Report

Programming Assignment 2

Xiangyu Gao

Answer 1) (20 points)

1.1 Cleaning

```
def cleaning(df):
  # Take all the features as input, and do any data cleaning necessary.
  # YOUR CODE HERE
    df_re = pd.DataFrame()
    df = df.drop('Id',axis = 1)
    #data_ = data_[(np.abs(stats.zscore(data_))<3).all(axis = 1)]
    if 'stress' in df.columns:
        for i in set(df['stress']):
            df_i = df.where(df['stress'] == i)
            df_i = df_i.dropna(axis=0,how='all')
            for i in df_i.columns:
                if df_i.count()[i] < len(df_i)*0.2:
                    del df_i[i]
            df_re = pd.concat([df_re,df_i],axis = 0)
    else:
        for i in df.columns:
            if df.count()[i] < len(df)*0.2:
                del df[i]
        df_re = pd.concat([df_re,df],axis = 0)
    return df_re
```

In this part, I dropped Ids in the data, which are the unique features of the samples. I suppose that this kind of feature is useless and unpractical. Furthermore, I used a for-loop in the function, to fill the missing data by groups. So that the samples that have the same label will have the features in the same format. Then, in each group, I removed the columns in which the number of samples has less than 20% of the total.

1.2 Normalizer

```
def normalizer(df):
  # Taken input the output of cleaning function,
  #and perform data normalization independently for all the features.
  # YOUR CODE HERE
    data_std = df.copy()
    std = StandardScaler()
    if 'stress' in df.columns:
        for col in df.columns.drop('stress'):
            data_std[col] = std.fit_transform(data_std[col].values.reshape(-1,1))
    else:
        for col in df.columns:
            data_std[col] = std.fit_transform(data_std[col].values.reshape(-1,1))
    return data_std
```

I used the StandardScaler function to normalize.

1.3 Discussion

```
data1 = cleaning(data)
data1 = missing_filler1(data1)
data1 = normalizer(data1)
data1
```

	HR	interval in seconds	NNRR	AVNN	SDNN	RMSSD	pNN50	TP	ULF	VLF	LF	HF	stress
0	1.317987	-1.285923	1.382054	-1.314653	-1.726485	-1.610495	-1.369549	0.041373	-0.998874	-0.004105	-0.000664	-0.003098	0.0
1	0.281139	-1.298530	1.382054	-1.314653	-1.726485	-1.610495	-1.369549	0.041373	-0.998874	-0.004105	-0.000664	-0.003098	0.0
2	0.348898	-1.298530	1.382054	-1.314653	-1.726485	-1.610495	-1.369549	0.041373	-0.998874	-0.004105	-0.000664	-0.003098	0.0
3	0.503595	-1.298530	1.382054	-1.314653	-1.726485	-1.610495	-1.369549	0.041373	-0.998874	-0.004105	-0.000664	-0.003098	0.0
4	-0.396196	-1.208366	1.192306	-1.221701	-1.013377	-1.030283	-0.463398	-1.051409	-0.998874	-0.161398	9.108869	3.051882	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3071	-0.163645	-0.062251	-0.147016	-0.072406	-0.155162	-0.121686	-0.320323	-0.381779	-0.329426	-0.231222	-0.117125	-0.113825	2.0
3072	-0.285324	-0.062251	-0.147016	-0.072406	-0.155162	-0.121686	-0.320323	-0.390228	-0.337752	-0.231222	-0.117125	-0.113825	2.0
3073	0.163489	-0.589647	0.569260	-0.601966	-0.863137	-0.796746	-0.397097	-0.844041	-0.784983	-0.231222	-0.117125	-0.113825	2.0
3074	0.252923	-0.589647	0.569260	-0.601966	-0.863137	-0.796746	-0.397097	-0.843466	-0.784416	-0.231222	-0.117125	-0.113825	2.0
3075	0.111785	-0.589647	0.569260	-0.601966	-0.863137	-0.796746	-0.397097	-0.846834	-0.787735	-0.231222	-0.117125	-0.113825	2.0

3129 rows × 13 columns

The dataset has been cleaned, filled, and normalized. The feature LF_HF has been removed since it has less than 20% data of the total.

The F1 score is 0.49 after doing these things.

```
X = np.array(data1.drop(['stress'],axis = 1))
y = np.array(data1['stress'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
y_pred_lr,LR_model = logisticregression(X_train,y_train,X_test,y_test)
metrics(y_pred_lr, y_train,'logisticregression')
```

```
Logistic Regression
train score: 0.5601278465840991
test score: 0.536741214057508
logisticregression
confusion_matrix:
[[ 275  382   20]
 [ 111 1072   29]
 [  37  522   55]]
---------------------------------
classification_report:
              precision    recall  f1-score   support

         0.0       0.65      0.41      0.50       677
         1.0       0.54      0.88      0.67      1212
         2.0       0.53      0.09      0.15       614

    accuracy                           0.56      2503
   macro avg       0.57      0.46      0.44      2503
weighted avg       0.57      0.56      0.50      2503

---------------------------------
accuracy_score:
0.5601278465840991
---------------------------------
f1_score:
0.49846722519528497
```

If we do not normalize it, the f1 score will be lower than before: 0.45

```
X = np.array(data1.drop(['stress'],axis = 1))
y = np.array(data1['stress'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
y_pred_lr,LR_model = logisticregression(X_train,y_train,X_test,y_test)
metrics(y_pred_lr, y_train,'logisticregression')
```

```
Logistic Regression
train score: 0.5373551737914503
test score: 0.5287539936102237
logisticregression
confusion_matrix:
[[ 248  452    0]
 [ 101 1087   13]
 [  23  569   10]]
---------------------------------
classification_report:
              precision    recall  f1-score   support

         0.0       0.67      0.35      0.46       700
         1.0       0.52      0.91      0.66      1201
         2.0       0.43      0.02      0.03       602

    accuracy                           0.54      2503
   macro avg       0.54      0.43      0.38      2503
weighted avg       0.54      0.54      0.45      2503

---------------------------------
accuracy_score:
0.5373551737914503
---------------------------------
f1_score:
0.4523359483304094
```

If we do not clean it, the f1 score will be 0.44

```
X = np.array(data1.drop(['stress'],axis = 1))
y = np.array(data1['stress'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
y_pred_lr,LR_model = logisticregression(X_train,y_train,X_test,y_test)
metrics(y_pred_lr, y_train,'logisticregression')
```

```
Logistic Regression
train score: 0.525769077107471
test score: 0.5463258785942492
logisticregression
confusion_matrix:
[[ 250  452    0]
 [ 114 1052   19]
 [  22  580   14]]
---------------------------------
classification_report:
              precision    recall  f1-score   support

         0.0       0.65      0.36      0.46       702
         1.0       0.50      0.89      0.64      1185
         2.0       0.42      0.02      0.04       616

    accuracy                           0.53      2503
   macro avg       0.53      0.42      0.38      2503
weighted avg       0.53      0.53      0.44      2503

---------------------------------
accuracy_score:
0.525769077107471
---------------------------------
f1_score:
0.4442183459969252
```

Answer 2) (15 points)

Briefly describe how much data is missing in the dataset provided.

How did you handle the missing data? Discuss at least 2 different methods used over here, and which one gives better increment in metrics and why?

2.1 dataset

```
data = load_csv('data_2-1.csv')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3129 entries, 0 to 3128
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Id                   3129 non-null   int64
 1   HR                   3020 non-null   float64
 2   interval in seconds  3129 non-null   float64
 3   NNRR                 3071 non-null   float64
 4   AVNN                 3055 non-null   float64
 5   SDNN                 3039 non-null   float64
 6   RMSSD                3129 non-null   float64
 7   pNN50                3129 non-null   float64
 8   TP                   2910 non-null   float64
 9   ULF                  3006 non-null   float64
 10  VLF                  3033 non-null   float64
 11  LF                   3033 non-null   float64
 12  HF                   3033 non-null   float64
 13  LF_HF                48 non-null     float64
 14  stress               3129 non-null   float64
dtypes: float64(14), int64(1)
memory usage: 366.8 KB
None
```

This dataset, like the picture above, has 14 kinds of features: Id, HR, the interval in seconds, NNRR, AVNN, SDNN, RMSSD, pNN50, TP, ULF, VLF, LF, HF, and LF_HF. The label is stress. As we can see, some features are intact: Id, the interval in seconds, RMSSD, and pNN50, while some are incomplete up to 7% lost, like HR, NNRR, AVNN, SDNN, TP, ULF, VLF, LF, and HF. What is worth to be noticed is that LF_HF only has 1.5% of the total: 48 samples, which is too small to be an effective feature.

2.2 Missing Filler 1

```
def missing_filler1(df):
  # Take input the raw data and fillup the missing values using first algorithm.
  # YOUR CODE HERE
    df_re = pd.DataFrame()
    if 'stress' in df.columns:
        for i in set(df['stress']):
            df_i = df.where(df['stress'] == i)
            df_i = df_i.replace(np.inf, np.nan)
            df_i = df_i.dropna(axis=0,how='all')
            df_i = df_i.fillna(df_i.mean())
            df_re = pd.concat([df_re,df_i],axis = 0)
    else:
            df_i = df.replace(np.inf, np.nan)
            df_i = df_i.dropna(axis=0,how='all')
            df_i = df_i.fillna(df_i.mean())
            df_re = pd.concat([df_re,df_i],axis = 0)
    return df_re
```

For method 1, I replaced all Inf with NaN, and fill NaN to be the mean value of the data in each label group.

```
X = np.array(data1.drop(['stress'],axis = 1))
y = np.array(data1['stress'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
y_pred_lr,LR_model = logisticregression(X_train,y_train,X_test,y_test)
metrics(y_pred_lr, y_train,'logisticregression')
```

```
train score: 0.5577307231322413
test score: 0.5591054313099042
logisticregression
confusion_matrix:
[[ 321  349   24]
 [ 137 1034   22]
 [  44  531   41]]
--------------------------------
classification_report:
              precision    recall  f1-score   support

         0.0       0.64      0.46      0.54       694
         1.0       0.54      0.87      0.67      1193
         2.0       0.47      0.07      0.12       616

    accuracy                           0.56      2503
   macro avg       0.55      0.47      0.44      2503
weighted avg       0.55      0.56      0.49      2503

--------------------------------
accuracy_score:
0.5577307231322413
--------------------------------
f1_score:
0.4947811677618696
```

The f1 score is 0.49 and the accuracy is 0.55.

2.3 Missing Filler 2

```
def missing_filler2(df):
  # Take input the raw data and fillup the missing values using first algorithm.
  # YOUR CODE HERE
    df_re = pd.DataFrame()
    if 'stress' in df.columns:
        for i in set(df['stress']):
            df_i = df.where(df['stress'] == i)
            df_i = df_i.replace(np.inf, np.nan)
            df_i = df_i.dropna(axis=0,how='all')
            df_i = df_i.interpolate()
            df_i = df_i.fillna(df_i.mean())
            df_re = pd.concat([df_re,df_i],axis = 0)
    else:
            df_i = df.replace(np.inf, np.nan)
            df_i = df_i.dropna(axis=0,how='all')
            df_i = df_i.interpolate()
            df_i = df_i.fillna(df_i.mean())
            df_re = pd.concat([df_re,df_i],axis = 0)
    return df_re
```

In method 2, similarly, I replaced Inf with NaN, while I filled it with interpolation and fill the rest of the values with mean.

```
X = np.array(data1.drop(['stress'],axis = 1))
y = np.array(data1['stress'])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
y_pred_lr,LR_model = logisticregression(X_train,y_train,X_test,y_test)
metrics(y_pred_lr, y_train,'logisticregression')
```

```
train score: 0.5685177786656013
test score: 0.5223642172523961
logisticregression
confusion_matrix:
[[ 310  361   14]
 [ 117 1067   29]
 [  33  526   46]]
---------------------------------
classification_report:
              precision    recall  f1-score   support

         0.0       0.67      0.45      0.54       685
         1.0       0.55      0.88      0.67      1213
         2.0       0.52      0.08      0.13       605

    accuracy                           0.57      2503
   macro avg       0.58      0.47      0.45      2503
weighted avg       0.57      0.57      0.51      2503

---------------------------------
accuracy_score:
0.5685177786656013
---------------------------------
f1_score:
0.5067786811038061
```

The f1 score is 0.50and the accuracy is 0.56. Obviously, interpolation is better than mean.

2.4 Discussion

Method 2 has a better performance. Linear interpolation is an approximate calculation method that uses the proportional relationship to find other values of the unknown function according to a set of known values of the independent variables of the unknown function and its corresponding function values. method. It finds the unknown point according to the linear formula of the known point, and the accuracy is higher.

Answer 3) (25 points)

Share results (F1score, precision, recall, and accuracy score) from at least 2 models in your report.

3.1 Logistic Regression

```
metrics(y_pred_lr, y_train,'Logistic Regression')
```

```
Logistic Regression
confusion_matrix:
[[ 310  361   14]
 [ 117 1067   29]
 [  33  526   46]]
--------------------------------
classification_report:
              precision    recall  f1-score   support

         0.0       0.67      0.45      0.54       685
         1.0       0.55      0.88      0.67      1213
         2.0       0.52      0.08      0.13       605

    accuracy                           0.57      2503
   macro avg       0.58      0.47      0.45      2503
weighted avg       0.57      0.57      0.51      2503

--------------------------------
accuracy_score:
0.5685177786656013
--------------------------------
f1_score:
0.5067786811038061
```

3.2 Decision Tree

```
metrics(y_pred_lr, y_train,'Decision Tree')
```

```
Decision Tree
confusion_matrix:
[[ 310  361   14]
 [ 117 1067   29]
 [  33  526   46]]
--------------------------------
classification_report:
              precision    recall  f1-score   support

         0.0       0.67      0.45      0.54       685
         1.0       0.55      0.88      0.67      1213
         2.0       0.52      0.08      0.13       605

    accuracy                           0.57      2503
   macro avg       0.58      0.47      0.45      2503
weighted avg       0.57      0.57      0.51      2503

--------------------------------
accuracy_score:
0.5685177786656013
--------------------------------
f1_score:
0.5067786811038061
```

3.3 Random Forest

```
metrics(y_pred_RF, y_train,'Random Forest')
```

```
Random Forest
confusion_matrix:
[[ 459  208   18]
 [  77 1060   76]
 [  24  332  249]]
---------------------------------
classification_report:
              precision    recall  f1-score   support

         0.0       0.82      0.67      0.74       685
         1.0       0.66      0.87      0.75      1213
         2.0       0.73      0.41      0.53       605

    accuracy                           0.71      2503
   macro avg       0.74      0.65      0.67      2503
weighted avg       0.72      0.71      0.69      2503

---------------------------------
accuracy_score:
0.706352377147423
---------------------------------
f1_score:
0.6939954927655633
```

Answer 4) (20 points)

4.1 Top 5 Features

```
import eli5
from eli5.sklearn import PermutationImportance
def FeatureImportanceForPairs(model):
  # Find the best pairs of features which are useful for making the prediction.
  # HINT: Can be done in a model agnostic way through pair generation and ranking.
  # YOUR CODE HERE
    perm = PermutationImportance(model).fit(X_test, y_test)
    print('feature importances:',perm.feature_importances_)
    print('feature importances std:',perm.feature_importances_std_)
    return eli5.show_weights(perm,feature_names=data1.columns.drop('stress').tolist())

FeatureImportanceForPairs(RF_model)
```

```
feature importances: [ 0.04792332  0.03738019  0.00415335  0.03769968  0.02779553  0.01501597
  0.00319489  0.02939297  0.0284345  -0.0086262   0.00095847 -0.00255591]
feature importances std: [0.0109748  0.01013339 0.00511182 0.01003215 0.00411633 0.00576852
 0.0049495  0.00594284 0.00592563 0.00127796 0.00162908 0.00127796]
```

Weight	Feature
0.0479 ± 0.0219	HR
0.0377 ± 0.0201	AVNN
0.0374 ± 0.0203	interval in seconds
0.0294 ± 0.0119	TP
0.0284 ± 0.0119	ULF
0.0278 ± 0.0082	SDNN
0.0150 ± 0.0115	RMSSD
0.0042 ± 0.0102	NNRR
0.0032 ± 0.0099	pNN50
0.0010 ± 0.0033	LF
-0.0026 ± 0.0026	HF
-0.0086 ± 0.0026	VLF

I processed features by PermutationImportance in eli5. In this function, the features will be instead by random values, so that the features can be checked for the importance of the prediction. If nothing changes after altering its value, that feature is insignificant.

So, by this way, I chose the top five features that influence the result of the random forest: HR, AVNN, the interval in seconds, TP, and ULF.

4.2 Most relevant pair of features

```
top5 = ['HR','AVNN','interval in seconds','TP','ULF']
while top5:
    for i in top5:
        top5.remove(i)
        for j in top5:
            data_tem = data1[i] + data1[j]
            X_tem = np.array(data_tem)
            y_tem = np.array(data1['stress'])
            X_train_tem, X_test_tem, y_train_tem, y_test_tem = train_test_split(X, y, test_size = 0.2)
            print('Feature Pair:',i,j)
            randomforest(X_train_tem,y_train_tem,X_test_tem,y_test_tem)
            print('----------------------------------')
```

```
Feature Pair: HR AVNN
Random Forest
train score: 0.6404314822213344
test score: 0.5111821086261981
----------------------------------
Feature Pair: HR interval in seconds
Random Forest
train score: 0.6308429884139033
test score: 0.5271565495207667
----------------------------------
Feature Pair: HR TP
Random Forest
train score: 0.630043947263284
test score: 0.5447284345047924
----------------------------------
Feature Pair: HR ULF
Random Forest
train score: 0.6448262085497403
test score: 0.5207667731629393
----------------------------------
Feature Pair: interval in seconds AVNN
Random Forest
train score: 0.6456252497003596
test score: 0.5159744408945687
----------------------------------
Feature Pair: interval in seconds TP
Random Forest
train score: 0.6372353176188573
test score: 0.5527156549520766
----------------------------------
Feature Pair: interval in seconds ULF
Random Forest
train score: 0.6256492209348782
test score: 0.5670926517571885
----------------------------------
Feature Pair: ULF AVNN
Random Forest
train score: 0.6452257291250499
test score: 0.5
----------------------------------
Feature Pair: ULF TP
Random Forest
train score: 0.6284458649620456
test score: 0.5591054313099042
----------------------------------
Feature Pair: AVNN TP
Random Forest
train score: 0.6404314822213344
test score: 0.5047923322683706
----------------------------------
```

So, the most significant pair of features is ‘interval in seconds’ and ‘ULF’, which have an accuracy of 0.62 in the training set and an accuracy of 0.56 in the testing score.

In the measurement of mental stress, the greater the pressure, the more nervous the mind, the faster the heartbeat, and the more obvious the ultra-low frequency signal of the heartbeat. In this way, it is possible to predict whether an individual has greater mental stress by measuring these data, such as HR, AVNN, ULF and other data. It matches the result of the code.

Answer 5) (10 points)

A random forest is a classifier that is built in a random fashion and consists of multiple decision trees. The category of its output is determined by the mode of the category output by each tree.

Randomness is mainly reflected in two aspects:(1) When training each tree, select a dataset with the same size as N from all training samples (the number of samples is N) for training (that is, bootstrap sampling); (2) At each node, a subset of all features is randomly selected to calculate the optimal segmentation method.

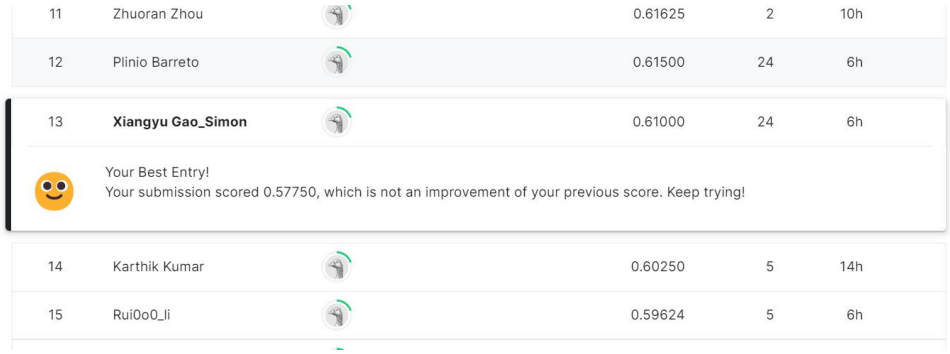
It has good performance and great advantages compared with other algorithms. Random forests can handle very high-dimensional data without feature selection. After training, the random forest can give which features are more important. The training speed is fast, and it is easy to make a parallel method. During the training process, the influence between features can be detected. For imbalanced datasets, random forests can balance the error. When there is a classification imbalance, random forests can provide an effective way to balance the errors in the dataset. The random forest has a strong anti-overfitting ability. So, it has the best performance in this classification task.

Logistic regression is widely used in regression problems and works well, but it also has some shortcomings, such as not being able to solve nonlinear problems, being sensitive to multicollinear data, and being difficult to deal with data imbalance problems, etc.

Decision trees work by finding interactions between variables. But each split of the tree reduces the dataset. Deliberately creating divisions will potentially introduce bias. Applying the greedy strategy to the decision tree variance, finding the correct starting point of the tree affects the final result. That is to say, small changes in the early stage will have a big impact later. Also, perturb a little, change a little value, and the decision tree changes. We want our classifier to be robust to noise, so this is not what we expect.

Answer 6) (5 points)

Describe the model that gets the best Kaggle metrics accuracy for you.



My best f1 score is 0.61 by using the random forest with the 54 estimators, 47 max_depth, and 0.005 min_weight_fraction_leaf.