Early Diagnosis of Parkinson’s Disease via keystrokes

Chin Min Tee
February 11, 2018

Definition

Project Overview

Parkinson’s disease sickened over 6 million individuals worldwide. The impact extended to their immediate families and friends, societies, medical communities and many more resources are not negligible.
This neurodegenerative disease is incurable but early discovery indisputably benefit the wellbeing of the patient and all involved. Diagnosis typically relies on neurologists but it is the individual patient (or close family member) who has to first discover that he or she is having the symptom of Parkinson’s disease. When the well-known symptom of movement disorder becomes obvious to the patient it typically has been five to ten years since the patient has the disease. More than half of the area affected in the brain are damaged and many episodes of falls and injuries have already occurred. It is certainly more stressful to care for a patient with Parkinson’s disease and physical injuries and some other form of damages like depression compare to just the disease itself.

A cost-effective test that can be done at home with simple and familiar apparatus would be one of the best solution for early detection.

Experiment has been conducted to use keyboard stroke as a mean to allow non-clinical personnel to make a guided diagnosis. This help captures the pre-motor phase of the disease, which could be years, or decades before the degeneration and the tell-tale symptoms of the failing motor control.

Inspired by the study 'High-accuracy detection of early Parkinson's Disease using multiple characteristics of finger movement while typing' recorded in physionet.org, I am eager to put machine learning to work to build a great classification model so that precious opportunity to start treating or researching the disease is not lost.

Problem Statement

The goal is to take in data sets provided by physionet.org for the study 'High-accuracy detection of early Parkinson's Disease using multiple characteristics of finger movement while typing' (Tappy keystroke data) and build a satisfactory binary classification model out of it. The model will be able to predict by using patient’s keystroke data input if he or she has Parkinson’s disease.

There is another project also hosted by physio.net: neuroQWERTY MIT-CSXPD. The way keystroke data was acquired is different from the Tappy project but the difference is in the soft wares and environments set up. The data collected by neuroQWERTY have the similar attribute that Tappy project collected. The preliminary exploration has shown that they can be used together with Tappy data set thereby increasing the total volume of available data.

Evaluation Metrics

Confusion matrix and F1 score will be used to evaluate models instead.

		Predicted	
		0	1
Actual	0	True Negative (**TN**)	False Positive (**FP**)
	1	False Negative (**FN**)	True Positive (**TP**)

Table 1. Confusion Matrix

Accuracy is intuitive and easy to understand. It is the correct predictions over the total predictions. However, it **<u>would not be suitable</u>** here since we have around 67 percent of data with positive Parkinson's disease. In this situation where majority belong to one of the two outcomes, the naïve model could achieve seemingly high performance by predicting everyone to have PD. There will be a lot of false positive, or Type 1 error. Even though it is preferable to err on Type I error than Type II error (False negative, result in missing treatment opportunity), there is better way to measure our model.

$$\boldsymbol{Accuracy} = \frac{\mathbf{TP + TN}}{(\boldsymbol{TP + TN + FP + FN})}$$

Precision is the correct positive predictions over all positive predictions. That is when a patient is predicted to have Parkinson's disease, how correct the prediction is. In this project, it is preferable to have high precision, which means there is low false positive, low rate of telling people falsely that they have the disease.

$$\boldsymbol{Precision} = \frac{\mathbf{TP}}{(\boldsymbol{TP + FP})}$$

Recall (also known as **Sensitivity**, or **True Positive Rate (TPR)**). When a patient is truly having Parkinson's disease, how often a model will predict such.   In this project, this is the key measurement. The higher the model could achieve a recall rate, the better the model can detect the disease if a patient does have the disease, thereby ensuring proper and prompt treatment.

$$\boldsymbol{Recall} = \textbf{Sensitivity} = \textbf{TPR} = \frac{\textbf{TP}}{(\boldsymbol{TP} + \boldsymbol{FN})}$$

Specificity explains that when a patient truly does not have Parkinson's disease, how often the model predicts such.

$$\boldsymbol{Specificity} = \frac{\textbf{TN}}{(\boldsymbol{TN} + \boldsymbol{FP})}$$

F1 score is a weighted average of precision (P) and recall (R).

$$\boldsymbol{F1} = \frac{\textbf{2} * \textbf{PR}}{(\boldsymbol{P} + \boldsymbol{R})}$$

FPR is False Positive Rate. When a patient truly has no Parkinson's disease, how often the model predicts there is Parkinson's disease.

$$\boldsymbol{FPR} = \frac{\textbf{FP}}{(\boldsymbol{TN} + \boldsymbol{FP})} = \textbf{1} - \boldsymbol{Specifity}$$

ROC (AUC) is Receiving Operating Characteristics curve and AUC is area under curve. ROC curve is a plot of true positive rate (TPR) vs false positive rate (FPR). The area under curve is an indicator on how well the model is. AUC approaching 1 is considered good and near zero is bad.

Due to the understanding that the majority of population will not have Parkinson's disease, the data will be imbalanced and the metric to measure a model will need to be agnostic of the underlying bias of the data set.

One of PRC and ROC/AUC big difference is that PRC does not use TN in its calculation, whereas ROC/AUC FPR(X axis) uses TN in its calculation. It is found that when there is imbalanced data involved where TN is huge, ROC does not reflect the true metric measurement. The ROC curve stays pretty much as before. The PRC will show degradation in performance of model since FP starts to grow. At the same time, PRC is not affected by the huge TN seen, thus it is more robust in the occasion of imbalanced data.

The major metric used here will be F1 score and PRC. ROC/AUC will also be used as reference. Same set of data is used throughout the modeling and it is still useful to see the ROC/AUC of different models.

Analysis

Data Exploration

There are 2 datasets from two different sources that we will combine to form a larger dataset. The two sources are both from physionet.org, one is ‘Tappy Keystroke Data’ (Tappy hereafter) and the other is ‘neuroQWERTY MIT-CSXPD Dataset’ (NQ hereafter).

In the Tappy dataset, there are two files that need to be zipped together to form pandas DataFrame for modeling. There are ArchivedUsers.zip (User files) and ArchivedData.zip (keystrokes data files pertaining to users in User files). The User file has 10 character code as filename that represent an individual user, that 10 character code is also present in corresponding keystroke data file for that user. A user have more than one keystroke data files.

The User file contains the following:

- Birth Year: Year of birth
- Gender: Male/Female
- Parkinsons: Whether they have Parkinson's Disease [True/False]
- Tremors: Whether they have tremors [True/False]
- Diagnosis Year: If they have Parkinson's, when was it first diagnosed
- Whether there is sidedness of movement [Left/Right/None] (self reported)
- UPDRS: The UPDRS score (if known) [1 to 5]
- Impact: The Parkinsons disease severity or impact on their daily life [Mild/Medium/Severe] (self reported)
- Levadopa: Whether they are using Sinemet and the like [Yes/No]
- DA: Whether they are using a dopamine agonist [Yes/No]
- MAOB: Whether they are using an MAO-B inhibitor [Yes/No]
- Other: Whether they are taking another Parkinson's medication [Yes/No]

The Data file:

- UserKey: 10 character code for that user
- Date: YYMMDD
- Timestamp: HH:MM:SS.SSS
- Hand: L or R key pressed
- Hold time: Time between press and release for current key mmmm.m milliseconds
- Direction: Previous to current LL, LR, RL, RR (and S for a space key)
- Latency time: Time between pressing the previous key and pressing current key. Milliseconds
- Flight time: Time between release of previous key and press of current key. Milliseconds

The field Parkinsons is the label for the Tappy dataset, and the rest would eventually be features. The medicine mitigating the symptoms is to show that the patient is prescribed to have the indicated medicines, but the data collected here is done in such a way that the medicine does not influence the keyboarding activities. Considered its nature, those medicine fields will be discarded.

As for the direction, it is a feature that could complicate the modeling without further investigation into medical domain knowledge of sidedness between Parkinson's disease patient and user with preferred side. There is also no such information in NQ dataset. Thus, the decision to not use this feature at this time is made.

The data file logged all 8 fields in a row and there could be over two hundred thousand (200,000) rows per user depending on the length of the keyboarding activity. There are latencies that exceeds a certain threshold and the row will be treated as irrelevant. Currently such latencies is treated as outliers and will be ruled out in individual keystroke data file. The method use here is the Tukey's Method.

As for the NQ dataset, the data consists of the following:

- The key pressed.
- The hold duration in seconds.
- The key release time in seconds from time 0.
- The key press time in seconds from time 0.

The 'hold duration' is the same type of information with the 'Hold time'. The 'Flight time' will need to be calculated in this test file from 'key release time' and 'key press time'. These times are the timer time stamp of the keystroke when it happened. Imagine a timer device in a track race is zeroed out and started counting upwards when activity begun. To get the 'Direction' from this test set, it needs some calculation as well from the 'key pressed' from two adjacent rows.

The Tappy dataset has 227 users, but only 217 users has actual activities. And Among those, 48 are healthy and 169 have Parkinson's disease.

For the NQ dataset, there are two sets of experiments, PD_MIT-CS1PD and PD_MIT-CS2PD. When combined they have 85 users, 43 are healthy and 42 have Parkinson's disease.

The combined dataset has 302 users. It was later than found out that 2 of the Tappy users only has one and two keystroke records and they are discarded. This results in 300 users with 203 contracted Parkinson's disease and 97 healthy.

The positive disease prevalence rate is 203/300, which is 67.67%.

Exploratory Visualization

Two individual combined files were randomly picked for exploratory activities.

Hold times for both PD group and no PD group are plotted below, and the hue separated out the direction from one key to another. Notice that they is no obvious trend when going from one zone of keyboard to another. However, there is a trend that no PD group has a somewhat lower hold time of approximately ~100ms, compare to PD group with ~200ms.

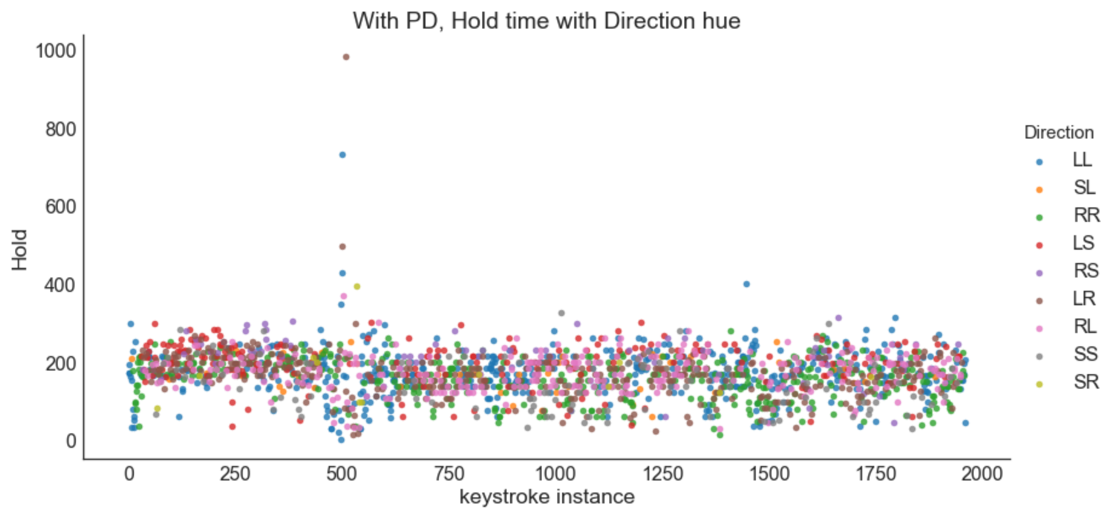


Figure 1. Exploratory Hold time plot for a user with PD

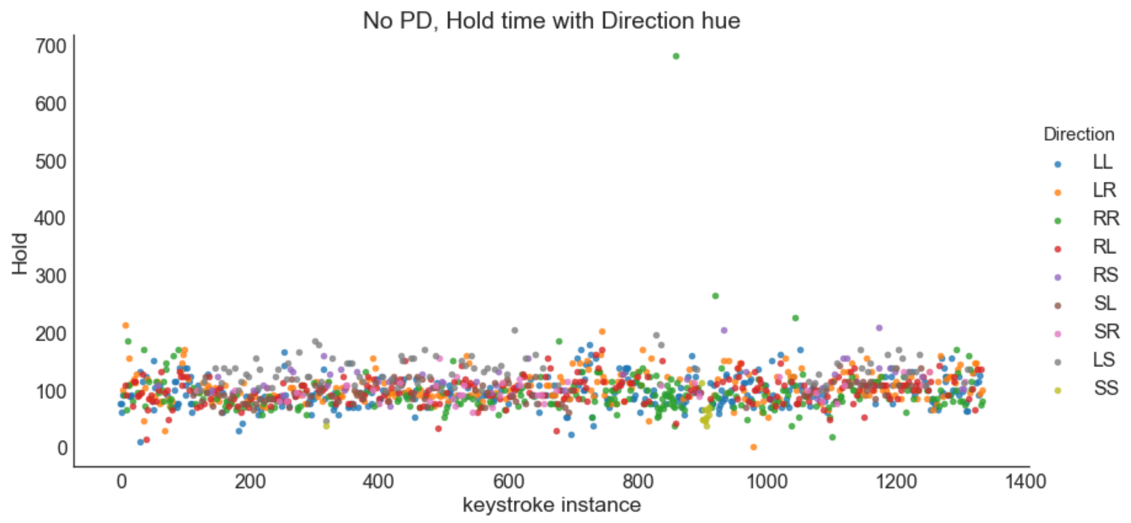


Figure 2 Exploratory Hold time plot for a user with no PD

As for hold time with hand hue:

Notice that there is no obvious trend whether a certain hand has advantages over the other, even in the case of PD sidedness.

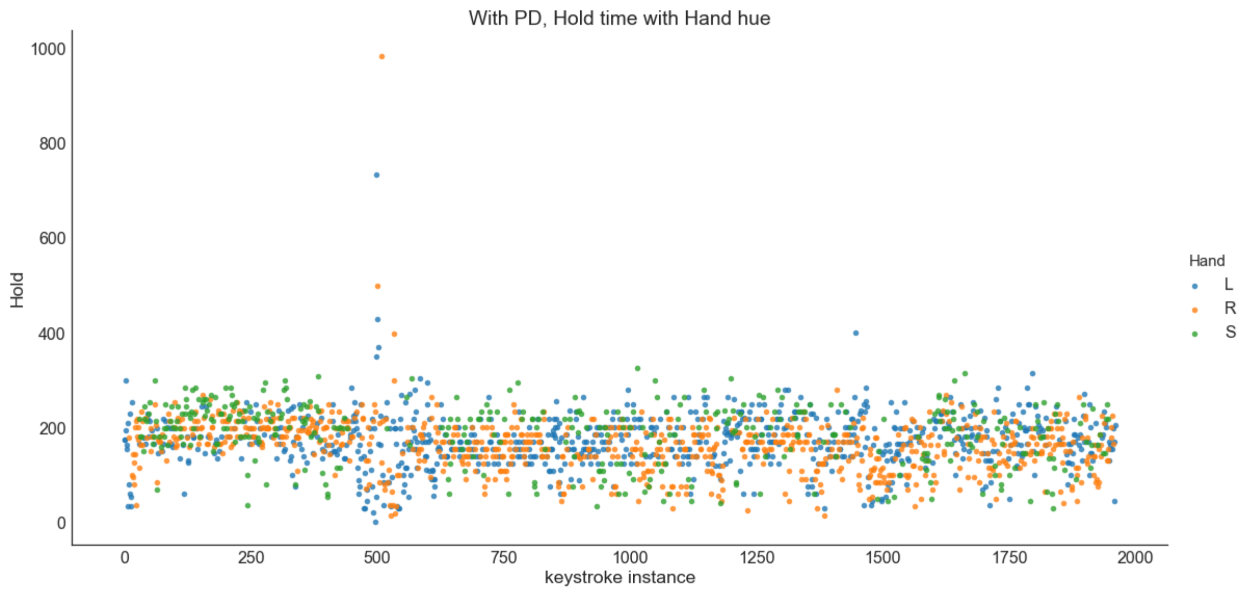


Figure 3 Exploratory Hold time plot for a user with PD

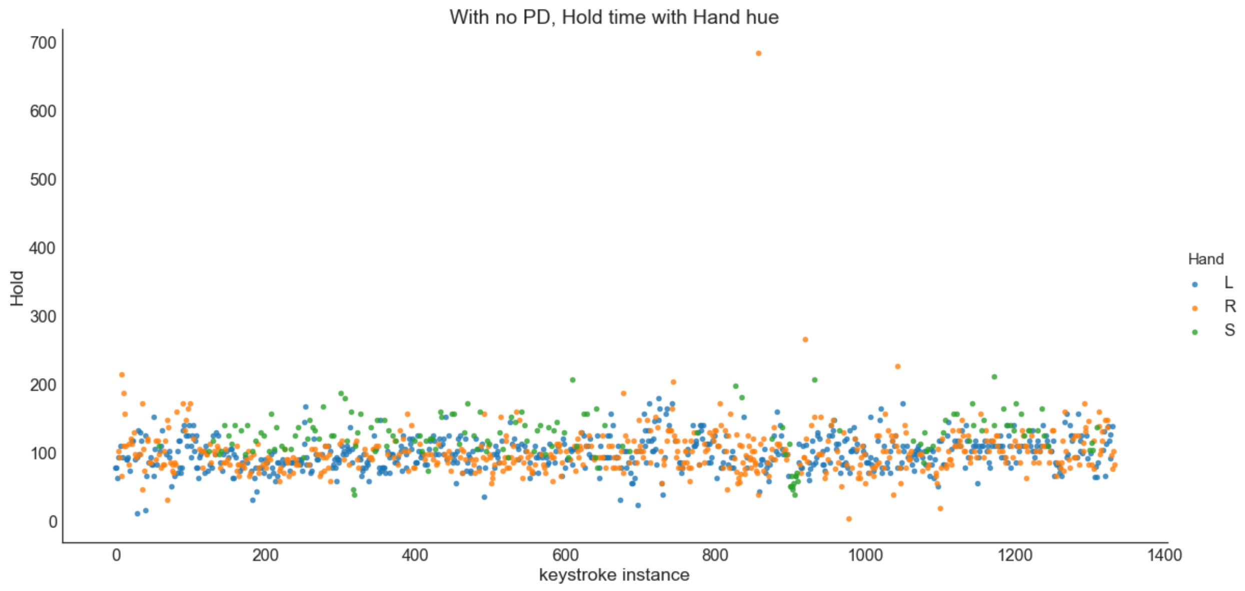


Figure 4 Exploratory Hold time plot for a user with no PD

As for flight time with hand hue:

Notice that there is no identifiable trend as well between PD and no PD group.

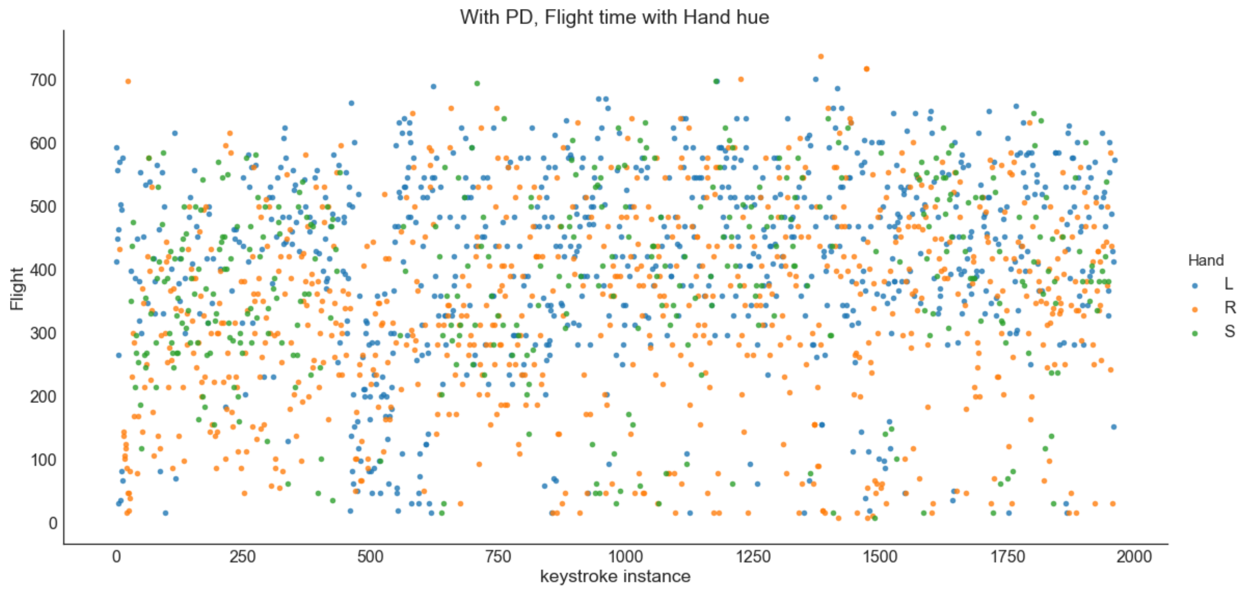


Figure 5 Exploratory Flight time plot for a user with PD

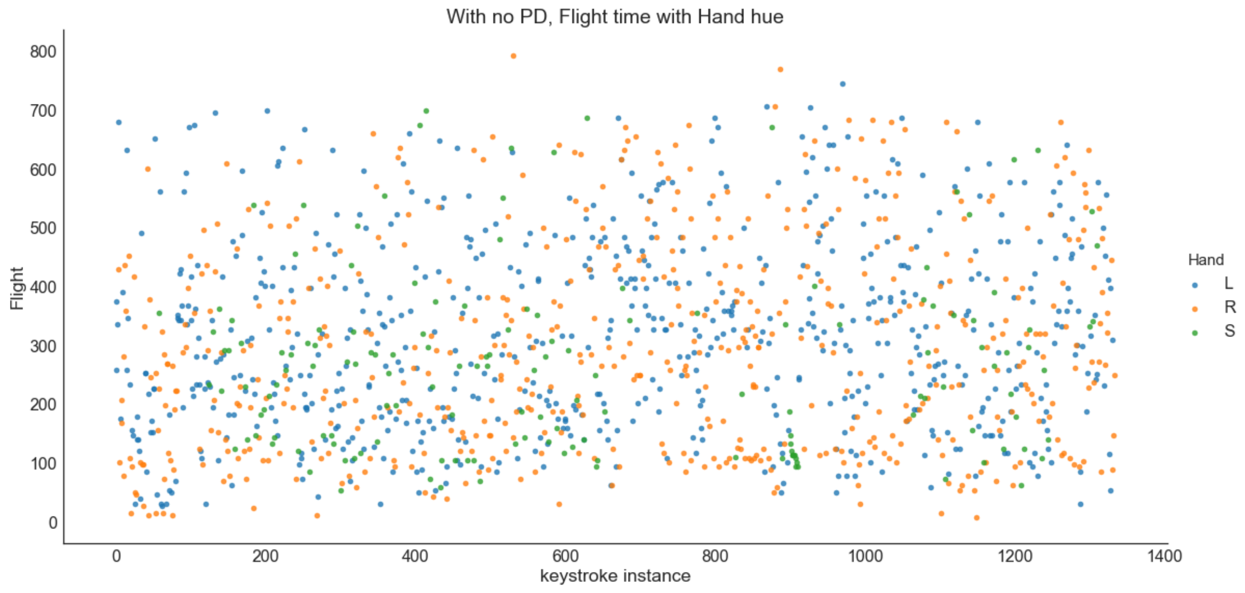


Figure 6 Exploratory Flight time plot for a user with no PD

Due to the attribute that these timing are so sparse, it was decided that some statistical means are needed to find some engineering features that could help modeling later. Variances, means, quartiles are tried and there is not much success in modeling. However, when a histogram is plotted, there is actually a normal distribution taking shape. Thus, binning was applied.

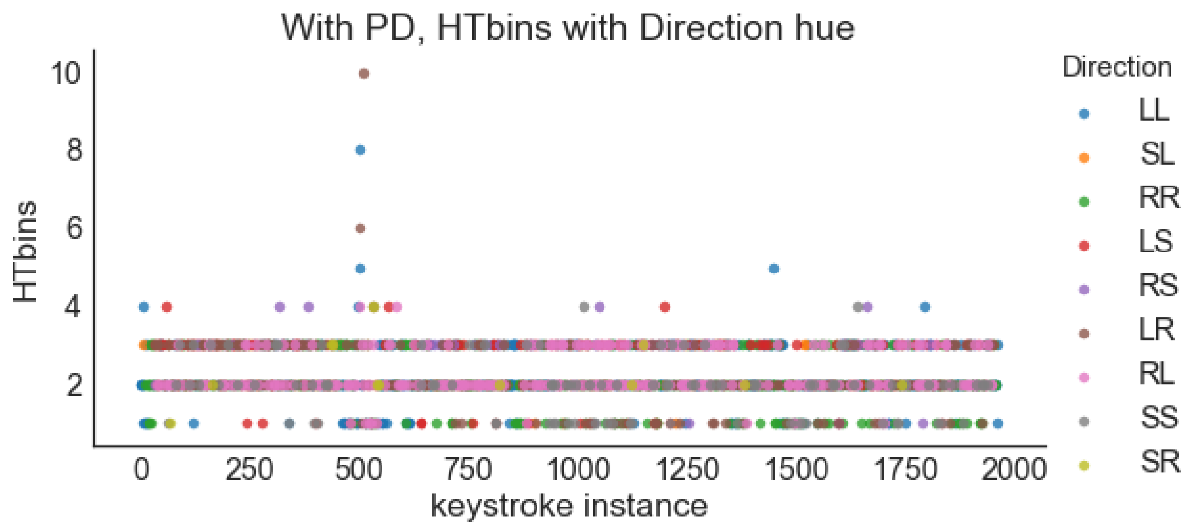


Figure 7 Exploratory HTbin plot for a user with PD

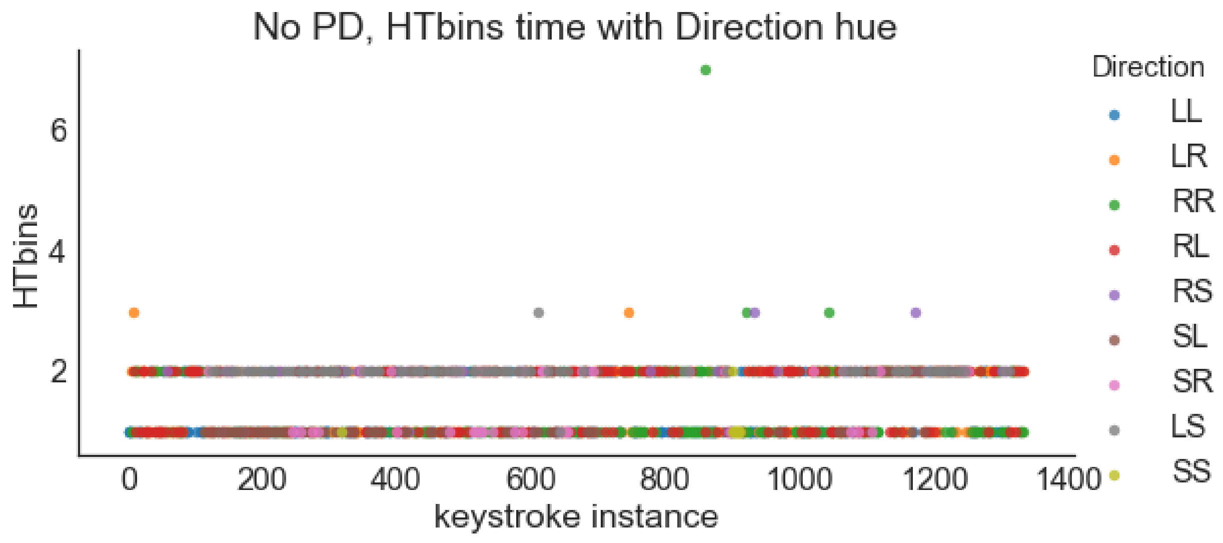


Figure 8 Exploratory HTbin plot for a user with no PD

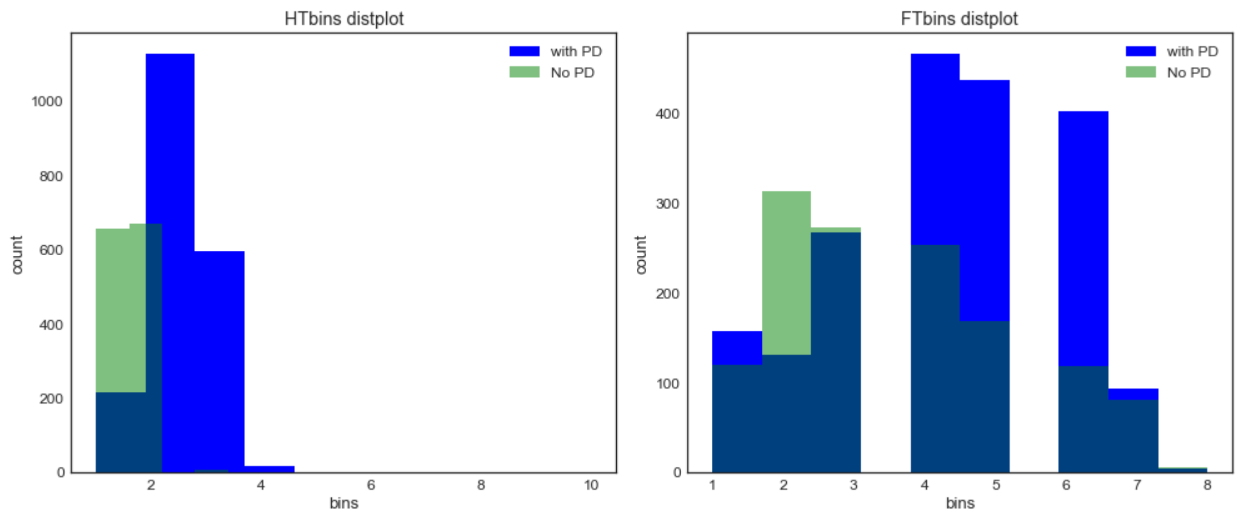


Figure 9 Histogram of HTbins and FTbins for all user separated by PD and no PD

Algorithms and Techniques

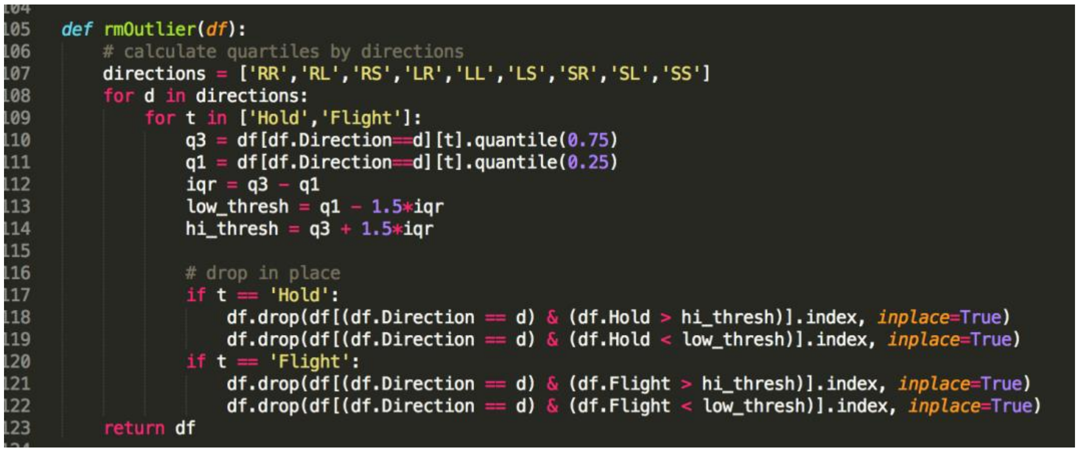
This is a binary classification problem that should focus on recall rate, with reasonable precision. Ideally all patients with Parkinson's disease should be detected, hence high recall rate for beneficial treatment and monitoring. This will invariably sacrifice some precision, causing inconveniences to falsely identified population that they are having Parkinson's disease. In this case, it is worthy since the inconvenience should be limited to monitoring and precautious action and not the actual treatment.

The algorithms used here are mainly ensemble machine learning techniques. There are

- Bagging algorithm:
  - bagging with decision tree classifier
  - random forest (essentially a bagging algorithm)
- boosting algorithm:
  - Gradient boosting
  - XG boosting
- Voting algorithm:
  - Combine best three of the above algorithms

Data are run through a few python functions to convert them from string form to float type when necessary, like for hold time and flight. Keystroke data are combined with User file to form a complete comma delimited file.

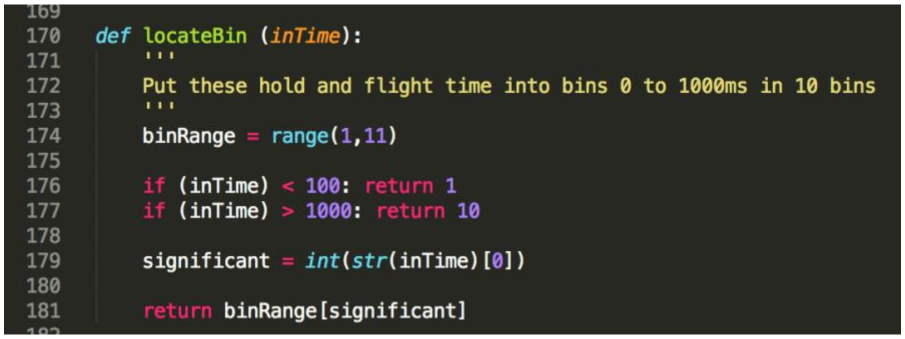
Outliers are discarded. Although it was decided to not include the direction feature, in the removal of outlier it was still used to group up keystroke data before the quartiles are calculated to detect outliers. The following code snippet reflected this. Note the use of filter ‘df.Direction==d’ when calculating the third quartile ‘q3’ and ‘q1’.



```python
def rmOutlier(df):
    # calculate quartiles by directions
    directions = ['RR','RL','RS','LR','LL','LS','SR','SL','SS']
    for d in directions:
        for t in ['Hold','Flight']:
            q3 = df[df.Direction==d][t].quantile(0.75)
            q1 = df[df.Direction==d][t].quantile(0.25)
            iqr = q3 - q1
            low_thresh = q1 - 1.5*iqr
            hi_thresh = q3 + 1.5*iqr

            # drop in place
            if t == 'Hold':
                df.drop(df[(df.Direction == d) & (df.Hold > hi_thresh)].index, inplace=True)
                df.drop(df[(df.Direction == d) & (df.Hold < low_thresh)].index, inplace=True)
            if t == 'Flight':
                df.drop(df[(df.Direction == d) & (df.Flight > hi_thresh)].index, inplace=True)
                df.drop(df[(df.Direction == d) & (df.Flight < low_thresh)].index, inplace=True)
    return df
```

Figure 10 rmOutlier code snippet

Hold times and flight times are separated into 10 bins each. Each bin carries 100ms, for a total of 1000ms to cover all duration needs. In the mytappy.py, the function ‘locateBin’ is doing just that:

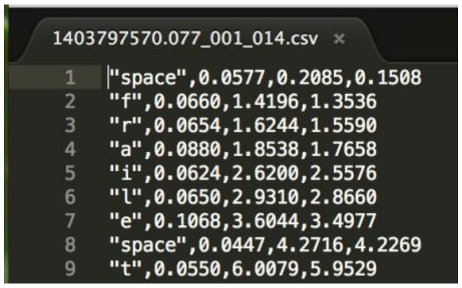


```python
def locateBin (inTime):
    '''
    Put these hold and flight time into bins 0 to 1000ms in 10 bins
    '''
    binRange = range(1,11)

    if (inTime) < 100: return 1
    if (inTime) > 1000: return 10

    significant = int(str(inTime)[0])

    return binRange[significant]
```

Figure 11 locateBin function

Variance and mean time for hold time and flight time are also calculated and written out to a csv file for python pandas to read in later on.

In NQ data set the timing is clock timing, meaning the keystroke in whole file is time stamps logging for all keys entered when activity get started. The file below is a sample of the raw keystroke data. The columns are key pressed, hold duration, key released time stamp, key pressed time stamp. Notice that starting from first row down and last column to third column, the number keep increasing. For e.g. from first row on, there is 0.1508, 0.2085, 1.3536, 1.4196, 1.5590 and so on. So the “space” hold duration of 0.0577 is computed by (0.2085 – 0.1508) which is (key released time stamp - key pressed time stamp). The “f” flight duration is computed by (1.3536 – 0.2085) which is (“f” key pressed time stamp – “space” key released time stamp).



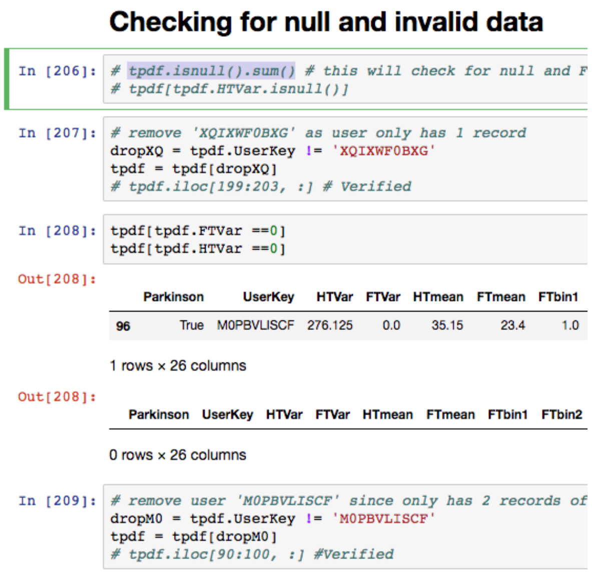
```
1403797570.077_001_014.csv
"space",0.0577,0.2085,0.1508
"f",0.0660,1.4196,1.3536
"r",0.0654,1.6244,1.5590
"a",0.0880,1.8538,1.7658
"i",0.0624,2.6200,2.5576
"l",0.0650,2.9310,2.8660
"e",0.1068,3.6044,3.4977
"space",0.0447,4.2716,4.2269
"t",0.0550,6.0079,5.9529
```

Figure 12 Hold time and Flight time from NQ data set raw data

They went through the same process to arrive at the same state. Spyder 3 in Anaconda is used as coding environment.

Two huge files are produced and jupyter notebook was used to further process them as pandas DataFrame.

In jupyter notebook the 2 files are combined to form a DataFrame. It is checked for null values and those rows are discarded as there is no way to predict a replacement. No data is better than misleading data which could confuse models. *isnull().sum()* go through all columns of dataframe and reports columns that have null values. In this case, FTVar and HTVar are reported and it was found that user 'XQIXWF0BXG' does not have variance since there is only one row of keystroke data. In line 208, a filter to check FTVar==0 is applied and the user is removed in line 209.



Checking for null and invalid data

```
In [206]: # tpdf.isnull().sum() # this will check for null and F
          # tpdf[tpdf.HTVar.isnull()]

In [207]: # remove 'XQIXWF0BXG' as user only has 1 record
          dropXQ = tpdf.UserKey != 'XQIXWF0BXG'
          tpdf = tpdf[dropXQ]
          # tpdf.iloc[199:203, :] # Verified

In [208]: tpdf[tpdf.FTVar ==0]
          tpdf[tpdf.HTVar ==0]

Out[208]:
```

	Parkinson	UserKey	HTVar	FTVar	HTmean	FTmean	FTbin1
96	True	M0PBVLISCF	276.125	0.0	35.15	23.4	1.0

1 rows × 26 columns

```
Out[208]:
```

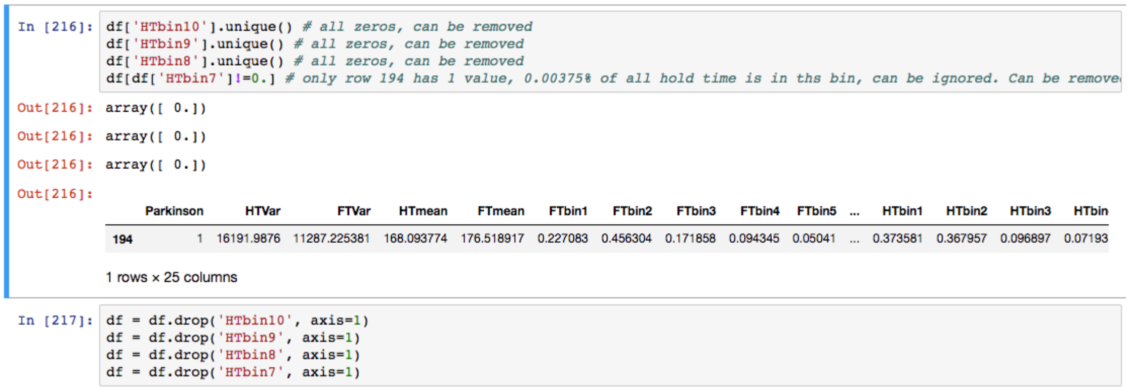
Parkinson	UserKey	HTVar	FTVar	HTmean	FTmean	FTbin1	FTbin2

0 rows × 26 columns

```
In [209]: # remove user 'M0PBVLISCF' since only has 2 records of
          dropM0 = tpdf.UserKey != 'M0PBVLISCF'
          tpdf = tpdf[dropM0]
          # tpdf.iloc[90:100, :] #Verified
```

Figure 13 Detect and remove null values in columns.

Then a few columns HTbin7 to HTbin10 were removed because they contained only zeros, which provide no useful knowledge for model in this case.

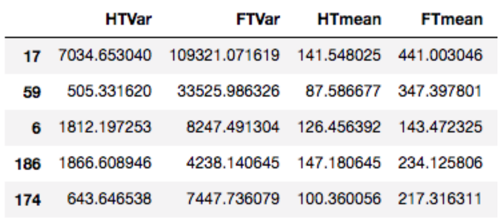


```
In [216]: df['HTbin10'].unique() # all zeros, can be removed
          df['HTbin9'].unique() # all zeros, can be removed
          df['HTbin8'].unique() # all zeros, can be removed
          df[df['HTbin7']!=0.] # only row 194 has 1 value, 0.00375% of all hold time is in ths bin, can be ignored. Can be remove

Out[216]: array([ 0.])

Out[216]: array([ 0.])

Out[216]: array([ 0.])

Out[216]:
               Parkinson      HTVar         FTVar      HTmean     FTmean    FTbin1    FTbin2    FTbin3    FTbin4   FTbin5  ...   HTbin1    HTbin2    HTbin3   HTbin
          194          1 16191.9876  11287.225381  168.093774 176.518917  0.227083  0.456304  0.171858  0.094345  0.05041  ...  0.373581  0.367957  0.096897  0.07193

          1 rows × 25 columns

In [217]: df = df.drop('HTbin10', axis=1)
          df = df.drop('HTbin9', axis=1)
          df = df.drop('HTbin8', axis=1)
          df = df.drop('HTbin7', axis=1)
```

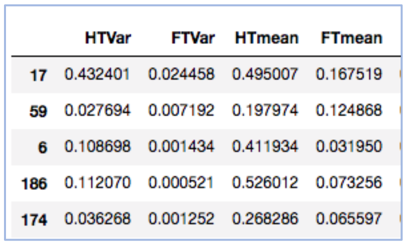
Figure 14 Columns containing zeros are removed.

Bins contains the normalized frequencies of the assigned intervals. Adding all values for all ten bins will equal one. And a value of 0.3 in a hold time bin means that the bin contains 30% of all hold times of that user, and those hold times are all within the assigned interval for that bin. Bins that are logging zero for all users are removed.

As for the variances and means they are run through models as is the first round. Then a MinMaxScaler from sklearn.preprocessing is used to transform them to between zero and one to facilitate the learning models. It was found that there is slight improvement.



	HTVar	FTVar	HTmean	FTmean
17	7034.653040	109321.071619	141.548025	441.003046
59	505.331620	33525.986326	87.586677	347.397801
6	1812.197253	8247.491304	126.456392	143.472325
186	1866.608946	4238.140645	147.180645	234.125806
174	643.646538	7447.736079	100.360056	217.316311



	HTVar	FTVar	HTmean	FTmean
17	0.432401	0.024458	0.495007	0.167519
59	0.027694	0.007192	0.197974	0.124868
6	0.108698	0.001434	0.411934	0.031950
186	0.112070	0.000521	0.526012	0.073256
174	0.036268	0.001252	0.268286	0.065597

Figure 15 MinMaxScaler transforms columns from left to right

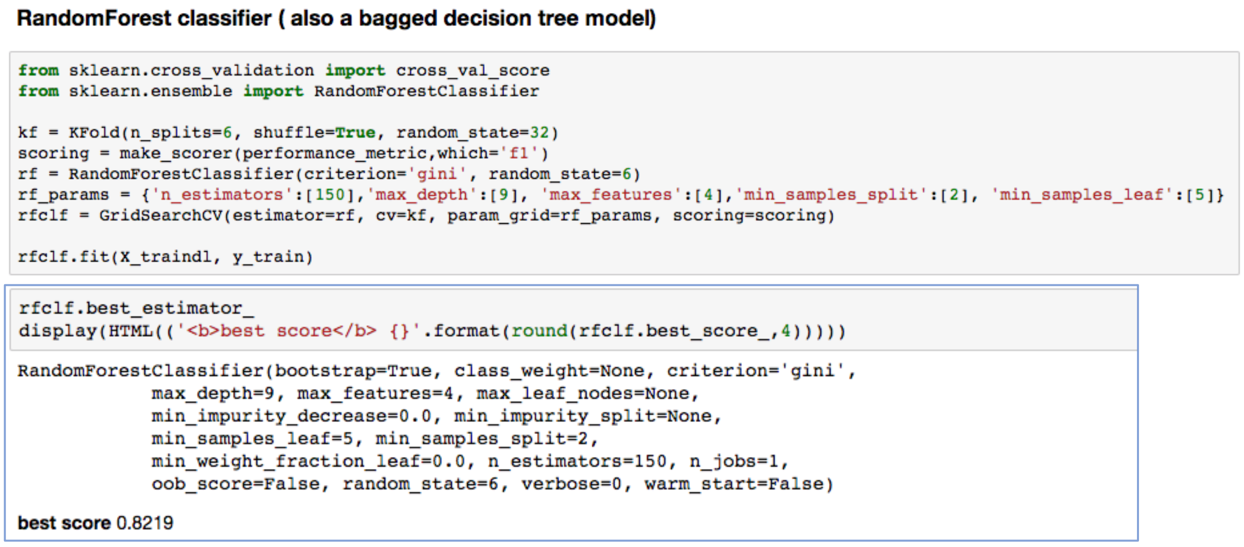
In this dataset, there is no categorical data being used in the modeling, except the label itself.

Since the label for Tappy data set is Boolean with True or False string, the LabelEncoder from sklearn.preprocessing is used to transform them to 1 and 0, 1 being positive in Parkinson's disease.

20% of all data, or 60 rows out of 300 total rows are reserved for testing strictly, they are not used for training at all. This is to guarantee that the test data truly represent an evaluation of the model. The following code snippet shows that train_test_split from sklearn is used to do the separation.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

The other 80% are training data and are used with cross validation technique. Kfold of various values and shuffling are used with the cross validation. GridSearchCV technique is applied as well to comb through parameters in search for optimum model. For example, the Random Forest classifier:



RandomForest classifier (also a bagged decision tree model)

```
from sklearn.cross_validation import cross_val_score
from sklearn.ensemble import RandomForestClassifier

kf = KFold(n_splits=6, shuffle=True, random_state=32)
scoring = make_scorer(performance_metric,which='f1')
rf = RandomForestClassifier(criterion='gini', random_state=6)
rf_params = {'n_estimators':[150],'max_depth':[9], 'max_features':[4],'min_samples_split':[2], 'min_samples_leaf':[5]}
rfclf = GridSearchCV(estimator=rf, cv=kf, param_grid=rf_params, scoring=scoring)

rfclf.fit(X_traindl, y_train)
```

```
rfclf.best_estimator_
display(HTML(('<b>best score</b> {}'.format(round(rfclf.best_score_,4)))))
```

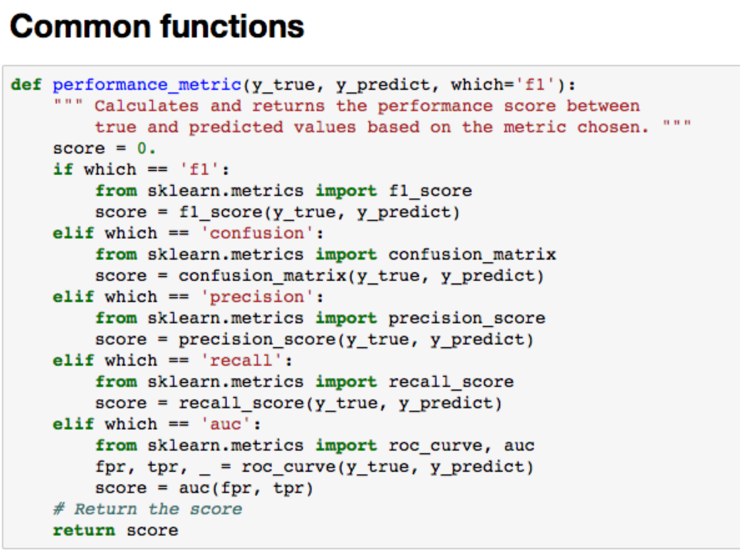
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=9, max_features=4, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=150, n_jobs=1,
            oob_score=False, random_state=6, verbose=0, warm_start=False)
```

best score 0.8219

Figure 16 Random Forest classifier

'kf' holds the indexes worked out by the KFold function call with 6 folds of split with shuffling. Among those 6 sets of sub data from the full training data, one set will always be hold out for validation, and each set get a chance to be a validation set. In the call to GridSearchCV function, 'kf' is presented to the parameter cv, which stands for cross validation.

A scoring function 'scoring' is also fed to GridSearchCV. 'make_scorer' from sklearn.metrics.scorer is used to produce this 'scoring' custom performance measurement and fed into GridSearchCV. In turn, make_scorer actually make a call to 'performance_metric' with parameter 'which='f1'' to get the actual f1 scoring function.



Common functions

```
def performance_metric(y_true, y_predict, which='f1'):
    """ Calculates and returns the performance score between
        true and predicted values based on the metric chosen. """
    score = 0.
    if which == 'f1':
        from sklearn.metrics import f1_score
        score = f1_score(y_true, y_predict)
    elif which == 'confusion':
        from sklearn.metrics import confusion_matrix
        score = confusion_matrix(y_true, y_predict)
    elif which == 'precision':
        from sklearn.metrics import precision_score
        score = precision_score(y_true, y_predict)
    elif which == 'recall':
        from sklearn.metrics import recall_score
        score = recall_score(y_true, y_predict)
    elif which == 'auc':
        from sklearn.metrics import roc_curve, auc
        fpr, tpr, _ = roc_curve(y_true, y_predict)
        score = auc(fpr, tpr)
    # Return the score
    return score
```

Figure 17 performance_metric()

GridSearchCV also takes in an intended classifier and the hyperparameters for the classifier. The classifier or estimator here is estimator=rf, which is RandomForestClassifier(). The hyperparameters is a python dictionary. Those keys are the parameters feeding into classifier that can be manipulated. The values here are our many guesses for each key that could potentially produce a great classifier. GridSearchCV exhaustively go through the combinations and will eventually spit out a best classifier with the best score according to the custom scoring function.

In this case, the best estimator is printed out, the score is 0.8219. Notice that the hyperparameter dictionary 'rf_params' only have one value for each key. This is because I have adopted the strategy to tune one parameter at a time while holding the rest at default. Once I have the best value for this particular training set, I left the value there. The lower screen capture of figure 17 showed the best values of the classifier churned out by GridSearchCV.

This is the time-consuming part of the training, and if the classifier were to run really slow it would be worthless to use even if it means we are giving up best classifier.

A few common functions are extracted out under the 'Common functions' session. They are performance metric for GridSearchCV, and printing and plotting of results and ROC/AUC and PR curve.

Benchmark

A crude, simple model like default parameterized support vector machine classifier or decision tree model could blatantly predict that everyone has Parkinson's disease and the precision would be 203/300 which is 67.7%, and recall will top at 100%. F1 score is logged at 80%. But the ROC AUC will suffer at 50%, which is useless as the model lose the ability to tell the difference.

Confusion Matrix	TN=5	FP=16
	FN=3	TP=36
F1	0.79	
Precision	0.6923	
Recall	0.9231	
ROC/AUC	0.6447	

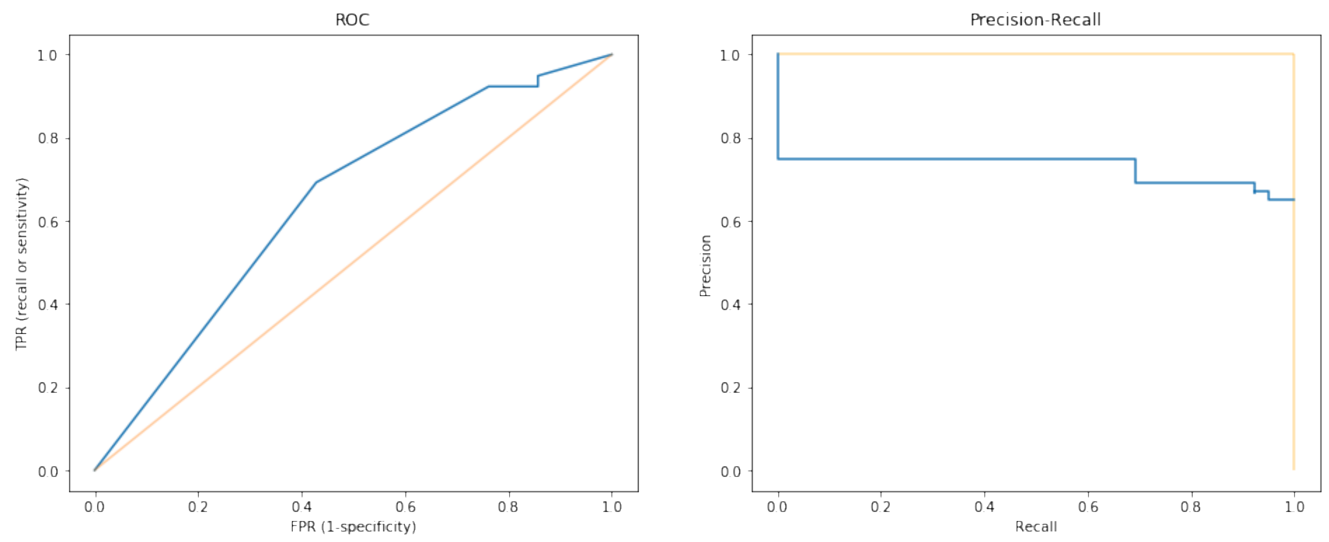
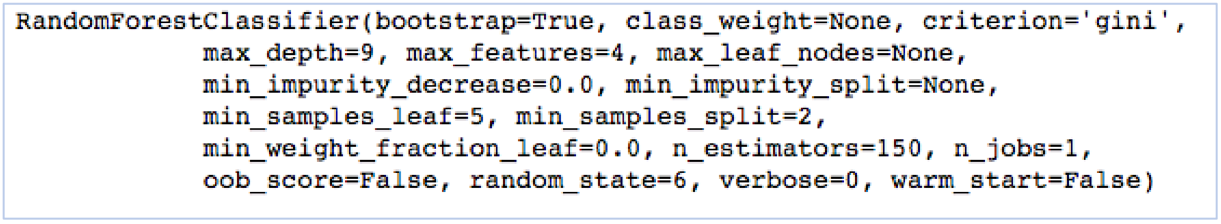


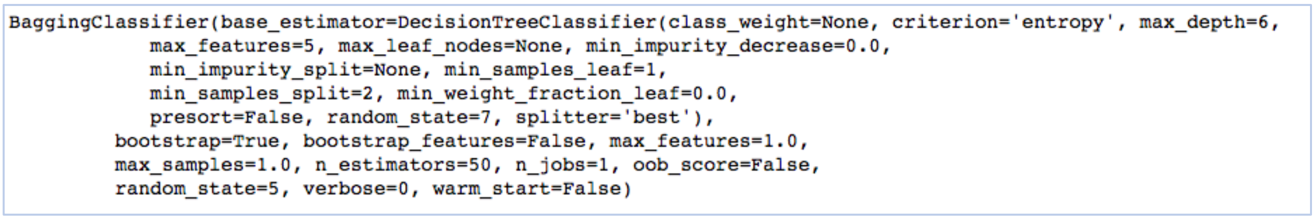
Figure 18 Result for benchmark Decision Tree classifier

Methodology

Implementation

Table below are the models used in voting classifier:


```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=9, max_features=4, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=150, n_jobs=1,
            oob_score=False, random_state=6, verbose=0, warm_start=False)
```


```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=6,
            max_features=5, max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=7, splitter='best'),
         bootstrap=True, bootstrap_features=False, max_features=1.0,
         max_samples=1.0, n_estimators=50, n_jobs=1, oob_score=False,
         random_state=5, verbose=0, warm_start=False)
```

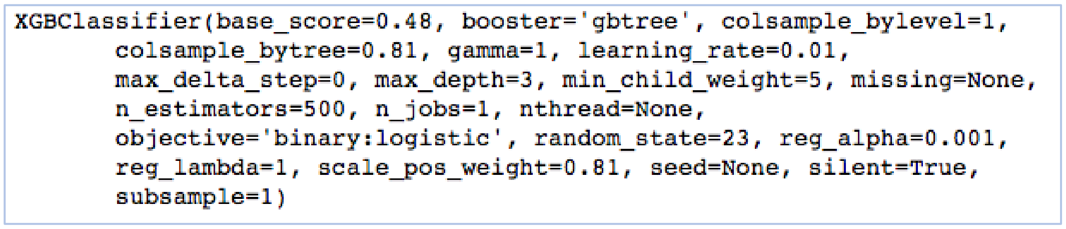

```
XGBClassifier(base_score=0.48, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=0.81, gamma=1, learning_rate=0.01,
       max_delta_step=0, max_depth=3, min_child_weight=5, missing=None,
       n_estimators=500, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=23, reg_alpha=0.001,
       reg_lambda=1, scale_pos_weight=0.81, seed=None, silent=True,
       subsample=1)
```

Figure 19 three classifiers with their best hyperparameters

Note that there are different parameters for different classifier algorithms. Random forest classifier is the best among all models shown. This model is of type bagging ensemble, the bootstrap=True indicates such.

Bootstrap here means randomly resamples the same dataset with replacement, generating new samples in a sense.

Bootstrap aggregation or also known as bagging is a technique that help reduces variance or random noise when training a model. Bagging takes these new samples and trains them on multiple models to get the average of the models' prediction as the final prediction. The models could be of single classifier like decision tree or of various classifier algorithms.

The bagging classifier with base_estimator=DecisionTreeClassifier applies this technique. The model DecisionTreeClassifier is the only classifier but it is trained on multitude of new samples before being averaged out for prediction.

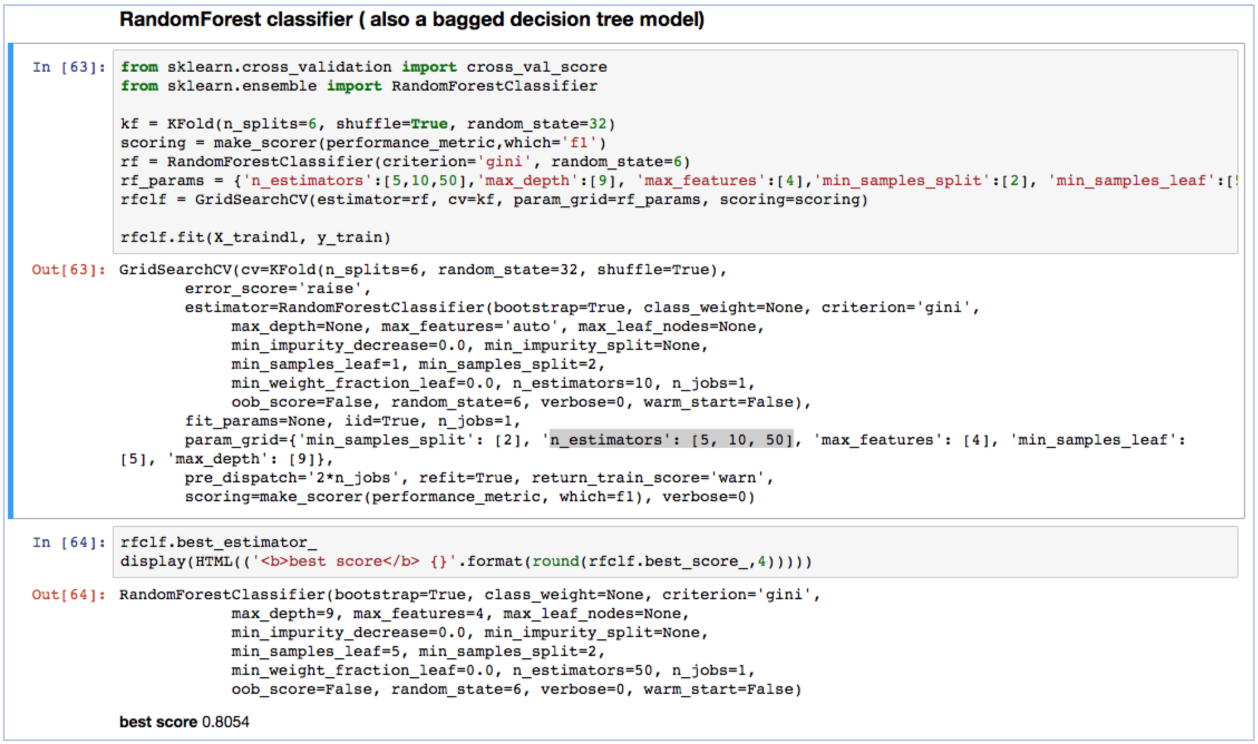
However, random forest classifier uses a different kind of bagging technique called feature bagging. The classifier randomly chooses fewer features when it needs to split data to grow a decision tree structure. It further avoids overly using strong features to do a split and this add to the strength of a random forest algorithm to predict.

XGBoost classifier uses boosting technique. It is a sequential and iterative tuning process with weak models (models that does not predict great) training on various part of training data and thereby getting improved predictions as the process move from model to model and easily predicted data to harder to predict data. Those models are then combined to produce a strong classifier.

Refinement

Classifiers are built from beginning with default hyperparameters or some reasonable guesses like max_depth of tree which is unlikely to be 1, or 20 levels. Parameters are tuned against the performance. For e.g. random forest classifier n_estomators could be set up with 5,10,50 in GridSearchCV and then run fit on it to find out which could gain best performance. GridSearchCV exhausted all the combinations and gave us the best out of what we speculated with all the ranges of values.

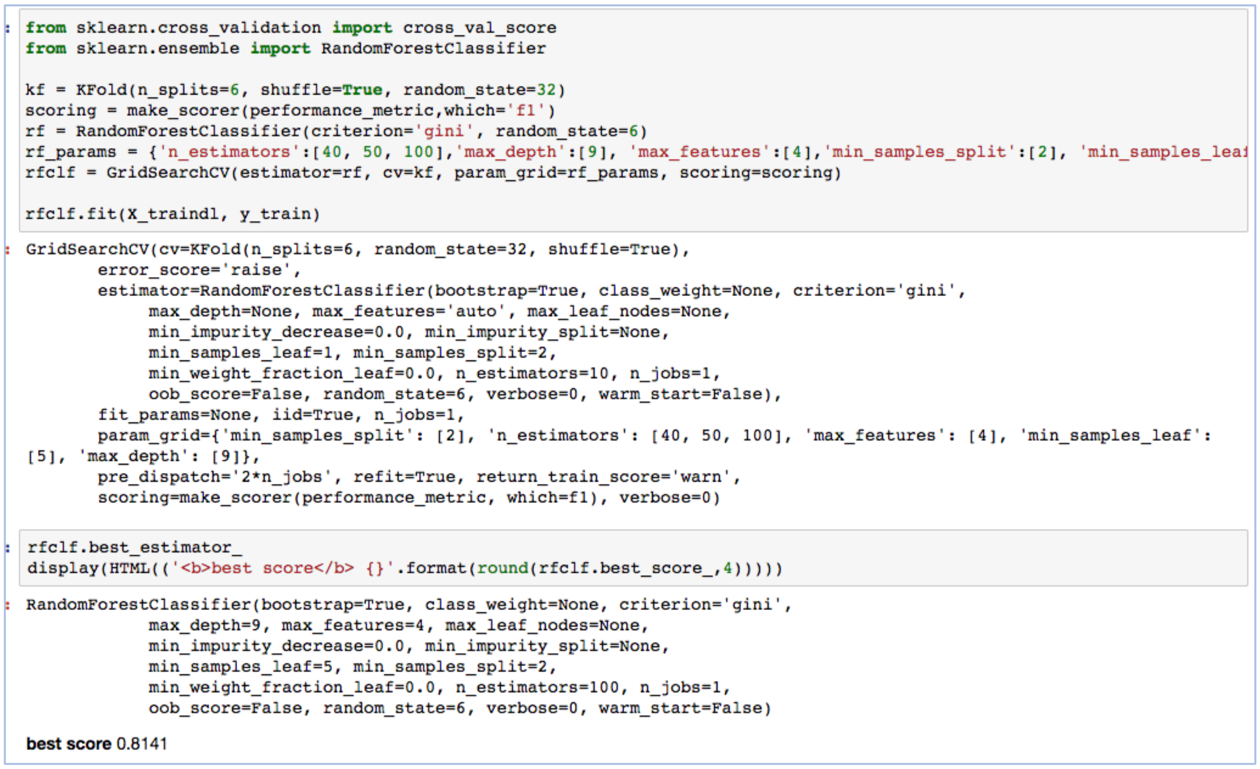
In this case, according to the screen shot Figure 20 below, n_estimators of 50 return score of 0.8054 which is the best. Then I will pick 40, 50, 100 for the next run, holding others constant. The next run showed that n_estimators=100 is better than 50.



RandomForest classifier (also a bagged decision tree model)

```
In [63]: from sklearn.cross_validation import cross_val_score
         from sklearn.ensemble import RandomForestClassifier

         kf = KFold(n_splits=6, shuffle=True, random_state=32)
         scoring = make_scorer(performance_metric,which='f1')
         rf = RandomForestClassifier(criterion='gini', random_state=6)
         rf_params = {'n_estimators':[5,10,50],'max_depth':[9], 'max_features':[4],'min_samples_split':[2], 'min_samples_leaf':[
         rfclf = GridSearchCV(estimator=rf, cv=kf, param_grid=rf_params, scoring=scoring)

         rfclf.fit(X_traindl, y_train)

Out[63]: GridSearchCV(cv=KFold(n_splits=6, random_state=32, shuffle=True),
                error_score='raise',
                estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                      oob_score=False, random_state=6, verbose=0, warm_start=False),
                fit_params=None, iid=True, n_jobs=1,
                param_grid={'min_samples_split': [2], 'n_estimators': [5, 10, 50], 'max_features': [4], 'min_samples_leaf':
         [5], 'max_depth': [9]},
                pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                scoring=make_scorer(performance_metric, which=f1), verbose=0)

In [64]: rfclf.best_estimator_
         display(HTML(('<b>best score</b> {}'.format(round(rfclf.best_score_,4)))))

Out[64]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=9, max_features=4, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=5, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=50, n_jobs=1,
                      oob_score=False, random_state=6, verbose=0, warm_start=False)
```

best score 0.8054



```
: from sklearn.cross_validation import cross_val_score
  from sklearn.ensemble import RandomForestClassifier

  kf = KFold(n_splits=6, shuffle=True, random_state=32)
  scoring = make_scorer(performance_metric,which='f1')
  rf = RandomForestClassifier(criterion='gini', random_state=6)
  rf_params = {'n_estimators':[40, 50, 100],'max_depth':[9], 'max_features':[4],'min_samples_split':[2], 'min_samples_lea
  rfclf = GridSearchCV(estimator=rf, cv=kf, param_grid=rf_params, scoring=scoring)

  rfclf.fit(X_traindl, y_train)

: GridSearchCV(cv=KFold(n_splits=6, random_state=32, shuffle=True),
         error_score='raise',
         estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
               max_depth=None, max_features='auto', max_leaf_nodes=None,
               min_impurity_decrease=0.0, min_impurity_split=None,
               min_samples_leaf=1, min_samples_split=2,
               min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
               oob_score=False, random_state=6, verbose=0, warm_start=False),
         fit_params=None, iid=True, n_jobs=1,
         param_grid={'min_samples_split': [2], 'n_estimators': [40, 50, 100], 'max_features': [4], 'min_samples_leaf':
  [5], 'max_depth': [9]},
         pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
         scoring=make_scorer(performance_metric, which=f1), verbose=0)

: rfclf.best_estimator_
  display(HTML(('<b>best score</b> {}'.format(round(rfclf.best_score_,4)))))

: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
               max_depth=9, max_features=4, max_leaf_nodes=None,
               min_impurity_decrease=0.0, min_impurity_split=None,
               min_samples_leaf=5, min_samples_split=2,
               min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
               oob_score=False, random_state=6, verbose=0, warm_start=False)
```

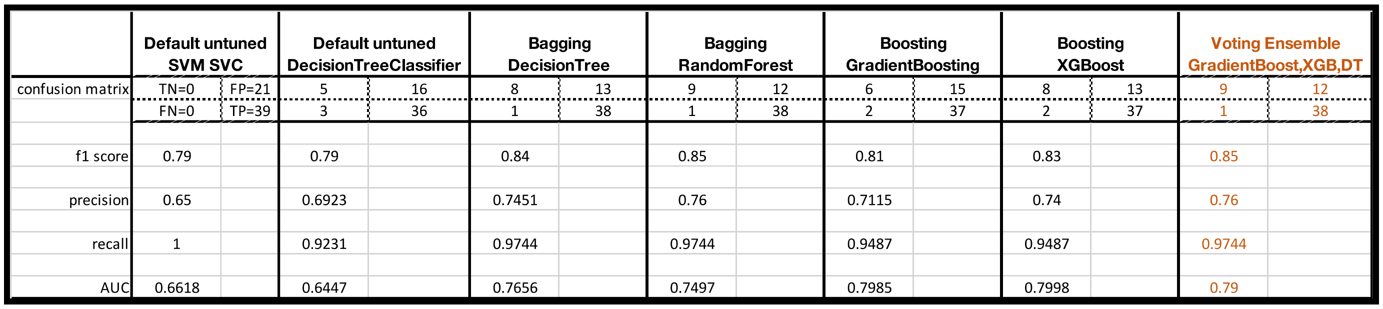
best score 0.8141

Figure 20 Random Forest classifier n_estimators tuning

RandomSearchCV is the other technique that could work as well. It has gained popularity and there are great results out of the RandomSearchCV.

Results

Model Evaluation and Validation



	Default untuned SVM SVC		Default untuned DecisionTreeClassifier		Bagging DecisionTree		Bagging RandomForest		Boosting GradientBoosting		Boosting XGBoost		Voting Ensemble GradientBoost,XGB,DT	
confusion matrix	TN=0	FP=21	5	16	8	13	9	12	6	15	8	13	9	12
	FN=0	TP=39	3	36	1	38	1	38	2	37	2	37	1	38
f1 score	0.79		0.79		0.84		0.85		0.81		0.83		0.85	
precision	0.65		0.6923		0.7451		0.76		0.7115		0.74		0.76	
recall	1		0.9231		0.9744		0.9744		0.9487		0.9487		0.9744	
AUC	0.6618		0.6447		0.7656		0.7497		0.7985		0.7998		0.79	

Figure 21 Results from various models

There are 60 rows reserved for testing. Among them, 21 are no PD, 39 are with PD.

Result of Voting Ensemble:

F1 score wise, only Bagging RandomForest and voting ensemble achieve 0.85.

As for recall, the best score is 0.9744, achieved by Bagging DT, Bagging RF and Voting Ensemble.

Among the Bagging RF and Voting Ensemble, they share the same high precision number. Thus, the AUC comes into play and the Voting Ensemble is a clear achiever, with 0.79.

In the voting classifier, I have given weight of 5,3,2 to random forest classifier, bagging algorithm with decision tree as estimator and xgboost classifier respectively.

In XGboost classifier, 4 features have been identified to be of utmost importance in differentiating the classes: FTbin2, FTbin3, FTbin4 and HTVar.

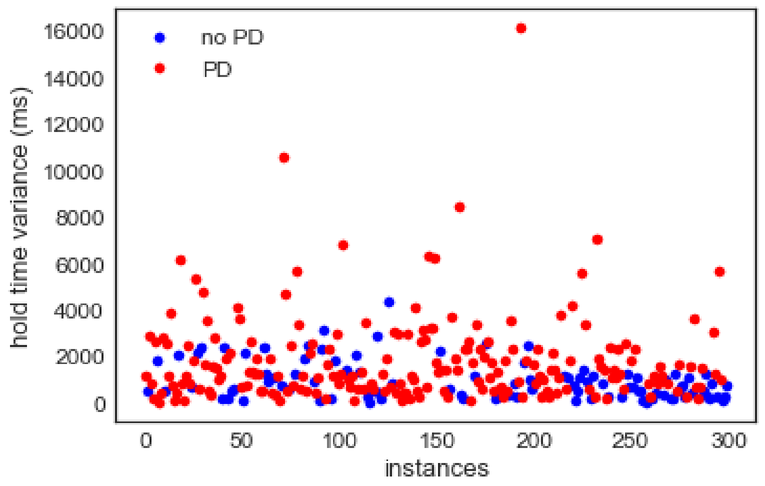


Figure 22 HTVar, one of the identified important feature

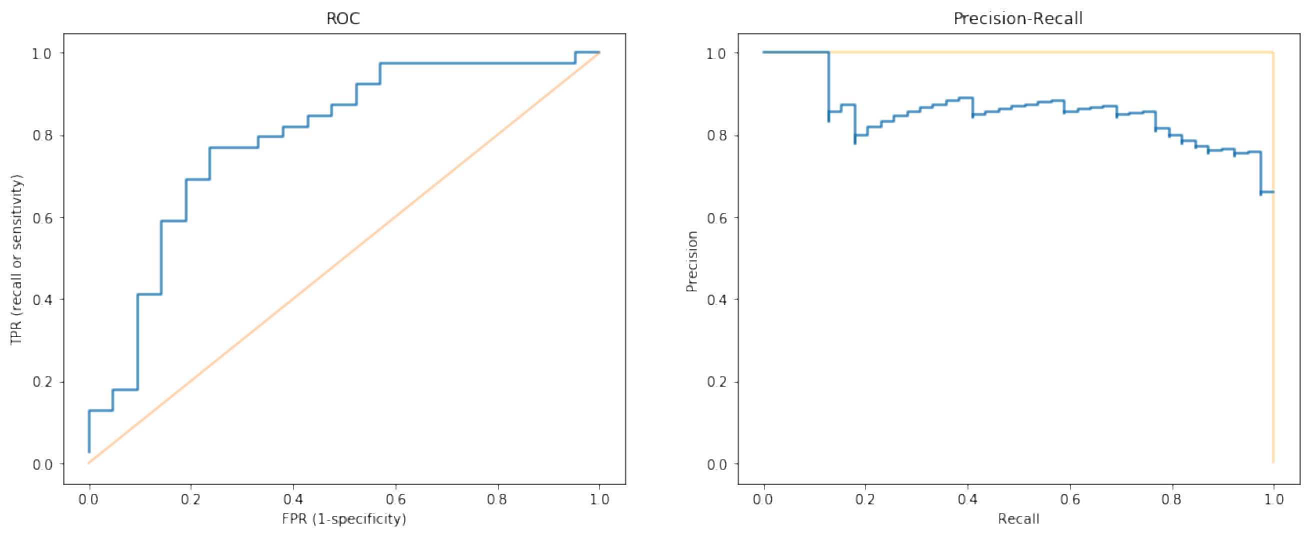


Figure 23 Result of Voting classifier

Justification

Recall of high percentage is mandatory here and precision of 0.76 seems to be a good enough sacrifice. Compared to benchmark model lightly tuned Decision Tree model, both the recall and especially the precision are improved.

The ROC curve of Voting Ensemble classifier is way better than the benchmark model.

Conclusion

Given the limited knowledge about the design of the two data collections and the attributes embedded within those keystroke data, the feature engineering is a challenge. Domain knowledge is a key factor as well in designing a successful machine learning model.

However, judging from the good PR curve and good ROC curve, together with the high recall percentage without totally thrashing precision, I believe a workable model has been achieved.

Voting ensemble borrows the strength of bagging algorithm and boosting algorithm. XG boost classifier has been flexible and highly effective and it truly contributes to the overall success.

Reflection

Clinical prediction is a very impactful area. Machine learning is a way that could make a huge difference. The project of using keystroke data to predict possibility of Parkinson's disease is just exciting.

The Tappy data set is a great place to start even though the legitimate data is only 217 records and there are 169 users with Parkinson's disease. This is a negative PD over positive PD ratio of 3.5. The imbalance is a problem. The better solution is to find more and NQ data set presented itself. After careful inspection of the data I have found that it is quite usable. There are some time stamps subtractions needed to obtain the equivalent hold duration and flight duration.

The raw data from NQ is sorted out so it is at the same stage as Tappy data set. The user files and the corresponding keystroke data file are in separate directories. A user could have multiple keystroke data files and they are combined together in a big file by python code before being preprocessed.

The preprocessing involves transforming numbers in string format to floating number, calculates means and variances, and group those hold and flight duration into bins for effective modeling. The 2 final csv files produced contains a row for every user with all their processed user data and keystroke data. Tappy csv and NQ csv files are ready for analysis.

The two files are read in as pandas DataFrame in jupyter notebook. They are detected for nulls so that blank data are tossed out. They are checked for zeroes where the whole column is zeroes. They are deleted since it is useless for modeling.

The two files are now containing similar columns and they contain valid data. They are combined to form a dataset. Many modules and functions in sklearn are used here. The label is encoded to use 0 or 1 to respectively represent no Parkinson's disease and with Parkinson's disease. Variances and means are regularized so their values are between 0 and 1. If this is not done, a 10000 in flight duration variance could be mistaken as important than 180 hold duration mean because classifier sees it as more weight for the feature with big number.

Data are split into 80/20 ratio, 80% for training and 20% solely for testing. Later on in the training there is KFold cross validation technique used to improve training and the 80% of data is further divided into chunks for cross validation.

Ensemble binary classifiers methods are the popular and effective methods for this type of problem. Therefore, after I benchmark the performance with somewhat default support vector machine svc and decision tree, I jumped right into the 3 types of ensemble: the bagging, the boosting and voting.

GridSearchCV technique is set up to take in multiple values for hyperparameters of a model. The model is then put through the fit phase and the best values out of those provided will be surfaced and reported. I then started to fine tune those values. I would start with one hyperparameter while holding the rest unchanged. The target hyperparameter will get values of much smaller interval in hope that I can pin down the best value.

F1 score, confusion matrix, precision, recall and ROC/AUC are all put to use to gauge the performance of a model. Recall is critical here as the goal is not to miss patient with Parkinson's disease and ok to sacrifice precision reasonably by predicting healthy individual to have Parkinson's disease.

All results are tabulated and the voting classifier stood out.

Improvement

There are still ample room to improve. The parameter tuning can certainly be vastly improved. A deep learning model has been attempted and it just need a lot more effort to achieve its best potential.

The keystroke data can use some deeper insight and coupled with medical knowledge a lot more better features can be developed to hugely improve model without resorting to unnecessary complex modeling.

References

1. Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. *Circulation* **101**(23):e215-e220 [Circulation Electronic Pages; http://circ.ahajournals.org/content/101/23/e215]; 2000 (June 13).

2. L. Giancardo, A. Sánchez-Ferro, T. Arroyo-Gallego, I. Butterworth, C. S. Mendoza, P. Montero, M. Matarazzo, J. A. Obeso, M. L. Gray, R. San José Estépar. Computer keyboard interaction as an indicator of early Parkinson's disease. Scientific Reports 6, 34468; doi: 10.1038/srep34468 (2016)

3. http://news.mit.edu/2015/typing-patterns-diagnose-early-onset-parkinsons-0401
   Anne Trafton, MIT News Office
   April 1, 2015

4. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5708704/
   Holger Fröhlich
   2017

5. http://xgboost.readthedocs.io/en/latest/python/python_api.html
6. https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/
7. https://www.kaggle.com/lct14558/imbalanced-data-why-you-should-not-use-roc-curve
8. https://acutecaretesting.org/en/articles/precision-recall-curves-what-are-they-and-how-are-they-used
9. https://jakevdp.github.io/PythonDataScienceHandbook/04.14-visualization-with-seaborn.html