Parkinson’s Disease Detection

By Amanda Foster

About the Dataset

- The dataset was created by Max Little of the University of Oxford, in collaboration with the National Centre for Voice and Speech, Denver, Colorado, who recorded the speech signals.
- The original study published the feature extraction methods for general voice disorders.
- This dataset is composed of a range of biomedical voice measurements from 31 people, 23 with Parkinson's disease (PD).
- Each column in the table is a particular voice measure, and each row corresponds one of 195 voice recording from these individuals ("name" column).
- The main aim of the data is to discriminate healthy people from those with PD, according to "status" column which is set to 0 for healthy and 1 for PD.

About the Dataset (Continued)

Attribute Information:

- Matrix column entries (attributes):
- name - ASCII subject name and recording number
- MDVP:Fo(Hz) - Average vocal fundamental frequency
- MDVP:Fhi(Hz) - Maximum vocal fundamental frequency
- MDVP:Flo(Hz) - Minimum vocal fundamental frequency
- MDVP:Jitter(%),MDVP:Jitter(Abs),MDVP:RAP,MDVP:PPQ,Jitter:DDP - Several measures of variation in fundamental frequency
- MDVP:Shimmer,MDVP:Shimmer(dB),Shimmer:APQ3,Shimmer:APQ5,MDVP:APQ,Shimmer:DDA - Several measures of variation in amplitude
- NHR,HNR - Two measures of ratio of noise to tonal components in the voice
- status - Health status of the subject (one) - Parkinson's, (zero) - healthy
- RPDE,D2 - Two nonlinear dynamical complexity measures
- DFA - Signal fractal scaling exponent
- spread1,spread2,PPE - Three nonlinear measures of fundamental frequency variation

Comparing K Nearest Neighbors, Logistic Regression, and Decision Tree

K Nearest Neighbors Classifier

Mean test accuracy:
0.8461538461538461

Precision = TP / (TP + FP) =
0.8399503722084367

Recall = TP / (TP + FN) =
0.8461538461538461

Confusion Matrix (test):

[[6 4]

 [2 27]]

Logistic Regression

Mean test accuracy:
0.7435897435897436

Precision = TP / (TP + FP) =
0.5529257067718607

Recall = TP / (TP + FN) =
0.7435897435897436

Confusion Matrix (test):

[[0 10]

 [0 29]]

Decision Tree

Mean test accuracy:
0.7692307692307693

Precision = TP / (TP + FP) =
0.8183760683760684

Recall = TP / (TP + FN) =
0.7692307692307693

Confusion Matrix (test):

[[8 2]

 [7 22]]

Why did K Nearest Neighbors Perform Best?

- For medical diagnostics K Nearest Neighbors frequently performs well
- Why? Simply put: the KNN model looks for similarity.
- K-nearest neighborhoods (k-NN) classifier is highly dependent on the distance metric used to identify the k nearest neighbors of the query points. So it uses this concept of distance and looks for patients that are the most similar to the one it is trying to predict. You would expect patients with Parkinson's to be similar, thus it will perform well classifying if someone has the disease.

Problem with this Dataset

- Very small → Only 195 rows that need to be broken up to training and test sets
- After the stratified shuffle split the training set was only 80% of the data
- Hypothesis: More training data = stronger model?

Training on More Data

- Used Decision Tree Classifier and I trained on all data but one row. Then I tested this model on the one row I dropped from the training data, keeping track if the status of this row was correctly predicted. I repeated this process removing every row in the data set and then calculated the score at the end by:

score: dividing # correctly classified/ #rows in the data set (aka # attempts)

```
#To increase the size of our training set. Larger training set = better model
def attemptTree(df, max_depth, min_samples_leaf):
    successes = 0
    attempts = 0
    # goes through the entire dataframe n,training on (n-1) data
    for row in range(len(df)):
        reduceddf = df.drop(row) #removes one row from the dataframe
        dtc = DecisionTreeClassifier(max_depth = max_depth, min_samples_leaf = min_samples_leaf, random_state = 5)
        result = runTest("", dtc,
                         reduceddf.drop(["status"], axis=1), #trains on all the data but the one removed row
                         reduceddf["status"],
                         df[row:row+1].drop(["status"], axis=1), #tests on the one removed row
                         df[row:row+1]["status"],
                         'weighted', True)
        successes += result
        attempts += 1
    #reduceddf.info()
    return successes/attempts
```

Results when leaving out different #s of rows to test on

testing on 1: Final model: max_depth= 4 , min_samples_leaf= 7 score= 0.9179487179487179

testing on 2: Final model: max_depth= 4 , min_samples_leaf= 7 score= 0.8994845360824743

testing on 3: Final model: max_depth= 4 , min_samples_leaf= 9 score= 0.8687392055267701

testing on 4: Final model: max_depth= 5 , min_samples_leaf= 9 score= 0.8372395833333334

testing on 5: Final model: max_depth= 4 , min_samples_leaf= 1 score= 0.8261780104712044

testing on 6: Final model: max_depth= 1 , min_samples_leaf= 1 score= 0.8219298245614032

testing on 7: Final model: max_depth= 1 , min_samples_leaf= 1 score= 0.8231292517006806

testing on 8: Final model: max_depth= 1 , min_samples_leaf= 1 score= 0.824468085106383

testing on 9: Final model: max_depth= 1 , min_samples_leaf= 1 score= 0.8259061200237664

testing on 10: Final model: max_depth= 1 , min_samples_leaf= 1 score= 0.8274193548387108

Final Thoughts

- Training on more data = stronger model
  - New accuracy: 0.9179487179487179
  - Old accuracy: 0.7692307692307693
- The Accuracy 0.9179487179487179 is a good predictor of what the accuracy our idea model would produce (ideal model would be if we used all 195 rows are training data for our model and then were given new data to predict on)
- Further Exploration
  - I wish I had more time to perform the same process with K Nearest Neighbors and see how the accuracy would change if I repeated the same process