scores['test_precision_macro'].mean()
LR_recall = scores['test_recall_macro'].mean()
LR_f1 = scores['test_f1_weighted'].mean()
LR_roc = scores['test_roc_auc'].mean()
```

[ ]:

```
LR_accuracy
```

[ ]:

```
decision_tree = DecisionTreeClassifier()

scoring = ['accuracy', 'precision_macro', 'recall_macro' , 'f1_weighted',␣
 ↪'roc_auc']
scores = cross_validate(decision_tree, X_train, y_train, scoring=scoring, cv=20)

sorted(scores.keys())
```

```
dtree_fit_time = scores['fit_time'].mean()
dtree_score_time = scores['score_time'].mean()
dtree_accuracy = scores['test_accuracy'].mean()
dtree_precision = scores['test_precision_macro'].mean()
dtree_recall = scores['test_recall_macro'].mean()
dtree_f1 = scores['test_f1_weighted'].mean()
dtree_roc = scores['test_roc_auc'].mean()
```

[ ]:

```
dtree_accuracy
```

[ ]:

```
SVM = SVC(probability = True)

scoring = ['accuracy','precision_macro', 'recall_macro' , 'f1_weighted',␣
 ↪'roc_auc']
scores = cross_validate(SVM, X_train, y_train, scoring=scoring, cv=20)

sorted(scores.keys())
SVM_fit_time = scores['fit_time'].mean()
SVM_score_time = scores['score_time'].mean()
SVM_accuracy = scores['test_accuracy'].mean()
SVM_precision = scores['test_precision_macro'].mean()
SVM_recall = scores['test_recall_macro'].mean()
SVM_f1 = scores['test_f1_weighted'].mean()
SVM_roc = scores['test_roc_auc'].mean()
```

[ ]:

```
SVM_accuracy
```

[ ]:

```
LDA = LinearDiscriminantAnalysis()

scoring = ['accuracy', 'precision_macro', 'recall_macro' , 'f1_weighted',␣
 ↪'roc_auc']
scores = cross_validate(LDA, X_train, y_train, scoring=scoring, cv=20)

sorted(scores.keys())
LDA_fit_time = scores['fit_time'].mean()
LDA_score_time = scores['score_time'].mean()
LDA_accuracy = scores['test_accuracy'].mean()
LDA_precision = scores['test_precision_macro'].mean()
LDA_recall = scores['test_recall_macro'].mean()
LDA_f1 = scores['test_f1_weighted'].mean()
LDA_roc = scores['test_roc_auc'].mean()
```

[ ]:

```
LDA_accuracy
```

[ ]:

```
QDA = QuadraticDiscriminantAnalysis()
```

```
scoring = ['accuracy', 'precision_macro', 'recall_macro' , 'f1_weighted',␣
 ↪'roc_auc']
scores = cross_validate(QDA, X_train, y_train, scoring=scoring, cv=20)

sorted(scores.keys())
QDA_fit_time = scores['fit_time'].mean()
QDA_score_time = scores['score_time'].mean()
QDA_accuracy = scores['test_accuracy'].mean()
QDA_precision = scores['test_precision_macro'].mean()
QDA_recall = scores['test_recall_macro'].mean()
QDA_f1 = scores['test_f1_weighted'].mean()
QDA_roc = scores['test_roc_auc'].mean()
```

[ ]:

```
QDA_accuracy
```

[ ]:

```
random_forest = RandomForestClassifier()

scoring = ['accuracy', 'precision_macro', 'recall_macro' , 'f1_weighted',␣
 ↪'roc_auc']
scores = cross_validate(random_forest, X_train, y_train, scoring=scoring, cv=20)

sorted(scores.keys())
forest_fit_time = scores['fit_time'].mean()
forest_score_time = scores['score_time'].mean()
forest_accuracy = scores['test_accuracy'].mean()
forest_precision = scores['test_precision_macro'].mean()
forest_recall = scores['test_recall_macro'].mean()
forest_f1 = scores['test_f1_weighted'].mean()
forest_roc = scores['test_roc_auc'].mean()
```

[ ]:

```
forest_accuracy
```

[ ]:

```
KNN = KNeighborsClassifier()

scoring = ['accuracy', 'precision_macro', 'recall_macro' , 'f1_weighted',␣
 ↪'roc_auc']
scores = cross_validate(KNN, X_train, y_train, scoring=scoring, cv=20)

sorted(scores.keys())
KNN_fit_time = scores['fit_time'].mean()
KNN_score_time = scores['score_time'].mean()
KNN_accuracy = scores['test_accuracy'].mean()
KNN_precision = scores['test_precision_macro'].mean()
KNN_recall = scores['test_recall_macro'].mean()
KNN_f1 = scores['test_f1_weighted'].mean()
KNN_roc = scores['test_roc_auc'].mean()
```

[ ]:
```
KNN_accuracy
```

[ ]:
```
bayes = GaussianNB()

scoring = ['accuracy', 'precision_macro', 'recall_macro' , 'f1_weighted',␣
 ↪'roc_auc']
scores = cross_validate(bayes, X_train, y_train, scoring=scoring, cv=20)

sorted(scores.keys())
bayes_fit_time = scores['fit_time'].mean()
bayes_score_time = scores['score_time'].mean()
bayes_accuracy = scores['test_accuracy'].mean()
bayes_precision = scores['test_precision_macro'].mean()
bayes_recall = scores['test_recall_macro'].mean()
bayes_f1 = scores['test_f1_weighted'].mean()
bayes_roc = scores['test_roc_auc'].mean()
```

[ ]:
```
bayes_accuracy
```

[ ]:
```
models_initial = pd.DataFrame({
    'Model'       : ['Logistic Regression', 'Decision Tree', 'Support Vector␣
 ↪Machine', 'Linear Discriminant Analysis', 'Quadratic Discriminant Analysis',␣
 ↪'Random Forest', 'K-Nearest Neighbors', 'Bayes'],
    'Fitting time': [LR_fit_time, dtree_fit_time, SVM_fit_time, LDA_fit_time,␣
 ↪QDA_fit_time, forest_fit_time, KNN_fit_time, bayes_fit_time],
    'Scoring time': [LR_score_time, dtree_score_time, SVM_score_time,␣
 ↪LDA_score_time, QDA_score_time, forest_score_time, KNN_score_time,␣
 ↪bayes_score_time],
    'Accuracy'    : [LR_accuracy, dtree_accuracy, SVM_accuracy, LDA_accuracy,␣
 ↪QDA_accuracy, forest_accuracy, KNN_accuracy, bayes_accuracy],
    'Precision'   : [LR_precision, dtree_precision, SVM_precision,␣
 ↪LDA_precision, QDA_precision, forest_precision, KNN_precision,␣
 ↪bayes_precision],
    'Recall'      : [LR_recall, dtree_recall, SVM_recall, LDA_recall,␣
 ↪QDA_recall, forest_recall, KNN_recall, bayes_recall],
    'F1_score'    : [LR_f1, dtree_f1, SVM_f1, LDA_f1, QDA_f1, forest_f1,␣
 ↪KNN_f1, bayes_f1],
    'AUC_ROC'     : [LR_roc, dtree_roc, SVM_roc, LDA_roc, QDA_roc, forest_roc,␣
 ↪KNN_roc, bayes_roc],
    }, columns = ['Model', 'Fitting time', 'Scoring time', 'Accuracy',␣
 ↪'Precision', 'Recall', 'F1_score', 'AUC_ROC'])

models_initial.sort_values(by='Accuracy', ascending=False)
```

[ ]:
```
#voting classifier
models = [LogisticRegression(),
         DecisionTreeClassifier(),
         SVC(probability = True),
         LinearDiscriminantAnalysis(),
         QuadraticDiscriminantAnalysis(),
         RandomForestClassifier(),
         KNeighborsClassifier(),
         GaussianNB()]

scoring = ['accuracy', 'precision_macro', 'recall_macro' , 'f1_weighted',␣
 ↪'roc_auc']
for model in models:
    scores = cross_validate(model, X_train, y_train, scoring=scoring, cv=20)
```

[ ]:

```
#hard voting
models_ens = list(zip(['LR', 'DT', 'SVM', 'LDA', 'QDA', 'RF', 'KNN', 'NB'],␣
 ↪models))

model_ens = VotingClassifier(estimators = models_ens, voting = 'hard')
model_ens.fit(X_train, y_train)
pred = model_ens.predict(X_test)
#prob = model_ens.predict_proba(X_test)[:,1]

acc_hard = accuracy_score(y_test, pred)
prec_hard = precision_score(y_test, pred)
recall_hard = recall_score(y_test, pred)
f1_hard = f1_score(y_test, pred)
```

[ ]:

```
#soft voting
model_ens = VotingClassifier(estimators = models_ens, voting = 'soft')
model_ens.fit(X_train, y_train)
pred = model_ens.predict(X_test)
prob = model_ens.predict_proba(X_test)[:,1]

acc_soft = accuracy_score(y_test, pred)
prec_soft = precision_score(y_test, pred)
recall_soft = recall_score(y_test, pred)
f1_soft = f1_score(y_test, pred)
roc_auc_soft = roc_auc_score(y_test, prob)
```

[ ]:

```
#voting comparison
models_ensembling = pd.DataFrame({
    'Model'       : ['Ensebling_hard', 'Ensembling_soft'],
    'Accuracy'    : [acc_hard, acc_soft],
    'Precision'   : [prec_hard, prec_soft],
    'Recall'      : [recall_hard, recall_soft],
    'F1_score'    : [f1_hard, f1_soft],

    }, columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1_score'])

models_ensembling.sort_values(by='Accuracy', ascending=False)
```