Parkinson Classification Based on Demographic Information and Voice Features

Tong Zou
Department of Statistics
University of Washington
Seattle, WA 98105
`tongzou@uw.edu`

Abstract

We are interested in using demographic information and GeMAPS extracted features of voice to classify a patient diagnosed with Parkinson's disease. We started with demographic data (80% negative cases), testing different classification methods (achieved 89% accuracy and 67% recall), and then went on combining with voice features and improved the results (achieved 91% accuracy and 77% recall).

1 Introduction

Parkinson's disease is a long-term degenerative disorder of the central nervous system, whose causes are still shrouded in mystery [3]. People who suffer from Parkinson's disease are likely to be faced with both serious thinking and behavioral problems. And the public's awareness is increased in recent years.

My project uses demographic and vocal features to diagnose Parkinson's disease by machine learning algorithms. A various previous works catch my eye, with different inputs and different models [1]. Compared with them, my project is special in the following two ways.

First of all, we pay much attention to demographic features, which are not heated signals in this field. The reason is that we want a 'taste' of causality, even without random experiments. A strongly correlated demographic feature will help with prevention. And we do uncover the correlation between age and Parkinson's disease.

Secondly, my project considers vocal features as well. The thing is different people have different numbers of vocal samples, which means if we just add the demographic features as prefixes to the vocal data, then we might obtain different diagnoses for a same patient. We can use Bagging to fix this problem, but patients are treated unevenly because we actually work with some unreasonable weights implicitly. Eventually, we apply a simple and straightforward data processing to handle this.

To sum up, my project systematically combines demographic and vocal features and uses machine learning models in Parkinson diagnosis. my results provide good insights for both prevention and management.

2 Data Preparation

The data used for this analysis was collected by Sage Bionetworks through the mPower research study `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4776701/pdf/sdata201611.pdf`.

We have 6668 patients totally and about 84% are free of Parkinson's disease. We always split the data into train set (80%) and test set (20%) in experiments, and calculate accuracy and recall on test set.

2.1 Demographic features

For demographic features, we focus on the following 11 demographic features, which are shown in Table 1. 'age' is the only continuous variable and all the other variables are categorical.

Table 1: Demographic Features

Feature	Description
'age'	continuous, range from 18 to 90
'are.caretaker'	categorical, 3 levels
'education'	categorical, 7 levels
'employment'	categorical, 7 levels
'gender'	categorical, 4 levels
'maritalStatus'	categorical, 7 levels
'medical.usage'	categorical, 3 levels
'medical.usage.yesterday'	categorical, 4 levels
'race'	categorical, 7 levels
'smoked'	categorical, 3 levels
'surgery'	categorical, 3 levels

At beginning, we group by the samples in each category to see their proportions. We find that only 16% people in this study have been diagnosed as Parkinson. 'age' is the only numerical feature, and most people are under 60. There are 78% males. More than 95% people are not care takers. Another interesting fact is that almost everyone has medical usage history but more than 68% of them didn't use medicine the day before the test. And missing values account for very small proportion for each demographic feature.

2.2 Vocal features

We have 62 vocal features in this study. These features are extracted based on their potential to index affective physiological changes in voice production, and they are well-used in multiple contexts [2].

3 Models and Experiments

my experiments can be divided into two parts. Firstly, we only consider demographic features. As is mentioned above, we are curious about the correlations between demographic features and Parkinson's disease. So we build two baseline models with Random Forrest and Logistic Regression correspondingly, and explore both variable importance and statistical significance.

After that, we try to add vocal features to improve my Random Forrest model. For each patient, we calculate the average of each vocal feature across all his/her vocal samples. And then, we put these averaged vocal features in Random Forrest model. We also tune several hyperparameters by cross validation.

3.1 baseline - demographic features only

In order to find out the feature importance, we employ the Random Forest method.

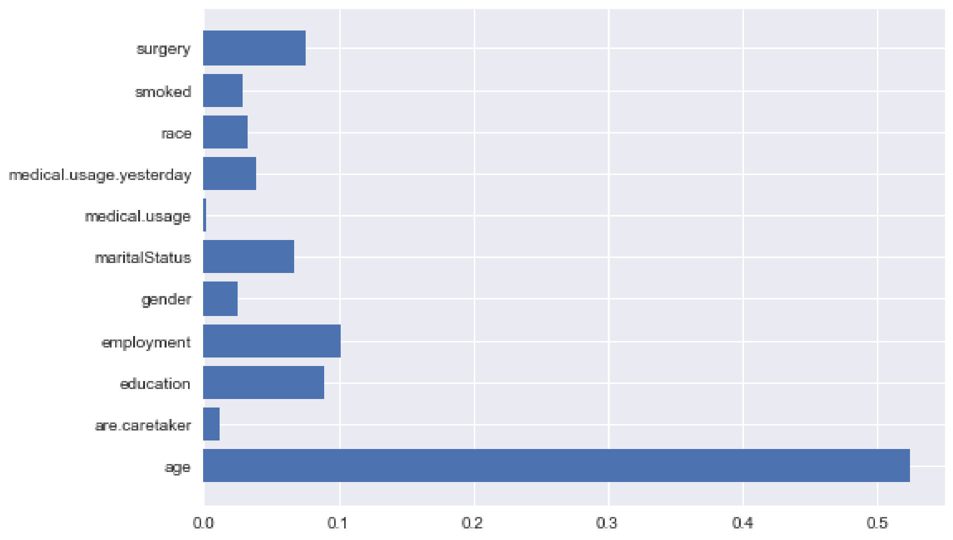

Figure 1: Demographic Feature Importance

As we can see, 'age' is the most important feature. This result makes sense intuitively because aged individuals usually have worse health condition compared with the youth.

The precision for this base Random Forrest model is 0.89 and the recall is 0.69.

Then we fit the data with Logistic Regression model, and we obtain precision of 0.90 and recall of 0.73. The corresponding p-values for each variable are revealed as following.

Table 2: P-values with Regard to Demographic Features

Feature	P-value
'age'*	2e-280
'are.caretaker'	0.09
'education'	0.24
'employment'*	7e-8
'gender'*	4e-6
'maritalStatus'*	0.08
'medical.usage'	0.66
'medical.usage.yesterday'*	0.0005
'race'*	0.05
'smoked'*	0.002
'surgery'*	6e-25

In Table 2, the features with small p-values (significantly smaller than $\alpha = 0.05$ confidence level) are stared. Moreover, 'age' is still the most important feature, which is consistent with what we obtain with Random Forest model.

3.2 advanced - demographic features and averaged vocal features

We combined demographic features and averaged vocal features mentioned above in an advanced Random Forrest model. To adjust the hyperparameters, such as maximum tree depth, maximum number of features and minimal samples split, we use 3-fold cross validation to obtain the best model. Finally, we get a model achieving precision of 0.91 and recall of 0.77, which is consistent with test scores in training.

The comparison of performance of baseline models and advanced model can be seen in Table 3.

Table 3: Performance Comparison

Model	Accuracy	Recall
Base - Random Forrest	0.89	0.69
Base - Logistic Regression	0.90	0.73
Advance - Random Forrest	0.91	0.77

As we can see, we improve the model by including vocal features.

4 Conclusion and Future Work

From what has been mentioned above, we can safely draw the conclusion that aged people are exposed to a relative high risk of Parkinson's disease. Thus, senior citizens should have regular physical examinations.

Moreover, vocal features do help with Parkinson diagnosis, which means a significant change or anormaly in voice could be a good signal in Parkinson diagnosis.

Three things are under consideration to improve my work.

Firstly, combining demographic and vocal features by simply calculating averages might lose a lot information. A bootstrapping framework could be an alternative option.

Secondly, we are using GeMAPS vocal features all the time. It brings more possibilities to touch the raw voice data.

Finally, we basically build a universal model, but in real medical scenario, different patients may have different constitutions. Since we have multiple vocal samples for a single patient, there is a chance to transfer the universal model to a personal model. Or we might think of modeling the change of voice.

References

[1] Bind, Shubham.(2015) Survey of Machine Learning Based Approaches for Parkinson Disease Prediction. International Jmynal of Computer Science and Information Technologies 6 : n. pag. International Jmynal of Computer Science and Information Technologies. 2015. Web. 8 Mar. 2017.

[2] Eyben, F., Scherer, K. R., Schuller, B. W., Sundberg, J., Andre, E., Busso,C., Devillers, L. Y., Epps, J., Laukka, P., Narayanan, S. S., and Truong, K.P. (2016). "The Geneva minimalistic acoustic parameter set (GeMAPS)for voice research and affective computing," IEEE Trans. Affect. Comput.7(2), 190–202.

[3] Wikipedia. Parkinson's disease - Wikipedia, the free encyclopedia, 2018. URL https://en.wikipedia.org/wiki/Parkinson%27s_disease [Online, accessed 06-December-2018].

Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Thu Dec 06 12:10:33 2018

@author: tong
"""
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# Set random seed
np.random.seed(546)

#################################data processing##############################
data = pd.read_csv('data.csv')
data_avg = pd.read_csv('data_avg.csv')

data['age'] = data['age'].fillna(-1)
data = data.fillna('UNK')

data_avg['age'] = data_avg['age'].fillna(-1)
data_avg = data_avg.fillna('UNK')

for i in range(2,13):
    data[data.columns[i]] = pd.factorize(data[data.columns[i]])[0]

for i in range(-1,-12,-1):
    data_avg[data_avg.columns[i]] = (pd.factorize(data_avg
                                     [data_avg.columns[i]])[0])

data_avg['is_train'] = np.random.uniform(0, 1, len(data_avg)) <= .8
train_avg, test_avg = (data_avg[data_avg['is_train']==True],
                       data_avg[data_avg['is_train']==False])
# Show the number of observations for the test and training dataframes
print('Number of people in the training data:', len(train_avg))
print('Number of people in the test data:',len(test_avg))

is_train = []
for i in range(data.shape[0]):
    is_train.append(data_avg[data_avg['ROW_ID']==(data['ROW_ID'][0]]
                                                        ['is_train'][0]))
data['is_train'] = is_train
train, test = data[data['is_train']==True], data[data['is_train']==False]

features_avg = (data_avg.columns[[i for i in range(1,len(list(data))-1)
                if i!=len(list(data_avg))-5]])
y_avg = pd.factorize(train_avg['professional.diagnosis'])[0]

features = data.columns[[i for i in range(1,len(list(data))-1) if i!=9]]
y = pd.factorize(train['professional.diagnosis'])[0]

######################baseline: Random Forrest###############################
# Create a random forest Classifier. By convention, clf means 'Classifier'
clf = RandomForestClassifier(n_jobs=2, random_state=0)

# Train the Classifier to take the training features and learn how they relate
# to the training y (the species)
clf.fit(train_avg[features_avg[-11:]], y_avg)
```

```
label = pd.factorize(test_avg['professional.diagnosis'])[0]
pred_base = clf.predict(test_avg[features_avg[-11:]])
cnt_base = 0
recall_base = 0
for i in range(len(label)):
    if label[i] == pred_base[i]:
        cnt_base += 1
        if label[i] == 1:
            recall_base += 1

print cnt_base/float(len(label))
print recall_base/float(np.count_nonzero(label))

plt.barh(list(train_avg[features_avg[-11:]]),list(clf.feature_importances_))
plt.show()

###########################baseline: Logistic Regression######################
from sklearn import linear_model
clf_lgt_base = linear_model.LogisticRegression(random_state=0, solver='lbfgs')
clf_lgt_base.fit(train_avg[features_avg[-11:]], y_avg)

label = pd.factorize(test_avg['professional.diagnosis'])[0]
pred_lgt_base = clf_lgt_base.predict(test_avg[features_avg[-11:]])
cnt_lgt_base = 0
recall_lgt_base = 0
for i in range(len(label)):
    if label[i] == pred_lgt_base[i]:
        cnt_lgt_base += 1
        if label[i] == 1:
            recall_lgt_base += 1

print (cnt_lgt_base/float(len(label)))
print (recall_lgt_base/float(np.count_nonzero(label)))

import scipy.stats as stat
class LogisticReg:
    """
    Wrapper Class for Logistic Regression which has the usual sklearn instance
    in an attribute self.model, and pvalues, z scores and estimated
    errors for each coefficient in

    self.z_scores
    self.p_values
    self.sigma_estimates

    as well as the negative hessian of the log Likelihood (Fisher information)

    self.F_ij
    """

    def __init__(self,*args,**kwargs):#,**kwargs):
        self.model = linear_model.LogisticRegression(*args,**kwargs)#,**args)

    def fit(self,X,y):
        self.model.fit(X,y)
        #### Get p-values for the fitted model ####
        denom = (2.0*(1.0+np.cosh(self.model.decision_function(X))))
        denom = np.tile(denom,(X.shape[1],1)).T
        F_ij = np.dot((X/denom).T,X) ## Fisher Information Matrix
        Cramer_Rao = np.linalg.inv(F_ij) ## Inverse Information Matrix
        # sigma for each coefficient
        sigma_estimates = (np.array([np.sqrt(Cramer_Rao[i,i])
                            for i in range(Cramer_Rao.shape[0])]))
        # z-score for eaach model coefficient
        z_scores = self.model.coef_[0]/sigma_estimates
        ### two tailed test for p-values
        p_values = [stat.norm.sf(abs(x))*2 for x in z_scores]

        self.z_scores = z_scores
        self.p_values = p_values
        self.sigma_estimates = sigma_estimates
        self.F_ij = F_ij

sol = LogisticReg(random_state=0, solver='lbfgs')
sol.fit(train_avg[features_avg[-11:]], y_avg)

list(zip(train_avg[features_avg[-11:]], sol.p_values))

#############################advanced: Random Forrest##########################
# Create the parameter grid based on the results of random search
from sklearn.model_selection import GridSearchCV

param_grid = {
    'bootstrap': [True],
    'max_depth': [10, 20, 50, 80, 100],
    'max_features': [3, 5, 10, 20, 30, 50],
    'min_samples_leaf': [2, 3, 4, 5],
    'min_samples_split': [2, 3, 5, 8, 10],
    'n_estimators': [500, 1000]
}
# Create a based model
rf = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)

# Fit the grid search to the data
grid_search.fit(train_avg[features_avg], y_avg)
best_grid = grid_search.best_estimator_

print grid_search.best_params_

print max(grid_search.cv_results_['mean_test_score'])

pred = best_grid.predict(test_avg[features_avg])
cnt = 0
recall = 0
for i in range(len(label)):
    if label[i] == pred[i]:
        cnt += 1
        if label[i] == 1:
            recall += 1

print cnt/float(len(label))
print recall/float(np.count_nonzero(label))
```