KEIGEL CHALLENGE

The goal of this exercise is to prepare a predictive model with whichever Python library/resource (e.g. sklearn, theano, keras, tensorflow, xgboost, etc) using the Apnea-ECG Database. This dataset has been described in T Penzel, GB Moody, RG Mark, AL Goldberger, JH Peter. Computers in Cardiology 2000;27:255-258

- **35 subjects, 300 samples each, totaling 10500 instances (rows in X)**
- **Instances 1-300 correspond to subject #1, 301-600 to subject #2, …**
- **Each row is a feature vector corresponding to a one-minute ECG segment manually labeled with 0 (no apnea), 1 (apnea) (Y)**
- **10 features have been engineered from raw ECG signals based on relevant literature.**

The features used in our experimentation were all metrics based around RR intervals. An RR interval is defined as the time between two consecutive R peaks.

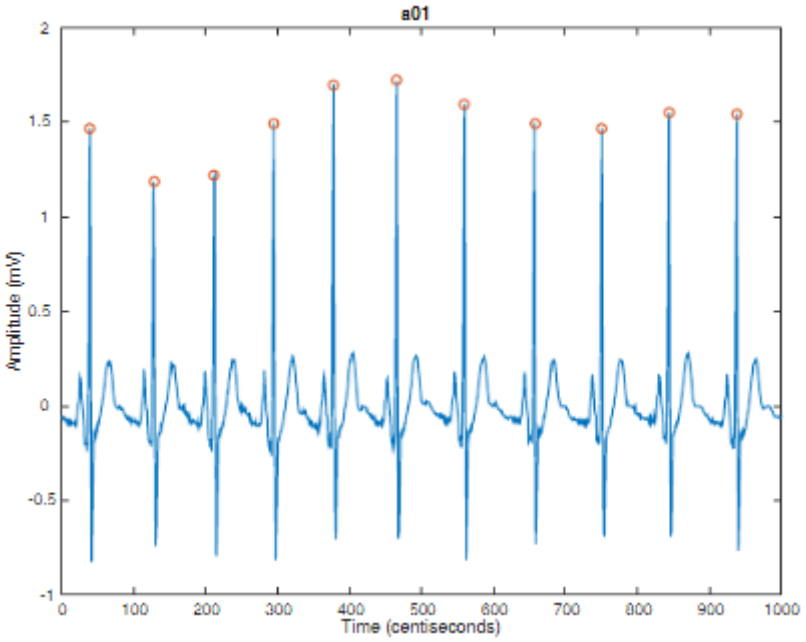


Figure 1- Ten seconds of ECG data from the Phys-ioNet Apnea-ECG database

0	**Mean RR interval**
1	**Standard deviation of the set of RR intervals**
2	**The NN50 measure (variant 1)**
3	**The NN50 measure (variant 2)**
4	**The pNN50 measure (variant 1)**
5	**The pNN50 measure (variant 2)**
6	**The SDSD measure**
7	**Inter-quartile range**
8	**The RMSSD measure**
9	**Median RR interval**

In Table 1, we report classification accuracy (percentage of correctly labeled test instances among all test instances) used in ours experiment. We also compare these accuracy percentages to similar scores reported for different classifiers. We have used f1_score as metric. F1 score is the weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0

Table 1

	F1_SCORE				
Model	**Train**	**Test**	**Preprocess**	**Selection**	**Time**
Random Forest	**0.77**	**0.79**	**No** (-0.66 SC)	**No**	**9:1min**
GradientBoost	**0.76**	**0.77**	**StandardScaler**	**No**	**21.6min**
DNN (400 epochs)	**-**	**0.74**	**No**	**No**	**59min**
Random Forest	**0.69**	**0.71**	**No**	**Yes**	**6:4min**
AdaBoost	**0.67**	**0.69**	**No** (-0.43 SC)	**No**	**5:6min**
Decision Tree	**0.59**	**0.69**	**No**	**No**	**1.5s**
MLP Classifier	**0.62**	**0.68**	**StandardScaler**	**No**	**145.8min**
SVM (kernel rbf)	**0.67**	**0.68**	**StandardScaler**	**No**	**3:9m**
GradientBoost	**0.76**	**0.64**	**StandardScaler**	**Yes**	**20.7min**
SVM (kernel Linear)	**0.55**	**0.62**	**StandardScaler**	**No**	**93.3m**
SVM (kernel rbf)	**0.57**	**0.57**	**StandardScaler**	**Yes**	**3:9m**
KerasClasifier (DNN)	**-**	**0.52**	**StandardScaler**	**No**	**9 min**
PCA, Linear SCV	**-**	**0.48**	**Normalize**	**No**	**7.88 s**
Catboot	**0.23**	**0.02**	**No**	**No**	**5.1s**

After this studio we realize that the best model for this problem is Random Forest with a score of 0.79%:

RAIN FOREST

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import f1_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import ParameterGrid
from sklearn.model_selection import GridSearchCV

X_train = np.load(open('X_train.npy','rb'))
Y_train = np.load(open('Y_train.npy','rb'))
X_test = np.load(open('X_test.npy','rb'))
Y_test = np.load(open('Y_test.npy','rb'))

# Preprocessing data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_train)


# We create a instance of model.
Estimator_RForest = RandomForestClassifier()

# Now, we are going to use a grid search cross-validation to explore
combinations of parameters.
param_grid = {'n_estimators': [80,85,100],
              'criterion':['gini'],
              'max_features':['auto'],
              'min_samples_split':[2,3,4],
              'max_depth': range(13,15),
              'min_samples_leaf':[1,2,3]}


Grid_RForest=
GridSearchCV(Estimator_RForest,param_grid,cv=10,scoring='f1',
verbose=2)
Grid_RForest.fit(X_train,Y_train)

# Once it has been fitted, we get several parameters.
print("ParameterGrid: ",'\n',list(ParameterGrid(param_grid)),'\n')
print("Best estimator: " , Grid_RForest.best_estimator_,'\n')
print("Best Score: ",round(Grid_RForest.best_score_,2))
print("Best Parameters ",Grid_RForest.best_params_)


# Now, we came back fit it Best_Grid_estimator with.

Best_Grid_estimator_RForest = Grid_RForest.best_estimator_
Best_Grid_estimator_RForest.fit(X_train,Y_train)

# We use best_estimator attribute and predict method to predict test
data.

Y_pred = Best_Grid_estimator_RForest.predict(X_test)

# and finally, the score test
print("f1_score:",round(f1_score(Y_test, Y_pred),2))
```

After tuning several times, the Random Forest model, we conclude with these best parameters:

```
Best estimator:  RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=14, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=2, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=85, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

Best Score:  0.77
Best Parameters  {'criterion': 'gini', 'max_depth': 14, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 2,
'n_estimators': 85}
f1_score: 0.7886944818304172
```

In this exercise we are working in a space that has been already processed their features, that is, a vector of a little more than ten features that condense in its definition and calculation a temporary structure. Between on feature and the next there aren´t a structure or correlation.

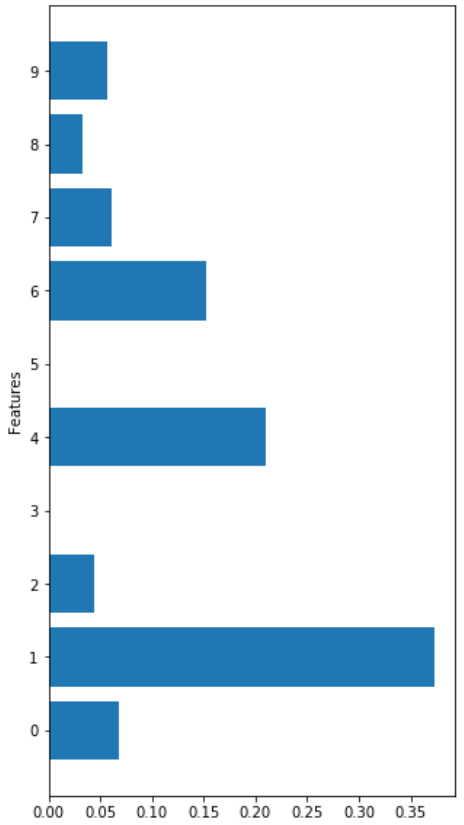
Anyway, to prove this, and to try to look for a best score we will try a little feature selection and to check if our assertion is true

<u>Feature Selection:</u>

We have tested different technics as "features selection" trying to find best scores and shorter training times but we didn´t find a model which improve before tests probably because the original data set was already refined (remember they selected 10 features very significant).

Given this context, we have used the "feature_importances_" of the decision tree model to see the more influential features and we realized that every time that we tried to execute this command always our results are different, that is to say that there is not correlation between the features.

In Spite of this, we had tried this method in some models (Random Forest, Gradient Boost and SVM) and all the scores has been far worse than without features selection



And the code:

```
#plotting the importance of features
# features=["0","1","2","3","4","5","6","7","8","9"]
caract=X_train.shape[1]
plt.figure(figsize=(5,10))
plt.barh(range(caract),Best_Grid_estimator_DTree.feature_importances_)
plt.yticks(np.arange(caract),features)
plt.ylabel('Features')
```

```
# deleting the columns (less meaningful features)

X_train_main_features =np.delete(X_train,[0,3,5,6,8],axis = 1)
X_test_main_features =np.delete(X_test,[0,3,5,6,8],axis = 1)
```

We use a selection method based in median:

```
mediana = np.median(Best_Grid_estimator_DTree.feature_importances_)
values = Best_Grid_estimator_DTree.feature_importances_

print(mediana)

list = []
for i in range(caract):
   if values[i]>mediana:
      list.append(i)

print("The main Features: ",list)
```

<u>Normalization:</u>

Also, to try to find best scores we apply normalization methods to our models. In this case we use the standard Scaler method. StandardScaler Standardize features by removing the mean and scaling to unit variance.

We have applied this method to several experiments and many of their scores were improved with this method but no one improved random forest classifier.

DEEP LEARNING

We were very interested in checking deep learning models in this exercise, but from the first moment we tried to use them we realized that the scores were very bad. The reason is the same because we had very bad results with feature selection: the data set was already refined and between on feature and the next there aren´t a structure or correlation, and there is not a temporal sequence that justifies to use CNN, LSTM or GRU models.

However, we have tried DNN model, where we got a 0,73 score that compare with the rest is pretty good. We though that it's due to DNN model is able to generalize anything model.