Automatic detection of Atrial Fibrillation episodes

Primary Topic: TS, Secondary Topic: SEMI

Course: Data Science **Additional Topics** – Group: 95 – Submission Date: 18-04-2021

Ravi Baligudam (S2352532)
University of Twente
Data Science & Technology
r.t.baligudam@student.utwente.nl

Jeroen Mulder (S2353393)
University of Twente
Civil Engineering and Management
j.d.mulder-1@student.utwente.nl

ABSTRACT

Atrial Fibrillation (AF) occurs as a complication postoperatively form cardiac surgery. On long term serious complications can lead to stroke and heart failure. AF is defined as a period of at least 30 seconds in which an irregular ventricular rate and P peaks are absent. Manual detection of AF in ECG record is time-consuming, especially in large datasets. Therefore, for faster analysis, an automatic AF detection algorithm is developed in this paper. Firstly, the ECG data is converted in a 30 second intervals and features like minimum, maximum, median, mean, and standard deviation have been extracted from the R-R Intervals. Then, the AF and Non-AF data has been balanced to a 50/50 ratio. Different types of Machine Learning Models such as Logistic Regression, Decision Trees, Random Forests, Artificial Neural Networks and Stacking models are developed based on the features extracted. Each model was evaluated using Accuracy, Precision, Recall and AUC Score. The stacking model has proven to be the best model for detecting AF episodes with an accuracy of 97.2 percent.

KEYWORDS

Atrial Fibrillation, R-R Intervals, Machine Learning, Stacking.

INTRODUCTION

People with atrial fibrillation, in short AF, suffer from an abnormal heart rhythm. As can be seen in Figure 1, these heart beats have alternating frequencies within a short time frame. Despite from the ECG readings, people mostly have no clear symptoms which means that AF episodes can pass by unnoticed. However, this arrhythmic beating of the heart can have some serious complications. Atrial fibrillation is associated with an heightened prospect of having heart failure, strokes, blood cloths and dementia [1]. It is therefore important to detect AF episodes as early as possible. By analyzing the ECG readings, a model can be created which detects an episode automatically. This will help to diagnose patients in an earlier stage.

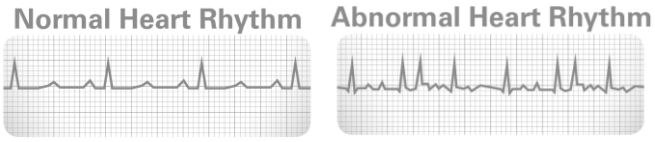

Figure 1 – Characteristics heart beat AF

BACKGROUND

Detecting AF episodes with machine learning is not new. There are already various studies done on this subject. Siontis et al., made a model using a convolutional network [2]. This model was able to predict AF episodes with an Sensitivity of 79% and a specificity of 79,5%. M. Lown et al., created a model with a higher overall accuracy. With a SVM algorithm, this study managed to achieve a sensitivity of 99.2% and a specificity of 99.5% [3]. Another research has been done with multiple machine learning methods. M. Shen et al., has tested random forest, svm, cnn, decision tree and stacking. The results were in this order from low to high in accuracy. The stacking method performed the best with an accuracy of 99.1%. This was significantly higher than the second best; the decision tree with an accuracy of 96.2%. In this research automatic as well as manual feature extracting is used. The manual features are bade on the R-R intervals and contain standard statistics such as mean, variance and interval range [4].

APPROACH

In this chapter, the steps towards the automatic detection system are described. In the first part, the preparation of the data is elaborated. In the second part, the training and testing methods are explained.

Description dataset

For creating an automatic atrial fibrillation detector, ECG data is used from the Erasmus Medical Centre in Rotterdam. This data is acquired from patients which had a CABG surgery at most 10 days ago. Right after such surgery, atrial fibrillation is more likely to occur. This data is useful for collecting normal heart beats as well as episodes of atrial fibrillation from the same patient. The readings of the heart beat is manually assessed to distinguish between the arrhythmia and normal heart beats.

When irregular heartbeats are present for 30 seconds, it is considered an episode of atrial fibrillation. Therefore the readings are divided in samples of 30 seconds. If more than 75% within this sample consists of irregular heartbeats, it is given a label 1 (AF episode). When heart beats are regular or are between the threshold of 75%, it is given the label 0. These labels are assigned manually by an physician by looking at the patterns of the ECG. When the data is incomplete or consists of errors, a label of -1 is given.

An episode of atrial fibrillation can best be recognized by analyzing the R peaks. Therefore the raw ECG signal is converted into R-R intervals with automatic peak detection. In Figure 2, the R-R interval of a normal heart beat is shown. The preprocessed data can later be used to subtract features directly. The dataset used in this report consists of files with R-R intervals with a timecode of 1Hz. Another set of files contains the classification labels per 30 second interval. These two sets of data are connected by its timecode.

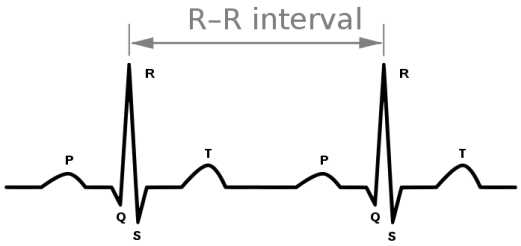

Figure 2 – R-R interval of heart beats

Preprocessing data

The provided dataset is only partly preprocessed. The data with the R-R intervals still contains faulty values. Sometimes the peak detector has mistaken the actual peak which results subsequently in a very high and low value. 30 second samples which contain unnatural values are deleted as this influences the performance of the classification in a later stage. Properly registered AF episodes can contain strong fluctuating values, so a threshold for unnatural values need to be chosen carefully. When the sample contains values which are 0.5 times lower or 1.5 times higher than the median, this sample is deleted. By manually analyzing the data, this threshold seems to be best balance between filtering errors and rejecting real episodes.

The dataset exists of 804 files with labeled 30 seconds intervals. In the data, the 0 labels are overrepresented. To balance the labels within the data, for each file the ratio is set to 50/50. If for example a file contains 50 labels of 1, the labels of 0 will be reduced to this same number. From this 0 labels, the remaining samples will be deleted randomly. If one file only contains labels with a 1, this file is not used. The same applies for files with only labels of zero. In this way, the model will be trained better in recognizing the transition between a normal and abnormal state of the heart. After cleaning the data according to the described filters, 13.758 samples are with label 0 and also 13.758 of label 1. This gives a total of 27.516 samples.

Subtraction features

The R-R intervals are varying frequencies whereof statistics can be subtracted. These features can later be used to train the model with a machine learning algorithm. The statistics used for the features are: minimum, maximum, median, mean, standard deviation and number of values. These characteristics are considered to add the most valuable information without adding strong correlating features. When converting the data into frequency domain, a clear difference can be seen between rhythmic and arrhythmic heart beats. In Figure 3, the distribution of normal heart beat is shown.

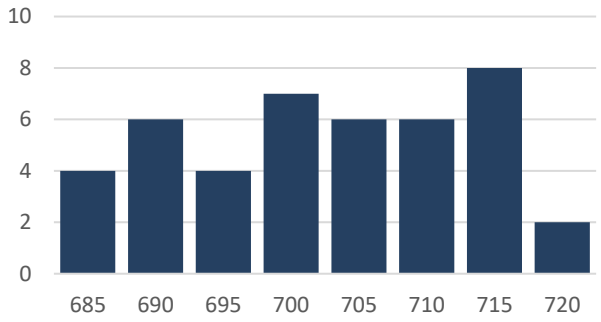

Figure 3 – Example normal distribution heart beat

In Figure 4, the distribution of an AF episode is displayed. It can be seen that all the mentioned statistics clearly differ from the normal heart beat. The python script of the data preparation can be found in Appendix I.

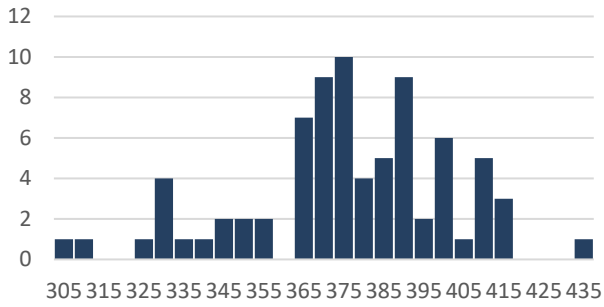

Figure 4 – Example AF distribution heart beat

Model creation

This sub section covers the machine learning models used in this project. Since this is a classification problem, the models are measured based on a different metrics for classification such as Accuracy, Precision, Recall and AUC-ROC Score.

The dataset is divided in a training set and testing set in 60:40 ratio. After this, different machine learning models are applied, such as: Logistic Regression, Decision Trees with AdaBoost Classifier, Random Forest Classifier, Artificial Neural Network and Stacking Ensemble Model inspired from M. Shen et al [4]. The Stacking Ensemble Model contains of 2 level models. The first level is known as “base-models” and the second level is known as “meta-model”. The first layer is composed of multiple base learners. The input is the original training set, and the second layer model is retrained by using the output of the first layer learner as a training set and predicted using the test dataset. In the stacking model, a 5-fold cross validation is used in order to avoid overfitting for the train set. At first the base learners are trained on fold2-fold5 to train base model 1 and predict using fold1 and so on. After training the model, the predictions from the base models are used as features for the meta-model. In this project, the base learners used are Decision Trees with AdaBoost Classifier, Random Forest Classifier, Gradient Boosting Classifier and Extra Trees Classifier and the meta model is XGBoost Classifier. All the machine

learning models mentioned above were applied on the dataset based on Hyper Parameter tuning. "GridSearchCV" is applied for parameter tuning for the models using cross validation technique on the training set. Once the best parameters were obtained, the model is trained and tested on the training and testing dataset. Different Evaluation Metrics were used such as Accuracy, Precision, Recall and AUC-ROC Score. The AUC-ROC score helps in understanding the degree of separability of classes by the models.

Accuracy (ACC): In binary classification, Accuracy is defined as the proportion of the correct predictions among the total number of predictions.

$$\text{ACC: } \frac{TP+TN}{TP+TN+FP+FN}$$

Where TP = True Positives, TN = True Negatives, FP = False Positives and FN = False Negatives.

Precision (P): The precision measures how precisely the classifier identifies positive examples by avoiding the incorrect identification of negative examples as positive.

$$\text{P: } \frac{TP}{TP+FP}$$

Recall (R): The recall measures the fraction of examples that were correctly identified to be positive among all examples that are actually positive.

$$\text{R: } \frac{TP}{TP+FN}$$

RESULTS

In Table 1, the table shows the various results obtained for the models created using the R-R interval dataset to predict the AF episodes whether they are abnormal or normal.

Table 1 – Test results model

Models	**ACC**	**P**	**R**	**AUC-ROC**
Logistic Regression	93	92.9	93.3	0.963
Decision Trees with AdaBoost	93.2	93.9	92.7	0.932
Random Forest	95	94.9	95.6	0.975
Artificial Neural Network	95.3	93.8	97.1	0.987
Stacking	97.2	95.6	98.8	0.972

The Accuracy, Precision and Recall are given in percentages. The AUC-ROC is a score which indicates that the closer the value is to 1, the better the model understands the separation of the classes. The python script for training and testing the model can be found in Appendix II.

DISCUSSION

In this section, we will be interpreting the results obtained by the models and also discuss some of the limitations that are faced while working with the dataset.

From Table 1, it can be seen that most models used in this project gives a good accuracy. Logistic Regression gives a 93% accuracy which is a good accuracy but may not be the best machine learning model to be used for detecting the AF episodes. Decision tree with AdaBoost classifier barely improves the performance compared to Logistic Regression, as it also gives approximately 93%. Random Forests and Artificial Neural Networks improve the performance to an accuracy of 95% but still fail to be the best model. For this project, Stacking did not need much parameter tuning, and still an accuracy of 97.2% is achieved. Also the precision and recall are the highest compared to all other models.

Nevertheless, there were a few limitations for this project. The dataset consists of preprocessed data which limits the possibilities in controlling the outcome. Unnatural low and high R-R intervals are filtered out but small errors in the values might still be in the data if these are close to the used threshold. In case of a raw ECG signal, more detailed information could be subtracted from the characteristics of the signals' waveform. Also, the data is manually assessed for AF episodes. Some episodes in the are lasting for many contiguous hours. For such cases it is likely that in this time also 30 seconds are present whereby the heart beat rhythm actually should be classified as normal. This could explain why the accuracy of the models is not closer to 100%. The differences between the statistics of an AF episode and normal heart beats are very obvious. A machine learning algorithm should distinguish these two classes with ease. Another explanation for a lower accuracy could be the threshold of 75% arrhythmic heart beats within the 30 second interval. If for example a sample contains 60% arrhythmic heart beats, the statistics may be close to an AF episode but is still labeled as a normal heart rhythm.

For future research, it is recommended to make a machine learning model from the raw ECG readings. As mentioned earlier, this will allow for more unique features. Also, this algorithm can then be applied directly on ECG device without extra steps. For the algorithm of this report, the ECG data must first be converted into R-R intervals. Ideally the same peak detection system (Synescope) should be used as the data preparation algorithm is set to this output.

From the data it is not clear which patients are monitored. In order to have a detection system which can be widely used, people with different characteristics should be included in the training of the model. Important characteristic which influences the features could be for example ethnicity, sex and age.

CONCLUSION

In general, Atrial Fibrillation evaluation can be very time-consuming. Machine learning models help in such task to automatically learn from the data and predict on unseen data. For developing a model which can detect AF episodes automatically, a dataset is used which contains the R-R intervals of a patient's heart beat. These R-R intervals were derived from the raw ECG signal with peak detection. For each 30 seconds sample, a label was manually assigned whether this signal is normal or arrhythmic. Some samples contained faulty values and are filtered out of the dataset. Also, the labels with normal heart rhythms were overrepresented. The labels are balanced by removing the labels without AF episodes to a 50/50 ratio. From this cleaned data, features are subtracted for each 30 seconds sample with R-R intervals. These statistical features are: minimum, maximum, median, mean, standard deviation and number of values. To automatically predict the AF episodes, machine learning models are developed. Out of different models, stacking model has proven to be the best approach for AF detection. This model gives an accuracy of 97.2%, a precision of 95.6% and a recall of 98.8%. This is a relatively high score for detecting a single AF episode. Most of the times an episode lasts longer than 30 seconds. For such cases the chance is even higher that the atrial fibrillation will be detected. One of the main limitations of this study is that the dataset consists of preprocessed data. This limits the possibilities in feature subtraction which may decrease the potential accuracy of the prediction model.

REFERENCES

[1] American Heart Association, "Why Atrial Fibrillation (AF or AFib) Matters," 2016. https://www.heart.org/en/health-topics/atrial-fibrillation/why-atrial-fibrillation-af-or-afib-matters.

[2] K. C. Siontis, X. Yao, J. P. Pirruccello, A. A. Philippakis, and P. A. Noseworthy, "How Will Machine Learning Inform the Clinical Care of Atrial Fibrillation?," *Circ. Res.*, pp. 155–169, 2020, doi: 10.1161/CIRCRESAHA.120.316401.

[3] M. Lown *et al.*, "Machine learning detection of atrial fibrillation using wearable technology," *PLoS One*, vol. 15, no. 1, pp. 1–9, 2020, doi: 10.1371/journal.pone.0227401.

[4] M. Shen, L. Zhang, X. Luo, and J. Xu, "Atrial fibrillation detection algorithm based on manual extraction features and automatic extraction features," *IOP Conf. Ser. Earth Environ. Sci.*, vol. 428, no. 1, 2020, doi: 10.1088/1755-1315/428/1/012050.

Appendix I – Python script preprocessing data

```
import numpy as np
from numpy import savetxt
import pandas as pd
import os


def create_features(sample, label):
    # Clean data
    if np.max(sample) >= 1000:
        label = -1
    if len(sample) < 20:
        label = -1
    if np.min(sample) < 0.5 * np.median(sample) or np.max(sample) > 1.5 *
np.median(sample):
        label = -1

    # Calculate features
    results = np.array(
        [label, len(sample), np.min(sample), np.max(sample),
np.median(sample), np.mean(sample), np.std(sample)])

    return results


def combine_data(no_file):
    # Locate specific file
    no_file = 2  # Number of data file from database

    local_dir = os.path.dirname(__file__)  # Set directory same as python
file
    path_data = (local_dir + '\Data\ECG_data\Data' + str(no_file) + '.txt')
    path_label = (local_dir + '\Data\Class\Control' + str(no_file) +
'.txt')

    # Load data
    data = pd.read_csv(path_data, header=None, usecols=[0, 1], sep=' ')
    label = pd.read_csv(path_label, header=None, delim_whitespace=True)

    # Add extra empty row to data (looking forward in loop)
    label = label.append(pd.DataFrame(['23:59:59', -1]).T)
    data = data.append(pd.DataFrame(['00:00:00', 0]).T)

    # Combine data and classes
    j = 0
    sample = np.empty(0)
    features = np.zeros((label.shape[0] - 1, 7))

    for i in range(0, data.shape[0] - 1):
        # Make exception when time has transition from PM to AM
        if label.iloc[j + 1, 0][:-4] < '00:00:30' and data.iloc[i, 0] >
'23:59:30':
            transition_time = True
        else:
            transition_time = False

        # Combine all values of one 30 sec period
        sample = np.append(sample, data.iloc[i, 1])
```

```
        # Calculate features when sample is complete
        if data.iloc[i + 1, 0] >= label.iloc[j + 1, 0][:-4] and
transition_time == False:
            properties = create_features(sample, label.iloc[
                j, 1]).T  # If row with values is filled, calculate the
properties of this sample
            features[j, :] = properties  # Add features properties
according to 30 sec label period
            sample = []  # Reset current list
            j = j + 1  # Go to next 30 sec sample

    # Remove unusefull data
    features = features[np.logical_not(features[:, 0] == -1)]  # Remove all
labels with -1 (unvalid data)
    features = features[:-1, :]  # Remove empty row at end

    return features


# Collect features from all files
total_files = 804

df = np.empty((0, 7), int)
for no in range(1, total_files):
    features = combine_data(no)
    df = np.append(df, features, axis=0)
    print(round(no / total_files * 100, 1), '% complete')  # Update
progress

# save features as csv file
savetxt('features training.csv', df, delimiter=';')
```

Appendix II – Python script training and testing model

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
ExtraTreesClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb
from vecstack import stacking
from sklearn.model_selection import cross_validate, GridSearchCV,
train_test_split, KFold, RepeatedKFold
from sklearn.metrics import accuracy_score, precision_score, recall_score,
roc_auc_score
from keras.models import Sequential
from keras.layers import Dense
import tensorflow as tf

features = pd.read_csv(r"/content/drive/MyDrive/DS Project/processed
features - Complete.csv", delimiter=";" ,names=["Label", "Number of
Samples", "Min", "Max", "Median", "Mean", "SD"])
features["Label"] = features["Label"].astype(int)
pd.options.display.float_format = "{:,.2f}".format

#converting into train-test set
cols = ["Min", "Max", "Median", "Mean", "SD"]
X = features[cols]
y = features["Label"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=None)

#Model Creation
# 1. Logistic Regression
#with hyperparameter tuning
LR = LogisticRegression()

#Search grid for optimal parameters
lr_param_grid = {"penalty": ["l1", "l2", "elasticnet", "none"],
                 "solver": ["newton-cg", "lbfgs", "liblinear", "sag",
"saga"],
                 "C": [0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 1.0]}

gsLR = GridSearchCV(LR, param_grid = lr_param_grid, cv=10,
scoring='accuracy', n_jobs= -1, verbose = 1)
gsLR.fit(X,y)
lr_best = gsLR.best_estimator_
scoring = {'accuracy': 'accuracy', 'precision': 'precision', 'recall':
'recall', 'auc': 'roc_auc'}
modelCV = LogisticRegression(C= 1.0, penalty='l1', solver='liblinear')
results = cross_validate(modelCV, X, y, cv=5,
scoring=list(scoring.values()),
                         return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print(modelCV.__class__.__name__+" %s: %.3f (+/-%.3f)" %
(list(scoring.keys())[sc], results['test_%s' %
list(scoring.values())[sc]].mean(),

results['test_%s' % list(scoring.values())[sc]].std()))
```

```
# 2. Decision Trees with AdaBoost Classifier
#with hyperparameter tuning
DTC = DecisionTreeClassifier()
adaDTC = AdaBoostClassifier(DTC, random_state=7)

#Search grid for optimal parameters
ada_param_grid = {"base_estimator__criterion" : ["gini", "entropy"],
              "base_estimator__splitter" :   ["best", "random"],
              "algorithm" : ["SAMME","SAMME.R"],
              "n_estimators" :[10,20,30,40,50],
              "learning_rate":  [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 1.5]}

gsadaDTC = GridSearchCV(adaDTC, param_grid = ada_param_grid, cv=10,
scoring='accuracy', n_jobs= -1, verbose = 1)
gsadaDTC.fit(X,y)
ada_best = gsadaDTC.best_estimator_
scoring = {'accuracy': 'accuracy', 'precision': 'precision', 'recall':
'recall', 'auc': 'roc_auc'}
modelCV = AdaBoostClassifier(DecisionTreeClassifier(criterion="entropy",
splitter='random'))
results = cross_validate(modelCV, X, y, cv=5,
scoring=list(scoring.values()),
                         return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print(modelCV.__class__.__name__+" %s: %.3f (+/-%.3f)" %
(list(scoring.keys())[sc], results['test_%s' %
list(scoring.values())[sc]].mean(),

results['test_%s' % list(scoring.values())[sc]].std()))

# 3. Random Forest
#with hyperparameter tuning
RFC = RandomForestClassifier()

#Search grid for optimal parameters
rf_param_grid = {"max_features": [1, 3, 10],
              "min_samples_split": [2, 3, 10],
              "min_samples_leaf": [1, 3, 10],
              "n_estimators" :[100,200,300],
              "criterion": ["gini", "entropy"]}

gsRFC = GridSearchCV(RFC,param_grid = rf_param_grid, cv=5,
scoring="accuracy", n_jobs= -1, verbose = 1)
gsRFC.fit(X,y)
rfc_best = gsRFC.best_estimator_
scoring = {'accuracy': 'accuracy', 'precision': 'precision', 'recall':
'recall', 'auc': 'roc_auc'}
modelCV = RandomForestClassifier(criterion="gini", min_samples_split=2,
min_samples_leaf=10, max_features=3, n_estimators=10)
results = cross_validate(modelCV, X, y, cv=5,
scoring=list(scoring.values()),
                         return_train_score=False)

print('K-fold cross-validation results:')
for sc in range(len(scoring)):
    print(modelCV.__class__.__name__+" %s: %.3f (+/-%.3f)" %
(list(scoring.keys())[sc], results['test_%s' %
list(scoring.values())[sc]].mean(),
```

```
results['test_%s' % list(scoring.values())[sc]].std()))

# 4. Artifical Neural Network
#defining the model
model = Sequential()
model.add(Dense(12, input_dim=5, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

#compile the keras model
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy', tf.keras.metrics.Precision(),
tf.keras.metrics.Recall(), tf.keras.metrics.AUC()])

#fit the keras model on the dataset
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=200,
batch_size=128)

#evaluate the keras model
_, train_acc, train_pre, train_rec, train_auc = model.evaluate(X_train,
y_train, verbose=0)
_, test_acc, test_pre, test_rec, test_auc = model.evaluate(X_test, y_test,
verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
print('Train: %.3f, Test: %.3f' % (train_pre, test_pre))
print('Train: %.3f, Test: %.3f' % (train_rec, test_rec))
print('Train: %.3f, Test: %.3f' % (train_auc, test_auc))

# 5. Stacking Model
#Put in our parameters for said classifiers
#Extra Trees Parameters
et_params = {
    'n_jobs': -1,
    'max_depth': 8,
    'min_samples_leaf': 2,
    'verbose': 0
}

#Gradient Boosting parameters
gb_params = {
    'n_estimators': 500,
    'max_depth': 5,
    'min_samples_leaf': 2,
    'verbose': 0
}

def param(clf, param):
  return clf(**param)

#Create 5 objects that represent our 4 models
lr = LogisticRegression(C= 1.0, penalty='l2')
rf = RandomForestClassifier(criterion="gini", min_samples_split=2,
min_samples_leaf=10, max_features=3, n_estimators=10)
et = param(ExtraTreesClassifier, et_params)
ada = AdaBoostClassifier(DecisionTreeClassifier(criterion="entropy",
splitter='random'))
gb = param(GradientBoostingClassifier, gb_params)

#Combine all into one model
models = [lr, rf, et, ada, gb]
```

```
#Create our OOF train and test predictions
L1_train, L1_test = stacking(models, X_train, y_train, X_test,
regression=False, n_folds=5)

#Using XGBoost as meta_model
gbm = xgb.XGBClassifier(
 n_estimators= 2000,
 max_depth= 4,
 min_child_weight= 2,
 gamma=0.9,
 subsample=0.8,
 colsample_bytree=0.8,
 objective= 'binary:logistic',
 nthread= -1,
 scale_pos_weight=1)

gbm.fit(L1_train, y_train)
y_pred = gbm.predict(L1_test)

score = accuracy_score(y_test, y_pred)
score1 = precision_score(y_test, y_pred)
score2 = recall_score(y_test, y_pred)
score3 = roc_auc_score(y_test, y_pred)
print(score)
print(score1)
print(score2)
print(score3)
```